Appendix L: Land Use Model Documentation

CONTENTS

1. MODELING OVERVIEW AND STRUCTURE

MODEL SYSTEM OBJECTIVES, CAPABILITIES, AND LIMITATIONS

DESIGN OBJECTIVES

UrbanSim is an urban simulation system developed over the past several decades to better inform deliberation on public choices with long-term, significant effects. A key motivation for developing such a model system is that the urban environment is complex enough that it is not feasible to anticipate the effects of alternative courses of action without some form of analysis that could reflect the cause and effect interactions that could have both intended and possibly unintended consequences.

UrbanSim was designed to attempt to reflect the inter-dependencies in dynamic urban systems, focusing on the real estate market and the transportation system, initially, and on the effects of individual interventions, and combinations of them, on patterns of development, travel demand, and household and firm location. Some goals that have shaped the design of UrbanSim, and some that have emerged through the past decades of seeing it tested in the real world, are the following:

OUTCOME GOALS

- Enable a wide variety of stakeholders (planners, public agencies, citizens and advocacy groups) to explore the potential consequences of alternative public policies and investments using credible, unbiased analysis.
- Facilitate more effective democratic deliberation on contentious public actions regarding land use, transportation and the environment, informed by the potential consequences of alternative courses of action that include long-term cumulative effects on the environment, and distributional equity considerations.
- Make it easier for communities to achieve a common vision for the future of the community and its broader environment, and to coordinate their actions to produce outcomes that are consistent with this vision.

IMPLEMENTATION GOALS

- Create an analytical capacity to model the cause and effect interactions within local urban systems that are sufficiently accurate and sensitive to policy interventions to be a credible source for informing deliberations.
- Make the model system credible by avoiding bias in the models though simplifying assumptions that obscure or omit important cause-effect linkages at a level of detail needed to address stakeholder concerns.
- Make the model design behaviorally clear in terms of representing agents, actions, and cause - effect interactions in ways that can be understood by non-technical stakeholders, while making the statistical methods used to implement the model scientifically robust.
- Make the model system open, accessible and transparent, by adopting an Open Source licensing approach and releasing the code and documentation on the web.

- Encourage the development of a collaborative approach to development and extension of the system, both through open source licensing and web access, and by design choices and supporting organizational activities.
- Test the system extensively and repeatedly, and continually improve it by incorporating lessons learned from applications, and from new advances in methods for modeling, statistical analysis, and software development.

DESIGN GOALS

The original design of UrbanSim adopted several elements to address these implementation goals, and these have remained foundational in the development of the system over time. These design elements include:

- The representation of individual agents: initially households and firms, and later, persons and jobs.
- The representation of the supply and characteristics of land and of real estate development, at a fine spatial scale: initially a mixture of parcels and zones, later gridcells of user-specified resolution.
- The adoption of a dynamic perspective of time, with the simulation proceeding in annual steps, and the urban system evolving in a path dependent manner.
- The use of real estate markets as a central organizing focus, with consumer choices and supplier choices explicitly represented, as well as the resulting effects on real estate prices. The relationship of agents to real estate tied to specific locations provided a clean accounting of space and its use.
- The use of standard discrete choice models to represent the choices made by households and firms and developers (principally location choices). This has relied principally on the traditional Multinomial Logit (MNL) specification, to date.
- Integration of the urban simulation system with existing transportation model systems, to obtain information used to compute accessibilities and their influence on location choices, and to provide the raw inputs to the travel models.
- The adoption of an Open Source licensing for the software, written originally in Java, and reimplemented using the Python language. The system has been updated and released continually on the web since 1998.

KEY FEATURES

- The model simulates the key decision makers and choices impacting urban development; in particular, the mobility and location choices of households and businesses, and the development choices of developers.
- The model explicitly accounts for land, structures (houses and commercial buildings), and occupants (households and businesses).
- The model simulates urban development as a dynamic process over time and space, as opposed to a cross-sectional or equilibrium approach.
- The model simulates the land market as the interaction of demand (locational preferences of businesses and households) and supply (existing vacant space, new construction, and redevelopment), with prices adjusting to clear the market.

- The model incorporates governmental policy assumptions explicitly, and evaluates policy impacts by modeling market responses.
- The model is based on random utility theory and uses logit models for the implementation of key demand components.
- The model is designed for high levels of spatial and activity disaggregation, with a zonal system identical to travel model zones.
- The model presently addresses both new development and redevelopment, using parcel-level detail.

CAPABILITIES

UrbanSim has been developed to support land use, transportation and environmental planning, with particular attention to the regional transportation planning process. It has been designed to perform several tasks.

1. It can predict land use information for input to the travel model, for periods of 10 to 40 years into the future, as needed for regional transportation planning.
2. It can predict the effects on land use patterns from alternative investments in roads and transit infrastructure, alternative transit levels of service, or alternative roadway and transit pricing, over long-term forecasting horizons. Scenarios can be compared using different transportation network assumptions to evaluate the relative effects on development from a single project or a more wide-reaching change in the transportation system, such as extensive congestion pricing.
3. It can predict the effects of changes in land use regulations on land use. This includes the effects of policies to relax or increase regulatory constraints on development of different types, such as an increase in the allowed Floor Area Ratios (FAR) on specific sites, or allowing mixed-use development in an area previously zoned only for one use.
4. It can predict land use development patterns induced by investments in transit.
5. It can predict the effects of environmental policies that impose constraints on development, such as protection of wetlands, floodplains, riparian buffers, steep slopes, or seismically unstable areas.
6. It can predict the effects of changes in the macroeconomic structure or growth rates on land use. Periods of rapid or slow growth, or even decline in some sectors, can lead to changes in the spatial structure of the city and the model system is designed to analyze these shifts.
7. It can predict the possible effects of changes in demographic structure and composition of the city on land use and on the spatial patterns of clustering of residents of different social characteristics, such as age, household size, and income.
8. It can examine the potential impacts of major development projects (both actual and hypothetical) on land use and transportation. This can be used to explore the impacts of a corporate relocation or to compare alternative sites for a major development project.

ASSUMPTIONS AND LIMITATIONS OF THE MODEL SYSTEM

UrbanSim is a model system, and models are abstractions, or simplifications, of reality. Only a small subset of the real world is reflected in the model system, as needed to address the kinds of uses outlined above. Like any model, or analytical method, that attempts to examine the potential

effects of an action on one or more outcomes, there are limitations to be aware of. Some of the assumptions made in developing the model system also imply limitations for its use. Some of the more important of the assumptions and limitations are:

- **Boundary effects are ignored**. Interactions with adjacent metropolitan areas are ignored.
- **The land use regulations are assumed to be binding constraints on the actions of developers**. This is equivalent to assuming that developers who wish to construct a project that is inconsistent with current land use regulations cannot get a waiver or modification of the regulations in order to accommodate the project. This assumption is more reflective of reality in some places than others, depending on how rigorously enforced land policies are in that location. Clearly there are cities in which developer requests for a variance from existing policies meets with little or no resistance. For the purposes the model system is intended, however, this assumption, and the limitation that it does not completely realistically simulate the way developers influence changes in local land use policies, may be the most appropriate. It allows examination of the effects of policies, under the expectation that they are enforced, which allows more straightforward comparisons of policies to be made. *[However, the Eugene-Springfield UrbanSim model system implementation is the first of its kind to include a capability for simulating rezoning and conditional uses, and the fees and costs involved, within the constraints of the existing comprehensive plans.]*
- **Large-scale and microscopic events cannot be accurately predicted**. While this limitation applies to any and every model, not just UrbanSim, it bears repeating since the microscopic level of detail of UrbanSim leads to more temptation to over-invest confidence in the micro-level predictions. Though the model as implemented in the Eugene-Springfield area predicts the location choices of individual jobs, households, and developers, the intent of the model is to predict patterns rather than discrete individual events. No individual prediction made by the model, such as the development of a specific development project on a single parcel in a particular year 20 years from now, is likely to be correct. But the tendencies for parcels in that area to have patterns or tendencies for development is what the model is intended to represent. Model users should therefore not expect to accurately predict large-scale, idiosyncratic events such as the development of a specific high-rise office building on a specific parcel. It would be advisable to aggregate results, and/or to generate multiple runs to provide a distribution of results. A related implication is that the lower level of sensitivity and appropriate use of the model system needs to be determined by a combination of sensitivity testing, experience from use, and common sense. It would not be likely, for example, that changing traffic signalization on a particular collector street intersection would be a large enough event to cause significant changes in model results. *[As part of model validation and sensitivity testing, the LCOG model was tested for the effect of variation in the random seed on model results. A large number of runs were performed with only the random seed varying. These runs were analyzed statistically and the model results were found to not be unrealistically affected by the stochasticity of the random seed.]*
- **Errors in input data will limit the model to some extent**. Efforts were made to find obvious errors in the data, and to prevent these from affecting the results, but there was not sufficient time or resources to thoroughly address all data problems encountered, including some extreme values, missing values, and inconsistencies within and among data sources. The noise in the input data limits to some extent the accuracy of the model, though the statistical estimation of the parameters should help considerably in developing unbiased parameters even in the presence of missing data and other data errors. Over a longer period of time, it would be well worth investigating how much difference errors in input data make in model results, and to fine-tune a strategy to invest in data where it makes the most effective use of scarce resources.
- **Behavioral patterns are assumed to be relatively stable over time**. One of the most common assumptions in models, and one rarely acknowledged, is that behavioral patterns will not change dramatically over time. Models are estimated using observed data, and the parameters reflect a

certain range of conditions observed in the data. If conditions were to change dramatically, such as massive innovation in currently unforeseen fuel technology, it is probably the case that fundamental changes in consumption behavior, such as vehicle ownership and use, would result.

MODEL SIMULATION SEQUENCE

(based on simulate.py)

The LCOG UrbanSim model is composed of many submodels. These run in a coordinated way for multiple simulation years, depending on the simulation set up.

These models are currently designed to run in the following order. Specific named models are identified in quotes. These named models are grouped into functional sets that represent the bigger working pieces of the model, representing demand, supply, price and allocation dimensions of the modeling process.

MODEL NAMES

```
start_of_year_models
    'skim_swapper'
    'scheduled_development_events_model'

transition_models
    'household_transition'
    'job_transition'

developer_models
    feasibility_step
        'feasibility'
    developer_steps
        if calibrated:
            'residential_developer_calib'
            'non_residential_developer_calib'
        else
            'residential_developer'
            'non_residential_developer'

price_models
    'repm_residential'
    'repm_rent_industrial'
    'repm_rent_retail'
    'repm_rent_office'

location_models
    if calibrated:
        hlcms
            'hlcm1_calib'
            'hlcm2_calib'
        elcms
            'elcm1_calib'
            'elcm2_calib'
```

```
            'elcm3_calib'
            'elcm4_calib'
            'elcm5_calib'
            'elcm6_calib'
            'elcm7_calib'
            'elcm8_calib'
            'elcm9_calib'
            'elcm10_calib'
            'elcm11_calib'
            'elcm12_calib'
            'elcm13_calib'
            'elcm14_calib'
    else
        hlcms
            'hlcm1'
            'hlcm2'
        elcms
            'elcm1'
            'elcm2'
            'elcm3'
            'elcm4'
            'elcm5'
            'elcm6'
            'elcm7'
            'elcm8'
            'elcm9'
            'elcm10'
            'elcm11'
            'elcm12'
            'elcm13'
            'elcm14'

end_of_year_models
    'generate_indicators'
```

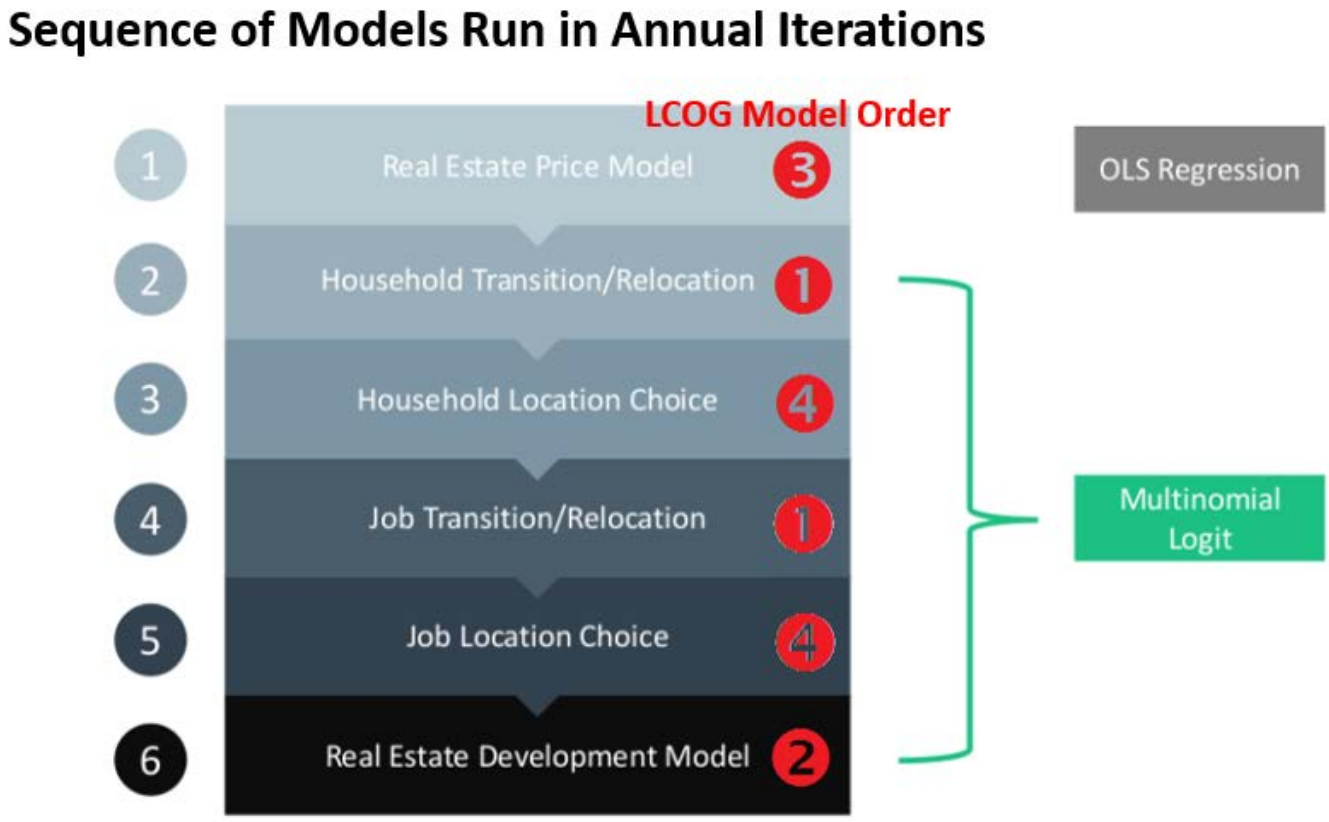
SIMULATE BASE YEAR (E.G. 2011)

```
'scenario_definition'
'build_networks'
'generate_indicators'
price_models
```

SIMULATE FORECAST YEARS (E.G. 2012-2035)

```
start_of_year_models
transition_models     (demand)
developer_models      (supply)
price_models          (price)
location_models       (allocation)
end_of_year_models
```

Note: Early presentations on the UrbanSim model sometimes depict this sequence in an different order. See for example the following image (correct sequence in red).



CUSTOMIZATIONS AND ENHANCEMENTS

This page lists the customizations, enhancements, and new features that the current phase of the LCOG UrbanSim model uses.

LCOG Model enhancements

- Various developer model enhancements
- Index-inspired indicators
- The random point generation within polygon functionality
- Prototype implementation (in branch) of pandana access variable involving querying the GTFS transit schedule
- Pandana mid-segment breaks for better pedestrian-scale accessibility

UrbanSim/UrbanCanvas features that the model was an early adopter of

- Latest implementation of gradient-based price equilibration
- Back-propagation-based calibration methodology
- Parcel version of UrbanCanvas Modeler
- Draft deployment of the UrbanCanvas Jupyterhub service

2. DATA

DATA INVENTORY

The data assessment and needs analysis presented here reflects a parcel-level UrbanSim implementation utilizing a proforma-based real estate supply model.

Typically, when a region embarks on implementing UrbanSim, large amounts of raw data must be processed into a form usable for model estimation and simulation (i.e. an ETL step). Data from many different sources are reconciled into one common framework, and the data is then subjected to cleaning and imputation. Because of LCOG's history of modeling and parcel-level data tracking, there were many existing data resources to draw from and we weren't starting from truly "raw" datasets, but there was still work involved in connecting various datasets together into a unified whole for UrbanSim.

The processed form of all the datasets described in this memo can be viewed in the UrbanCanvas Modeler user interface.

PARCELS AND BUILDINGS

LCOG provided building and parcel data, which UrbanSim staff then processed for use in both UrbanSim and UrbanCanvas Modeler. The provided building and parcel data represent the year 2020 and are the result of aggregation into super-parcels (dissolved tax-lots) and aggregate buildings (all improvements for each parcel combined). The processed buildings/parcels were then used in later data processing steps such as household allocation to building, and the tables were uploaded to the UrbanCanvas platform. The building/parcel tables have been incorporated into the model system.

UrbanSim documentation

Parcel geometry:
https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#parcel-geometry

Parcel attributes:
https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#parcel-attributes

Building attributes:
https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#buildings

Original files on Google Drive

Parcels: https://drive.google.com/open?id=1QebxHoM8OtMRE-eqBddLL0MvjB3lj1OW

Buildings: https://drive.google.com/open?id=1bFHW_Ynr48V5KbptaKjQu04ljijyV4-O

Github Issues

Parcels: https://github.com/urbansim/lcog/issues/13

Buildings: https://github.com/urbansim/lcog/issues/12

HOUSEHOLDS

UrbanSim staff synthesized a population at the block group level and then prepared scripts/notebooks to validate the population and allocate household records to the building level. The household/person tables have been incorporated into the model system.

UrbanSim documentation

Household table: https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#households

Persons table: https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#persons-optional

Original files on Google Drive

Households: https://drive.google.com/open?id=11wcBPsT7KIbFozvw5sKNKTovgPpIfc5W

Persons: https://drive.google.com/open?id=1La1E0GGeZqUj_vQki_oxs-Hl9BNNIW_s

Github Issues

https://github.com/urbansim/lcog/issues/3

JOBS

LCOG provided a shapefile of job points with firm_id and sector_id. UrbanSim used these points to assign jobs to a building, and where no building exists, to impute a building (imputation was necessary because 2010 data did not include all non-residential buildings; 2020 data is more complete and this step should be unnecessary going forward). The jobs table has been incorporated into the model system.

UrbanSim documentation

Jobs table: https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#jobs

Original files on Google Drive

Jobs shapefile with firm_id: https://drive.google.com/open?id=1yZjtCbQ9P9Y_5LiZgUHFkF5_YiE8CZGr

Github Issues

Define employment sectors: https://github.com/urbansim/lcog/issues/5

Prepare jobs table: https://github.com/urbansim/lcog/issues/6

TRAVEL_DATA AND ZONES

LCOG provided a shapefile of travel model zones, along with AM auto skims and mid-day auto skims. These were processed into an UrbanSim `travel_data` table, and are a key input to skim-based accessibility variables used in various UrbanSim submodels.

UrbanSim documentation

Zones geometry: https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#travel-model-zones

Skims: https://cloud.urbansim.com/docs/general/documentation/urbansim%20parcel%20model%20data.html#travel-model-skims

Original files on Google Drive

Directory containing zones and skims: https://drive.google.com/open?id=1VBjamnzcJlpdmdHauAmRxGuSKho4kjGt

Github Issues

https://github.com/urbansim/lcog/issues/17

NETWORK NODES/EDGES

UrbanSim staff prepared an initial pedestrian network based on publicly available OpenStreetMap data, and processed into UrbanSim-ready nodes/edges table. Similarly, a transit network based on Lane Transit District schedules from September, 2018 (as reflected in their GTFS feed) was prepared. These tables allow the simulation to calculate on-the-fly network-based ped/transit accessibility variables, such as "retail jobs within 400 meters" or "population within 10 minutes transit time". Based on LCOG feedback, pedestrian edges are divided at their mid-point to more

accurately reflect the starting-points and ending-points of trips, as UrbanSim's network accessibility calculator (pandana) associates parcels with nodes.

UrbanSim documentation

Network overview: https://cloud.urbansim.com/docs/general/documentation/uploads.html#transportation-networks

Nodes: https://cloud.urbansim.com/docs/general/documentation/uploads.html#nodes

Edges: https://cloud.urbansim.com/docs/general/documentation/uploads.html#edges

Original files on Google Drive

Pedestrian edges/nodes, and transit edges/nodes can be found at: https://drive.google.com/open?id=1540xOl2sNjcfmIbTWQqc7ukji-HG-FnQ

Github Issues

https://github.com/urbansim/lcog/issues/49

PROFORMA INPUTS

UrbanSim staff prepared a configuration file named `proforma.yaml` that contains all proforma input parameters. The various cost inputs were adjusted from Bay Area values to Eugene-Springfield values based on a regional scaling factor from R.S. Means, a vendor of construction cost data that has regional construction cost indexes. The configuration file lives in Github as: https://github.com/urbansim/lcog/blob/master/lcog/configs/proforma.yaml

UrbanSim documentation

Proforma docstrings: https://github.com/UDST/developer/blob/master/developer/sqftproforma.py#L14-L175

Existing proforma overview wiki: https://github.com/urbansim/lcog/wiki/Developer-model-memo

Original files on Google Drive

N/A- the data went directly into the Github repo as /configs/proforma.yaml

Github Issues

https://github.com/urbansim/lcog/issues/32

https://github.com/urbansim/lcog/issues/31

BUILDING_TYPES

LCOG has decided on an UrbanSim building typology based on their available data/categories and also considering UrbanSim considerations such as the use of building types for segmenting vacancy rates, the price model, building type dummies in location choice models, proforma inputs, and zoning. The current building types table used by the model is located on Github at: https://github.com/urbansim/lcog/blob/master/lcog/data/building_types.csv

UrbanSim documentation

> https://cloud.urbansim.com/docs/general/documentation/urbansim%20block%20model%20data.html#block-building-types-table

Original files on Google Drive

> Original is in buildings.zip: https://drive.google.com/open?id=1bFHW_Ynr48V5KbptaKjQu04ljijyV4-O
>
> Formatted: https://drive.google.com/open?id=1D2Q8wwD3zVrATDLfUjCAqYr0lLudf4cJ

Github Issues

> https://github.com/urbansim/lcog/issues/10

DATA PRE-PROCESSING PIPELINE

PROCESSING

A script was developed to create a single, repeatable pipeline of steps that processes most data provided by the client and outputs a simulation-ready model_data.h5 file. This makes the LCOG data pre-processing more replicable and cleaner.

POPULATION SYNTHESIS

One step in preparing the model input data is synthesizing a population, and then allocating the resulting synthetic population from the aggregate synthesis geography (e.g. block group) to the building level. UrbanSim is a microsimulation that operates on disaggregate data (individual household and persons). Since we don't observe disaggregate data, we synthesize it based on the aggregate information we observe ("marginals") combined with a sample of disaggregate records ("microdata" e.g. PUMS). The synthesizer tries to expand the microdata in such a way as to approximate the marginals.

We use the SynthPop synthesizer, available in the Urban Data Science Toolkit on Github: https://github.com/UDST/synthpop

The synthesis methodology that SynthPop implements is detailed in the PopGen population synthesis paper: http://www.scag.ca.gov/Documents/PopulationSynthesizerPaper_TRB.pdf

Synthesis-related considerations include:

Are there enough dwelling units in each control geography for the households to occupy?

What marginal variables to utilize in the aggregate data?

What kind of logic to use in block-group -> building allocation?

How well does the resulting synthetic population match controlled variables of interest?

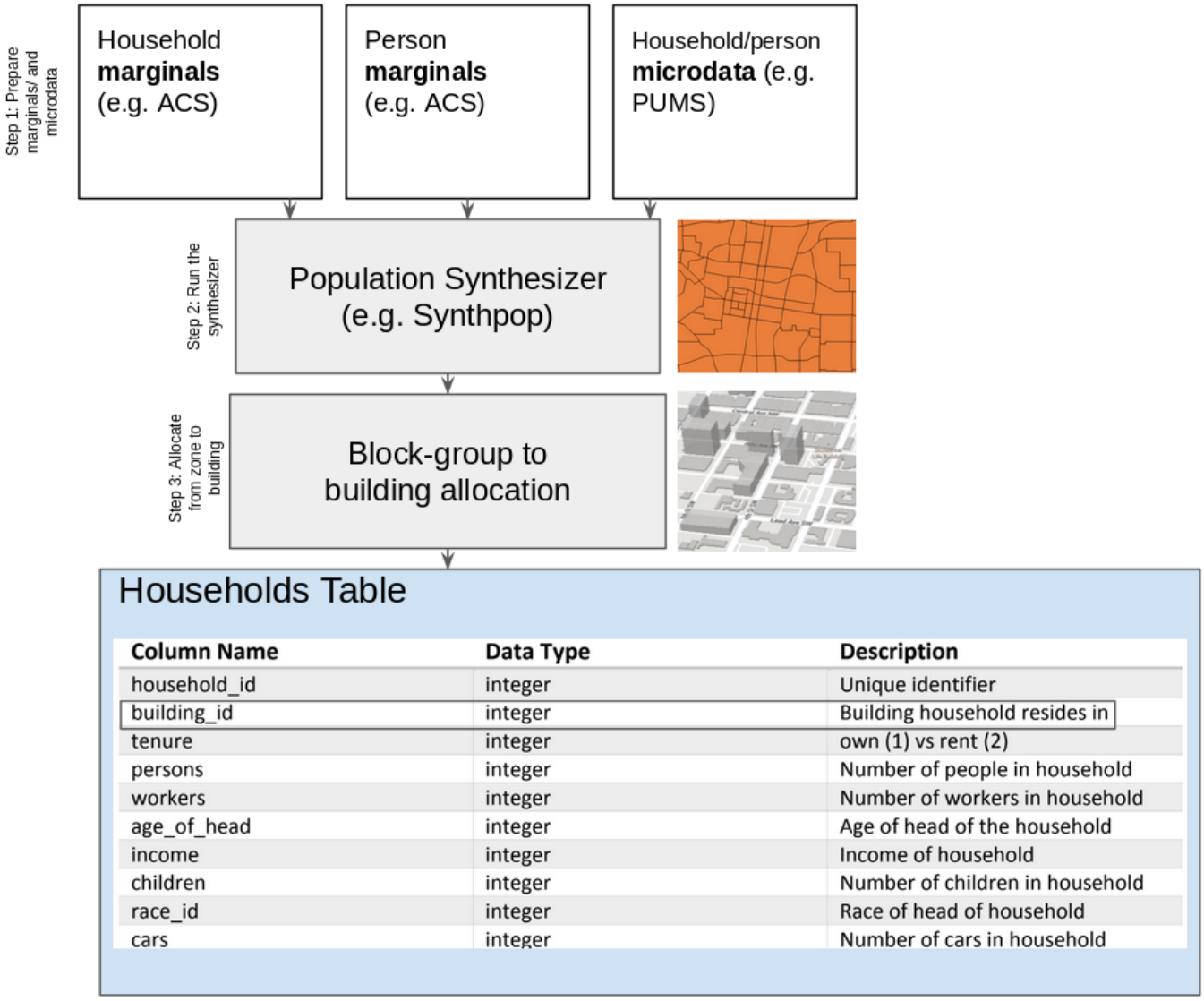
In the LCOG repo, in the scripts/data_pipeline subdirectory, population synthesis is represented by the following scripts/notebooks:

lane_county_pop_synthesis.py: synthesis script that synthesizes population using SynthPop, formats records according to the UrbanSim schema, and then writes out the results to .csv.

`Population_Synthesis_Validation.ipynb: calculates validation metrics

household_allocation.py: probabilistic assignment of households to buildings using ACS table B25124 "Tenure by Household Size by Units in Structure" to inform the type of structure that households by tenure/size are likely to occupy.

The following diagram gives an overview of the population synthesis workflow:



ZONING IMPUTATION

Zoning imputation in the LCOG model is represented by the following steps:

1. Get building and parcel data
2. Group the building and parcel data by zoning polygon
3. Calculate the 90th percentile floor-area ratio and 90th percentile dwelling units per acre within each zoning polygon
4. Use the 90th percentiles as the maximum-allowable density value within each zoning polygons
5. Calculate the unique list of building_type_id's that have been built and are existing in the base data within each zoning polygon. Use this as the allowable building types that may be constructed within each zoning polygon.

6. In the zoning table, for null values in the max_far / max_dua columns, use the 90th percentile in each zoning polgyon as calculated above. Similarly, if the allowable building types need to be imputed, use the unique building types within each zoning polygon as the imputed placeholder set.
7. The code that performs the above zoning imputation logic lives in datasources.py in a function named "imputed_zoning". The imputation occurs at the beginning of each simulation run, so if new zoning data is swapped in the imputation procedure will just do less work when the simulation starts.

3. DEMAND-SIDE MODELS: LOCATION CHOICE AND PRICE

SPECIFICATION STRATEGY

This section discusses the approach taken with respect to model estimation. The following UrbanSim models in the LCOG model system required statistical estimation of parameters.

- the household location choice model (MNL)
- the employment location choice model (MNL)
- the hedonic model of real estate prices (OLS regression)

All estimated coefficients were generated within UrbanSim via Jupyter notebooks. Coefficients are estimated on local LCOG data and not borrowed.

Specification of the location choice models in UrbanSim involves deciding which `alternative` (i.e. location) characteristics to be considered in the model (i.e. explanatory variables). It also involves determining whether to stratify the estimation by some characteristic of the agents making location choices (i.e. segmentation). Stratification reflects the hypothesis that different groups of agents have different locational preferences. For specifying price models, the modeler decides which observations dataset to use (e.g. buildings), which explanatory variables to use, and how to segment the model into submodels (e.g. by building type).

Both adding/dropping explanatory variables and changing the model stratification are easy to do in the UrbanSim framework and the notebooks that have been prepared for LCOG. New variables are defined using simple `pandas` expressions (syntax of the Python pandas library). Each model can be iteratively re-specified and re-estimated quickly during the process of developing a desired model specification. In UrbanSim, the model estimation process is tied closely to simulation. Estimation and simulation both take place within the same code-base and framework. In a properly configured model, simulation can occur right after estimation.

We have variable categories in mind when starting the specification/estimation process (based on hypotheses in the literature), but the specific variables to use depend on local data, review of estimation results (examining coefficient sign, significances, measures-of-fit, and other diagnostics), and an iterative process of trying different specifications.

Variable categories we seek to include in location choice model specifications include real estate characteristics, regional accessibility variables, local accessibility variables, and price. For example, a regional accessibility variable we might try is: employment within 20 minutes auto travel time in the A.M. peak period. This variable would be calculated based on skims from the travel model (stored in the UrbanSim travel_data table). A local accessibility variable we might try is whether there is a school within one mile along the local street network, or retail square footage within a half mile. These kinds of variables would be calculated using the Pandana network accessibility library. In the location choice models, price is a key variable that we try in the specifications. It is hypothesized that, ceteris paribus, households/employment will prefer lower prices (i.e. price will have a negative coefficient), although it is not uncommon in discrete choice models of housing location to find insignificant or even counter-intuitive signs on price variables due to omitted variables that are correlated with price. We also typically include clustering variables. For example, household income interacted with mean income within 400 meters may be tried as an explanatory variable to identify tendencies for income clustering. Similarly, in the employment location choice model, we may try a variable for the number of jobs of the same sector within the zone to capture agglomeration economies.

We start the variable selection process by adding variables to the specification based on behavioral considerations. For example, typical household location choice model explanatory variable categories include:

- price
- residential building characteristics (e.g. year_built)
- neighborhood characteristics
- local and regional accessibility
- interaction variables such as price interacted with income, or a demographic attribute interacted with a location attribute

Typical employment location choice model explanatory variables include:

- price
- building characteristics (e.g. building type, year_built)
- agglomeration/clustering (e.g. number of jobs within same sector within one mile)
- density (e.g. employment density, population density)
- regional accessibility (skim-based or logsums, e.g. population_within_20_minutes)
- local accessibility (e.g. local street-network based variable)
- composition of households and employment in neighborhood
- If retail-sector, population-seeking variables

Typical real estate price model explanatory variables include:

- distance to local amenities/disamenities
- building characteristics (e.g. year_built)
- regional accessibility (skim-based or logsums, e.g. employment_within_15_minutes)

- neighborhood characteristics (e.g. density, local accessibility, composition)
- Small-area vacancy rates
- Possible (only as needed): geographic dummies for local fixed effects

New variables are defined as python/orca functions in variables.py, and then the variable is added to a model specification using the notebooks, and then the model is estimated and evaluated. We check for fit and significance. If a key behavioral variable (e.g. accessibility) has an intuitive sign but is not significant, we may still retain it for sensitivity reasons.

After trying a set of intuitive behavioral variables, if the model fit is still low, we iteratively try other variables in the specification which have less intuitive interpretations. These less intuitive behavioral variables may be proxying for unobserved factors / unaccounted behaviors, and they help the model to have appropriate spatial associations if behavioral variables alone result in low measures of fit.

For any variable added to a model specification, we consider the resulting metrics:

- Variable significance (t-score)
- Model fit (r2, pseudo-r2)
- Inter-variable correlation matrix to check for multicollinearity (see the plots in the notebook). Correlation coefficients above .6 or so may lead us to reject a variable.
- Variable skew. Excessively skewed variables can result in unreliably estimated parameters. A skew value of greater than 5 or 10 often means we'll try log-transforming the variable to reduce skewnesss.
- Visual assessment of probability plots, or predicted price plots in the case of price models
- If a specification results in a warning being printed about lack of convergence, we make sure to re-run estimation, as the coefficients may not be valid.

MODEL ESTIMATION RESULTS

INTRODUCTION

A set of model estimation notebooks have been prepared to facilitate re-estimation of the parameters of the LCOG UrbanSim Model:

- Relocation-Tenure Binary Logit Models.ipynb
- Location Choice Model Estimation.ipynb
- Price Model Estimation.ipynb

When executed, these notebooks generate configuration files with persisted coefficients that can be committed back to the project repository. Each notebook allows for adding/dropping explanatory variables, reconfiguring and re-segmenting models, running estimation, and visualizing various diagnostics.

RELOCATION-TENURE BINARY LOGIT MODELS

The **relocation choice model** simulates whether a household decides to move to another housing unit or not. The discrete choice model used for this decision is the Binary Logit Model where the choice is represented in the *recent mover oneyear* column with 1: moved within the last year or 0: not moved within the last year. For better accuracy of the data, the team has decided to estimate the model based on the households table created from the disaggregate PUMS data.

After deciding to relocate or not, the households that do decide to move (their building_id is equal to -1), the **tenure choice model** will indicate whether they will rent or own their next living space. This model is also a Binary Logit Choice model and estimated on the pums data for data accuracy reasons. The decision variable is a dummy one called "tenure_own", where 1 means they own the house and 0 they rent it.

To account for the different behavior among the income groups, three more tenure choice models are calculated: 'tenure_choice_model1', 'tenure_choice_model2' and 'tenure_choice_model3'. The process is the same as the one explained above with the exception that, in the definition and register cells, a filter for income_quartile is added.

LOCATION CHOICE MODEL

After deciding to move and deciding on owning or renting a place, the newly-moved or created households have to choose a housing unit in a building to live in. The **households location choices** are simulated with a large multinomial logit model. To estimate the parameters, the definition cell sets that the *households* (choosers), specifically the ones who recently moved (chooser_filter), choose from the *buildings* (alternatives) a _building_id (choice_column) which have a constraint capacity based on the *residential units* (alt_capacity). The variables in the model specification are defined in the select_variables list. So, if the user would like to re-estimate the model with different variables, the select_variables list will have to be modified.

In the same way as the tenure choice model, different models are estimated for households in different income segments: low-income (first income quartile), Mid-income (second income quartile) and High-income households (third and fourth income quartile).

Employment location choice models are also estimated in this notebook. Large Multinomial logit models are estimated based on the 'jobs' table as choosers and the buildings and building_id as the alternative and choices respectively with the capacity constraint of job spaces in each building. Likewise the households location choices, the location decisions for jobs are also segmented. A model for each job sector is estimated.

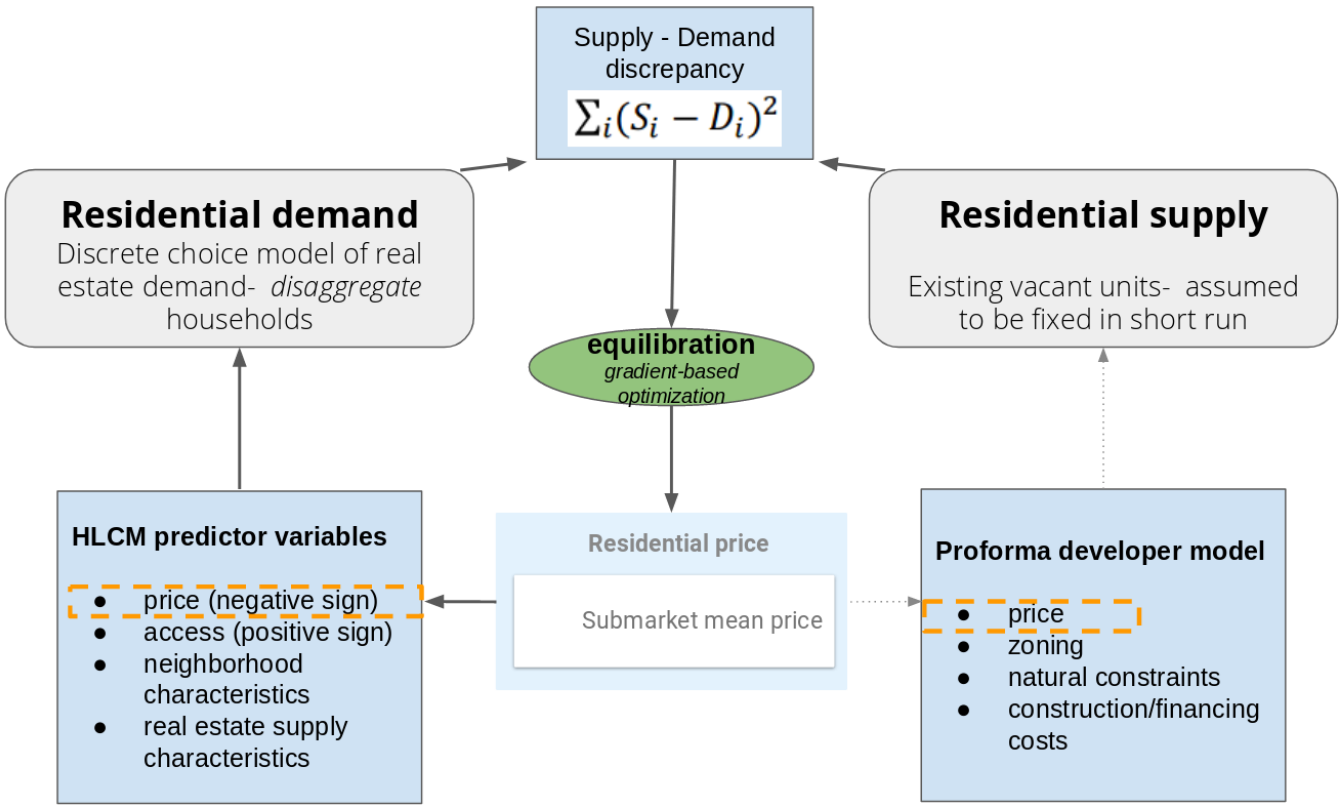
PRICE MODEL

Price data from current buildings enables the usage of regression models to estimate future buildings' prices based on various factors. Ordinary Least Squared- Regression Models are estimated. A specific model is calculated for each type of building for which we observ rents.

DEMAND – PRICE EQUILIBRIUM

Figure 1 summarizes the structure of the equilibration setup.

Figure 1



4. SUPPLY-SIDE MODEL: URBANSIM'S PROFORMA

DEVELOPER MODEL OVERVIEW

SUMMARY

This section provides a high-level overview of UrbanSim's developer model and its input parameters. The aim is to provide a general description of the process through which the model represents decisions taken by developers in the real estate market. Understanding the general logic behind the model, as well as the role of each input parameter, will allow refining the proforma inputs to better represent the context in LCOG's region.

Broadly speaking, the developer model is divided in two steps: feasibility and developer. The feasibility step tests multiple combinations of land use and Floor-Area-Ratio (FAR) for every parcel in the model, returning the most profitable FAR and building configurations for each land use combination in each parcel. This information is then used by the developer step to select the

parcels in which new buildings will be built to match existing residential and non-residential demand. A more detailed description of the two steps is given below, followed by a documentation of the parameters that go into the proforma.

FEASIBILITY STEP

The feasibility step simulates the typical process that a developer would undergo when deciding what type of development would be most profitable for a given parcel, and applies this same logic to all the parcels in the model at a time. The main process can be outlined as follows:

- The proforma is initialized based on the user inputs from the proforma.yaml file, including information about the specific forms that will be tested. Here, each form will represent a combination of land uses that could potentially be built in a parcel (i.e. 80% retail, 20% residential).
- The sites to analyze and their characteristics are defined based on the parcels table, removing previously pipelined development sites.
- For each form (corresponding to a given land use mix):
    - Each potential development site is assigned an acquisition cost that comes from the current yearly rent (either empirical data of rents in the city or forecasts).
    - The model estimates the costs and revenues that would result from building at different alternative densities in the site (This is done by estimating costs and revenues that could be obtained from different FARs in each site, with the list of FARs to test being specified by the user inside proforma.yaml).
    - Profit calculations for each potential FAR include the effect of parking requirements, parking costs, building costs at different heights, profit ratio requirements, building efficiency, parcel coverage, cap rate, among others.
    - Zoning constraints such as maximum FAR and allowable uses are taken into account at this point, filtering out those developments that are unfeasible or not allowed. For maximum FAR, the model selects the minimum between the max_far field, and the max FAR that would result from other zoning limits (max heights, max dua, etc).
    - The model generates a feasibility table with the building characteristics that yielded maximum profit for each development site. Building characteristics that make part of the feasibility table include FAR, parking configuration, building sqft, parking ratio, stories, construction time, residential sqft, non-residential sqft, building cost, finacing cost, total cost, building revenue, and profit.

The core cost and revenue calculations performed to select the most profitable FAR for each development site for each potential form (land use or land use combination) take place within the Square Foot Proforma API, inside the lookup() function of the feasibility step. The general logic for these calculations is the following:

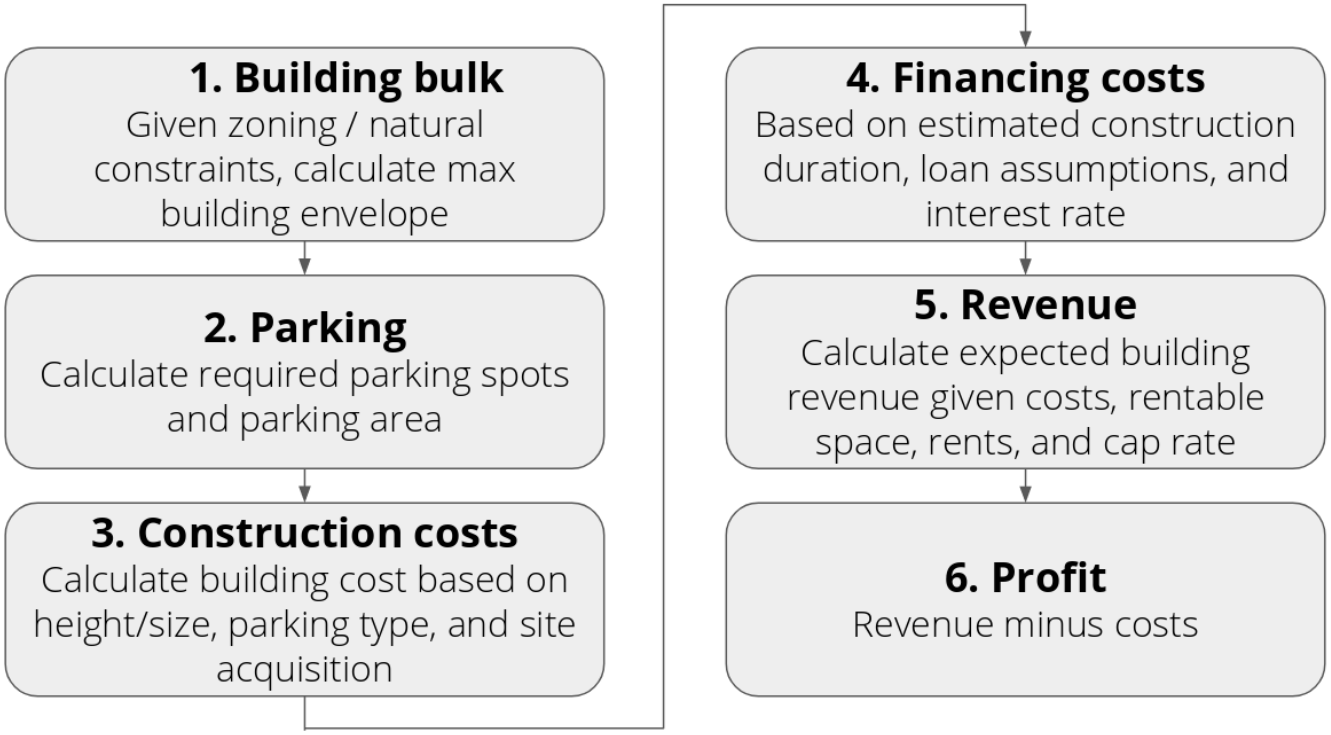
- Total building area (building bulk) is calculated multiplying FAR by the site area (square feet).
- Building costs are calculated multiplying built area by cost per square foot for the given building configuration.
- Total construction costs are calculated as the sum of building costs and land costs.
- The loan amount is calculated as total construction costs times loan-to-cost ratio.

- Financing costs are calculated based on the loan amount using the following variables: construction time, drawdown factor, interest rate, loan fees.
- Total development costs are calculated as the sum of construction costs and financing costs.
- To calculate the area that will generate rent, common areas and parking are subtracted from the total building area using the parking_sqft_ratio and building_efficiency variables.
- The area that generates rent is multiplied by weighted rent values and divided by the cap rate to calculate the revenue that will be generated by the building.
- Finally, the profit is calculated as the revenue minus total development costs.
- Costs, revenues, and profits are all allowed to be modified by the user through custom callback functions.

One important thing to note is that the feasibility step does all the profit calculations in terms of square feet, and has no representation of units (it does not differentiate between rent attained by 1BR, 2BR, or 3BR). Since getting data on unit mixes in the current building stock is extremely difficult, most feasibility computations here happen on a square foot basis, and the developer step handles the translation to units.

Figure 1 shows the sequence in which the various categories of computations undertaken by the feasibility step take place.

Figure 1: Feasibility steps in the UrbanSim proforma



DEVELOPER STEP

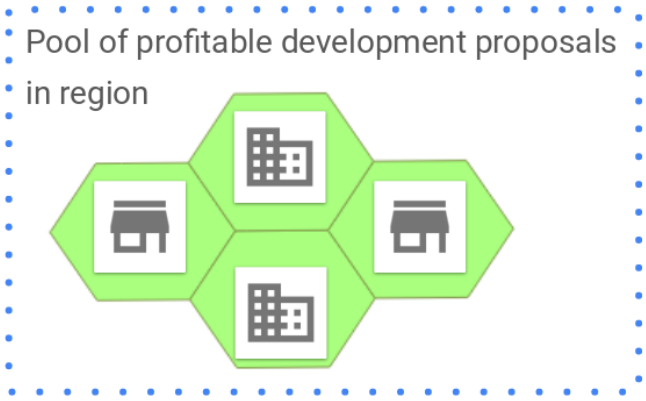
Having identified the development configuration that would maximize profit for each site-form combination, the main objective of the developer step is to select the sites where buildings will be added on a given simulation year to satisfy demand, and to modify the buildings table to reflect this extra capacity. The main input for the developer step is the feasibility table resulting from the previous step, as well as the demand for residential units and non-residential space on a given simulation year.

For a given simulation year, the developer step can be described as follows:

- The demand for residential units (target_units) is calculated based on the number of forecasted households, the number of existing residential units, and the target vacancy rate. Similarly, the demand for non-residential square footage is calculated based on the number of jobs generated in a given year, the number of available job spaces, and a target vacancy rate.
- The probability of selecting a given building/development is calculated based on the profit values from the feasibility table. The default function calculates this probability for each site in the feasibility table as the ratio between the profit per unit of area of the site and the sum of profit per unit of area over all feasible sites.
- Using the probability distribution over the potential development sites, the model runs a random function to select specific sites where new developments will be built to meet existing residential demand.
- Both the function to calculate probability based on profit values, and the function to select development sites based on the probability distribution can be customized by the user.
- Selected developments are dropped from the feasibility table.
- The buildings table is updated, adding extra capacity in terms of new buildings and new residential units.

Figure 2 summarizes the functionality of the developer step.

Figure 2: Developer step overview



PROFORMA CUSTOMIZATION POINTS

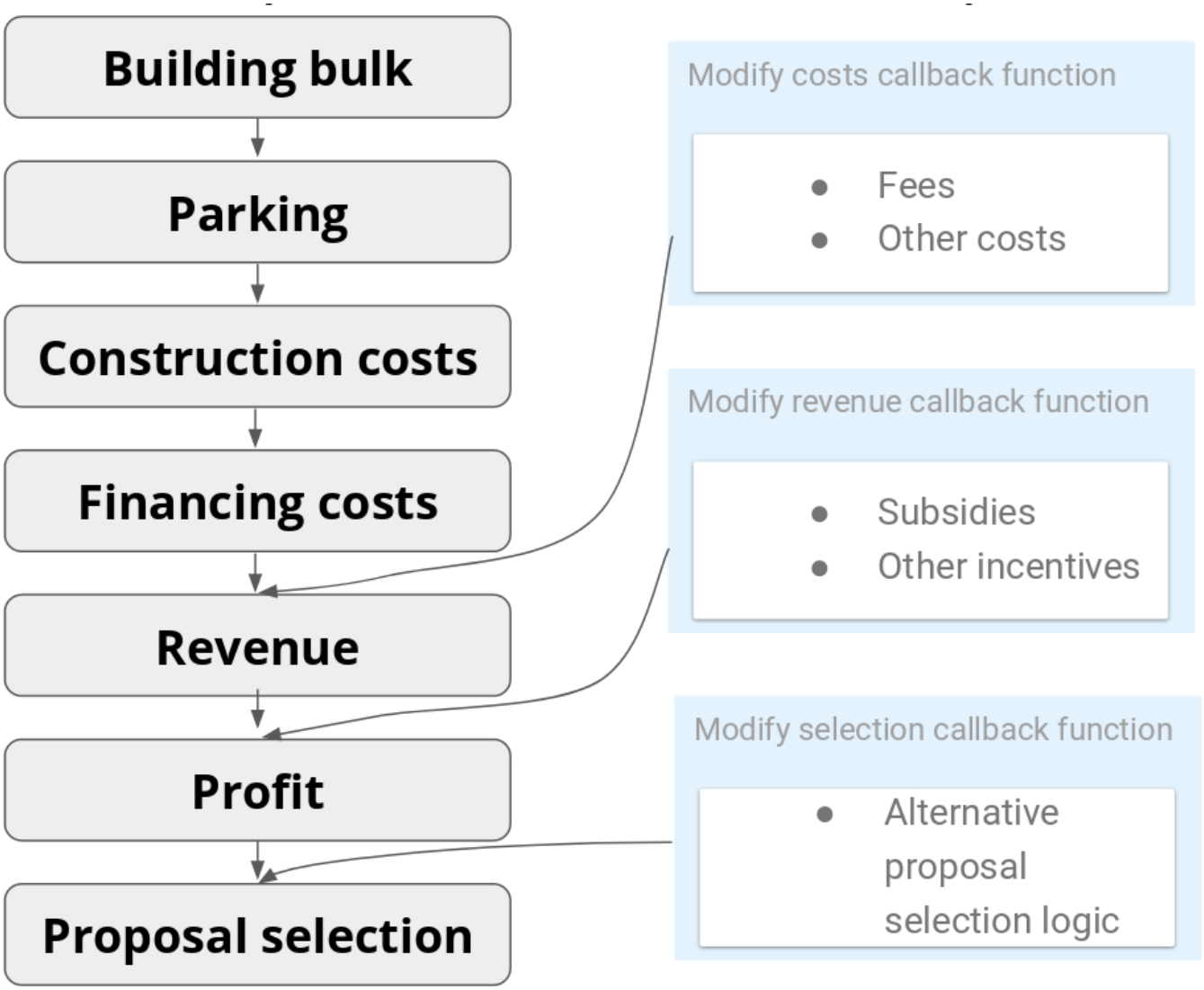
The UrbanSim proforma has particular points in its model flow where custom user logic can be readily inserted (see Figure 3 below). The user provides a function for one or more of these points, and the proforma model applies that function to customize the standard proforma mechanics. The customization points are:

1. Modification of costs
2. Modification of revenue
3. Modification of proposal selection

The first two options are the typical route for applying region-specific costs and incentives. For example, geographic-specific fees can be applied via #1, and geographic-specific subsidies can be applied via #2. The third option facilitates modification of the developer model's proposal selection logic- so adjustments to probabilities, or adjustments to the standard profitability-weighted random sampling, can be applied. These various hooks are already utilized by the LCOG model to apply various enhancements undertaken as part of the model development project. For example, rezoning fees are applied via #1, and #3 is applied to scale the selection probability of conditional uses.

Figure 3 shows the main points in the developer model flow where custom user logic can be inserted.

Figure 3: Customization steps in the UrbanSim proforma



PROFORMA PARAMETERS

The proforma parameters are used by the feasibility step to calculate profitability.

PARAMETERS LIST

fars: (float)

FAR is the ratio between built area (building bulk) and parcel area. The fars parameter in the proforma corresponds to a list of FARs that will be tested in each parcel in terms of profit.

uses: (list)

A list of land uses that will be represented by the model. This list only includes uses for which there is observed or estimated rent data. By default, the uses are retail, industrial, office, and residential.

residential_uses: (list)

A list with “true” for those uses that are residential, and “false” for those that aren’t.

forms: (dict)

In the developer model logic, a form represents a land use or land use mix to test. The forms parameter corresponds to a dictionary where keys are names for the form and values are also dictionaries where keys are uses and values are the proportion of that use used in this form. The values of the dictionary should sum to 1.0. For instance, a form called “residential” might have a dictionary of space allocations equal to {“residential”: 1.0} while a form called “mixedresidential” might have a dictionary of space allocations equal to {“retail”: .1, “residential” .9] which is 90% residential and 10% retail.

parking_configs: (list)

An expert parameter that is usually unchanged. By default, it is set to [‘surface’, ‘deck’, ‘underground’]. Very semantic differences in the computation are performed for each of these parking configurations. Generally speaking it will break things to change this array, but an item can be removed if that parking configuration should not be tested.

parking_rates: (dict)

A dictionary of rates per thousand square feet where keys are the land uses from the list specified in the uses parameter. The ratios are typically in the range 0.5 - 3.0 or similar. A key-value pair of “retail”: 2.0 would be two parking spaces per 1,000 square feet of retail. Since this is a per square foot pro forma, the typical parking ratio of spaces per residential unit must be converted to parking spaces per 1,000 square feet.

sqft_per_rate: (float)

The number of square feet per parking unit for use in the parking_rates parameter above. By default, this is set to 1,000 but can be overridden.

parking_sqft_d: (dict)

A dictionary where keys are the three parking configurations listed in the parking_configs parameter, and values are square foot uses of parking spaces in that configuration. This is to capture the fact that surface parking is usually more space intensive than deck or underground parking.

parking_cost_d: (dict)

The parking cost for each parking configuration. Keys are the name of the three parking configurations listed in the parking_configs parameter, and values are dollars per square foot for

parking in that configuration. Used to capture the fact that underground and deck are far more expensive than surface parking.

building_efficiency: (float)

In the model, the building_efficiency parameter turns total FAR into the amount of space which gets a square foot rent, since there are some common and open spaces in a building. On the other hand, the cost is calculated with the entire building area.

parcel_coverage: (float)

The ratio of the building footprint to the parcel size. Also used to turn an FAR into a height to cost properly.

height_per_story: (float)

The per-story height for the building used to turn an FAR into an actual height.

max_retail_height: (float)

The maximum height of retail buildings to consider.

max_industrial_height: (float)

The maximum height of industrial buildings to consider.

heights_for_costs: (list)

A list of “break points” as heights at which construction becomes more expensive. Generally, these are the heights at which construction materials change from wood, to concrete, to steel. Costs are also given as lists by use for each of these break points and are considered to be valid up to the break point. A list would look something like [15, 55, 120, np.inf].

costs: (dict)

The keys are uses from the uses parameter, and the values are a list of floating point numbers of same length as the “height_for_costs” parameter. A key-value pair of “residential”: [160.0, 175.0, 200.0, 230.0] would say that the residential use if $160/sqft up to 15ft in total height for the building, $175/sqft up to 55ft, $200/sqft up to 120ft, and $230/sqft beyond.

construction_sqft_for_months: (list)

Analogous to heights_for_costs, but for building construction time. A list of “break points” as building square footage at which construction takes a different length of time. Default values are [10000, 20000, 50000, np.inf].

construction_months: (dict)

Analogous to the costs parameter, but for building construction time. The keys are land uses from the uses parameter above and the values are a list of floating-point numbers of same length as the construction_sqft_for_months parameter. A key-value pair of “residential”: [12.0, 14.0, 18.0, 24.0] along with the default values for construction_sqft_for_months below would say that buildings with 10,000 sq. ft. or less take 12 months, those between 10,000 and 20,000 sq. ft. take 14 months, etc.

profit_factor: (float)

The ratio of profit a developer expects to make above the break-even rent. Should be greater than 1.0, (i.e. a 10% profit would be a profit factor of 1.1.)

cap_rate: (float)

A cap rate is often described as the ratio of annual revenue to initial investment. A developer will require a certain cap rate to consider a property to be profitable. Another way to think of the cap rate is as the maximum rate a developer is willing to invest initially in return for a certain cash flow per year in the future. This means a cash flow of $1/year is profitable if it costs no more than 1/cap_rate in present dollars. The ratio of 1/cap_rate can also be thought of as the acceptable number of years to reach full (100%) return on the initial investment. For example, a cap rate of 10% (cap_rate: 0.10) would have full ROI after 10 years (1/0.10 = 10). From this third way of thinking, a final definition of cap rate as 1/years-to-reach-full-ROI can be also be expressed. A cap rate is a macroeconomic input that is widely available on the internet.

loan_to_cost_ratio: (float)

The proportion of construction loans to the total construction cost.

interest_rate: (float)

The interest rate for construction loans

drawdawn_factor: (float)

The factor by which financing cost is reduced by applying interest only to funds withdrawn in phases.

loan_fees: (float)

The percentage of loan size that is added to costs as other fees

residential_to_yearly: (boolean)

Whether to use the cap rate to convert the residential price from total sales price per sqft to rent per sqft

forms_to_test: (list of strings – optional)

Pass the list of the names of forms to test for feasibility - if set to None will use all the forms available in config

pass_through: (list of strings - optional)

Will be passed to the feasibility lookup function - is used to pass variables from the parcel dataframe to the output dataframe, usually for debugging

simple_zoning: (boolean – optional)

This can be set to use only max_dua for residential and max_far for non-residential. This can be handy if you want to deal with zoning outside of the developer model.

only_built: (boolean - optional)

The feasibility step will return the buildings that are profitable when the only_built parameter is set to “True”, and will return both profitable and not profitable buildings when the parameter is set to “False”.

parcel_filter: (string - optional)

A filter to apply to the parcels data frame to remove parcels from consideration - is typically used to remove parcels with buildings older than a certain date for historical preservation, but is generally useful.

proposals_to_keep: (int - optional)

The number of feasible proposals to keep per parcel. This allows sub-optimal proposals with a given form to be retained. Sub-optimal proposals often represent lower-density outcomes. Defaults to 1, meaning that only the most profitable proposal for a given form is retained.

LCOG DEVELOPER MODEL ENHANCEMENTS

Enhanced developer model functionality was implemented for the LCOG model so as to represent re-zoning, special planning costs, and attribute updating related to city/UGB/overlay status. This section describes the enhancements, with a focus how things are implemented in the model.

REZONING

The first step in representing rezoning in the context of the current UrbanSim proforma is to create a table that represents every parcel/zoning_type combination, so that the feasibility step of the developer model can calculate the feasibility of every possible combination of parcel and zoning_type. The result is a computed table that we are calling `site_proposals` in the model, and it represents every combination of parcel/zoning that could be developed (given the input data), along with associated costs/probabilities. The resulting table is fed into the proforma so that the

model calculates the development feasibility of each possible parcel/zoning combination, accounting for associated costs.

If 5 zoning designations could potentially apply to a given parcel, then there will 5 rows in the `site_proposals` table for this parcel. The plan_compatible_zones table indicates which zoning_id's are associated with each plan, and the plan_id of each parcel is then used to look up the possible zoning_id's a parcel could have given re-zoning. Only zoning designations where `can_rezone == True` are allowed to be re-zoned.

Rezoning costs are from the plan_compatible_zones table, and they account for whether a parcel is within-the-ugb / outside-the-ugb / within-city. The parcel table notes the status of each parcel with regard to it's location within-the-ugb / outside-the-ugb / within-city.

COST ADJUSTMENTS

LCOG has provided UrbanSim with development costs related to factors related to zoning, rezoning, conditional-uses, and annexation. These costs vary by location. UrbanSim coded a function to apply the relevant costs associated with each site proposal (combination of parcel and possible zoning ID), and then applied these costs in the proforma via the cost call-back functionality. For example, LCOG provided cost columns in the `zone_overlay_types`, `allowable_building_types`, and `plan_compatible_zones` tables, and these are now incorporated into the feasibility calculations.

With these costs in place, profitability in the proforma is influenced. The cost functions were tested by simulating with various input values for an example parcel. When costs exceeded a certain threshold, development was no longer financially feasible. When costs were decreased, profitability increased.

CAPACITY REDUCTION FACTORS

A developer model enhancement was implemented to adjust developable capacity downwards for large parcels. This adjustment represents the notion that net developable land should account for land set-aside for infrastructure and amenities[1], and the adjustment will be different for different sized parcels. The model does not adjust the land area attribute on the parcels table directly, but only adjusts land area as perceived by the developer model. This functionality will help to prevent over-building on larger parcels in the model.

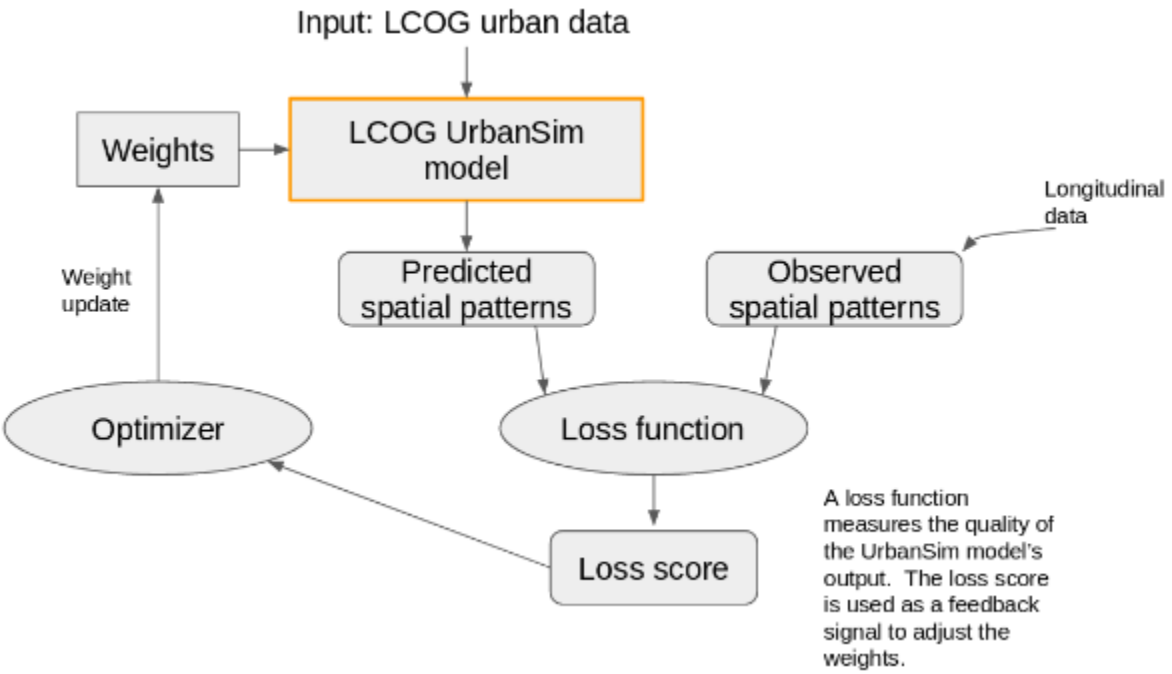
5. CALIBRATION, VALIDATION, AND MODEL OUTPUTS

MODEL CALIBRATION METHODOLOGY

This section documents the approach to UrbanSim model calibration that we took for the LCOG UrbanSim land use model project. The core concepts behind the UrbanSim calibration approach taken for this project are *automatic differentiation* and *gradient-based optimization*. Whereas in past UrbanSim projects we have used brute force or black-box optimization methods to calibrate

the model to observed longitudinal data, for this project we use automatic differentiation plus gradient descent to calibrate the model. Figure 1 shows a high-level overview of the calibration process.

Figure 1: Calibration overview



The general idea is to have each model component (the location choice and proforma models) generate summed probabilities across alternatives. Add an objective function at the end (e.g. how well observed longitudinal patterns are matched), and use differentiation to calculate the gradient of the loss with respect to the model's parameters, which we can then optimize with gradient descent. Auto-differentiation plus gradient-descent is the same way a neural network is trained (i.e. the backpropagation algorithm), so we can use the same libraries to implement.

Summed probabilities, in particular capacity-constrained summed probabilities, are a good proxy for actual simulation outcomes, as they represent all simulation logic except the monte carlo step (so outcomes are float values of expected growth instead of discrete values). The summed probabilities are then calibrated to approximate observed longitudinal data on spatial patterns of growth using gradient descent. Being able to calculate exact gradients speeds up the calibration process compared to previous approaches, and also facilitates calibrating behavioral variable coefficients directly instead of just spatial dummies.

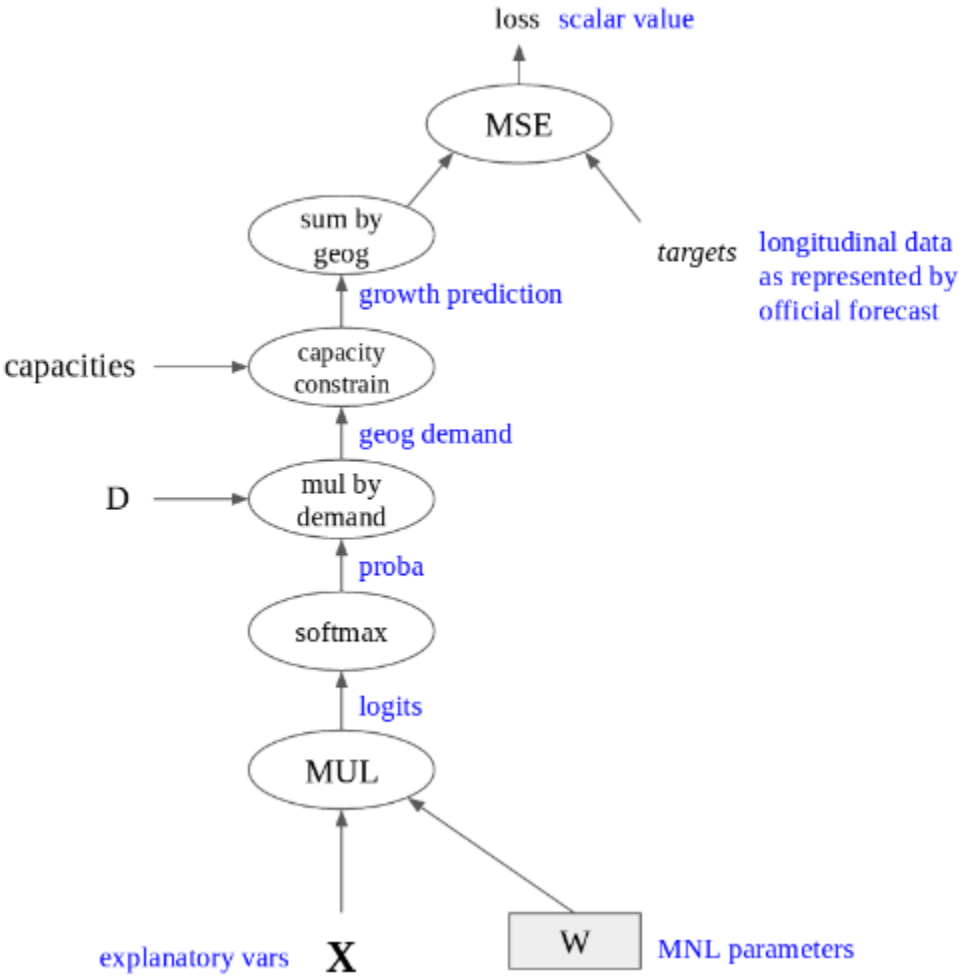
For calibrating location choice models, the steps are:

1. Multiply explanatory variables by coefficients to get logits
2. Apply softmax transform to logits to get probabilities
3. Sum the probabilities by geography

4. Calculate loss by taking the mean squared error (MSE) between summed probabilities and observed data on spatial patterns of longitudinal growth
5. Take gradient of the loss function with respect to the location choice model parameters
6. Pass gradients to an optimizer that performs gradient descent to adjust the model parameters in the direction that lowers the loss.

Figure 2 shows the computation graph for calibrating a particular location choice submodel. This is the graph through which we trace gradients and conduct gradient-based optimization. The backpropagation algorithm, as implemented in automatic differentiation libraries, allows us to compute the derivatives of this graph via the chain rule of calculus.

Figure 2: Computation graph for calibrating a location choice submodel

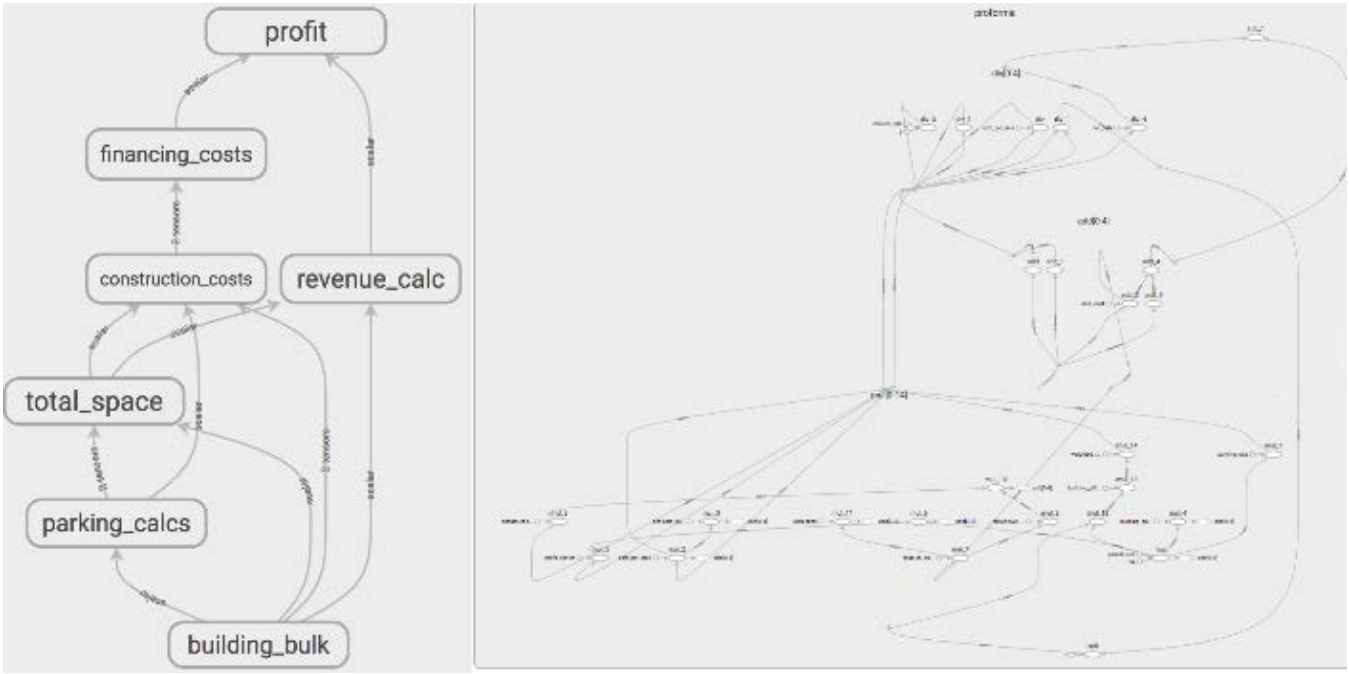


For calibrating the proforma model, the first step was to code a differentiable version of the current UrbanSim proforma, as differentiability is a prerequisite for calculating exact gradients using automatic differentiation. Given gradients, we can then optimize the proforma. The parameters being optimized are spatial cost shifters (coefficients on geographic dummies) as well as cost shift coefficients on behavioral variables from the demand models. These various cost-shift parameters are tuned such that the proforma generates results that approximate observed longitudinal

patterns of supply growth. A differentiable proforma, through which we can trace gradients, also means that any of the proforma parameters could be optimized given data, not just cost shifters, but for now we restrict the calibration to the cost shifters. A differentiable proforma that is optimized via gradient descent given observed data can be considered more of a `learned` model rather than simply a rule-based profitability calculator that previous proformas represented. To make the proforma differentiable, the main change was to replace the height-to-cost and space-to-construction-months step functions with continuous approximations (since step functions aren't differentiable).

To calibrate the proforma, we calculate the gradient of the MSE loss function with respect to the proforma's cost shifter parameters. We then optimize the parameter values to minimize the loss. Figure 3 shows the computation graph of a proforma model- the chain rule is applied to this graph to get derivatives.

Figure 3: Computation graph of the current UrbanSim proforma



In both location choice model calibration and proforma calibration, we conduct reverse-mode differentiation (i.e. backpropagation) on the computation graph to calculate gradients of the scalar-valued loss function (mean-squared error between simulated/observed outcomes) with respect to array-valued arguments (the various model parameters we want to calibrate). We then pass the gradients to an optimizer and do gradient-based optimization to adjust parameter values and minimize the loss.

MODEL CALIBRATION RESULTS

CALIBRATION GOALS

- Move relative spatial variation of simulated growth towards observed patterns

- Proxy for unobserved costs and variables not accounted for by the location choice models as specified
- Incorporate information from longitudinal data (model estimation is based on cross-sectional data)

CALIBRATING THE PROFORMA MODEL

UrbanSim's model of real estate supply simulates the location, type and density of real estate development at the level of specific parcels. Proforma-based profitability calculations for every parcel in the region are run each simulation year. The calculations account for variables such as price, costs by structure type, fees, and zoning. This section describes the calibration of the real estate supply predictions. simulated, with adjustments made after each iteration to move the simulation in the direction of the spatial pattern of growth represented by tract-level data. Note: the calibration period was simulated without the scheduled_development_model (i.e. pipeline projects), so that UrbanSim was responsible for all supply predictions during calibration runs.

The 2010-2015 period was iteratively simulated (the model base-year is currently 2010). After each iteration, cost shifters in the proforma were updated in the direction that would move the simulation towards the targets. Cost shifters are a function of the model's explanatory variables, and the parameters of the cost shifters are learned as part of the calibration process. For example, if low density locations tends to have growth undershot, parameters associated with low density locations will be adjusted so that new real estate development becomes more attractive. Calibrated parameters are tuned separately for residential/non-residential. Parameter updates are based on gradient descent, with gradients calculated based on auto-differentiation through the proforma logic and a mean-squared-error loss function.

It’s important to note that although the model was calibrated over a few simulation years, calibration does not pre-determine model outcomes: typical simulations will be run well beyond the calibration period to 2040 and beyond, and UrbanSim accounts for a wide variety of variables that will influence growth separate from the calibration process. For example, locations that have historically grown very rapidly can run out of zoned capacity halfway through the simulation, shifting growth to locations that historically experienced slower growth. Changing congestion effects and price effects can also influence the spatial distribution of growth in the simulation. These are examples of complex feedbacks that UrbanSim is designed to represent.

Calibration notebooks were prepared so that the proforma calibration process can be replicated.

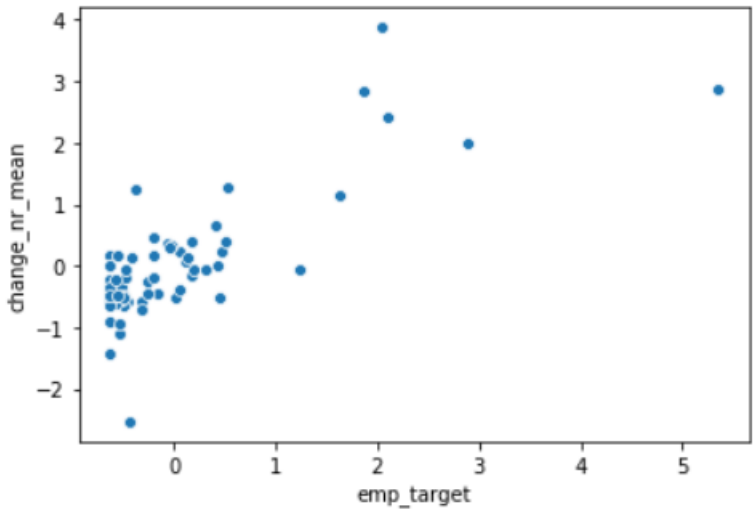
METRICS

Validation metrics for Dwelling unit growth, 2010 - 2015

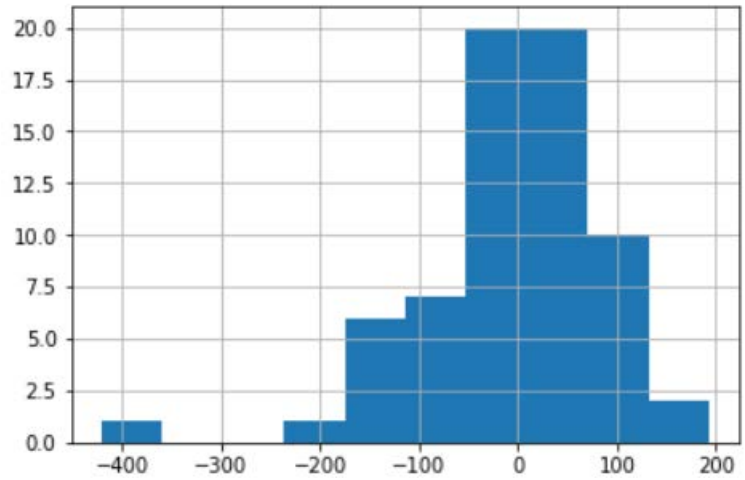
- MSE is 1.8948610753713644
- RMSE is 1.37653952917138
- Prediction R2 is 0.5207503259568577
- Correlation is 0.760375162978429

Validation metrics for Nonres-space growth, 2010 - 2015

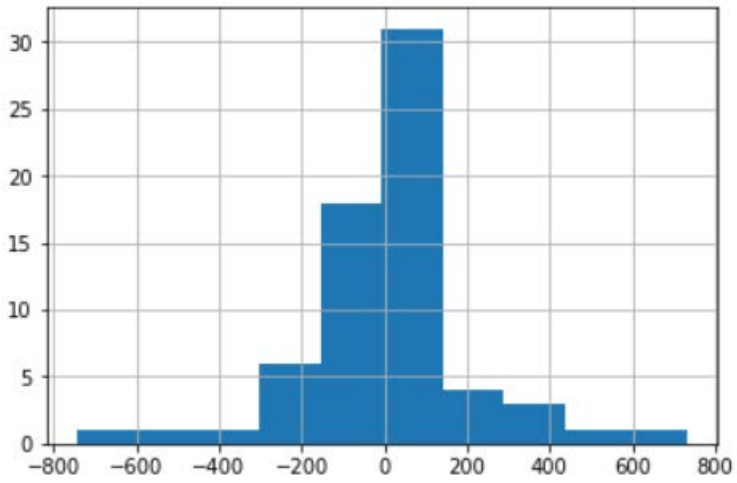
- MSE is 1.868264823081719
- RMSE is 1.3668448423583852
- Prediction R2 is 0.566758741640798
- Correlation is 0.7833793708203989



Dwelling unit deltas



Nonres-space deltas



MODEL VALIDATION

We run land use models to try and forecast future socio-economic and land use patterns given various sets of assumptions and policies. We desire to validate the land use model to help confirm that the model reasonably represents urban growth and to better understand the model so as to use it appropriately. Model validation can take many forms. In the current project, model validation will consist of 1.) comparing simulation results to observed longitudinal data, and 2.) sensitivity tests.

1. COMPARING MODEL RESULTS TO LONGITUDINAL DATA

The model system as a whole was run from 2010 to 2016, and the simulated changes were compared to longitudinal data on observed changes in the same period to assess model performance (for unit and household change, the 2013 ACS 5-year and the 2018 ACS 5-year were used for observed data). The table below shows the tract-level correlation between simulated and observed for each dimension of change, and the plots below summarize the comparison.

Table 1: Tract-level correlation between simulated and observed, 2010 - 2016

change_type	correlation
dwelling unit	0.618
household	0.472
non-residential sqft	0.704
job	0.656

Figure 1: Scatter plot of simulated vs observed dwelling units, 2010 – 2016

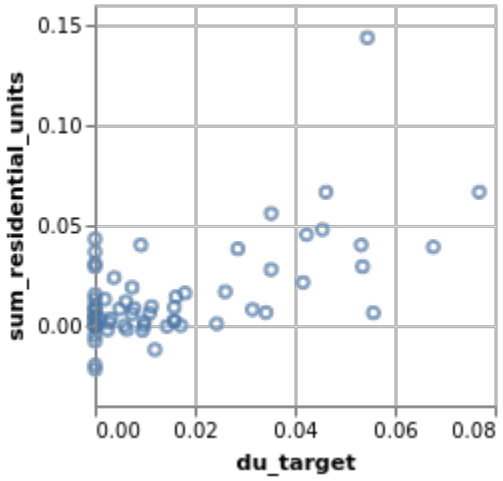


Figure 2: Scatter plot of simulated vs observed nonres-sqft, 2010 – 2016

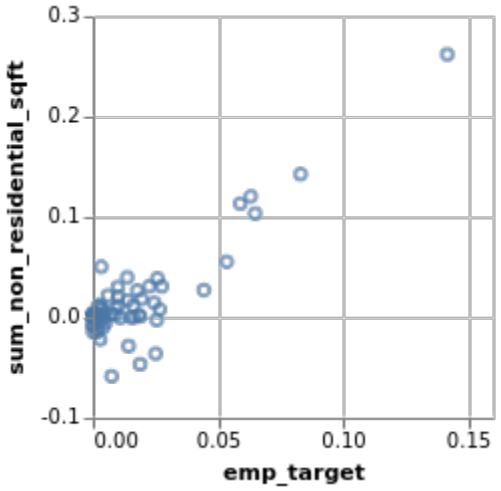


Figure 3: Scatter plot of simulated vs observed households, 2010 – 2016

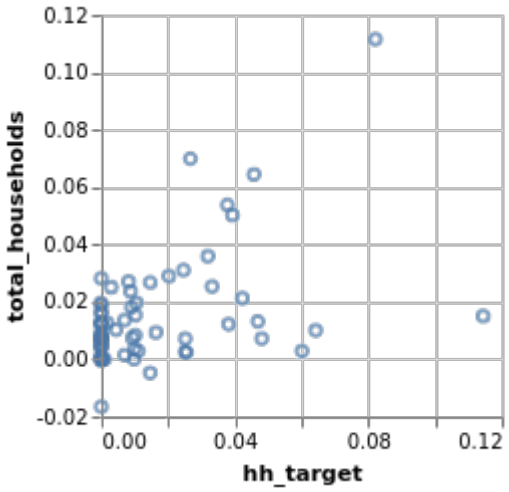
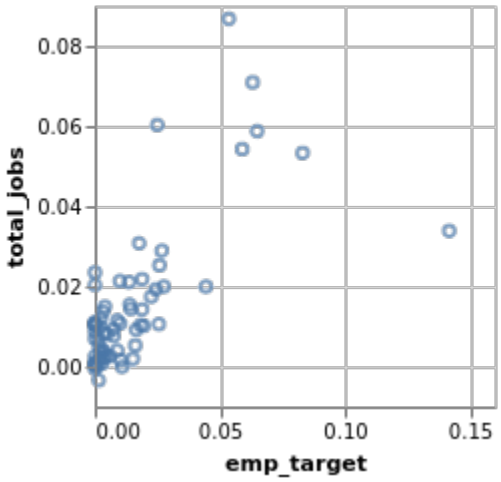


Figure 4: Scatter plot of simulated vs observed jobs, 2010 – 2016



STOCHASTICITY EVALUATION

UrbanSim is a microsimulation-based model, and there will be random variation from run to run when the random seed is allowed to vary. The forecast will be different for each possible value of the random seed. Most major model components in UrbanSim have a random component, often involving converting a probability distribution into a set of discrete decisions using monte carlo simulation. Monte carlo simulation is used because we are modeling the discrete decisions of individual decision-makers: UrbanSim's data structures are disaggregate, and UrbanSim treats aggregate outcomes as the result of many individual decisions. This has implications for interpretation of model results. Output from a single model run is a single "draw" from the distribution of possible model outputs. Benefits that are a byproduct of microsimulation include more realistic accounting of agent heterogeneity, being able to summarize outputs for any agent type (supporting things like equity analysis), and allowing model specifications to focus on behavioral factors that apply in a disaggregate modeling context. If results need to be replicable, the random seed can always be fixed.

To evaluate the extent of stochasticity, so as to inform appropriate use of the model, the 2010 to 2015 period was simulated 120 times, and results summarized at the tract level. Note that these runs assumed a simple %1 annual growth rate for both households and jobs, to make the effect of randomness easier to isolate. For each major dimension of change (households, dwelling units, employment, non-residential square footage), the mean and standard deviation of tract outcomes was calculated. 120 runs provides a large enough sample of runs to get a reasonable sense for the central tendency and breadth of model outputs. The results indicate that tract-level results do vary from run to run, but are reasonably stable overall. Figures 1-4 below show, for a single example tract (tract_id: 41039005200), histograms of indicator outcomes across the runs.

Figure 1: Histogram of indicator outcomes for tract 41039005200

OUTPUT INDICATORS AND CHARTS

INTRODUCTION

The LCOG model currently exports key output indicators and charts to the `lcog/lcog/runs` folder in `.csv` and `.json` format, which then get exported to the platform user interface for visualization.

This section describes the main logic behind the process that generates output indicators and charts, and presents instructions that will enable the user to generate custom indicators and charts. The code structure is explained first, followed by a description of the parameters that can be changed to customize outputs.

CODE STRUCTURE

Output indicators and charts are generated in the `models.py` script, as part of the `generate_indicators()` function. The function itself is divided into three main sections, which are presented below.

1. General output indicators:

Based on custom parameters that can be defined in `configs/output_parameters.yaml`, this section exports two `.csv` files for each simulation year. The first file (`parcel_indicators_general_year.csv`) contains the selected output variables for each parcel, and the second one (`zone_indicators_general_year.csv`)contains the selected output variables for each zone.

2. Output indicators by building type:

This section exports the same output variables defined in `configs/output_parameters.yaml`, but disaggregated by building type. Variables that are exclusive of residential building types are only calculated for buildings that have `is_residential==True`, and variables that are exclusive of non residential types are only calculated for building types that have `is_non_residential==True`. As with the general output indicators, this section exports one file for parcels and one file for zones (`parcel_indicators_building_type_year.csv` and `zone_indicators_building_type_year.csv`, respectively) .

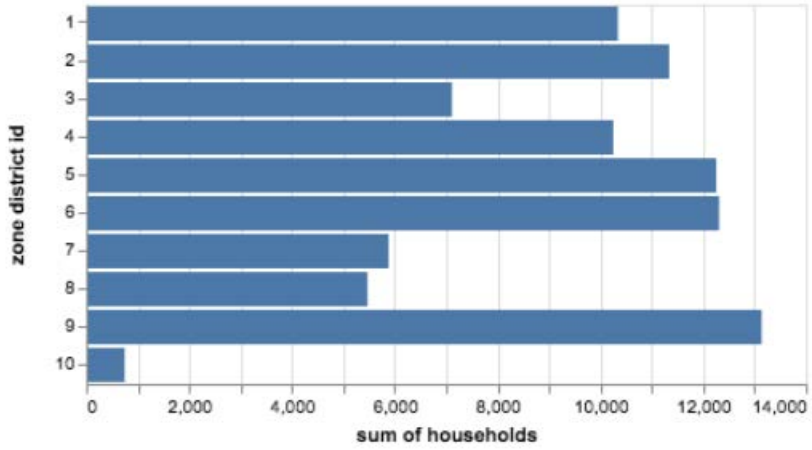
3. Chart indicators:

When the simulation reaches the `forecast_year`, five main types of charts are exported for each output variable, and also for each variable by acre. The five types of charts are described below, together with examples of the charts generated when using `total_households` as a variable, `zone_district_id` as `large_geography`, and `block_id` as `small_geography`. Output variables and aggregation geographies can be modified in `configs/output_parameters.yaml`, as it is described in the last section of this page.

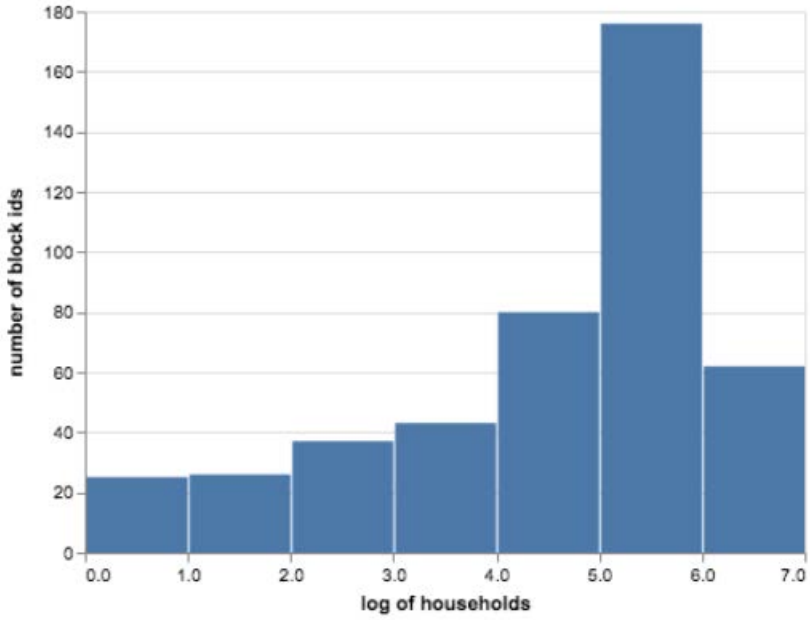
Charts generated using data from the zones table

Most relevant charts can be created by aggregating data available at the parcel and zone level. To facilitate plotting in terms of the size of the dataset to plot, the main charts generated by the model start from data at the zone level. Four types of charts are generated from zone data for each variable. The examples below show these four types of charts, where `total_households` is the variable, `zone_district_id` is the value for the `large_geography` parameter, and `block_id` is the value for the `small_geography` parameter:

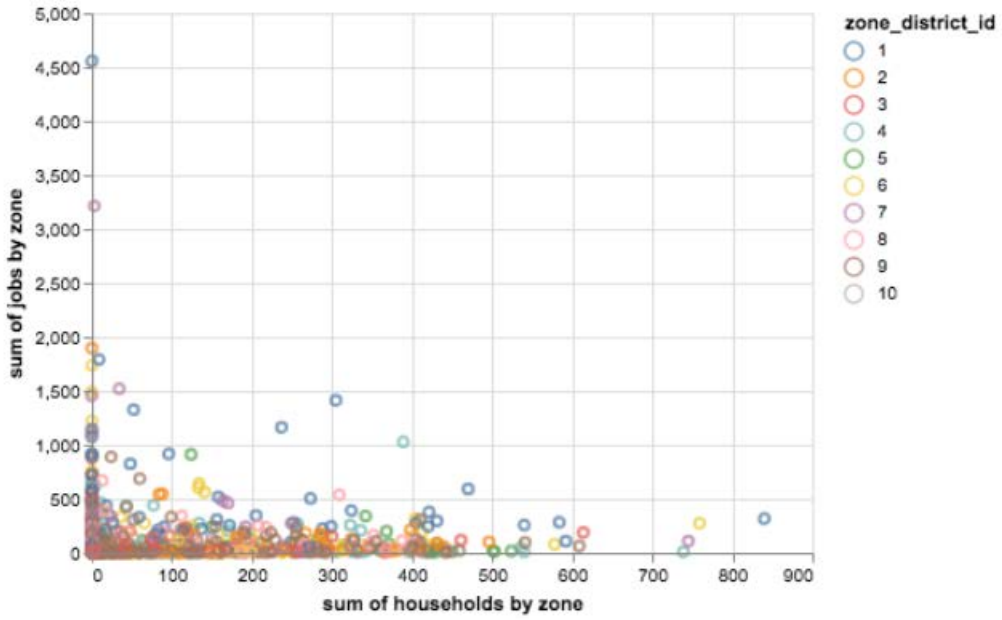
- Bar chart showing the sum or mean of the variable by an aggregate geography.



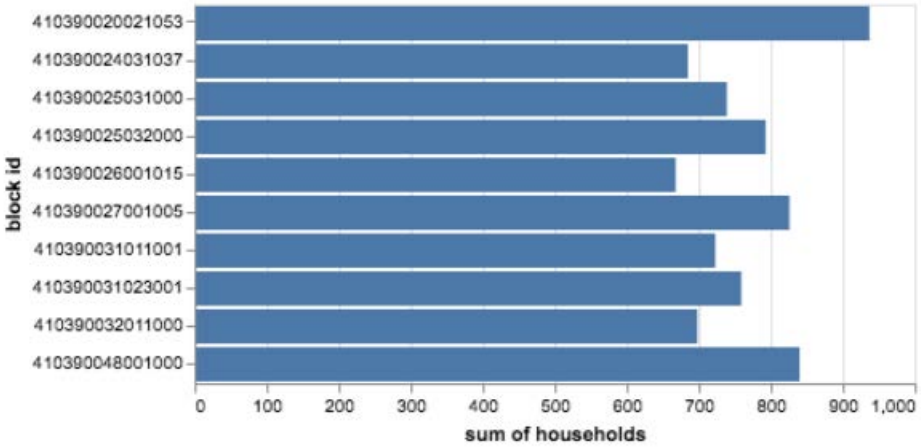
- Histogram showing the count of geographies that have different values for the sum/mean of the variable. The x scale was transformed to the logarithm of the variable:



- Scatter plots comparing the sum/mean of the variable with the sum/mean of all other predefined output variables (Individual data points correspond to zones):

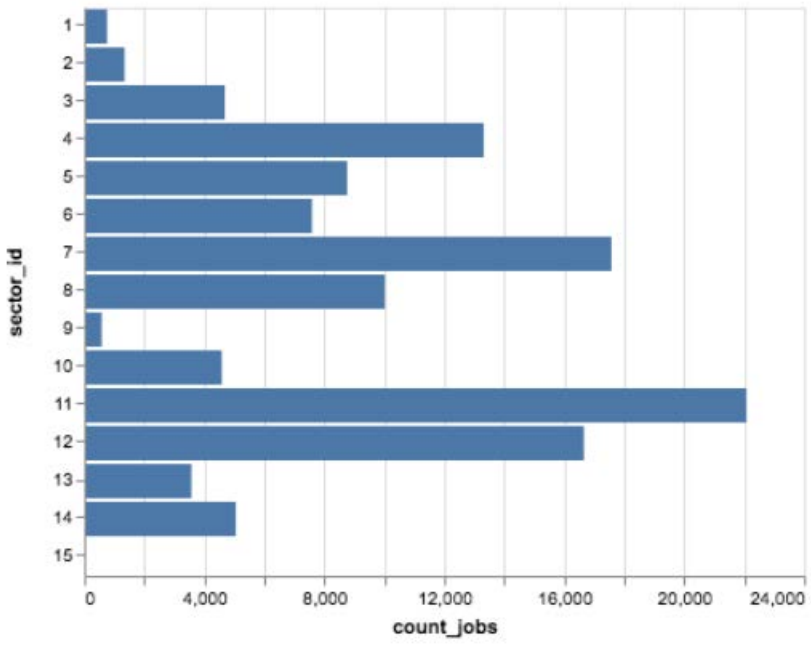


- Bar chart with the `block_id`'s that have the top 10 sum or mean values for the variable



Charts generated using data from the jobs table

While the most relevant charts can be created by aggregating data available at the zone level, the user might also need to generate charts from other types of data. One example of this is a bar chart of number of jobs by `sector_id`, which is currently being generated by the model:



INPUT PARAMETERS

The main input parameters used to generate output indicators and charts can be customized within the `configs/output_parameters.yaml` file. To modify the variables that are exported after each simulation year or add new charts for the forecast year, the following fields can be modified by the user within the `.yaml` file:

- output_variables:
  - parcels: list of the parcel variables that will be exported as a `.csv` for every simulation year. These can include the variables shown below, plus any of the the proportion variables and spatially smoothed variables described in `adds_dict_proportions()` and `adds_derived_vars_dict()` :
    - total_households
    - total_jobs
    - sum_residential_units
    - sum_residential_sqft
    - sum_job_spaces
    - sum_non_residential_sqft
    - sum_persons
    - sum_workers
    - sum_children
    - sum_cars
    - sum_acres
    - sum_income

        - sum_recent_mover
        - mean_sqft_per_unit
        - mean_job_spaces
        - mean_year_built
        - mean_sector_id
        - mean_acres
        - mean_persons
        - mean_workers
        - mean_children
        - mean_cars
        - mean_income
        - mean_age_of_head
        - mean_x
        - mean_y
        - mean_value_per_unit
        - mean_value_per_sqft
        - median_building_type_id
        - median_income_quartile
        - median_tenure
        - median_race_of_head
        - median_sector_id
        - density_households
        - density_jobs
        - ratio_jobs_to_households
        - ratio_workers_to_persons
        - ratio_households_to_residential_units
        - residential_vacancy_rate
        - non_residential_vacancy_rate
        - remaining_nonresidential_sqft_capacity
        - remaining_residential_unit_capacity
    - zones: list of the zone variables that will be exported for every simulation year. These can include all of the parcel variables, plus
        - residential_vacancy_rate
        - non_residential_vacancy_rate
- chart data:
    - geography_large (str, default is `zone_district_id`): id of the geography that will be used to aggregate zone data into categories for the first type of chart presented above. To generate a manageable number of categories in the chart, this field should have 15 or less unique values.

- geography_small (str, default is `block_id`): id of the geography that will be used to generate a histogram of the variable, and to create a bar chart with the top 10 values of the variable.
- chart_variables (dict): dictionary of the variables that will be used to generate the four initial types of charts described above. The keys are the aggregation functions that will be used (sum or mean), and the values are the variables that will generate charts for each aggregation function.
  - sum (list): variables that will be aggregated into charts using sums. These can include variables such as total_households, total_jobs, sum_residential_units, sum_non_residential_sqft, sum_job_spaces, sum_persons, sum_workers, sum_children, sum_cars, sum_hispanic_head, sum_recent_mover, sum_acres, sum_land_value.
  - mean (list): variables that will be aggregated into charts using the mean value. These include variables such as residential_vacancy_rate, non_residential_vacancy_rate, mean_income, mean_value_per_unit.
- custom_charts (dict): dictionary of the variables that will be used to generate the fifth type of chart described in section 3 of this wiki, which is not based on zones data but directly on households, buildings, or jobs data. The keys are the names of the tables containing the variables of interest (households, buildings, or jobs), and the values are the variables from these tables that will be used to generate custom bar charts.

Note: the `gen_custom_barchart()` function currently only supports the creation of bar charts of an agent aggregated by existing categories of a variable (`households` by `income_quartile`, `buildings` by `building_type_id`, `jobs` by `sector_id`).

6. PROJECT COORDINATION

HIGH-LEVEL PROJECT GOALS

This application of UrbanSim to the Eugene-Springfield Metro region has been developed in close collaboration with LCOG/CLMPO staff.

The focus of the model development effort was on developing a model system for forecasting and scenario analysis, tied to the UrbanCanvas Cloud Platform. Location choice models and real estate price models were implemented, along with a proforma-based developer model and other supporting model components.

The first phase of the LCOG UrbanSim model development process began in February 2018 and is expected to be completed in July 2019. This is the first version of the model, with 2010 input data. The second model development phase will update the base-year data to 2016, re-estimate certain models based on new data/variables, and conduct additional calibration, validation, and sensitivity testing. The goal of the first two phases of development is to have a well-built and well-understood model by January of 2020.

HIGH-LEVEL PROJECT SCHEDULE

- *2018*- Initial base-year data development and setting up initial model structures
- *2019*- Model estimation, calibration, deployment, indicators, and enhancements
- *2020*- Finish current-phase model, base-year data transition

DELIVERABLES

This page lists the deliverables associated with the complete working model. These deliverables will allow LCOG to run the model (both locally and using the cloud platform) and to run associated model development scripts.

- Documentation of data processing
  - ☑ Data pre-processing pipeline
- Documentation of enhancements
  - ☑ Proforma enhancements
  - ☑ Other enhancements
- Documentation of model estimation
  - ☑ Use of the model estimation notebooks
  - ☑ Model estimation results
- Documentation of model function
  - ☑ Developer Model Memo
- Documentation of model calibration/validation/sensitivity-testing
  - ☑ Summarizing model evaluation and use of the relevant notebooks
  - ☑ Calibration
    - > Methodogy
    - > Results
  - ☑ Validation
  - ☐ Sensitivity Testing

- ☑ Stochasticity

- Documentation of outputs
  - ☑ Output indicators and charts
  - ☑ How to change the standard output indicators of the model
- Documentation of model installation and synching
  - ☑ Installing the model locally
  - ☑ Keeping the cloud-based model synched via Github
- On-site training meeting
  - ☑ *Occurred on January 28-29, 2019*

MODEL DEVELOPMENT WORKFLOW DIAGRAM