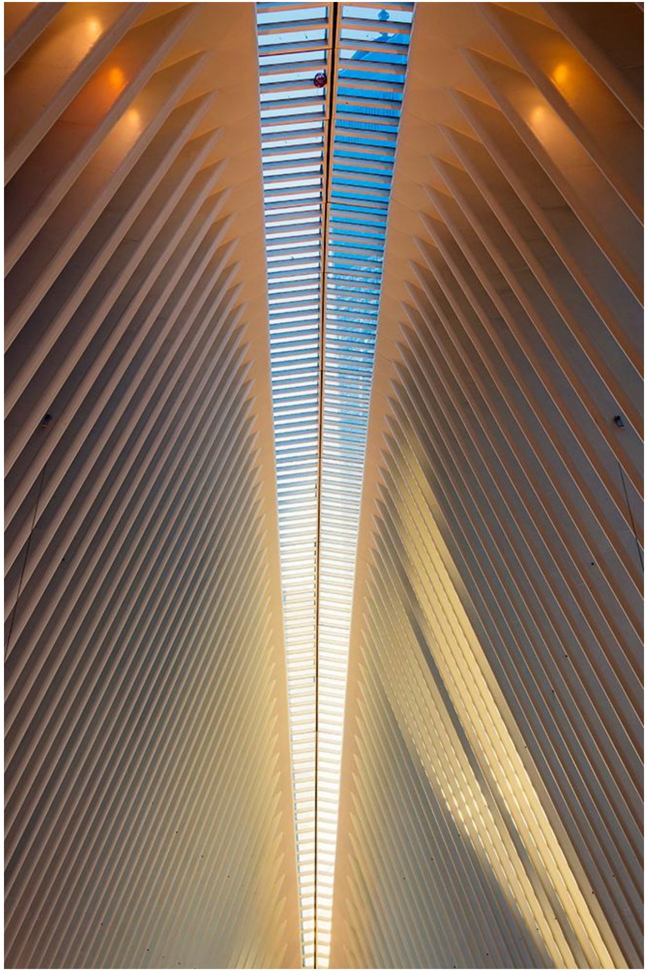
Housing Market Price Forecasting

CSCI 4360 Term Project
Ayush Kumar, Brandon Amirouche,
Faisal Hossain
Spring - April 27, 2021

Objective

Use housing data to forecast pricing using monthly and yearly data



Agenda

Datasets 01

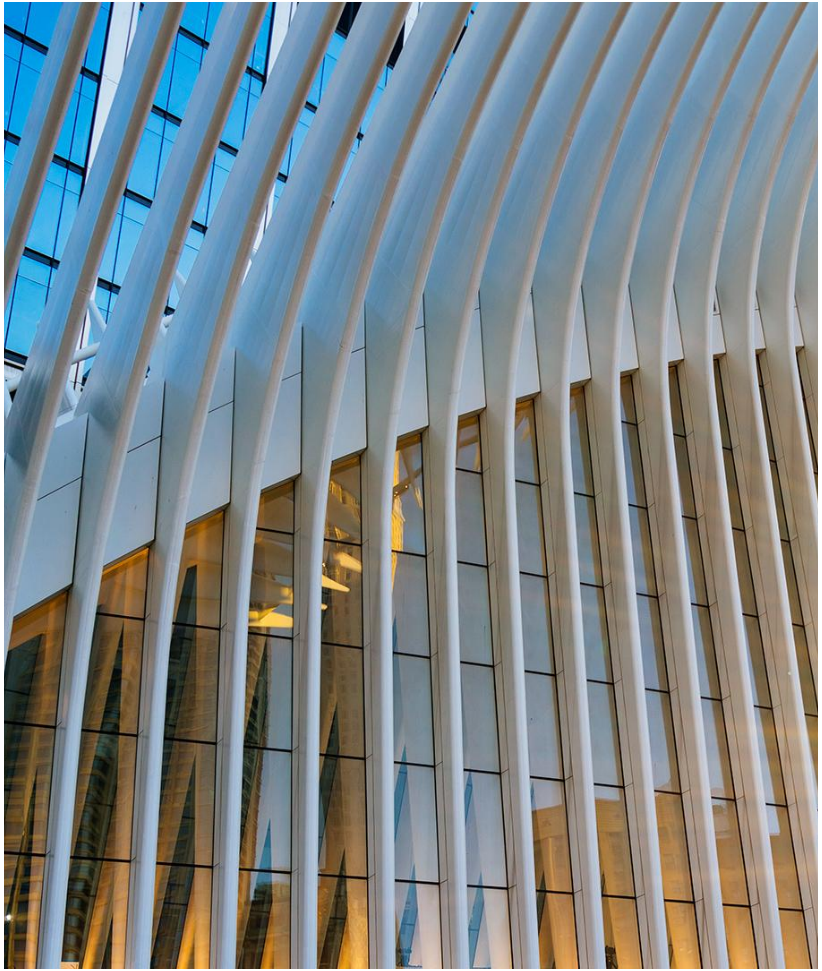
Exploratory Data Analysis and Pre-processing

Model Analysis 02

Autoregressive, SARIMA, Long Short Term Memory, Gated Recurrent Unit

Conclusion 03

Model Performance Comparison



Quandl Zillow Real Estate Dataset

The Zillow data feed contains real estate market indicators, such as market indices, rental, sales, and inventories for thousands of geographical areas across the United States. Data is monthly and updated weekly on Sundays by 1PM UTC.

78,200+

Regions in the US

56

Indicators

3

Categories:
Home values, Rentals, Sales & Inventories

Housing Mortgage Disclosure Act

The **HMDA** was enacted in 1975, and required mortgage lenders to report lendings to government agencies. This data is now publicly available and is you can download it by year.

- Our project used data from **2014 - 2017**
- **40 million+** observations
- Aggregated to produce yearly summary data that supplemented our RNN models
- Data has much more potential, but we had to limit the scope to just time-series analysis
- Total size of was **40GB** so we had to create a **SQLite3 database** and use pipelines to transform the data

Preprocessing - HMDA

The HMDA dataset contains 79 features, including many about the race of applicants, the location they applied, and the loan amount. For the purposes of this project we used the following filters:

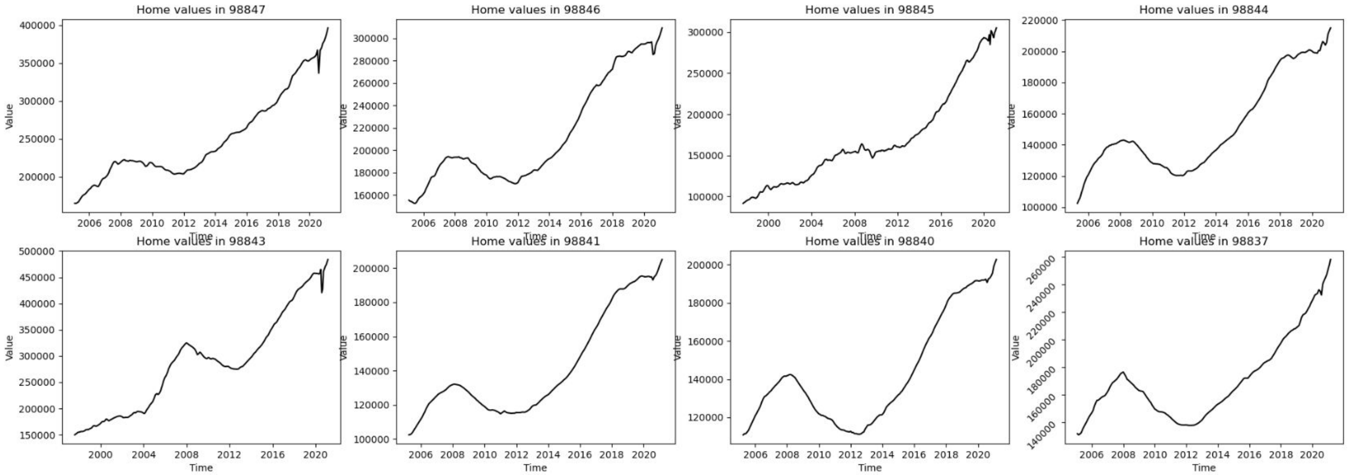
- Property_type is only one-to-four family dwelling
- Approved mortgages (house was actually bought)
- Loan_purpose was home-purchase

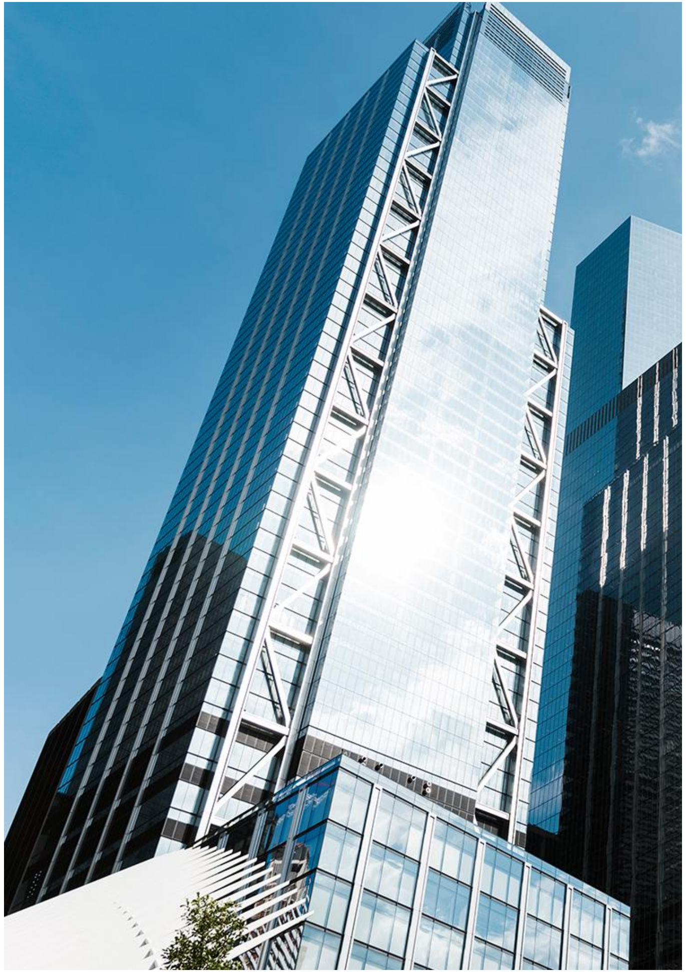
We then aggregated the following features on state

- Loan_amount_000s
- Population
- Minority_population
- median_family _income

Preprocessing - Zillow

The zillow dataset included only three important variables timestamps, values, and location. We will be using the zillow dataset to build our time-series models. Here are some aggregate time series plots for some locations in WA. There are 43 total regions in the dataset, and each would require its own dataset





02
Model Analysis

Autoregressive, SARIMA, Long Short Term Memory, Gated Recurrent Unit

Autoregression Model

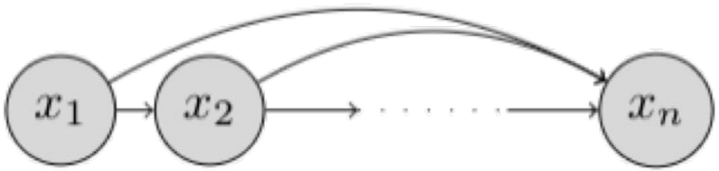
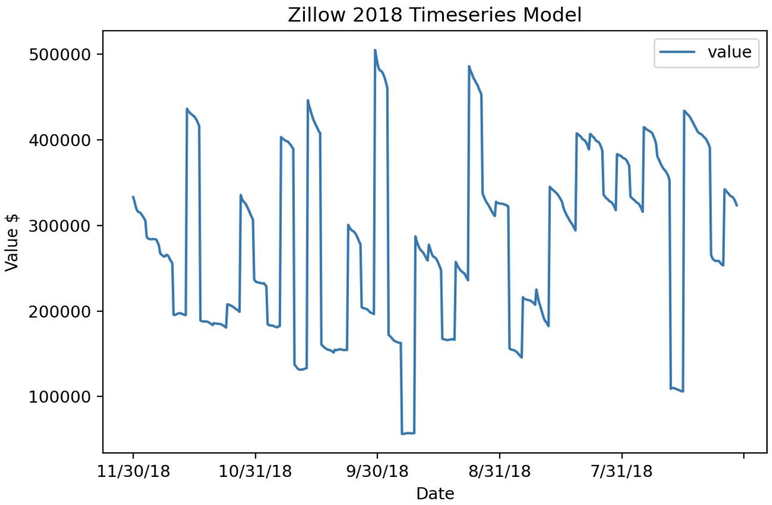


Fig. Graphical model for an autoregressive Bayesian network with no conditional independence assumptions. Stanford CS Depart.

Autoregression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step - resulting in accurate forecasts on a range of time series problems.

Autoregression Model

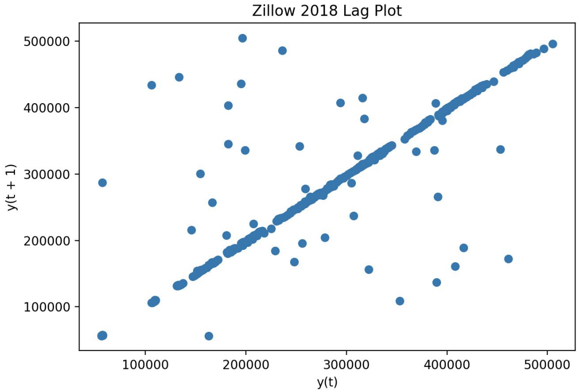


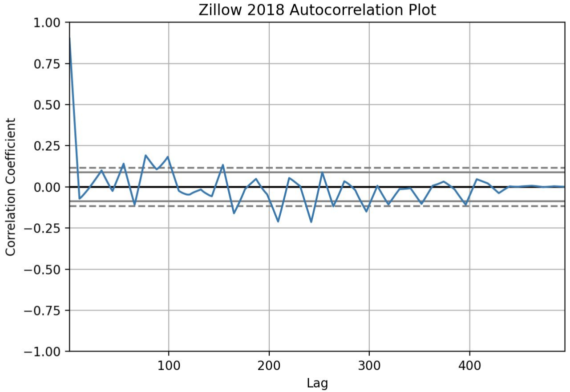
Autoregression

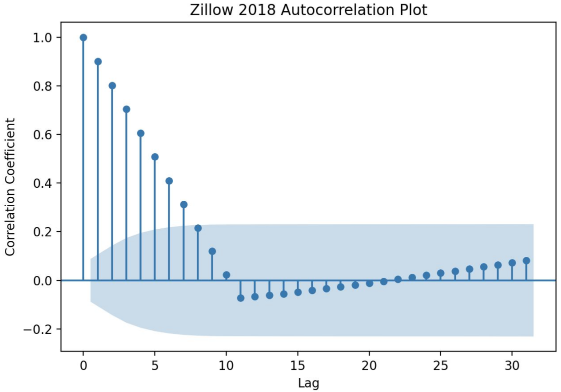
X(t+1) = b0 + b1*X(t-1) + b2*X(t-2)

- Using **lag variables**, input variables are taken as observations at previous time steps, the linear regression technique can be used on time series
- A regression model predicts the value for the next time step (t+1) given the observations at the last two time steps (t-1 and t-2)

Autoregression Model



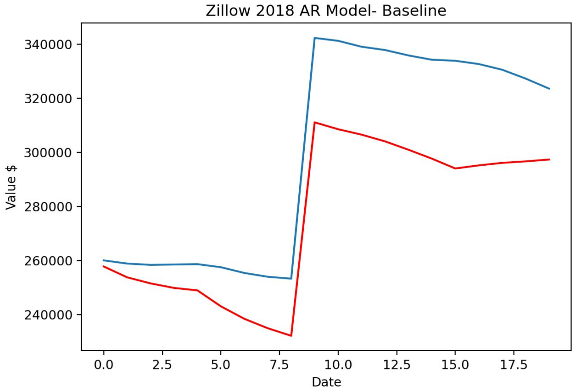


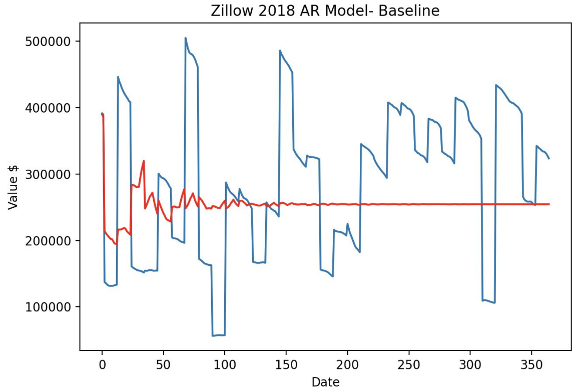


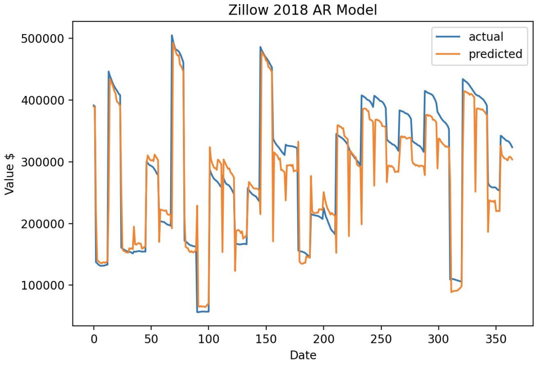
	t-1	t+1
t-1	1.000000	0.901274
t+1	0.901274	1.000000

- Linear pattern displays that the the data are strongly non-random and an autoregressive model might be appropriate.
- Strong positive correlation (0.901) between the observation and the lag=1 value

Autoregression Model





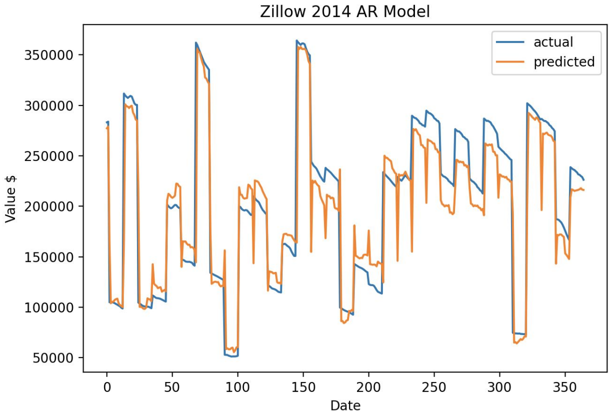


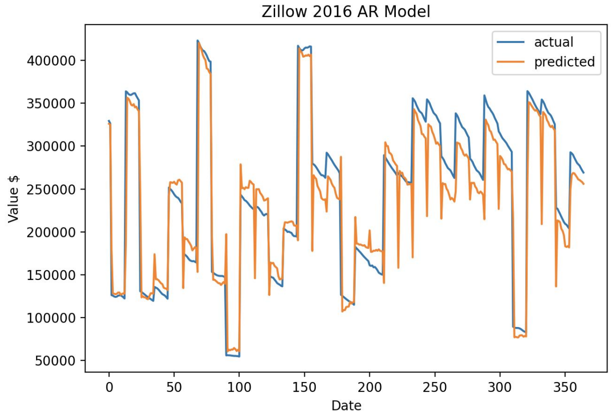
- AR Model Baseline - 20 days
  - RMSE: 26,568.42
  - Rsq: 0.527
- AR Model Baseline - 365 days
  - RMSE: 116,649.77
  - Rsq: -0.114
- AR Model Tuned - 365 days
  - RMSE: 45,571.05
  - Rsq: 0.830

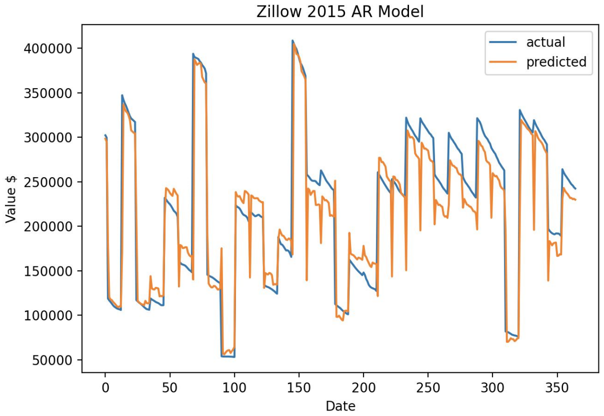
- Tuning: The coefficients are provided in an array with the intercept term followed by the coefficients for each lag variable starting at t-1 to t-n. We simply need to use them in the right order on the history of observations, as follows:

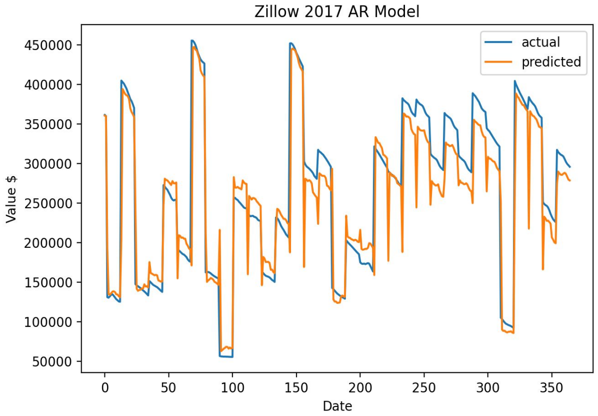
  yhat = b0 + b1*X1 + b2*X2 … bn*Xn

Autoregression Model









Seasonal Autoregressive Integrated Moving-Average

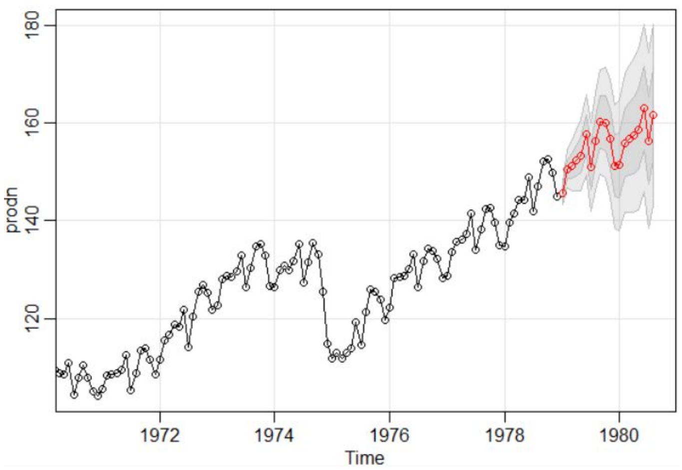
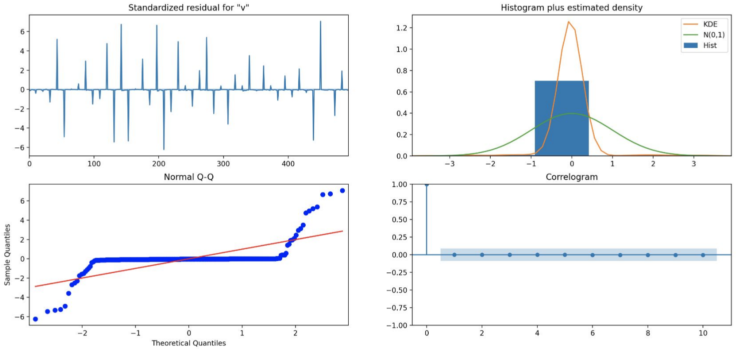


Fig. Visualization example of a SARIMA model. The grey bands depict prediction bounds +-standard error. Stanford CS Depart.

- The SARIMA forecasting is models univariate time series data that may contain trend and seasonal components.
- An effective approach for time series forecasting but it requires careful analysis to configure the seven or more model hyperparameters

Seasonal Autoregressive Integrated Moving-Average




```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  value   No. Observations:                  495
Model:               SARIMAX(1, 1, 1)   Log Likelihood               -6007.240
Date:                Tue, 27 Apr 2021   AIC                          12020.480
Time:                        11:29:38   BIC                          12033.088
Sample:                             0   HQIC                         12025.430
                                - 495
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1844    114.784      0.002      0.999    -224.788     225.157
ma.L1         -0.1886    114.537     -0.002      0.999    -224.677     224.300
sigma2      2.155e+09      0.000   1.87e+13      0.000    2.15e+09    2.15e+09
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             14827.12
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.61   Skew:                             1.43
Prob(H) (two-sided):                  0.00   Kurtosis:                        29.69
===================================================================================
```

Configuring a SARIMA requires selecting hyperparameters for both the trend and seasonal elements of the series

- Three trend elements:
  - p: Trend autoregression order
  - d: Trend difference order
  - q: Trend moving average order

- Four seasonal elements:
  - P: Seasonal autoregressive order
  - D: Seasonal difference order
  - Q: Seasonal moving average order
  - m: Number of time steps for a single seasonal period

Seasonal Autoregressive Integrated Moving-Average

```
## SARIMA Model - Grid Search
# one-step sarima forecast
def sarima_forecast(history, config):
	order, sorder, trend = config
	# define model
	model = SARIMAX(history, order=order, seasonal_order=sorder, trend=trend, en
	# fit model
	model_fit = model.fit(disp=False)
	# make one step forecast
	yhat = model_fit.predict(len(history), len(history))
	return yhat[0]

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:-n_test], data[-n_test:]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
	predictions = list()
	# split dataset
	train, test = train_test_split(data, n_test)
	# seed history with training dataset
	history = [x for x in train]
	# step over each time-step in the test set
	for i in range(len(test)):
		# fit model and make forecast for history
		yhat = sarima_forecast(history, cfg)
		# store forecast in list of predictions
		predictions.append(yhat)
		# add actual observation to history for the next loop
		history.append(test[i])
	# estimate prediction error
	error = measure_rmse(test, predictions)
	return error

# score a model, return None on failure
def score_model(data, n_test, cfg, debug=False):
	result = None
	# convert config to a key
	key = str(cfg)
	# show all warnings and fail on exception if debugging
	if debug:
		result = walk_forward_validation(data, n_test, cfg)
	else:
		# one failure during model validation suggests an unstable config
		try:
			# never show warnings when grid searching, too noisy
			with catch_warnings():
				filterwarnings("ignore")
				result = walk_forward_validation(data, n_test, cfg)
		except:
			error = None
	# check for an interesting result
	if result is not None:
		print(' > Model[%s] %.3f' % (key, result))
	return (key, result)
```

```
# grid search configs
def grid_search(data, cfg_list, n_test, parallel=True):
	scores = None
	if parallel:
		# execute configs in parallel
		executor = Parallel(n_jobs=cpu_count(), backend='multiprocessing')
		tasks = (delayed(score_model)(data, n_test, cfg) for cfg in cfg_list)
		scores = executor(tasks)
	else:
		scores = [score_model(data, n_test, cfg) for cfg in cfg_list]
	# remove empty results
	scores = [r for r in scores if r[1] != None]
	# sort configs by error, asc
	scores.sort(key=lambda tup: tup[1])
	return scores

# create a set of sarima configs to try
def sarima_configs(seasonal=[0]):
	models = list()
	# define config lists
	p_params = [0, 1, 2]
	d_params = [0, 1]
	q_params = [0, 1, 2]
	t_params = ['n','c','t','ct']
	P_params = [0, 1, 2]
	D_params = [0, 1]
	Q_params = [0, 1, 2]
	m_params = seasonal
	# create config instances
	for p in p_params:
		for d in d_params:
			for q in q_params:
				for t in t_params:
					for P in P_params:
						for D in D_params:
							for Q in Q_params:
								for m in m_params:
									cfg = [(p,d,q), (P,D,Q,m), t]
									models.append(cfg)
	return models
data = list(df_zillow.values.flatten())
# data split
n_test = 4
# model configs
cfg_list = sarima_configs()
# grid search
scores = grid_search(data, cfg_list, n_test)
print('done')
# list top 3 configs
for cfg, error in scores[:3]:
	print(cfg, error)
```

Gated Recurrent Unit Model

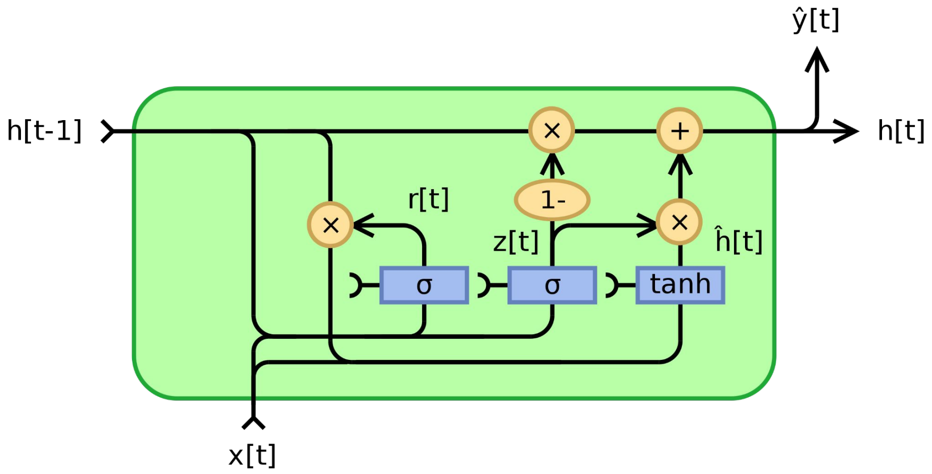


Fig. Gated Recurrent Unit Source: WikiMedia Foundation

GRU RNN's are similar to LSTM's, but they have no forget gate. They can perform well with irregular patterns, and support the use of exogenous variables.

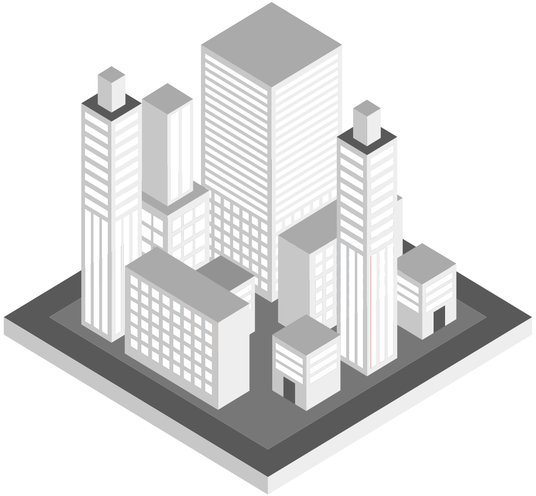
Shaping Data for GRU

GRU’s need 3D inputs based on values at previous periods. This requires us to reshape and rescale data. Imagine a 1D Time Series of shape (m, 1) [t0, t1, t2, … tk]

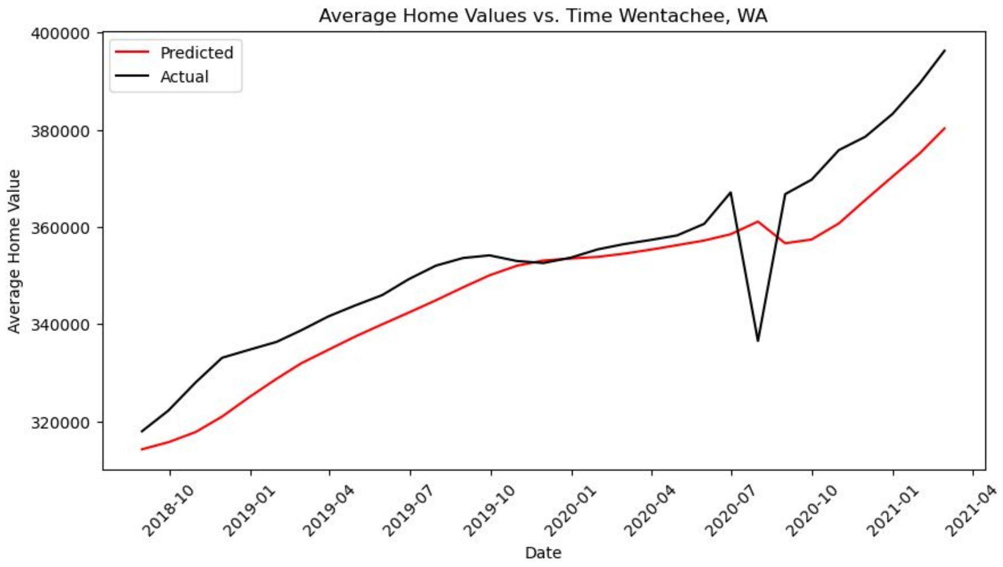
- The value at a given t is assumed to be a function of values before it, how far we want to go back is known as the lookback, for a lookback of 5 we have a shape of (m-lb, 5, 1)

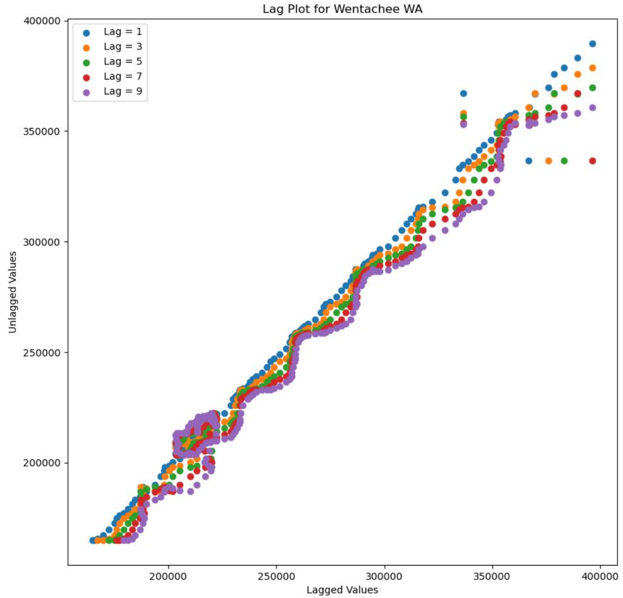
```
X                          y
[ [t0, t1, t2, t3, t4] [[t5]
  [t1, t2, t3, t4, t5]  [t6]
  [t2, t3, t4, t5, t6]  [t7]
       …
  [tk-5, …       tk-1]] [tk]]
```

- This can be expanded to more than 1 feature, and is how we’ll use exogenous variables in our GRU.



GRU With Single Value Prediction





For purposes of the presentation we used a single county in Wenatchee, WA to demonstrate the power of GRU. **R-squared** = 0.69 **RMSE** = 9449.20

Exogenous Variables

From the HMDA dataset we have some variables tracking yearly population growth and some other factors.

	loan_amount_000s	population	minority_population	hud_median_family_income
2014	276.019697	5325.572092	24.663210	75150.266877
2015	296.667876	5344.553204	24.600485	76803.155761
2016	315.794318	5332.407140	24.617906	76418.259577
2017	341.408179	5579.166888	26.248856	79440.841106

Trying to use these variables in our models was extremely difficult as we only had a small subset of the timeframe and on a much larger scale. The RNN models saw no change whatsoever because from month to month there was no change in these variables.

Long-Short Term Memory

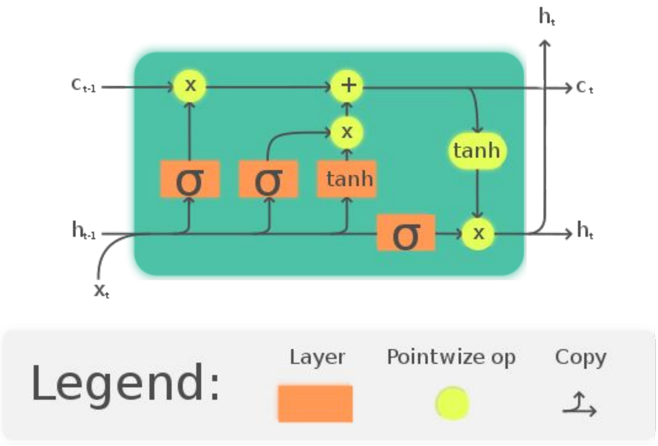
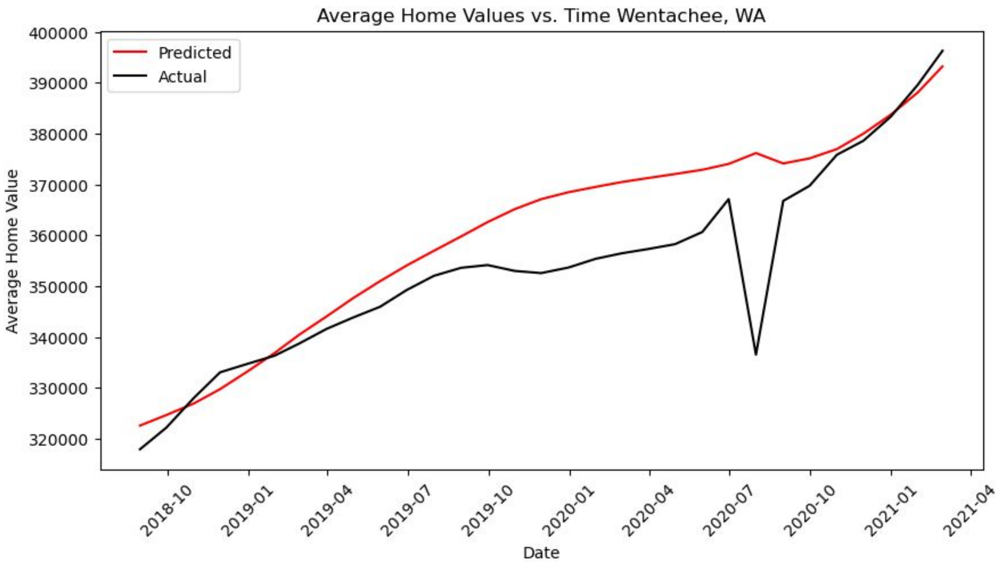
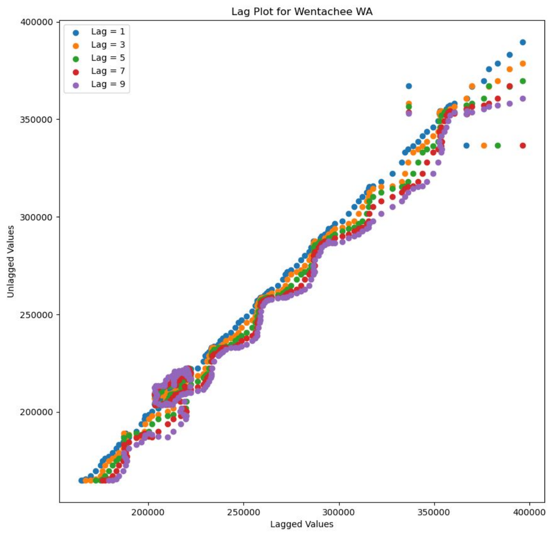


Fig. LSTM Source: WikiMedia Foundation

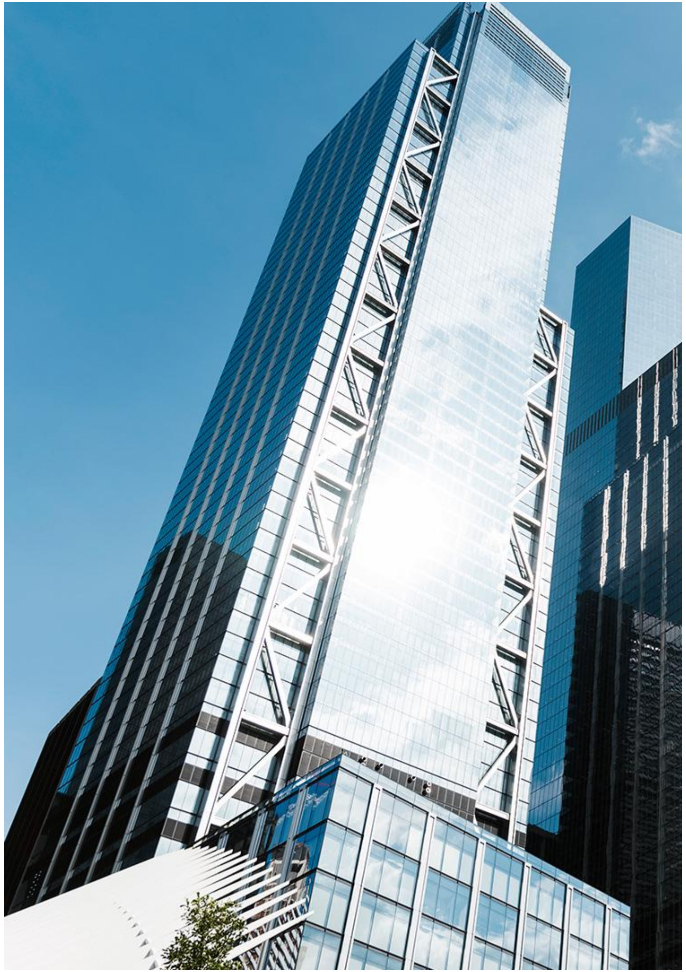
LSTM uses the same shaped data as GRU would and retains a similar control flow as an RNN. LSTM processes the data as it propagates forward, continuously building like an avalanche. The main differences are found in the operations within the cells of this model. These operations are used to allow the LSTM to keep or forget information as an instruction.

LSTM With Single Value Prediction





For purposes of the presentation we used a single county in Wenatchee, WA to demonstrate the power of LSTM. R-squared = 0.70 RMSE = 10582.98



03
Conclusions

Model Performance Comparison

Overall Model Performance

Model	Rsq	RMSE
Auto-Regressive	0.83	45,571.05
Gated Recurrent Unit RNN	0.69	9,449.19
Long Short-Term Memory	0.70	10,582.98

Thanks!

Do you have any questions?