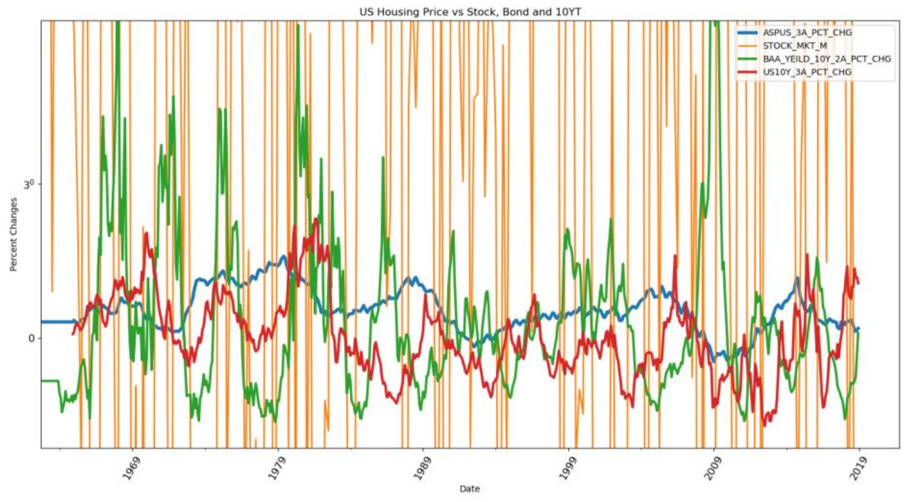
Predicting Housing Market – Milestone Report

Table of Contents

Create Data sets and Data Exploration (Part 1)

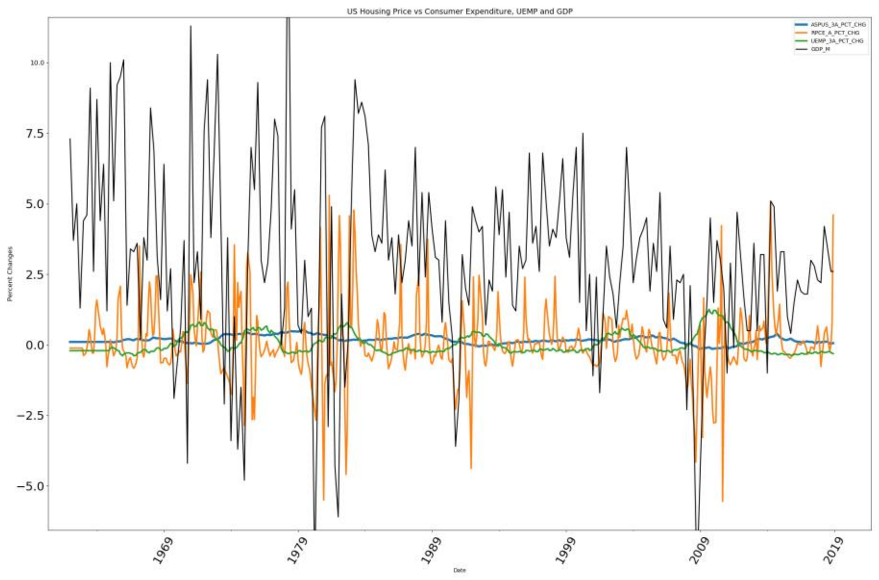
- During Housing market crash, average housing price drops 10% or more in 12 months
- This graph clearly indicate that during housing market crash (1981, 1992 & 2007) Personal Consumer Expenditures drops rapidly
- We can see positive growth in bond market during housing market crash (1981 & 2007)
- We can see that strong unemployment growth 1981 & 2007 or US Unemployment Growth is over 10% during housing market crash
- We can see that GDP growth is negative during 1981 & 2007 or GDP Growth is below -5% during housing market crash
- Higher interest rate is one of the major factor for housing market crash
- Normally stock market goes down sharply along with housing market
- We can see that housing supply ratio jump quickly right before the housing market crash. In another words, rapid housing supply growth eventually brings the housing market down.
- During the Housing market (1981, 1992 & 2007), Number of new 1F house sold in US sharply goes down.
- During the Housing market crash (1981, 1992 & 2007), Number of US Construction PERMIT in US sharply goes down.

Compare US average housing price with stock market, bonds and interest rates



Before the housing market crash, generally interest rates are higher and bond markets are lower. During Market Crash (1981, 1992 & 2007), stock market goes down, bonds goes up and interest rates goes down. Recent Data (2018) shows strong housing market, because strong stock market, low interest rate and healthy bond market.

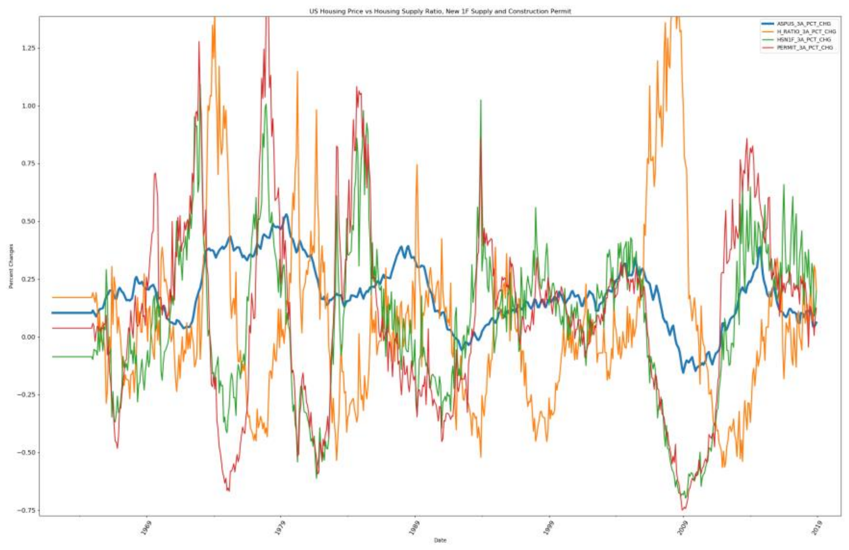
Compare US average housing price with Consumer expenditures, Unemployment market and US GDP Growth



During Market Crash (1981, 1992 & 2007), stock US GDP Growth drop sharply, US Unemployment Rate jump quickly and Personal Consumer Expenditures goes down.

If we look at the recent data, GDP is positive, Unemployment rate is very low, Personal Consumer expenditures is positive. Based on these data housing market still remain positive.

Compare US average housing price with Housing Supply Ratio, New 1F house sold in US and US Construction Permit



During Market Crash (1981, 1992 & 2007), Housing Supply Ratio jump sharply, Both number of new 1F houses sold & number of construction permits drop rapidly.

If we closely look at the graph for 2018 data, which shows Housing Supply Ratio is increasing slowly. Both number of new 1F houses sold & number of construction permits dropping slowly, which indicates downtrend market.

Machine Learning (Part 2)

Importing Housing Data

housing_df=pd.read_csv('C:/scripts/capstone2/housing_df2.csv', index_col=0)

ASPUS_3A_PCT_CHG US Average housing price movement (3 years % change)
H_RATIO_3A_PCT_CHG Housing Ratio: Number of house available vs. sold. (3 years % change)
HSN1F_3A_PCT_CHG New 1F House sold (3 years % change)
PERMIT_3A_PCT_CHG Current construction permits across USA (3 years % change)
STOCK_MKT_3A_PCT_CHG Stock market movement (3 years % change)
BAA_YEILD_10Y_2A_PCT_CHG Bond Market movement (2 years % change)
US10Y_3A_PCT_CHG US 10 Years Treasury Rate (3 years % change)
RPCE_A_PCT_CHG Personal consumer expenditures (1 years % change)
UEMP_3A_PCT_CHG Long Term Unemployment Rate (3 years % change)
RGDP_M_PCT_CHG Current US GDP (3 years % change)

Linear Regression

The R^2 in scikit learn is the coefficient of determination. It is 1 - residual sum of square / total sum of squares.

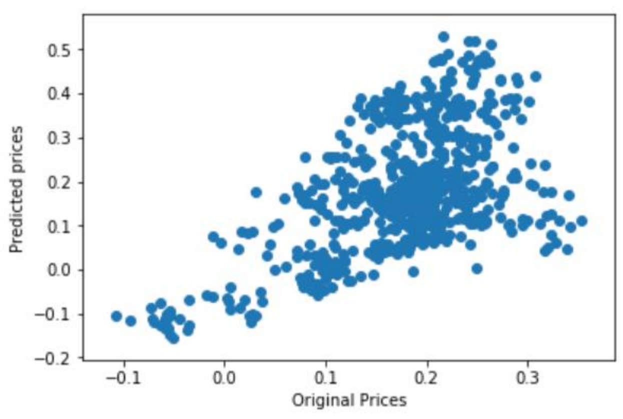
RMSE of the test data is closer to the training RMSE (and lower) if you have a well trained model. It will be higher if we have an overfitted model.

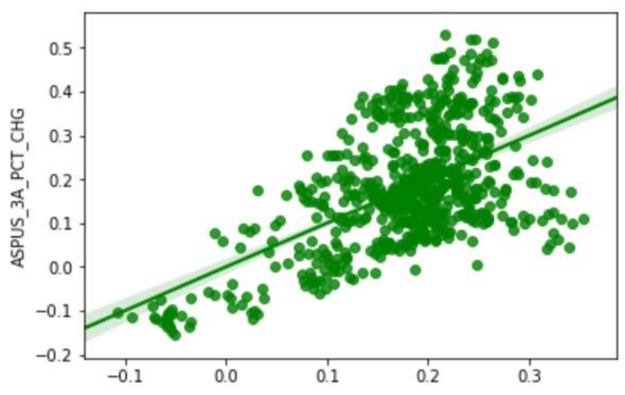
K fold cross validation

with Linear regressor we are able to predict the model with .114 RMSE and r squared 0.33 and cross validation root mean squarred error is : 0.1507

Fitting Linear Regression using statsmodels¶

Statsmodels is a great Python library for a lot of basic and inferential statistics. It also provides basic regression functions using an R-like syntax, so it's commonly used by statisticians. While we don't cover statsmodels officially in the Data Science Intensive workshop, it's a good library to have in your toolbox. Here's a quick example of what you could do with it. The version of least-squares we will use in statsmodels is called ordinary least-squares (OLS). There are many other versions of least-squares such as partial least squares (PLS) and weighted least squares (WLS).





Correlation Matrix with Heatmap

Correlation states how the features are related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable)

Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.
Number of new 1F house sold is positively (0.87) correllated with construction permits
Number of new 1F house sold is negatively (-0.56) correllated with Housing supply ratio
US Ave Houe price is negatively (-0.35) correllated with US Unemployment rate
Number of new 1F house sold is positively (0.87) correllated with construction permit

Monthly housing DATA ERROR TESTING

Each time series in the h_m_df DataFrame have very different seasonality patterns!

Correlations between multiple time series Earlier, we have extracted the seasonal component of each time series in the h_m_df DataFrame and stored those results in new DataFrame called seasonality_df. In the context of h_m_df data, it can be interesting to compare seasonality behavior, as this may help uncover which h_m_df indicators are the most similar or the most different.

This can be achieved by using the seasonality_df DataFrame and computing the correlation between each time series in the dataset. Here, we will compute and create a clustermap visualization of the correlations between time series in the seasonality_df DataFrame.

The train root mean squarred error is : 70267.9274598809

The test root mean squarred error is : 77818.89478743549

The Linear Regression coefficient parameters are : [1788.79238781 216.56884311 -75.96446774 1773.71494551 56562.90186394 -16389.03294286 -9061.49679158 -5763.22028744 2896.18391227]

The Linear Regression intercept value is : 147549.50571956355

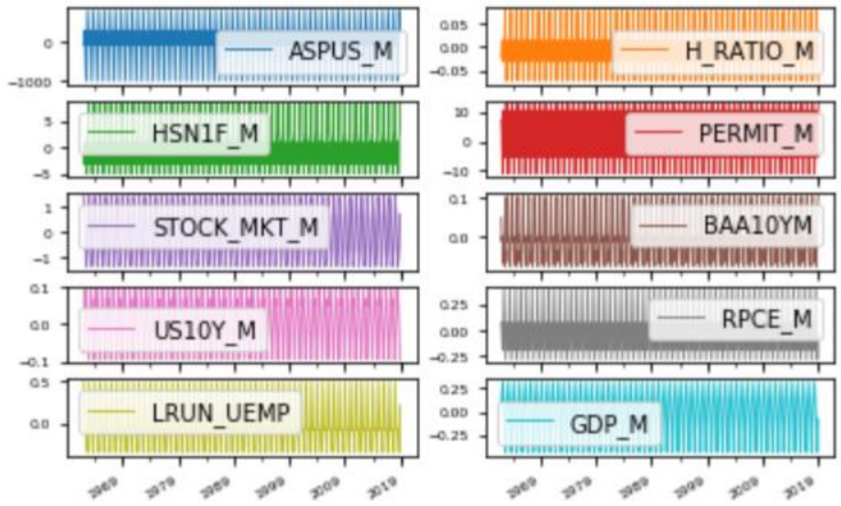
Visualize the seasonality of multiple time series

We will now extract the seasonality component of h_m_df_decomp to visualize the seasonality in these time series. Note that before plotting, you will have to convert the dictionary of seasonality components into a DataFrame using the pd.DataFrame.from_dict() function.

An empty dictionary h_m_df_seasonal and the time series decompisiton object h_m_df_decomp created.

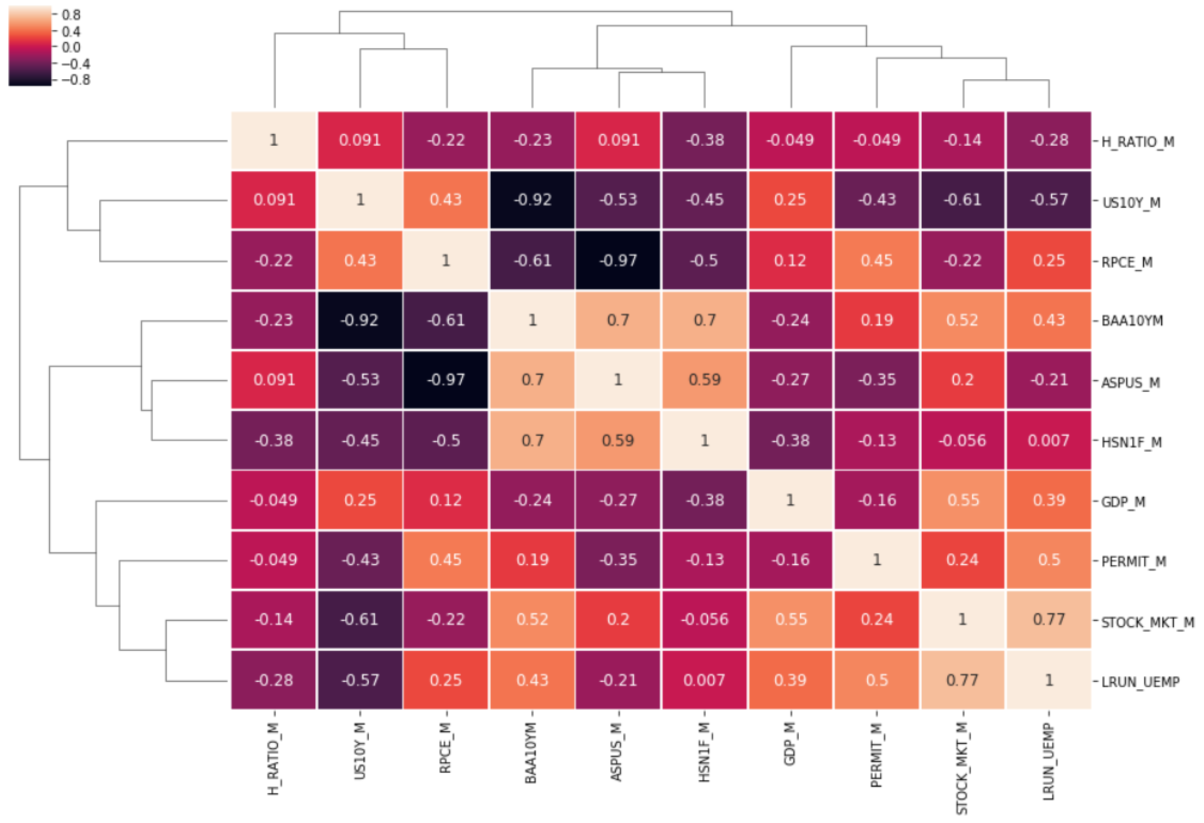
Iterate through each column name in jobs_names and extract the corresponding seasonal component from h_m_df_decomp. Place the results in the jobs_seasonal, where the column name is the name of the time series, and the value is the seasonal component of the time series. Convert h_m_df_seasonal to a DataFrame and call it seasonality_df. Create a facetted plot of all 10 columns in seasonality_df. Ensure that the subgraphs do not share y-axis.

The seasonal component can be extracted using the .seasonal attribute. Use the pd.DataFrame.from_dict() to convert a dictionary to a DataFrame. Faceted plots of DataFrame df can be generated by setting the subplots argument to True



Compute the correlation between all columns in the seasonality_df DataFrame using the spearman method and assign the results to seasonality_corr. Create a new clustermap of your correlation matrix.

Use the .corr() method along with the method argument to create a correlation matrix. To plot a clustermap, use the sns.clustermap() function.



Visualizing predicted values¶

When dealing with time series data, it's useful to visualize model predictions on top of the "actual" values that are used to test the model.

In this exercise, after splitting the data (stored in the variables X and y) into training and test sets, you'll build a model and then visualize the model's predictions on top of the testing data in order to estimate the model's performance.

Split the data (X and y) into training and test sets. Use the training data to train the regression model. Then use the testing data to generate predictions for the model.

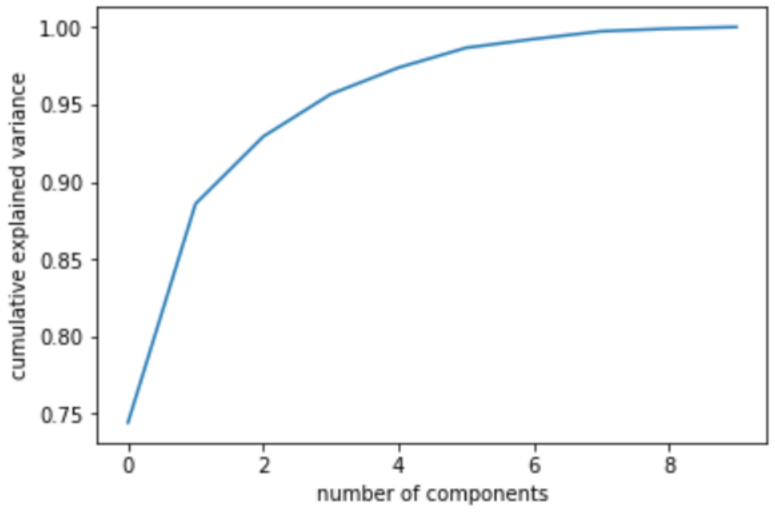
You should be splitting up the arrays X and y into training and test sets with train_test_split().

Coefficient of Determination (R^2) The value of R is bounded on the top by 1, and can be infinitely low Values closer to 1 mean the model does a better housing price of predicting outputs

Variance of the PCA features

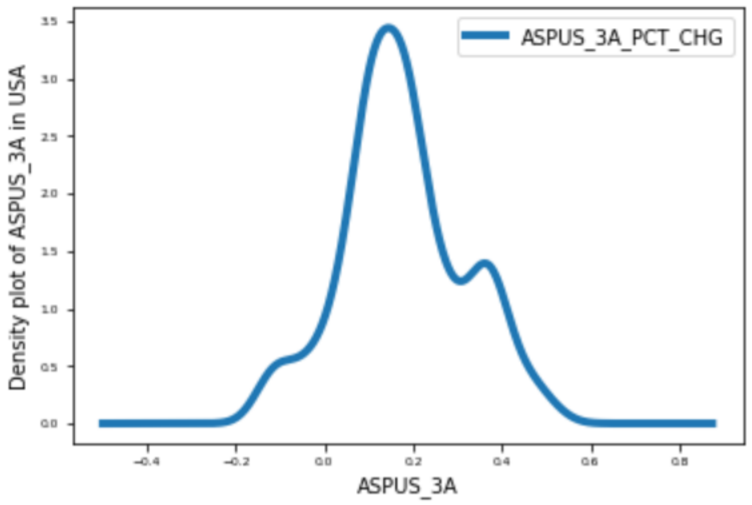
The dataset is 10-dimensional. But what is its intrinsic dimension? Make a plot of the variances of the PCA features to find out. As before, samples is a 2D array, where each row represents a fish. You'll need to standardize the features first.

We want to know how many principal components we can choose for our new feature subspace? A useful measure is the so-called “explained variance ratio“. The explained variance ratio tells us how much information (variance) can be attributed to each of the principal components. We can plot bar graph between no. of features on X axis and variance ratio on Y axis



Density plots¶

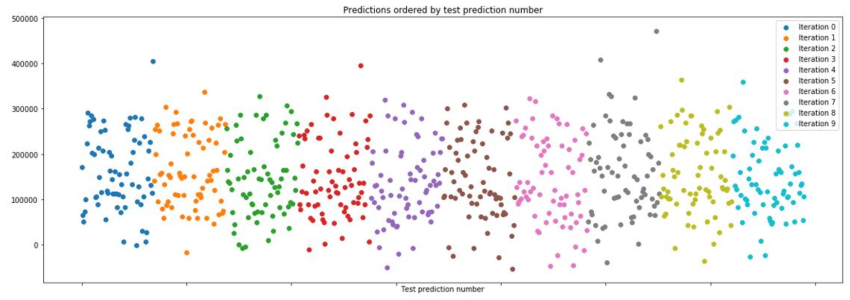
In practice, histograms can be a substandard method for assessing the distribution of data because they can be strongly affected by the number of bins that have been specified. Instead, kernel density plots represent a more effective way to view the distribution of your data. An example of how to generate a density plot of is shown below:

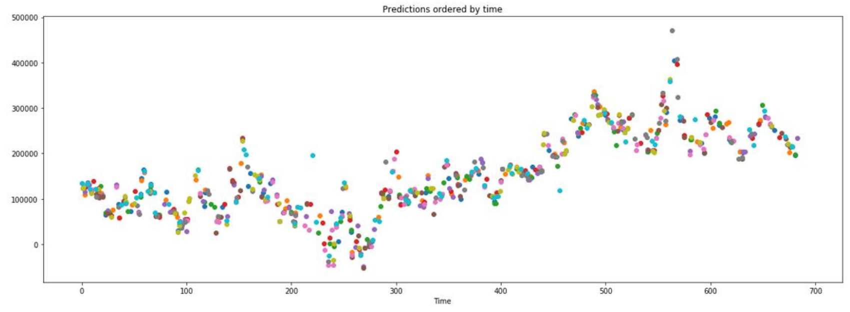


Cross-validation with shuffling¶

As you'll recall, cross-validation is the process of splitting your data into training and test sets multiple times. Each time you do this, you choose a different training and test set. In this exercise, you'll perform a traditional

ShuffleSplit cross-validation on the company value data from earlier. Later we'll cover what changes need to be made for time series data. The data we'll use is the same historical price data for several large companies.



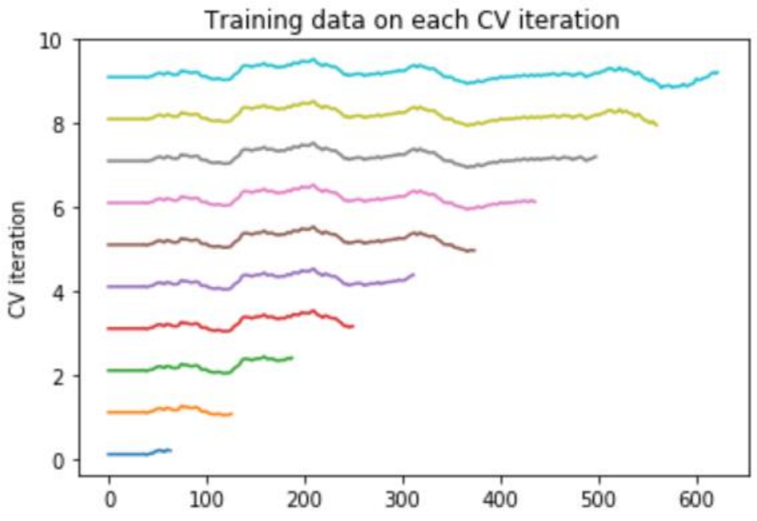


Time-based cross-validation¶

Finally, let's visualize the behavior of the time series cross-validation iterator in scikit-learn. Use this object to iterate through your data one last time, visualizing the training data used to fit the model on each iteration.

An instance of the Linear regression model object is available in your workpsace. Also, the arrays X and y (training data) are available too.

Import TimeSeriesSplit from sklearn.model_selection. Instantiate a time series cross-validation iterator with 10 splits. Iterate through CV splits. On each iteration, visualize the values of the input data that would be used to train the model for that iteration.



Note that the size of the training set grew each time when you used the time series cross-validation object. This way, the time points you predict are always after the timepoints we train on.

Time Series (Part 3)

Autocorrelation plots can be used to quickly discover patterns into your time series, so let's delve a little bit deeper into that!

Partial autocorrelation in time series data Like autocorrelation, the partial autocorrelation function (PACF) measures the correlation coefficient between a time-series and lagged versions of itself. However, it extends upon this idea by also removing the effect of previous time points. For example, a partial autocorrelation function of order 3 returns the correlation between our time series (t_1, t_2, t_3, ...) and its own values lagged by 3 time points (t_4, t_5, t_6, ...), but only after removing all effects attributable to lags 1 and 2.

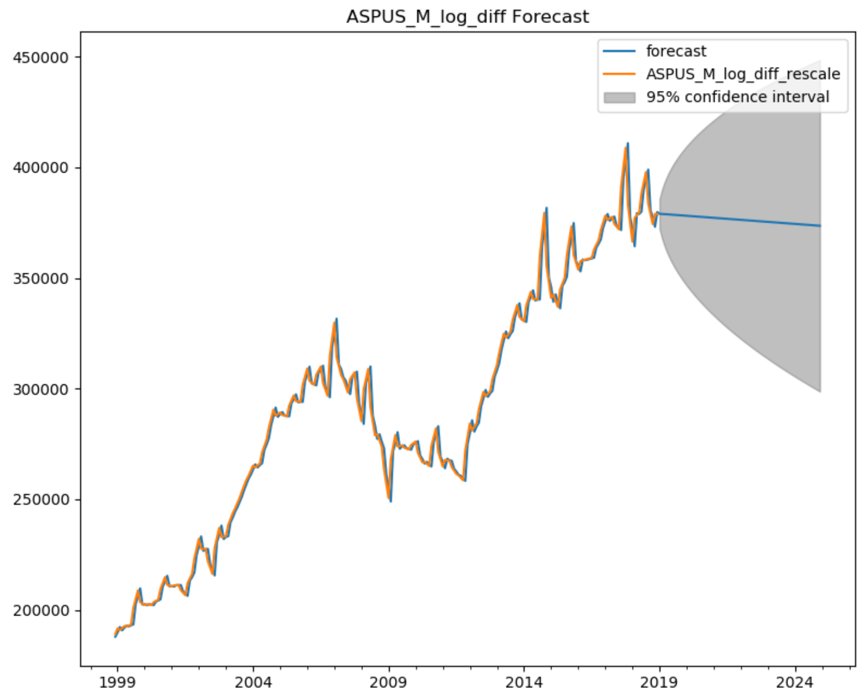
Time series decomposition

When visualizing time series data, we should look out for some distinguishable patterns:

seasonality: does the data display a clear periodic pattern?
trend: does the data follow a consistent upwards or downward slope?
noise: are there any outlier points or missing values that are not consistent with the rest of the data?

FORECASTING ARIMA

Applying statistical modeling and machine learning to perform time-series forecasting.



Forecasting Housing Market using FBProphet (Part 4)

Let us model some time-series data! using Facebook Prophet package.

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

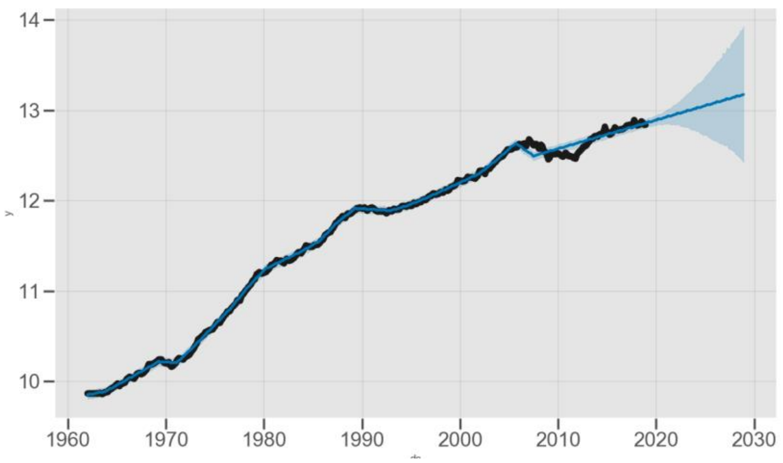
Accurate and fast.

Prophet is used in many applications across Facebook for producing reliable forecasts for planning and goal setting. We’ve found it to perform better than any other approach in the majority of cases. We fit models in Stan so that you get forecasts in just a few seconds. Get a reasonable forecast on messy data with no manual effort. Prophet is robust to outliers, missing data, and dramatic changes in your time series.

Tunable forecasts.

The Prophet procedure includes many possibilities for users to tweak and adjust forecasts. You can use human-interpretable parameters to improve your forecast by adding your domain knowledge.

Forecast using the ‘predict’ command:



'''That looks pretty good. Now, let’s take a look at the seasonality and trend components of our /data/model/forecast.'''

If you want to see the forecast components, you can use the Prophet.plot_components method. By default you’ll see the trend, yearly seasonality, and weekly seasonality of the time series. If you include holidays, you’ll see those here, too.

Import monthly housing price data

Let's rename the columns as required by fbprophet. Additioinally, fbprophet doesn't like the index to be a datetime...it wants to see 'ds' as a non-index column, so we won't set an index differnetly than the integer index.

df2.set_index('ds').y.plot()

Adjusting trend flexibility

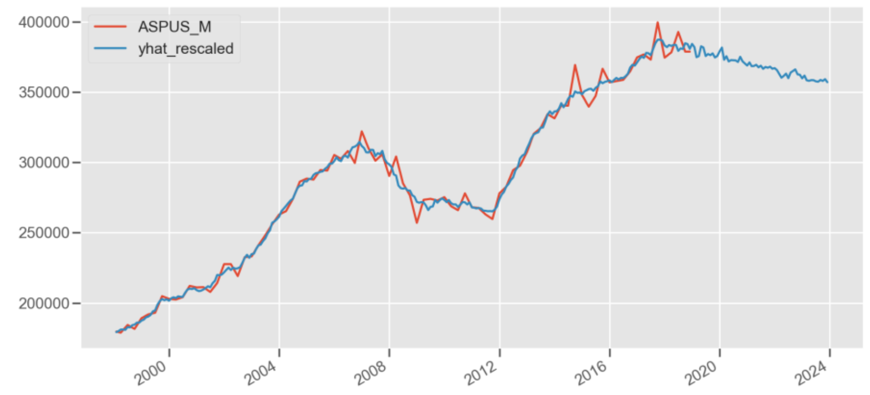
If the trend changes are being overfit (too much flexibility) or underfit (not enough flexibility), you can adjust the strength of the sparse prior using the input argument changepoint_prior_scale. By default, this parameter is set to 0.05. Increasing it will make the trend more flexible:'''
#95-96 .9-.95 -60-80

Plotting Prophet results
Prophet has a plotting mechanism called plot. This plot functionality draws the original data (black dots), the model (blue line) and the error of the forecast (shaded blue area).
model.plot(forecast_data);

Visualizing Prophet models

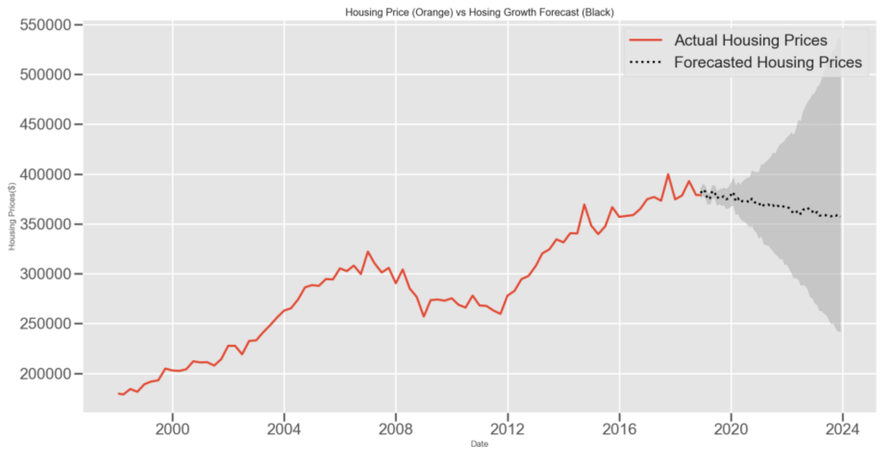
In order to build a useful dataframe to visualize our model versus our original data, we need to combine the output of the Prophet model with our original data set, then we'll build a new chart manually using pandas and matplotlib. First, let's set our dataframes to have the same index of ds

Let's take a look at the sales and yhat_rescaled data together in a chart.'
viz_df[['ASPUS_M', 'yhat_rescaled']].plot()



We can see downward trends in 60 months for AVE US HOUSING PRICE

Time to plot: let's plot everything to get the 'final' visualization of our monthly housing data and forecast with errors



This visualization is much better (in my opinion) than the default fbprophet plot. It is much easier to quickly understand and describe what's happening. The orange line is actual sales data and the black dotted line is the forecast. The gray shaded area is the uncertaintity estimation of the forecast.

For next Five Years, we can see Ave Housing Price downward trend. Price range will remain between $248000 and $550000

Forecasting US housing market using Deep Learning (Part 5)

Time series forecasting is challenging, especially when working with long sequences, noisy data, multi-step forecasts and multiple input and output variables.

Deep learning methods offer a lot of promise for time series forecasting, such as the automatic learning of temporal dependence and the automatic handling of temporal structures like trends and seasonality.

Time series data can be phrased as supervised learning.
Given a sequence of numbers for a time series dataset.

- We will discover how to develop a Long Short-Term Memory Neural Network model or LSTM for univariate time series forecasting.
- We can define a simple univariate problem as a sequence of integers, fit the model on this sequence and have the model predict the next value in the sequence.
- The LSTM model expects three-dimensional input with the shape [samples, timesteps, features]. We will define the data in the form [samples, timesteps] and reshape it accordingly.

- We will use one LSTM layer to process each input sub-sequence of 3 time steps, followed by a Dense layer to interpret the summary of the input sequence. The model uses the efficient Adam version of stochastic gradient descent and optimizes the mean squared error (‘mse‘ or 'rmse') loss function.

- Once the model is defined, it can be fit on the training data and the fit model can be used to make a prediction.

Import Monthly Housing Price Data

h_m_df = pd.read_csv('C:/scripts/capstone2/h_m_df.csv', index_col='DATE', parse_dates=True)

Specifying a model

Now you'll get to work with your first model in Keras, and will immediately be able to run more complex neural network models on larger datasets compared to the first two chapters.

To start, you'll take the skeleton of a neural network and add a hidden layer and an output layer. You'll then fit that model and see Keras do the optimization so your model continually gets better.

As a start, you'll predict workers wages based on characteristics like their industry, education and level of experience. You can find the dataset in a pandas dataframe called df. For convenience, everything in df except for the target has been converted to a NumPy matrix called predictors. The target, wage_per_hour, is available as a NumPy matrix called target.

Compiling the model

We're now going to compile the model you specified earlier. To compile the model, you need to specify the optimizer and loss function to use. The Adam optimizer is an excellent choice.
Here we'll use the Adam optimizer and the mean squared error loss function.

Compile the model using model.compile(). Your optimizer should be 'adam' and the loss should be 'mean_squared_error'.

Fitting the model

We'll now fit the model. Recall that the data to be used as predictive features is loaded in a NumPy matrix called predictors and the data to be predicted is stored in a NumPy matrix called target. Your model is pre-written and it has been compiled with the code from the previously compiled.

Fit the model. Remember that the first argument is the predictive features (predictors), and the data to be predicted (target) is the second argument.

Classification models

We'll now create a classification model using the **housing_df** dataset. The predictive variables are stored in a NumPy array predictors. The target to predict is in df.survived, though you'll have to manipulate it for keras. The number of predictive features is stored in n_cols.

Here, we'll use the 'sgd' optimizer, which stands for Stochastic Gradient Descent.
Convert df.survived to a categorical variable using the to_categorical() function.
Specify a Sequential model called model.
Add a Dense layer with 32 nodes. Use 'relu' as the activation and (n_cols,) as the input_shape.
Add the Dense output layer. Because there are two outcomes, it should have 2 units, and because it is a classification model, the activation should be 'softmax'.

Compile the model, using 'sgd' as the optimizer, 'categorical_crossentropy' as the loss function, and metrics=['accuracy'] to see the accuracy (what fraction of predictions were correct) at the end of each epoch.
Fit the model using the predictors and the target.

Making predictions¶

The trained network from your previous coding exercise is now stored as model. New data to make predictions is stored in a NumPy array as pred_data. Use model to make predictions on your new data.

In this exercise, your predictions will be probabilities, which is the most common way for data scientists to communicate their predictions to colleagues.

Create your predictions using the model's .predict() method on pred_data. Use NumPy indexing to find the column corresponding to predicted probabilities of survival being True. This is the second column (index 1) of predictions. Store the result in predicted_prob_true and print it.

Housing price Prediction_Using_CNN

h_m_df = pd.read_csv('C:/scripts/capstone2/h_m_df.csv', index_col='DATE', parse_dates=True)

Modeling

Define Network

Compile Network¶

`Before training, configure the learning process by specifying:`
`Optimizer to be 'adam';`
`Loss function to be 'mse';`
`Evaluation metric to be 'accuracy'.`

Train Network¶

`Fit the model to training data to learn the parameters.`

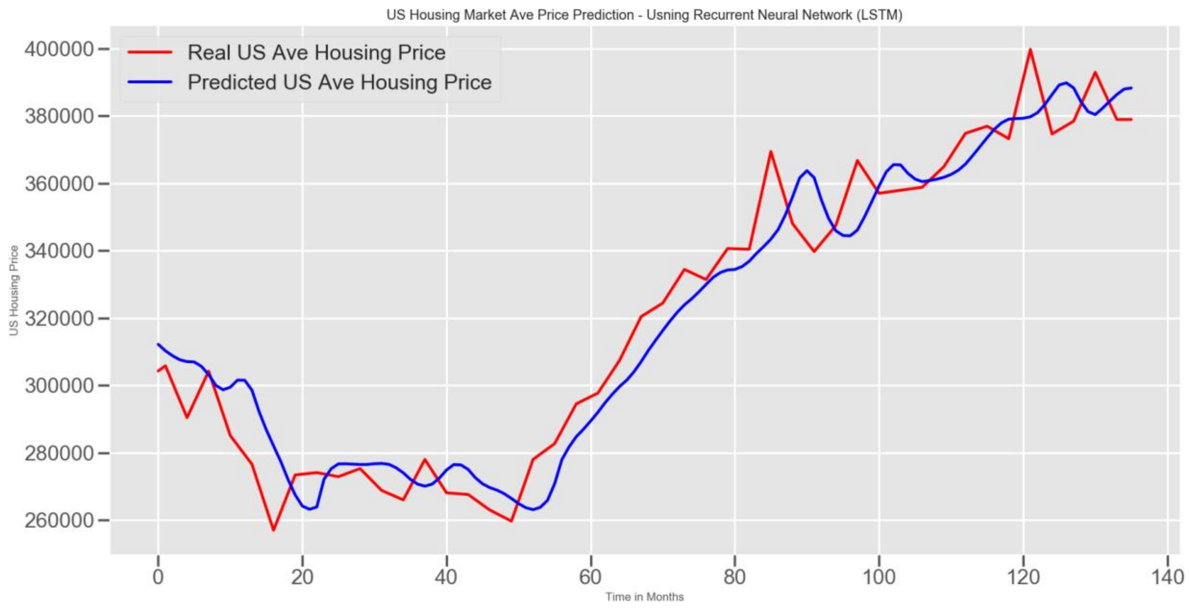
Evaluate Network

`Evaluate model on the test set`

LSTM-RNN to forecast time-series

RNN's (LSTM's) are pretty good at extracting patterns in input feature space, where the input data spans over long sequences. Given the gated architecture of LSTM's that has this ability to manipulate its memory state, they are ideal for such problems.

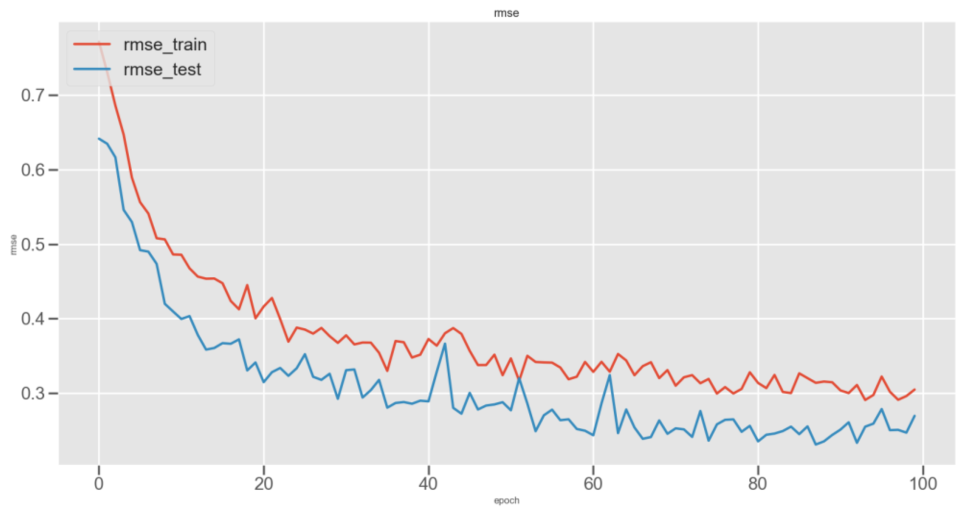
Visualising the results

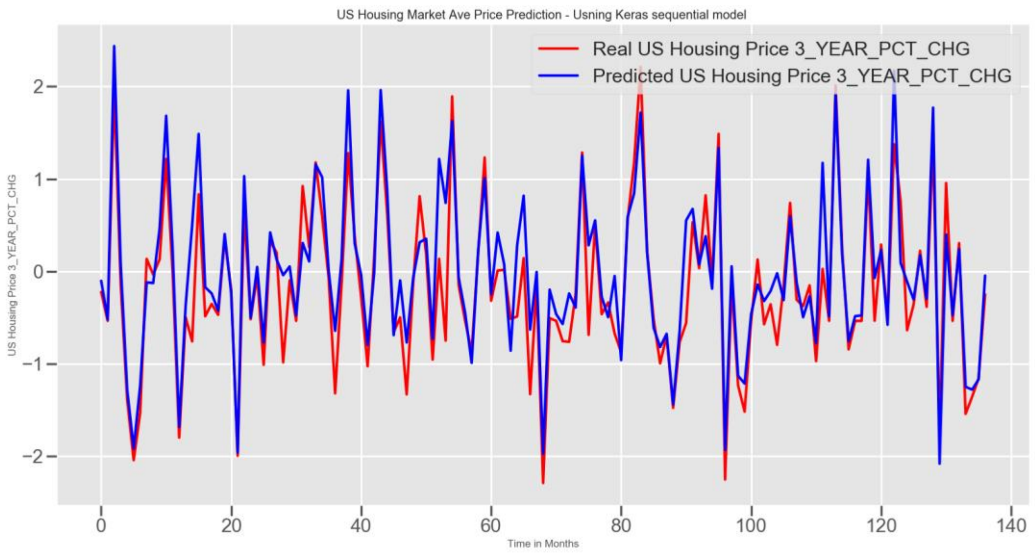


R^2 is a statistic that will give some information about the goodness of fit of a model. In regression, the R^2 coefficient of determination is a statistical measure of how well the regression predictions approximate the real data points. An R^2 of 1 indicates that the regression predictions perfectly fit the data.

Values of R^2 outside the range 0 to 1 can occur when the model fits the data worse than a horizontal hyperplane. This would occur when the wrong model was chosen, or nonsensical constraints were applied by mistake. If equation 1 of Kvålseth[11] is used (this is the equation used most often), R^2 can be less than zero.

Both Actual price and Predicted price is indicating that overheated housing market is slowing down. Pick Ave Housing Price was $400000 and now down to $378000. We can see 5.5% down from High Price.





US Housing Market Ave Price Prediction - Usning Keras sequential model
Real US Housing Price 3_YEAR_PCT_CHG
Predicted US Housing Price 3_YEAR_PCT_CHG
US Housing Price 3_YEAR_PCT_CHG
Time in Months
2
1
0
−1
−2
0
20
40
60
80
100
120
140