AI Project Report

Stig Thomas Gulbrandsen

CC-BY 2019/07/18

Contents

Figures

Tables

Chapter 1

Introduction

Out of the three project suggestions, I've chosen Project 1, Predicting Total Sales For Every Product And Store In The Next Month. Due to me being more comfortable with predicting using machine learning as opposed to recognition and image detection. This report details the timeline and decision making process of the project. let it be known that this project was done on my own, with no outside help from other groups or participants from the course.

Chapter 2

Background

2.1 Terminology

Machine Learning
Computer algorithms that improve/learn through experience (shorthand: ML)

Data frame
Data structure that contains labeled rows and columns, provided by the Pandas library. Used in the project for complex data manipulation on files containing data.

.csv file format
text file using commas to separate values, popularly used for storing tabular data sets.

data set
Collection of data, in the project the data sets are retrieved from .csv files and stored as pandas data frames for future usage

X and y
The independent and dependent data of a machine learning model. X is always spelled in capital, and refers to the data that helps predict y, and y refers to the data we wish to predict.

2.2 Data

These are the data contained in the files initially given for the project.

item_name	item_id	item_category_id

Table 2.1: Items

item_category_name	item_category_id

Table 2.2: item_categories

shop_name	shop_id

Table 2.3: Shop

ID	shop_id	item_id

Table 2.4: Test set

date	date_block_num	shop_id	item_id	item_price	item_cnt_day

Table 2.5: Train set

Chapter 3

Implementation

The implementation of the project can be split into three parts

- Initial processing of data into appropriate data frames
- Calculation of each shop/item/month combination's monthly item count
- The application of prediction model on the complete data set

In addition, on the bottom of the project file are tests for each ML model tested for the project.

3.1 Data processing

The project starts with dividing the given .csv files into respective data frames. The train and test set are both missing columns from each other, which would make training and testing the model difficult, as their current states would require computation-heavy workarounds to apply any sort of ML model on the data sets. Because of this they're initially merged into a combined data frame.

```
#dataframes

#the training set. Daily historical data from January 2013 to October 2015
df_train = pd.read_csv('sales_train.csv')
#You need to forecast the sales for these shops and products for November 2015.
df_test = pd.read_csv('test.csv')
#supplemental information about the items/products.
df_items = pd.read_csv('items.csv')
#supplemental information about the items categories.
df_categories = pd.read_csv('item_categories.csv')
#supplemental information about the shops.
df_shops = pd.read_csv('shops.csv')

#combine the train and test set + suplementaries into a single dataset
df_traintest = pd.merge(df_test, df_train)
df_itemcategories = pd.merge(df_items,df_categories)
df_traintestshops = pd.merge(df_traintest, df_shops)
#the combined dataset
df = pd.merge(df_traintestshops,df_itemcategories)
df.to_csv(r'df.csv', index = 'date', header=True)
```

Figure 3.1: Initialization of data frames

Similarly, the item categories, shops and the like are merged into combined data frames, and they're all finally combined with the train/test data frame into one, all-encompassing data frame containing all the data we'll ever need for the project, to be split into X/y train/test sets later in the program. For safety reasons it is then exported to a .csv file for future usage if that should ever be needed

ID	shop_id	item_id	date	date_block_num
item_price	item_cnt_day	shop_name	item_name	item_category_id
item_category_name				

Table 3.1: Combined data frame columns

3.2 Monthly item count calculation

This section of the program assigns each shop/item combination per month in the data frame based on the sum of item_cnt_day for the month in question. Only the useful data is retained during this process, as label columns (names) serve no purpose for predicting the desired results. Date was originally retained as well, however the difference in predictions with it removed were minuscule, so it was omitted as well.

Originally, it looped through the whole combined dataset, and the columns of the current row served as identifiers for a Pandas groupby object, which then had

ID	shop_id	item_id	date_block_num
item_price	item_cnt_day	item_category_id	monthly_item_cnt

Table 3.2: Current data frame columns after removal and insertion

it's data and a new monthly_item_cnt column appended onto a data frame. This became exponentially slower over time due to the data frame gaining size with each iteration. This approach was later changed to iterate directly on the group object instead, and the data was inserted into a dictionary, which led to less than half the compilation time and iterations of the original section. It is then saved as a .csv file for future use.

```
#**********Set the monthly item count**********************
#every unique combination of shops and items

df_predictOnThis = pd.DataFrame()
#the item/shop pair further divided into date blocks
grouped_df = df.groupby(['shop_id','item_id','date_block_num'])
dict = {}
#for every dateblock in our shop/item pairing, set the monthly item count
i = 0
for index, group in grouped_df:
    #iterate through every shop item combination
    if(i % 1000 == 0):
        print("iteration no. ", i)
    i += 1
    df_n=group
    df_n['monthly_item_cnt'] = df_n['item_cnt_day'].sum()
    monthitemcnt = df_n.iloc[-1]['monthly_item_cnt']
    #take group size into acount,
    dict[i] = {'ID': int(df_n['ID'].values[0]),
               'shop_id':int(df_n['shop_id'].values[0]),
               'item_id':int(df_n['item_id'].values[0]),
               'date_block_num':int(df_n['date_block_num'].values[0]),
               'item_price':int(df_n['item_price'].values[0]),
               'item_cnt_day':int(df_n['item_cnt_day'].values[0]),
               'item_category_id':int(df_n['item_category_id'].values[0]),
               'monthly_item_cnt':monthitemcnt}
print("Finished compilation")
df_predictOnThis = pd.DataFrame.from_dict(dict,'index')
```

Figure 3.2: monthly count calculation for data set

3.3 Application of ML model

The main section of the project, and where the ML model is applied onto the entire 1.2 million line data set starts by first reading the .csv file created during the monthly item count calculation. A data frame consisting of each shop/item

combination is created and thereafter iterated through in the main for loop of the program.

```
dict = {}
df_predict = pd.read_csv('df_predictonthis.csv')

grouped_df = df_predict.groupby(['shop_id','item_id'])
k = 0
print(len(grouped_df))
#Go through all the shop/item pairings
for index, group in grouped_df:
    if(k % 100 == 0):
        print("iteration no.",k)
    k += 1
    #Random Forest
    model = RandomForestRegressor()
   #the current shop item combination
    #df_n= grouped_df.get_group((row['shop_id'],row['item_id']))
    #the row we want to predict
    y = group['monthly_item_cnt']
    #our independent variables
    X = group.drop(['monthly_item_cnt'], axis = 1 )
```

Figure 3.3: initialization of main program loop

There were two iteration approaches tested for this part, one where the program iterates on the grouped_df's group elements, and another where the program iterates on df_predict's rows and thereafter assigns a second data frame df_n. One commented out section in this image shows remnants of this second approach. Here, df_n is set to the shop/item combination for the current row, and y would've been set to be df_n's monthly_item_cnt. The speed and end result remain the same for both of these approaches, however the group-wise iteration was chosen for sake of readability.

```
    y = group['monthly_item_cnt']
    #our independent variables
    X = group.drop(['monthly_item_cnt'], axis = 1 )

    numIter = int(X.iloc[-1]['date_block_num'])
    ID = int(group.iloc[-1]['ID'])
```

Figure 3.4: X and y declaration

X is made up of every row in the data frame excluding y's column. Although we only really need the date_block_num for the monthly predictions, The value for the date block is stored as it is necessary to know the amount of future predictions necessary to reach November 2015, our desired monthly prediction.

ID	shop_id	item_id	date_block_num
item_price	item_cnt_day	item_category_id	

Table 3.3: X data

monthly_item_cnt

Table 3.4: y data

```
81      #Divide the data set into training data and testing data
82     #if there's enough samples to split, else just use X and y
83      size = len(group.index)
84      if  size > 1:
85          X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
86
87          X_train = preprocessing.scale(X_train)
88          #Fit the model using training data
89          model.fit(X_train, y_train)
```

Figure 3.5: Splitting data into training and test sets if possible

Sometimes, there's not enough samples per month to properly split to X/y train/test sets, so a precaution was put in place so the data is only split if there's enough samples for a proper split.

The X training data is first scaled, and later fitted onto the model with the y training data. Scoring isn't done in this section of the program as the data being worked on is far too large in scope, so this will instead be shown in a later section, where all the different model's performances were tested out.

```
91      #Fit the model again using the whole data set
92      X = preprocessing.scale(X)
93      model.fit(X,y)
94      #predict the unseen data
95      predict = model.predict(X)
96      #print("MSE: %.4f" % mse)
97      #print("unseen data:", predict)
98      #next months value casted to int, our desired number
99      #print("forecast for next month: ", int(predict))
```

Figure 3.6: Fitting the model with the whole X y data set

regardless of whether or not the data is split into train and test sets, X is later scaled, and the original X and y are fitted to the model. Fitting the model twice gives us the prediction we desire once we run model.predict(X)

```
    if numIter != 33:
        for j in range(34 - numIter):
            monthModel = RandomForestRegressor()
            group['monthly_item_cnt'] = predict
            month_y = group['monthly_item_cnt']
            month_X = X

            #Divide the data set into training data and testing data
            if  size > 1:
                X_train, X_test, y_train,
                y_test = train_test_split(month_X,month_y,test_size = 0.2)

                X_train = preprocessing.scale(X_train)
                #Fit the model using training data
                monthModel.fit(X_train, y_train)

            #Fit the model again using the whole data set
            monthModel.fit(month_X,month_y)
            month_X = preprocessing.scale(month_X)
            #some scoring
            predict = monthModel.predict(month_X)
            #our desired value, casted to int for convenience
            item_cnt_next_month = int(predict[-1])
            #export to submission file
        dict[k] = {'ID': ID,'item_cnt_month':item_cnt_next_month}
    else:
        item_cnt_next_month = int(predict[-1])
```

Figure 3.7: Monthly predictions

Depending on if there's more months than 1 between the final month of the shop/item set and month 34(our desired month), we will need to run the prediction again, for each month until we reach the final outcome. This process is nearly identical to the previous, but we take the monthly_item_cnt of the group and set it to the previously predicted values. Thereby predicting on the already predicted values, doing out-of-sample prediction one step at a time. For every iteration in the loop, the program predicts another step in the future. The program takes the ID value and the final element of the predicted values and sends it to a dictionary as the item_cnt_month for that item/shop set. Once the loop finishes, the dictionary is ported to a final csv file called "submissionFile.csv". A second, sorted version of this file is saved under "submissionFile_sorted.csv"

```
    else:
        item_cnt_next_month = int(predict[-1])
        #export to submission file
        dict[k] = {'ID': ID,'item_cnt_month':item_cnt_next_month}
df_submission = pd.DataFrame.from_dict(dict,'index')
df_submission.to_csv(r'submissionFile.csv', index = False, header=True)
print("finished compiling")
```

Figure 3.8: Program conclusion

ID	item_cnt_month

Table 3.5: final submission data frame

This whole section is estimated to take approximately 30 hours to compile on mid-end modern hardware, where the main bulk of the compilation time is spent on fitting the data onto the model. The program has been overhauled where possible to more efficiently calculate the predictions, but sadly there was not much to be done in regards to the model fitting functions provided by the library.

Chapter 4

Experiments and Testing

Due to the immense size of the data set, the project was initially tested on a single shop/item set. The ratings given for the following models in this section of the report will thus vary slightly between these data frames.

4.1 Machine Learning models

The following machine learning models were tested for the project on shop/item set **(5, 5037)**, and could, with slight alteration to the code, be re-inserted back into the main program.

- Random Forest Regression
- Linear Regression
- Logistic Regression
- polynomial Support Vector Machine
- MLP Neural Network

Confusion matrices were made in addition to the graphs and scoring of the models, however they were omitted from the rapport due to them ending up serving little purpose for the overall quality of the models. However, these can still be viewed, and are readily available in the source code. The logistic regression and classification models include f1-scores as well as the R2 scores and Mean Squared Error.

4.1.1 Linear Regression

Linear Regression performed the worst of all the models tested and was thus not considered an option for the final model to be used for the project. It is simply too simplistic and gets outclassed by the more advanced models in terms of accuracy, however, because of this, it's compilation speed was also slightly faster than it's counterparts.
R2 Score: 0.08
Mean Squared Error: 0.42

```
R2 score: 0.08392418606581797
MSE: 0.4298
unseen data: [1.3280009  1.58313042 1.20259094 1.33015571 1.24762648 1.54455932
 1.67212408 1.79968885 1.29212331]
real data: 31802    1.0
31803    1.0
31804    2.0
31805    2.0
31806    1.0
31807    1.0
31808    1.0
31809    3.0
31810    1.0
Name: monthly_item_cnt, dtype: float64
forecast: 31802    1.328001
31803    1.583130
31804    1.202591
31805    1.330156
31806    1.247626
31807    1.544559
31808    1.672124
31809    1.799689
31810    1.292123
Name: forecast, dtype: float64
```

Figure 4.1: Scores and data for linear regression model

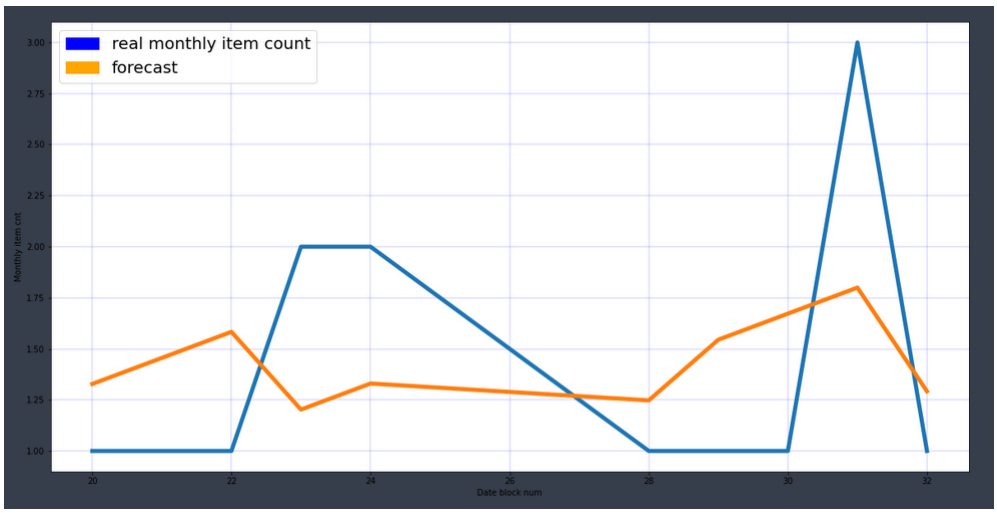

Figure 4.2: Graph of linear regression

4.1.2 Logistic Regression

score-wise, Logistic regression performed better than it's linear counterpart, however there's still room for improvement, as is evident by its visual plot. It sticks to an average of 1 sale monthly with no variation, this model was removed from the

list of potential candidates because of this.
R2 Score: 0.66
Mean Squared Error: 0.66

```
R2 score: 0.6666666666666666
MSE: 0.6667
unseen data: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
real data: 31802    1.0
31803    1.0
31804    2.0
31805    2.0
31806    1.0
31807    1.0
31808    1.0
31809    3.0
31810    1.0
Name: monthly_item_cnt, dtype: float64
forecast: 31802    1.0
31803    1.0
31804    1.0
31805    1.0
31806    1.0
31807    1.0
31808    1.0
31809    1.0
31810    1.0
Name: forecast, dtype: float64
```

Figure 4.3: Scores and data for logistic regression model

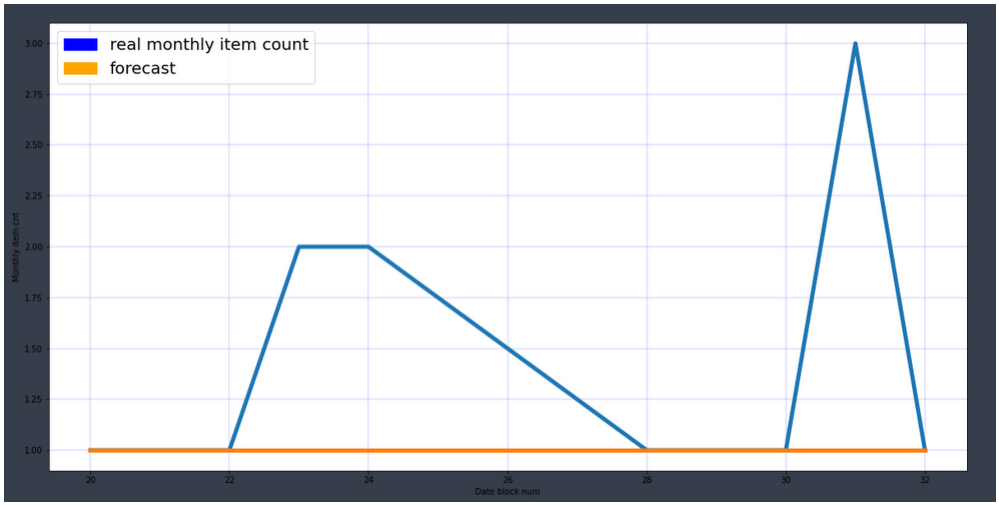

Figure 4.4: Graph of logistic regression

	precision	recall	f1-score	support
1.0	0.67	1.00	0.80	6
2.0	0.00	0.00	0.00	2
3.0	0.00	0.00	0.00	1
accuracy			0.67	9
macro avg	0.22	0.33	0.27	9
weighted avg	0.44	0.67	0.53	9

Figure 4.5: f1-scores for logistic model

4.1.3 Random Forest

The Random Forest regressor performed admirably, with a score that makes it a worthy consideration for the predictions. It also worth noting that it is one of only two models tested to score less than 0.4 for the mean squared error.
R2 Score: 0.77
Mean Squared Error: 0.10

```
R2 score: 0.7767999999999999
MSE: 0.1047
unseen data: [1.1  1.22 1.88 1.92 1.02 1.08 1.12 2.14 1.32]
real data:
 31802    1.0
31803    1.0
31804    2.0
31805    2.0
31806    1.0
31807    1.0
31808    1.0
31809    3.0
31810    1.0
Name: monthly_item_cnt, dtype: float64
forecast:
 31802    1.10
31803    1.22
31804    1.88
31805    1.92
31806    1.02
31807    1.08
31808    1.12
31809    2.14
31810    1.32
Name: forecast, dtype: float64
```

Figure 4.6: Scores and data for random forest model

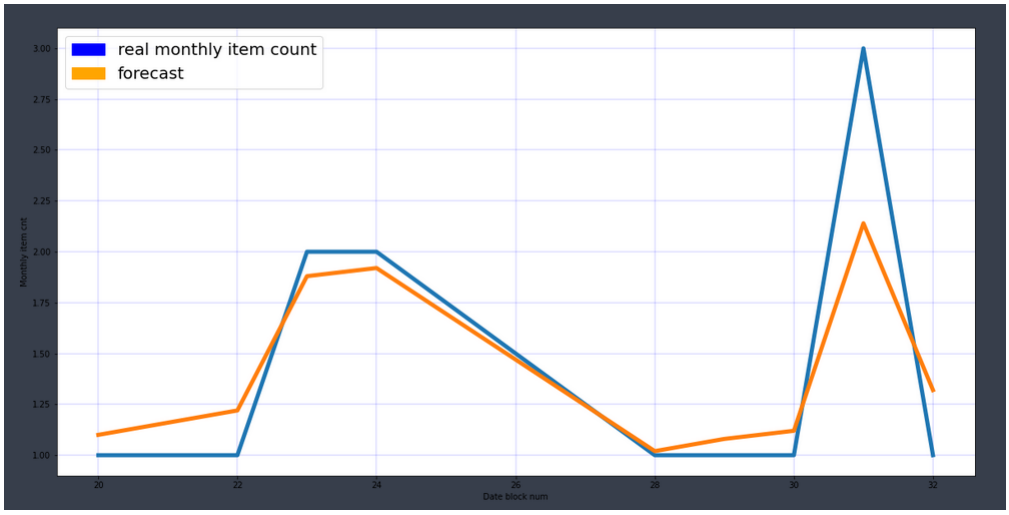

Figure 4.7: Graph of random forest model

4.1.4 Support Vector Machine

SVM performs identically to logistic regression both with it's linear and polynomial kernels, thus this model was also crossed off the list as a potential predictor model for the data set.
R2 Score: 0.66
Mean Squared Error: 0.66

```
R2 score: 0.6666666666666666
MSE: 0.6667
unseen data: [1. 1. 1. 1. 1. 1. 1. 1. 1.]
real data: 31802    1.0
31803    1.0
31804    2.0
31805    2.0
31806    1.0
31807    1.0
31808    1.0
31809    3.0
31810    1.0
Name: monthly_item_cnt, dtype: float64
forecast: 31802    1.0
31803    1.0
31804    1.0
31805    1.0
31806    1.0
31807    1.0
31808    1.0
31809    1.0
31810    1.0
Name: forecast, dtype: float64
```

Figure 4.8: Scores and data of SVM model

Figure 4.9: Graph of SVM model

	precision	recall	f1-score	support
1.0	0.67	1.00	0.80	6
2.0	0.00	0.00	0.00	2
3.0	0.00	0.00	0.00	1
accuracy			0.67	9
macro avg	0.22	0.33	0.27	9
weighted avg	0.44	0.67	0.53	9

Figure 4.10: f1-scores for SVM model

4.1.5 MLP Neural Network

With smaller data sets, the neural network can be a wildcard option that varies its results greatly between runs. However, with the data set for the project, the model manages to perform quite impressive predictions. It's a perfect fit of the original model for its first iteration, it's scores and visualization reflect this. However, it's compilation speed can be a problem if it were to be used on the whole 1.2 million line data set.
R2 Score: 1.00
Mean Squared Error: 0.00

```
R2 score: 1.0
MSE: 0.0000
unseen data: [1. 1. 2. 2. 1. 1. 1. 3. 1.]
real data: 31802    1.0
31803    1.0
31804    2.0
31805    2.0
31806    1.0
31807    1.0
31808    1.0
31809    3.0
31810    1.0
Name: monthly_item_cnt, dtype: float64
forecast: 31802    1.0
31803    1.0
31804    2.0
31805    2.0
31806    1.0
31807    1.0
31808    1.0
31809    3.0
31810    1.0
Name: forecast, dtype: float64
```

Figure 4.11: Scores and data of neural network

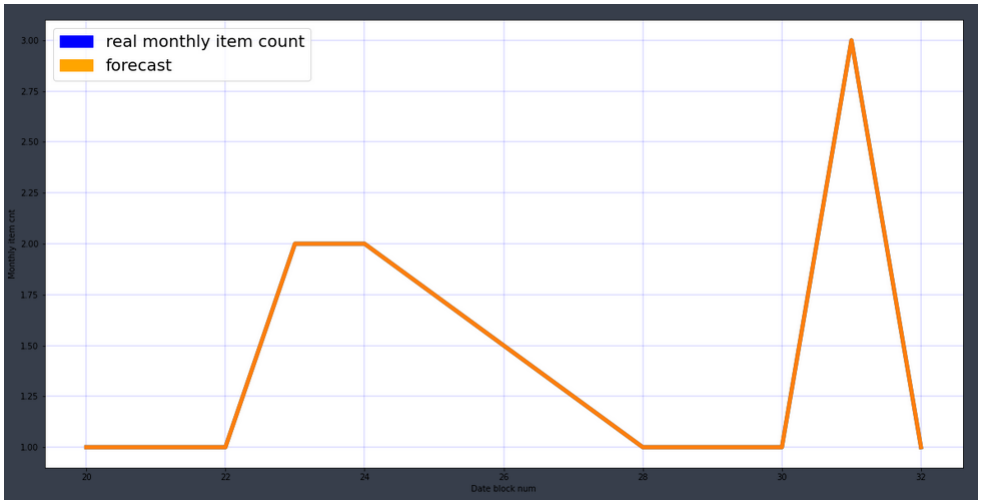

Figure 4.12: graph of neural network model

	precision	recall	f1-score	support
1.0	1.00	1.00	1.00	6
2.0	1.00	1.00	1.00	2
3.0	1.00	1.00	1.00	1
accuracy			1.00	9
macro avg	1.00	1.00	1.00	9
weighted avg	1.00	1.00	1.00	9

Figure 4.13: f1-scores of neural network

4.2 decision

In the end, the choice stood between the Random Forest regressor and the MLP Neural Network model. The Random Forest was chosen over MLP Neural Network, as upon inspection on the visual plots of each model, although I believe the neural network to be a better predictor of potential future data, it would be hard to calculate the predictions for the whole data set given with the neural network model, as it's compilation time would be too unwieldy with data of this size. Thus, the second best model was chosen for calculating on the entire data set. A more lightweight model was chosen not for precision, but for efficiency's sake.

4.3 methods

There were several "methods" devised to complete the task given. In order to predict future events after fitting the model on the data set. The one thing these all had in common was an implementation of multi-step out-of-sample prediction, the initial approach was to create synthetic samples for the X variables. This approach however did not make unique dummies, thus the making of the synthetic X variables proved pointless.

Another idea was to predict the X data for each following step in the prediction chain, to give a more accurate y prediction. The function for doing this is still accessible in the source code, although, it isn't used anywhere. This method was deemed too computation heavy and slow on a slice of the full data set consisting of 2000 samples, thus it was abandoned.

For out-of-sample prediction the use of an ARIMA model was also considered during the process, however it was quickly abandoned as it's syntax was too unwieldy and overall compilation speed was unacceptable for the full size of the data set

The use of Tensorflow and Keras models were also considered, but deep learning in general was deemed overkill for a project of this scope, and the compilation time would've suffered if it were implemented.

In the end, no artificial sample for X data was utilized, the predicted y data utilizes the same X as the original data. The predictions still seem fairy accurate despite this, so nothing of value was lost by not incorporating these extra steps.

Chapter 5

Conclusion

In the end, the values predicted were satisfactory, and the algorithm was applied to the entire data set without fault. The scope of the project was of a suitable size, although the long compilation times came as a bit of a surprise, and having to resort to using the second best performing model instead of the neural network was a disappointing development, but one that was necessary to efficiently complete the project in a realistic time frame.

Were I to do this again I would've spent less time trying out the Keras and ARIMA deep learning models as they turned out to be largely a waste of time. It would've been desirable to find a method to calculate the predictions for the entire data set using the neural network approach, but from experience, it seems models like these will always take a fair amount of processing power, so I am unsure if there was ever anything to be done about that issue.

I do not regret doing the project by my lonesome, as I feel I was more than capable of handling a programming task of this magnitude. I had little to no previous experience with manipulating data sets of this scale, in that sense, it was a learning experience that I was more than happy to handle on my own.