Esther Edith Spurlock (12196692)
CAPP 30254
Assignment 3: Writeup

Note: The day before the assignment was due, I had some Git issues that I could not resolve. I believe I found a solution to this issue and was able to update my code in git, but there is a chance the code in my GitHub will not be able to run. I am *not* asking for your leniency, I am merely trying to apologize for any bugs in my code.

Code Files:
My code contains 3 .py files:

- Full_pipeline: where I call to functions from the beginning through to the end of the pipeline
- Prep_data: where I clean and prepare the data for analysis
- Modeling: where I split the data into training and testing sets by date, create the models, and evaluate those models

Important prep_data.py Functions:
Before I began my modeling, I needed to prepare my data for analysis.

clean_data:
Because most of the columns for this data are specific to a school, I did not feel it was appropriate to merely fill NA values with the mean of the column, so I decided to fill the NA values with None.

After this, I discretized all columns whose $\frac{Number\ of\ Unique\ Values}{Total\ number\ of\ Rows} > 0.6$

```
for col in [LAT, LONG, SUBJECT, AREA, SUBJECT_2, AREA_2, RESOURCE, GRADE,
    PRICE, STUDENTS, DOUBLE]:
    num_entries = df_all_data[col].value_counts().size
    ratio = num_entries / all_rows
    if ratio > 0.60 and ratio != 1.0:
        curr_series = df_all_data[col]
        df_all_data[col] = pd.cut(curr_series, bins=10, labels=False,
            include_lowest=True)
```

generate_var_feat:
The variable for this data set depended on whether Date Fully Funded - Date Posted <= 60. I applied this logic in the following code.

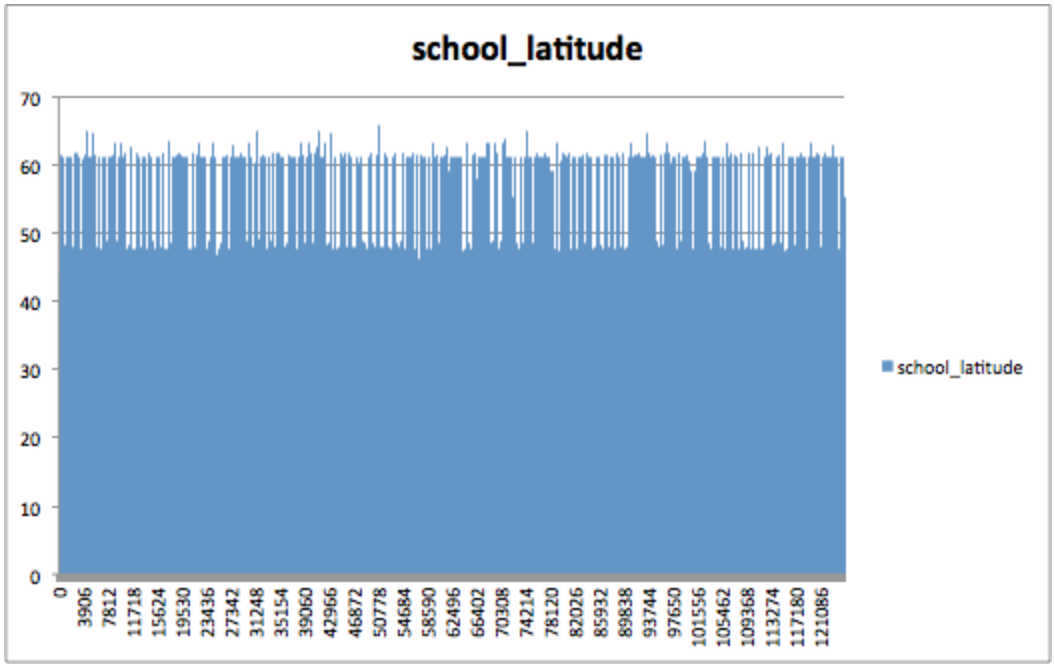
```
#First, we create the variable: 0 of finded within 60 days and 1 if not
df_all_data[VAR] = df_all_data[FUNDED] - df_all_data[POSTED]
df_all_data[VAR] = df_all_data[VAR]\
    .apply(lambda x: 1 if x.days <= 60 else 0)
```

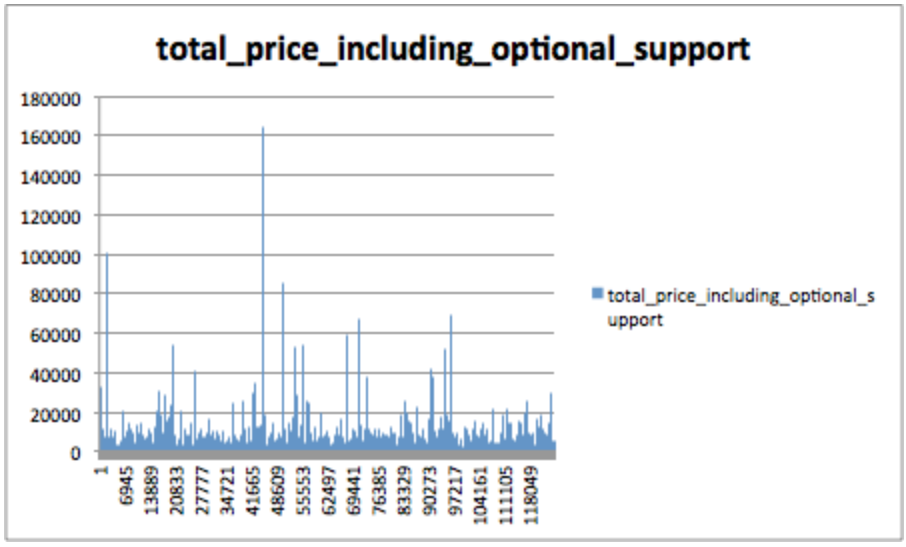
If I were a better coder these two lines would have been merged into one, but my skills being as they are, this works just fine.

After generating the variable columns, I moved on to features. Because of the nature of the models I used, I needed all features to be in float form. After transforming the features into floats, I then checked to see if there was a correlation of more than 0.01. I reasoned that if a column had a correlation less than this, it would not be useful in modeling and I dropped it.

```
for col in possible_features:
    ser = df_all_data[col].astype(dtype='float64', errors='ignore')
    if ser.dtype == 'float64':
        correlation = var_series.corr(ser, method='pearson')
        if abs(correlation) > 0.01:
            df_all_data[col] = ser
            features.append(col)
```

After going through this, I was left with only 2 features: school_latitude and total_price. While I believe I may have ruled out too many columns, I managed to end up with some pretty decent models.





Important modeling.py Functions:
Now that I had my data prepared, I needed to focus on splitting the data and putting it into models.

split_by_date:
I decided to split the data into rolling 180-day segments (about 6 months). The only exception to this is the final testing data which has an extra few days in it so I could keep the data in 3 chunks instead of 4. I created the training and testing splits using the following code.

```
while end_test < final_date:
    #The training data ends 180 days after the beginning of the train
    #the training data begins the day after the ending of train data
    begin_train = end_train + timedelta(days=1)
    end_train = begin_train + timedelta(days=180)
    #Testing data begins the day after training data ends
    #Testing data ends 180 days after it begins
    begin_test = end_train + timedelta(days=1)
    end_test = begin_test + timedelta(days=180)
    #Prevents there being a set that is just a few days
    if (final_date - end_test).days <= 30:
        end_test = final_date
```

After I figured out which dates to use, I then filtered the data by the dates. Then, for use in modeling, I split the training and testing data into their respective variable and feature columns.

```
#Now we create the training and testing data
train_filter =\
    (df_all_data[split] <= end_train) &\
    (df_all_data[split] >= begin_train)
train_data = df_all_data[train_filter]
test_filter =\
    (df_all_data[split] <= end_test) &\
    (df_all_data[split] >= begin_test)
test_data = df_all_data[test_filter]

#Now we have to create the variable and features data
train_variable = train_data[variable]
train_features = train_data[features]
test_variable = test_data[variable]
test_features = test_data[features]
```

Finally, after this, I began a dictionary that would contain all of my information. In this function, I used the dates of the testing data as my key and the value associated with it is the data I received by creating, training, and testing models.

```
models_dict[dates] = training_models(train_variable, train_features,\
    test_variable, test_features)
```

training_models:
In this function, I expand on the dictionary I created in the above function. For this, I created another dictionary. This time, I used the model name as the key and the information about the different models as the value.

```
models_dict[REGRESSION], models_dict[SVM] =\
    regression_svm_modeling(train_variable, train_features, test_variable,\
    test_features)
models_dict[KNN] = knn_modeling(train_variable, train_features,\
    test_variable, test_features)
models_dict[FOREST], models_dict[EXTRA], models_dict[TREE] =\
    forest_modeling(train_variable, train_features, test_variable,\
    test_features)
models_dict[ADA_BOOSTING] = ada_boost_modeling(train_variable,\
    train_features, test_variable, test_features)
models_dict[BAGGING] = bagging_modeling(train_variable, train_features,\
    test_variable, test_features)
```

As you can see from this code, I wrote different functions to create the different models. This is because most models required different parameters. For those that needed the same parameters, I used the same function to call them. After creating an empty model, each of the model functions then called to the training function.

test_models:
This function begins by fitting the current model to the data.

```
model = model_unfit.fit(train_features, train_variable)
```

I could have put the fit into the different model functions, but I wanted to abstract the code a little and not have to write this same line of code for each model I created in each model function.

After fitting the model, I then predicted the model probability that the testing data would result in funding within 60 days.

```
if is_svm:
    probabilities = model.decision_function(test_features)
else:
    probabilities = model.predict_proba(test_features)[:,1]
```

I then added to the dictionary with another dictionary. For this one, the threshold was the key and the value was the different evaluations.

I first called to ROC AUC evaluation since that evaluation only needs the probabilities. I then created a best guess as to whether the output of a row would be 1 or 0 depending on the threshold and used that column to evaluate the model using different evaluations.

```
key = "No Threshold"
roc_auc = roc_auc_score(y_true=test_var, y_score=probabilities)
eval_dict[key] = {ROC_AUC: roc_auc}

#All other evaluations need to loop through the thresholds
for thresh in THRESHOLDS:
    calc_threshold = lambda x,y: 0 if x < y else 1
    predicted = np.array([calc_threshold(score, thresh) for score in
        probabilities])
    key = "Threshold: " + str(thresh)
    eval_dict[key] = evaluate_models(test_var, predicted)
```

evaluate_models:
This function creates the final dictionary. The key is the name of the evaluation metric and the value is how poorly or well the model performs on the metric.

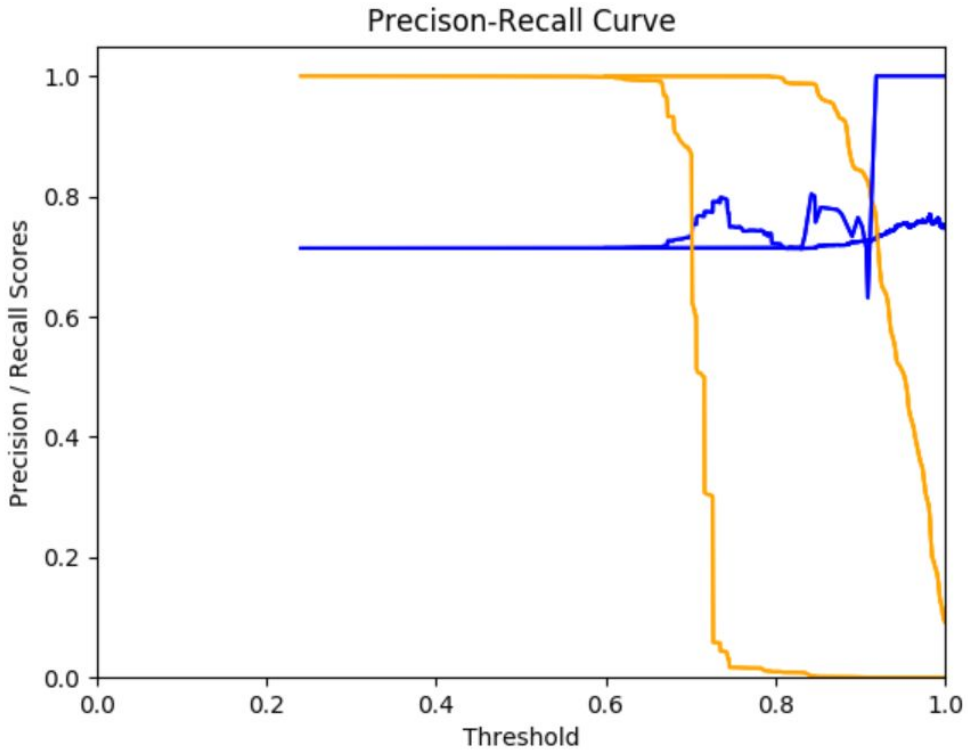
```
eval_dict[ACCURACY] = accuracy(y_true=true, y_pred=predicted)
eval_dict[PRECISION] = precision_score(y_true=true, y_pred=predicted)
eval_dict[RECALL] = recall_score(y_true=true, y_pred=predicted)
eval_dict[F1] = f1_score(y_true=true, y_pred=predicted)
```

plot_pre_rec:
I do not have a call to this function in my code. This function creates a precision-recall curve and saves it to my folder. However, since I did not want to have several thousand graphs saved on my VM, I decided I would only call this function for the models I wanted to

see the full curve for. Based on my analysis, I decided to create a Precision-Recall curve for SVM with a C value of 1.5 and Extra Trees with 5 trees and a depth of 5. For simplicity, I used the last 6 months as testing data and all previous data as training data.

The graph I created is not pretty and I do apologize for the ugliness. I am not good at creating graphs with matplotlib and I hope to get better over the course of this quarter.

And here is the graph I created. Feel free to tear it apart



Important full_pipeline.py Functions:
I created my final return object back in this .py file.

table_models_eval:
Once I have my full dictionary of the dates, models, and evaluations, I needed to put the data into the proper return format. I did this by looping through the dictionary to put the data into a pandas dataframe to return.

```
for dates, model_dict in models_eval.items():
    for model, param_dict in model_dict.items():
        for param, eval_dict in param_dict.items():
            for threshold, eval_outcome_dict in eval_dict.items():
                for eval_name, outcome in eval_outcome_dict.items():
                    this_lst = [dates, model, param, eval_name, threshold,\
                        outcome]
                    df_lst.append(this_lst)

df_evaluated_models = pd.DataFrame(np.array(df_lst), columns=col_lst)
```

While I understand that a dictionary is not necessarily the best way to store data and it would probably have been best if I had put all data directly into the pandas dataframe, this is the way that was easiest for me considering my current coding expertise.

Places For Improvements:

My Biggest Weaknesses:
I easily spent 30 hours or more on this assignment and could have spent another 30 hours on this assignment if given the time.

Probably my biggest weakness was in the prep_data.py file. I need to figure out a better way of exploring the data, a better way to generate features, and a better method for filling na values.

My second biggest weakness is my inability to graph data nicely using matplotlib.

I also suspect that I made errors in how I called to model functions and how I evaluated those models. I would greatly appreciate any feedback you can give on where I can improve my modeling and evaluation.

Parameters:
For this assignment, I looked through sklearn documentation of all the models I used and I decided to try different parameters for a few variables. If I had more time to experiment and if my VM had more processing power, I would have tried more values on each of the parameters I used and I would have tried using more parameters.

In particular, for Extra Trees and Random Forests, there is another parameter to limit the maximum number of leaf nodes. Originally, I had planned to try out different parameters for this, but when I began running my code, I saw that I had too many models being created and I felt the need to cut that parameter from things I would play around with.

Data Details:
Here are some high-level metrics of what is in my data. For my full dataset of metrics and evaluations, please see Modeling_Projects_2012_2013.csv.

Training and Testing Splits:
For this data, I had 3 testing sets

- 6/30/2012 - 12/27/2012
- 12/28/2012 - 6/26/2013
- 6/27/2013 - 12/31/2013

The training data for each of these sets is the 180-day period directly before these dates.

Analysis by Dates:

6/30/2012 - 12/27/2012

- Best Accuracy: 0.743185299
    - K Nearest Neighbor with 10 neighbors
    - Random Forests with 25 trees at a depth of 50 at thresholds of 0.01 and 0.02
- Best Precision: 1
    - Support Vector Machines with a c-value of 1.0 at a 0.5 threshold
    - Support Vector Machines with a c-value all have a precision of greater > 0.9
- Bes Recall: 1
    - There are 260 recall values of 1
    - They occur using different parameters and different thresholds for all models
- Best F1: 0.852639348
    - Bagging with 30 estimators and 100 samples with a threshold of 0.2
- 2nd Best F1: 0.852609383
    - Extra trees with 5 trees and a max depth of 5 at all thresholds
    - Bagging with 10 estimators and 50 samples at a threshold of 0.2
- Best ROC AUC: 0.647017898
    - Support Vector Machines with a C value of 1.5

12/28/2012 - 6/26/2013

- Best Accuracy: 0.690637720488466
    - SVM with a c-value of 1.0
    - This model works relatively well for all thresholds
- Best Precision: 0.981818182
    - SVM with a c-value of 2.5
    - SVMs with a c-value of 1.5 and 2.5 work relatively well for all thresholds
- Best Recall: 1
    - There are 289 recall values of 1

    - They occur using different parameters and different thresholds for all models
- Best F1: 0.815404678005604
    - Bagging with 5 estimators and 50 samples
    - The model that consistently does well for all thresholds is Extra Trees with 5 trees at a max depth of 5 (other Extra Tree models also work well, but this is the simplest one)
- Best ROC AUC: 0.660041011408685
    - Ada Boosting with 200 estimators and a learning rate of 0.5
    - Generally, Ada Boosting models work well for this evaluation metric

6/27/2013 - 12/31/2013

- Best Accuracy: 0.713972234661889
    - Extra trees with 75 trees and a depth of 20 at a 0.1 threshold
    - Extra trees with 5 trees and a depth of 5 work well for all thresholds
- Best Precision: 1
    - SVM with c values of 1.2 and 1.0 at thresholds of 0.3 and 0.2
    - SVM with these c values work well for all thresholds
- Best Recall: 1
    - There are 264 recall values of 1
    - They occur using different parameters and different thresholds for all models
- Best F1: 0.833076340067428
    - Bagging with 10 estimators and 10 samples at a threshold of 0.2
    - Extra Trees with 5 trees and a depth of 5 work well for all thresholds
- Best ROC AUC: 0.658466837693059
    - Ada boosting with 10 estimators and a learning rate of 0.5