Final Case Study

MSDS Fall 2019

7333 Quantifying the World

Brandon de la Houssaye, Bruce Granger, Daniel Serna

# Abstract

In this paper, we demonstrate the use of multiple machine learning techniques to garner insight from an obfuscated data set. Despite being provided little context into the provided data, we are able to produce business value by leveraging a variety of traditional classifiers along with neural networks. Through comparison of different approaches, we recommend an algorithm that achieves effective accuracy metrics while minimizing false positives and false negatives.

# 1 Introduction

We have been approached with a dataset and an interesting problem to solve. Our business stakeholders would like us to consume a dataset that has had domain context removed. All feature column names have been obfuscated, leaving it up to the team to garner insight as best we can. Ultimately, we have been tasked with building a model that can accurately predict the value of a column labeled “y.”

Through consultation with our business stakeholders, we have gained some minor context around the provided data set. At a high level, we know the data relates to insurance claims in some manner. We also know that our model predictions have monetary impact to the business. Most importantly, bad predictions cost the business money. Additionally, we have also been informed that false positives incur a 10x greater cost than false negatives. Thus, while achieving high accuracy with our model is desirable, we must ensure we minimize our false predictions and specifically focus on limiting false positives as much as possible. Because of this constraint, the recall metric of our models becomes important for analysis.

The rest of this paper is organized as follows. We will first discuss the data provided to us in section two with an explanation of data cleansing methods following in section three. Section four gives an overview of the models and techniques we applied against the data. Our results follow in section five. Lastly, our final conclusions are presented in section six.

# 2 Data

As mentioned above, the data provided has been obfuscated by having all feature column names removed. The data contains fifty feature columns labeled x0 through x49 and one prediction column labeled y. Of the fitty feature columns, forty-seven are numeric variables and three are categorical variables. We are dealing with a binary prediction variable as column y only contains the values 1 or 0.

The dataset contains 160,000 records and for the most part is well formed. Fortunately, we do not have much missing data to deal with as can be seen in Figure 1 (note we have renamed some of the original columns).



Figure – Count of missing values in original data

As can be seen from Figure 1, all columns have less than fifty missing data points. When compared to the total record count of 160,000, this is almost a negligible number of missing values (<0.03% for each column). We discuss our handling of the missing values in the Data Cleansing section.

Our data also appears to follow normal distributions and no significant deviations from normality were observed as can be from the boxplots in Figure 2.



Figure - Boxplots

As can be seen in the figure of above, we do not see evidence of any left or right skewness in the distributions. Two columns (x6 and x47) appear to have a wider distribution than the others, but still appear to follow a normal distribution. The boxplots also show little evidence of outliers. Thus, no transformations of the original data were deemed necessary.

We do have some collinearity in the data as can be seen from the Pearson correlation matrix in Figure 3.



Figure - Pearson correlation

As can be seen in Figure 3, columns x2 and x6 are highly correlated with each other and columns x38 and x41 are highly correlated with each other. We will discuss our handling of these correlated columns in the Data Cleansing section.

# 3 Data Cleansing

We first decided to deal with our correlated columns (x2 and x6, x38 and x41). Since these columns were perfectly correlated with each other (correlation coefficient = 1), we decided to simply drop the x2 and x38 columns.

Through exploratory data analysis, we were able to gain insight into the meaning of some of our columns. Column x24 contained the unique values of euorpe, asia, and america. Thus, it was clear to us this was data specifying a continent and we renamed the column to “Continent.” We also cleaned up the existing data by rectifying the spelling mistake of Europe and capitalizing the existing data.

Column x29 contained the unique values of July, Aug, Jun, May, sept., Apr, Nov, Oct, Mar, Feb, Dev, and January. Clearly, this column represents month data so we renamed the column to “Month.” We also standardized the existing values to a three-letter abbreviation and rectified the spelling mistake of Dev.

Column x30 contained the unique values of monday, tuesday, wednesday, thurday, and friday. It is evident this column represents the day of the week so we renamed the column to “DayOfWeek.” We also standardized the existing values to a three-letter abbreviation (while accounting for the misspelling of Thursday).

All values in column x37 had ‘$’ as the first character value, so this column was renamed to “Money.” We also stripped off the dollar sign and converted existing values to float datatypes so this column could be treated as a numerical variable. Similarly, all values in column x32 had ‘%’ as the last character value, so we stripped off the percent sign and converted existing values to float datatypes as well.

With our data standardized, money and percent columns converted to floats, we next handled our missing values. As mentioned previously, the percentage of missing values we are dealing with is very small (< 0.03% for each column). We decided to impute the missing values for our numeric variables with their respective column means. After this imputation, the only remaining columns left with missing values were the categorical columns as seen in Figure 4.



Figure – Count of missing values after mean imputation

As we can see in Figure 4, we still have 28 missing values in Continent, and 30 missing values in both Month and DayOfWeek. Since these represent such a low percentage of our total dataset, we decided to simply drop these records. We felt attempting to come up with a placeholder for these missing values could do more harm to our analysis than simply dropping the records.

# 4 Methods

We decided to employ multiple different modeling approaches against our data with a desire to compare the results to determine the optimal model choice. We utilized traditional classifiers, neural networks, and ensemble approaches as part of our model experimentation. All of our model runs employed an 80/20 train/test split.

We first attempted a number of Naïve Bayes classifiers including Gaussian, Multinomial, Bernoulli, and Complement variations. For the Complement and Multinomial variations, we utilized a Min-Max scaler between 0 and 1 on the data prior to running the models.

We next attempted a K-Nearest Neighbors model combined with a custom grid search method to determine optimal parameter choices. Our custom grid search method experimented with the number of neighbors as well as different weights (uniform and distance) and powers (1 and 2).

Since we are dealing with a binary prediction variable, a Logistic Regression model should work well and was attempted next.

We next utilized a Random Forest classifier with 100 estimators against the data. Random Forest is an ensemble-based algorithm that utilizes bagging techniques and is built from decision tree frameworks. Bagging is where a subset (or ‘bagged’) of variables/features are selected at random in order to build a collection of decision trees.

Next, an XGBoost classifier was used. XGBoost is an open source Algorithm that uses gradient boosting in order to decrease or minimize errors in sequential models. XGBoost is in of itself an ensemble algorithm that is built from the basic concepts of decision trees.

We next utilized Principle Component Analysis and leveraged this with both a Support Vector Machine model and neural network implementation. Our neural network implementation consisted of eight layers utilizing both dense and dropout layers.

Finally, we attempted an ensemble approach. For the first level of stacking, we utilized Gaussian Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Complement Naïve Bayes, and Random Forest. The predictions from this level were fed into a subsequent level consisting of Support Vector Machine and XGBoost models. This output was fed into a final level consisting of a four-layer neural network.

# 5 Results

Given the constraint that false positives incur a 10x greater cost than false negatives, it is important that we look at both precision and recall metrics when determining the optimal model. Because both precision and recall are important for our problem at hand, we utilized model F1 score as the gauge for model effectiveness.

Figure 5 shows the results from our Logistic Regression model.

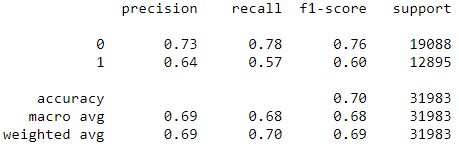


Figure - Logistic Regression Results

It looks like Logistic Regression model performs fairly well at predicting the 0 class with a 0.76 F1-score, however, it performs poorer at predicting the 1 class only achieving an F1-score of 0.60.

As mentioned, for our K Nearest Neighbors (KNN) model, we utilized a custom grid search method to find optimal neighbors, weight function, and power parameters. Our results for optimal F1-scores for class 0 prediction are shown in Figure 6 and optimal F1-scores for class 1 prediction are shown in Figure 7.

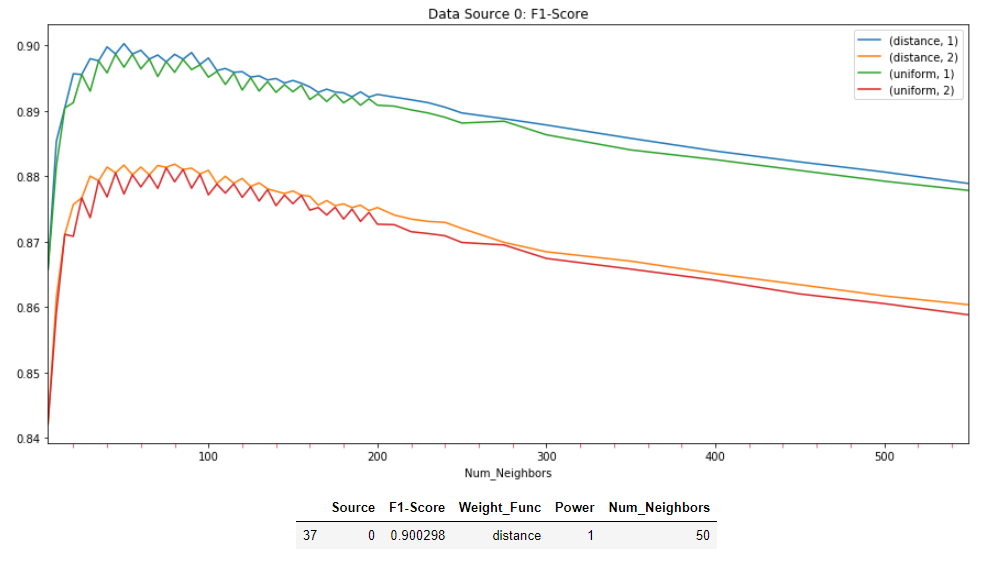


Figure - KNN Optimal Class 0 Prediction

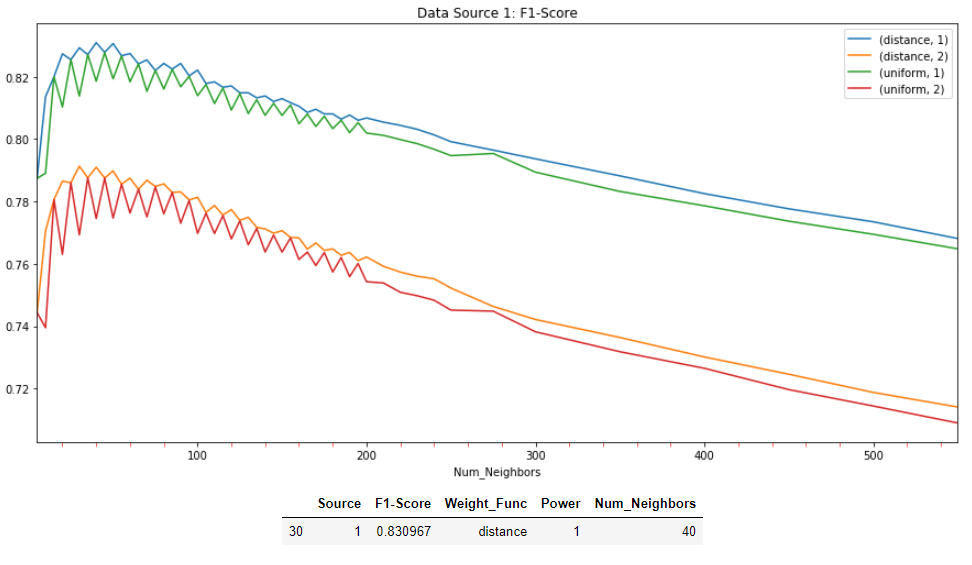


Figure - KNN Optimal Class 1 Prediction

As we can see from Figure 6 and Figure 7, we had much greater success with our KNN approach. We achieved an F1-score of .90 for predicting class 0 utilizing 50 neighbors, power of 1, and distance weight function. For predicting class 1, we achieved an F1 score of 0.831 using 40 neighbors, power of 1, and distance weight function. It appears a power of 1 and distance weight function is the optimal choice regardless of the class we are trying to predict.

The results of our Naïve Bayes Gaussian variant are shown in Figure 8. We received an F1-score of 0.75 for class 0 prediction and F1-score of 0.61 for class 1 prediction. Thus, Naïve Bayes Gaussian did not outperform our KNN approach.

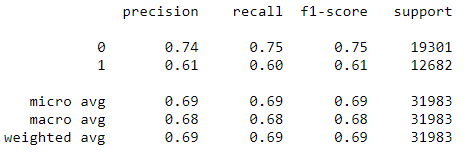


Figure - Naive Bayes Gaussian Results

The results of our Multinomial Naïve Base model are shown in Figure 9. We received and F1-score of 0.75 for class 0 prediction and 0.00 for class 1 prediction. We are not sure why we received a 0.00 for class 1 prediction, thus we cannot recommend the Multinomial Naïve Bayes model.

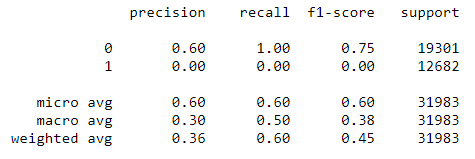


Figure - Multinomial Naive Bayes Results

The results of our Complement Naïve Bayes model are shown in Figure 10. We encountered a similar problem as with the Multinomial Naïve Bayes model. Our class 0 prediction has an F1-score of 0.00. Since we cannot readily explain these results, we do not recommend the Complement Naïve Bayes model.

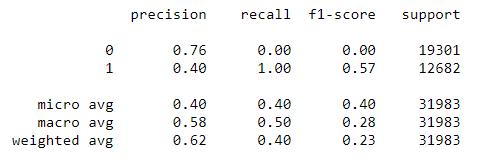


Figure - Complement Naive Bayes Results

The results of our Random Forest model are show in Figure 11. We achieved an F1-score of 0.58 for class 0 predictions and an F1-score of 0.47 for class 1 predictions. Our KNN model performs better than the Random Forest model.

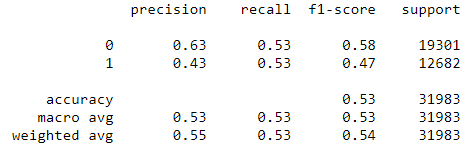


Figure - Random Forest Results

The results of our XGBoost model are shown in Figure 12. We achieved an F1-score of 0.87 for class 0 predictions and an F1-score of 0.78 for class 1 predictions. These are great results and the XGBoost model performed just slightly worse than our KNN model.

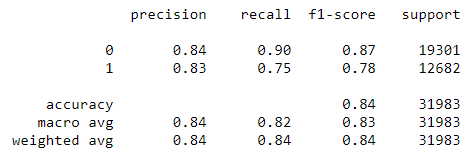


Figure - XGBoost Results

After assessing all of the model results, the [optimal model for class 0] performed the best at classifying class 0 with an F1 score of [optimal F1 score for class 0]. The [optimal model for class 1] performed the best at classifying class 1 with an F1 score of [optimal F1 score for class 1].

# 6 Conclusions

We have demonstrated an ability to draw insights from an obfuscated dataset. Despite having domain context removed from our dataset, we have provided our business stakeholders with well performing models. Knowing false positives incur a 10x greater cost than false negatives, we focus on a blended metric (the F1-score) which takes both precision and recall into account. Through our analysis, we determined the optimal model choice will depend upon the desired class prediction. Since we are not sure what class 0 or 1 represent, we present our optimal model choices for each class and leave it up to our stakeholders to determine which class prediction they desire. For predicting class 0, we recommend the [class 0 optimal model choice]. For predicting class 1, we recommend the [class 1 optimal model choice]. We also recommend the models are monitored on initial deployment prior to enabling full automation.

# Appendix

## Logistic Regression

import pandas as pd

import pandas\_profiling

import numpy as np

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# ## Load Data

# In[3]:

data = pd.read\_csv('final\_project.csv')

# In[4]:

data.head()

# ## Boxplots

# In[5]:

data.boxplot(grid = False,figsize=(40,20)) #figsize=(Width,Height)

# ## Boxplot By Y Attribute

# In[6]:

data.y.unique()

# In[7]:

data\_zero = data[data['y']== 0]

data\_one = data[data['y']== 1]

# In[8]:

data\_zero.boxplot(grid = False, figsize=(40,20)) #figsize=(Width,Height)

# In[9]:

data\_one.boxplot(grid = False, figsize=(40,20)) #figsize=(Width,Height)

# ### Delete temp datasets

# In[10]:

del([data\_zero, data\_one])

# ## Pandas Profile data

# In[11]:

profile = pandas\_profiling.ProfileReport(data)

# In[12]:

profile

# As a result of the above Profiling analysis, which was

# - x6 is highly correlated with x2 (ρ = 1) Rejected

# - x41 is highly correlated with x38 (ρ = 1) Rejected

#

# Attributes X2 and x38 will be removed from the data set.

# In[13]:

drop\_columns = ['x2', 'x38']

data = data.drop(drop\_columns, 1)

del(drop\_columns)

# ## Attributes x24, x29, x30, x37

# - Convert x24, x29, and x30 Datatype to Categorical

# - Rename Attribute:

# - x24 to Continent

# - x29 to Month

# - x30 to DayOfWeek

# - x37 to Money

# In[14]:

data.x24 = data.x24.astype('category')

data.x29 = data.x29.astype('category')

data.x30 = data.x30.astype('category')

data.rename(columns={'x24': 'Continent', 'x29': 'Month', 'x30': 'DayOfWeek', 'x37': 'Money'}, inplace=True)

# ### Standardize Continent

# In[15]:

data['Continent'].unique().to\_list()

# In[16]:

data['Continent'] = data['Continent'].apply(lambda x: x.replace('euorpe', 'Europe').replace('asia', 'Asia') .replace('america', 'America'))

# In[17]:

data['Continent'].unique().to\_list()

# ### Standardize Month Abbrevations

# - Jan

# - Feb

# - Mar

# - Apr

# - May

# - Jun

# - Jul

# - Aug

# - Sep

# - Oct

# - Nov

# - Dec

# In[18]:

data.Month.unique().to\_list()

# In[19]:

data['Month'] = data['Month'].apply(lambda x: x.replace('January', 'Jan').replace('July', 'Jul').replace('sept.', 'Sep').replace('Dev','Dec'))

# In[20]:

data.Month.unique().to\_list()

# ### Standardize DayOfWeek Abbrevations

# In[21]:

data.DayOfWeek.unique().to\_list()

# In[22]:

data['DayOfWeek'] = data['DayOfWeek'].apply(lambda x: x.replace('monday', 'Mon').replace('tuesday', 'Tue').replace('wednesday', 'Wed') .replace('thurday','Thu').replace('friday','Fri'))

# In[23]:

data.DayOfWeek.unique().to\_list()

# ### Money Attribute

# In[24]:

data['Money'] = data['Money'].astype('str')

# In[25]:

data['Money'] = data['Money'].apply(lambda x: x.replace('$', '').replace(',', '')).astype('float')

# In[26]:

data.info()

# ### Convert x32 column from object to float

# In[27]:

data.x32 = data.x32.str.strip('%')

data.x32 = data.x32.astype('float')

# ## Write CSV

# In[28]:

data.to\_csv('final\_project\_mod.csv', index=False)

# ### Delete Raw Data Set

# In[29]:

del(data)

# In[30]:

import pandas as pd

import numpy as np

from pandas.api.types import CategoricalDtype

# ## Load Data

# In[31]:

data = pd.read\_csv('final\_project\_mod.csv', dtype = {'Continent': 'category', 'Month':'category', 'DayOfWeek':'category'})

# ### Explicitly Define Order Category for Month and DayOfWeek

# In[32]:

# The fist step in ordering a category feature is to define a custom order by way of CategoricalDtype

month\_ordered\_categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Next, the attribute needs to be explicitly re\_ordered:

cat\_dtype = pd.api.types.CategoricalDtype(categories=month\_ordered\_categories, ordered=True)

data.Month = data.Month.astype(cat\_dtype)

# Repeat steps for DayOfWeek

dayofweek\_ordered\_categories = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']

cat\_dtype = pd.api.types.CategoricalDtype(categories=dayofweek\_ordered\_categories, ordered=True)

data.DayOfWeek = data.DayOfWeek.astype(cat\_dtype)

# Repate Steps for Continent

# Ordering based upon population

continent\_ordered\_categories = ['Asia', 'America', 'Europe']

cat\_dtype = pd.api.types.CategoricalDtype(categories=continent\_ordered\_categories, ordered=True)

data.Continent = data.Continent.astype(cat\_dtype)

# In[33]:

data.Month.astype("category").cat.codes.unique()

# In[34]:

data['Month'].dtypes

# In[35]:

data.DayOfWeek.astype("category").cat.codes.unique()

# In[36]:

data['DayOfWeek'].dtypes

# In[37]:

data.Continent.astype("category").cat.codes.unique()

# In[38]:

data['Continent'].dtypes

# ## Replace NaN records with column Mean

# In[39]:

data.isnull().sum()

# In[40]:

column\_means = {}

for column in data.columns:

if (data[column].dtype.name != "category" and column != 'y'):

column\_means[column] = data[column].mean()

# In[41]:

for key in column\_means:

data[key] = data[key].fillna(column\_means[key])

# ### Check that we no longer have nulls in our numeric columns.

# In[42]:

data.isnull().sum()

# ### Remove records with null continent, month, or day of week.

# In[43]:

data.count

# In[44]:

data = data[pd.notnull(data.Continent)]

data = data[pd.notnull(data.Month)]

data = data[pd.notnull(data.DayOfWeek)]

# In[45]:

data.count

# ### Recheck our null counts

# In[46]:

data.isnull().sum()

# What is the mode for column x0

# In[47]:

data['x49'].mean()

# Return to nan within the first location (row 1282) within the x0 column to see that it has been replace with the mode

# In[48]:

data.to\_csv('final\_project\_mod.csv', index=False)

# ### Logistic Regression Classifier

# In[49]:

from sklearn.linear\_model import LogisticRegressionCV

# In[50]:

data = pd.read\_csv('final\_project\_mod.csv', dtype = {'Continent': 'category', 'Month':'category', 'DayOfWeek':'category'})

# In[51]:

X = data.drop('y', axis=1)

y = data['y']

# In[52]:

X = pd.get\_dummies(X,prefix = ['Continent', 'Month', 'DayOfWeek'])

# In[53]:

clf = LogisticRegressionCV(cv=5, random\_state=0, multi\_class='multinomial').fit(X, y)

# In[54]:

clf.score(X, y)

# In[55]:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# In[64]:

from sklearn.linear\_model import LogisticRegression

clf = LogisticRegression(random\_state=0, solver='lbfgs', multi\_class='multinomial').fit(X\_train, y\_train)

# In[65]:

y\_pred = clf.predict(X\_test)

# In[67]:

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

# In[69]:

logit\_class\_report = pd.DataFrame(classification\_report(y\_test, y\_pred, output\_dict=True)).transpose().reset\_index()

logit\_class\_report["source\_model"] = "LogisticRegression"

logit\_class\_report.to\_csv('logit\_classification\_report.csv', index=False)

# In[ ]:

## K Nearest Neighbors

import pandas as pd

import pandas\_profiling

import numpy as np

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# ## Load Data

# In[2]:

data = pd.read\_csv('final\_project.csv')

# In[3]:

data.head()

# ## Boxplots

# In[4]:

data.boxplot(grid = False,figsize=(40,20)) #figsize=(Width,Height)

# ## Boxplot By Y Attribute

# In[5]:

data.y.unique()

# In[6]:

data\_zero = data[data['y']== 0]

data\_one = data[data['y']== 1]

# In[7]:

data\_zero.boxplot(grid = False, figsize=(40,20)) #figsize=(Width,Height)

# In[8]:

data\_one.boxplot(grid = False, figsize=(40,20)) #figsize=(Width,Height)

# ### Delete temp datasets

# In[9]:

del([data\_zero, data\_one])

# ## Pandas Profile data

# In[9]:

profile = pandas\_profiling.ProfileReport(data)

# In[10]:

profile

# As a result of the above Profiling analysis, which was

# - x6 is highly correlated with x2 (ρ = 1) Rejected

# - x41 is highly correlated with x38 (ρ = 1) Rejected

#

# Attributes X2 and x38 will be removed from the data set.

# In[10]:

drop\_columns = ['x2', 'x38']

data = data.drop(drop\_columns, 1)

del(drop\_columns)

# ## Attributes x24, x29, x30, x37

# - Convert x24, x29, and x30 Datatype to Categorical

# - Rename Attribute:

# - x24 to Continent

# - x29 to Month

# - x30 to DayOfWeek

# - x37 to Money

# In[11]:

data.x24 = data.x24.astype('category')

data.x29 = data.x29.astype('category')

data.x30 = data.x30.astype('category')

data.rename(columns={'x24': 'Continent', 'x29': 'Month', 'x30': 'DayOfWeek', 'x37': 'Money'}, inplace=True)

# ### Standardize Continent

# In[12]:

data['Continent'].unique().to\_list()

# In[13]:

data['Continent'] = data['Continent'].apply(lambda x: x.replace('euorpe', 'Europe').replace('asia', 'Asia') .replace('america', 'America'))

# In[14]:

data['Continent'].unique().to\_list()

# ### Standardize Month Abbrevations

# - Jan

# - Feb

# - Mar

# - Apr

# - May

# - Jun

# - Jul

# - Aug

# - Sep

# - Oct

# - Nov

# - Dec

# In[15]:

data.Month.unique().to\_list()

# In[16]:

data['Month'] = data['Month'].apply(lambda x: x.replace('January', 'Jan').replace('July', 'Jul').replace('sept.', 'Sep').replace('Dev','Dec'))

# In[17]:

data.Month.unique().to\_list()

# ### Standardize DayOfWeek Abbrevations

# In[18]:

data.DayOfWeek.unique().to\_list()

# In[19]:

data['DayOfWeek'] = data['DayOfWeek'].apply(lambda x: x.replace('monday', 'Mon').replace('tuesday', 'Tue').replace('wednesday', 'Wed') .replace('thurday','Thu').replace('friday','Fri'))

# In[20]:

data.DayOfWeek.unique().to\_list()

# ### Money Attribute

# In[21]:

data['Money'] = data['Money'].astype('str')

# In[22]:

data['Money'] = data['Money'].apply(lambda x: x.replace('$', '').replace(',', '')).astype('float')

# In[23]:

data.info()

# ### Convert x32 column from object to float

# In[24]:

data.x32 = data.x32.str.strip('%')

data.x32 = data.x32.astype('float')

# ## Write CSV

# In[25]:

data.to\_csv('final\_project\_mod.csv', index=False)

# ### Delete Raw Data Set

# In[26]:

del(data)

# In[27]:

import pandas as pd

import numpy as np

from pandas.api.types import CategoricalDtype

# ## Load Data

# In[28]:

data = pd.read\_csv('final\_project\_mod.csv', dtype = {'Continent': 'category', 'Month':'category', 'DayOfWeek':'category'})

# ### Explicitly Define Order Category for Month and DayOfWeek

# In[29]:

# The fist step in ordering a category feature is to define a custom order by way of CategoricalDtype

month\_ordered\_categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Next, the attribute needs to be explicitly re\_ordered:

cat\_dtype = pd.api.types.CategoricalDtype(categories=month\_ordered\_categories, ordered=True)

data.Month = data.Month.astype(cat\_dtype)

# Repeat steps

dayofweek\_ordered\_categories = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']

cat\_dtype = pd.api.types.CategoricalDtype(categories=dayofweek\_ordered\_categories, ordered=True)

data.DayOfWeek = data.DayOfWeek.astype(cat\_dtype)

# In[30]:

data.Month.astype("category").cat.codes.unique()

# In[31]:

data['Month'].dtypes

# In[32]:

data.DayOfWeek.astype("category").cat.codes.unique()

# In[33]:

data['DayOfWeek'].dtypes

# ## Replace NaN records with column Mean

# In[34]:

column\_means = {}

for column in data.columns:

if (data[column].dtype.name != "category" and column != 'y'):

column\_means[column] = data[column].mean()

# In[35]:

for key in column\_means:

data[key] = data[key].fillna(column\_means[key])

# ### Check that we no longer have nulls in our numeric columns.

# In[36]:

data.isnull().sum()

# ### Remove records with null continent, month, or day of week.

# In[37]:

data.count

# In[38]:

data = data[pd.notnull(data.Continent)]

data = data[pd.notnull(data.Month)]

data = data[pd.notnull(data.DayOfWeek)]

# In[39]:

data.count

# ### Recheck our null counts

# In[40]:

data.isnull().sum()

# What is the mode for column x0

# In[41]:

data['x49'].mean()

# Return to nan within the first location (row 1282) within the x0 column to see that it has been replace with the mode

# In[42]:

data.to\_csv('final\_project\_mod.csv', index=False)

# # Nearest Neighbor Analysis

# ## Nearest Neighbor Analysis - Part I: Perform Calculations

# In[1]:

import pandas as pd

import numpy as np

from pandas.api.types import CategoricalDtype

# ## Load Data

# In[2]:

data = pd.read\_csv('final\_project\_mod.csv', dtype = {'Continent': 'category', 'Month':'category', 'DayOfWeek':'category'})

# In[3]:

#data.info()

# In[4]:

#data.Month.unique()

# ### Order the Categorical Varaibles

# In[3]:

# The fist step in ordering a category feature is to define a custom order by way of CategoricalDtype

month\_ordered\_categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Next, the attribute needs to be explicitly re\_ordered:

cat\_dtype = pd.api.types.CategoricalDtype(categories=month\_ordered\_categories, ordered=True)

data.Month = data.Month.astype(cat\_dtype)

# Repeat steps for DayOfWeek

dayofweek\_ordered\_categories = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']

cat\_dtype = pd.api.types.CategoricalDtype(categories=dayofweek\_ordered\_categories, ordered=True)

data.DayOfWeek = data.DayOfWeek.astype(cat\_dtype)

# Repate Steps for Continent

# Ordering based upon population

continent\_ordered\_categories = ['Asia', 'America', 'Europe']

cat\_dtype = pd.api.types.CategoricalDtype(categories=continent\_ordered\_categories, ordered=True)

data.Continent = data.Continent.astype(cat\_dtype)

# In[6]:

#data['Continent'].dtypes

# In[7]:

#data['Continent'].cat.codes.unique()

# In[8]:

#data['Continent'].cat.codes

# In[9]:

#data['Month'].dtypes

# In[10]:

#data['Month'].cat.codes.unique()

# In[11]:

#data['DayOfWeek'].dtypes

# In[12]:

#data['DayOfWeek'].cat.codes.unique()

# In[13]:

#cat = ['Continent', 'Month', 'DayOfWeek']

#data[cat]

# In[14]:

#data[data['DayOfWeek']==0]

# ### Do Categorical Attributes Need to be dropped?

# In[4]:

#drop\_columns = ['x2', 'x38']

#del(drop\_columns)

for col in data.columns:

if col == 'x2':

data = data.drop('x2', 1)

print('Attribute x1 was dropped from the dataset')

elif col == 'x38':

data = data.drop('x38', 1)

print('Attribute x38 was dropped from the dataset')

# ### Map Categorical Features To Numeric Equivalent

# In[5]:

# data['Continent\_Val'] = data['Continent'].map({'Asia': 0, 'America': 1, 'Europe':2})

data['Continent\_Val'] = data.Continent.cat.codes

data['Month\_Val'] = data.Month.cat.codes

data['DayOfWeek\_Val'] = data.DayOfWeek.cat.codes

# In[6]:

# Re-order Columns

reorder\_columns = ['x0', 'x1', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19', 'x20', 'x21', 'x22', 'x23', 'x25', 'x26', 'x27', 'x28', 'x31', 'x32', 'x33', 'x34', 'x35', 'x36', 'Money', 'x39', 'x40', 'x41', 'x42', 'x43', 'x44', 'x45', 'x46', 'x47', 'x48', 'x49', 'Continent\_Val', 'Month\_Val', 'DayOfWeek\_Val', 'y']

data = data[reorder\_columns]

# In[48]:

#len(reorder\_columns)

# ### Map Categorical Features To Dummies

# In[49]:

#data = pd.get\_dummies(data, columns=['Continent'])

#data = pd.get\_dummies(data, columns=['Month'])

#data = pd.get\_dummies(data, columns=['DayOfWeek'])

# In[50]:

# Re-order Columns

#reorder\_columns = ['x0', 'x1', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18',\

# 'x19', 'x20', 'x21', 'x22', 'x23', 'x25', 'x26', 'x27', 'x28', 'x31', 'x32', 'x33', 'x34', 'x35', 'x36', 'Money',\

# 'x39', 'x40', 'x41', 'x42', 'x43', 'x44', 'x45', 'x46', 'x47', 'x48', 'x49', 'Continent\_Asia', 'Continent\_America',\

# 'Continent\_Europe', 'Month\_Jan', 'Month\_Feb', 'Month\_Mar', 'Month\_Apr', 'Month\_May', 'Month\_Jun', 'Month\_Jul',\

# 'Month\_Aug', 'Month\_Sep', 'Month\_Oct', 'Month\_Nov', 'Month\_Dec', 'DayOfWeek\_Mon', 'DayOfWeek\_Tue', 'DayOfWeek\_Wed',\

# 'DayOfWeek\_Thu','DayOfWeek\_Fri', 'y']

#data = data[reorder\_columns]

# ### Split data into X and Y attributes

# In[7]:

X = data.iloc[:, :-1].values

y = data.iloc[:, 48].values

# In[52]:

#X = data.iloc[:, :-1].values

#y = data.iloc[:, 65].values

# ### Create NearestNeighbor Parameter Tuning DataFrame

# In[53]:

#nn\_parameter = pd.DataFrame(columns = ['precision', 'recall', 'f1-score', 'support', 'Weight\_Func','Power','Num\_Neighbors'], index = ['0','1', 'accuracy', 'macro avg', 'weighted avg'])

# ### Create Training and Test Split

# In[8]:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# ### Feature Scaling

# In[9]:

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X\_train = scaler.transform(X\_train)

X\_test = scaler.transform(X\_test)

# ### Training and Predictions

# #### Nearest Neighbor - Parameter Tuning Function

# In[10]:

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report

# In[11]:

def nn\_parameter\_tuning(num\_neighbors, power, weight, parallel\_jobs):

# from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=num\_neighbors, p = power, weights = weight, n\_jobs = parallel\_jobs)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# GENERATE MODEL PERFORMANCE METRICS

nn\_class\_report = pd.DataFrame(classification\_report(y\_test, y\_pred, output\_dict=True)).transpose()

# MEMORALIZE PREDICTION AND ASSOCIATED PARAMETERS

data = [weight, power, num\_neighbors], [weight, power, num\_neighbors], [weight, power, num\_neighbors], [weight, power, num\_neighbors], [weight, power, num\_neighbors]

nn\_parameters = pd.DataFrame(data, columns = ['Weight\_Func','Power','Num\_Neighbors'], index = ['0','1', 'accuracy', 'macro avg', 'weighted avg'])

#global nn\_class\_report\_paramters

#nn\_class\_report\_paramters = pd.merge(nn\_class\_report, nn\_parameters, right\_index=True, left\_index=True)

return(pd.merge(nn\_class\_report, nn\_parameters, right\_index=True, left\_index=True))

# In[12]:

weight\_list = ['uniform', 'distance']

power\_list = [1, 2]

for main\_idx, main\_val in enumerate(range(400, 551, 50)):

if main\_idx == 0:

for weight\_val in weight\_list:

for power\_val in power\_list:

if (weight\_val == 'uniform') & (power\_val == 1):

nn\_class\_report\_paramters = nn\_parameter\_tuning(num\_neighbors = main\_val, power = power\_val, weight = weight\_val, parallel\_jobs = -1)

else:

temp = nn\_parameter\_tuning(num\_neighbors = main\_val, power = power\_val, weight = weight\_val, parallel\_jobs = -1)

nn\_class\_report\_paramters = pd.concat([nn\_class\_report\_paramters, temp], axis=0)

elif main\_idx > 0:

for weight\_val in weight\_list:

for power\_val in power\_list:

temp = nn\_parameter\_tuning(num\_neighbors = main\_val, power = power\_val, weight = weight\_val, parallel\_jobs = -1)

nn\_class\_report\_paramters = pd.concat([nn\_class\_report\_paramters, temp], axis=0)

# In[13]:

nn\_class\_report\_paramters['source\_model'] = 'KNeighborsClassifier'

nn\_class\_report\_paramters.reset\_index(inplace = True)

# In[14]:

nn\_class\_report\_paramters[nn\_class\_report\_paramters['precision'] == nn\_class\_report\_paramters['precision'].max()]

# In[15]:

nn\_class\_report\_paramters[nn\_class\_report\_paramters['recall'] == nn\_class\_report\_paramters['recall'].max()]

# In[16]:

nn\_class\_report\_paramters[nn\_class\_report\_paramters['f1-score'] == nn\_class\_report\_paramters['f1-score'].max()]

# In[17]:

nn\_class\_report\_paramters.head()

# In[18]:

nn\_class\_report\_paramters.Num\_Neighbors.unique()

# In[19]:

nn\_class\_report\_paramters.to\_csv('KNeighborsClassifier\_400\_550.csv', index=False)

# In[20]:

del(nn\_class\_report\_paramters)

# ## Nearest Neighbor Analysis - Part II: Combine nn\_class\_report\_paramters files into one

# In[1]:

import pandas as pd

import numpy as np

# In[50]:

data = pd.read\_csv('KNeighborsClassifier\_5\_20.csv')

temp = pd.read\_csv('KNeighborsClassifier\_25\_30.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_35\_40.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_45\_50.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_55\_60.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_65\_70.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_75\_80.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_85\_90.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_95\_100.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_105\_110.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_115\_120.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_125\_130.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_135\_140.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_145\_150.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_155\_160.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_165\_170.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_175\_180.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_185\_190.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_195\_200.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_210\_220.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_230\_240.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_250\_275.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_300\_350.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

temp = pd.read\_csv('KNeighborsClassifier\_400\_550.csv')

data = pd.concat([data, temp], axis=0)

del(temp)

# In[51]:

data.rename(columns={"index": "Source", "precision": "Precision", "recall":"Recall", "f1-score":"F1-Score", "support":"Support",

"Weight\_Func":"Weight\_Func", "Power":"Power", "Num\_Neighbors":"Num\_Neighbors", "source\_model":"Source\_Model"},

inplace = True)

data.Source = data.Source.astype(str)

# ### Split Data based Source

# In[52]:

data\_Source\_0 = data[data['Source']== '0'].sort\_values(by='Num\_Neighbors').reset\_index(drop=True)

data\_Source\_1 = data[data['Source']== '1'].sort\_values(by='Num\_Neighbors').reset\_index(drop=True)

# ### Filter for Best Precision, Recall, and F1 by Source [0, 1]

# #### Precision

# In[5]:

data\_Source\_0[data\_Source\_0['Precision'] == data\_Source\_0['Precision'].max()]

# In[6]:

data\_Source\_1[data\_Source\_1['Precision'] == data\_Source\_1['Precision'].max()]

# #### Recall

# In[7]:

data\_Source\_0[data\_Source\_0['Recall'] == data\_Source\_0['Recall'].max()]

# In[8]:

data\_Source\_1[data\_Source\_1['Recall'] == data\_Source\_1['Recall'].max()]

# #### F1

# In[9]:

data\_Source\_0[data\_Source\_0['F1-Score'] == data\_Source\_0['F1-Score'].max()]

# In[10]:

data\_Source\_1[data\_Source\_1['F1-Score'] == data\_Source\_1['F1-Score'].max()]

# In[11]:

import matplotlib.pyplot as plot

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# In[12]:

data\_Source\_0.groupby(['Weight\_Func','Power'])['Precision'].plot(legend=True, title="Data Source 0: Precision")

#data\_Source\_0['Precision'].plot.line(title="Data Source 0: Precision");

plot.show(block=True)

# In[13]:

data\_Source\_0[data\_Source\_0['Precision']==data\_Source\_0['Precision'].max()][['Source', 'Precision', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

# In[14]:

data\_Source\_1.groupby(['Weight\_Func','Power'])['Precision'].plot(legend=True, title="Data Source 1: Precision")

#data\_Source\_1['Precision'].plot.line(title="Data Source 1: Precision");

plot.show(block=True)

# In[16]:

data\_Source\_1[data\_Source\_1['Precision']== data\_Source\_1['Precision'].max()][['Source', 'Precision', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

# In[17]:

data\_Source\_0.groupby(['Weight\_Func','Power'])['Recall'].plot(legend=True, title="Data Source 0: Recall")

#data\_Source\_0['Recall'].plot.line(title="Data Source 0: Recall");

plot.show(block=True)

# In[19]:

data\_Source\_0[data\_Source\_0['Recall']== data\_Source\_0['Recall'].max()][['Source', 'Recall', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

# In[20]:

data\_Source\_1.groupby(['Weight\_Func','Power'])['Recall'].plot(legend=True, title="Data Source 1: Recall")

#data\_Source\_1['Recall'].plot.line(title="Data Source 1: Recall");

plot.show(block=True)

# In[21]:

data\_Source\_1[data\_Source\_1['Recall']== data\_Source\_1['Recall'].max()][['Source', 'Recall', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

# In[22]:

data\_Source\_0.groupby(['Weight\_Func','Power'])['F1-Score'].plot(legend=True, title="Data Source 0: F1-Score")

plot.show(block=True)

# In[23]:

data\_Source\_0[data\_Source\_0['F1-Score'] == data\_Source\_0['F1-Score'].max()][['Source', 'F1-Score', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

# In[24]:

data\_Source\_1.groupby(['Weight\_Func','Power'])['F1-Score'].plot(legend=True, title="Data Source 1: F1-Score")

plot.show(block=True)

# In[25]:

data\_Source\_1[data\_Source\_1['F1-Score']== data\_Source\_1['F1-Score'].max()][['Source', 'F1-Score', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

# ### WHERE DO THEY STOP MOVING - ROUND

# ## PRECISION

# In[55]:

data\_Source\_1\_Precision = data\_Source\_1[['Source','Precision', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

data\_Source\_1\_Precision['Precision\_Whole'] = data\_Source\_1\_Precision.apply(lambda row: row.Precision, axis=1)

# In[59]:

data\_Source\_1\_Precision = data\_Source\_1\_Precision.round({'Precision': 2, 'Power': 0, 'Num\_Neighbors':0})

# In[62]:

temp = data\_Source\_1\_Precision[data\_Source\_1\_Precision['Precision'] == data\_Source\_1\_Precision['Precision'].max()]

temp = temp[temp['Num\_Neighbors'] == temp['Num\_Neighbors'].min()]

temp[temp['Precision\_Whole'] == temp['Precision\_Whole'].min()]

# ## RECALL

# In[41]:

data\_Source\_0\_Recall = data\_Source\_0[['Source','Recall', 'Weight\_Func', 'Power', 'Num\_Neighbors']]

data\_Source\_0\_Recall['Recall\_Whole'] = data\_Source\_0\_Recall.apply(lambda row: row.Recall, axis=1)

# In[43]:

data\_Source\_0\_Recall = data\_Source\_0\_Recall.round({'Recall': 2, 'Power': 0, 'Num\_Neighbors':0})

# In[45]:

temp = data\_Source\_0\_Recall[data\_Source\_0\_Recall['Recall'] == data\_Source\_0\_Recall['Recall'].max()]

temp = temp[temp['Num\_Neighbors'] == temp['Num\_Neighbors'].min()]

temp[temp['Recall\_Whole'] == temp['Recall\_Whole'].min()]

# In[ ]:

# In[ ]:

# #### -------------------------------------------------------------- ARCHIVE------------------------------------------------------------------

# In[72]:

#num\_neighbors = 3

# When p = 1, this is equivalent to using manhattan\_distance (l1),

# p = 2 euclidean\_distance (l2)

#power = 1

# weight function used in prediction.

# ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.

# ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will

# have a greater influence than neighbors which are further away.

#weight\_func = 'uniform'

#n\_jobs The number of parallel jobs to run for neighbors search.

#parallel\_n\_jobs = -1

# In[34]:

#from sklearn.neighbors import KNeighborsClassifier

#classifier = KNeighborsClassifier(n\_neighbors=num\_neighbors, p = power, weights = weight\_func, n\_jobs = parallel\_n\_jobs)

#classifier.fit(X\_train, y\_train)

# In[74]:

#y\_pred = classifier.predict(X\_test)

# ### Evaluating the Algorithm - Categorical - Integers

# In[35]:

#from sklearn.metrics import classification\_report, confusion\_matrix

#print(confusion\_matrix(y\_test, y\_pred))

# In[36]:

#print(classification\_report(y\_test, y\_pred))

# In[ ]:

#x = pd.DataFrame(classification\_report(y\_test, y\_pred, output\_dict=True)).transpose()

# In[102]:

#data = [weight\_func, power, num\_neighbors], [weight\_func, power, num\_neighbors], [weight\_func, power, num\_neighbors], \

# [weight\_func, power, num\_neighbors], [weight\_func, power, num\_neighbors]

# x1 = pd.DataFrame(data, columns = ['Weight\_Func','Power','Num\_Neighbors'], index = ['0','1', 'accuracy', 'macro avg', 'weighted avg'])

# In[103]:

#pd.merge(x, x1, right\_index=True, left\_index=True)

# ### Evaluating the Algorithm - Categorical - Dummy

# In[129]:

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

# In[ ]:

## Naïve Bayes, Random Forest, XGBoost

import pandas as pd

import pandas\_profiling

import numpy as np

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# ## Load Data

# In[3]:

data = pd.read\_csv('final\_project.csv')

# In[4]:

data.head()

# ## Boxplots

# In[5]:

data.boxplot(grid = False,figsize=(40,20)) #figsize=(Width,Height)

# ## Boxplot By Y Attribute

# In[6]:

data.y.unique()

# In[7]:

data\_zero = data[data['y']== 0]

data\_one = data[data['y']== 1]

# In[8]:

data\_zero.boxplot(grid = False, figsize=(40,20)) #figsize=(Width,Height)

# In[9]:

data\_one.boxplot(grid = False, figsize=(40,20)) #figsize=(Width,Height)

# ### Delete temp datasets

#

# In[10]:

del([data\_zero, data\_one])

# ## Pandas Profile data

# In[11]:

profile = pandas\_profiling.ProfileReport(data)

# In[12]:

profile

# As a result of the above Profiling analysis, which was

# - x6 is highly correlated with x2 (ρ = 1) Rejected

# - x41 is highly correlated with x38 (ρ = 1) Rejected

#

# Attributes X2 and x38 will be removed from the data set.

# In[13]:

drop\_columns = ['x2', 'x38']

data = data.drop(drop\_columns, 1)

del(drop\_columns)

# ## Attributes x24, x29, x30, x37

# - Convert x24, x29, and x30 Datatype to Categorical

# - Rename Attribute:

# - x24 to Continent

# - x29 to Month

# - x30 to DayOfWeek

# - x37 to Money

# In[14]:

data.x24 = data.x24.astype('category')

data.x29 = data.x29.astype('category')

data.x30 = data.x30.astype('category')

data.rename(columns={'x24': 'Continent', 'x29': 'Month', 'x30': 'DayOfWeek', 'x37': 'Money'}, inplace=True)

# ### Standardize Continent

# In[15]:

data['Continent'].unique().to\_list()

# In[16]:

data['Continent'] = data['Continent'].apply(lambda x: x.replace('euorpe', 'Europe').replace('asia', 'Asia') .replace('america', 'America'))

# In[17]:

data['Continent'].unique().to\_list()

# ### Standardize Month Abbrevations

# - Jan

# - Feb

# - Mar

# - Apr

# - May

# - Jun

# - Jul

# - Aug

# - Sep

# - Oct

# - Nov

# - Dec

# In[18]:

data.Month.unique().to\_list()

# In[19]:

data['Month'] = data['Month'].apply(lambda x: x.replace('January', 'Jan').replace('July', 'Jul').replace('sept.', 'Sep').replace('Dev','Dec'))

# In[20]:

data.Month.unique().to\_list()

# ### Standardize DayOfWeek Abbrevations

# In[21]:

data.DayOfWeek.unique().to\_list()

# In[22]:

data['DayOfWeek'] = data['DayOfWeek'].apply(lambda x: x.replace('monday', 'Mon').replace('tuesday', 'Tue').replace('wednesday', 'Wed') .replace('thurday','Thu').replace('friday','Fri'))

# In[23]:

data.DayOfWeek.unique().to\_list()

# ### Money Attribute

# In[24]:

data['Money'] = data['Money'].astype('str')

# In[25]:

data['Money'] = data['Money'].apply(lambda x: x.replace('$', '').replace(',', '')).astype('float')

# In[26]:

data.info()

# ### Convert x32 column from object to float

# In[27]:

data.x32 = data.x32.str.strip('%')

data.x32 = data.x32.astype('float')

# In[28]:

data.to\_csv('final\_project\_mod.csv', index=False)

# # Update Null Values

# In[29]:

del(data)

# In[30]:

import pandas as pd

import numpy as np

from pandas.api.types import CategoricalDtype

# ## Load Data

# In[31]:

data = pd.read\_csv('final\_project\_mod.csv', dtype = {'Continent': 'category', 'Month':'category', 'DayOfWeek':'category'})

# ### Explicitly Define Order Category for Month and DayOfWeek

# In[32]:

from pandas.api.types import CategoricalDtype

# In[33]:

# The fist step in ordering a category feature is to define a custom order by way of CategoricalDtype

month\_ordered\_categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

# Next, the attribute needs to be explicitly re\_ordered:

cat\_dtype = pd.api.types.CategoricalDtype(categories=month\_ordered\_categories, ordered=True)

data.Month = data.Month.astype(cat\_dtype)

# Repeat steps for DayOfWeek

dayofweek\_ordered\_categories = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']

cat\_dtype = pd.api.types.CategoricalDtype(categories=dayofweek\_ordered\_categories, ordered=True)

data.DayOfWeek = data.DayOfWeek.astype(cat\_dtype)

# Repate Steps for Continent

# Ordering based upon population

continent\_ordered\_categories = ['Asia', 'America', 'Europe']

cat\_dtype = pd.api.types.CategoricalDtype(categories=continent\_ordered\_categories, ordered=True)

data.Continent = data.Continent.astype(cat\_dtype)

# In[34]:

data.Month.astype("category").cat.codes.unique()

# In[35]:

data['Month'].dtypes

# In[36]:

data.DayOfWeek.astype("category").cat.codes.unique()

# In[37]:

data['DayOfWeek'].dtypes

# In[38]:

data.Continent.astype("category").cat.codes.unique()

# In[39]:

data['Continent'].dtypes

# ## Replace NaN records with column Mean

# In[40]:

column\_means = {}

for column in data.columns:

if (data[column].dtype.name != "category" and column != 'y'):

column\_means[column] = data[column].mean()

# In[41]:

for key in column\_means:

data[key] = data[key].fillna(column\_means[key])

# ### Check that we no longer have nulls in our numeric columns.

# In[42]:

data.isnull().sum()

# ### Remove records with null continent, month, or day of week.

# In[43]:

data.count

# Proving there are NaN's within the first ten locations within the x0 column

# In[44]:

data = data[pd.notnull(data.Continent)]

data = data[pd.notnull(data.Month)]

data = data[pd.notnull(data.DayOfWeek)]

# In[45]:

data.count

# In[46]:

data.isnull().sum()

# In[47]:

data.to\_csv('final\_project\_mod.csv', index=False)

# In[48]:

data.info()

# # BEGIN MODEL

# In[ ]:

# In[49]:

#define X

x = data.drop(['y'], axis = 1)

#define y

y = data['y']

x1 = pd.get\_dummies(x,prefix = ['Continent', 'Month', 'DayOfWeek'])

# In[50]:

x1.info()

# # Naive Bayes: Gaussian

#

# ### assuming a normal distribution for the float64 datatypes given the earlier histogram plots

# In[51]:

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

model = GaussianNB()

model.fit(x1,y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x1, y, test\_size=0.20, random\_state=102)

# In[52]:

y\_pred\_gau = model.predict(x\_test)

# In[53]:

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred\_gau))

print(confusion\_matrix(y\_test, y\_pred\_gau))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred\_gau,y\_test))

# # MultinomialNB

# In[54]:

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_train = scaler.fit\_transform(x1)

# Print out the adjustment that the scaler applied to the lepton pT column of data

#print("Note: median values were scaled by multiplying by {:.10f} and adding {:.6f}".format(scaler.scale\_[0],scaler.min\_[0]))

#multiplied\_by = scaler.scale\_[0]

#added = scaler.min\_[0]

scaled\_train\_df = pd.DataFrame(scaled\_train, columns = x1.columns)

# In[55]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(scaled\_train\_df, y, test\_size=0.20, random\_state=102)

from sklearn.naive\_bayes import MultinomialNB

model = MultinomialNB()

model.fit(scaled\_train\_df,y)

y\_pred\_multi = model.predict(x\_test)

# In[56]:

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred\_multi))

print(confusion\_matrix(y\_test, y\_pred\_multi))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred\_multi,y\_test))

# # Bernoulli

# In[57]:

from sklearn.naive\_bayes import BernoulliNB

model = BernoulliNB()

model.fit(x\_train, y\_train)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x1, y, test\_size=0.20, random\_state=102)

y\_pred\_ber = model.predict(x\_test)

# In[58]:

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred\_ber))

print(confusion\_matrix(y\_test, y\_pred\_ber))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred\_ber,y\_test))

# # Complement Naive Bayes

# In[59]:

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_train = scaler.fit\_transform(x1)

# Print out the adjustment that the scaler applied to the lepton pT column of data

#print("Note: median values were scaled by multiplying by {:.10f} and adding {:.6f}".format(scaler.scale\_[0],scaler.min\_[0]))

#multiplied\_by = scaler.scale\_[0]

#added = scaler.min\_[0]

scaled\_train\_df = pd.DataFrame(scaled\_train, columns = x1.columns)

# In[60]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(scaled\_train\_df, y, test\_size=0.20, random\_state=102)

# In[61]:

from sklearn.naive\_bayes import ComplementNB

model = ComplementNB()

model.fit(x\_train, y\_train)

y\_pred\_comp = model.predict(x\_test)

# In[62]:

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred\_comp))

print(confusion\_matrix(y\_test, y\_pred\_comp))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred\_comp,y\_test))

# In[63]:

from sklearn.metrics import accuracy\_score, log\_loss

models = [

GaussianNB(),

MultinomialNB(),

BernoulliNB(),

ComplementNB(),

]

# Logging for Visual Comparison

log\_cols=["Classifier", "Accuracy", "Log Loss"]

log = pd.DataFrame(columns=log\_cols)

# In[64]:

import seaborn as sns

# In[65]:

for clf in models:

clf.fit(x\_train, y\_train)

name = clf.\_\_class\_\_.\_\_name\_\_

print("="\*30)

print(name)

print('\*\*\*\*Results\*\*\*\*')

train\_predictions = clf.predict(x\_test)

acc = accuracy\_score(y\_test, train\_predictions)

print("Accuracy: {:.4%}".format(acc))

log\_entry = pd.DataFrame([[name, acc\*100, 'Log Loss']], columns=log\_cols)

log = log.append(log\_entry)

print("="\*30)

sns.set\_color\_codes("muted")

sns.barplot(x='Accuracy', y='Classifier', data=log, color="b")

plt.xlabel('Accuracy %')

plt.title('Classifier Accuracy')

plt.show()

# # Random Forest

# In[66]:

feature\_list = list(x1.columns)

y = np.array(y)

x1 = np.array(x1)

# In[67]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x1, y, test\_size=0.20, random\_state=102)

# In[68]:

print('Training Features Shape:', x\_train.shape)

print('Training Labels Shape:', y\_train.shape)

print('Testing Features Shape:', x\_test.shape)

print('Testing Labels Shape:', y\_test.shape)

# In[69]:

from sklearn.metrics import confusion\_matrix

import itertools

from sklearn.metrics import precision\_score, recall\_score, roc\_auc\_score, roc\_curve

# In[70]:

# Import the model we are using

from sklearn.ensemble import RandomForestClassifier

# Instantiate model with 1000 decision trees

rf = RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_leaf\_nodes=None,

min\_impurity\_split=1e-03, min\_samples\_leaf=1,

min\_samples\_split=2, n\_estimators = 100, random\_state = 102,

max\_features = 'sqrt', n\_jobs = 2,verbose = 1)

# Train the model on training data

rf.fit(x\_train, y\_train);

# In[71]:

n\_nodes = []

max\_depths = []

for ind\_tree in rf.estimators\_:

n\_nodes.append(ind\_tree.tree\_.node\_count)

max\_depths.append(ind\_tree.tree\_.max\_depth)

print(f'Average number of nodes {int(np.mean(n\_nodes))}')

print(f'Average maximum depth {int(np.mean(max\_depths))}')

# In[72]:

train\_rf\_predictions = model.predict(x\_train)

train\_rf\_probs = model.predict\_proba(x\_train)[:, 1]

rf\_predictions = model.predict(x\_test)

rf\_probs = model.predict\_proba(x\_test)[:, 1]

# In[73]:

from sklearn.tree import DecisionTreeClassifier

# Make a decision tree and train

tree = DecisionTreeClassifier(random\_state=102)

tree.fit(x1, y)

# In[74]:

# Make probability predictions

train\_probs = tree.predict\_proba(x\_train)[:, 1]

probs = tree.predict\_proba(x\_test)[:, 1]

train\_predictions = tree.predict(x\_train)

predictions = tree.predict(x\_test)

# In[75]:

def evaluate\_model(predictions, probs, train\_predictions, train\_probs):

"""Compare machine learning model to baseline performance.

Computes statistics and shows ROC curve."""

baseline = {}

baseline['recall'] = recall\_score(y\_test, [1 for \_ in range(len(y\_test))])

baseline['precision'] = precision\_score(y\_test, [1 for \_ in range(len(y\_test))])

baseline['roc'] = 0.5

results = {}

results['recall'] = recall\_score(y\_test, predictions)

results['precision'] = precision\_score(y\_test, predictions)

results['roc'] = roc\_auc\_score(y\_test, probs)

train\_results = {}

train\_results['recall'] = recall\_score(y\_train, train\_predictions)

train\_results['precision'] = precision\_score(y\_train, train\_predictions)

train\_results['roc'] = roc\_auc\_score(y\_train, train\_probs)

for metric in ['recall', 'precision', 'roc']:

print(f'{metric.capitalize()} Baseline: {round(baseline[metric], 2)} Test: {round(results[metric], 2)} Train: {round(train\_results[metric], 2)}')

# Calculate false positive rates and true positive rates

base\_fpr, base\_tpr, \_ = roc\_curve(y\_test, [1 for \_ in range(len(y\_test))])

model\_fpr, model\_tpr, \_ = roc\_curve(y\_test, probs)

plt.figure(figsize = (8, 6))

plt.rcParams['font.size'] = 16

# Plot both curves

plt.plot(base\_fpr, base\_tpr, 'b', label = 'baseline')

plt.plot(model\_fpr, model\_tpr, 'r', label = 'model')

plt.legend();

plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate'); plt.title('ROC Curves');

# In[76]:

evaluate\_model(rf\_predictions, rf\_probs, train\_rf\_predictions, train\_rf\_probs)

# In[77]:

from sklearn.metrics import confusion\_matrix

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Oranges):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

Source: http://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_confusion\_matrix.html

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.figure(figsize = (10, 10))

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title, size = 24)

plt.colorbar(aspect=4)

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45, size = 14)

plt.yticks(tick\_marks, classes, size = 14)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

# Labeling the plot

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt), fontsize = 20,

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.grid(None)

plt.tight\_layout()

plt.ylabel('True label', size = 18)

plt.xlabel('Predicted label', size = 18)

# In[78]:

cm = confusion\_matrix(y\_test, rf\_predictions)

plot\_confusion\_matrix(cm, classes = ['Good', 'Bad'],

title = 'Ouput Confusion Matrix')

# In[79]:

from sklearn.metrics import classification\_report, confusion\_matrix

# print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, rf\_predictions))

# In[143]:

# Features for feature importances

feat = pd.DataFrame(x\_train)

features = list(feat.columns)

# In[144]:

fi\_rf = pd.DataFrame({'feature': features,

'importance': rf.feature\_importances\_}).\

sort\_values('importance', ascending = False)

fi\_rf.head(20)

# In[145]:

import os

print('Current working directory path:',os.getcwd())

# In[ ]:

# # XGBoost

# In[80]:

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

# In[81]:

x2 = x1

y1 = y

x2 = StandardScaler().fit\_transform(x2)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x2, y1, test\_size=0.20, random\_state=102)

# In[82]:

XG = XGBClassifier()

XG.fit(x\_train,y\_train)

# In[83]:

print(XG)

# In[84]:

y\_pred = XG.predict(x\_test)

predictions = [round(value) for value in y\_pred]

# In[85]:

accuracy = accuracy\_score(y\_test, predictions)

print("Accuracy: %.2f%%" % (accuracy \* 100.0))

# In[86]:

cmXG = confusion\_matrix(y\_test, y\_pred)

plot\_confusion\_matrix(cm, classes = ['Good', 'Bad'],

title = 'Ouput Confusion Matrix')

# In[87]:

print(classification\_report(y\_test, y\_pred))

# In[ ]:

# In[ ]: