Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

Questions for Analysis:

1. Identify at least 3 indicators or predictors of potential risk from the variables provided. Please review the variables provided and determine which variables would be valuable to predict the MIS_Status class. Please provide a rationale for your decision as well as one quantitative method for confirming this. Please answer with at least 1 paragraph. (15 points)

Response:

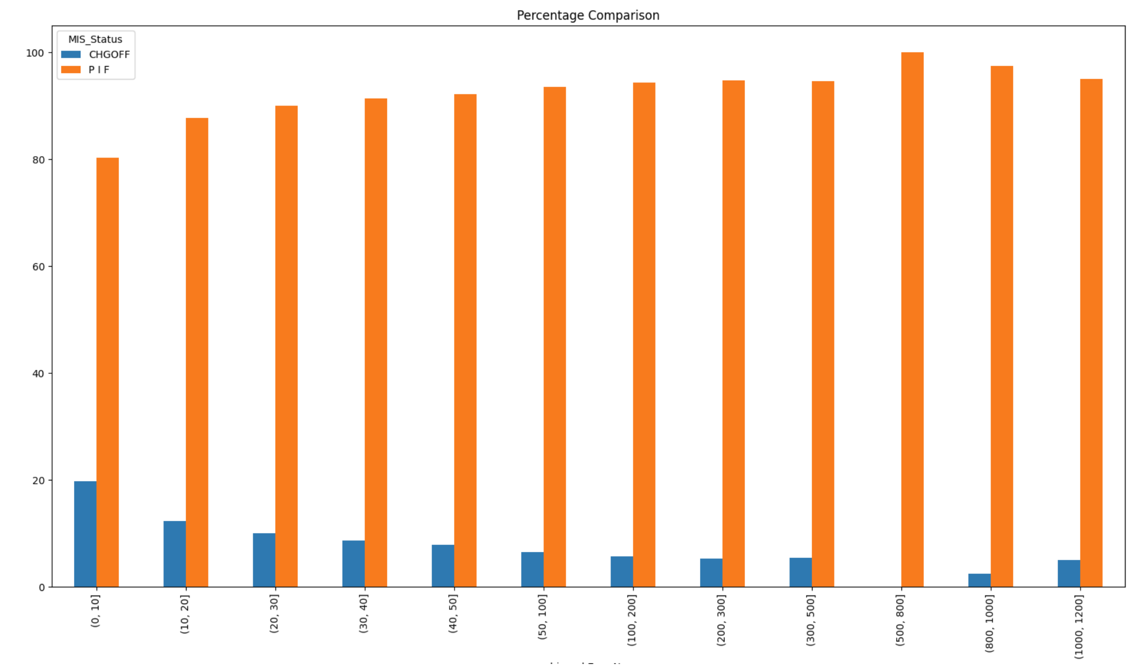
The top indicators appear to be the size of the company (Employee Count), the size of the loan (Gross Disbursement), the ratio of covered approval by the SBA and the state it was approved in. I believe that all four of these make logical sense, as those within a smaller company are more likely to fail, and especially those with 10 employees or less, there is just less revenue and less of a company propping up the loan repayment. This also goes with loan size, if a company can only afford to pay back a small loan in the first place, that seems like an apparent risk, that it is unlikely that they will turn $20,000 into an immense amount of profit, compared to say $500,000. Additionally, I assumed that certain states may just be less “friendly” to smaller companies, meaning that it is harder to get a product or service up off of the ground and running sustainably, so it makes sense that there are specific states the bank may want to avoid approving, though this might be the weakest of the four. Lastly, with SBA Approved amount, I made a ratio from Gross amount approved and SBA Approved to get the percentage that would be fully covered. It is logical to assume that the higher percentage of the loan the SBA is willing to cover, the less likely it is to default.

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

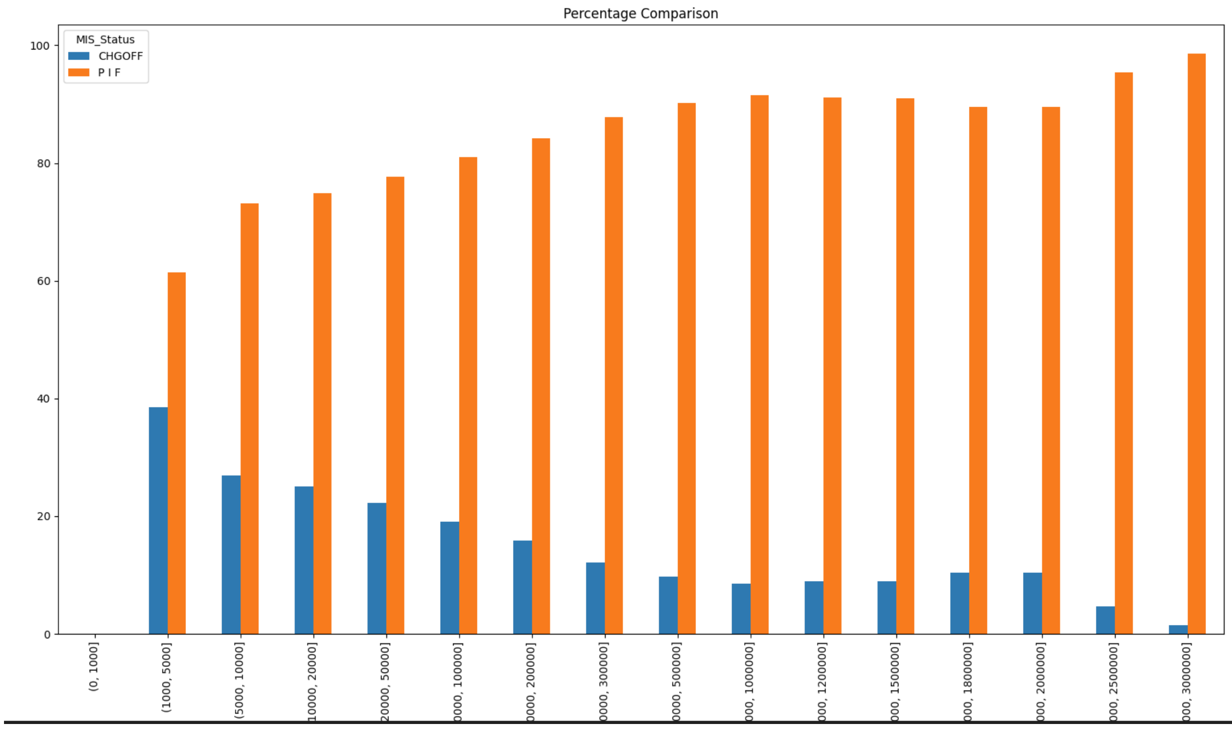
As can be seen by the attached visuals, there are some pretty drastic differences in some of these factors. Analyzing the amount of employees those with 10 or less have about a 20% Charge Off Rate, where as those with 50-100 employees have less than 10%. For size of the loan, those borrowing between $1,000 and $5,000 have almost a 40% Charge Off Rate, compared to those borrowing between $200,000 and $300,000 who have less than a 20% charge off rate. State was a surprising factor as well, with states like Florida and Georgia having an associated charge off rate of around 25%, where as states like Montanna and Wyoming have less than 10%. Lastly, for SBA Approval ratio, we can see that there are some jumping around of charge off percentage within the binned segments. However, when we just sum up the total amount of approved percentage between those that were and weren't charged off, the story becomes much clearer. This is one of the quantitative ways of verifying the associated impact of factors, but pulling out causation through model results will be more impressive, which I cover at the end of this paper.

Binned employee count:

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

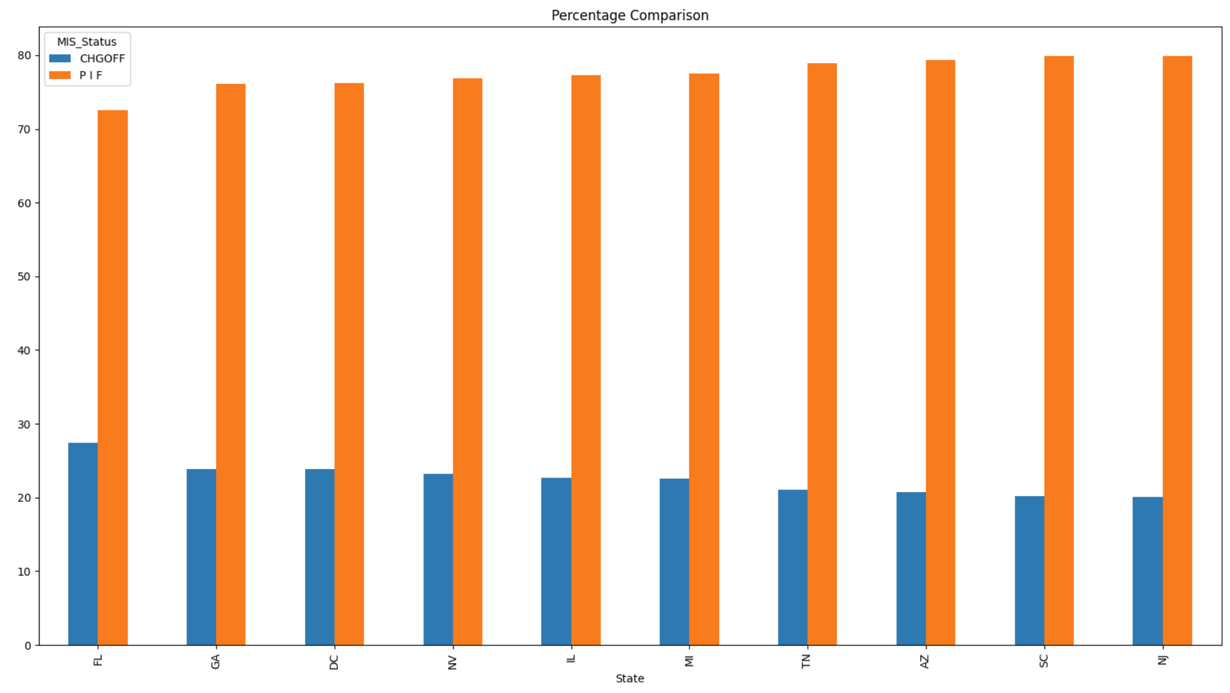


Binned Loan Size:

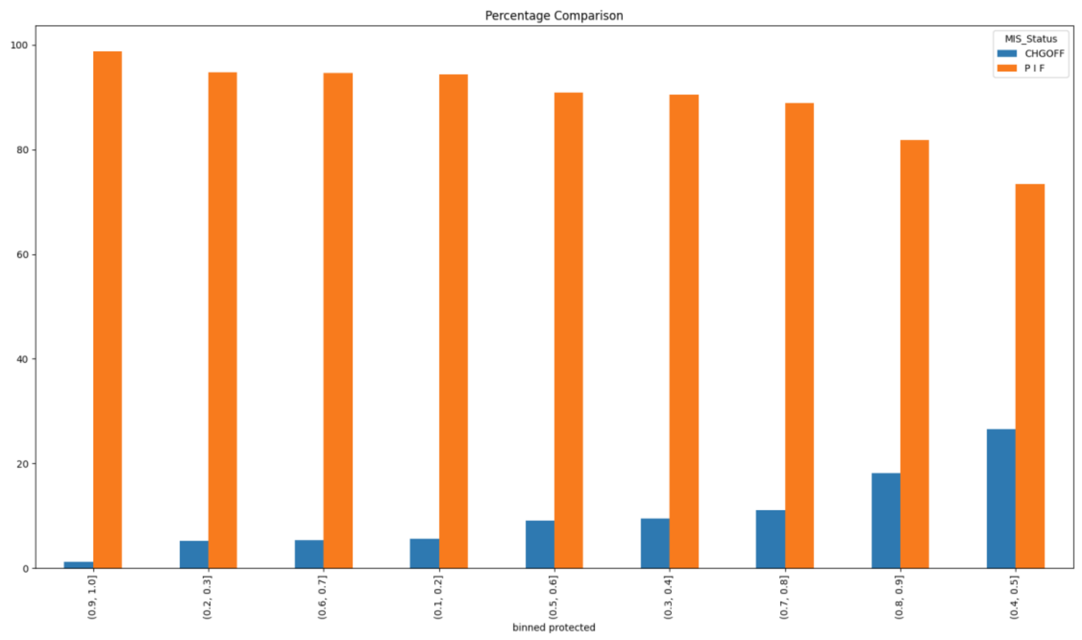


Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

Charge off by State: Top 10 states:

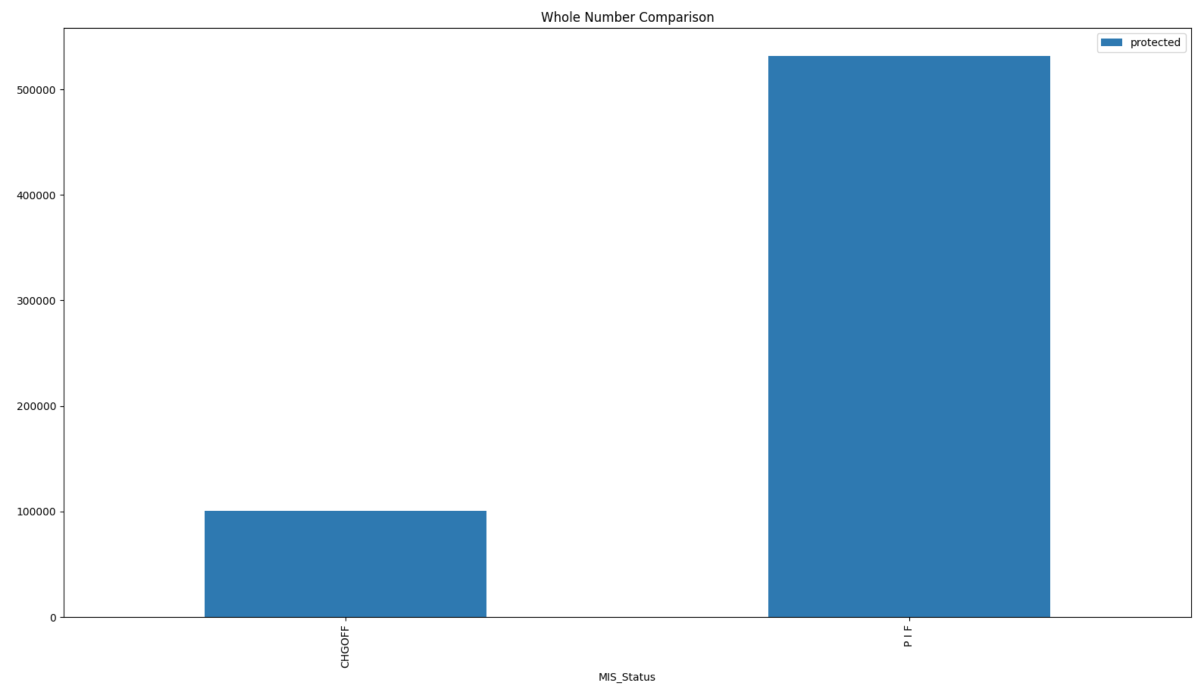


Protected Ratios Binned, with comparison of Charged Off Percents:



Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

Protected Ratios Summed between Paid In Full and Charged Off:



2. Identify any preprocessing and data cleaning that might need to be done to this data beforehand. You do not have a copy of the data so this is a projection of possible preprocessing or data cleaning that might be helpful. (15 points)

Response:

I would say that the missing values, data class imbalance and outliers are the largest risks for this data set. I will go ahead and use 'MIS_Status' as our binary variable of interest and drop Charge Off Date (Due to missing values and not being as useful to our prediction), drop Charged Off Amount (due to it being a data leak in the model, directly showing when a charge off does happen), drop Loan ID (not useful serial number), drop Revolving Line of Credit (Too many data inconsistencies and unidentified output) and will take the date factor and extract year and month. Additionally, the imbalance of loans that were charged off is pretty drastic, with only 156,041 needing the be Charged off and 734,598 being paid in full. To resolve this, I am going to use either SMOTE or ADASYN to improve the balance of our data set, lower the bias of our results and improve our overall metrics.

There a few numeric variables as well with extremely high skew, such as the total amount loaned, the amount SBA Approved and number of employees. To fix the skew that is present in the data set, I am going to use the Min-Max Scaler and help bring most of the skewed quantitative attributes into the range of 0-1, so that they have less influence over the model as a whole. As far as the dollar amounts in the data set, I will need to remove the currency sign, and adjust the format so they can be normalized. Additionally, I will need to do some label encoding to move the categorical objects into a numeric format so they can be included in the model as well. Lastly, I will do some Chi Square checking to see if there are any redundant variables and help keep those from interfering in the model. In this case after collecting month and year, I dropped the date variable and removed gross approved in favor of gross disbursed.

3. There are class imbalances, especially in the class we are trying to predict. Discuss the issues surrounding having class imbalances. Now discuss possible solutions to this class imbalance problem, with at least 2 paragraphs. (20 points)

Response:

When working with a data set that is seeking to do a classification prediction, if the majority of the classifications are negative, in this case ‘Paid In Full’, then it is more likely for the model to interpret the results as such, skewing them. Since the Charge Off occurrences are the important event, we are trying to classify and predict it is imperative that we make it easier for our model to identify this minority class, and be able to have enough information to make sound classification. We will be using Logistic, Decision Tree and Random Forest models, all of which will assume that there is no class imbalance in our data set and will assume that the inputs are equally distributed. Since the data set is initially 82% of one class, it is likely that without rebalancing the data, that our results would show around 82% accuracy, because the models are guessing the majority class, and getting it correct most of the time.

In addressing this issue there are options such as lowering the sample of the majority, or re sample the minority over and over to duplicate our set. However, this is a bit sloppy, and can cause more skew and bias than it will help to undo. So, I turned to data augmentation, with techniques such as SMOTE and ADASYN, known respectively as Synthetic Minority Oversampling Technique and Adaptive Synthetic. SMOTE focuses on finding the nearest neighbors in the N-Dimensional space that is of the minority class, and then plotting a new synthetic point at a decided ratio of the line between the neighboring minor class points. ADASYN is the improved version of this technique, which is very similar and just adds another small random amount to the point in the N-Dimensional space to make it more realistic, and slightly less contrived. I will be using ADASYN, which takes a very long time to run on our data set but will help me rebalance all of the X and Y points, which will then allow me to test-train split them with a healthy ratio of 0.8 train, 0.2 test.

4. At this point, the valuable features have been determined, the preprocessing and data cleaning completed and the class imbalances are handled. Now, we must choose a selection of 3 models to test and get results from. Identify these three models and explain in detail the process of using them as well as a brief discussion of the mathematical theory of each model you chose. Explain why these models were chosen. Consider issues around overfitting and underfitting as well as any hyperparameters that might need to be tuned. What solutions are fitting issues could you apply? You are not expected to have selected a perfect set of models and hyperparameters but rather to show critical thinking in your decisions. Please answer with a least 3 paragraphs. (40 points)

Response

After cleaning and balancing the data set, I proceeded to test a Decision Tree Model, Logarithmic Model and Random Forest Classifier Model. I chose to start with a decision tree model because it is something that can be visualized and should be easier to understand how it is making its decisions. I felt that the complexity of the dimensions would allow for some high-performance decision trees, especially since it is just a binary variable classification problem. I set it up with all of the default parameters, and allowed it to run to its fullest depth, due to the complexity of the data set. My initial results were not what I had hoped for, so I did a grid search with the splitting parameters and max depth, to try and decrease any over fitting that may have occurred. What I found was that some early stopping was actually required to help improve the trees performance. Instead of letting run its full depth of 52, I stopped it at 20, with the min_samples_leaf argument still at 1, and min_samples_split equal to 2. These parameters improve the performance by decreasing the overfitting of the model, which is good. The performance of the overall model I will discuss in length in the next section, but the cross-validation performance was still an average of 0.951 Accuracy. This type of performance from a 10 split cross validation was very satisfying for our final Decision Tree. The early stopping is key to prevent the over fitting, but I think a Random Forest model, which would not use every attribute each time in its massive set of trees, would perform even better while preventing over fitting.

Next, I utilized a Logistic Regression to try and classify the data set. I understood that it would perform best with a binary data set, which we have, and I also was interested in observing the coefficients of the different attributes, to help us better understand the causation impact power of each potential attribute. I utilized the Stochastic Average Gradient Adjusted because I had hoped it would converge more rapidly than normal SAG and would enable the model to remember the last sets of inputs/outputs to be able to adapt more rapidly. The first sets of trials with this model type didn’t converge at 1000 iterations, 4000 iterations, or 6000 iterations. It took 10,000 iterations for the model to converge, and when it did, its performance was considerably worse than our other models, so instead of taking a dozen hours to let grid search run, I just ran the cross validation overnight. As expected, the cross validation had an abysmal performance, without too much over fit present, indicating the model’s poor quality. The average accuracy score of the 10-fold cross validation was 0.64. This model was interesting to look at, and a good avenue to investigate, but ultimately isn’t what we would want the bank to rely on. Further tuning may improve the performance, but since we investigated both Random Forest and Decision Tree here, with great performance, I believe that XGBoost would be another algorithm that may have even better performance on this data set.

Due to the reasonable performance of the Decision Tree model, I felt that the balancing effect and thoroughness of a Random Forest Classifier would be another great step in the right direction of improving model performance. What is also interesting, is that it gives us a much better visual for Causation in the form of SHAP values. I set the Random Forest Classifier up so that it initially ran with the default parameters as well as n_estimators = 100. This provided reasonable results, but I performed a large grid search, which took almost 14 hours to run, and found the best parameters. 10 features, with 150 estimators provided the best results for our data set. With this in mind, I then took the new "best model" and ran a 10-fold cross validation test. The test output was accuracy and was useful in checking for over fitting. The Random Forest model is highly tuned, and running on a very complex data set, with complex parameters, so there is large risk for over fitting. As a result of the cross validation, I got an average accuracy score of 0.969, which was very pleasing, indicating that our model's general performance and cross validation performance isn't too heavily marred by over fitting. I will review the general performance in the next section, but I was pleased that the data normalization, ADASYN class balancing, and grid search tuning had resulted in a very reliable model.

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

5. Finally, discuss the metrics that you would use for evaluation. Explain your rationale behind using those metrics. Please answer with at least 1 paragraph. (10 points)

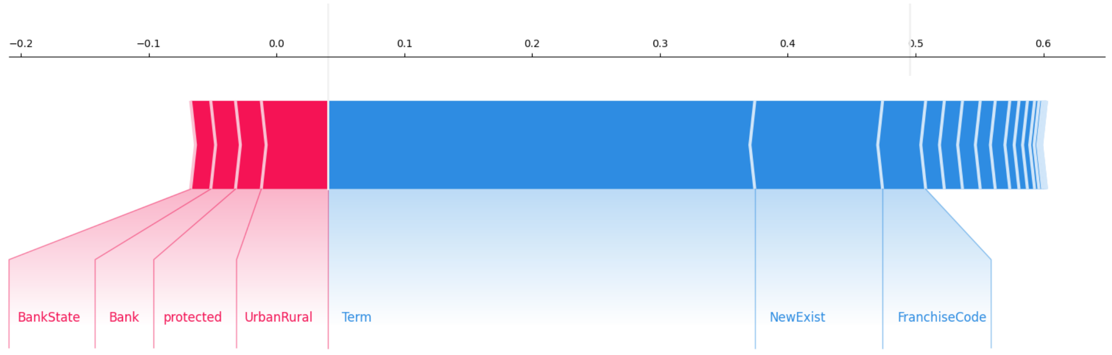
Response:

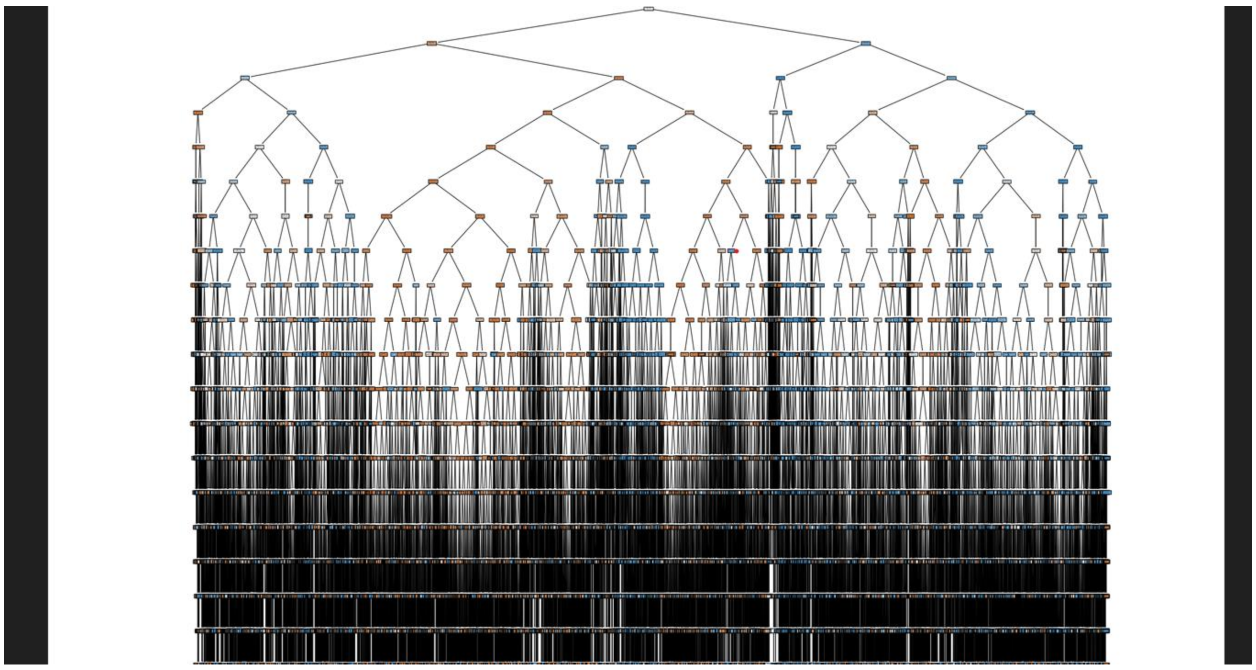
I would certainly say that both my Random Forest Model and Decision Tree are over fitting to an extent, but I feel it is justified by the main performance and cross validation scores. Additionally, since these models are so complex for a binary classification problem, there is bound to be some over fitting, even with the reduced attribute utilization. For each model I used R-Squared, Precision, Accuracy and F1 score. I could have also used ROC, MAE or MAPE, but felt that the metrics I had were substantial enough to paint the picture of my models' abilities and adding more metrics might muddy the water. I utilized R-Squared because I feel that it is so important to know what portion of your data your model is able to explain for, as I have had models in the past with really good performance, but with little model explanation. R-Squared is different from the R score in the sense that it is the direct proportion of the variation that is explained by the model, compared to just using the average. So if we have an 80% R-Squared that means that there is 80% less variation apart from the model, than just drawing a line at the average. The utilization of Precision and Accuracy is so that I can check the different aspects of a confusion matrix. I am thinking about this scenario from the perspective that we would want to avoid as many False Positives (saying someone will default when they won't), so that the bank can be as fair as possible. However, I do understand that Accuracy and Precision tend to give a better picture to the "True" values and neglect "False" results.

There is certainly an argument for prioritizing False Negatives (saying someone is risk free, when they will default) that would be in favor of the bank losing as little money as possible but might sacrifice the lenders fairness. So, Accuracy checks to see the ratio of all truly classified items, over all items and Precision checks to see the ratio of Truly Positive items over all positive (false and true). These would help me to identify bias in the models and help paint a picture of how "fair" these models would be for the clients of the business. Lastly, my utilization of the F1 Score was to help further with potential class imbalances (even after the correction of ADASYN), and also to give an ear to the incorrectly predicted classes, as it gives a standardized mean of Precision and Recall. With all of these metrics included, I feel that my models are able to express their performance, and I am able to justify my conclusions.

Reviewing the Performances, for the Decision Tree Model I had a satisfying performance after grid search parameters tuning. The R-Squared came up from a .79 to a .813 after tuning, with precision and accuracy rising from .947 to .958 and 953 respectively. The F1_score arrived at 0.953 as well after tuning. This performance increase was met with slight overfitting found in the cross validation which performed at 0.952 average accuracy. These results are still very reasonable for our model, and I anticipate that this would be satisfying for the client in trying to protect their interests and still be fair to the customers. However, this performance was bested by the performance of the Random Forest model. Before arriving there, we must acknowledge the poor performance of the Logistic Regression first. Despite attempts at iteration increases, the models r-squared was negative (-0.407) and its accuracy and precision were at 0.64 and 0.67 respectively with an F1 Score of 0.609. An abysmal model, which kept this performance at the cross-validation check, with an average accuracy of 0.644**.** This indicates that the model is having issues with both false negatives and false positives. It is very likely that it is almost only classifying one type of our binary output well. Instead of tuning further, the Random Forest Classification model is what I focused on.

With the Random Forest Model, we had a reasonable initial performance, that was improved through grid search. With the final parameters of 10 max features and 150 estimators, the r-squared settled at 0.884, up from 0.873, with precision and accuracy settling at 0.972 each, up from 0.968 each. The F1 Score was at 0.971 after tuning, displaying this models ability to perform well with true positive and true negatives, while minimizing false positives and false negatives. The cross validation confirmed this, with an average accuracy score of 0.969, which I am very pleased with. I believe that it is also worth mentioning the Shapley values and the Tree print from the decision tree. The SHAP visual informs us of the Causation impact of the state, term of the loan, and age of the company, along with the ratio of protected funds from the SBA. These visual measures this impact, which confirms our initial findings from the 2-dimensional visual analysis we did at the beginning. The Decision tree is a mess to look at but is a stunning visualization of the complexity the model had to pursue to gain such performance. Please find both visuals below.





The Shap values show us that Bank State, Bank, protected ratio, Urban Rural all have some variety of impact that is negative. This makes sense for the others, but for Protected Ratio, I imagine that if I were to bin it into groups like 0-30%, 30-60% and 60-100%, we would lose some of our model performance (due to over generalization), but that it would be possible to see which of the ratios negatively impact default and which ratios indicate a lack of likelihood for default. Additionally, using LIME values to attribute the thresholds of impact would be interesting as well. However, causation analysis isn't the driving factor of this assignment, so that is something to look at another time.

As far as concluding thoughts go, I felt that the Random Forest outperformed both the Decision Tree and Logistic Regression, with the latter being abysmal in comparison. I am still curious how a fully tuned XGBoost would do with this data set, and if it might be able to outperform the Random Forest. Beyond this curiosity, I feel that I have thoroughly investigated the data set, visualized, and scrubbed it, then worked on thorough model development. This has resulted in a set of models, with the Random Forest being the highest recommended model for usage by the client. This model should greatly aid the lender with discerning the highest risk customers to avoid, while also not neglecting those that might be classified as a False Negative.

Code Appendix:

Data Cleaning:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import decimal
import seaborn as sns
import warnings
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
from scipy.stats import chi2_contingency
import copy

warnings.simplefilter(action='ignore', category=FutureWarning)


low_memory = False

path = 'C:/Users/jorda/OneDrive/Desktop/PyCharm Community Edition
2021.2.2/5500 Capstone/Mid Term/Raw ' \
       'Data/'
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

```
raw_data_large= pd.read_csv(path + 'SBAnational.csv')

raw_data_large.describe()


# imports and read ins run as normal

raw_data_large.columns
# Index(['LoanNr_ChkDgt', 'Name', 'City', 'State', 'Zip', 'Bank',
'BankState',
#       'NAICS', 'ApprovalDate', 'ApprovalFY', 'Term', 'NoEmp', 'NewExist',
#      'CreateJob', 'RetainedJob', 'FranchiseCode', 'UrbanRural', 'RevLineCr',
#    'LowDoc', 'ChgOffDate', 'DisbursementDate', 'DisbursementGross',
#   'BalanceGross', 'MIS_Status', 'ChgOffPrinGr', 'GrAppv', 'SBA_Appv'],



##INITIAL CHANGES:
raw_data_large['ApprovalDate_Num'] =
pd.to_datetime(raw_data_large['ApprovalDate']).dt.date
raw_data_large['ApprovalYear'] =
pd.to_datetime(raw_data_large['ApprovalDate_Num']).dt.year
raw_data_large['ApprovalMonth'] =
pd.to_datetime(raw_data_large['ApprovalDate_Num']).dt.month
raw_data_large['DisbursementDate'] =
pd.to_datetime(raw_data_large['DisbursementDate']).dt.date
raw_data_large['ChgOffDate'] =
pd.to_datetime(raw_data_large['ChgOffDate']).dt.date
raw_data_large['NoEmp_num'] = raw_data_large['NoEmp'].values





###################################################
##### DATA MODIFICATION BEGINS #####
###################################################

### Missing data
raw_data_large.isna().sum()
# Name                        14
# City                        30
# State                       14
# Bank                      1559
# BankState                 1566
# NewExist                   136
# RevLineCr                 4528
# LowDoc                    2582
# ChgOffDate              736465 # just needs to be filled in as not having one!
# DisbursementDate          2368
# MIS_Status                1997

########################
# impute or remove NAs
########################
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

```
clean_v1 = copy.deepcopy(raw_data_large)
clean_v1['ChgOffDate'] = raw_data_large['ChgOffDate'].fillna(0)
clean_v1['ChgOffDate'] = (clean_v1['ChgOffDate']!= 0).astype(int)

# dropping due to ubundance of un identified signal
clean_v1 = clean_v1.drop('RevLineCr', axis=1)


# drop empty disbursement dates and MIS_status and LowDoc
clean_v2 = clean_v1.copy()
# clean_v2.dropna(subset=['BankState', 'MIS_Status', 'LowDoc']) # not
necessary?
clean_v2 = clean_v2.dropna()
clean_v2.isna().sum()
# looks good!

# strip off dollar none of these are working lol
# will need to do this for 'BalanceGross', 'ChgOffPrinGr', 'GrAppv' and
'SBA_Appv'
print(clean_v2.dtypes)
clean_v2['BalanceGross'] = (clean_v2['BalanceGross'].str.strip('$'))
clean_v2['BalanceGross'] = clean_v2['BalanceGross'].str.replace(',',
'').astype(float).astype(int)
clean_v2['BalanceGross'] = pd.to_numeric(clean_v2['BalanceGross'])  # finally
works!

clean_v2['ChgOffPrinGr'] = (clean_v2['ChgOffPrinGr'].str.strip('$'))
clean_v2['ChgOffPrinGr'] = clean_v2['ChgOffPrinGr'].str.replace(',',
'').astype(float).astype(int)
clean_v2['ChgOffPrinGr'] = pd.to_numeric(clean_v2['ChgOffPrinGr'])

clean_v2['GrAppv'] = (clean_v2['GrAppv'].str.strip('$'))
clean_v2['GrAppv'] = clean_v2['GrAppv'].str.replace(',',
'').astype(float).astype(int)
clean_v2['GrAppv'] = pd.to_numeric(clean_v2['GrAppv'])

clean_v2['SBA_Appv'] = (clean_v2['SBA_Appv'].str.strip('$'))
clean_v2['SBA_Appv'] = clean_v2['SBA_Appv'].str.replace(',',
'').astype(float).astype(int)
clean_v2['SBA_Appv'] = pd.to_numeric(clean_v2['SBA_Appv'])

clean_v2['DisbursementGross'] =
(clean_v2['DisbursementGross'].str.strip('$'))
clean_v2['DisbursementGross'] =
clean_v2['DisbursementGross'].str.replace(',', '').astype(float).astype(int)
clean_v2['DisbursementGross'] = pd.to_numeric(clean_v2['DisbursementGross'])

# FEATURE ENGINEER: COVERED RATIO:
clean_v2['protected']= (clean_v2['SBA_Appv'] / clean_v2['GrAppv'])

# SAVE FOR VISUALIZATION:
clean_v2.to_csv(path+'data_v2.csv')

# CHECK FOR OUTLIERS:
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

```
plt.boxplot(clean_v2[['BalanceGross', 'ChgOffPrinGr', 'GrAppv', 'SBA_Appv']])
plt.xticks([1, 2, 3, 4], ['Gross Balance', 'Charge off Amount', 'Gross Amount
Approved', 'SBA Gauranteed Amount'])
plt.show() # works well, heavy skew!


print(max(clean_v2['NoEmp_num']))
print(max(clean_v2['Term']))
plt.boxplot(clean_v2[['DisbursementGross', 'Term', 'NoEmp_num']])
plt.xticks([1, 2, 3], ['Disbursement Amount', 'Term of Loan', 'Number of
Employees'])
plt.show() # disbursement amount and emp num are heavily skewed, Term is fine


## DATA NORMALIZATION (FOR SKEW): MIN MAX
# create a scaler object
scaler = MinMaxScaler()
# fit and transform the data
clean_v2= clean_v2.reset_index(drop=True)
clean_v3 = clean_v2.copy()

num_cols = clean_v2.select_dtypes(include=np.number).columns.tolist()
num_cols.remove('ApprovalYear') # removing non scale items
num_cols.remove('protected')

clean_v3[num_cols] = pd.DataFrame(scaler.fit_transform(clean_v3[num_cols]),
columns=clean_v3[
    num_cols].columns)
clean_v3.isna().sum() # this was the issue
print(clean_v3.head(3))
#worked!




#Look at data imbalance: USE ADASIGN or Oversample or Undersample?
#https://www.kaggle.com/code/residentmario/oversampling-with-smote-and-
adasyn/notebook

print(clean_v3['MIS_Status'].value_counts())
#P I F     734598
#CHGOFF    156041
# we will want to use adasyn when we are training our model!



# label encode:
labelencoder = LabelEncoder()

all_cols = clean_v3.dtypes

clean_v4 = clean_v3.copy()

# is there a better way to do this?
clean_v4['ApprovalDate_Num'] =
labelencoder.fit_transform(clean_v3['ApprovalDate_Num'])
clean_v4['Bank'] = labelencoder.fit_transform(clean_v3['Bank'])
clean_v4['BankState'] = labelencoder.fit_transform(clean_v3['BankState'])
clean_v4['City'] = labelencoder.fit_transform(clean_v3['City'])
clean_v4['LowDoc'] = labelencoder.fit_transform(clean_v3['LowDoc'])
clean_v4['MIS_Status'] = labelencoder.fit_transform(clean_v3['MIS_Status'])

# all others work perfect!

# select final desired columns: Chi Squared
# Dont Need: DisbursementDate, State, Name (very miscelanious)


cross = pd.crosstab(clean_v4['NoEmp'], clean_v4['NewExist'])
(chi, p_val, d_f, exp_val) = stats.chi2_contingency(cross)
print('chi val:', chi, 'p_val:', p_val, 'Degrees of Freedom:', d_f, 'expected
val:', exp_val)
# very high chi: chi val: 60628.066960848424

cross = pd.crosstab(clean_v4['DisbursementGross'], clean_v4['BalanceGross'])
(chi, p_val, d_f, exp_val) = stats.chi2_contingency(cross)
print('chi val:', chi, 'p_val:', p_val, 'Degrees of Freedom:', d_f, 'expected
val:', exp_val)
# very high chi: chi val: 2377087

cross = pd.crosstab(clean_v4['ApprovalDate_Num'], clean_v4['ApprovalYear'])
(chi, p_val, d_f, exp_val) = stats.chi2_contingency(cross)
print('chi val:', chi, 'p_val:', p_val, 'Degrees of Freedom:', d_f, 'expected
val:', exp_val)
# very high chi: 40767040

cross = pd.crosstab(clean_v4['CreateJob'], clean_v4['RetainedJob'])
(chi, p_val, d_f, exp_val) = stats.chi2_contingency(cross)
print('chi val:', chi, 'p_val:', p_val, 'Degrees of Freedom:', d_f, 'expected
val:', exp_val)
# very high chi: 13180830

# I will drop date_num and gross approved

clean_v5 = clean_v4[['LoanNr_ChkDgt', 'City', 'Zip', 'Bank', 'BankState',
'NAICS',
                     'ApprovalYear', 'ApprovalMonth', 'Term', 'NoEmp',
'NewExist', 'CreateJob',
                    'RetainedJob', 'FranchiseCode', 'UrbanRural', 'LowDoc',
'DisbursementGross',
                     'BalanceGross','MIS_Status','SBA_Appv','ApprovalYear',
'protected']]

# took out 'ChgOffDate',  for now
clean_v5.to_csv(path+'data_final.csv')
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

Model Testing:

Decision Tree:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import decimal
import seaborn as sns
import warnings
import shap
from sklearn.metrics import r2_score, accuracy_score, precision_score,
f1_score
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import ADASYN
from sklearn.model_selection import cross_val_score

low_memory = False
warnings.simplefilter(action='ignore', category=FutureWarning)

path = 'C:/Users/jorda/OneDrive/Desktop/PyCharm Community Edition
2021.2.2/5500 Capstone/Mid Term/Raw ' \
       'Data/'

## First Model Decision Tree! ##
data_final = pd.read_csv(path + 'data_final.csv')
data_final.dtypes


X = data_final[['LoanNr_ChkDgt', 'City', 'Zip', 'Bank', 'BankState', 'NAICS',
                     'ApprovalYear', 'ApprovalMonth', 'Term', 'NoEmp',
'NewExist', 'CreateJob',
                    'RetainedJob', 'FranchiseCode', 'UrbanRural', 'LowDoc',
'DisbursementGross',
                     'BalanceGross','SBA_Appv','ApprovalYear', 'protected']]


y = data_final['MIS_Status']

######################
#ADASYN ADJUSTMENT! RE BALANCING DATA:
#https://www.kaggle.com/code/drscarlat/fraud-detection-under-oversampling-
smote-adasyn

adasyn = ADASYN()
X_adjusted, y_adjusted = adasyn.fit_resample(X,y)
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

```
print('ready')
#
X_train, X_test, y_train, y_test = train_test_split(X_adjusted, y_adjusted,
test_size= 0.2, random_state=
21)

###################
#DECISION TREE
##################
clf = DecisionTreeClassifier(max_depth= 20, min_samples_leaf= 1,
min_samples_split= 2)
clf1 = clf.fit(X_train, y_train)
pred1 = clf1.predict(X_test)

print('r2', r2_score(y_test, pred1))
print('Accuracy', accuracy_score(y_test, pred1))
print('Precision', precision_score(y_test, pred1))
print('F1 Score', f1_score(y_test, pred1))
#r2 0.7901012137794616
#Accuracy 0.9475311034782637
#Precision 0.9475346540056756
# F1 Score 0.9469390258863943
# Wow, somewhat ok first try.

########################
# Grid Search, Decision Tree
########################
p = {'max_depth': [5, 10, 20, 30, 40, 50, 60],
         'min_samples_split': [2,3,4],
         'min_samples_leaf': [1,2]}


clf1_gs = DecisionTreeClassifier()
grid = GridSearchCV(estimator=clf1_gs, param_grid=p)
grid.fit(X_train, y_train)
print('ready')
print(grid.best_params_) # {'max_depth': 20, 'min_samples_leaf': 1,
'min_samples_split': 2}
best_model = grid.best_estimator_
best_model.fit(X_train,y_train)
pred_grid = best_model.predict(X_test)
print('r2', r2_score(y_test, pred_grid))
print('Accuracy', accuracy_score(y_test, pred_grid))
print('Precision', precision_score(y_test, pred_grid))
print('F1 Score', f1_score(y_test, pred1))
#r2 0.8133888409028633
#Accuracy 0.9533523667631821
#Precision 0.9583054657280168
# Very good results!

########################
# Cross Validation
#########################

tree_scores = cross_val_score(best_model, X_train, y_train, scoring =
"accuracy", cv=10)

print('Scores', tree_scores)
print('Mean', tree_scores.mean()) #
print('SD', tree_scores.std())
print('ready')
#Scores [0.95195038 0.95308761 0.95181515 0.9515792  0.95152863 0.95219436
 #0.95263255 0.95290221 0.95243874 0.95235447]
#Mean 0.9522483298358588
#SD 0.0005071618719732867
# Very reasonable! I am happy with that




fig1 = plt.figure(figsize=(25,20))
tree.plot_tree(best_model, feature_names=['LoanNr_ChkDgt', 'City', 'Zip',
'Bank', 'BankState', 'NAICS',
                    'ApprovalYear', 'ApprovalMonth', 'Term', 'NoEmp',
'NewExist', 'CreateJob',
                   'RetainedJob', 'FranchiseCode', 'UrbanRural', 'LowDoc',
'DisbursementGross',
                    'BalanceGross','SBA_Appv','ApprovalYear', 'protected',
'MIS_Status'],
                   class_names=['P I F', 'CHGOFF'],
                   filled=True) # works! completed on 1.26
```

Logistic Regression:

```
####################
#Logistic Regression
####################
#May need to reduce number of variables observed?
#ss = StandardScaler()
#ss_X = ss.fit_transform(X_adjusted)

#X_train_ss, X_test_ss, y_train, y_test = train_test_split(ss_X, y_adjusted,
test_size= 0.2,
 #                                                   random_state=21)

log_reg = LogisticRegression(max_iter=10000, solver='saga')
clf2 = log_reg.fit(X_train, y_train)
pred2 = clf2.predict(X_test)

print('r2', r2_score(y_test, pred2))
print('Accuracy', accuracy_score(y_test, pred_grid))
print('Precision', precision_score(y_test, pred_grid))
print('F1_score', f1_score(y_test, pred2))
#r2 r2 -0.4071821749216562
#Accuracy 0.6482433402658163
#Precision 0.6760721223290332
# F1_score 0.60950164273045
# not great
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

```
########################
#Cross Validation
#######################
log_scores = cross_val_score(best_model, X_train, y_train, scoring =
"accuracy", cv=10)


print('Scores', log_scores)
print('Mean', log_scores.mean()) #
print('SD', log_scores.std())
print('ready')
#Scores [0.64183569 0.64248997 0.64614723 0.64504331 0.64543938 0.64415849
# 0.64350962 0.64348434 0.64345064 0.64298716]
#Mean 0.6438545843566131
#SD 0.0012799615431548315
```

Random Forest:

```
#########################
#Random Forest Classifier
#########################
clf_rf = RandomForestClassifier()
clf_rf.fit(X_train, y_train)

# performing predictions on the test dataset
predictionrf= clf_rf.predict(X_test)
print('ready')

print('r2', r2_score(y_test, predictionrf))
print('Accuracy', accuracy_score(y_test, predictionrf))
print('Precision', precision_score(y_test, predictionrf))
print('F1 score', f1_score(y_test, predictionrf))
#r2 0.8738395834588618
#Accuracy 0.9684633819955439
#Precision 0.9654707548064547
# F1 score
# Very reasonable results


#########################
# Grid Search, Random Forest
#########################

param_grid = [
    {'n_estimators': [3, 10, 30, 50, 100, 150], 'max_features':[2, 4, 6, 8,
10]}
]

forest_class2 = RandomForestClassifier()
grid_search = GridSearchCV(forest_class2, param_grid, cv=5,
                           scoring="neg_mean_squared_error",
                           return_train_score=True)
```

Jordan Gropper
Machine Learning Process: Loan Approval
1/26/23

```
grid_search.fit(X_train, y_train)
#the below takes a long time to run
grid_search.best_params_ # best parameters!

best_model = grid_search.best_estimator_
best_model.fit(X_train,y_train)
pred_gridrf = best_model.predict(X_test)
print('r2', r2_score(y_test, pred_gridrf))
print('Accuracy', accuracy_score(y_test, pred_gridrf))
print('Precision', precision_score(y_test, pred_gridrf))
print('F1 score', f1_score(y_test, pred_gridrf))
#r2 0.8847619795040009
#Accuracy 0.97119367919345
#Precision 0.9718971473829389 # slightly better!
# F1 score 0.9707459227321529
 #INPUTS:
 #{'max_features': 10, 'n_estimators': 150}


########################
# Cross Validation
#########################

forest_scores = cross_val_score(clf_rf, X_train, y_train, scoring =
"accuracy", cv=10)

print('Scores', forest_scores)
print('Mean', forest_scores.mean()) #
print('SD', forest_scores.std())
print('ready')
#Scores [0.97024497 0.96986551 0.96966326 0.96947787 0.9698318  0.9696127
# 0.97028685 0.97047224 0.96984865 0.97007618]
#Mean 0.9699380032991479
#SD 0.00030626278551882405




###########################
# SHAP VALUE EXTRACTION
############################


# Attempt on full set WORKING:
group = X_test.iloc[880]
shp = shap.Explainer(clf_rf)
shap_values = shp.shap_values(group)
print(shp.expected_value, shap_values)
shap.initjs()
shap_plot = shap.force_plot(shp.expected_value[1], shap_values[1],
matplotlib=True,
                feature_names=X_test.columns[0:22],
                show=True, plot_cmap=['#77dd77', '#f99191'])
```