Star Classification Project (Github)

CS 4641 - Fall 2024 - Group 58

Introduction/Background

Stellar classification is used by scientists to categorize stars based on their spectral characteristics (temperature, size, composition, color, brightness, etc.). By classifying these stars, scientists can better understand them by analyzing their patterns and trends, which will provide us with more knowledge and insight into the universe.

Literature Review:

- Armstrong et al. utilized a combination of Kohonen Self-Organizing Maps (SOMs) and Random Forest as a new method for variable star classification [1]. SOMs were used to effectively parameterize light curve shapes while Random Forest were useful for their classification schemes, especially for larger data sets [1].
- Zhang et al. focused on improving Guide Star Catalogues (GSCs) used in star identification on star sensors in satellites [2]. Zhang et al. evaluated the performance of GSCs generated by various machine learning classification algorithms [2]. K-Nearest Neighbours (KNN) produced the best GSC [2].
- Qi classified celestial bodies into stars, galaxies, and quasars using Decision Tree, Random Forest, and SVM models [3]. SMOTE, normalization, and data splitting were used for preprocessing [3]. Random Forest had the best computing performance and accuracy [3].

Dataset:

This Star Dataset uses spectral data to distinguish whether a star is a dwarf or giant.

- The features are visual apparent magnitude, distance between star and earth, standard error of the distance, spectral type, absolute magnitude, and the target class (0 - Dwarf, 1- Giant)
- The dataset has either 9,999/99,999 rows of raw data or 3642/39552 rows of preprocessed data. We can choose the type and how many data points we want.

Problem Definition

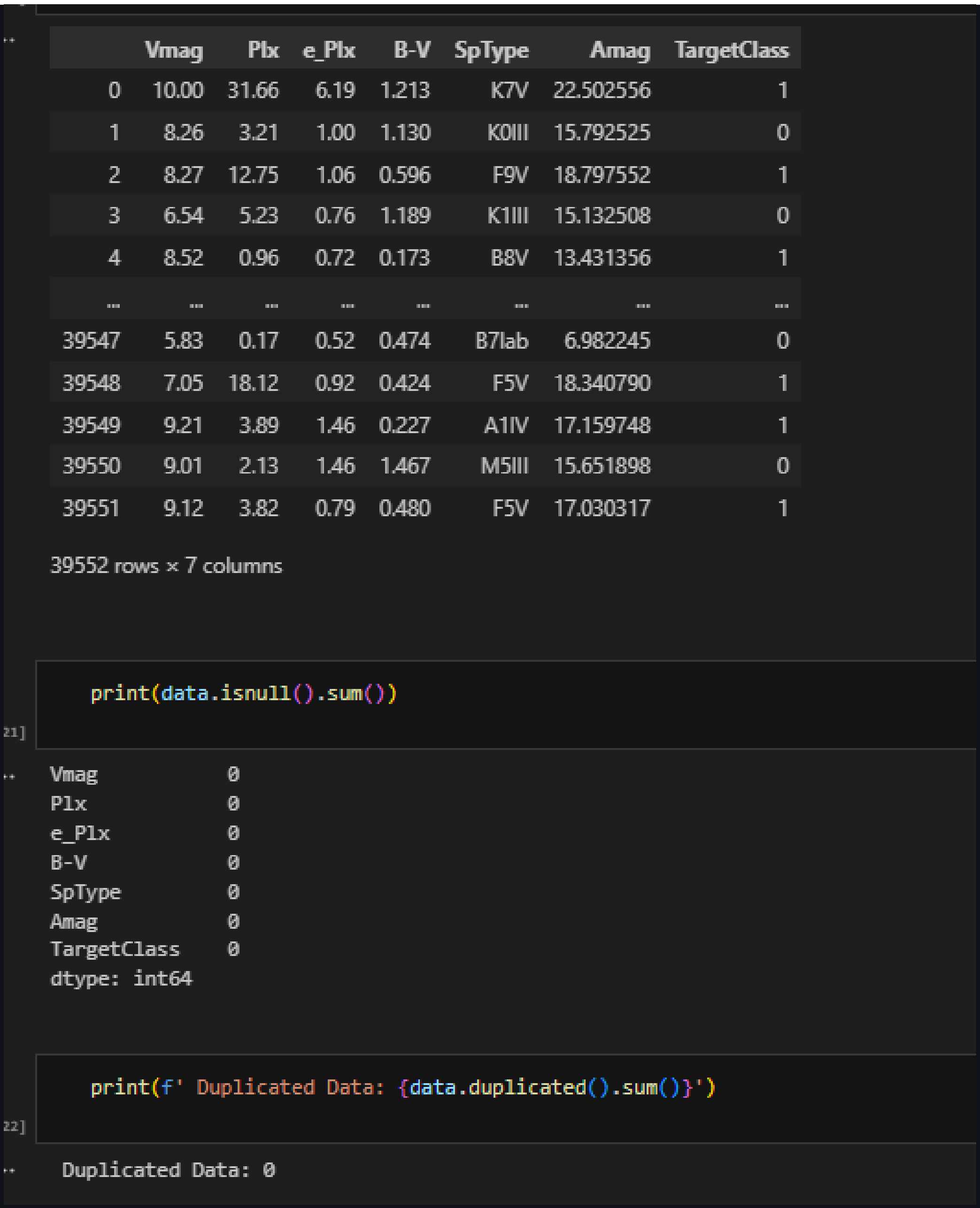
Given that technology is advancing, more data on stellar bodies will become available. Instead of manually categorizing the data everytime, we can automate the process to make it more consistent and efficient with machine learning. Our problem is a binary classification; we will classify whether a star is dwarf or giant using our ML models.

Methods

Preprocessing methods:

Our dataset was already preprocessed and pre-balanced.

We still checked to see if there were any null or duplicated data in the dataset so that we could remove them, but we got zero for both. We also tried to check and remove outliers in our dataset by visualizing the features with box plots. Our attempt to remove outliers was by using the IQR method, which can be seen in the code below:



	Vmag	Plx	e_Plx	B-V	SpType	Amag	TargetClass
0	10.00	31.66	6.19	1.213	K7V	22.502556	1
1	8.26	3.21	1.00	1.130	K0III	15.792525	0
2	8.27	12.75	1.06	0.596	F9V	18.797552	1
3	6.54	5.23	0.76	1.189	K1III	15.132508	0
4	8.52	0.96	0.72	0.173	B8V	13.431356	1
...	...	...	...	...	...	...	...
39547	5.83	0.17	0.52	0.474	B7Iab	6.982245	0
39548	7.05	18.12	0.92	0.424	F5V	18.340790	1
39549	9.21	3.89	1.46	0.227	A1IV	17.159748	1
39550	9.01	2.13	1.46	1.467	M5III	15.651898	0
39551	9.12	3.82	0.79	0.480	F5V	17.030317	1

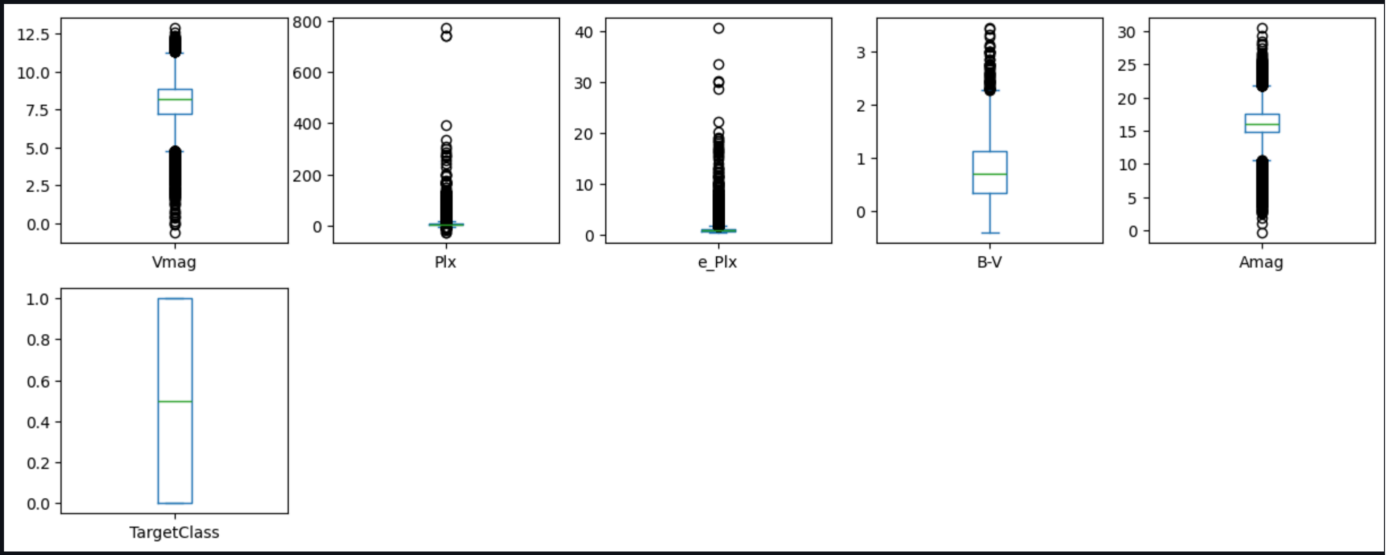
39552 rows × 7 columns

```
print(data.isnull().sum())
```

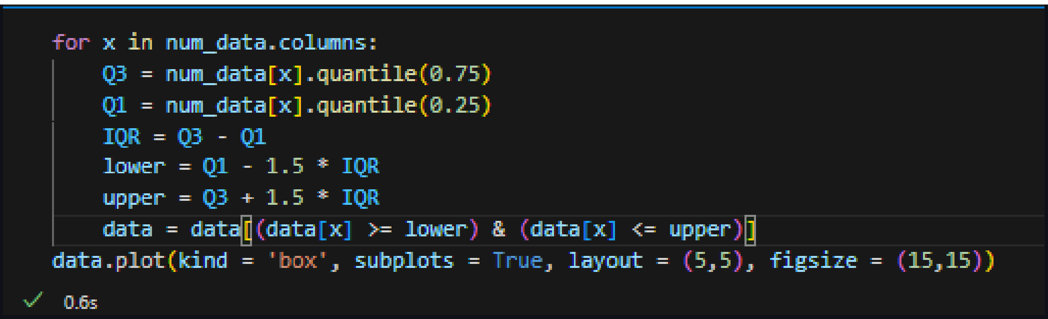
```
Vmag           0
Plx            0
e_Plx          0
B-V            0
SpType         0
Amag           0
TargetClass    0
dtype: int64
```

```
print(f' Duplicated Data: {data.duplicated().sum()}')
```

```
 Duplicated Data: 0
```



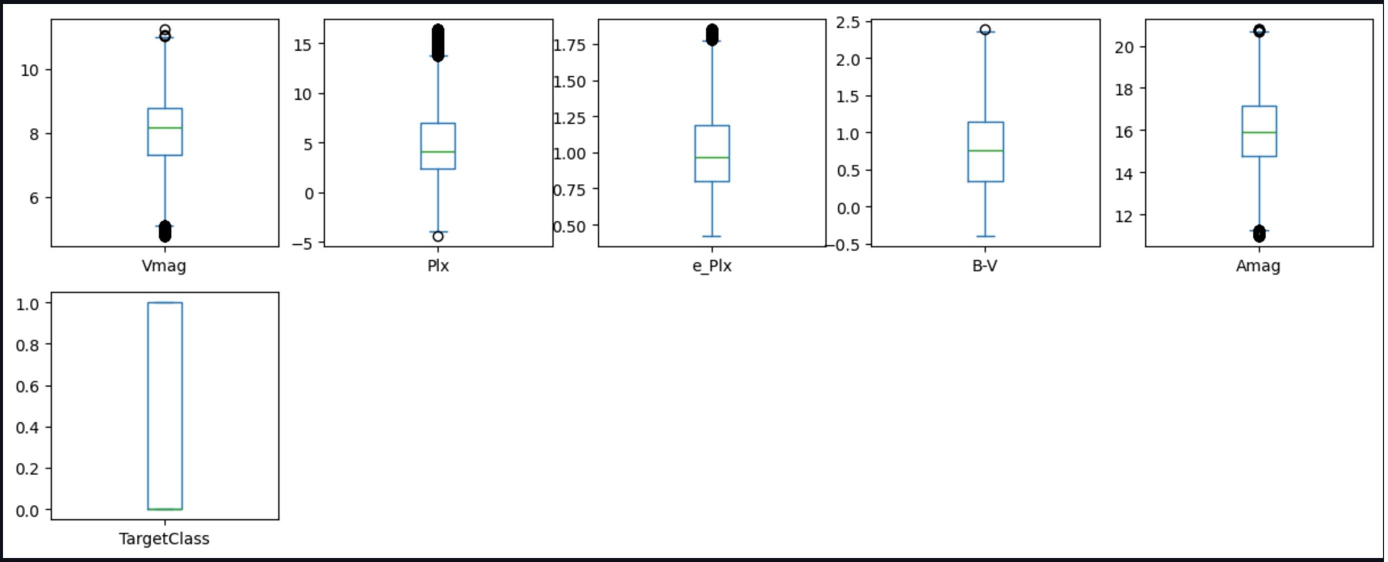
Box plot of the features in the dataset



```
for x in num_data.columns:
    Q3 = num_data[x].quantile(0.75)
    Q1 = num_data[x].quantile(0.25)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    data = data[(data[x] >= lower) & (data[x] <= upper)]
data.plot(kind = 'box', subplots = True, layout = (5,5), figsize = (15,15))
```

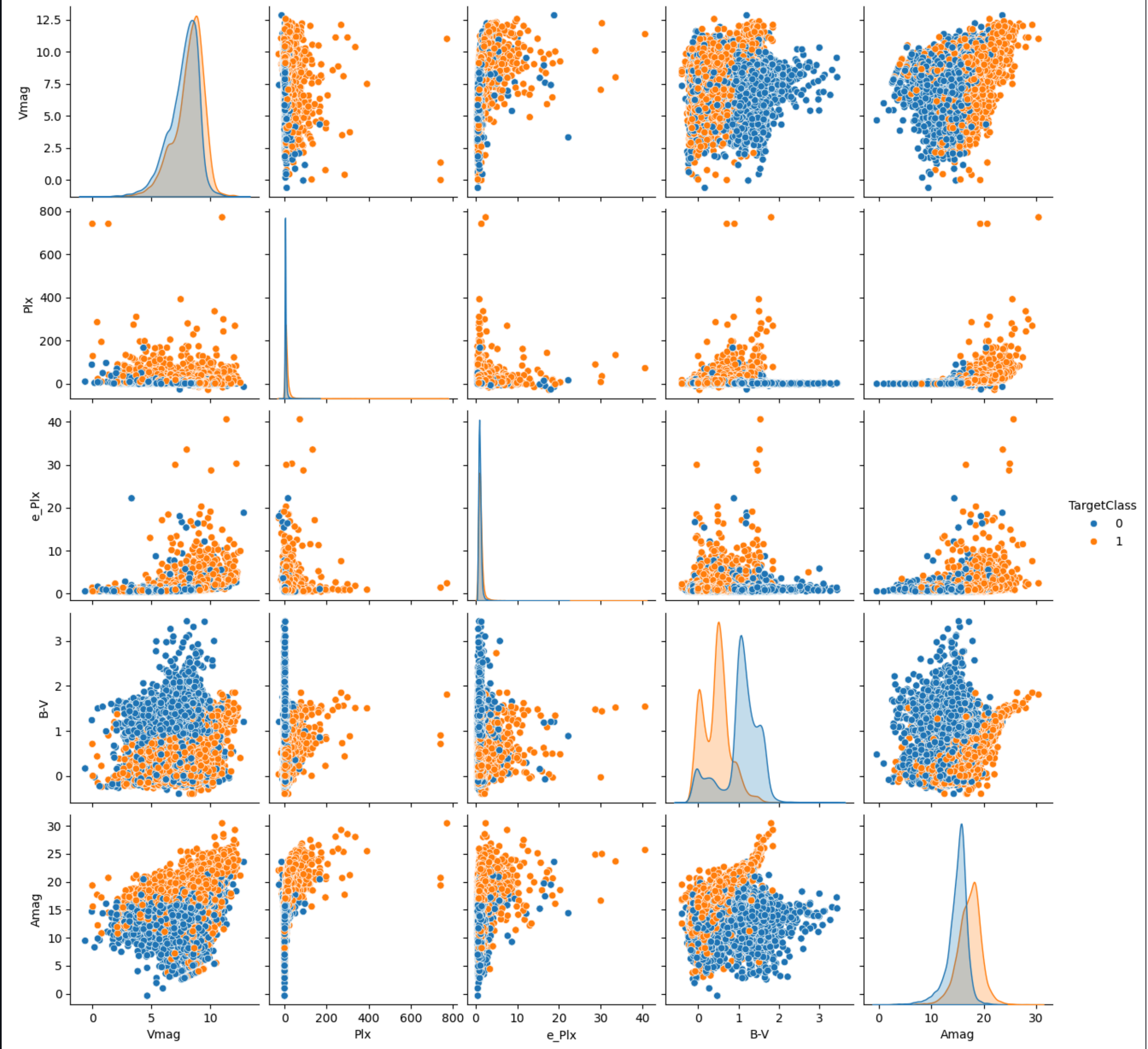
✓ 0.6s

Removing outliers using the IQR method



Box Plot without outliers

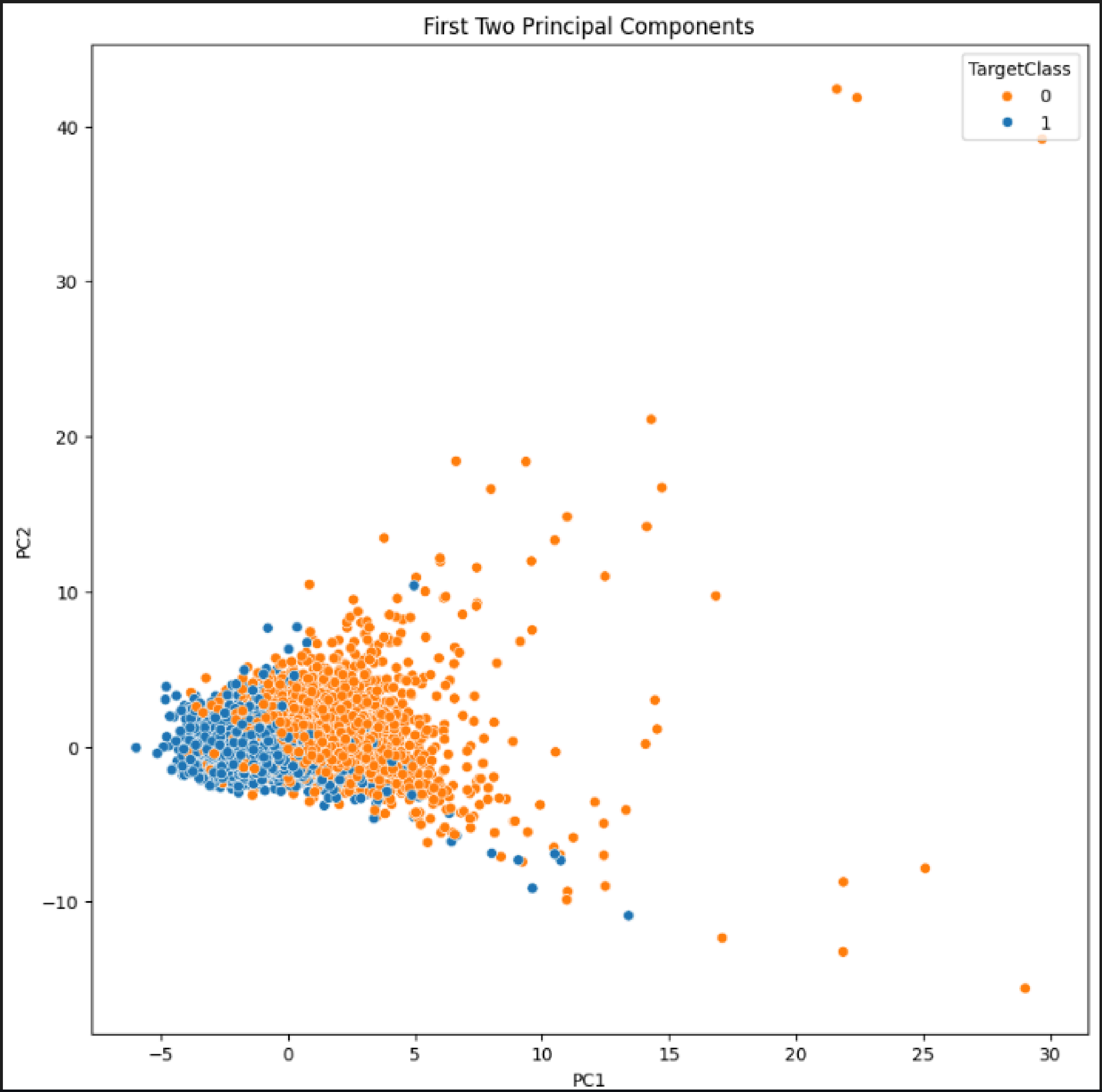
We also plotted the graphs of each feature to see the correlations between them. The 'Plx' and 'e_Plx' features had what seems to be outliers and weird patterns, so we conducted PCA to see if we could reduce the dimensionality of the dataset.



Feature Plots

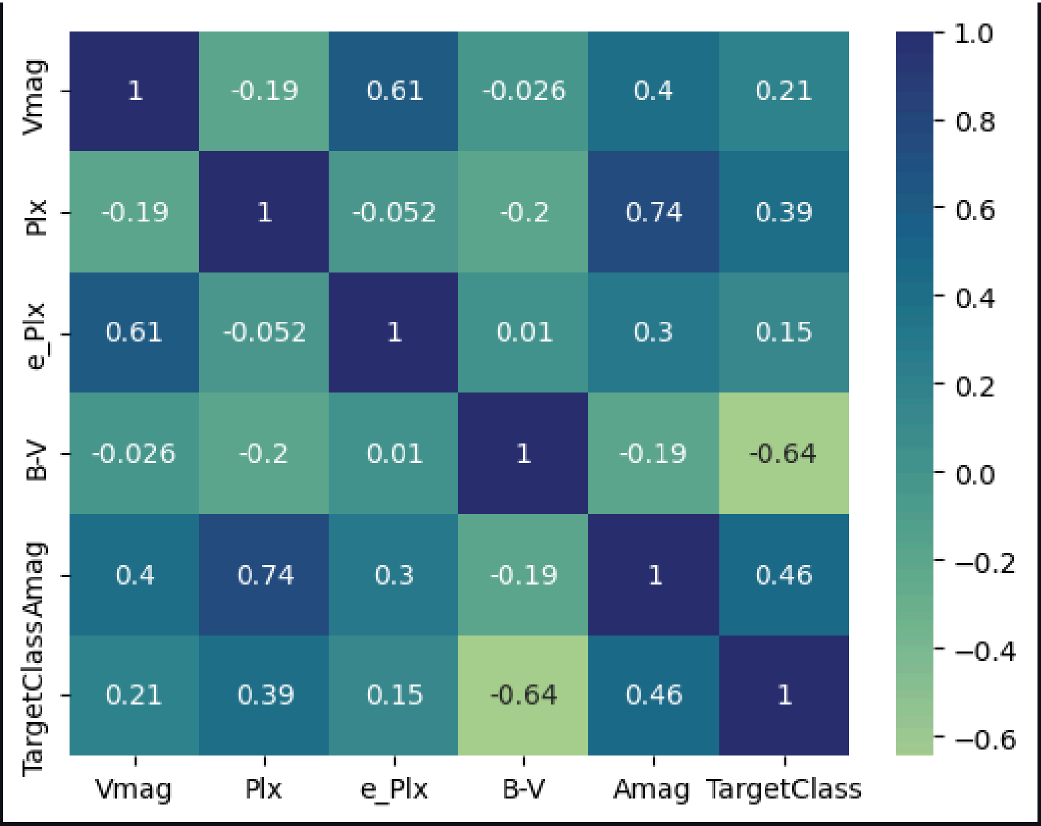
Conducting PCA, we were able to see the variance of each component, and as expected, 'Vmag', 'B-V', and 'Amag', comprised most of the variance, but 'Plx' and 'e_Plx' still had about 20% of the variance together.

```
Components Variance Rations:
PC1: 0.3361 (33.61%)
PC2: 0.2546 (25.46%)
PC3: 0.1974 (19.74%)
PC4: 0.1456 (14.56%)
PC5: 0.0663 (6.63%)

<matplotlib.legend.Legend at 0x20b41c2acc0>
```



PCA Variance

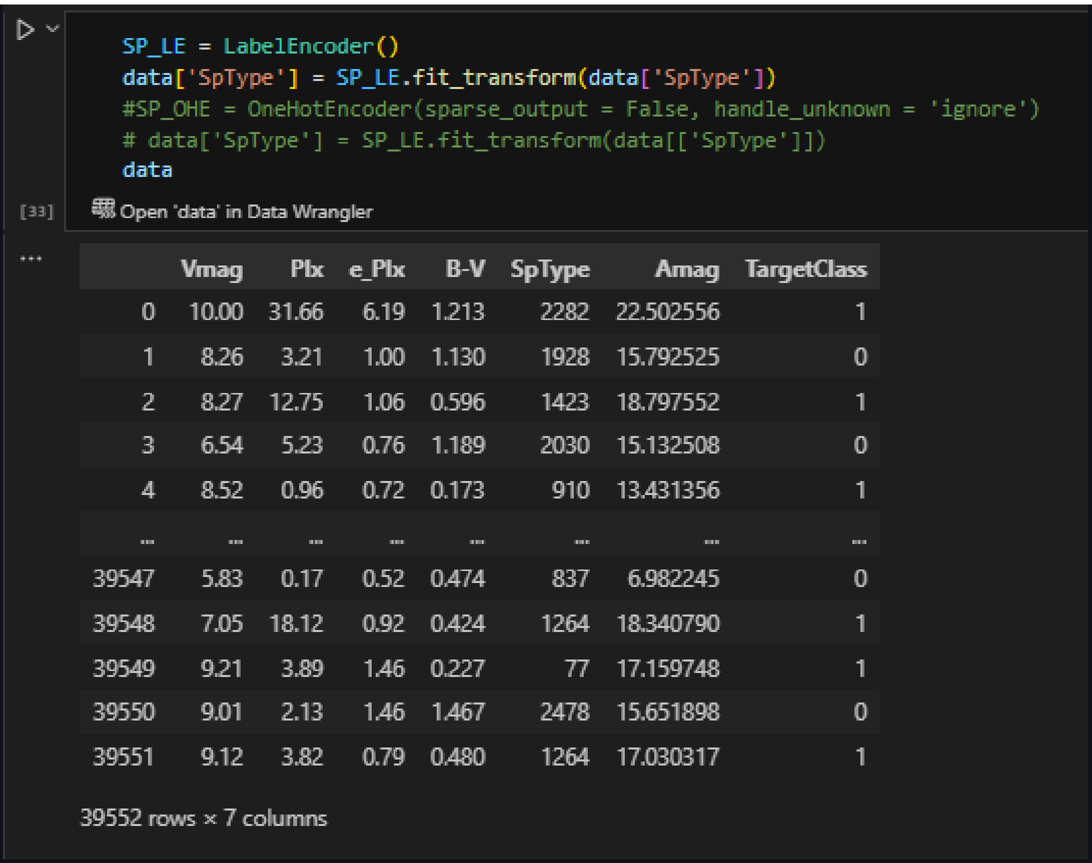
We also plotted a heatmap of feature correlations and saw that e_Plx and Plx had a high correlations with Vmag and Amag, respectively, so we wanted to see how our model's accuracy would be affected by removing these two features (e_Plx and Plx).



	Vmag	Plx	e_Plx	B-V	Amag	TargetClass
Vmag	1	-0.19	0.61	-0.026	0.4	0.21
Plx	-0.19	1	-0.052	-0.2	0.74	0.39
e_Plx	0.61	-0.052	1	0.01	0.3	0.15
B-V	-0.026	-0.2	0.01	1	-0.19	-0.64
Amag	0.4	0.74	0.3	-0.19	1	0.46
TargetClass	0.21	0.39	0.15	-0.64	0.46	1

Heatmap of Feature Correlations

Finally, we used a label encoder to encode the spectral type feature, which was a string, into a numerical value so that our model can train on it properly. We saw there was no significant difference between using a label encoder or a one hot encoder, so we went with one hot encoder because it is used for categorical features that are not ordinal or don't have an inheret ordering.



```
SP_LE = LabelEncoder()
data['SpType'] = SP_LE.fit_transform(data['SpType'])
#SP_OHE = OneHotEncoder(sparse_output = False, handle_unknown = 'ignore')
# data['SpType'] = SP_LE.fit_transform(data[['SpType']])
data
```

[33] Open 'data' in Data Wrangler

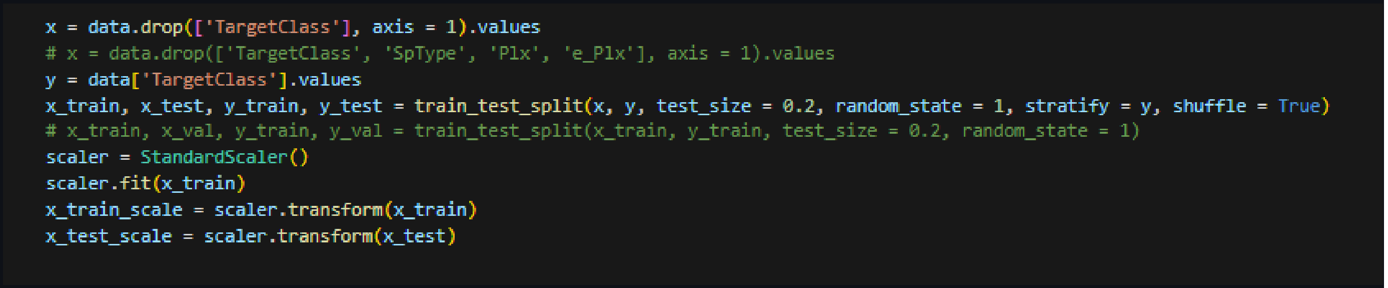
	Vmag	Plx	e_Plx	B-V	SpType	Amag	TargetClass
0	10.00	31.66	6.19	1.213	2282	22.502556	1
1	8.26	3.21	1.00	1.130	1928	15.792525	0
2	8.27	12.75	1.06	0.596	1423	18.797552	1
3	6.54	5.23	0.76	1.189	2030	15.132508	0
4	8.52	0.96	0.72	0.173	910	13.431356	1
...	...	...	...	...	...	...	...
39547	5.83	0.17	0.52	0.474	837	6.982245	0
39548	7.05	18.12	0.92	0.424	1264	18.340790	1
39549	9.21	3.89	1.46	0.227	77	17.159748	1
39550	9.01	2.13	1.46	1.467	2478	15.651898	0
39551	9.12	3.82	0.79	0.480	1264	17.030317	1

39552 rows × 7 columns

Label Encoding

We then dropped the 'Target Class' feature from our training data, otherwise we would be cheating, and assigned 'Target Class' values to our testing data. We then split our data into training and testing data with a 80/20 split, and we normalized the data using Standard Scaler so that we can identify true effects.

Normalizing the data is important because it allows the model to converge faster and prevents the model from being biased towards features with larger scales, and it helped improve the performance and stability of our model.



```
x = data.drop(['TargetClass'], axis = 1).values
# x = data.drop(['TargetClass', 'SpType', 'Plx', 'e_Plx'], axis = 1).values
y = data['TargetClass'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 1, stratify = y, shuffle = True)
# x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.2, random_state = 1)
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scale = scaler.transform(x_train)
x_test_scale = scaler.transform(x_test)
```

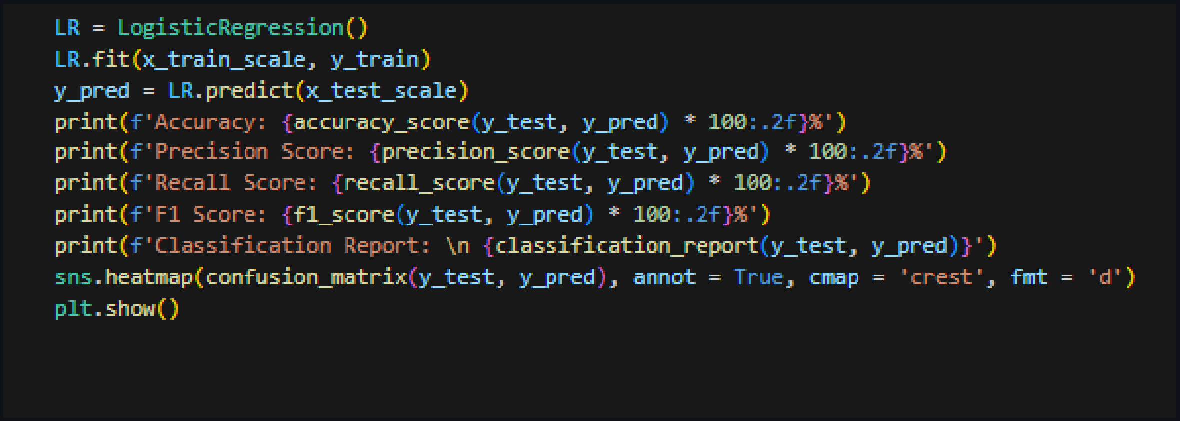
Splitting Data

ML Models:

We used logistic regression for our first model because it is a relatively simple model that is good for binary classification. The model assumes that the data is linearly separable and that there is a linear relationship, so it serves as a simple, interpretable model for binary classification.

Logistic regression uses the sigmoid functions which handles the decision boundary, so that is why it is good for binary classification. Also, logistic regression is good for a dataset with fewer features and can be combined with regularization to prevent overfitting.

Since this is a relatively simple model to implement, we decided it would be good to use as our first model because it is simple to understand and will serve as a benchmark for future models.



```python
LR = LogisticRegression()
LR.fit(x_train_scale, y_train)
y_pred = LR.predict(x_test_scale)
print(f'Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%')
print(f'Precision Score: {precision_score(y_test, y_pred) * 100:.2f}%')
print(f'Recall Score: {recall_score(y_test, y_pred) * 100:.2f}%')
print(f'F1 Score: {f1_score(y_test, y_pred) * 100:.2f}%')
print(f'Classification Report: \n {classification_report(y_test, y_pred)}')
sns.heatmap(confusion_matrix(y_test, y_pred), annot = True, cmap = 'crest', fmt = 'd')
plt.show()
```

Logistic Regression

The second model used was random forest classification. As an extension of the decision tree model, it consists of branching nodes separated via thresholds of certain feature values. Randomness is introduced from using random subsets of data to evaluate which features to partition from.

While this model is suitable for highly accurate binary classifications, it can also be very computationally intensive to run. It also has many hyperparameters that can be tuned to alter its efficiency. The ones we have chosen to callibrate are the number of estimators, the maximum depth, and the max percentage of features to pull from.

```
rf = RandomForestClassifier()
rf.fit(x_train, y_train)

y_pred = rf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
classification = classification_report(y_test, y_pred)

cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot()


print("Accuracy: ", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1: ", f1)
print("Classification Report: \n", classification)
```

Random Forest Classification

The third and final model we used to solve the classification problem is a Support Vector Machine (SVM). The idea is that this model can project the data into a higher-dimensional space where the classes become linearly separable.

Other advantages of using SVM are the ability to tune various hyperparameters to improve the model's performance (such as choosing different kernel types and kernel coefficients), and the option to include a regularization term so that we don't overfit the data.

However, since the optimization problem for this model grows quadratically with the dataset size, tuning and running SVM can be very expensive computationally and take a relatively long time compared to other simpler models.

Nonetheless, for our first attempt at SVM, we decided to use the simple linear kernel to see how well it would perform before tuning the hyperparameters.

```
# Split data into training and testing partitions
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Initialize SVM model
svm = SVC(kernel='linear', C=1.0, random_state=42)
svm.fit(x_train, y_train)

# Predict labels
y_pred = svm.predict(x_test)
```

SVM Classifier

Results and Discussion

Logistic Regression:

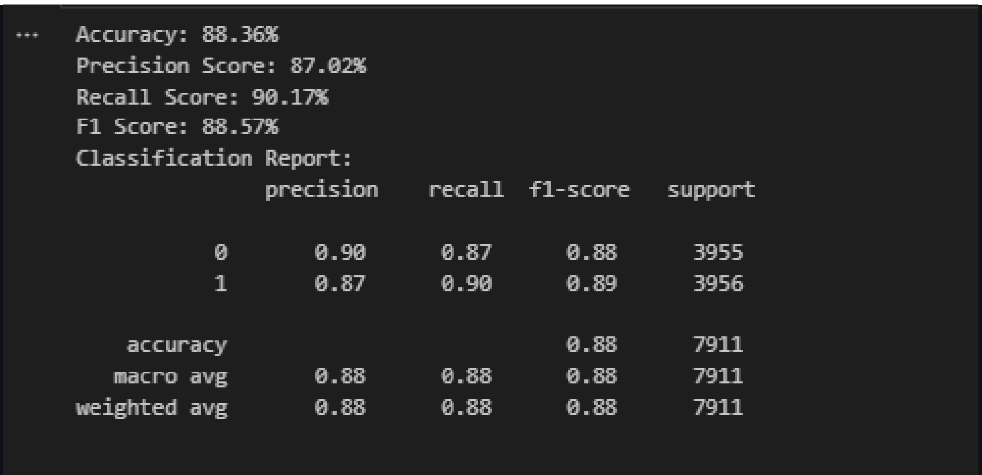
In our proposal, we wanted each metric to be at least 85% for our models, and we achieved that with logistic regression. For our initial results, we tested not removing outliers, using all features, and using a label encoder. As can be seen in the figure, we achieved:

- 88.36% Accuracy
- 87.02% Precision
- 90.17% Recall
- 88.57% F1 Score

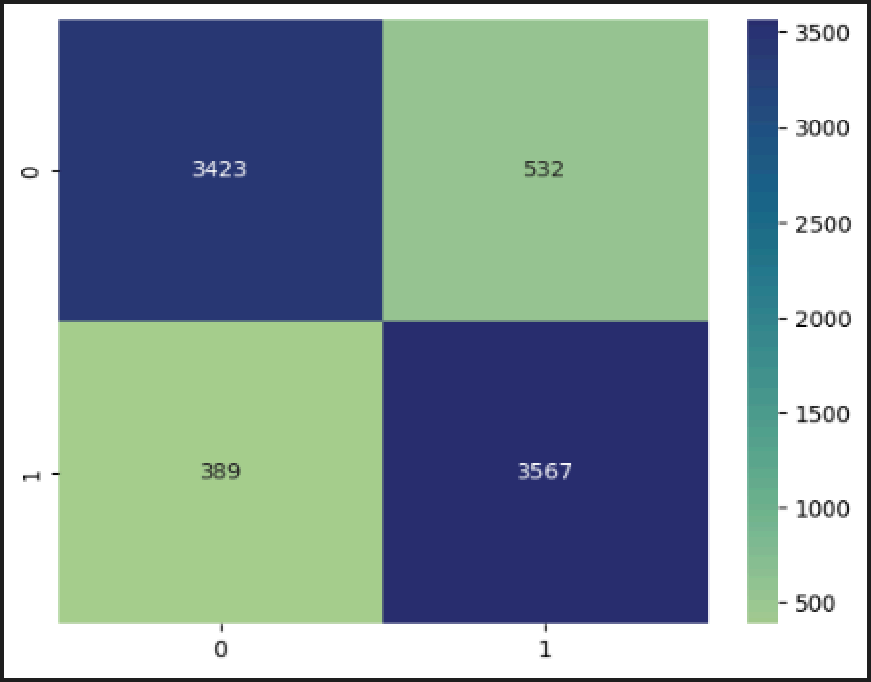
Our confusion matrix can be seen below, as well, and it shows that our model did a relatively good job identifying true positive and false negatives, around 3500 in both cases.

Our model performed well because we used a balanced dataset, encoded the categorical features, and split and normalized the data. Logistic regression does well with binary classification, which is why we saw good results since that is what our dataset and problem is.

We didn't see amazing results because logistic regression is one of the simpler models, and we only used the default parameters.



```
Accuracy: 88.36%
Precision Score: 87.02%
Recall Score: 90.17%
F1 Score: 88.57%
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.87      0.88      3955
           1       0.87      0.90      0.89      3956

    accuracy                           0.88      7911
   macro avg       0.88      0.88      0.88      7911
weighted avg       0.88      0.88      0.88      7911
```



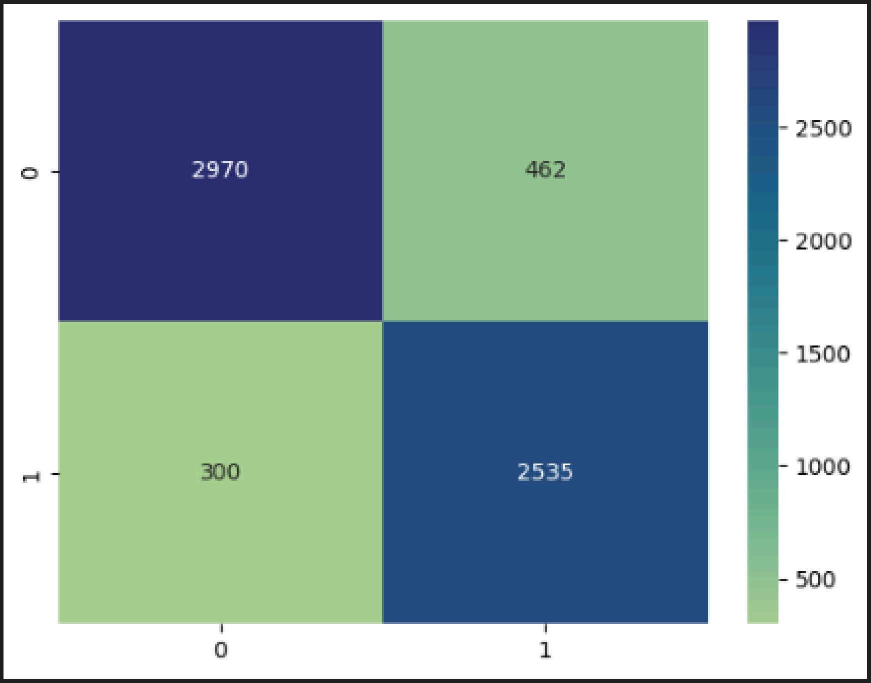
Basline Results

When we tried to remove the outliers in our dataset, we saw that our accuracy, precision, recall, and F1 all decreased. There are many possible explanations for this; one being that we might have removed outliers in the wrong way, especially for our specific data set.

Our dataset can be considered as relatively small, so removing outliers results in less data points and training data for our model to use. Additionally, the outliers may have been informative data points that reflect important variability in the data, which could have influenced the decision boundary as well.

Outliers also represent edge cases or rare cases that are present in the real world, so removing them might make the model less generalizable to unseen data, especially if the outliers were meaningful data points.

```
...   Accuracy: 87.84%
      Precision Score: 84.58%
      Recall Score: 89.42%
      F1 Score: 86.93%
      Classification Report:
                    precision    recall  f1-score   support

                 0       0.91      0.87      0.89      3432
                 1       0.85      0.89      0.87      2835

          accuracy                           0.88      6267
         macro avg       0.88      0.88      0.88      6267
      weighted avg       0.88      0.88      0.88      6267
```

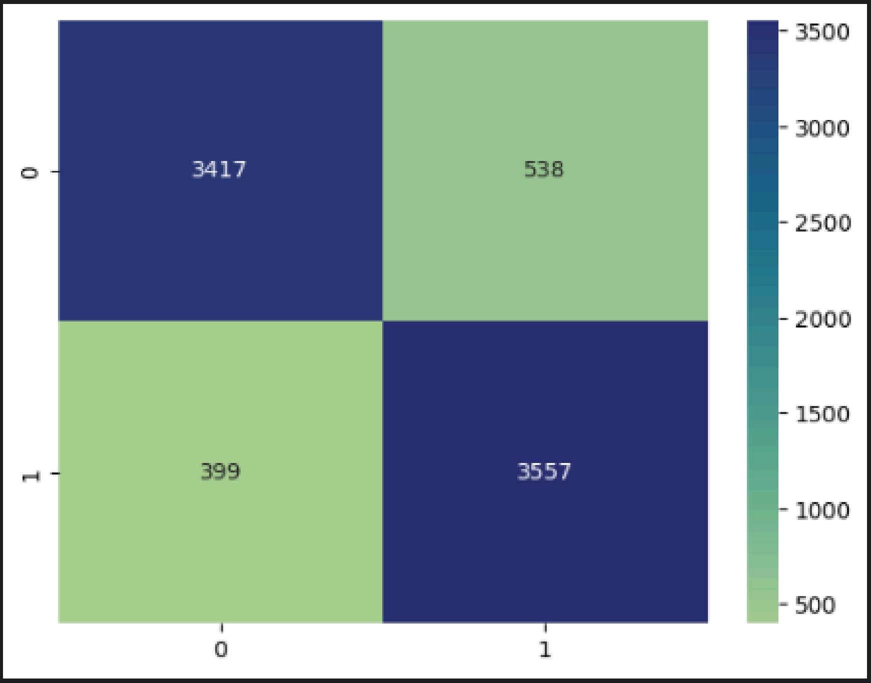


Outlier Results

Going off of the results of PCA, if we removed the 'Plx', 'e_Plx', and Spectral Type features from the dataset, we found that the our performance metrics only slightly decreased. Although there are slight changes in performance, we are decreaseing the dimensionality of our dataset by removing three features, and we are getting basically the same accuracy, precision, and recall.

This is good because it shows that our model is not relying on these features to make predictions, and it can still make accurate predictions without them.

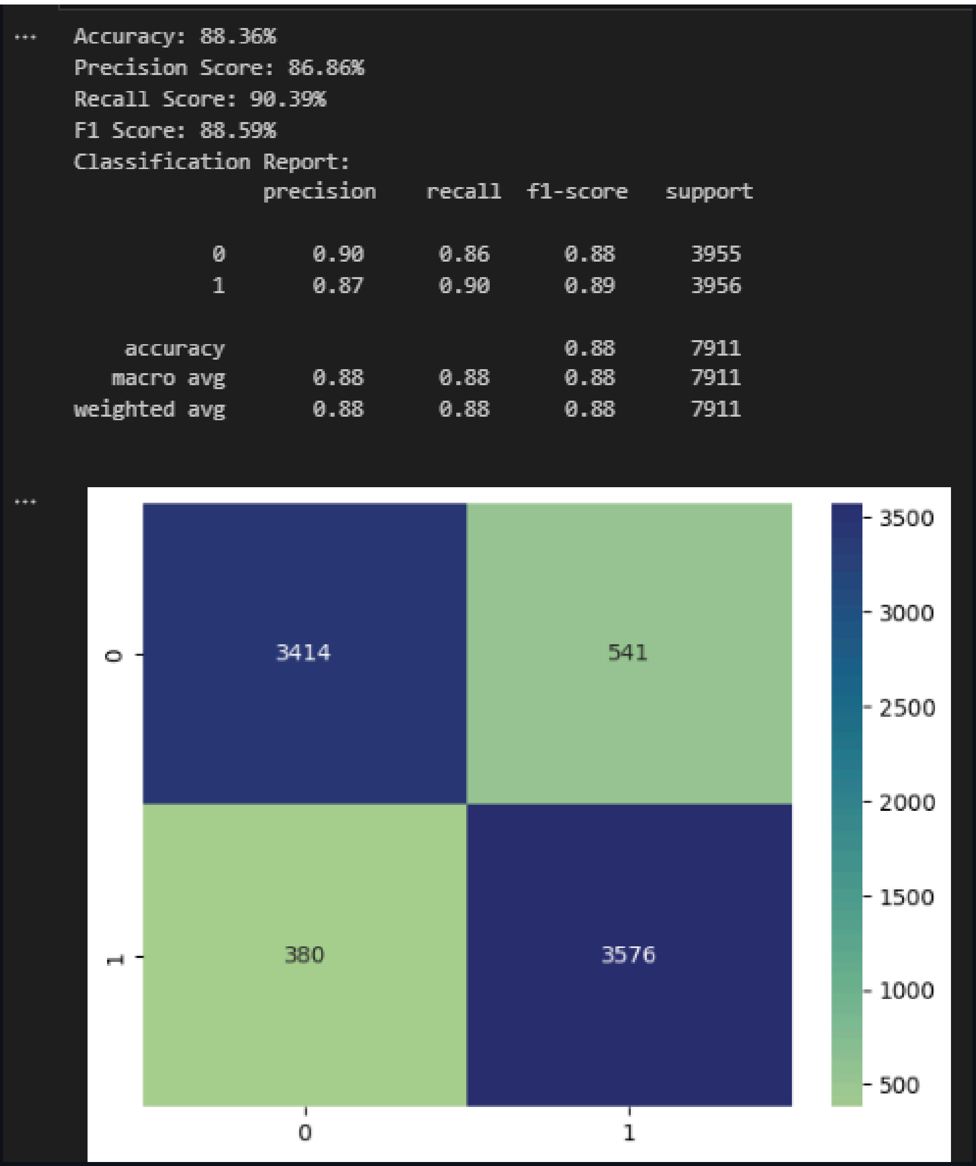
```
Accuracy: 88.16%
Precision Score: 86.86%
Recall Score: 89.91%
F1 Score: 88.36%
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88      3955
           1       0.87      0.90      0.88      3956

    accuracy                           0.88      7911
   macro avg       0.88      0.88      0.88      7911
weighted avg       0.88      0.88      0.88      7911
```



PCA Results

Finally, we wanted to see the differences in performance if we used label encoding compared to one hot encoding. We found that one hot encoding had a slightly lower precision score, but slightly higher f1 and recall score.

However, using one hot encoding makes the most sense because it is used for categorical features that are not ordinal or don't have an inheret ordering, and it is better for our model to train on. Therefore, moving forward, we will use a one-hot encoder for our data for other models to train on.



```
Accuracy: 88.36%
Precision Score: 86.86%
Recall Score: 90.39%
F1 Score: 88.59%
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88      3955
           1       0.87      0.90      0.89      3956

    accuracy                           0.88      7911
   macro avg       0.88      0.88      0.88      7911
weighted avg       0.88      0.88      0.88      7911
```

One Hot Encoding Results

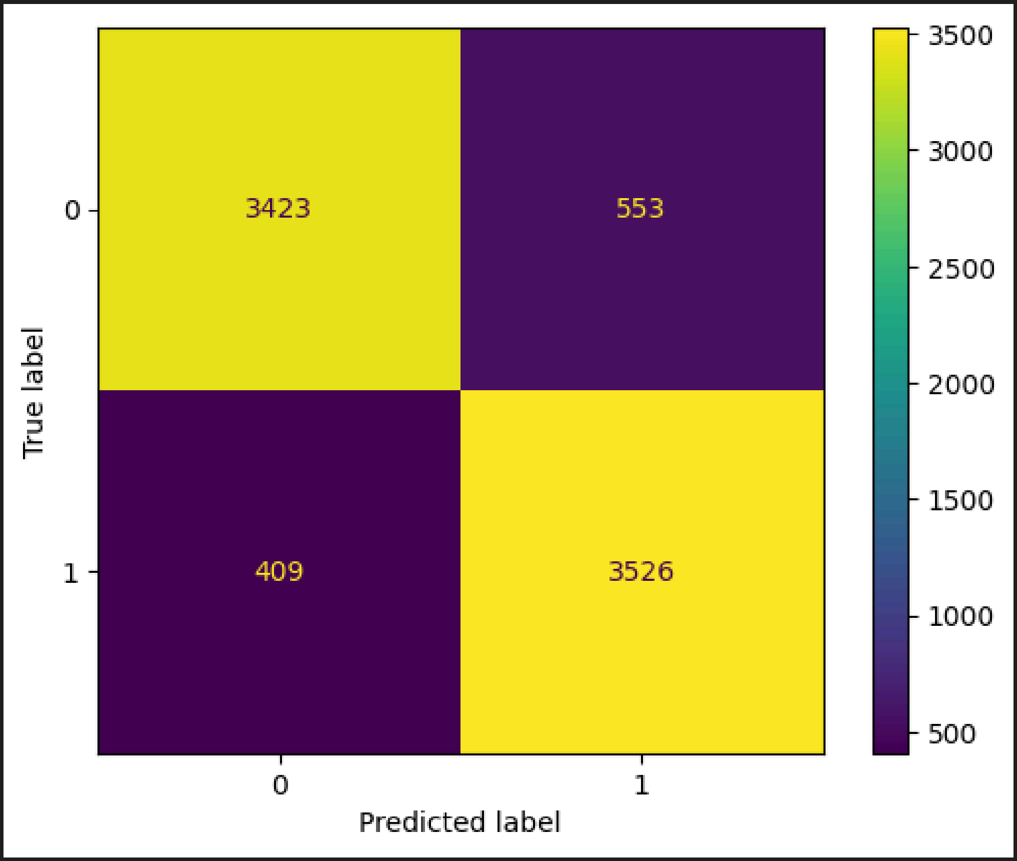
Overall, our model performed well as every metric was above 85%, and we were able to see how different preprocessing methods affected our model's performance.

We can attribute the model's success to the nature of the classification and the data pre-processing. We only used the default parameters for logistic regression, so the results may not have been the best they could've been.

Random Forest Classification:

For our random forest classification model produced fairly accurate results, as seen below. This model was tested with the default hyperparameters provided by sklearn and one hot encoding. It crosses over the 85% threshold sought after.

```
Accuracy:  0.8783971684995576
Precision: 0.8644275557734739
Recall: 0.8960609911054638
F1:  0.879960069877714
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.86      0.88      3976
           1       0.86      0.90      0.88      3935

    accuracy                           0.88      7911
   macro avg       0.88      0.88      0.88      7911
weighted avg       0.88      0.88      0.88      7911
```

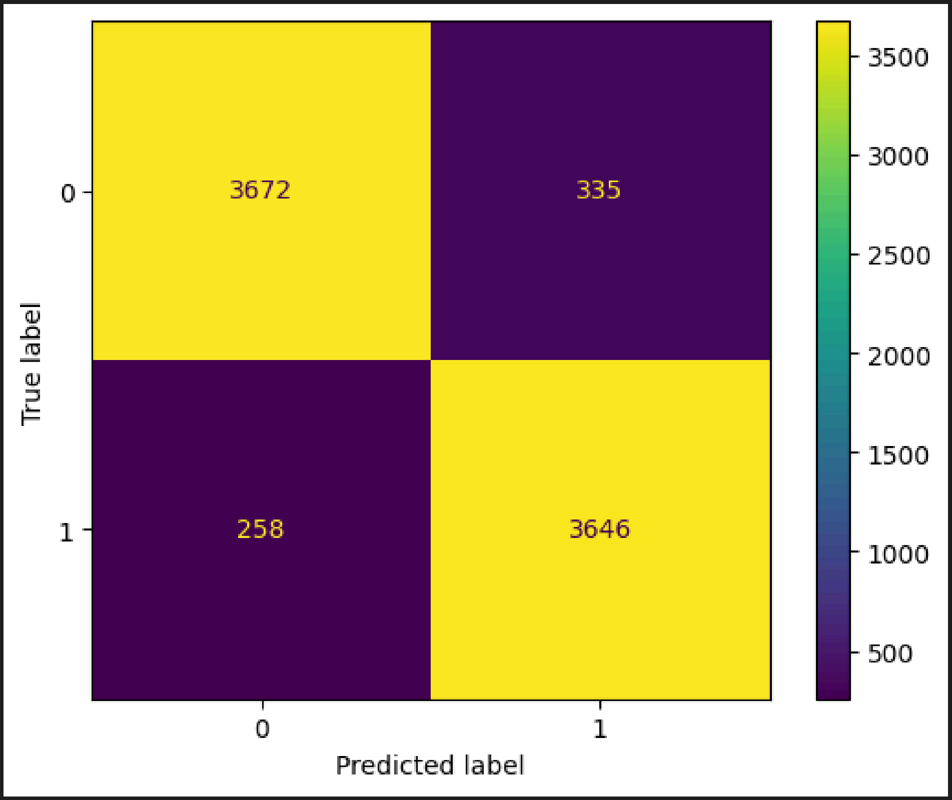


Random Forest Base Results

There are more values we can adjust to further increase the efficiency of the model. For one, we found that swapping back to label encoding over one hot encoding resulted in an increase in performace across the board, as seen below. This increase in performace is still apparent in future changes made to the model's values.

This could be perhaps due to random forest tree's model fitting better to label encoding's categorical values rather than one hot encodings feature values.

```
Accuracy:  0.9250410820376691
Precision: 0.9158502888721427
Recall: 0.9339139344262295
F1:  0.9247939124920735
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.92      0.93      4007
           1       0.92      0.93      0.92      3904

    accuracy                           0.93      7911
   macro avg       0.93      0.93      0.93      7911
weighted avg       0.93      0.93      0.93      7911
```

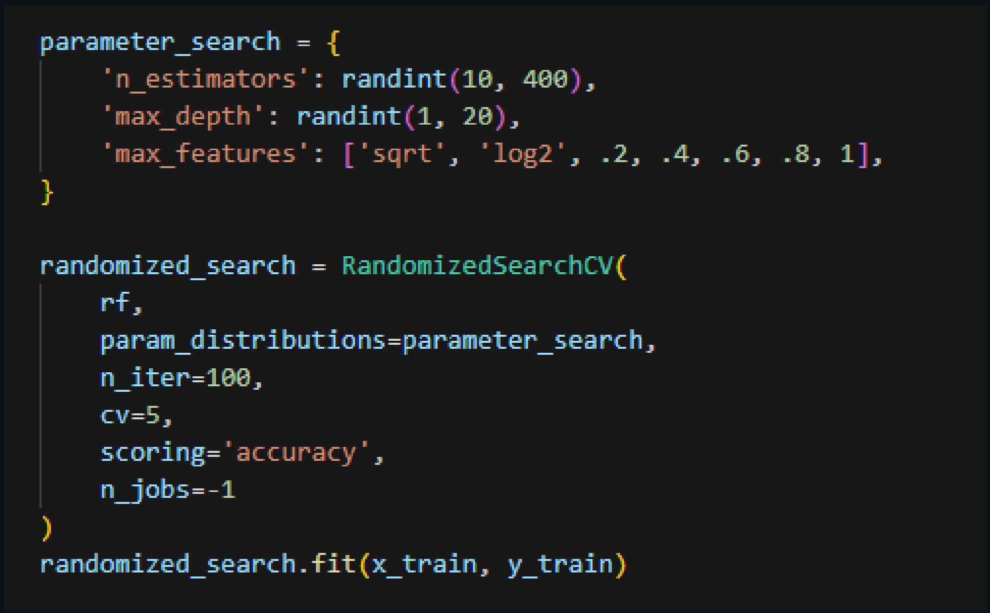


Random Forest w/ Label Encoding

The model can be further improved by finding specific hyperparameter values that better fit the given dataset. As mentioned before, the hyperparameters we will be attempting to tune are the number of estimators, maximum depth, and the percentage of features to use.

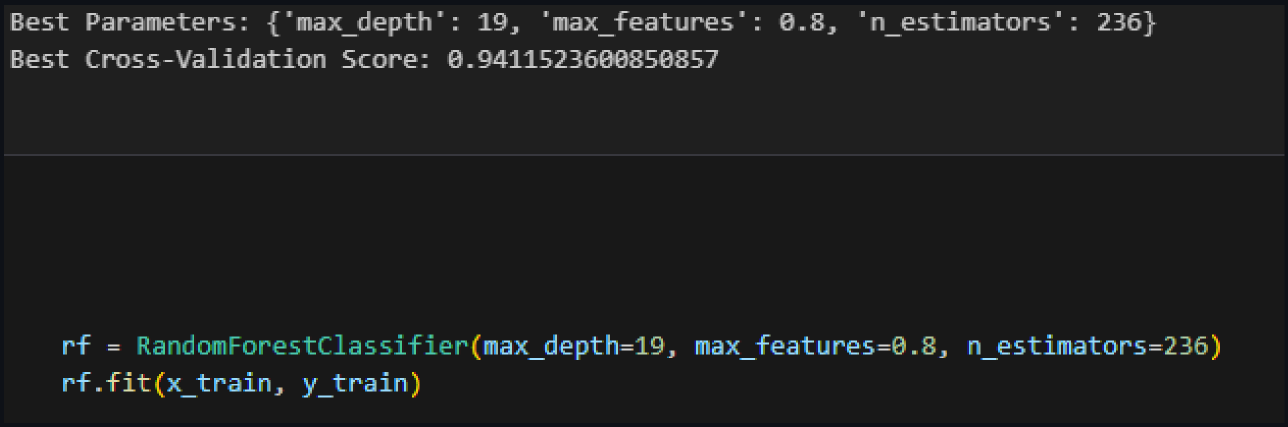
Given the range and amount of hyperparameters to determine, finding the optimal combination can be an tedious and extraneous process. As such, we will be utilizing sklearn's RandomizedSearchCV model to search for it. It runs via testing random sets of hyperparameter values, and returns the combination with the highest accuracy.

We then used the hyperparameters provided by the search to set intial values for the random forest model.



```
parameter_search = {
    'n_estimators': randint(10, 400),
    'max_depth': randint(1, 20),
    'max_features': ['sqrt', 'log2', .2, .4, .6, .8, 1],
}

randomized_search = RandomizedSearchCV(
    rf,
    param_distributions=parameter_search,
    n_iter=100,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
randomized_search.fit(x_train, y_train)
```

RandomizedSearchCV



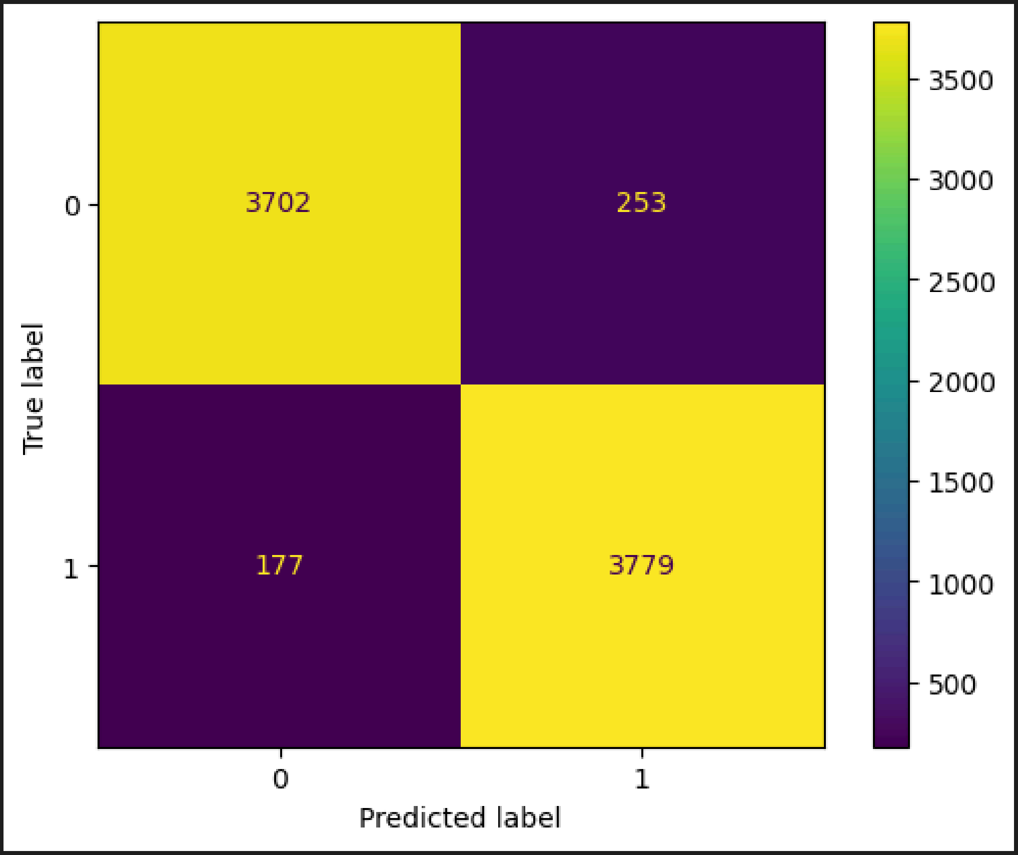
```
Best Parameters: {'max_depth': 19, 'max_features': 0.8, 'n_estimators': 236}
Best Cross-Validation Score: 0.9411523600850857
```

```
rf = RandomForestClassifier(max_depth=19, max_features=0.8, n_estimators=236)
rf.fit(x_train, y_train)
```

RandomizedSearchCV Results

When running the model with these new hyperparameters, we see a substantial increase in performance in accuracy, precision, recall, and F1 scores. This is likely due to the new hyperparameters scaling better with larger values and ranges. Compared to the default values, with values relatively smaller, the new values fit better for larger datasets.

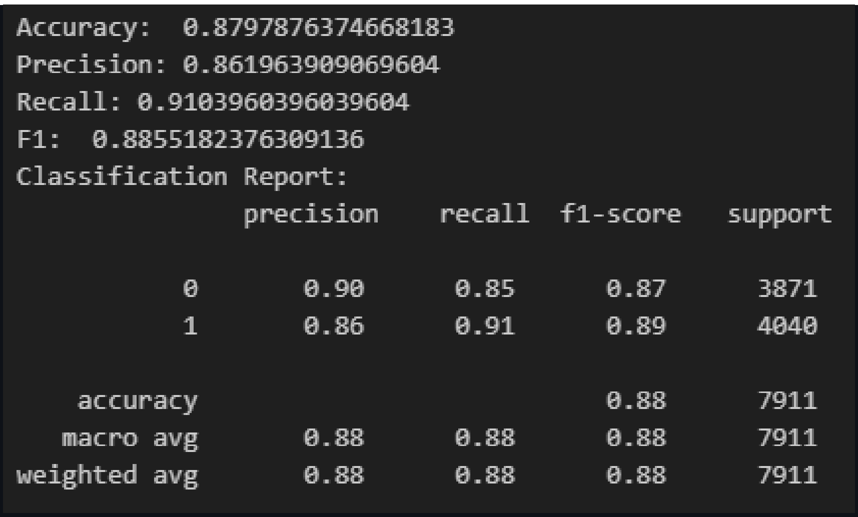
```
Accuracy:  0.9456453040070788
Precision: 0.9372519841269841
Recall: 0.9552578361981799
F1:  0.9461692538808212
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.95      3955
           1       0.94      0.96      0.95      3956

    accuracy                           0.95      7911
   macro avg       0.95      0.95      0.95      7911
weighted avg       0.95      0.95      0.95      7911
```



Random Forest w/ Hyperparameter Tuning

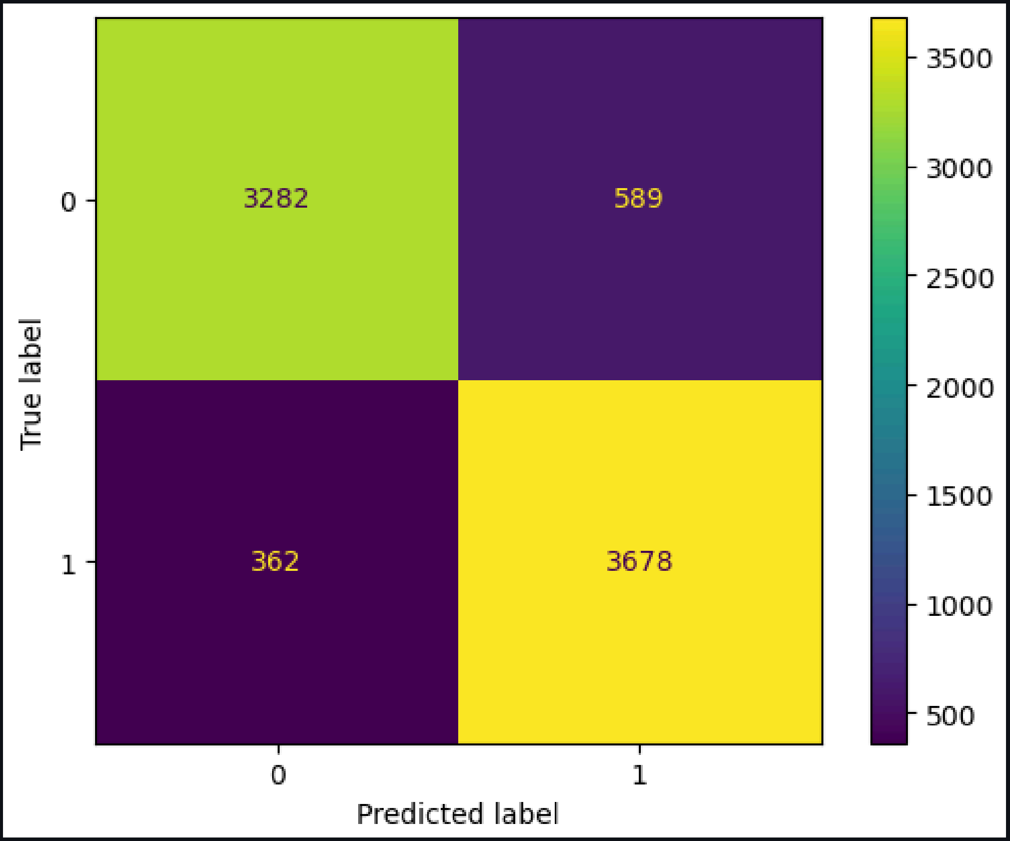
SVM Classifier:

The default linear SVM classifier produced similar initial results as the random forest model, which is a decent result for this version of SVM which runs the fastest.



```
Accuracy:  0.8797876374668183
Precision: 0.861963909069604
Recall: 0.9103960396039604
F1:  0.8855182376309136
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.85      0.87      3871
           1       0.86      0.91      0.89      4040

    accuracy                           0.88      7911
   macro avg       0.88      0.88      0.88      7911
weighted avg       0.88      0.88      0.88      7911
```

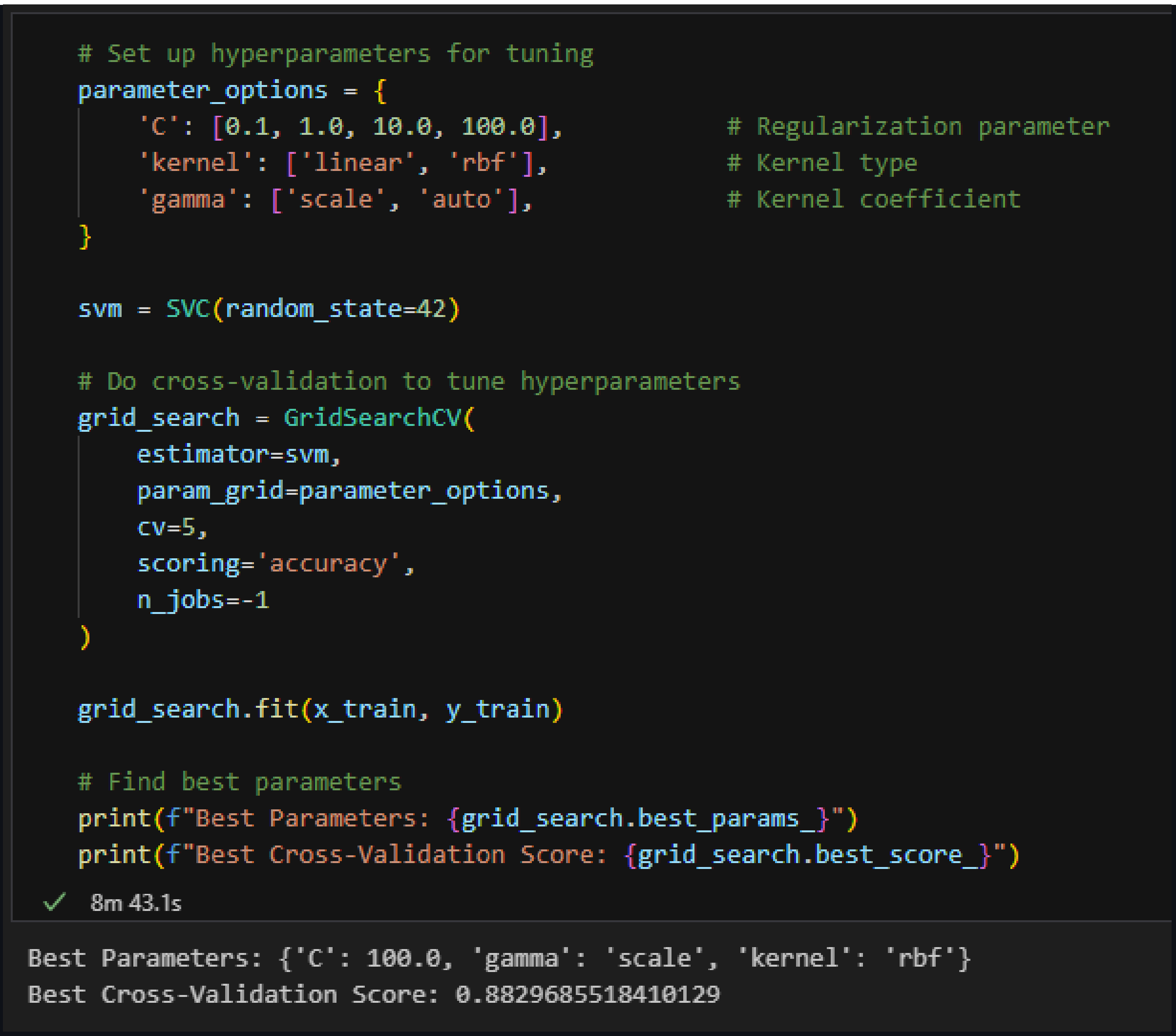
Linear SVM Results



Linear SVM Confusion Matrix

We can rely on the results from previous models as evidence that Label Encoding is better than One-Hot Encoding, so we can immediately try to tune the parameters to attempt to gain even better performance.

Shown below is the entire hyperparameter space that was tested to see which combination performed better.



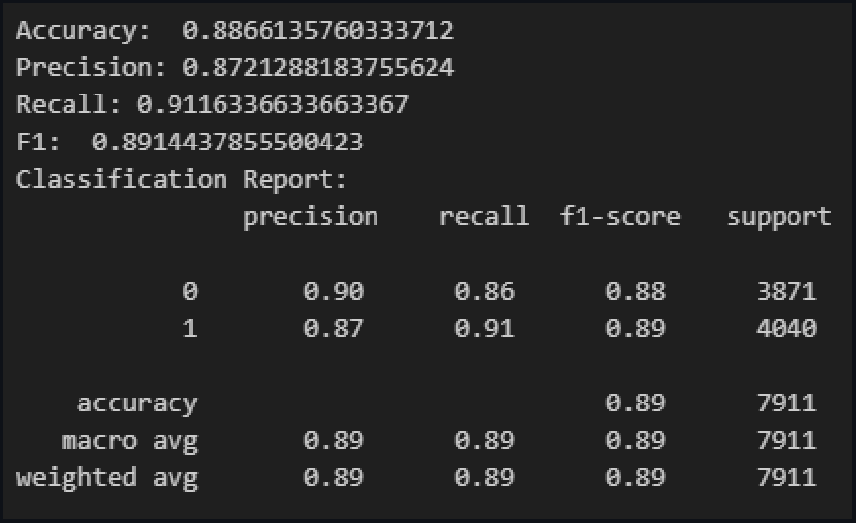
```
# Set up hyperparameters for tuning
parameter_options = {
    'C': [0.1, 1.0, 10.0, 100.0],           # Regularization parameter
    'kernel': ['linear', 'rbf'],             # Kernel type
    'gamma': ['scale', 'auto'],              # Kernel coefficient
}

svm = SVC(random_state=42)

# Do cross-validation to tune hyperparameters
grid_search = GridSearchCV(
    estimator=svm,
    param_grid=parameter_options,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search.fit(x_train, y_train)

# Find best parameters
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-Validation Score: {grid_search.best_score_}")
```

✓ 8m 43.1s

```
Best Parameters: {'C': 100.0, 'gamma': 'scale', 'kernel': 'rbf'}
Best Cross-Validation Score: 0.8829685518410129
```

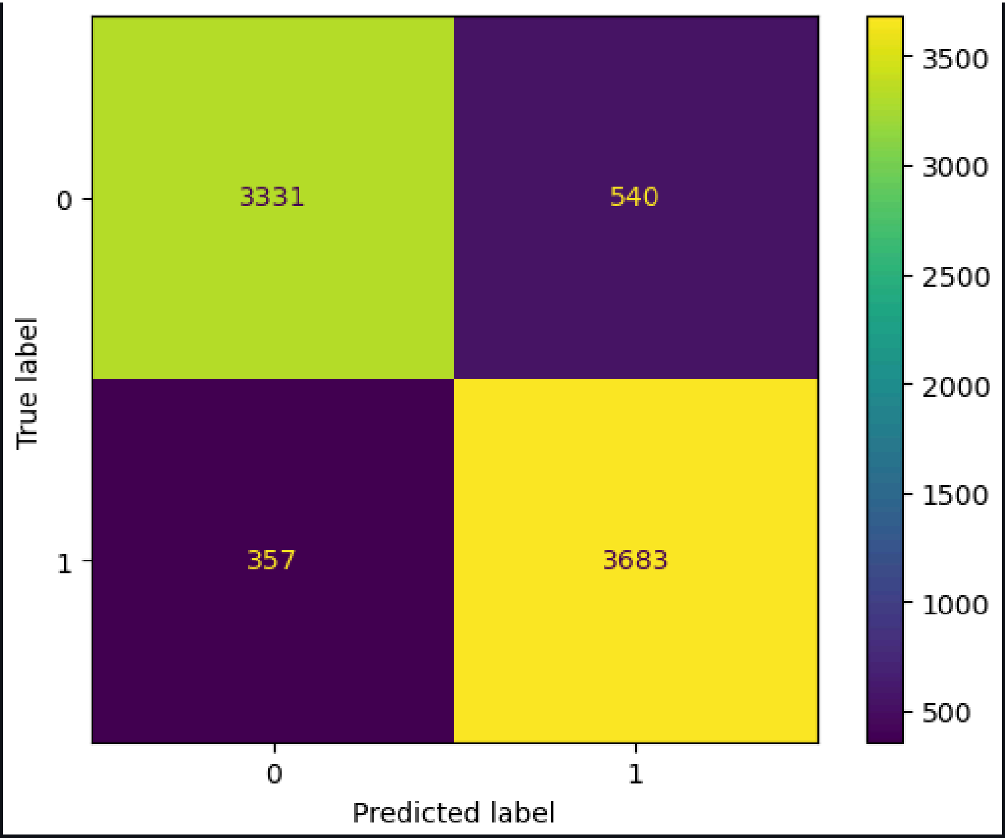
SVM Hyperparameter Tuning

After finding the best SVM parameters, we trained a new model to see how it compared to the previous linear SVM.



```
Accuracy:  0.8866135760333712
Precision: 0.8721288183755624
Recall: 0.9116336633663367
F1:  0.8914437855500423
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88      3871
           1       0.87      0.91      0.89      4040

    accuracy                           0.89      7911
   macro avg       0.89      0.89      0.89      7911
weighted avg       0.89      0.89      0.89      7911
```

Tuned SVM Results



Tuned SVM Confusion Matrix

As you might observe, in this situation, tuning the hyperparameters from the ones we chose only improved performance by a near-negligible amount.

Furthermore, the time that it took to tune the hyperparameters and then train the model was too long for the results we obtained.

In terms of binary star classification, we would be better of using a different model.

Conclusion

We have incorporated GridSearchCV and RandomizedSearchCV to tune the hyperparameters of our models, which has helped us achieve better results.

The Random Forest model has higher performance metrics and more concentrated true positive/negatives in the confusion matrix

One reason that the Random Forest performed the best could be that the classification involved more complex, non-linear boundaries. Random Forests are known to be good at handling more complex, non-linear data since they are a collection of decision trees.

Random Forests are also good at handling outliers and extreme values because the individual trees help average out the noise.

Another reason could be that the Random Forest model was able to capture the interactions between the features better than the other models.

The Logistic Regression and SVM Models may have assumed a linear decision boundary or separability which may not be ideal for this calssification. These models were sensitive to outliers and could not capture the complex, non-linear feature relationship as well as random forest.

Although the random forest model performed the best, it is important to note that it is less interpretable, uses more memory and computational reosources, and the time to predict increases with the number of trees. Overall, it is more complex and computationally intensive compared to the other models.

Logistic Regression and SVM are relatively simple to implement and explain, especially from scratch. They are less computationally expensive, require lower memory, and have relatively faster prediction times, compared to Random Forest.

However, as we have seen, they don't proide the best results for this classification problem, which was expected as we predicted Random Forest to perform the best, initially.

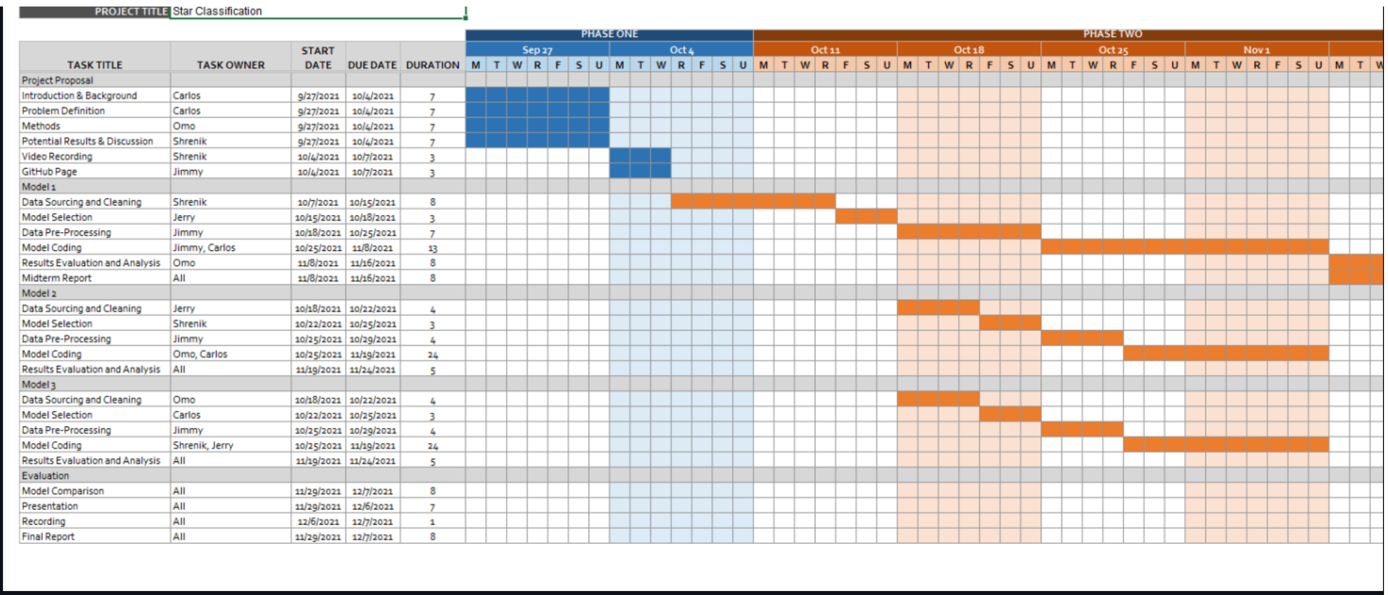
References

[1] D. G. Armstrong et al., “K2 variable catalogue – II. Machine learning classification of variable stars and eclipsing binaries in K2 fields 0–4,” Monthly Notices of the Royal Astronomical Society, vol. 456, no. 2, pp. 2260–2272, Feb. 2016, doi: https://doi.org/10.1093/mnras/stv2836.

[2] J. Zhang et al., “High-Accuracy Guide Star Catalogue Generation with a Machine Learning Classification Algorithm,” Sensors, vol. 21, no. 8, p. 2647, Apr. 2021, doi: https://doi.org/10.3390/s21082647.

[3] Z. Qi, “Stellar Classification by Machine Learning,” SHS Web of Conferences, vol. 144, p. 03006, 2022, doi: https://doi.org/10.1051/shsconf/202214403006.

Gantt Chart



PROJECT TITLE Star Classification

TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION
Project Proposal				
Introduction & Background	Carlos	9/27/2021	10/4/2021	7
Problem Definition	Carlos	9/27/2021	10/4/2021	7
Methods	Omo	9/27/2021	10/4/2021	7
Potential Results & Discussion	Shrenik	9/27/2021	10/4/2021	7
Video Recording	Shrenik	10/4/2021	10/7/2021	3
GitHub Page	Jimmy	10/4/2021	10/7/2021	3
Model 1				
Data Sourcing and Cleaning	Shrenik	10/7/2021	10/15/2021	8
Model Selection	Jerry	10/15/2021	10/18/2021	3
Data Pre-Processing	Jimmy	10/18/2021	10/25/2021	7
Model Coding	Jimmy, Carlos	10/25/2021	11/8/2021	13
Results Evaluation and Analysis	Omo	11/8/2021	11/16/2021	8
Midterm Report	All	11/8/2021	11/16/2021	8
Model 2				
Data Sourcing and Cleaning	Jerry	10/18/2021	10/22/2021	4
Model Selection	Shrenik	10/22/2021	10/25/2021	3
Data Pre-Processing	Jimmy	10/25/2021	10/29/2021	4
Model Coding	Omo, Carlos	10/25/2021	11/19/2021	24
Results Evaluation and Analysis	All	11/19/2021	11/24/2021	5
Model 3				
Data Sourcing and Cleaning	Omo	10/18/2021	10/22/2021	4
Model Selection	Carlos	10/22/2021	10/25/2021	3
Data Pre-Processing	Jimmy	10/25/2021	10/29/2021	4
Model Coding	Shrenik, Jerry	10/25/2021	11/19/2021	24
Results Evaluation and Analysis	All	11/19/2021	11/24/2021	5
Evaluation				
Model Comparison	All	11/29/2021	12/7/2021	8
Presentation	All	11/29/2021	12/6/2021	7
Recording	All	12/6/2021	12/7/2021	1
Final Report	All	11/29/2021	12/7/2021	8

There is a zoom in feature when you hover over the image.

Contribution Table

Name	Contributions
Jimmy	Random Forest Model and Write-Up
Carlos	Linear SVM Model and Write-Up
Omo	Put Slides Together
Shrenik	Conclusion Write-Up and Video Recordign
Jerry	Put Slides Together