Introduction:

The dataset I chose is NASA's TESS (Transiting Exoplanet Survey Satellite) dataset. This dataset consists of information about stellar objects, and whether or not they turned out to be a known exoplanet, a possible planetary candidate, a false alarm, a confirmed planet, an ambiguous planetary candidate, or a false positive. My goal was to try and use this dataset to try and classify stellar objects into these six different categories, based on the information provided about it. NASA's TESS dataset was a good choice as it contains a large amount of information about the stellar objects they've detected, along with whether they turned out to be a planet or not. So it should contain all the information needed to classify these stellar objects properly. I used a Random Forest classifier and a Support Vector Machine (SVM) for my model. These models seemed like good choices due to the ability to handle non-linear relationships, for an SVM, and easily rank feature importance, for a random forest.

Data Exploration and Preprocessing:

For exploration and preprocessing, I first examined the structure of the data, so how large the dataset was, if it had any categorical values, etc. I found that the dataset was fairly large, about 86 features with 7351 entries each. It contained a few categorical variables, and there were a large amount of NA values for certain features. I then began preprocessing my data.

I first removed features that weren't necessary for the problem I was trying to solve. The dataset contained features such as the data the object was discovered, and the date it was added to the dataset. As these had no bearing on the classification of the stellar object, I removed them by simply dropping them from the dataset.

Next I wanted to handle any categorical values in my dataset. I only really had one categorical feature that was necessary to solve my problem, which was actually my target variable itself, the others had been dropped as they were just dates. To handle this I used label encoding to convert each category to a numerical value, so Ambiguous Planetary Candidate became 0, Confirmed Planet became 1, etc. This made it much easier to handle, and later on when evaluating the performance of the dataset I could convert them back to their categorical labels to make the confusion matrix easier to understand.

Next there was an issue where certain features were in sexagesimal format, or base 60. To handle this I used the Astropy library to convert these features into decimal format, as that made them much easier to use in training my model.

I found that some of the features were missing quite a lot of data, while some were missing only a small amount. About half of my total features were missing at least some data. I decided to remove any features that were missing over 75% of their data, and for those that weren't, if the feature was highly skewed I used the median to fill in the empty values, and if not I used the mean. I used the simple_imputer class from sklearn to do this, since it made the process much easier

I then tested to see how skewed my data was, to see if any normalization was necessary, I discovered that it was, so I used Z-score Normalization, since it scales the data such that it has a mean of 0 and a standard deviation of 1. I did this because the features consisted of different scales and units. This was the last step I took before I began model selection and training.

Model Selection and Training:

I began by splitting my model into training and test datasets. I did a 80% train and 20% test split. Our target category consists of 6 different labels, which are as follows, APC: Ambiguous Planetary Candidate, CP: Confirmed Planet, FA: False Alarm, FP: False Positive, KP: Known Planet, and PC: Planetary Candidate.

The first model I tested as a Random Forest. I chose 200 trees since that seemed like a reasonable balance between accuracy and performance. I set max depth to none, to ensure that it would find all patterns that may appear in the data. I set class weight to balanced, to hopefully handle the somewhat imbalance classes for my target variable, e.g. there are more planets than the other categories such as false positives.

I choose a random forest as one of the models since The TESS dataset consists of multiple features and a Random Forest is good for capturing non-linear relationships between these features, and the TESS dataset includes imbalanced data e.g. more planetary candidates, a Random Forest is good for preventing overfitting.

To evaluate the Random Forest's performance, I used a confusion matrix and a classification report to see the precision, recall, and f1-scores.

My results turned out interesting, for precision I found that:

APC (0.60): 60% of the instances predicted as APC are actually APC.

CP (0.84): 84% of the instances predicted as CP are actually CP.

FA (0.00): This is problematic, as no instances predicted as FA are actually FA. The model is failing to detect this class. I looked more into this and the dataset actually does contain FA values, so our model should detect at least a few.

FP (0.71): 71% of the instances predicted as FP are actually FP.

KP (0.85): 85% of the instances predicted as KP are actually KP.

PC (0.69): 69% of the instances predicted as PC are actually PC.

For recall I found:

APC (0.03): Only 3% of the actual APC instances are correctly identified. This is a very low recall, indicating that the model is missing most APC instances.

CP (0.17): Only 17% of the actual CP instances are correctly identified. This is also a low recall, suggesting that the model was struggling to detect CP instances.

FA (0.00): No actual FA instances are being correctly identified. This is a significant issue, as the model isn't detecting any FA instances at all.

FP (0.22): Only 22% of the actual FP instances are correctly identified. The model is missing many FP instances.

KP (0.39): 39% of the actual KP instances are correctly identified. The recall is not very high, indicating room for improvement in detecting KP instances.

PC (0.99): 99% of the actual PC instances are correctly identified. This is an excellent recall, meaning the model is very good at detecting PC instances.

F1-Score gives:

APC (0.06): The F1-score is very low for APC, indicating poor performance in terms of both precision and recall.

CP (0.28): The F1-score for CP is low, reflecting the low recall and slightly better precision.

FA (0.00): The F1-score is also 0 for FA, reflecting the model's inability to detect any instances of FA.

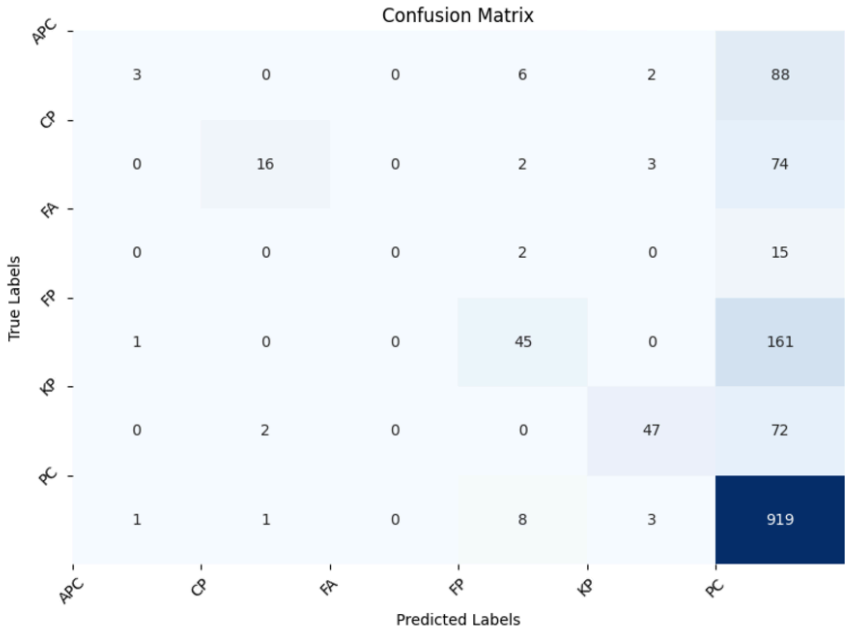
FP (0.33): The F1-score for FP is low, indicating some issues with precision and recall.

KP (0.53): The F1-score is moderate for KP, showing that the model performs better on this class but still has room for improvement.

PC (0.81): The F1-score for PC is quite good, reflecting the high recall and moderate precision for this class.

The reason for it failing to detect any False Alarms could be due to insufficient data, as the dataset only contains about 90 false alarms. I would want to get a larger or improved dataset so I could increase the precision score for Planetary Candidates, since I feel that is the most important one, as that is what we would want to detect the most if we added our own values to the dataset. It is interesting that overall the model is most successful at detecting Planetary Candidates, and it makes me wonder that the low scores for Known Planets, Ambiguous Planetary Candidates, and Confirmed Planets, are because the model is classifying those as Planetary Candidates instead, since there is likely an overlap between these classes, and as seen by the confusion matrix those classes are often misclassified as a Planetary Candidate instead. Though that doesn't explain its failure to detect False Alarms. Or its poor job at detecting False Positives. Though it may make sense that it was misclassifying false positives, since those would look like actual planets until further investigation is done into them to confirm they are not. This could be improved by using a larger dataset that contains more False Positives so the model has more data to compare.

Here is the confusion matrix for the Random Forest:



After analysing the performance of the Random Forest, I looked to see what features it considered most important, to see if it correlated with my own assumptions. I found that the most important feature was pl_tranmid, which is the midpoint of the time the stellar object moves in front of its host star. The other features are of all similar importance, though only half has much as pl_tranmid. The least important features were related to the luminosity and temperature of the host star. I thought those would be a bit more important, since the dimmer the star the harder it would be to detect the objects orbiting it. Pl_tranmid being the most important makes sense though. Since the time it takes the stellar object to move in front of its host star would be both easier to spot and faster for planets closer to the star, and it would be easier to detect and classify if they are actually planets as well. Also I believe that objects such as an asteroid tend to have a far longer pl_tranmid time than a planet, since they tend to have a longer orbit.

Next I used an SVM. I choose one as our other model since the TESS dataset has a high number of features, as SVM works well in such a case, and since our dataset includes imbalanced classes, as an SVM can handle class imbalance well by adjusting the class_weight parameter. I set the kernel to use rbfm or Radial Basis Function, since it works well for complex, non-linear relationships between features especially when the relationship between features is not clear or linear. I set C=1.0, as this balances the risk of overfitting and underfitting. I used gamma = 'scale' since it also provides a good balance between underfitting and overfitting.

To evaluate the SVM's performance, I used a confusion matrix and a classification report to see the precision, recall, and f1-scores.

The results were once again interesting, for precision I found that

APC (0.25): 25% of the instances predicted as APC are actually APC. This is relatively low, indicating that the model has a high number of false positives for this class.

CP (0.27): 27% of the instances predicted as CP are actually CP. Similar to APC, this is not great, indicating that a lot of CP instances are misclassified.

FA (0.07): Only 7% of the instances predicted as FA are actually FA. This is very low, suggesting a high number of false positives for this class.

FP (0.51): 51% of the instances predicted as FP are actually FP. This is a reasonable precision, but there's still room for improvement.

KP (0.33): 33% of the instances predicted as KP are actually KP. This is relatively low, indicating that the model is not very accurate when predicting KP.

PC (0.89): 89% of the instances predicted as PC are actually PC. This is very high, indicating that the model is performing well in predicting PC.

For recall I found:

APC (0.58): 58% of the actual APC instances are correctly identified. This is a better recall compared to precision, but there's still significant room for improvement.

CP (0.66): 66% of the actual CP instances are correctly identified. This is decent but still leaves 34% of CP instances undetected.

FA (0.24): Only 24% of the actual FA instances are correctly identified. This is quite low, indicating that the model is missing a large portion of the FA instances.

FP (0.46): 46% of the actual FP instances are correctly identified. The recall is moderate but needs improvement.

KP (0.69): 69% of the actual KP instances are correctly identified. This is a better recall compared to the other classes, but still leaves room for improvement.

PC (0.48): 48% of the actual PC instances are correctly identified. While this is decent, the model is missing half of the PC instances, which could be improved.

F1-Score gave:

APC (0.35): The F1-score is quite low for APC, reflecting the poor precision and moderate recall for this class.

CP (0.39): The F1-score for CP is low, indicating a trade-off between precision and recall, but the model's performance is not optimal for this class.

FA (0.11): The F1-score is very low for FA, reflecting the model's inability to detect this class properly.

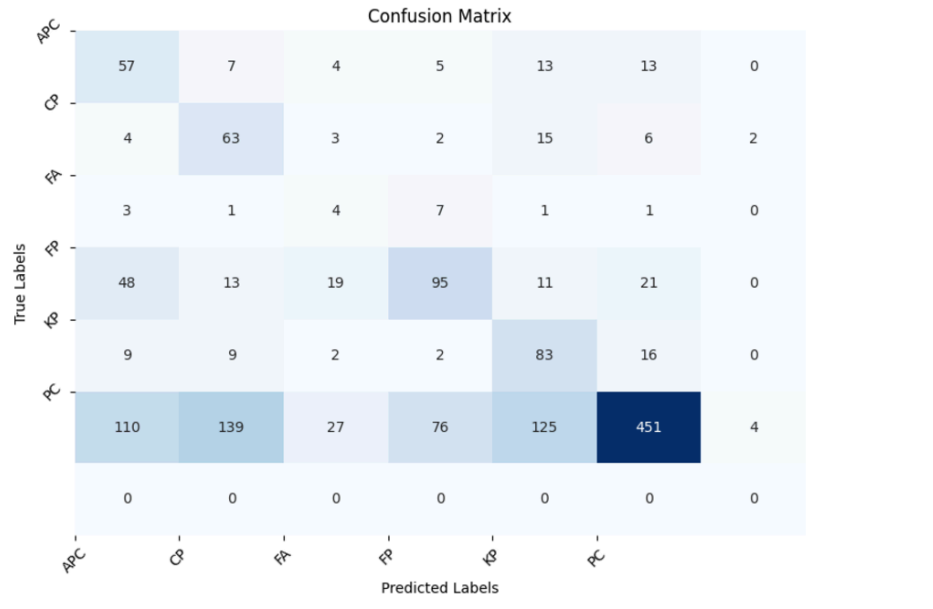
FP (0.48): The F1-score for FP is moderate, showing that the model has some balance between precision and recall, but there's still significant room for improvement.

KP (0.45): The F1-score for KP is moderate, showing that the model performs better for this class, but again, it is not excellent.

PC (0.63): The F1-score for PC is the highest, indicating a good balance of precision and recall, even though recall could still be improved.

In comparison with our Random Forest, the SVM is more successful at detecting False Alarms. This could be due to how SVM uses a kernel trick technique for transforming the data into higher dimensions, though I am not completely sure and would need to compare perhaps with other models or a larger similar dataset. It is interesting that the model was overall worse at detecting Planetary Candidates, and better at detecting Known Planets, Ambiguous Planetary Candidates, and Confirmed Planets. It made wonder if we are having the same issue but in reverse that I had with the Random Forest, with it mixing up these categories, as examining the confusion matrix I noticed that it was misclassifying Planetary Candidates as Known Planets, Ambiguous Planetary Candidates, and Confirmed Planets a lot more than with the Random Forest.

Here is the confusion matrix for the SVM:



To improve this I would want a larger dataset with this information, or to try other models as well to compare them. An issue with this dataset is that there are far more Planetary Candidates than the other categories, so one that has a better balance may be more useful.

In conclusion, I compared two different models for NASA's TESS planetary candidates dataset. A Random Forest and an SVM. The performance of the models was not perfect, and could use some improvement. The models struggled to detect False Alarms, likely due to the insufficient data, though the SVM did a better job at it. It seems that in both models the overlap between classes, such as Known Planets, Ambiguous Planetary Candidates, and Confirmed Planets, might be contributing to misclassifications, with these classes being misidentified as Planetary Candidates, as indicated by the confusion matrix.

For the importance of the features, the midpoint of the time a stellar object moves in front of its host star, was the most important feature. This made sense, as the transit time is easier to detect for planets closer to their stars, and planets typically have shorter transit times compared to asteroids. Other features have similar but lesser importance, while luminosity and temperature of the host star were found to be less important than expected. This suggests that the difficulty in detecting objects orbiting dimmer stars is not as significant in this model's predictions, which is interesting to know.

In overall comparison between the two models we tested, the SVM model performed better at detecting False Alarms compared to the Random Forest, potentially due to the kernel trick, which allows SVM to handle high-dimensional data better. However, the SVM model was worse at detecting Planetary Candidates, and performed better at detecting Known Planets, Ambiguous Planetary Candidates, and Confirmed Planets. This could indicate that the model is misclassifying Planetary Candidates as these other classes, similar to the issues seen with the Random Forest model, but in reverse, as shown by the two confusion matrices

To improve performance, especially for Planetary Candidates, a larger or more balanced dataset is likely needed. The current dataset has a far larger number of Planetary Candidates, which could be skewing the results. Exploring other models or comparing SVM with different settings may also provide better insights. Balancing the dataset might help the model better distinguish between these categories and improve classification accuracy. Perhaps the dataset could also benefit from excluding Confirmed and Known Planets, since they have such a large overlap with the Planetary Candidate class. Future improvements would involve me testing this as well, to see if that makes a difference.

Challenges & Future Work:

A major challenge I faced was misclassification in the models. I used both a Random Forest and an SVM, and they had a similar issue with classifying the categories Planetary Candidates, Known Planets, Ambiguous Planetary Candidates, and Confirmed Planets. The problem arises from what I believe to be an overlap between these classes. Since a Planetary Candidate would obviously have similar features as a Known Planet, and the same for Confirmed Planets and Ambiguous Planetary Candidates. It is unfortunate that I didn't consider this issue when I began the project, as I would have likely merged these categories, so future work on this would be to go back and when I was performing cleanup and normalization on the dataset to combine them. This should result in the model having a greatly improved performance.

Another challenge I faced was setting the data up to be able to be used in the model. The data contained variables that while not exactly categorical, were also not in a usable numerical format. These variables were in sexagesimal format, which is used for the calculation of time and angles. While this makes sense for analyzing planetary orbits, to be used in a model these variables had to be converted to decimal format instead. To do this I used functions given from the Astropy library, which while not difficult, did require research on how to convert the sexagesimal values to decimal.

One last challenge was that the target variable was categorical. Because of this I needed to use encoding to get them in a numerical format that was easier to use in my model. I couldn't use one hot encoding as there were 5 possible values for my target variable, and that would've proved difficult to parse. I used label encoding instead to convert each label to have a value of 1 to 5. This made it much easier to use, though when testing my model and looking at the performance, I did need to decode the values to get them back to their original for, as that was much easier that memorizing which value went to which number when the label encoding was done.

For future work I would try different models, to see how they compared to the two I tested during this project. I would also, as mentioned earlier, combine the overlapping categories to see if that helped with model performance. I would also like to try a new model as well, perhaps an XGBoost to see how that handles the relatively imbalanced classes, or to help see how it compares in analyzing feature importance compared to say how the Random Forest decided to analyze the feature importance. If possible I would also want to find a larger dataset to use, possibly I could try the same idea with NASA's K2 planetary candidate dataset. I think it

would be interesting to see how the models performed at classifying planets using that dataset as well, and compare the two.