CS 559-WS Final Report

Authors: Vincent Lin and Benjamin Tobio

1. Workflow Summary

We began our project with EDA and data pre-processing. From our analysis we found that HomePlanet, CryoSleep, Destination, and VIP were our categorical features and Age, RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck were our numerical features. The first thing we did was look at the distribution of the categorical and numerical features.

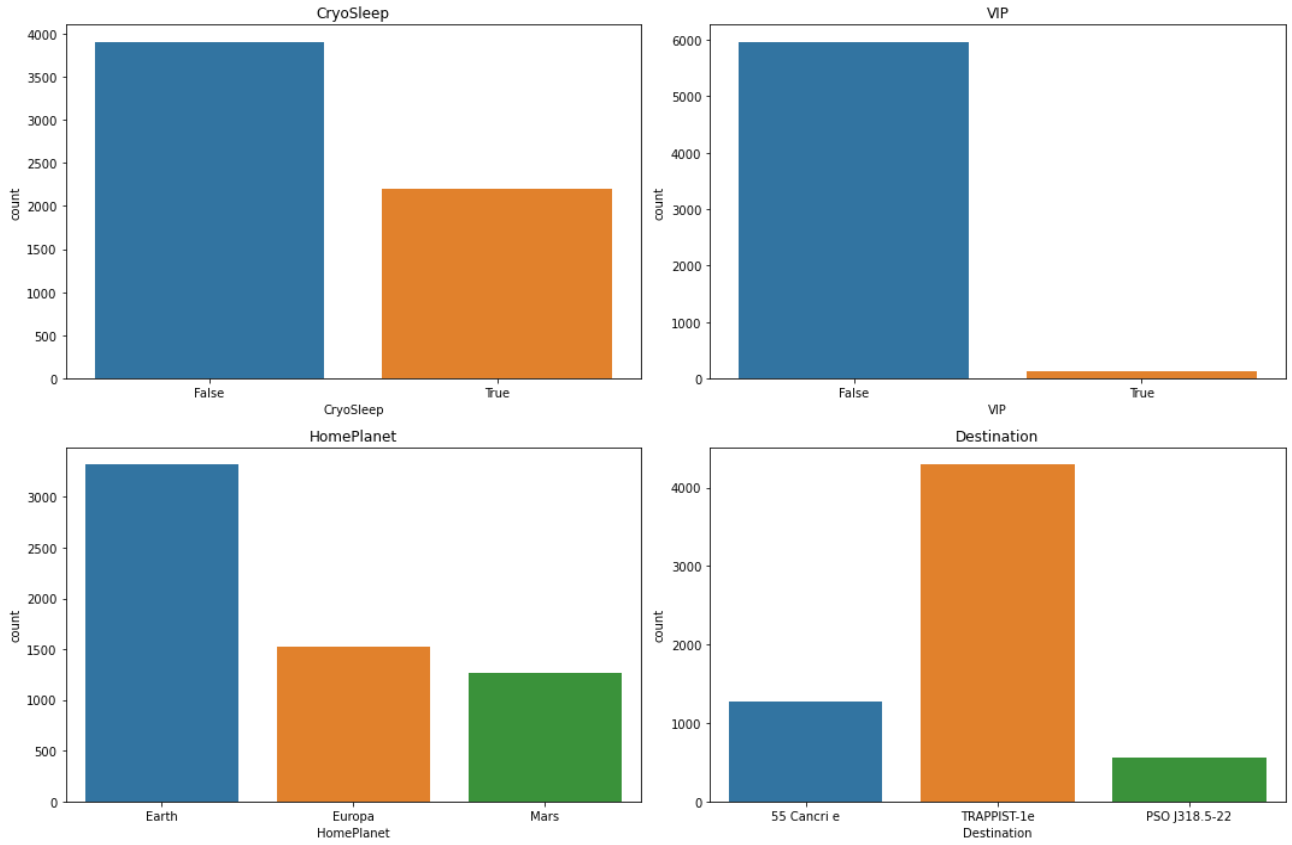

Figure 1.1: Distribution of Categorical Features

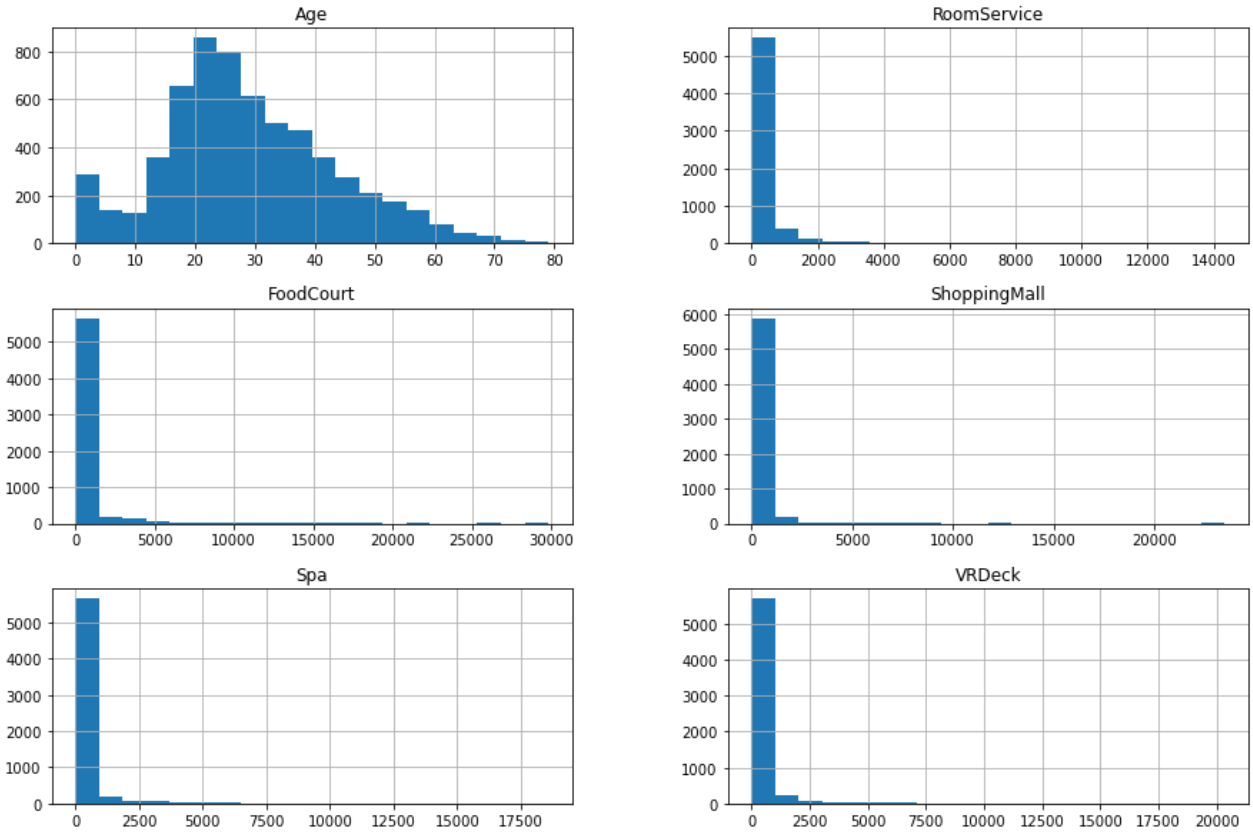

Figure 1.2: Distribution of Numerical Columns

As we can see from the distributions, each categorical feature has a dominant variable: False for CryoSleep, False for VIP, Earth for HomePlanet, and TRAPPIST-1e for Destination. For the numerical columns, Age is slightly skewed while RoomService, ShoppingMall, FoodCourt, Spa, and VRDeck are all heavily skewed with a majority of their values being zero. This information is very useful in terms of filling in the missing data points which we will discuss later.

We next performed some feature engineering on the data set. We first created a new feature called GroupSize by extracting each passenger's group number from their PassengerId. Using this, we counted the number of passengers with that group number and created the GroupSize feature. Next, we broke the Cabin feature into three individual features: Deck, Num

(cabin number), and Side. This made it possible to work with the data originally stored in the Cabin feature. Finally, we created a new feature called TotalSpent by aggregating RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck.

The last step of our EDA and pre-processing workflow was handling the missing values in the dataset. For this, we first looked at how many missing values we had for each feature.

```
PassengerId       0
HomePlanet      142
CryoSleep       156
Cabin           144
Destination     122
Age             130
VIP             152
RoomService     126
FoodCourt       126
ShoppingMall    151
Spa             118
VRDeck          129
Name            145
Transported       0
```

As we can see, each feature column contains just over 100 missing values (about 2% of each column). We then looked at the distribution of the missing data to see if there was a pattern of the missing data or if the missing values are relatively random. To do this we utilized the MissingNo package.

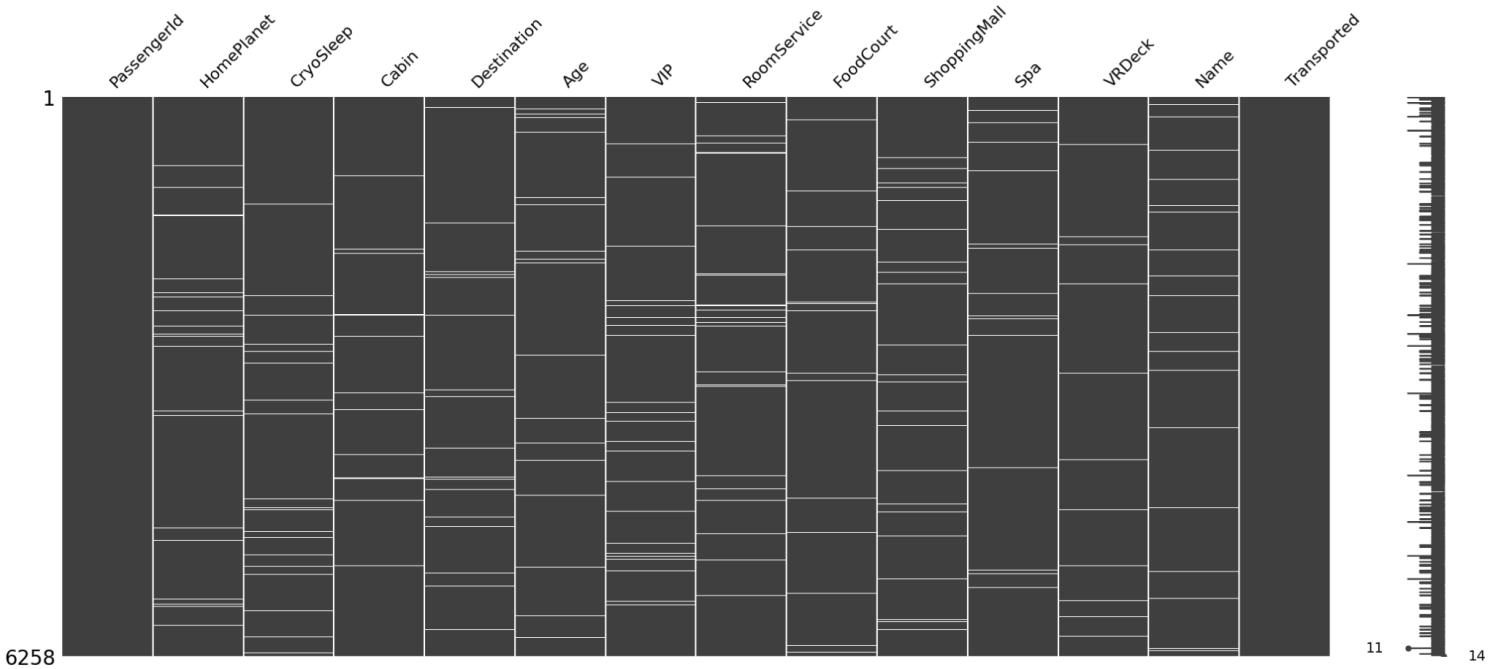

Figure 1.3: MissingNo Plot

We can see that the missing values are relatively randomly distributed, and since the percentage of missing values is so low for each feature, we are going to need to figure out how to manage these missing values. This is why looking at the distributions of each feature was helpful. Since each categorical feature had a dominant value, we can fill the missing values with the mode of that feature. And for the numerical features, we can just fill the missing values with the median of that feature. Finally, we converted HomePlanet, Destination, Deck, and Side to dummy variables and scaled the numerical features using Sklearn's StandardScaler.

2. Model Implementation and Results

For this project we implemented three parametric models (logistic regression, linear support vector machine, and a simple neural network), three non-parametric models (random forest, gradient boosted classifier, and XGBoost), and a stacked model. Including the dummy

variables, all models were built on 29 features. The following code snippet shows an example of each model's hyperparameter tuning and implementation.

```
xgb_param_grid = {'eta':[.04, .05, .06],
                  'max_depth':[5],
                  'gamma':[.03, .035, .04],
                  'subsample':[.35, .4, .45]}

xgb = xgboost.XGBClassifier()
xgb_grid = GridSearchCV(xgb, xgb_param_grid, scoring='accuracy', cv = 5, n_jobs=-1)
xgb_grid.fit(valid_X, valid_y)

xgb_tuned = xgboost.XGBClassifier(**xgb_grid.best_params_)
xgb_tuned.fit(train_X, train_y)

train_pred = xgb_tuned.predict(train_X)
test_pred = xgb_tuned.predict(test_X)
print(f'Train Accuracy: {accuracy_score(train_y, train_pred)}')
print(f'Test Accuracy: {accuracy_score(test_y, test_pred)}')
```

✓ 18.5s

```
Train Accuracy: 0.8410035155001598
Test Accuracy: 0.8154111558366878
```

Figure 2.1: XGBoost Model Tuning, Implementation, and Score

The test accuracy for each model are as follows:

Model	Logistic Regression	Linear SVM	Simple NN	Random Forest	Gradient Boosted	XGBoost	Stacked
Accuracy	80.16%	80.97%	81.83%	80.85%	81.48%	81.54%	81.71%

The simple neural network was our best performing model overall while logistic regression was our worst performer. The XGBoost was our best performing non-parametric model while the random forest model was the worst non-parametric performer. Our stacking model performed well, however it was actually slightly worse than the neural network on its own.

3. Group Member Work Distribution

Both group members did a similar amount of work on this project. We both performed EDA and pre-processing on the data and shared our results with one another. We each implemented all six models on our own and compared results. Finally, we both worked on the presentation and this report.

4. Kaggle Competition Screenshot

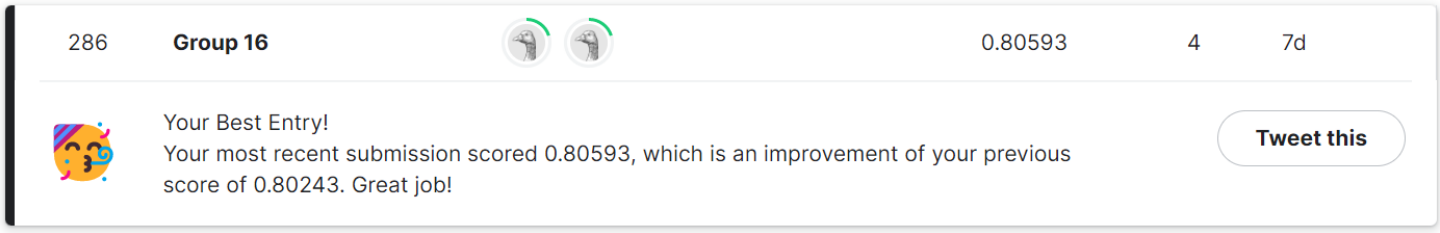

Figure 4.1: Kaggle Competition Ranking Screenshot