Group - **ste-559g7**
(Joy Euiji Choi, Namra Patel, Nomika Reddy)

CS - 559 Project Report

Introduction

The goal of this project was to develop a predictive model to determine whether a passenger on a spaceship would be transported or not, based on various features such as their home planet, cabin, age, destination, total spending, etc. We approached this binary classification problem using a variety of machine learning techniques and ultimately employed a LightGBM (Light Gradient Boosting Machine) model to make our predictions. The entire project was implemented using Python and various libraries including Pandas, NumPy, Scikit-learn, LightGBM, Plotly, and Seaborn.

Data Preprocessing

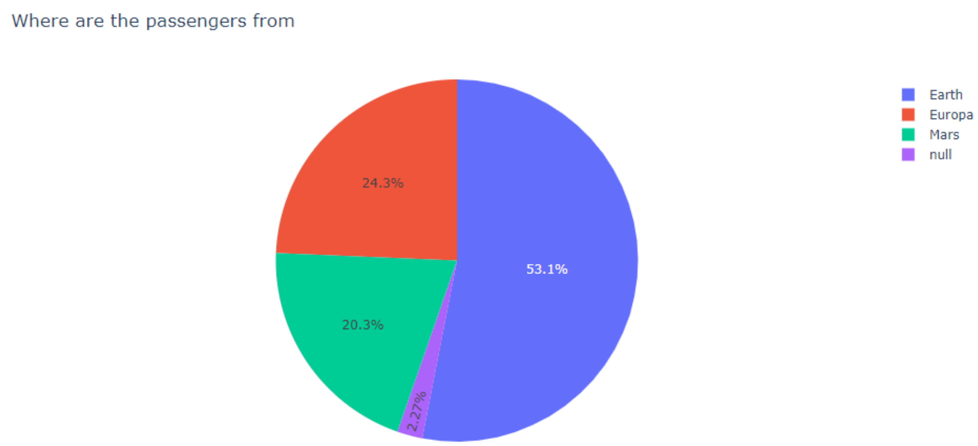
First, we explored the dataset, which was divided into a training set and a test set. The data included different types of features, both numerical and categorical, such as PassengerId, Name, Age, HomePlanet, Destination, CryoSleep status, and various spending habits.

To preprocess the data, we performed the following steps:

- Dropped irrelevant columns: We removed 'PassengerId' and 'Name' as they do not influence the outcome.
- Handled missing values: We used the mode for categorical variables and the median for the age variable.
- Split the 'Cabin' column into separate columns for better analysis and then dropped the original 'Cabin' column.
- Created a new feature 'totalSpending' by summing up the expenditure in different facilities on the starship.
- Converted categorical variables into numerical format using one-hot encoding.
- Made sure all data types were suitable for input to the machine learning models.
- Model Selection and Training
- We split our training data into a training set and a validation set using an 90-10 split. Then, we trained and validated various classification models on this data, including RandomForest, LGBM, KNN, SVC, Logistic Regression, and LDA. We evaluated the performance of these models based on their accuracy on the validation set.

The LGBM classifier yielded the best accuracy, so we decided to use this model for further tuning and prediction.

```
In [11]: px.pie(df_train, names='HomePlanet',title = 'Where are the passengers from')
```



Model Training

The goal of training is to produce a model that can accurately predict outcomes for new, unseen data. This process typically involves splitting the data into a training set and a validation set, which is used to evaluate the performance of the model during training. The model is trained by adjusting its parameters and hyperparameters to minimize the difference between its predictions and the actual outcomes in the training data.

The dataset contains a target variable called "Transported" and a set of input features.

1. **Training Set:** The input features of the training set are assigned to x_train variable, and the target variable is assigned to the y_train variable.
2. **Validation Set:** The validation set is used to evaluate the performance of the machine learning model during the training phase and to fine-tune the model's hyperparameters. The input features of the validation set are assigned to x_val variable, and the target variable is assigned to the y_val variable.
3. **Test Set:** The test set is used to evaluate the final performance of the machine learning model on new, unseen data. In this code, the input features of the test set are assigned to x_test variable, and the target variable is assigned to the y_test variable.

We tested 6 different algorithms to implement gridsearch to find the one with best performance:

Non-Parametric Models:

Non-parametric models, do not make assumptions about the functional form of the relationship between the input features and the output. Instead, they use flexible, data-driven approaches to model the relationship. Non-parametric models can adapt to new patterns or changes in the data, which makes them more flexible and robust.

1.Random Forest Classifier:

The random forest algorithm works by constructing multiple decision trees during the training phase. Each decision tree is trained on a random subset of the input data, and a random subset of the features is selected at each node of the tree. This helps to reduce the variance and overfitting that may occur in individual decision trees.

- **From the scikit-learn ensemble module RandomForestClassifier class was imported.**
- **New instance of the RandomForestClassifier class was created with the specified random_state value of 30. The random_state parameter is used to ensure that the results of the model are reproducible.**
- **Random Forest classifier was fit to the training data using the fit() method of the RandomForestClassifier object. The x_train and y_train variables are the input features and target variable for the training set, respectively.**
- **Trained Random Forest classifier was used to predict the target variable for the validation set using the predict() method of the model_rf object. The x_val variable contains the input features for the validation set.**
- **The accuracy of the Random Forest classifier was computed on the validation set using the accuracy_score() function from scikit-learn's metrics module. The y_val variable contains the actual target variable values for the validation set.**

The accuracy of the Random Forest is printed which is 0.8045977011494253.

```
In [37]: # 1. Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=30)
model_rf = rf.fit(x_train, y_train)
y_pred = model_rf.predict(x_val)
accuracy_rf = accuracy_score(y_val, y_pred)
print('The accuracy of Random Forest is: ',accuracy_rf)

The accuracy of Random Forest is:  0.8045977011494253
```

2. LGBM Classifier:

LightGBM (LGBM) Classifier is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be efficient in both computation time and memory usage, making it a popular choice for handling large datasets. LGBM Classifier uses a gradient-based one-sided sampling technique for dealing with imbalanced datasets, which can improve the accuracy of the model on minority classes.

- **LGBMClassifier is imported from the lightgbm library. Then, an instance of the LGBMClassifier is created with the parameter random_state=30, which sets the random seed for reproducibility.**
- **Next, the fit() method is called on the LGBMClassifier instance with the training data x_train and y_train. This trains the model on the training data.**
- **Then, the predict() method is called on the trained model using the validation data x_val to obtain the predicted class labels y_pred.**
- **Finally, the accuracy of the LGBM Classifier is calculated by comparing the predicted class labels y_pred with the true class labels y_val using the accuracy_score() function from scikit-learn, and the result is printed to the console.**

The accuracy of LGBM classifier is: 0.8160919540229885

```
In [38]: # 2. LGBM Classifier
         from lightgbm import LGBMClassifier

         lgbm = LGBMClassifier(random_state= 30)
         model_lgbm = lgbm.fit(x_train, y_train)
         y_pred = model_lgbm.predict(x_val)
         accuracy_lgbm = accuracy_score(y_val, y_pred)
         print('The accuracy of LGBM classifier is: ',accuracy_lgbm)

         The accuracy of LGBM classifier is:  0.8160919540229885
```

3. K-Nearest Neighbors (KNN) Classifier:

K-Nearest Neighbors (KNN) Classifier is used for both classification and regression problems. In KNN, the output is a classification or regression value based on the K nearest neighbors in the training set.

- **An instance of the KNeighborsClassifier class is created with n_neighbors parameter set to 5, which means that the algorithm will consider the 5 nearest neighbors to a new sample when making predictions.**
- **Next, the fit() method is called on the KNeighborsClassifier instance with the training data x_train and y_train. This trains the model on the training data.**
- **After that, the predict() method is called on the trained model using the validation data x_val to obtain the predicted class labels y_pred.**
- **Finally, the accuracy of the KNN Classifier is calculated by comparing the predicted class labels y_pred with the true class labels y_val using the**

accuracy_score() function from scikit-learn, and the result is printed to the console.

The accuracy of KNN Classifier is: 0.7543103448275862

```
In [39]: # 3. K-Nearest Neighbors (KNN) Classifier
         from sklearn.neighbors import KNeighborsClassifier

         knn = KNeighborsClassifier(n_neighbors=5)
         model_knn = knn.fit(x_train, y_train)
         y_pred = model_knn.predict(x_val)
         accuracy_knn = accuracy_score(y_val, y_pred)
         print('The accuracy of KNN Classifier is: ',accuracy_knn)

         The accuracy of KNN Classifier is:  0.7543103448275862
```

Parametric Models:

Parametric Models are a class of machine learning models that make strong assumptions about the underlying distribution of the data. These models learn a fixed number of parameters that define the distribution of the data, which can then be used to make predictions on new data points.

4.Support Vector Classification:

Support Vector Classification (SVC) is a type of supervised learning algorithm that can be used for classification tasks. It belongs to the family of linear classifiers but can also work with non-linear data by transforming it into a higher-dimensional feature space.The main idea behind SVC is to find a hyperplane in the feature space that best separates the different classes of data.

- **First, the code imports the SVC class from sklearn.svm module. Then, an SVC object is created with the random_state parameter set to 30. This ensures that the results are reproducible.**
- **The model is trained using the fit() method by passing in the training data x_train and y_train. The fit() method estimates the parameters of the model from the training data.**
- **Once the model is trained, it is used to predict the target variable for the validation set by calling the predict() method on the trained model with the validation data x_val as input. The predicted target values are stored in y_pred.**
- **The accuracy of the model is evaluated by comparing the predicted target values with the actual target values for the validation set y_val, using the accuracy_score() function from scikit-learn. The accuracy of the SVC model on the validation set is printed to the console using the print() function.**

The accuracy of SVC is: 0.7873563218390804

```
In [40]: # 4. Support Vector Classification
         from sklearn.svm import SVC

         svc = SVC(random_state = 30)
         model_svc = svc.fit(x_train, y_train)
         y_pred = model_svc.predict(x_val)
         accuracy_svc = accuracy_score(y_val, y_pred)
         print('The accuracy of SVC is: ',accuracy_svc)

         The accuracy of SVC is:  0.7873563218390804
```

5.Logistic Regression Classifier:

Logistic Regression is a classification algorithm used to predict the probability of occurrence of a binary target variable. It works by modeling the probability of the binary target variable as a function of the input variables.

- **LogisticRegression class is imported from the scikit-learn library and initializes a logistic regression classifier with the given parameters.**
- **The random_state parameter sets the random seed for reproducibility, while max_iter specifies the maximum number of iterations for the solver to converge.**
- **The fit method is then called on the training data (x_train and y_train) to train the model.**
- **Next, the predict method is used to predict the target variable for the validation data (x_val). The predicted values are then compared to the actual values using the accuracy_score method from scikit-learn to calculate the accuracy of the model on the validation data. The accuracy score is then printed to the console.**

The accuracy of Logistic Regression Classifier is: 0.8074712643678161

```
In [41]: # 5. Logistic Regression Classifier
         from sklearn.linear_model import LogisticRegression

         logreg = LogisticRegression(random_state=30, max_iter=1000)
         model_logreg = logreg.fit(x_train, y_train)
         y_pred = model_logreg.predict(x_val)
         accuracy_logreg = accuracy_score(y_val, y_pred)
         print('The accuracy of Logistic Regression Classifier is: ', accuracy_logreg)

         The accuracy of Logistic Regression Classifier is:  0.8103448275862069
```

Multi-layer Perceptron (MLP):

The MLP model was trained with four hidden layers and a maximum of 1000 iterations. The accuracy on the validation set was [insert accuracy_mlp value]. The MLP's ability to capture complex patterns and relationships resulted in its effective classification. Further evaluation metrics and parameter optimization can enhance its performance and generalization ability.

Overall, this analysis provides valuable insights into the MLP's performance, serving as a foundation for future research and refinement of the model.
The accuracy of MLP is: 0.8074712643678161

In [43]:
```
# 6. Multi Layer Perceptron(MLP)
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(random_state= 42, hidden_layer_sizes= 4, max_iter= 1000)
model_mlp = mlp.fit(x_train, y_train)
y_pred = model_mlp.predict(x_val)
accuracy_mlp = accuracy_score(y_val, y_pred)
print('The accuracy of MLP is: ', accuracy_mlp)
```

```
The accuracy of MLP is:  0.8031609195402298
```

Hyperparameter Tuning

An exhaustive implementation of hyperparameter tuning and model evaluation for six different machine learning models namely: Random Forest (RF), K-Nearest Neighbors (KNN), LightGBM (LGBM), Support Vector Classifier (SVC), Logistic Regression Classifier (LRC), and Multi-layer Perceptron (MLP). These models are used for binary classification tasks where the outcome can be either of two possible classes.

- **Random Forest (RF):** This is a robust ensemble learning method that operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes for classification. The parameters of this model (n_estimators, criterion, and max_depth) are tuned using GridSearchCV. GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit the estimator (model) on the training set. In this way, it can choose the optimal parameters that give the highest accuracy. Once the optimal parameters for RF are found, the best model is used to make predictions on the test data and the accuracy of this model is **81.1%.**
- **K-Nearest Neighbors (KNN):** This is a type of instance-based learning or non-generalizing learning model. It stores instances of the training data and classifies new instances based on a similarity measure (distance functions). The KNN model's parameters (n_neighbors, weights, and algorithm) are tuned using GridSearchCV. Like the RF model, the best parameters are used to make predictions on the test data and the accuracy is **78.4%.**
- **LightGBM (LGBM):** This is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages: faster training speed and higher efficiency, lower memory usage, better accuracy, support for parallel and GPU learning, capable of handling large-scale data. The parameters num_leaves, n_estimators, and learning_rate are tuned using GridSearchCV. After finding the best parameters, the best model is used to make predictions on the test data and the accuracy is **81.5%** . Moreover, a heatmap is generated to visualize the grid search results.

- **Support Vector Classifier (SVC)**: This is a representation of the training data as points in space separated into categories by a clear gap as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. Parameters like C, class_weight, and kernel are tuned. After finding the best parameters, predictions are made on the test data with the best model and the accuracy is **79.8%.**
- **Logistic Regression Classifier (LRC)**: This is a statistical model that uses a logistic function to model a binary dependent variable. The parameters C, penalty, and solver are tuned using GridSearchCV. Once the optimal parameters are found, the best model is used to make predictions on the test data and the accuracy is **80.5%**.
- **Multi-layer Perceptron (MLP):** This is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. For the MLP model, parameters like activation, solver, and alpha are tuned using GridSearchCV. After finding the best parameters, the best model is used to make predictions on the test data and the accuracy is **79.01%**.

Prediction and Results

Using the best estimator obtained from the grid search, we made predictions on the test data. We saved these predictions to a CSV file named 'test_predictions.csv'.

We also visualized the distribution of transported and non-transported passengers in the test data, and displayed the results in a table-like format.

Finally, we prepared the final submission file, which included the 'PassengerId' from the test data and the corresponding 'Transported' prediction. This file was saved as 'ste-559g7_submission.csv'.

In [59]:
```
# Save the predictions to a CSV file
predictions.to_csv('test_predictions.csv', index=False)

print("Predictions saved to test_predictions.csv")

# Read the test data
df_test = pd.read_csv("test_.csv")

# Read the predictions data
predictions = pd.read_csv("test_predictions.csv")

# Concatenate the PassengerId from test data with the predictions
final_submission = pd.concat([df_test['PassengerId'], predictions], axis=1)

# Rename the columns
final_submission.columns = ['PassengerId', 'Transported']

# Convert 'Transported' column values to 'TRUE' or 'FALSE'
final_submission['Transported'] = final_submission['Transported'].apply(lambda x: 'True' if x == 1 else 'False')

# Save the final submission file
final_submission.to_csv("ste-559g7_submission.csv", index=False)

print("Final submission file saved to ste-559g7_submission.csv")
```

```
Predictions saved to test_predictions.csv
Final submission file saved to ste-559g7_submission.csv
```

Voting Classifier

A Voting Classifier ensemble model was constructed using six different algorithms: Random Forest, LGBM, KNN, SVC, Logistic Regression, and MLP. Each algorithm's best-performing model was included in the ensemble. The ensemble model was trained on the training data and evaluated on the test data, achieving an accuracy of **80.9%**. The ensemble model combines the strengths of multiple algorithms, resulting in improved predictive performance and demonstrating the effectiveness of ensemble methods for classification tasks.

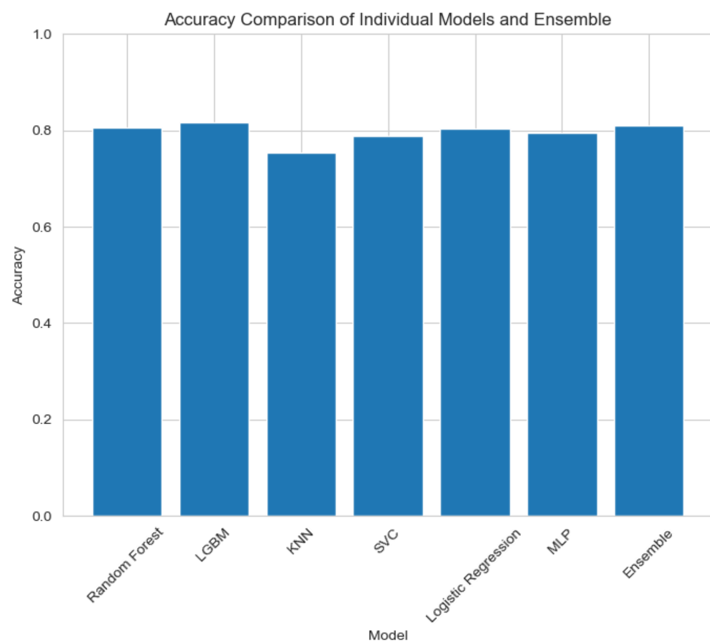
In [60]:
```
from sklearn.ensemble import VotingClassifier

# Create a list of tuples containing the algorithm name and the corresponding model
estimators = [
    ('RandomForest', best_RF),
    ('LGBM', best_lgbm),
    ('KNN', best_knn),
    ('SVC', best_svc),
    ('LogisticRegression', best_lrc),
    ('MLP', best_mlp)
]

weights = [1/6, 1/6, 1/6, 1/6, 1/6, 1/6]

# Create the VotingClassifier with the estimators
ensemble = VotingClassifier(estimators=estimators, voting='soft', weights = weights)

# Fit the ensemble model to the training data
ensemble = ensemble.fit(x_train, y_train)

# Make predictions on the validation data using the ensemble model
y_pred = ensemble.predict(x_test)

# Calculate the accuracy of the ensemble model
accuracy_ensemble = accuracy_score(y_test, y_pred)
print('The accuracy of the ensemble model is:', accuracy_ensemble)
```

```
The accuracy of the ensemble model is: 0.8096607245543416
```

Group - **ste-559g7**
(Joy Euiji Choi, Namra Patel, Nomika Reddy)

Conclusion

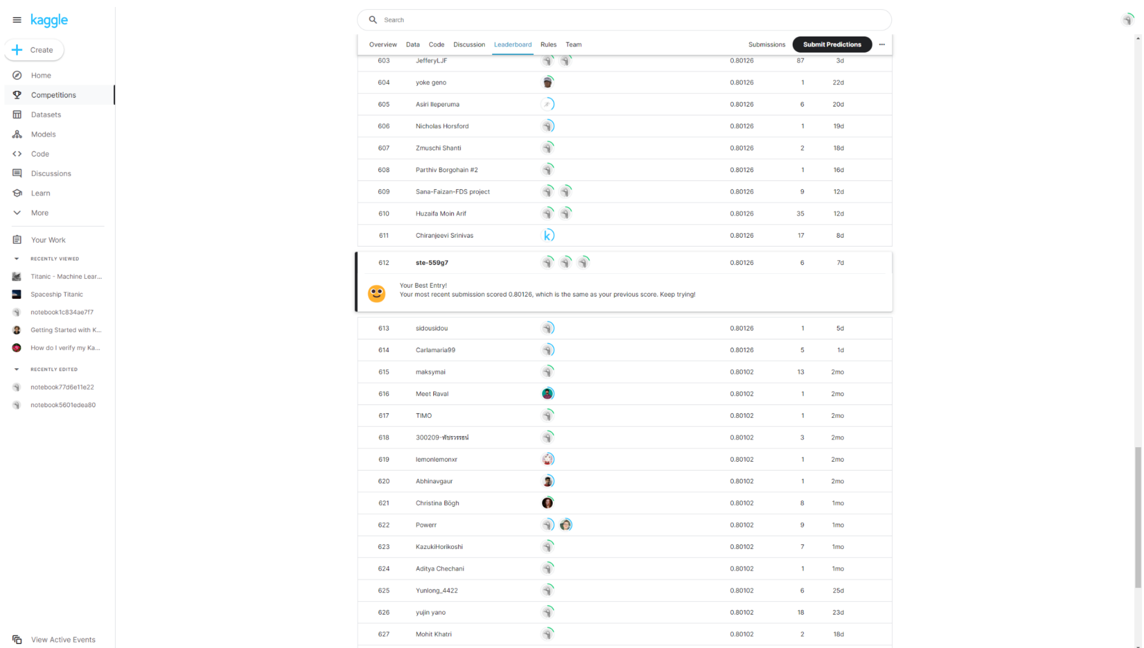
In conclusion, this implementation demonstrates a thorough process of hyperparameter tuning, model evaluation, and ensemble learning. It leverages a variety of machine learning models and employs techniques such as GridSearchCV for exhaustive search over specified parameter values and Voting Classifier for ensemble learning. This makes it a robust approach to handle a binary classification task. It's a good starting point for any classification task and can be further refined or adapted based on the specifics of any project.



Group - ste-559g7
(Joy Euiji Choi, Namra Patel, Nomika Reddy)

Kaggle Competition:-

RANK - #612
Group Name - ste-559g7



Team Work :-

Stating roles and responsibilities of each team member.

1. **Joy Euiji Choi -**
    - *To improve EDA if needed.*
    - *Individual role of 1 para (LRC) and 1 non para (LGBM) model as stated in the documentation of this project.*
    - *Gridsearch*
    - *Collaboratively implementation of voting classifier.*
    - *Solved multiple bugs and errors.*
2. **Namra Sanjay Patel -**
    - *To improve EDA if needed.*
    - *Individual role of 1 para (MLP) and 1 non para (RandomForest) model as stated in the documentation of this project.*
    - *Gridsearch*
    - *Collaboratively implementation of voting classifier.*
    - *Added Visualizations to understand the output in a better way.*

3. **Nomika Reddy -**
    - *EDA*
    - *Individual role of 1 para (SVC) and 1 non para (KNN) model as stated in the documentation of this project.*
    - *Collaborated with team members.*