Transported or Not: Enhancing Rescue Operations for the Spaceship Titanic Dimensional Drift

Group Members: Kevin Li, Siyan Li, Xinran Wang, Yumin Zhang

Introduction

This project aims to solve a mysterious problem: in the year 2912, *the Spaceship Titanic* was traveling in the sea of stars to its destinations, three newly habitable exoplanets orbiting nearby stars. Sadly, it faced the same tragedy as its namesake 1000 thousand years ago: while the spaceship remained intact, almost half of its passengers were lost to an alternate dimension. In order to save both the current passengers and passengers in the future, as data scientists, our group needs to build predictive models to predict what kind of passengers are likely to be lost to another dimension, using the data records of the current passengers.

At first glance, this mystery problem may seem to be unrelated to real-world applications, as we are trying to solve a fictional prediction problem. However, we believe this project has a significant underlying realistic impact. In the real world of predictive modeling, it is very common that the target value and features seem not to be very related according to common sense, and even worse, some features may also remain concealed inside the raw data records. In such scenarios, the responsibility of data scientists is to try to figure out possible related features, extract and format out useful features from the raw data through EDA, and subsequently choose suitable models and fine-tune the hyperparameters. We believe that through this project, we will be able to build a systematic approach applicable to not only this fictional problem but also other similar predictive tasks.

Data Identification and Exploration

The initial stage of our analysis began with a comprehensive exploration of the Spaceship Titanic dataset to understand its structure, identify patterns, and uncover potential anomalies. This foundational work aimed to predict passengers' transportation status, leveraging a blend of personal and trip-related information as predictors within a carefully structured supervised learning framework.

Raw Data

This dataset, comprising 8693 rows and 14 columns, encapsulates detailed passenger information aboard the fictional Spaceship Titanic. Each row represents one passenger, featuring a range of

attributes including PassengerId, HomePlanet, CryoSleep status, Cabin, Destination, Age, VIP status, and expenditures on different services such as RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck, alongside the passenger's Name. Most crucially, it includes the Transported column, serving as the response variable for the supervised learning model, which indicates whether the passenger was transported to another dimension.

EDA

Missing Value: The EDA began with a thorough examination of missing values across the dataset, revealing patterns of data absence that could impact model performance. Techniques such as sorting missing values and visualizing them with the missing matrix provided a clear picture of data completeness, guiding our strategy for handling missing data. To address missing values, we employed median imputation for numeric variables and mode imputation for categorical variables, ensuring a robust dataset ready for further analysis.

Numerical Data Distribution: The analysis proceeded with a deep dive into the dataset's summary statistics and distributions, particularly focusing on the age distribution of passengers and their expenditures on various services aboard the spaceship, such as RoomService, FoodCourt, ShoppingMall, Spa, and VRDeck. We observed a significant skew in the distribution of all expenditure-related variables and decided to apply a log transformation to these expenditure variables.

Target Data Distribution: Our data set is balanced in terms of target value distribution, which is about 1:1.

Categorical Data Exploration: We delved into categorical variables including PassengerId, Cabin, and Name, exploring their patterns with the objective of extracting meaningful insights from these concatenated fields.

Feature Engineering

New Feature Creation:

- Total Expenditure: We introduced a new feature, Expenditure, representing the sum of all expenditures (RoomService, FoodCourt, ShoppingMall, Spa, VRDeck) by a passenger. This aggregated measure provides a holistic view of a passenger's spending behavior on board.
- No Spending Indicator: Alongside, we crafted a binary feature, No_spending, to flag passengers with zero expenditure across all amenities. This feature helps in identifying

passengers who did not utilize any paid services aboard, potentially indicating different passenger segments.

- Passenger Groups and Sizes: From the PassengerId field, we extracted information regarding passenger groups and group sizes. Additionally, we introduced a new feature named "Solo" to represent the most common value observed in the group size attribute.
- Cabin Details: The Cabin field was dissected to extract the deck, number, and side of the ship, offering insights into the spatial distribution of passengers and their potential preferences or socioeconomic status.
- Family Names: By parsing the Name field, we identified family names, allowing us to group passengers by families and explore familial ties' impact on the transportation outcome.

Feature Selection:

We removed the features PassengerId, Group_size, Cabin, and Name from the dataset after deriving new features from these variables. Including these original features could potentially lead to multicollinearity, given their direct correlation with the newly created attributes. Additionally, the Group feature was excluded due to its high cardinality, with 6217 unique values out of 8693 rows, as its presence could increase the risk of overfitting our model by overly fitting the noise in the training data rather than identifying underlying patterns.

Data Transformation, Standardization, and Encoding:

To address skewness in expenditure distributions, log transformations were applied to normalize these variables, enhancing model interpretation. Categorical variables were transformed using label encoding, and numerical values were standardized to a common scale.

Models

We implemented 6 different models for prediction: Logistic Regression, SVC, Decision Tree & Random Forest, Neural Network, LGBMclassifier, and XGBoost.

Logistic Regression

Logistic Regression is one of the most basic statistical methods used for binary classification, where it models the probability that a given input belongs to one of two possible classes, employing a logistic function to map inputs to outputs.

We used the linear kernel and the training process only took 0.04 sec. It yields a validation accuracy of 76.94% and an AUC of 0.85, denoting a moderate predictive capability.

Support Vector Classifier (SVC)

SVC is a supervised machine learning algorithm that finds the optimal hyperplane in a high-dimensional space to classify data points into different classes, aiming to maximize the margin between them.

The training process took 10.81 sec, which is much longer than logistic regression. The validation accuracy is 78.49% and the AUC score is 0.87, denoting a slightly better predictive capability than logistic regression.

Decision Tree and Random Forest

A Decision Tree is a hierarchical model that splits the dataset into subsets based on the value of input features, aiming to classify or predict the target variable. Random Forest, on the other hand, is an ensemble learning method that constructs multiple decision trees and combines their predictions to improve accuracy and reduce overfitting.

In the process of hyperparameter tuning and validation for classification models, the Decision Tree's optimal depth was identified as 7 after assessing depths from 1 to 20 through exhaustive GridSearchCV with Stratified 10-fold cross-validation, yielding a validation accuracy of 78.21%. The training process took 0.06 sec. For the Random Forest model, RandomizedSearchCV with 100 iterations over the same cross-validation strategy was employed to tune parameters such as the number of estimators, maximum features, tree depth, minimum samples for a split, and minimum samples per leaf.

The best-performing Random Forest model, with a depth of 10, no restriction on maximum features, a minimum of 5 samples per leaf, and 15 samples for a split, surpassed the Decision Tree with a validation accuracy of 79.87%. The training process took 6.15 sec. The effectiveness of the Random Forest model was further underscored by a ROC curve reflecting an AUC of 0.89, denoting a strong predictive capability.

Neural Network

A neural network (NN) is a computational model inspired by the structure and function of the human brain, composed of interconnected nodes arranged in layers. It's trained using algorithms to learn complex patterns in data and make predictions or decisions.

We used one of the most simplified versions of NN which only contains one hidden layer and sigmoid activation function. The hyperparameters of hidden layer size were tuned using the same 10-fold cross-validation. The best hidden size was 10. The NN yields an accuracy of 78.43% and an AUC score of 0.88, denoting a strong predictive capability. The training process took 3.79 sec, which is also relatively fast.

LGBMclassifier

LGBMClassifier stands for LightGBM Classifier. LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed for distributed and efficient training, particularly suited for large datasets.

Similar to previous modeling, we used cross-validation on the same folds to fine-tune the LGBMclassifier, and we found that the best combination of hyperparameters is boosting_type of gbdt, feature fraction of 0.6, learning rate of 0.05, and num leaves of 75. With these optimized parameters, the model achieves a mean cross-validation accuracy of 80.73% and AUC of 0.91. The training process only took 0.56 sec.

XGBoost

XGBoost, which stands for eXtreme Gradient Boosting, is a popular open-source machine learning library known for its high performance and effectiveness in handling structured data. It is an implementation of gradient boosted decision trees designed for speed and performance.

The optimized combination of hyperparameters is learning rate of 0.1, maximum depth of 9, and number of estimators of 200. Similar performance is also achieved, with a mean cross-validation accuracy of 80.68% and AUC of 0.90. The training process only took 4.20 sec.

Results

Model Comparison

The accuracies, AUC score, and time for the training process for each model are summarized in **Table 1**. Among all the models, the LGBMclassifier shows the highest accuracy and AUC score, with a relatively quick training process.

The ROC curves from different models are shown in **Figure 1**. The ROC curve with the best AUC of 0.91 shows that the model has an excellent ability to discriminate between the positive class (transported) and the negative class (not transported). The precision for predicting 'False' (not transported) is 0.82, meaning that when the model predicts a passenger is not transported, it is correct 82% of the time. Conversely, the precision for 'True' (transported) is slightly lower at 0.79. Recall measures the proportion of actual positives that were correctly identified. Here, the recall for 'True' is 0.83, meaning the model correctly identifies 83% of passengers who were actually transported. The recall for 'False' is 0.78, indicating that 78% of passengers who were not transported were correctly identified by the model.

A false positive, in this case, would mean that a passenger was predicted to be transported but was actually not. Depending on the context, this could mean allocating resources (such as rescue operations or investigations) to someone who doesn't need them, potentially wasting resources. A false negative would mean that a passenger was predicted to not be transported but actually was. This is usually more serious, as it could mean overlooking a passenger in need, possibly resulting in missing a person who might require urgent attention or follow-up actions.

Feature Importance Analysis

Based on the provided feature importance charts from six different models applied to the Spaceship Titanic dataset, shown in **Figure 2**, we can draw several conclusions about which features are most influential in predicting outcomes, as well as how feature importance rankings differ across models.

Common Trends:

- Across all models, No_Spending (or its variations like Expenditure in some charts) is consistently identified as a key feature, suggesting that passengers' spending behavior is a significant predictor of the target variable.
- FoodCourt, Spa, and VRDeck expenditures are also commonly ranked as important features by all models, indicating that spending in these areas may have a meaningful relationship with the target variable.

Differences:

- Logistic Regression gives the highest importance to Spa, followed closely by FoodCourt, which is a slight deviation from the Random Forest model.
- The SVC model also emphasizes whether it has room service besides FoodCourt, Spa, and VRDeck.

- The Random Forest model shows a more evenly distributed importance across features, with Expenditure being the most prominent, followed by No_Spending, and FoodCourt.
- The Neural Network input layer weights place the highest importance on FoodCourt, followed by No_Spending, and VRDeck. This is markedly different from Logistic Regression, which gives less importance to VRDeck.
- XGBoost shows a clear preference for No_Spending as the most important feature by a significant margin, followed by CryoSleep, and HomePlanet. It is notable that CryoSleep and HomePlanet are given more importance in the XGBoost model compared to others.
- LGBMClassifier, which is identified as the best model, has a more evenly distributed feature importances than XGBoost. It identifies Cabin_number and Surname as the most predictive feature, and it does agree with the common trend that VRDeck, Spa, and Foodcourt have strong predictability. However, it regards the No_spending as the least important feature.

Overall Most Important Features:

Taking into account the best-performing model, LGBMClassifier, the top features are Cabin_number, Surname, Age, VRDeck, and Spa. However, when considering all models, No_Spending, FoodCourt, Spa, and VRDeck can be considered as overall important features due to their recurrent high importance across different models.

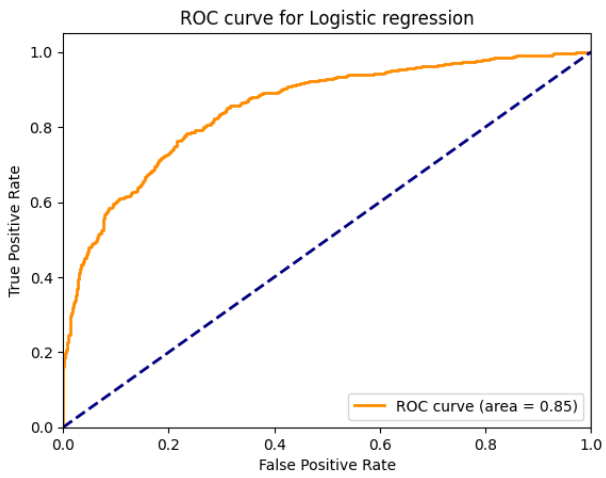
Conclusions and Implications

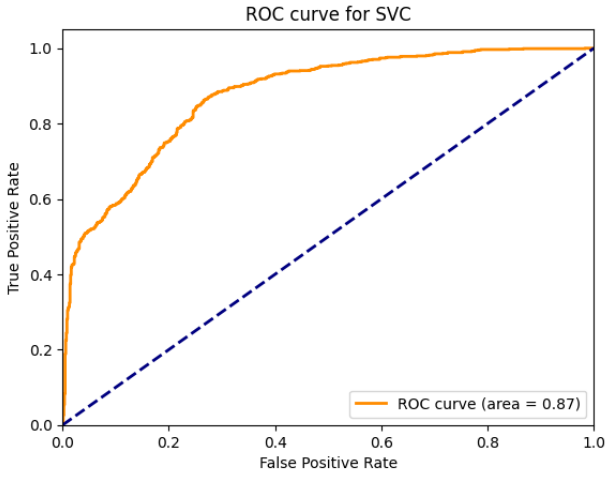
The model with the best performance is identified as LGBMClassfier. The model is robust and provides a high level of accuracy, precision, and recall, making it reliable for predictive purposes in a variety of contexts. However, it's crucial to implement a cost-benefit analysis of false positives versus false negatives to inform how the model's predictions should be acted upon. For instance, if the cost of a false negative is high (missing out on providing critical services), the business might want to aim for a higher recall for the 'True' class even at the expense of precision. Continuous monitoring and model retraining with new data should be recommended to maintain or improve model performance over time.

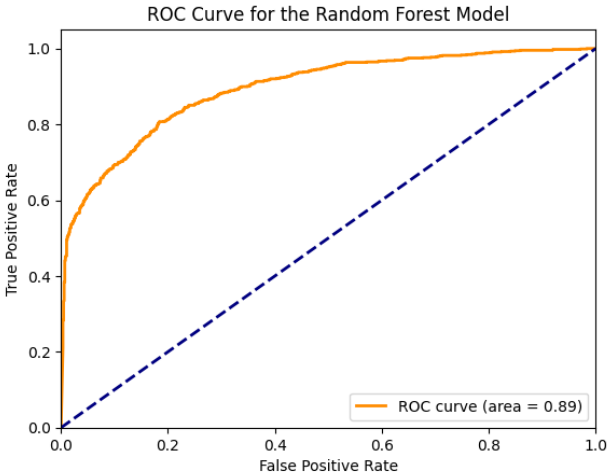
Appendix: Tables and Figures

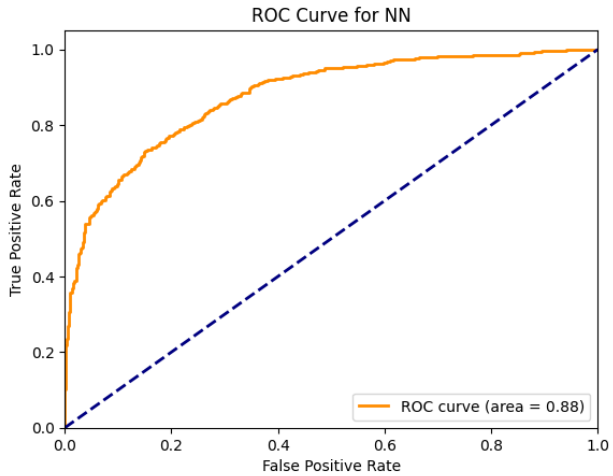
Model Name	Validation Accuracy	AUC score	Training Time (sec)
Logistic Regression	76.94%	0.85	0.04
SVC	78.49%	0.87	10.81
Random Forest	79.87%	0.89	6.15
Neural Network	78.43%	0.88	3.79
LGBMclassifier	80.73%	0.91	0.56
XGBoost	80.68%	0.9	4.2

Table 1. Accuracies, AUC score, and Training Time Summary









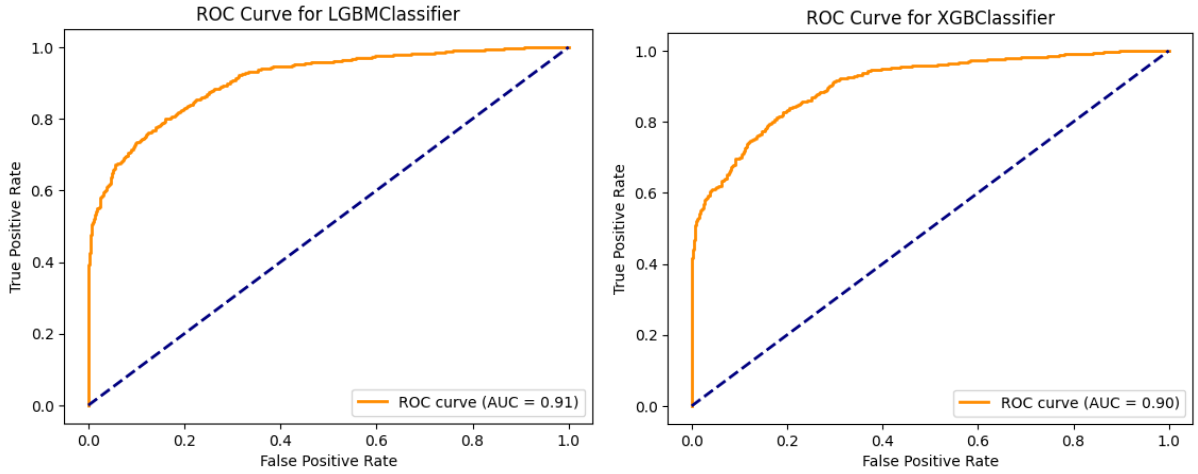
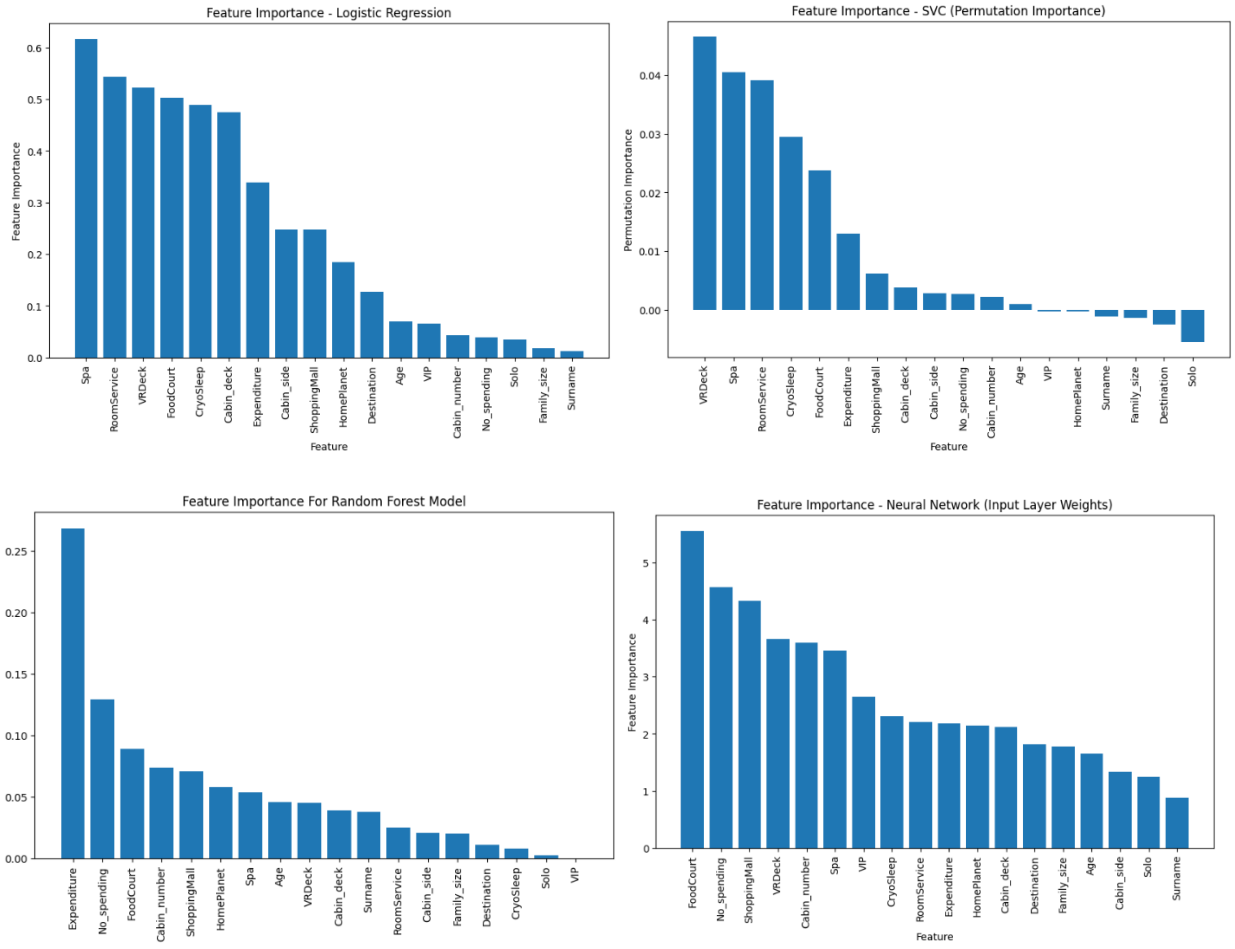


Figure 1. ROC Curve from Different Models



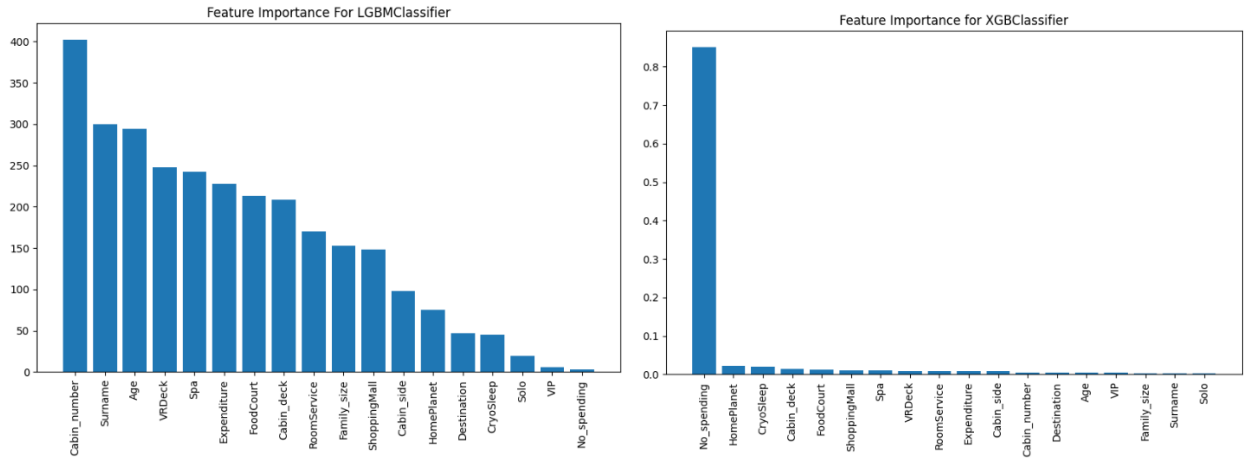


Figure 2. Feature Importance from Different Models

Spaceship_Titanic

March 6, 2024

[ ]:
```
# Core
import numpy as np
import pandas as pd
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline

import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
# Import data
train = pd.read_csv('train.csv')
#Kevin edit: file name different. Sorry!
#test = pd.read_csv('test (1).csv')
test = pd.read_csv('test.csv')
```

[ ]:
```
train.head()
```

[ ]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	\
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	

	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	\
0	0.0	0.0	0.0	0.0	0.0	Maham Ofracculy	
1	109.0	9.0	25.0	549.0	44.0	Juanna Vines	
2	43.0	3576.0	0.0	6715.0	49.0	Altark Susent	
3	0.0	1283.0	371.0	3329.0	193.0	Solam Susent	
4	303.0	70.0	151.0	565.0	2.0	Willy Santantines	

	Transported

```
0        False
1         True
2        False
3        False
4         True
```

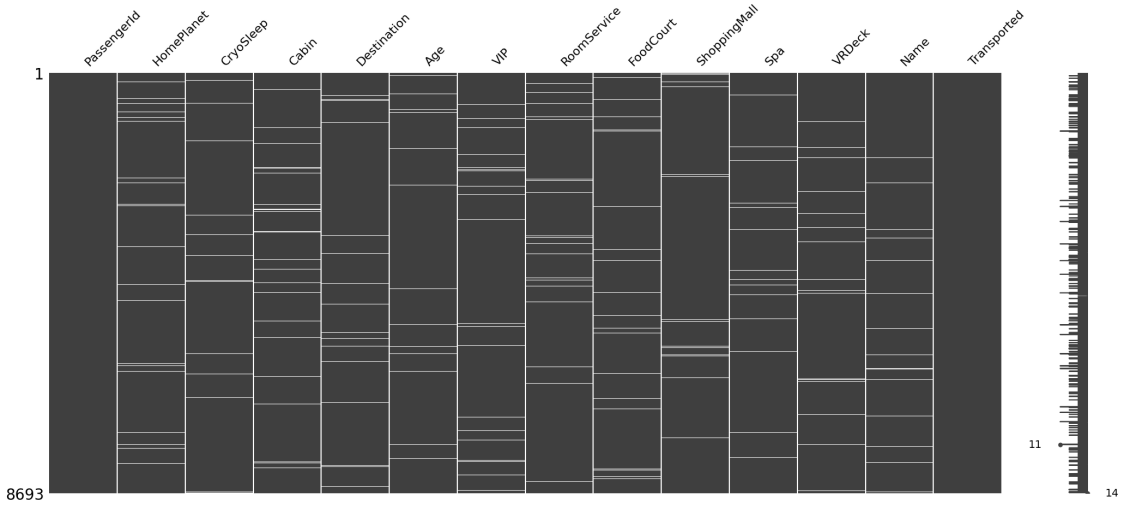
0.1 EDA

[ ]:
```
# Missing value
train.isna().sum().sort_values(ascending = False)
```

[ ]:
```
CryoSleep       217
ShoppingMall    208
VIP             203
HomePlanet      201
Name            200
Cabin           199
VRDeck          188
FoodCourt       183
Spa             183
Destination     182
RoomService     181
Age             179
PassengerId       0
Transported       0
dtype: int64
```

[ ]:
```
msno.matrix(train)
```

[ ]:
```
<Axes: >
```



[ ]:
```
# Fill null values with median (numeric) and frequent values (categoric)
numeric_data = [column for column in train.select_dtypes(["int", "float"])]
categoric_data = [column for column in train.select_dtypes(exclude = ["int",␣
 ↪"float"])]
test_categoric_data = [column for column in test.select_dtypes(exclude =␣
 ↪["int", "float"])]

for col in numeric_data:
    train[col].fillna(train[col].median(), inplace = True)
    test[col].fillna(test[col].median(), inplace = True)

#replace missing values in each categorical column with the most frequent value
for col in categoric_data:
    train[col].fillna(train[col].value_counts().index[0], inplace = True)
for col in test_categoric_data:
    test[col].fillna(test[col].value_counts().index[0], inplace = True)

# Check null values
train.isnull().sum().sum() + test.isnull().sum().sum()
```

[ ]:
```
0
```

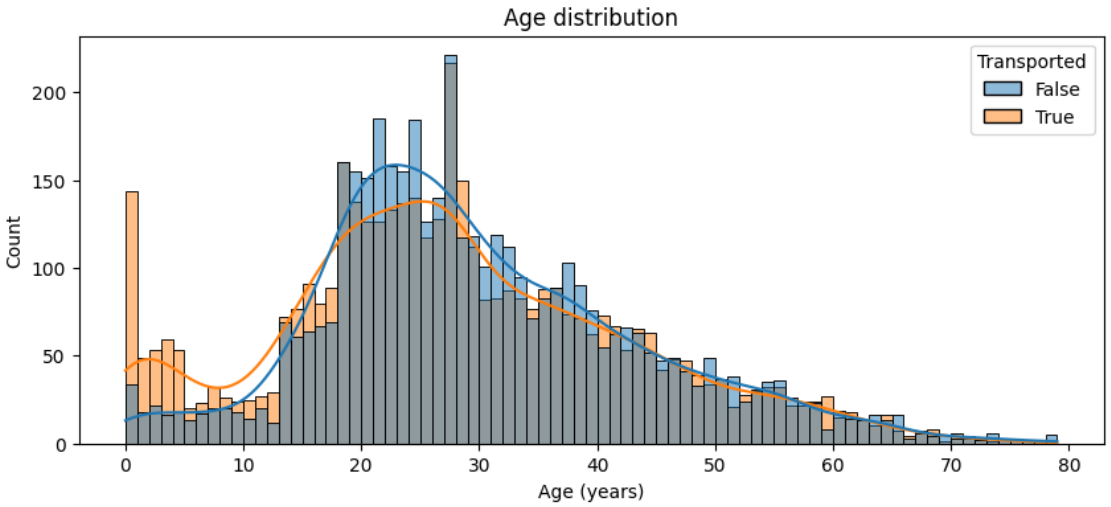
[ ]:
```
# Summary statistics
train.describe()
```

[ ]:
```
                Age    RoomService      FoodCourt  ShoppingMall           Spa  \
count  8693.000000   8693.000000   8693.000000   8693.000000   8693.000000
mean     28.790291    220.009318    448.434027    169.572300    304.588865
std      14.341404    660.519050   1595.790627    598.007164   1125.562559
min       0.000000      0.000000      0.000000      0.000000      0.000000
25%      20.000000      0.000000      0.000000      0.000000      0.000000
50%      27.000000      0.000000      0.000000      0.000000      0.000000
75%      37.000000     41.000000     61.000000     22.000000     53.000000
max      79.000000  14327.000000  29813.000000  23492.000000  22408.000000

             VRDeck
count   8693.000000
mean     298.261820
std     1134.126417
min        0.000000
25%        0.000000
50%        0.000000
75%       40.000000
max    24133.000000
```

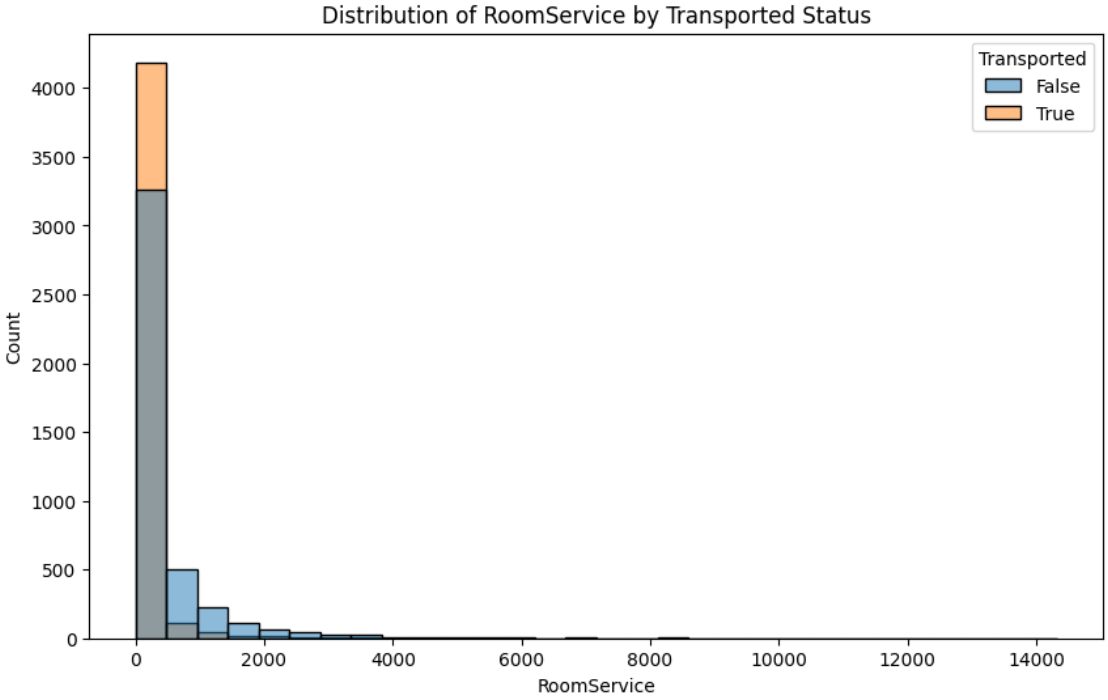
[ ]:
```
# Age distribution
# Figure size
```

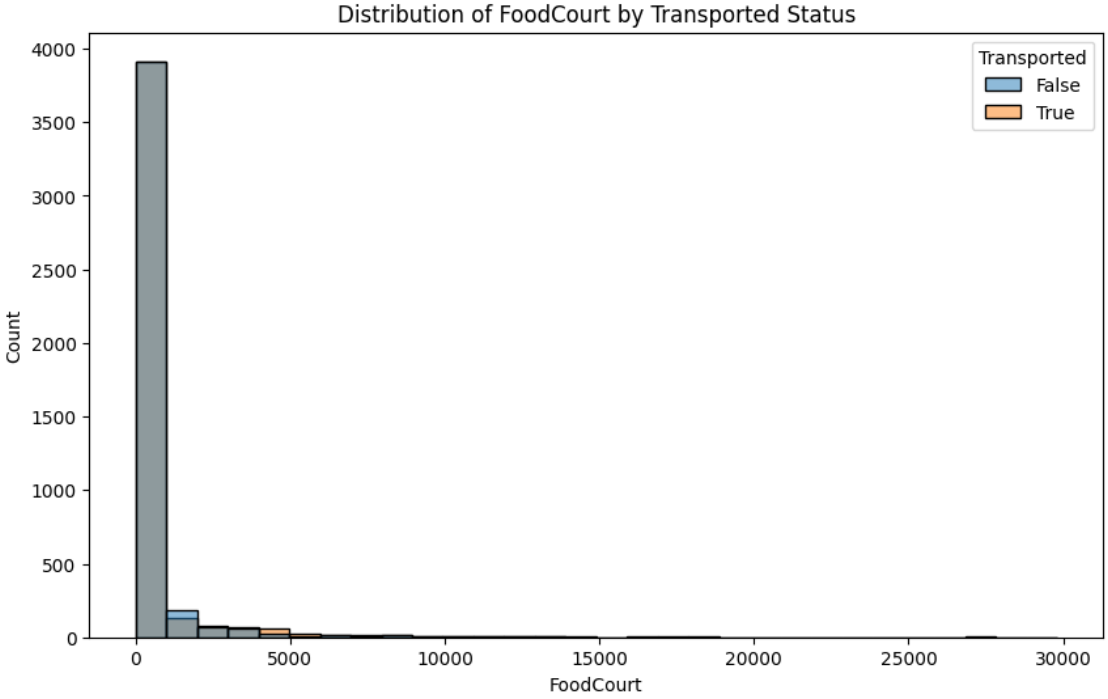
```
plt.figure(figsize=(10,4))

# Histogram
sns.histplot(data=train, x='Age', hue='Transported', binwidth=1, kde=True)

# Aesthetics
plt.title('Age distribution')
plt.xlabel('Age (years)')
```

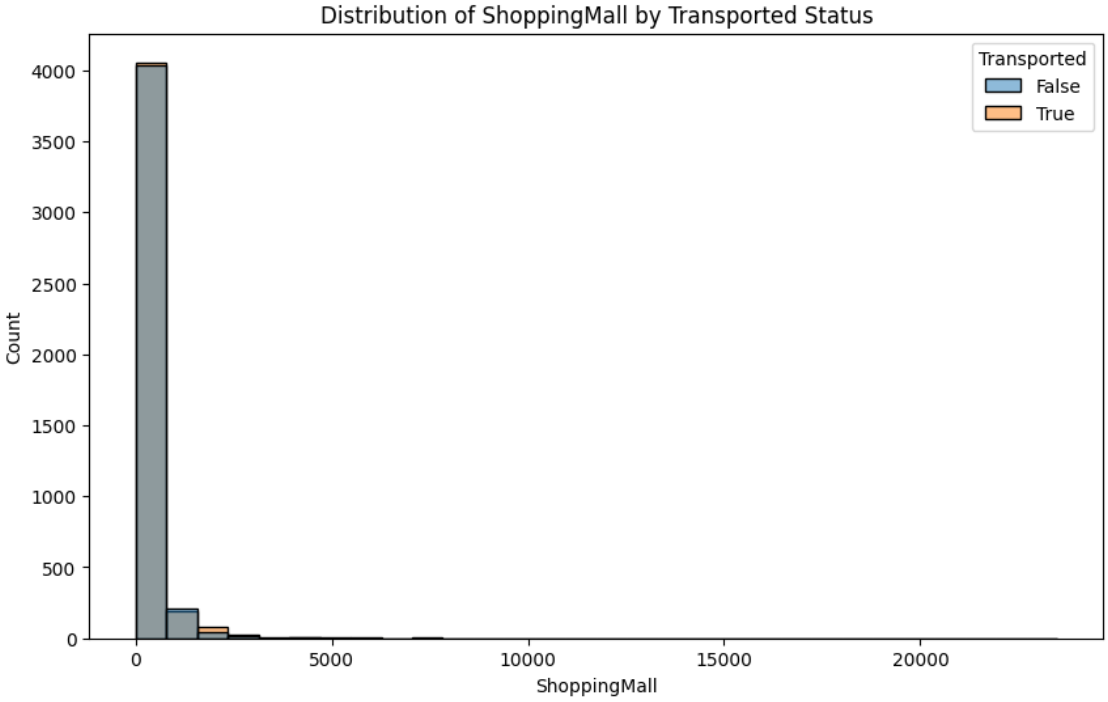
[ ]:
```
Text(0.5, 0, 'Age (years)')
```

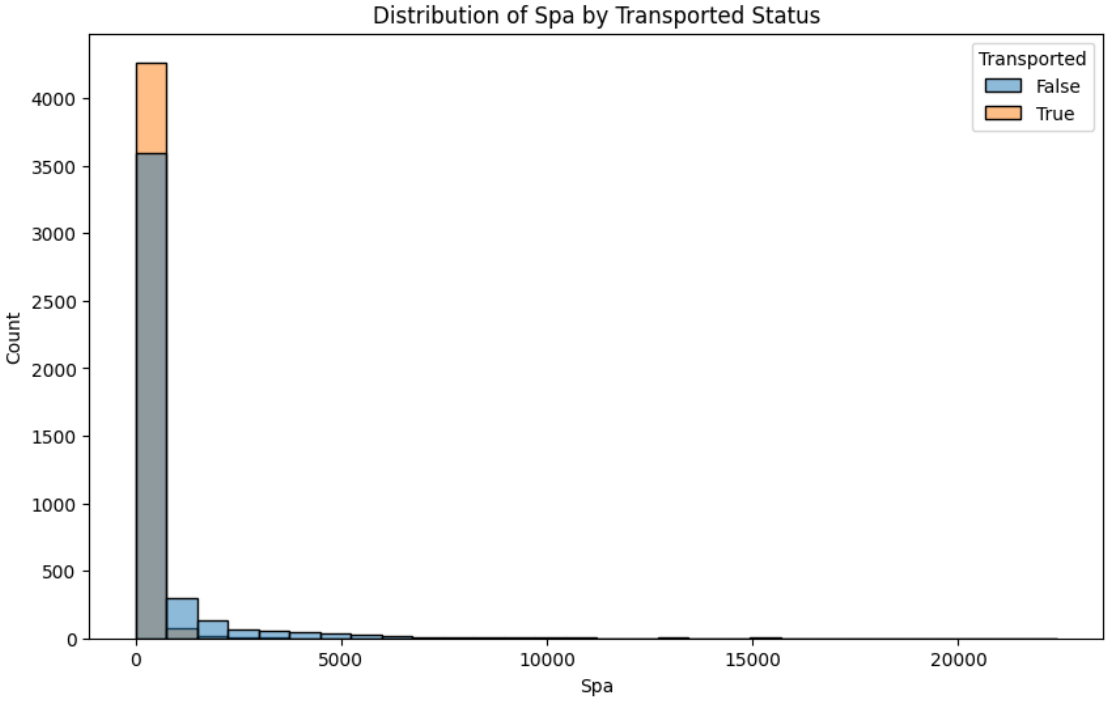


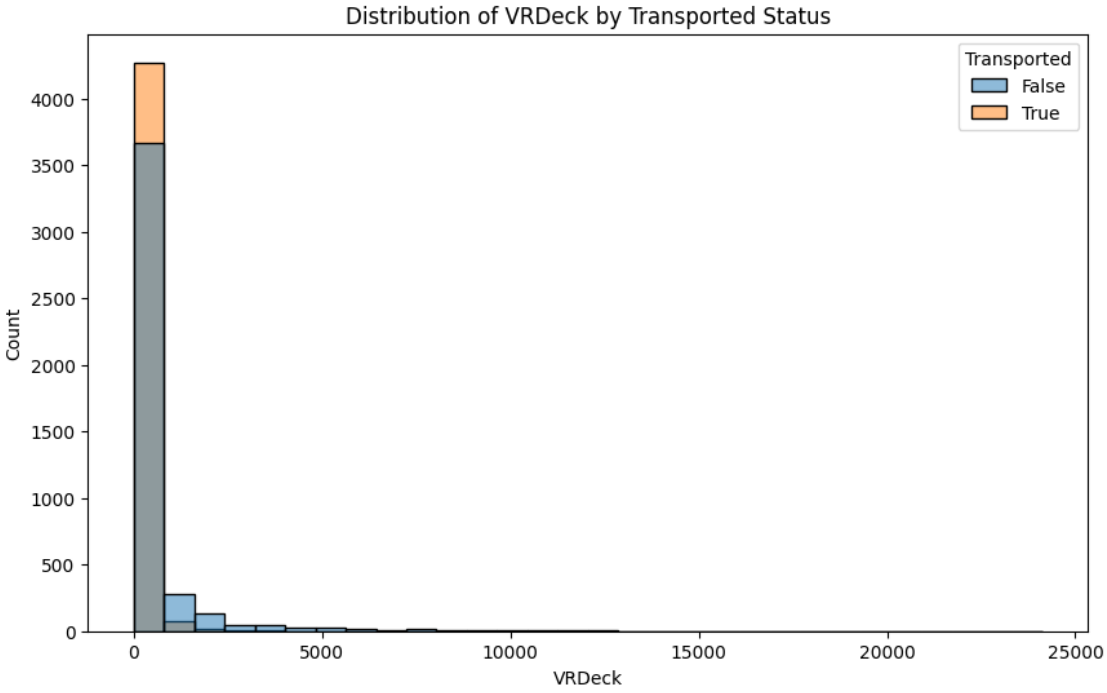
[ ]:
```
# Expenditure features
exp_feats=['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
for i, var_name in enumerate(exp_feats):
    plt.figure(figsize=(10, 6))
    sns.histplot(data=train, x=var_name, bins=30, kde=False, hue='Transported')
    plt.title(f'Distribution of {var_name} by Transported Status')
    plt.xlabel(var_name)
    plt.ylabel('Count')
    plt.show()
```


Distribution of RoomService by Transported Status
Transported
False
True
Count
RoomService
0
500
1000
1500
2000
2500
3000
3500
4000
0
2000
4000
6000
8000
10000
12000
14000


Distribution of FoodCourt by Transported Status
Transported
False
True
Count
FoodCourt
0
500
1000
1500
2000
2500
3000
3500
4000
0
5000
10000
15000
20000
25000
30000

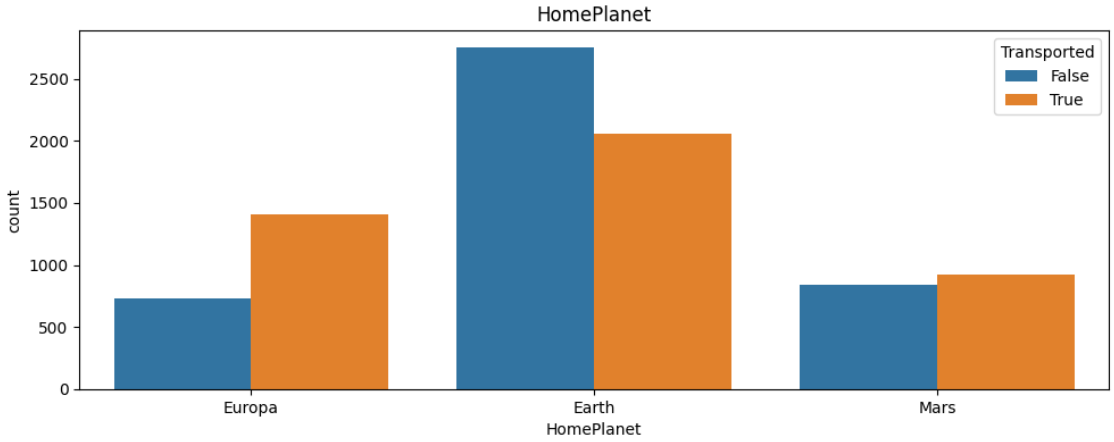

Distribution of ShoppingMall by Transported Status
Transported
False
True
Count
ShoppingMall
0
500
1000
1500
2000
2500
3000
3500
4000
0
5000
10000
15000
20000

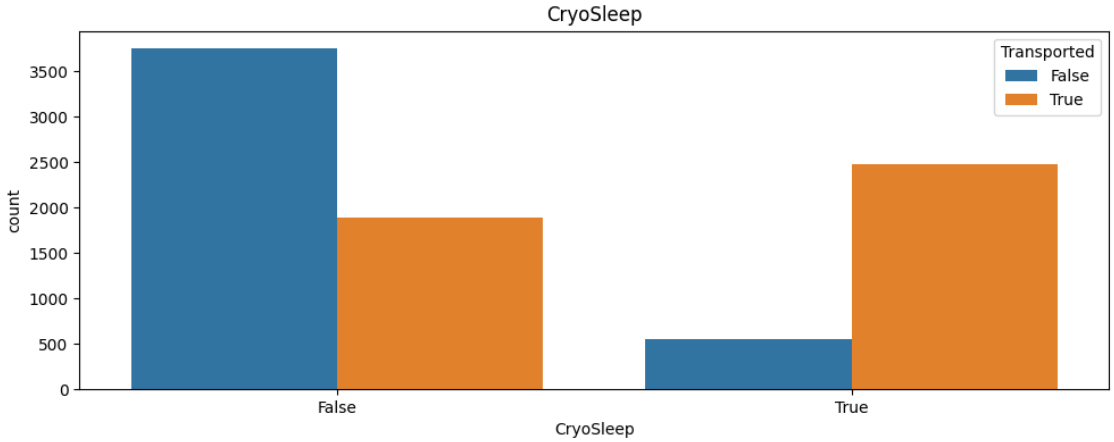

Distribution of Spa by Transported Status
Transported
False
True
Count
Spa
0
500
1000
1500
2000
2500
3000
3500
4000
0
5000
10000
15000
20000

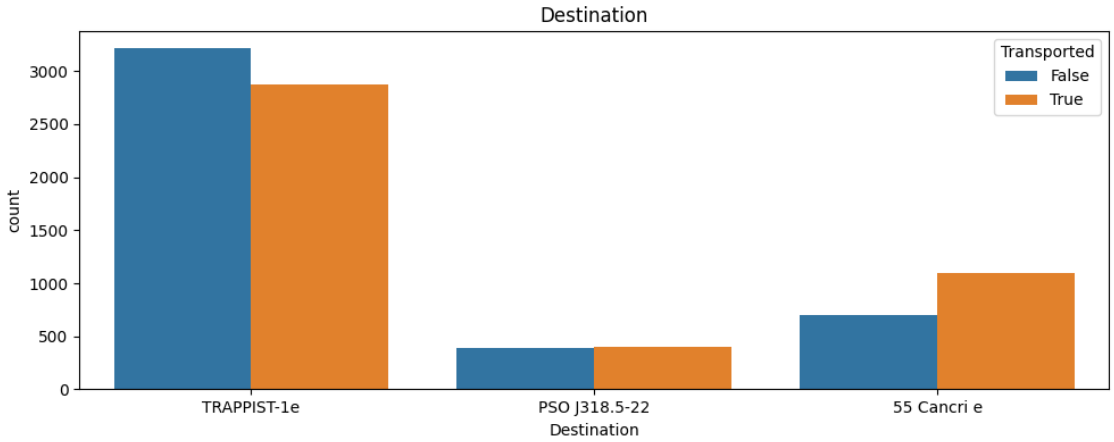


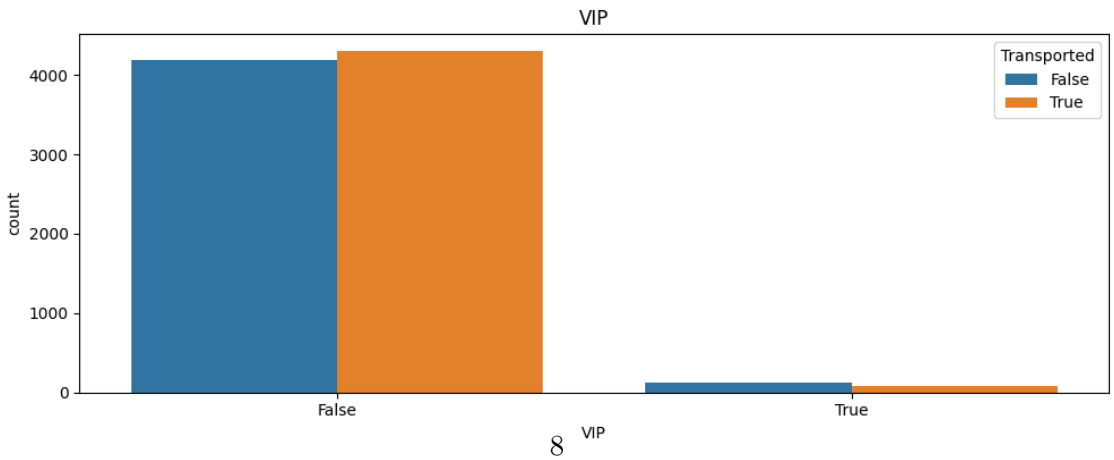
Feature Engineering Idea: - Create a new feature that tracks the total expenditure across all 5 amenities. - Take the log transform to reduce skew.

[ ]:
```
# Categorical features
cat_feats=['HomePlanet', 'CryoSleep', 'Destination', 'VIP']

# Plot categorical features
fig=plt.figure(figsize=(10,16))
for i, var_name in enumerate(cat_feats):
    ax=fig.add_subplot(4,1,i+1)
    sns.countplot(data=train, x=var_name, axes=ax, hue='Transported')
    ax.set_title(var_name)
fig.tight_layout()  # Improves appearance a bit
plt.show()
```


HomePlanet
Transported
False
True
count
2500
2000
1500
1000
500
0
Europa
Earth
Mars
HomePlanet


CryoSleep
Transported
False
True
count
3500
3000
2500
2000
1500
1000
500
0
False
True
CryoSleep


Destination
Transported
False
True
count
3000
2500
2000
1500
1000
500
0
TRAPPIST-1e
PSO J318.5-22
55 Cancri e
Destination


VIP
Transported
False
True
count
4000
3000
2000
1000
0
False
True
VIP

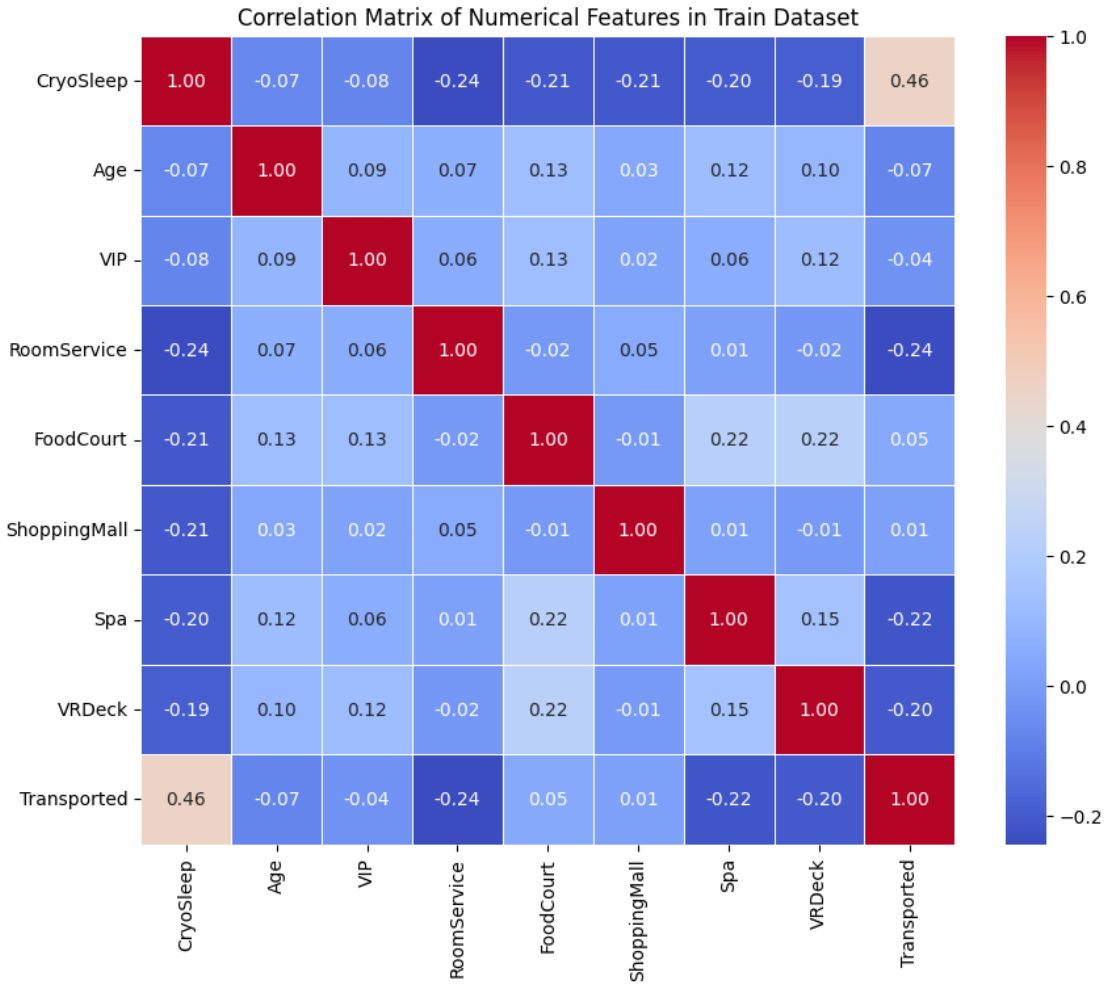
[ ]:
```
# Qualitative features
qual_feats=['PassengerId', 'Cabin' ,'Name']

# Preview qualitative features
train[qual_feats].head()
```

[ ]:

	PassengerId	Cabin	Name
0	0001_01	B/0/P	Maham Ofracculy
1	0002_01	F/0/S	Juanna Vines
2	0003_01	A/0/S	Altark Susent
3	0003_02	A/0/S	Solam Susent
4	0004_01	F/1/S	Willy Santantines

Feature Engineering Idea: - We can extract the group and group size from the PassengerId feature. - We can extract the deck, number and side from the cabin feature. - We could extract the surname from the name feature to identify families.

[ ]:
```
# Correlation Matrix
train_numerical = train.select_dtypes(include=['number','bool'])

# Calculate the correlation matrix for numerical columns only
correlation_matrix_numerical = train_numerical.corr()

# Create a heatmap to visualize the correlation matrix of numerical columns
plt.figure(figsize=(10, 8))  # Adjust the size as needed
sns.heatmap(correlation_matrix_numerical, annot=True, cmap='coolwarm', fmt=".
↪2f", linewidths=.5)
plt.title('Correlation Matrix of Numerical Features in Train Dataset')  #␣
↪Optional: Add a title
plt.show()
```



[ ]:
```
train.dtypes
```

[ ]:
```
PassengerId      object
HomePlanet       object
CryoSleep          bool
Cabin            object
Destination      object
Age             float64
VIP                bool
RoomService     float64
FoodCourt       float64
ShoppingMall    float64
Spa             float64
VRDeck          float64
Name             object
Transported        bool
```

```
dtype: object
```

0.2 Feature Engineering

[ ]:
```
train.shape
```

[ ]:
```
(8693, 14)
```

[ ]:
```
test.shape
```

[ ]:
```
(4277, 13)
```

[ ]:
```
# Expenditure
# New features - training set
train['Expenditure']=train[exp_feats].sum(axis=1)
train['No_spending']=(train['Expenditure']==0).astype(int)

# New features - test set
test['Expenditure']=test[exp_feats].sum(axis=1)
test['No_spending']=(test['Expenditure']==0).astype(int)
```

[ ]:
```
# Extract passenger group and group size from PassengerID
# New feature - Group
train['Group'] = train['PassengerId'].apply(lambda x: x.split('_')[0]).
 ↪astype(int)
test['Group'] = test['PassengerId'].apply(lambda x: x.split('_')[0]).astype(int)

# New feature - Group size
train['Group_size']=train['Group'].map(lambda x: pd.concat([train['Group'],␣
 ↪test['Group']]).value_counts()[x])
test['Group_size']=test['Group'].map(lambda x: pd.concat([train['Group'],␣
 ↪test['Group']]).value_counts()[x])
```

[ ]:
```
# Not use 'Group' in the model to avoid overfitting
train['Group'].nunique()
```

[ ]:
```
6217
```

[ ]:
```
# Check group size and create 'solo' column
train['Group_size'].value_counts()

# New feature
train['Solo']=(train['Group_size']==1).astype(int)
test['Solo']=(test['Group_size']==1).astype(int)
```

[ ]:
```
# Create 'deck' and 'side' features from 'cabin' column
```

```
# New features - training set
train['Cabin_deck'] = train['Cabin'].apply(lambda x: x.split('/')[0])
train['Cabin_number'] = train['Cabin'].apply(lambda x: x.split('/')[1]).
 ↪astype(int)
train['Cabin_side'] = train['Cabin'].apply(lambda x: x.split('/')[2])

# New features - test set
test['Cabin_deck'] = test['Cabin'].apply(lambda x: x.split('/')[0])
test['Cabin_number'] = test['Cabin'].apply(lambda x: x.split('/')[1]).
 ↪astype(int)
test['Cabin_side'] = test['Cabin'].apply(lambda x: x.split('/')[2])
```

[ ]:
```
# New feature - Surname
train['Surname']=train['Name'].str.split().str[-1]
test['Surname']=test['Name'].str.split().str[-1]

# New feature - Family size
train['Family_size']=train['Surname'].map(lambda x: pd.
 ↪concat([train['Surname'],test['Surname']]).value_counts()[x])
test['Family_size']=test['Surname'].map(lambda x: pd.
 ↪concat([train['Surname'],test['Surname']]).value_counts()[x])
```

[ ]:
```
train.drop(['PassengerId', 'Group', 'Group_size','Cabin', 'Name'], axis=1,␣
 ↪inplace=True)
test.drop(['PassengerId', 'Group', 'Group_size', 'Cabin', 'Name'], axis=1,␣
 ↪inplace=True)
```

[ ]:
```
train.dtypes
```

[ ]:
```
HomePlanet       object
CryoSleep          bool
Destination      object
Age             float64
VIP                bool
RoomService     float64
FoodCourt       float64
ShoppingMall    float64
Spa             float64
VRDeck          float64
Transported        bool
Expenditure     float64
No_spending       int64
Solo              int64
Cabin_deck       object
Cabin_number      int64
Cabin_side       object
Surname          object
```

```
Family_size       int64
dtype: object
```

[ ]:
```
# Update 'Cabin_number' to object
train['Cabin_number'] = train['Cabin_number'].astype('object')
```

[ ]:
```
# Apply log transform
for col in␣
 ↪['RoomService','FoodCourt','ShoppingMall','Spa','VRDeck','Expenditure']:
    train[col]=np.log(1+train[col])
    test[col]=np.log(1+test[col])
```

[ ]:
```
# Encoding categorical features (Label Encoding)
label_cols = ['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'No_spending',␣
 ↪'Solo', 'Cabin_deck', 'Cabin_number', 'Cabin_side', 'Surname' ]
def label_encoder(train,test,columns):
    for col in columns:
        train[col] = train[col].astype(str)
        test[col] = test[col].astype(str)
        train[col] = LabelEncoder().fit_transform(train[col])
        test[col] =  LabelEncoder().fit_transform(test[col])
    return train, test

train, test = label_encoder(train,test, label_cols)
```

[ ]:
```
# Standardization
numerical_cols = train.select_dtypes(include=['number']).columns

# Initializing the StandardScaler
scaler = StandardScaler()

# Fitting the scaler to the numerical columns and transforming the data
train[numerical_cols] = scaler.fit_transform(train[numerical_cols])
test[numerical_cols] = scaler.fit_transform(test[numerical_cols])
```

[ ]:
```
# Splitting the train
X = train.drop('Transported', axis=1)
y = train['Transported']
X_train, X_val, y_train, y_val = train_test_split(X,
                                                  y,
                                                  test_size=0.2,
                                                  stratify=train['Transported'],
                                                  random_state=42)
X_train.to_csv("X_train.csv")
X_val.to_csv("X_val.csv")
y_train.to_csv("y_train.csv")
y_val.to_csv("y_val.csv")
```

0.3 Kevin: Tree and Random Forest (Using random_state = 42 for CV)

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, StratifiedKFold,␣
↪train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.model_selection import GridSearchCV
```

[ ]:
```
tree_params = {'max_depth': np.arange(1, 21)}

# Initialize cross-validation
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Initialize Decision Tree classifier
tree_clf = DecisionTreeClassifier(random_state=42)

# Perform cross-validation for hyperparameter tuning
tree_grid = GridSearchCV(tree_clf, tree_params, cv=cv, scoring='accuracy')
tree_grid.fit(X_train, y_train)

# Get the best Decision Tree model
best_tree_model = tree_grid.best_estimator_
print(best_tree_model)
```

```
DecisionTreeClassifier(max_depth=7, random_state=42)
```

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV

# Random Forest hyperparameter tuning with RandomizedSearchCV
# Define hyperparameters to tune
rf_params = {'n_estimators': [100, 200],
             'max_features': ['auto', None],
             'max_depth': [None, 10, 20, 30],
             'min_samples_split': [2, 5, 15],
             'min_samples_leaf': [1, 5, 12]}

# Initialize Random Forest classifier
rf_clf = RandomForestClassifier(random_state=42)

# Perform cross-validation for hyperparameter tuning using RandomizedSearchCV
rf_random = RandomizedSearchCV(estimator = rf_clf, param_distributions =␣
↪rf_params,
                               n_iter = 100, cv = cv, scoring='accuracy',␣
↪random_state=42, verbose = 5)

# Perform cross-validation for hyperparameter tuning
```

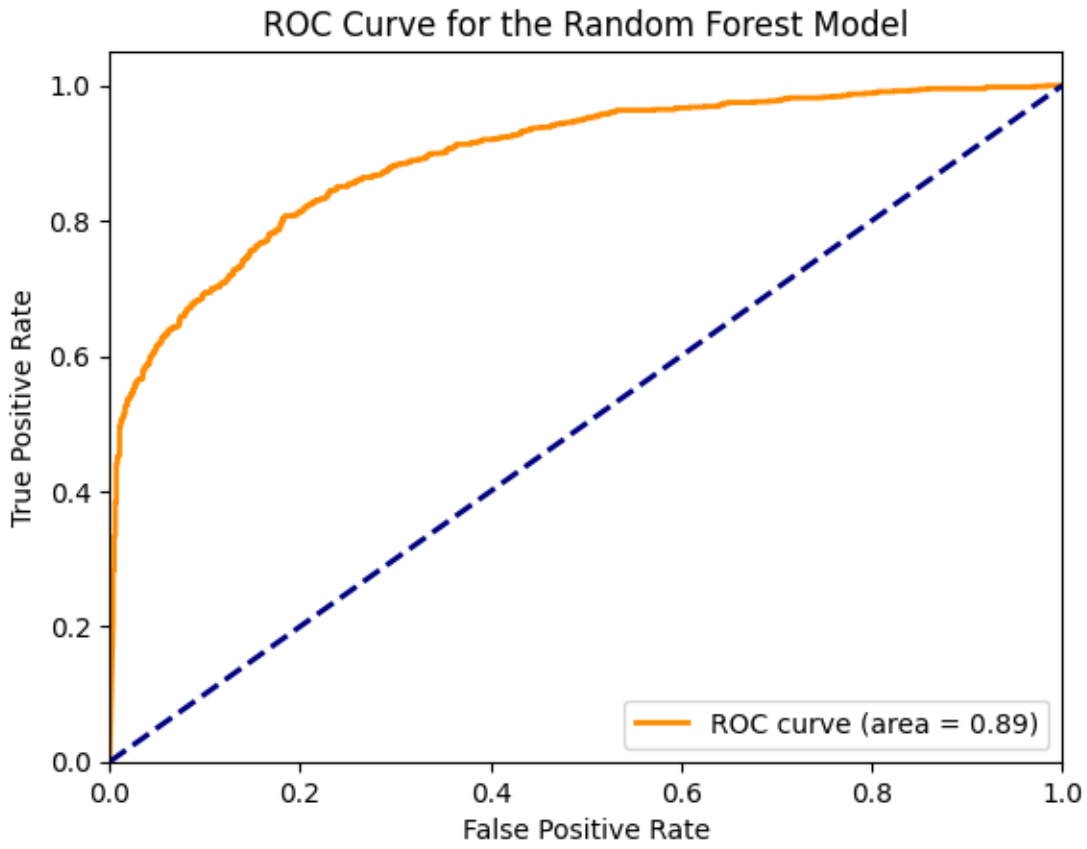
```
#rf_grid = GridSearchCV(rf_clf, rf_params, cv=cv, scoring='accuracy',␣
 ↪verbose=10)
rf_random.fit(X_train, y_train)

# Get the best Random Forest model
best_rf_model = rf_random.best_estimator_
print(best_rf_model)
```

[ ]:
```
import time

best_tree_model = DecisionTreeClassifier(max_depth=7, random_state=42)
best_rf_model = RandomForestClassifier(max_depth=10, max_features=None,␣
 ↪min_samples_leaf=5,
                       min_samples_split=15, random_state=42)

# Record the start time
start_time = time.time()
# Fit the model on the entire training set
best_tree_model.fit(X_train, y_train)
# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")


# Decision Tree
tree_val_pred = best_tree_model.predict(X_val)
tree_val_accuracy = accuracy_score(y_val, tree_val_pred)

# Record the start time
start_time = time.time()
# Fit the model on the entire training set
best_rf_model.fit(X_train, y_train)
# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")

# Random Forest
rf_val_pred = best_rf_model.predict(X_val)
rf_val_accuracy = accuracy_score(y_val, rf_val_pred)

print("Decision Tree validation accuracy:", tree_val_accuracy)
print("Random Forest validation accuracy:", rf_val_accuracy)
```

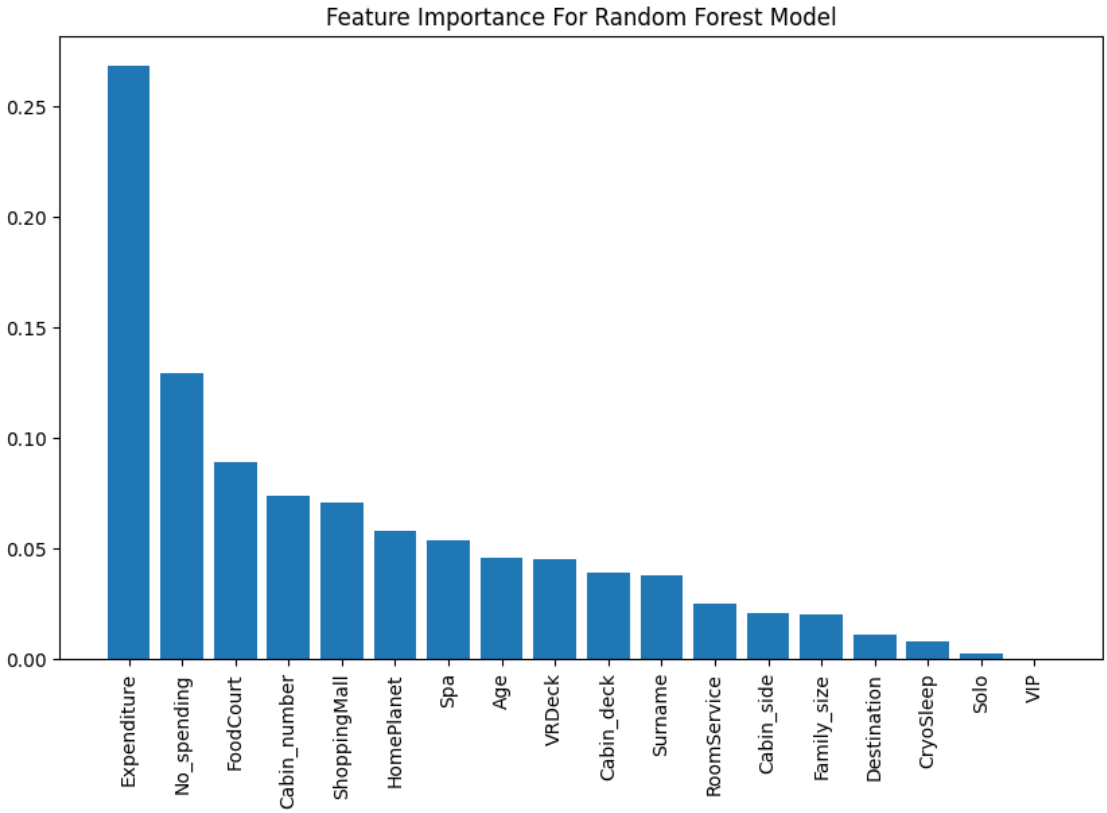
```
Training time: 0.061949968338012695 seconds
```

```
Training time: 6.15765118598938 seconds
Decision Tree validation accuracy: 0.7814836112708453
Random Forest validation accuracy: 0.8033352501437608
```

[ ]:
```python
# Plot ROC curve for the selected model (Random Forest in this case)
rf_probs = best_rf_model.predict_proba(X_val)[:, 1]
fpr, tpr, thresholds = roc_curve(y_val, rf_probs)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for the Random Forest Model')
plt.legend(loc="lower right")
plt.show()
```



[ ]:
```
# Assuming best_rf_model is your trained RandomForest model
feature_importances = best_rf_model.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(feature_importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]

# Plot
plt.figure(figsize=(10, 6))
plt.title("Feature Importance For Random Forest Model")
plt.bar(range(X_train.shape[1]), feature_importances[indices])
plt.xticks(range(X_train.shape[1]), names, rotation=90)
plt.show()
```



1 Xinran: Logistic, svm, basic NN

1.1 Logistic Regression

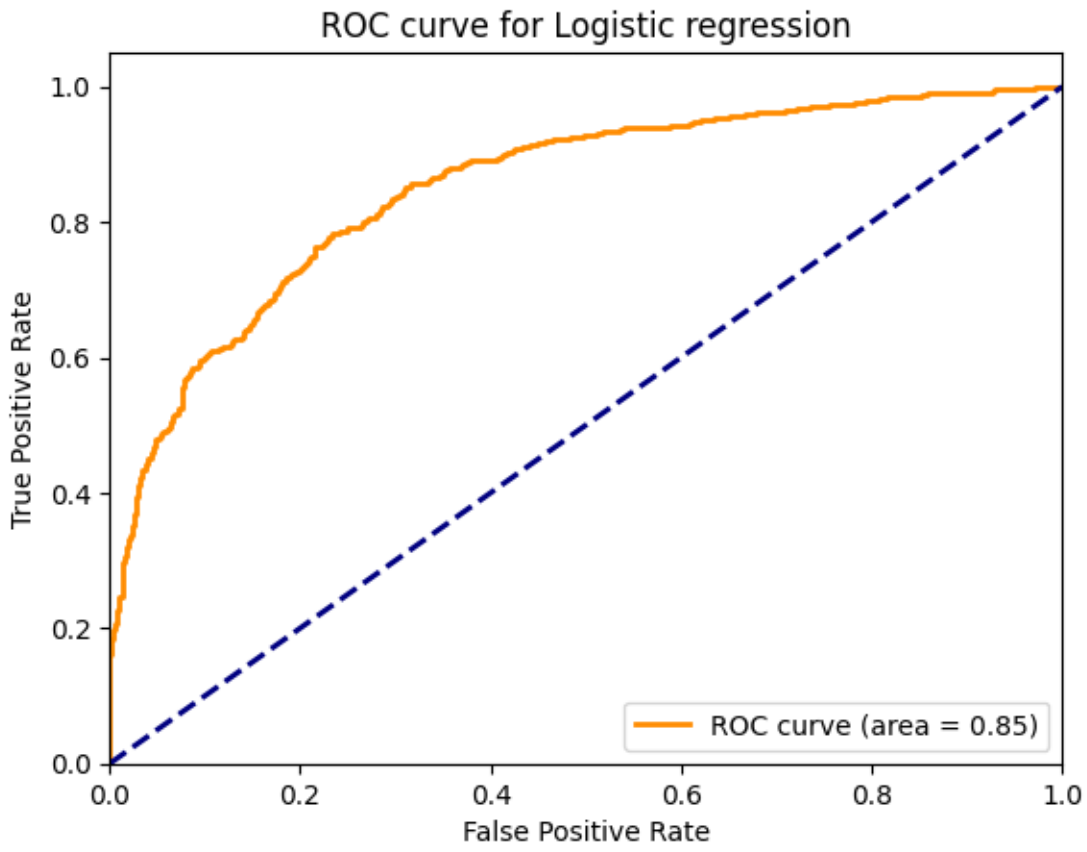
[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve, auc
import numpy as np

import time
# Record the start time
start_time = time.time()

# Prepare the logistic regression model
logistic_model = LogisticRegression(max_iter=1000, random_state=42)

'''
# Prepare cross-validation (k=10, random_state=42)
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# training accuracy
logistic_accuracy_training = cross_val_score(logistic_model,
                                             X_train, y_train.values.ravel(),
                                             cv=cv,
                                             scoring='accuracy')

# training AUC
y_scores_logistic = cross_val_predict(logistic_model,
                                      X_train, y_train.values.ravel(),
                                      cv=10,
                                      method="decision_function")
logistic_auc_training = roc_auc_score(y_train, y_scores_logistic)
'''

# Fit the model on the entire training set
logistic_model.fit(X_train, y_train)

# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")

# Predict on validation set
y_val_pred = logistic_model.predict(X_val)
y_val_pred_proba = logistic_model.predict_proba(X_val)[:,1]
```

```
# Calculate accuracy and ROC AUC score for the validation set
accuracy_val = accuracy_score(y_val, y_val_pred)
roc_auc_val = roc_auc_score(y_val, y_val_pred_proba)

# Calculate mean accuracy
logistic_mean_accuracy = np.mean(accuracy_val)

logistic_mean_accuracy, roc_auc_val
```

```
Training time: 0.03689742088317871 seconds
```

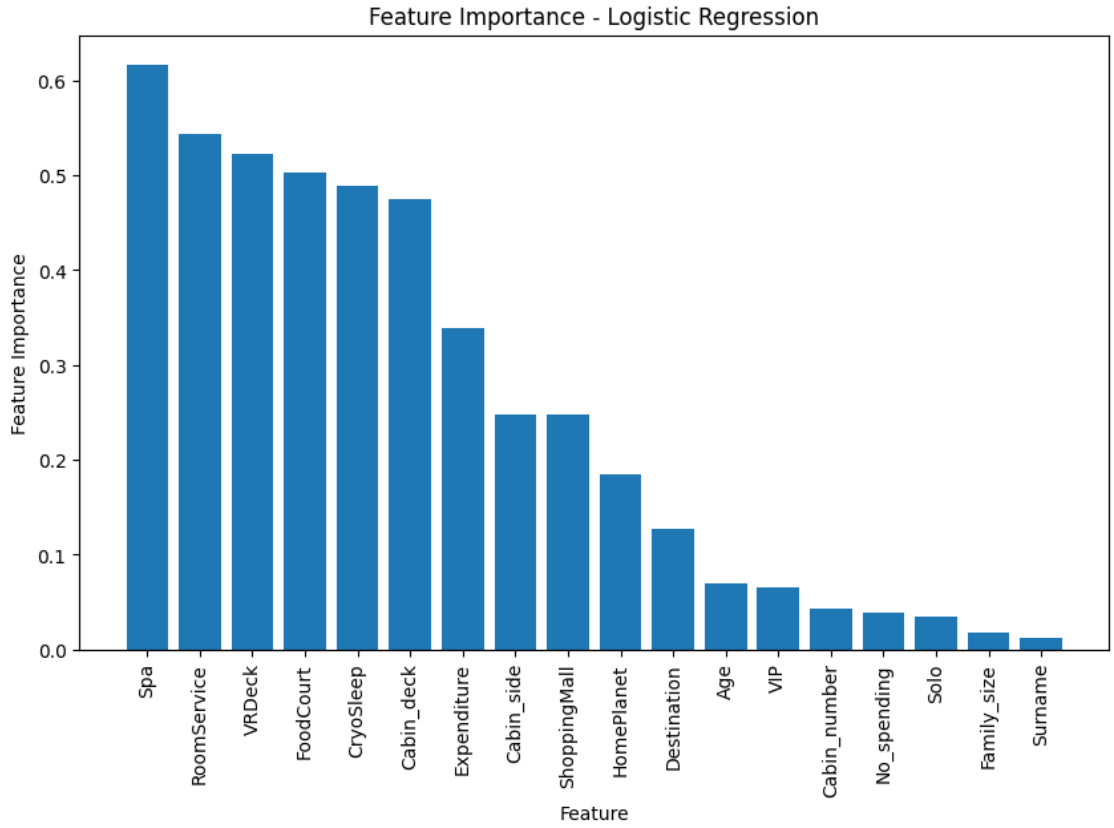
[ ]:
```
(0.7694077055779184, 0.8518045259977671)
```

[ ]:
```
# ROC curve for logistic
fpr, tpr, thresholds = roc_curve(y_val, y_val_pred_proba)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
  ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for Logistic regression')
plt.legend(loc="lower right")
plt.show()
```



[ ]:
```
import matplotlib.pyplot as plt

# Get the coefficients of the logistic regression model
coefficients = logistic_model.coef_[0]

# Get the absolute values of coefficients for feature importance
feature_importance = abs(coefficients)

# Get the names of the features
feature_names = X_train.columns

# Sort feature importance and feature names in descending order
sorted_indices = feature_importance.argsort()[::-1]
sorted_feature_importance = feature_importance[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

# Plot the bar chart
plt.figure(figsize=(10, 6))
```

```
plt.bar(range(len(sorted_feature_importance)), sorted_feature_importance,␣
  ↪align='center')
plt.xticks(range(len(sorted_feature_importance)),␣
  ↪sorted_feature_names,rotation=90)
plt.ylabel('Feature Importance')
plt.xlabel('Feature')
plt.title('Feature Importance - Logistic Regression')
plt.show()
```



1.2 SVM

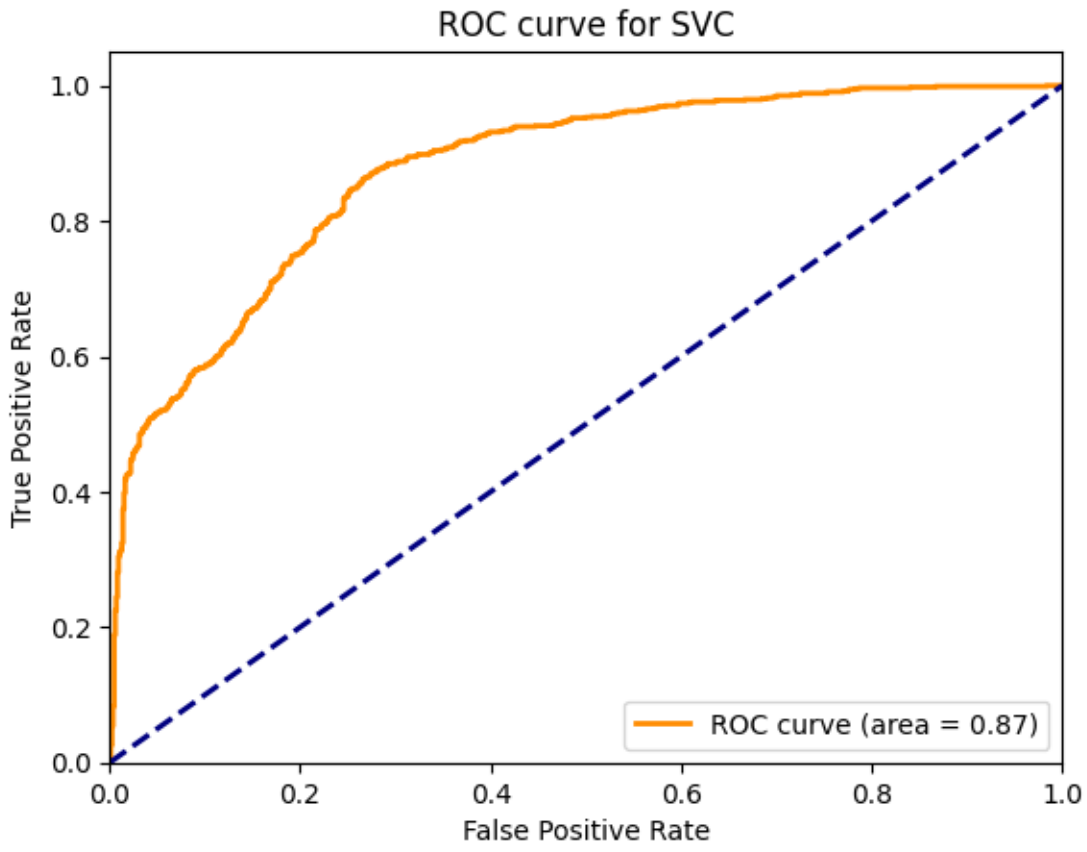
[ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_predict
import numpy as np
import matplotlib.pyplot as plt

import time
# Record the start time
```

```
start_time = time.time()

# Prepare the model
svc_model = SVC(probability=True, random_state=42)

'''
# Prepare cross-validation (k=10, random_state=42)
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Perform 10-fold cross-validation to estimate accuracy
sum_accuracy_scores = cross_val_score(model, X_train, y_train, cv=cv,␣
 ↪scoring='accuracy')

# Predict probabilities for ROC curve using cross-validation
y_probas = cross_val_predict(model, X_train, y_train, cv=cv,␣
 ↪method='predict_proba')
sum_roc_auc_score = roc_auc_score(y_train, y_probas)

# Calculate mean accuracy
sum_mean_accuracy = np.mean(sum_accuracy_scores)
'''

# Fit the model on the entire training set
svc_model.fit(X_train, y_train)

# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")

# Predict on validation set
y_val_pred = svc_model.predict(X_val)
y_val_pred_proba = svc_model.predict_proba(X_val)[:,1]

# Calculate accuracy and ROC AUC score for the validation set
accuracy_val = accuracy_score(y_val, y_val_pred)
roc_auc_val = roc_auc_score(y_val, y_val_pred_proba)

# Calculate val accuracy
svc_mean_accuracy = np.mean(accuracy_val)

svc_mean_accuracy, roc_auc_val
```

```
Training time: 10.810791254043579 seconds
```

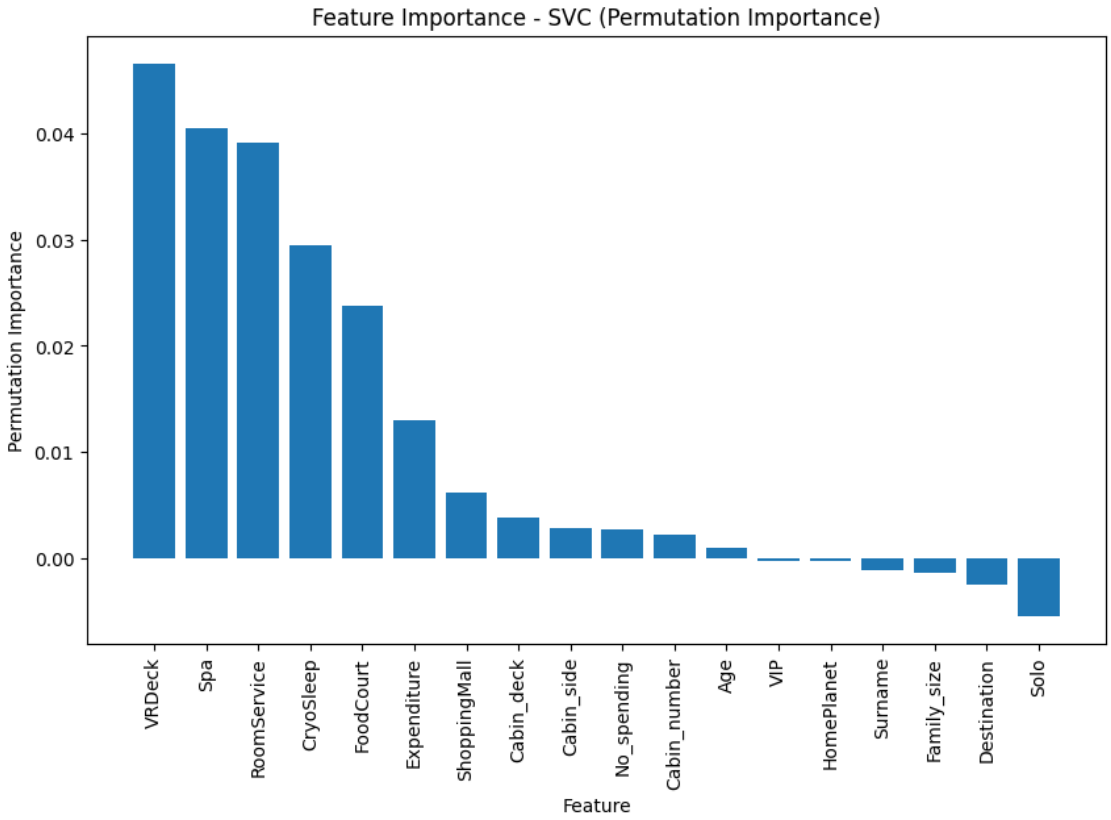
[ ]: (0.7849338700402531, 0.8745390138467808)

[ ]:
```
# ROC curve for logistic
fpr, tpr, thresholds = roc_curve(y_val, y_val_pred_proba)
roc_auc = auc(fpr, tpr)

# Plotting the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
 ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for SVC')
plt.legend(loc="lower right")
plt.show()
```



[ ]:
```
from sklearn.inspection import permutation_importance
```

```
# Compute permutation importance
result = permutation_importance(svc_model, X_val, y_val, n_repeats=10,␣
 ↪random_state=42)

# Get sorted indices and feature names
sorted_indices = result.importances_mean.argsort()[::-1]
sorted_feature_importance = result.importances_mean[sorted_indices]
sorted_feature_names = X_train.columns[sorted_indices]

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(range(len(sorted_feature_importance)), sorted_feature_importance,␣
 ↪align='center')
plt.xticks(range(len(sorted_feature_importance)), sorted_feature_names,␣
 ↪rotation =90)
plt.ylabel('Permutation Importance')
plt.xlabel('Feature')
plt.title('Feature Importance - SVC (Permutation Importance)')
plt.show()
```



1.3 Basic NN

[ ]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

# Define different sizes for the hidden layer
hidden_layer_sizes = [(5,), (10,), (20,), (50,), (100,)]

# Record the average cross-validation score for each configuration
cv_scores = {}
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

for size in hidden_layer_sizes:
    model = MLPClassifier(hidden_layer_sizes=size, max_iter=1000,␣
 ↪random_state=42)
    scores = cross_val_score(model, X_train, y_train, cv=kfold,␣
 ↪scoring='accuracy')
    cv_scores[size] = scores.mean()

# Find the best hidden layer size based on the highest mean accuracy
best_hidden_layer_size = max(cv_scores, key=cv_scores.get)
best_accuracy = cv_scores[best_hidden_layer_size]

best_hidden_layer_size, best_accuracy
```

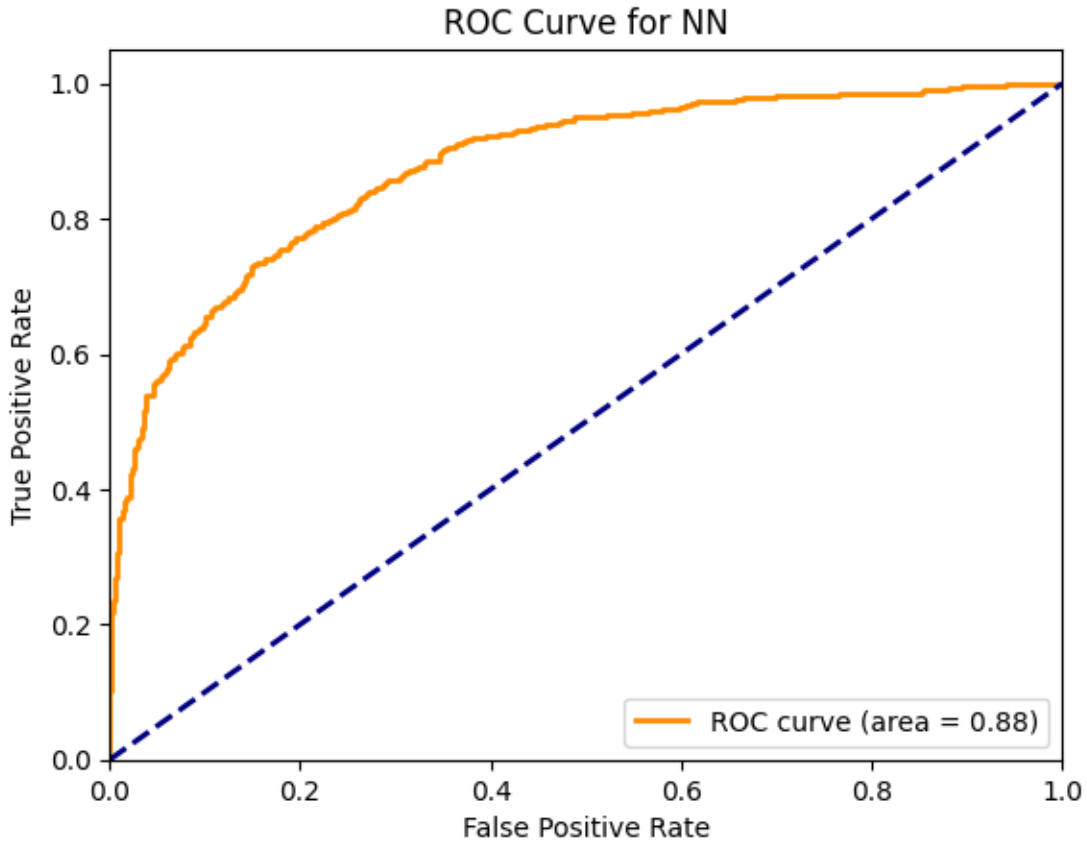
[ ]:
```
((10,), 0.7886159348383363)
```

[ ]:
```
import time
# Record the start time
start_time = time.time()

#best_nn_model = MLPClassifier(hidden_layer_sizes= best_hidden_layer_size,␣
 ↪max_iter=1000, random_state=42)
best_nn_model = MLPClassifier(hidden_layer_sizes= 10, max_iter=1000,␣
 ↪random_state=42)
best_nn_model.fit(X_train, y_train)

# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")

nn_val_pred = best_nn_model.predict(X_val)
nn_val_accuracy = accuracy_score(y_val, nn_val_pred)
```

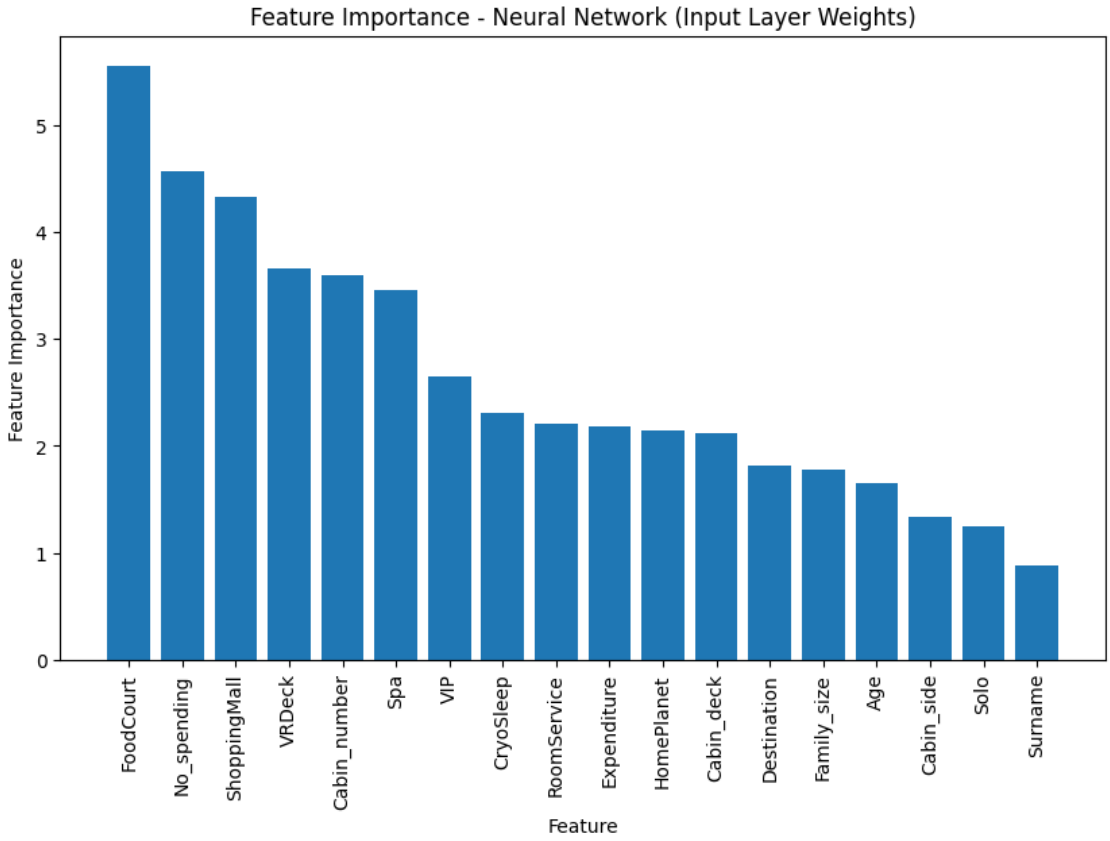
```
print("NN validation accuracy:", nn_val_accuracy)
```

```
Training time: 3.796481132507324 seconds
NN validation accuracy: 0.7843588269120184
```

[ ]:
```
# Plot ROC curve for the selected model (Random Forest in this case)
nn_probs = best_nn_model.predict_proba(X_val)[:, 1]
fpr, tpr, thresholds = roc_curve(y_val, nn_probs)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for NN')
plt.legend(loc="lower right")
plt.show()
```



[ ]:
```
# Get the weights connecting input layer to the first hidden layer
input_layer_weights = best_nn_model.coefs_[0]

# Compute feature importance based on the sum of absolute weights for each␣
 ↪feature
feature_importance = np.sum(np.abs(input_layer_weights), axis=1)

# Get the feature names
feature_names = X_train.columns

# Sort feature importance and feature names in descending order
sorted_indices = feature_importance.argsort()[::-1]
sorted_feature_importance = feature_importance[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(range(len(sorted_feature_importance)), sorted_feature_importance,␣
 ↪align='center')
plt.xticks(range(len(sorted_feature_importance)), sorted_feature_names,␣
 ↪rotation =90)
plt.ylabel('Feature Importance')
plt.xlabel('Feature')
plt.title('Feature Importance - Neural Network (Input Layer Weights)')
plt.show()
```



1.4 Siyan: gradientboosting (LGBMClassifier+XGBClassifier)

[ ]:
```
X_train.head()
```

[ ]:
```
      HomePlanet  CryoSleep  Destination       Age       VIP  RoomService  \
4561    0.440385  -0.732770     0.620545  0.084356 -0.153063     2.458002
3305   -0.817259   1.364685     0.620545 -0.543234 -0.153063    -0.638181
2225   -0.817259   1.364685     0.620545 -0.822163 -0.153063    -0.638181
447    -0.817259  -0.732770     0.620545 -0.473502 -0.153063    -0.638181
1373   -0.817259  -0.732770     0.620545  2.106589 -0.153063    -0.638181

      FoodCourt  ShoppingMall       Spa    VRDeck  Expenditure  No_spending  \
4561   1.177977      0.311020  1.164139  0.293812     1.151670    -0.851353
3305  -0.650080     -0.622995 -0.664035 -0.640034    -1.152845     1.174601
2225  -0.650080     -0.622995 -0.664035 -0.640034    -1.152845     1.174601
447   -0.039138     -0.622995  2.004663 -0.640034     0.851217    -0.851353
1373  -0.650080     -0.622995 -0.664035  1.773101     0.643792    -0.851353

          Solo  Cabin_deck  Cabin_number  Cabin_side   Surname  Family_size
4561  0.899532   -1.320046     -0.749609   -1.032865 -0.079495    -0.228971
```

```
3305 -1.111690    0.932414      0.756244    0.968181 -1.205166    -0.132793
2225 -1.111690    0.932414      0.345198    0.968181  0.695967    -0.004555
447   0.899532   -0.193816      0.163389   -1.032865  0.361393    -0.196912
1373  0.899532    0.932414      0.003318   -1.032865  1.136855    -0.068674
```

[ ]:
```
X_train.columns
```

[ ]:
```
Index(['HomePlanet', 'CryoSleep', 'Destination', 'Age', 'VIP', 'RoomService',
       'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck', 'Expenditure',
       'No_spending', 'Solo', 'Cabin_deck', 'Cabin_number', 'Cabin_side',
       'Surname', 'Family_size'],
      dtype='object')
```

1.4.1 LGBMClassifier

[ ]:
```
import lightgbm as lgb
from sklearn.metrics import classification_report
```

[ ]:
```
# Define the parameter grid for grid search
param_grid = {
    'objective': ['binary'],
    'boosting_type': ['gbdt'],
    'num_leaves': [30, 50, 75,100],
    'learning_rate': [0.05, 0.1, 0.2],
    'feature_fraction': [0.9, 0.8, 0.7,0.6,0.5]
}

cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
# Create an LGBMClassifier
clf = lgb.LGBMClassifier(verbose=-1)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid,␣
 ↪scoring='accuracy', cv=cv)

# Fit the model to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Get the best parameters and best estimator
best_params = grid_search.best_params_
best_clf = grid_search.best_estimator_

# Print the best parameters
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'boosting_type': 'gbdt', 'feature_fraction': 0.9,
'learning_rate': 0.1, 'num_leaves': 30, 'objective': 'binary'}
Training time: 0.18375754356384277 seconds
```

```
Cross-Validation Scores: [0.82758621 0.8045977  0.80028736 0.81896552 0.82877698
0.78273381
 0.8057554  0.79136691 0.82158273 0.80143885]
Mean CV Score: 0.8083091457868188
Standard Deviation of CV Scores: 0.014674015531041905
```

[ ]:
```
# Record the time
start_time = time.time()

best_clf = lgb.LGBMClassifier(verbose=-1,
                              boosting_type = 'gbdt',
                              feature_fraction= 0.9,
                              learning_rate = 0.1,
                              num_leaves = 30,
                              objective = 'binary')
# Fit the best model on the entire training set
best_clf.fit(X_train, y_train)

# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")

# Perform cross-validation with the best model
cross_val_scores = cross_val_score(best_clf, X_train, y_train, cv=cv,␣
 ↪scoring='accuracy')

# Print the cross-validation scores
print("Cross-Validation Scores:", cross_val_scores)

# Print the mean and standard deviation of cross-validation scores
print("Mean CV Score:", cross_val_scores.mean())
print("Standard Deviation of CV Scores:", cross_val_scores.std())
```

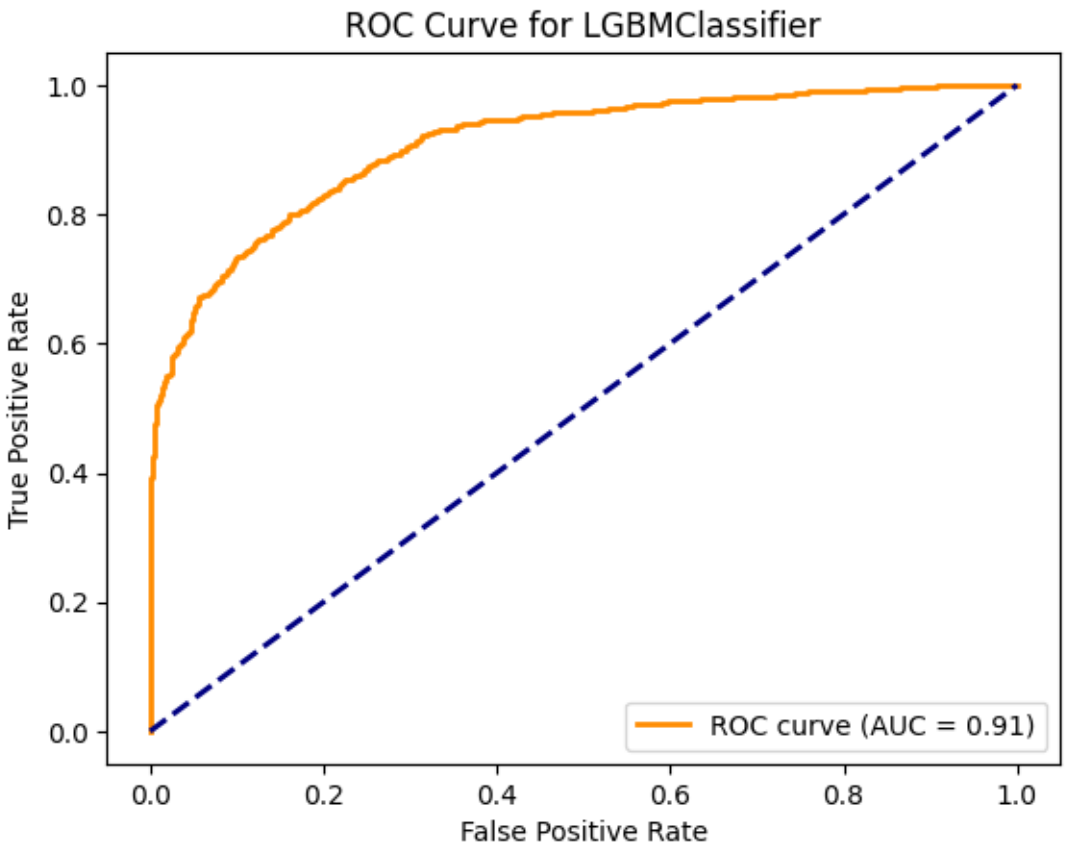
```
Training time: 0.5622401237487793 seconds
Cross-Validation Scores: [0.82758621 0.8045977  0.80028736 0.81896552 0.82877698
0.78273381
 0.8057554  0.79136691 0.82158273 0.80143885]
Mean CV Score: 0.8083091457868188
Standard Deviation of CV Scores: 0.014674015531041905
```

[ ]:
```
predictions = best_clf.predict(X_val)
print(classification_report(y_val, predictions))
```

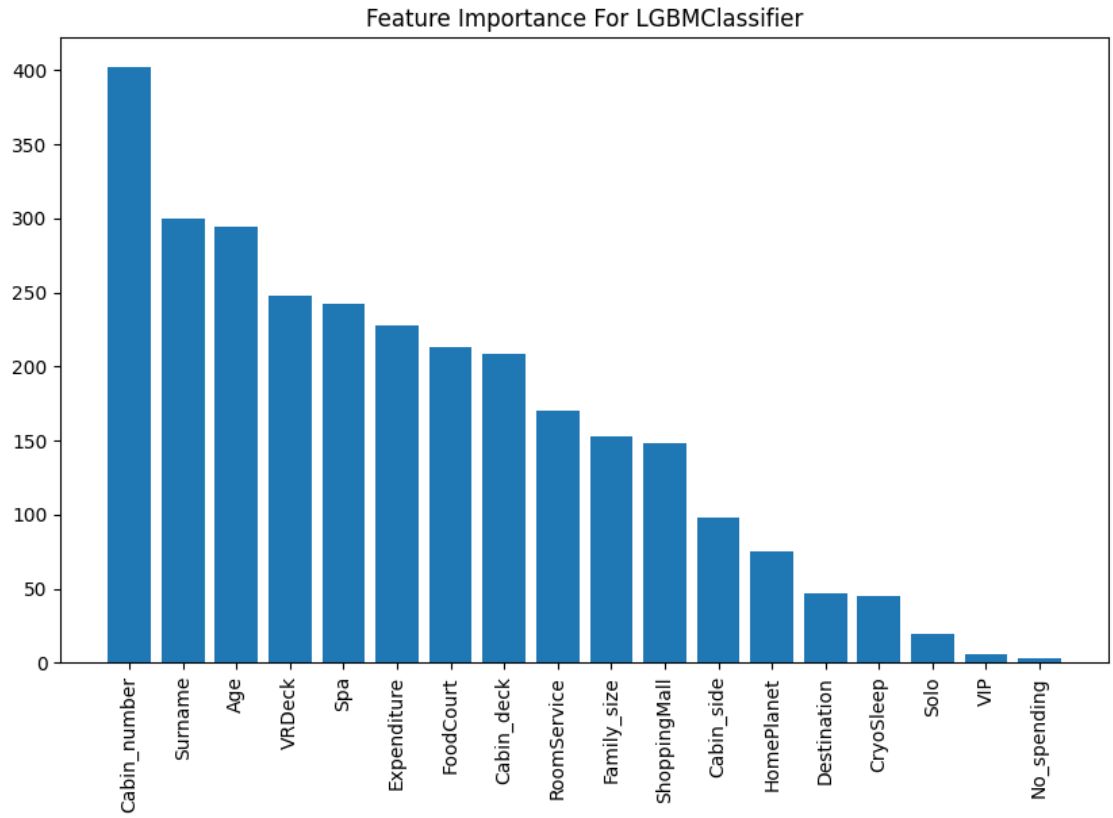
```
              precision    recall  f1-score   support

       False       0.82      0.81      0.81       863
        True       0.81      0.82      0.82       876
```

```
    accuracy                           0.81      1739
   macro avg       0.81      0.81      0.81      1739
weighted avg       0.81      0.81      0.81      1739
```

[ ]:
```python
## ROC of LGBMClassifier
# Predict probabilities on the validation set
from sklearn.metrics import roc_curve, auc
y_prob = best_clf.predict_proba(X_val)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.
 ↪2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for LGBMClassifier')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
# Get feature importance
feature_importances = best_clf.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(feature_importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]

# Plot
plt.figure(figsize=(10, 6))
plt.title("Feature Importance For LGBMClassifier")
plt.bar(range(X_train.shape[1]), feature_importances[indices])
plt.xticks(range(X_train.shape[1]), names, rotation=90)
plt.show()
```



1.4.2 XGBClassifier

[ ]:
```
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, StratifiedKFold,␣
↪train_test_split,GridSearchCV
```

[ ]:
```
y_train.head()
```

[ ]:
```
3600     True
1262     True
8612    False
5075     True
4758    False
Name: Transported, dtype: bool
```

[ ]:
```
# create model instance
# Initialize cross-validation
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
from sklearn.model_selection import GridSearchCV
```

```
# Define the XGBoost classifier
xgb_model = XGBClassifier()

# Define the parameter grid to search
param_grid = {
    'n_estimators': [2,5,10,30,50, 100, 200],
    'max_depth': [3, 5, 7,9],
    'learning_rate': [0.01, 0.1, 0.2,0.5,0.7,1,2,3],
    # Add other hyperparameters you want to tune
}

# Create the GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid,␣
 ↪scoring='accuracy', cv=cv)

# Fit the model to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators':
200}
```

[ ]:
```
# Get the best model from the grid search
best_model = XGBClassifier(n_estimators = 200,max_depth = 5,learning_rate = 0.1)

# Record the time
start_time = time.time()

# Fit the best model on the entire training set
best_model.fit(X_train, y_train)

# Record the end time
end_time = time.time()
# Calculate the duration
duration = end_time - start_time
print("Training time:", duration, "seconds")

# Make predictions on the validation set
predictions = best_model.predict(X_val)

# Perform cross-validation with the best model
cross_val_scores = cross_val_score(best_model, X_train, y_train, cv=cv,␣
 ↪scoring='accuracy')

# Print the cross-validation scores
```

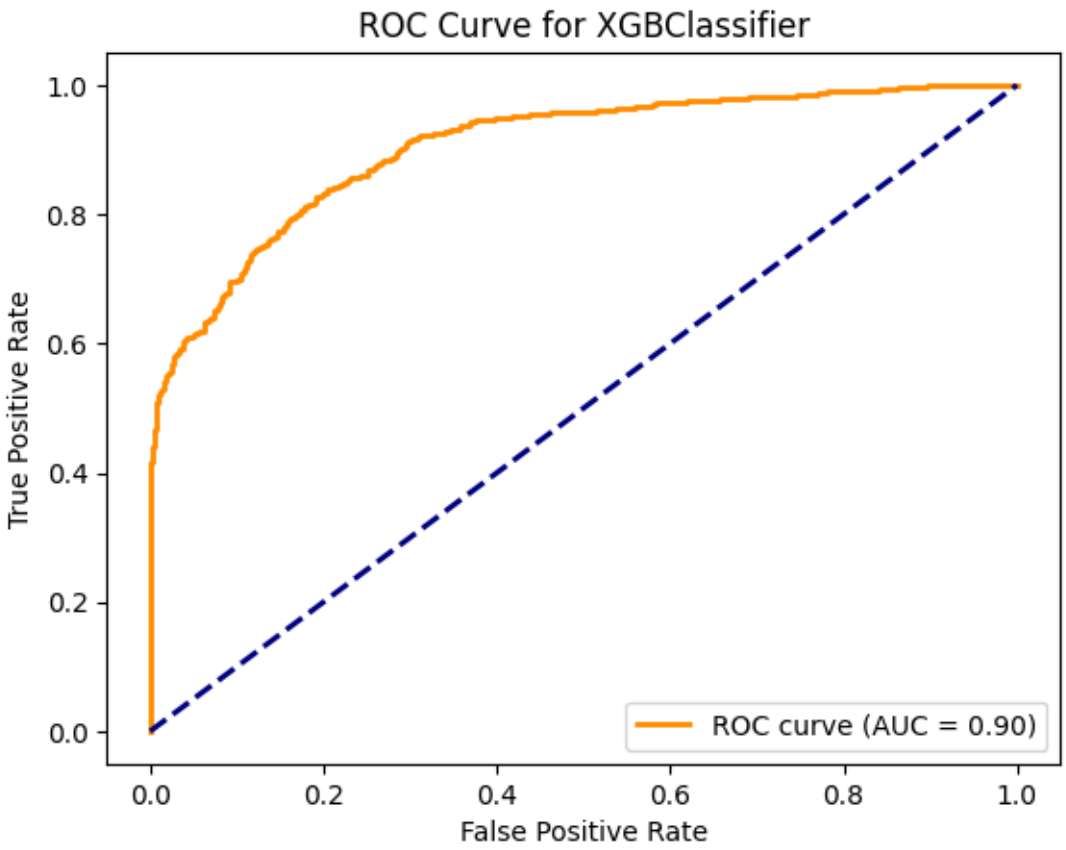
```
print("Cross-Validation Scores:", cross_val_scores)

# Print the mean and standard deviation of cross-validation scores
print("Mean CV Score:", cross_val_scores.mean())
print("Standard Deviation of CV Scores:", cross_val_scores.std())
```

```
Training time: 6.281451463699341 seconds
Cross-Validation Scores: [0.82183908 0.79741379 0.80747126 0.82471264 0.82302158
0.78273381
 0.80863309 0.78417266 0.81582734 0.8028777 ]
Mean CV Score: 0.8068702968659556
Standard Deviation of CV Scores: 0.01445715691287694
```

[ ]:
```
# Print classification report or other metrics
# Make predictions on the validation set
predictions = best_model.predict(X_val)
print(classification_report(y_val, predictions))
```

```
              precision    recall  f1-score   support

       False       0.81      0.82      0.81       863
        True       0.82      0.81      0.82       876

    accuracy                           0.81      1739
   macro avg       0.81      0.81      0.81      1739
weighted avg       0.81      0.81      0.81      1739
```

[ ]:
```
## ROC of XGB
# Predict probabilities on the validation set
from sklearn.metrics import roc_curve, auc
y_prob = best_model.predict_proba(X_val)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_val, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.
  ↪2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for XGBClassifier')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'best_clf_xgb' is your best XGBoost model obtained from GridSearchCV

# Get feature importance
feature_importances = best_model.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(feature_importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]

# Plot
plt.figure(figsize=(10, 6))
```

```
plt.title("Feature Importance for XGBClassifier")
plt.bar(range(X_train.shape[1]), feature_importances[indices])
plt.xticks(range(X_train.shape[1]), names, rotation=90)
plt.show()
```