PDS Challenge 4:

Trees and Forests-- Spaceship Titanic Data Science Challenge

Group 5 蔡長頤、方品仁、劉資浩

Video link:

https://drive.google.com/file/d/1SQ4fVSwDGpr3Nx88O6WkW0vRIeoXGWTt/view?usp=sharing

Table of contents:

1. Introduction

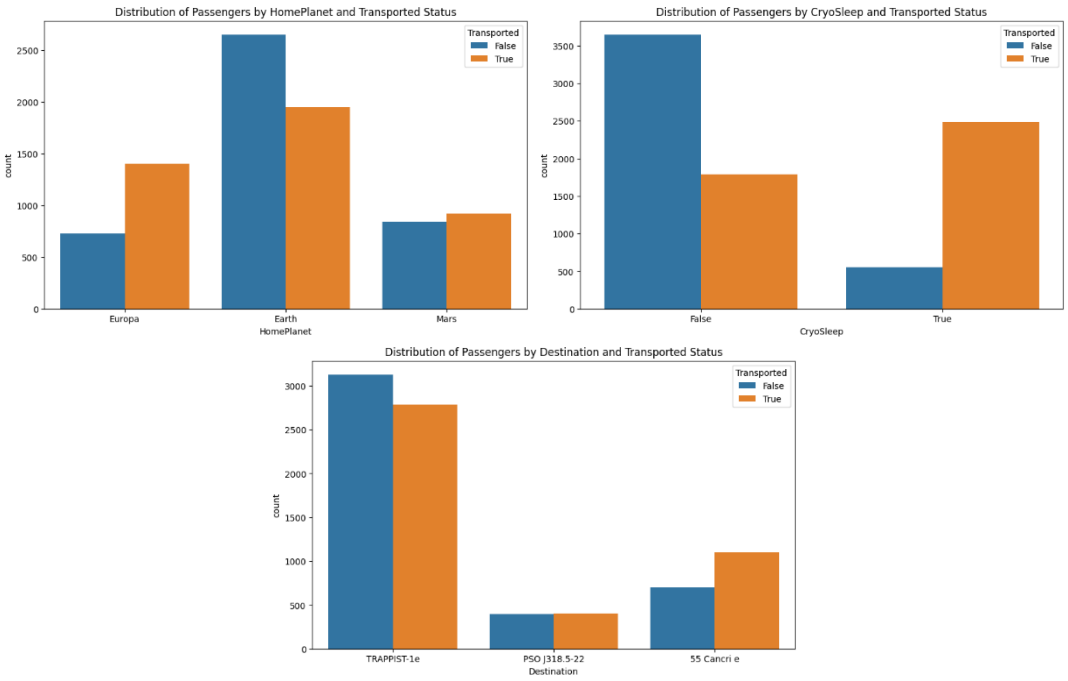
The challenge is set in the year 2912 and involves the Spaceship Titanic, which encountered a spacetime anomaly, leading to passengers being transported to an alternate dimension. The objective is to use decision trees and random forests to predict which passengers were transported.

2. Data Understanding and Visualization

- **Identify Missing Values**: Columns except `Passenger Id` and `Transported` have missing values. The number of missing values in each column is shown in the below.

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

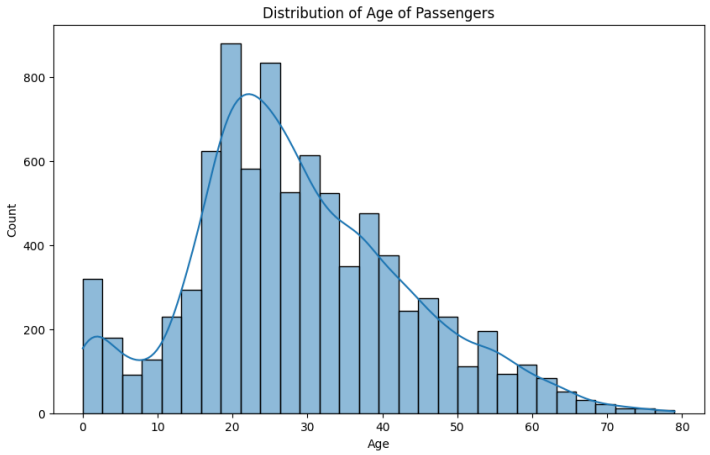
- **Basic Statistics:**
  - The amenities (`RoomService`, `FoodCourt`, `ShoppingMall`, `Spa`, `VRDeck`) are numerical and seem to have a broad range of values, indicating varied spending habits.
- Also, for the convenience of further computation, we split the 'Cabin' column into 'Deck', 'Num', and 'Side'.
- **Data visualization:**
  1.Distribution between features and target:



As we can see in the figure of the distribution of CryoSleep vs Transported, CryoSleep seems to be highly related to transported. If CryoSleep = False, more likely the passenger is not transported, while if CryoSleep = True, more likely the passenger is transported.

2. Target Variable Distribution:

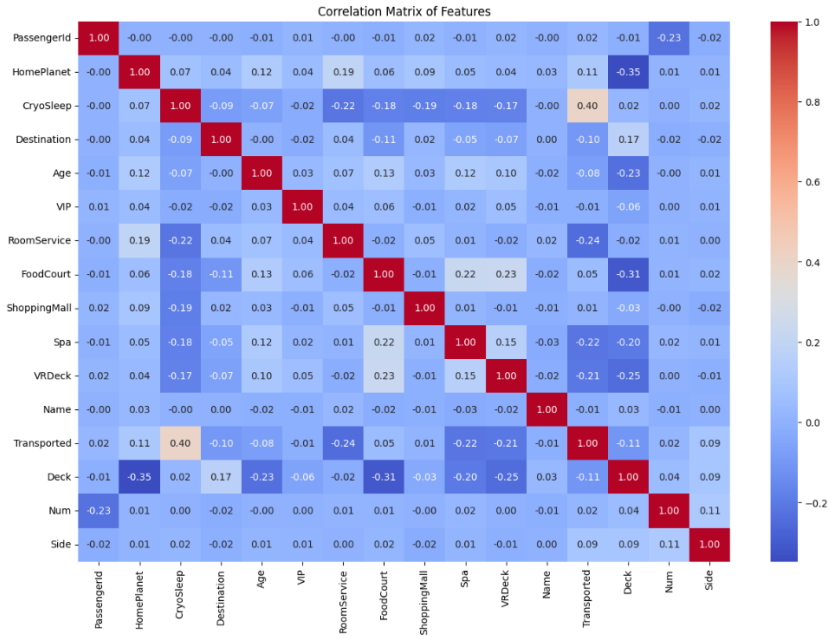
The target variable `Transported` is fairly balanced.with approximately 50.36% of passengers being transported.



3. Exploratory Data Analysis (EDA)

Correlation Analysis: A heatmap of the correlation matrix was plotted to understand the relationships between different features.

We can see a relatively high correlation between CryoSleep and Transported. This observation is identical to the observation in the figure about the distribution of CryoSleep vs Transported provided in the previous section.



4. Feature Engineering and Preprocessing: Imputation and One-hot encoding

Categorical and Numerical Columns: Identified categorical and numerical columns for separate preprocessing.

- categorical_cols = ['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'Deck', 'Side']
- numerical_cols = ['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']

Preprocessing Steps:

- Categorical columns were processed with a pipeline that included imputation (using the most frequent strategy) and one-hot encoding.
- Numerical columns were imputed using the median strategy.
- Column Transformer: A ColumnTransformer combined these preprocessing steps for both types of columns.

5. Decision Tree

(1) Model Implementation:

- A Decision Tree Classifier was created using scikit-learn's tree module.

(2)Model Evaluation: Cross-validation with 5 folds was also used to evaluate the Decision Tree model, using accuracy as the scoring metric.

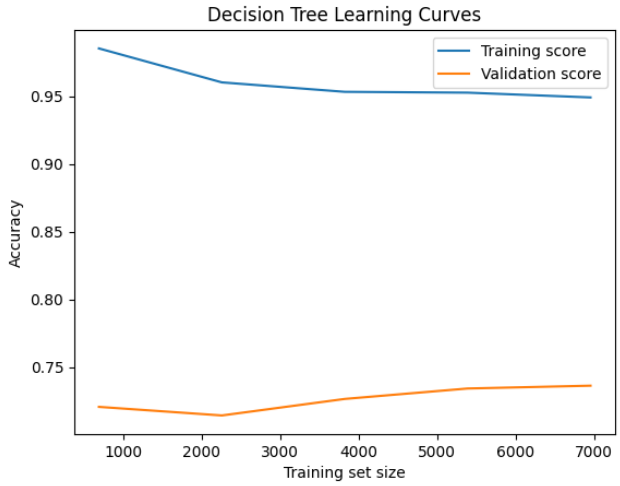
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

# Create a DecisionTreeClassifier
decision_tree_model = DecisionTreeClassifier(random_state=42)

# Plot learning curves for the decision tree
train_sizes, train_scores, validation_scores = learning_curve(
        estimator=decision_tree_model,
        X=X_train,    # Your feature matrix
        y=y_train,    # Your target variable
        train_sizes=np.linspace(0.1, 1.0, 5),
        cv=5,
        scoring='accuracy'
)
```

- **Average cross-validation score =** `0.7363392847098745`

(3) Learning Curve: visualize the performance of the Decision Tree model.

The learning curves were plotted against different training set sizes, ranging from 10% to 100% of the training data, to observe how the model learns as more data is provided. Also, the curves displayed both the training score and the validation score, which helps in understanding the model's learning behavior and whether it is underfitting or overfitting.

As we can see from the learning curve, the validation score increases as the training set size increases. This indicates that our model is improving. Also, there is a big gap between the accuracy between the training and validation score.



6. Random Forests:

(1) Model Implementation:

- Initialized the Random Forest Classifier from scikit-learn's ensemble module.
- The model was set up with 100 estimators (n_estimators=100), which indicates it uses 100 individual decision trees.

```
random_forest_model = RandomForestClassifier(n_estimators=100,
random_state=42)
```

- **Parameter tuning:**
  - `n_estimators:`

    Through various tries, we concluded that `n_estimators=100` performs better than other values.

    This parameter specifies the number of trees in the forest. Generally, increasing the number of trees (n_estimators) can improve the model's accuracy, as the ensemble's predictions become more stable and robust. Each additional tree contributes to an average prediction, reducing the variance and the risk of overfitting. However, there is a trade-off. More trees increase computational complexity, meaning the model will take longer to train and require more memory. Beyond a certain point, the marginal gain in accuracy diminishes.
  - `max_depth:`
    The maximum depth of each tree. Deeper trees can capture more complex patterns but might lead to overfitting. Not setting this parameter means unlimited depth.

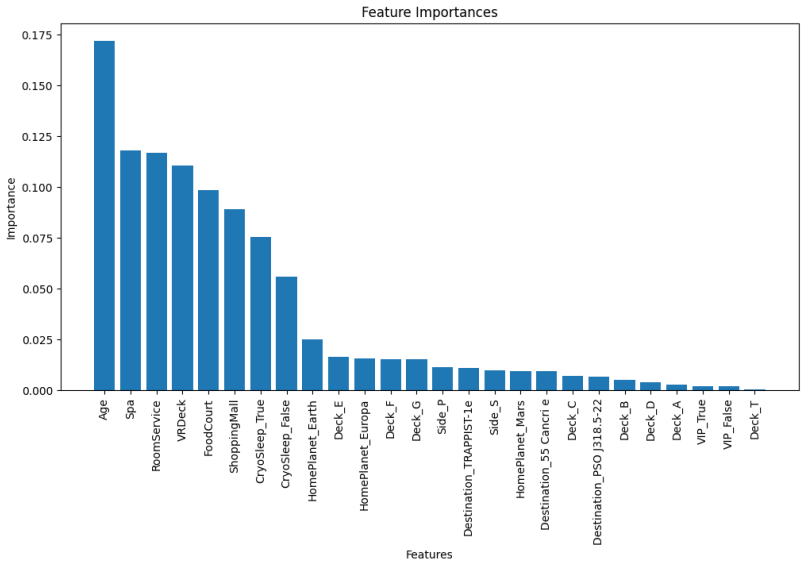
(2) Model Evaluation:

We employed **cross-validation** with 5 folds (cv=5) to assess the model's performance. This approach splits the training data into 5 parts, trains the model on 4 parts, and validates it on the 5th part, repeating this process 5 times.

- The scoring metric used for evaluation was 'accuracy'.

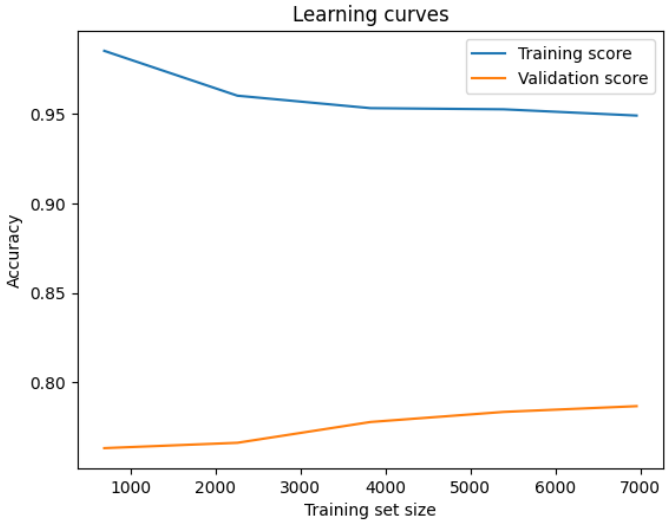
- The mean of the cross-validation scores was calculated to get an overall performance metric for the model.
- **Average cross-validation score =** `0.7873005463902313`

(3) Feature importance:



(4) Learning Curve:

General trend is similar to that of decision tree model.



7. Comparison of the 2 models:

The observation that the Random Forest model performed slightly better than the Decision Tree in our project is consistent with general expectations in machine learning. Here's a comparison of the two models and potential reasons for the superior performance of Random Forest:

(1) Diversity of Trees: The aggregation of multiple trees in Random Forest helps capture a broader range of patterns and relationships in the data, leading to better generalization.

(2) Reduced Overfitting: The model averages out biases and reduces variance, leading to improved accuracy on test data compared to a single decision tree.
(3) Robustness to Noise: Random Forest is more robust to noise and outliers than a single decision tree.
(4) Better Handling of Unbalanced Datasets: If the dataset is unbalanced, Random Forest can handle this better than a single decision tree.

8. Additional work: kNN

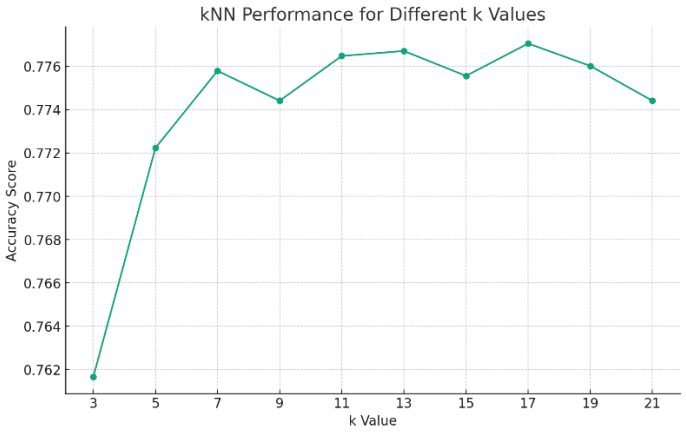
(1) Model Implementation:

- A loop was implemented to try different values of k (number of neighbors) ranging from 3 to 21, increasing by 2 each time.
- K-Fold cross-validation (with 5 splits) was used to evaluate the model's performance for each value of k.
- For each fold, the dataset was split into training and validation sets, and the kNN model was trained and evaluated.

```
# kNN
kf = KFold(n_splits=5)

X = X.values
y = y.values



X = preprocessing.scale(X)

all_score = {}

for k in range(3, 22, 2):
        score = []

        for train_indices, validation_indices in kf.split(X):
                X_train, X_validation, y_train, y_validation = X[train_indices, :], X[validation_indices, :], y[train_indices], y[validation_indices]
                knnclf = KNeighborsClassifier(n_neighbors=k, algorithm="kd_tree")
                knnclf.fit(X_train, y_train.ravel())
                score.append(knnclf.score(X_validation, y_validation))

        all_score[k] = sum(score)/len(score)
```

(2) Result:

From the plot, we can observe how the accuracy of the kNN model changes as the number of neighbors varies. It appears that the model achieves the highest accuracy when k =17, with an accuracy score of approximately **0.7771**. This kind of visualization is helpful for selecting the optimal k value in kNN models, balancing the trade-off between underfitting and overfitting.



9. Additional work: Logistic Regression

(1) Model Implementation:

- A Logistic Regression model was initialized with specific parameters: C=0.5, penalty="l2", and solver="newton-cg".
- C=0.5 indicates regularization strength (inverse of regularization parameter).
- penalty="l2" implies using L2 regularization.
- solver="newton-cg" specifies the algorithm used for optimization.

```
#logistic regression
all_score = []
for train_indices, validation_indices in kf.split(X):
        X_train, X_validation, y_train, y_validation = X[train_indices, :], X[validation_indices, :], y[train_indices], y[validation_indices]
        LR = LogisticRegression(C=0.5, penalty="l2", solver="newton-cholesky")
        LR.fit(X_train, y_train.ravel())
        all_score.append(LR.score(X_validation, y_validation.ravel()))

print(all_score)
print(sum(all_score)/len(all_score))
```

```
[0.7763082231167338, 0.78205865439908, 0.7975848188614146, 0.7807825086306099, 0.7756041426927502]
0.7824676695401177
```

(2) Result:

Average accuracy = 0.7824676695401177