Importing The Important Modules From The Library

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

Load The Dataset

```
df=pd.read_csv("Titanic-Dataset.csv")#Load the dataset

df.head()#Check the head
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                           Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
# check the total row and the column of the dataset
df.shape
```

```
(891, 12)
```

Data Cleaning

```
#check the missing value
mv=df.isnull().sum()
print(mv)
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
# Fill missing values in 'Age' with their respective means
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Let's also fill 'Embarked' with its mode, as it's categorical
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Drop 'Cabin' due to too many missing values
df.drop(columns=['Cabin'], inplace=True)

# Check if any missing values remain
ms = df.isnull().sum()
ms
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

```
df.head()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
```

```
3              4          1       1
4              5          0       3

                                                     Name     Sex   Age
SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                              Heikkinen, Miss. Laina  female  26.0
0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                            Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Embarked
0      0         A/5 21171   7.2500        S
1      0          PC 17599  71.2833        C
2      0  STON/O2. 3101282   7.9250        S
3      0            113803  53.1000        S
4      0            373450   8.0500        S
```

```
df.info() #for knowing the information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Embarked     891 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

Encode the dataset

```
print(df.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
'SibSp',
```

```
       'Parch', 'Ticket', 'Fare', 'Embarked'],
      dtype='object')

#for binary categories like Sex put Male equals to 0 and female equals
to 1
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
# for Embarked, which has more than two values like C, Q, S so we use
dummy veriable
df = pd.get_dummies(df, columns=['Embarked'], drop_first=True)
#It automatically dropped the first category (likely 'C') to avoid the
dummy variable trap (multicollinearity).
# Embarked_Q = 1 → Passenger embarked at Queenstown

# Embarked_S = 1 → Passenger embarked at Southampton

# If both are 0, it means the passenger embarked at Cherbourg (C).

df.head()

   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name  Sex   Age  SibSp
Parch  \
0                            Braund, Mr. Owen Harris    0  22.0      1
0
1  Cumings, Mrs. John Bradley (Florence Briggs Th...    1  38.0      1
0
2                             Heikkinen, Miss. Laina    1  26.0      0
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    1  35.0      1
0
4                           Allen, Mr. William Henry    0  35.0      0
0

             Ticket     Fare  Embarked_Q  Embarked_S
0         A/5 21171   7.2500           0           1
1          PC 17599  71.2833           0           0
2  STON/O2. 3101282   7.9250           0           1
3            113803  53.1000           0           1
4            373450   8.0500           0           1
```

Find the Correlation

```
# Correlation with Survived
correlation = df[['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex',
```

```
'Embarked_Q', 'Embarked_S', 'Survived']].corr()
print(correlation['Survived'].sort_values(ascending=False)) #
ascending = false means order will be highest to lowest

Survived      1.000000
Sex           0.543351
Fare          0.257307
Parch         0.081629
Embarked_Q    0.003650
SibSp        -0.035322
Age          -0.069809
Embarked_S   -0.149683
Pclass       -0.338481
Name: Survived, dtype: float64
```

Split the dataset

```
# Define features (X) and target (y)
X = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
'Embarked_Q', 'Embarked_S']]
y = df['Survived']

#Split data (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# check the shape of the training data
X_train.shape

(623, 8)

# check the shape of the testing data
X_test.shape

(268, 8)
```

Model Buiding

At the first time I use logistic regression model and then I use the random forest model for better model prediction making

Logistic Regression Model

```
# Initialize and train the model
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)
```

Test the model

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Evaluate Logistic Regression
print("Logistic Regression Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Logistic Regression Performance:
Accuracy: 0.8097014925373134

Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.87      0.84       157
           1       0.79      0.73      0.76       111

    accuracy                           0.81       268
   macro avg       0.81      0.80      0.80       268
weighted avg       0.81      0.81      0.81       268


Confusion Matrix:
 [[136  21]
 [ 30  81]]
```

Checking the model with new data

```
new_passenger = pd.DataFrame({
    'Pclass': [3], 'Sex': [0], 'Age': [25], 'SibSp': [0], 'Parch':
[0],
    'Fare': [7.5], 'Embarked_Q': [0], 'Embarked_S': [1]
})
prediction = model.predict(new_passenger)
print("Survived" if prediction[0] == 1 else "Did Not Survive")
```

```
Did Not Survive
```

```
new_passenger = pd.DataFrame({
    'Pclass': [3], 'Sex': [1], 'Age': [15], 'SibSp': [0], 'Parch':
[0],
    'Fare': [8.5], 'Embarked_Q': [1], 'Embarked_S': [1]
})
prediction = model.predict(new_passenger)
print("Survived" if prediction[0] == 1 else "Did Not Survive")
```

```
Survived
```

Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize and train Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test set
y_pred_rf = rf_model.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# Optionally, evaluate Random Forest
print("\nRandom Forest Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test,
y_pred_rf))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest Performance:
Accuracy: 0.7910447761194029

Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.84      0.83       157
           1       0.76      0.72      0.74       111

    accuracy                           0.79       268
   macro avg       0.79      0.78      0.78       268
weighted avg       0.79      0.79      0.79       268


Confusion Matrix:
 [[132  25]
 [ 31  80]]
```

```python
new_passenger = pd.DataFrame({
    'Pclass': [3], 'Sex': [0], 'Age': [25], 'SibSp': [0], 'Parch':
[0],
    'Fare': [7.5], 'Embarked_Q': [0], 'Embarked_S': [1]
})
prediction = model.predict(new_passenger)
print("Survived" if prediction[0] == 1 else "Did Not Survive")
```

```
Did Not Survive
```

```
new_passenger = pd.DataFrame({
    'Pclass': [3], 'Sex': [1], 'Age': [15], 'SibSp': [0], 'Parch':
[0],
    'Fare': [8.5], 'Embarked_Q': [1], 'Embarked_S': [1]
})
prediction = model.predict(new_passenger)
print("Survived" if prediction[0] == 1 else "Did Not Survive")
```

```
Survived
```

Conclusion

When to choose Logistic Regression (accuracy = 0.81):

Simpler, faster, and easier to interpret.

Works well if the relationship between features and outcome is linear.

Less prone to overfitting with smaller datasets.

Good if you want to explain results clearly.

When to choose Random Forest (accuracy = 0.79):

Handles non-linear relationships and interactions better.

More robust to outliers and noisy data.

Better if your data is large, complex, or includes categorical features.

Can rank feature importance.

So, I thik logistic regression is slightly better then random forest model. I always prefer the logistic regreesion model