TITANIC SURVIVAL PREDICTION

Objective:

The goal of this case study is to predict whether a passenger survived the Titanic disaster or not, based on various features such as gender, age, class, and other passenger characteristics. Machine learning models, specifically logistic regression and random forest classifiers, are applied to achieve this goal.

Data Dictionary:

The Titanic dataset contains the following features:

- **Survived**: The target variable. Whether the passenger survived (1) or not (0).
- **Pclass**: The passenger's class (1 = 1st, 2 = 2nd, 3 = 3rd).
- **Sex**: Gender of the passenger (male or female).
- **Age**: The age of the passenger in years.
- **SibSp**: The number of siblings/spouses aboard the Titanic.
- **Parch**: The number of parents/children aboard the Titanic.
- **Fare**: The fare the passenger paid for the ticket.
- **Embarked**: The port where the passenger boarded (C = Cherbourg; Q = Queenstown; S = Southampton).
- **Cabin**: The cabin the passenger stayed in (many missing values).
- **Name**: Name of the passenger (not used in the model).
- **Ticket**: The ticket number (not used in the model).
- **PassengerId**: The unique ID for each passenger (not used in the model).

Task Steps:

1. **Data Exploration**: Understand the structure of the dataset.
2. **Data Preprocessing**: Clean the dataset by handling missing values, encoding categorical variables, and scaling numerical features.
3. **Model Training**: Train machine learning models, such as Logistic Regression and Random Forest.
4. **Model Evaluation**: Evaluate the performance of the models using classification metrics.
5. **Model Optimization**: Fine-tune the Random Forest model to improve its performance

Packages:

- **pandas**: For data manipulation and analysis (reading CSV files, handling missing data).
- **numpy**: For numerical operations (e.g., handling arrays, missing values).
- **matplotlib**: For creating static, interactive, and animated visualizations.
- **seaborn**: For statistical data visualization built on top of matplotlib.
- **sklearn**: For machine learning algorithms, data preprocessing, splitting datasets, and evaluating models.

- **train_test_split**: To split the data into training and test sets.
- **LogisticRegression**: To implement the Logistic Regression model.
- **RandomForestClassifier**: For implementing the Random Forest Classifier.
- **StandardScaler**: For scaling numerical features.
- **LabelEncoder**: For encoding categorical variables.
- **accuracy_score, precision_score, recall_score, f1_score, classification_report**: For model evaluation metrics.

#Loading the Dataset and Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

 # Load the dataset

```
data = pd.read_csv('Titanic-Dataset.csv')
print(data)
```

 # Exploring the Dataset

```
print("Dataset Head:\n", data.head())
print("\nDataset Info", data.info())
print("\nMissing Values:\n", data.isnull().sum())
print(data.describe())
```

Output:

Dataset first few rows:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Range Index: 891 entries, 0 to 890

Data columns (total 12 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	714 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 Cabin 204 non-null object

11 Embarked 889 non-null object

dtypes: float64(2), int64(5), object(5)

memory usage: 83.7+ KB

Dataset Info None

Missing Values:

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

PassengerId Survived Pclass Age SibSp Parch Fare

count 891.000000 891.000000 891.000000 714.000000 891.000000 891.000000 891.000000

mean 446.000000 0.383838 2.308642 29.699118 0.523008 0.381594 32.204208

std 257.353842 0.486592 0.836071 14.526497 1.102743 0.806057 49.693429

min 1.000000 0.000000 1.000000 0.420000 0.000000 0.000000 0.000000

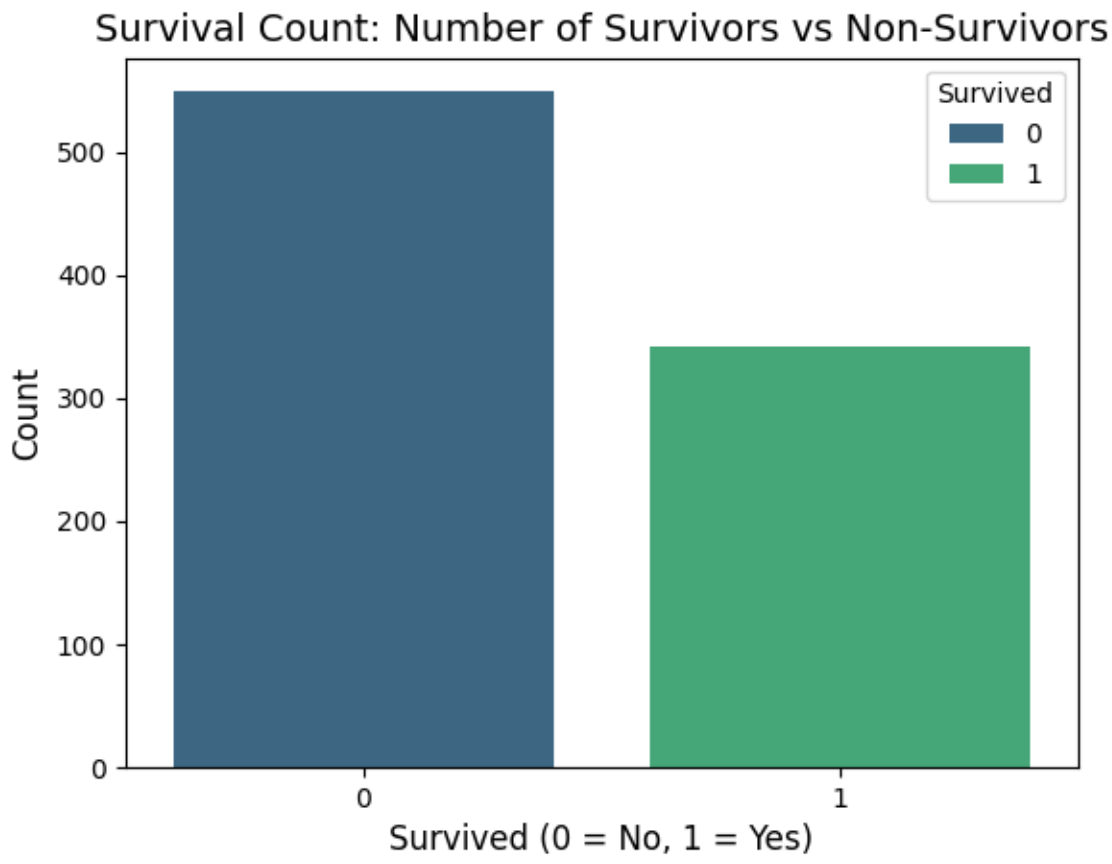
25% 223.500000 0.000000 2.000000 20.125000 0.000000 0.000000 7.910400

50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

#3. Data Visualization

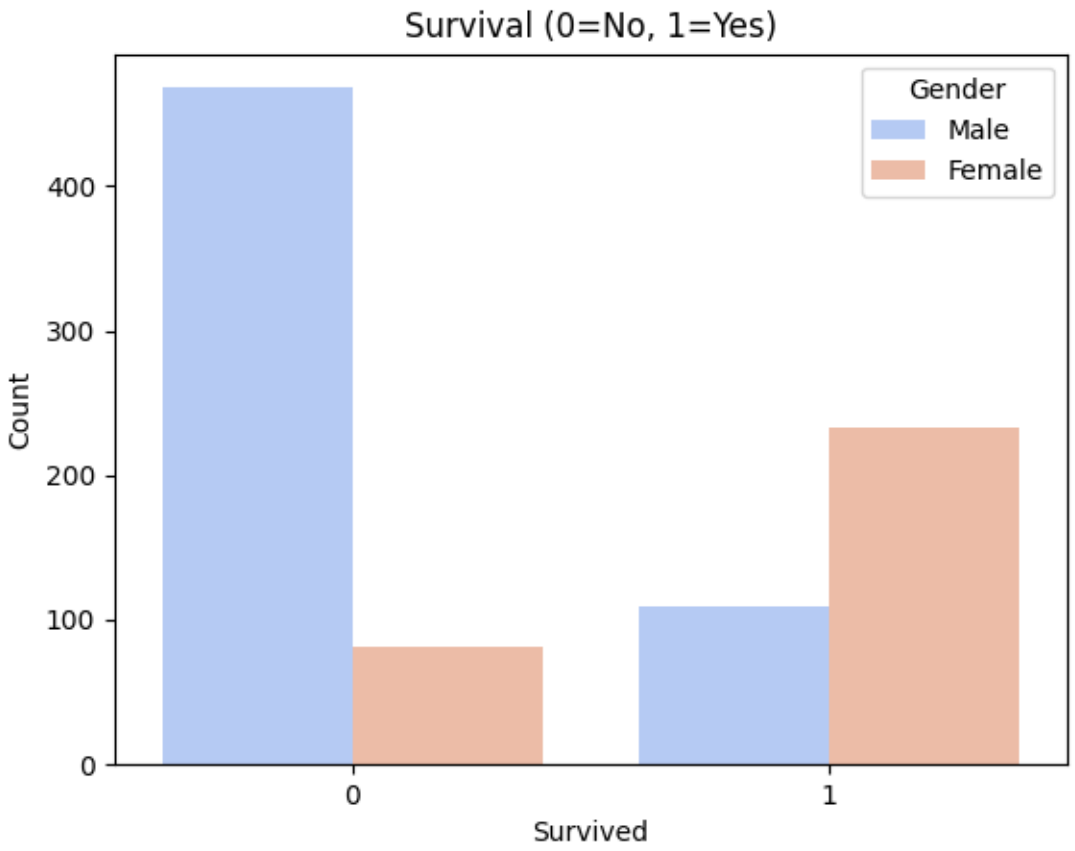
#Survival Distribution

```
sns.countplot(x='Survived', hue='Survived', data=data, palette='viridis',
legend=True)
plt.title('Survival Count: Number of Survivors vs Non-Survivors', fontsize=14)
plt.xlabel('Survived (0 = No, 1 = Yes)', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show()
```



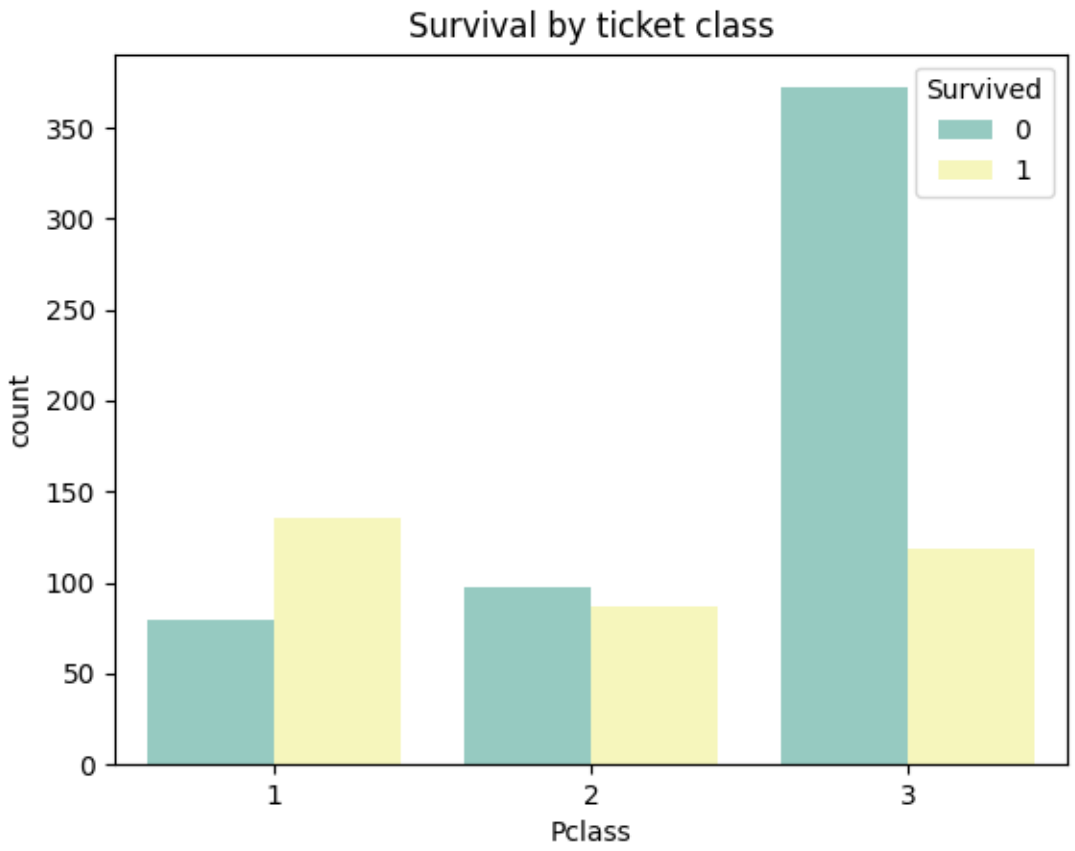
 # Survival by Gender

```
sns.countplot(data=data, x='Survived', hue='Sex', palette='coolwarm')
plt.title('Survival (0=No, 1=Yes)')
plt.ylabel('Count')
plt.legend(title='Gender', labels=['Male', 'Female'])
plt.show()
```



 # Survival by Passenger Class

```
sns.countplot(x='Pclass', hue='Survived', data=data, palette='Set3')
plt.title('Survival by ticket class')
plt.show()
```



#Age Distribution by Survival

```
sns.histplot(data[data['Survived'] == 1]['Age'], kde=True, label='Survived',
color='green')
sns.histplot(data[data['Survived'] == 0]['Age'], kde=True, label='Not Survived',
color='red')
plt.legend()
plt.title("Age Distribution by survival")
plt.show()
```

#4. Data Cleaning

```
# Fill missing Age values with the median
data['Age'].fillna(data['Age'].median(), inplace=True)

# Fill missing Embarked values with the mode
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Drop the 'Cabin' column due to too many missing values
data.drop('Cabin', axis=1, inplace=True)

# Drop irrelevant features (Name, Ticket, and PassengerId)
data.drop(['Name', 'Ticket', 'PassengerId'], axis=1, inplace=True)
```

#5. Encoding Categorical Variables

```
encoder = LabelEncoder()
data['Sex'] = encoder.fit_transform(data['Sex'])
data['Embarked'] = encoder.fit_transform(data['Embarked'])
```

#6. Feature Engineering

Create a family Size feature

```
data['FamilySize'] = data['SibSp'] + data['Parch'] + 1
data.drop(['SibSp', 'Parch'], axis=1, inplace=True)
```

#7. Splitting the Data into Features and Target

Split the data into features (X) and target (y)

```
x = data.drop('Survived', axis=1)
y = data['Survived']
```

#8. Feature Scaling

Scaling the features

```
scaler = StandardScaler()
x[['Age', 'Fare']] = scaler.fit_transform(x[['Age', 'Fare']])
```

#9.Train-Test Split

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

#10.Building and Training the Models

Logistic Regression Model

```
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(x_train, y_train)
```

#11. Model Evaluation

```
models = {'Logistic Regression': log_reg, 'Random Forest': rf}
for name, model in models.items():
    y_pred = model.predict(x_test)
    print(f"\n{name} Metrics:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
```

```
    print("F1 Score:", f1_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

#12. Optimizing the Random Forest Model

Optimization: Tune Random Forest

```
tuned_rf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
tuned_rf.fit(x_train, y_train)
y_pred_tuned = tuned_rf.predict(x_test)
```

Evaluating Tuned Random Forest

```
print("\nTuned Random Forest Metrics:")
print("Accuracy:", accuracy_score(y_test, y_pred_tuned))
print("Precision:", precision_score(y_test, y_pred_tuned))
print("Recall:", recall_score(y_test, y_pred_tuned))
print("F1 Score:", f1_score(y_test, y_pred_tuned))
```

output:

Logistic Regression Metrics:

Accuracy: 0.8044692737430168

Precision: 0.782608695652174

Recall: 0.7297297297297297

F1 Score: 0.7552447552447552

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.86	0.84	105
1	0.78	0.73	0.76	74
accuracy			0.80	179
macro avg	0.80	0.79	0.80	179
weighted avg	0.80	0.80	0.80	179

Random Forest Metrics:

Accuracy: 0.8268156424581006

Precision: 0.7945205479452054

Recall: 0.7837837837837838

F1 Score: 0.7891156462585034

Classification Report:

	precision	recall	f1-score	support
0	0.85	0.86	0.85	105
1	0.79	0.78	0.79	74
accuracy			0.83	179
macro avg	0.82	0.82	0.82	179
weighted avg	0.83	0.83	0.83	179

Tuned Random Forest Metrics:

Accuracy: 0.8100558659217877

Precision: 0.8125

Recall: 0.7027027027027027

F1 Score: 0.7536231884057971

Conclusion:

The Titanic survival prediction case study demonstrates the effectiveness of machine learning models in analyzing survival patterns. Key factors influencing survival were **gender**, **passenger class**, and **family size**, with women and first-class passengers having higher survival rates. Logistic Regression and Random Forest models were applied, with Random Forest outperforming Logistic Regression after hyper parameter tuning. The study highlights the importance of data preprocessing, feature engineering, and model evaluation for building robust predictive models.