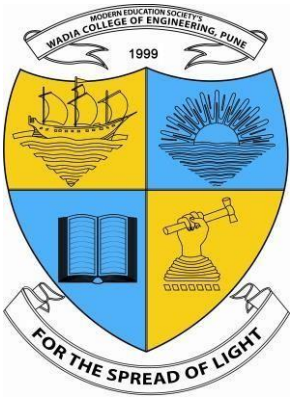
Savitribai Phule Pune University
Modern Education Society's Wadia College of Engineering, Pune

19, Bund Garden, V.K. Joag Path, Pune – 411001.

ACCREDITED BY NAAC
WITH "A++" GRADE

DEPARTMENT OF COMPUTER ENGINEERING



A REPORT ON

Machine Learning - Mini Project

" Design and Develop a Titanic Passenger Survival Model"

B.E (COMPUTER)

SUBMITTED BY

Megha (PRN: F21113029)
Samruddhi Kale (PRN: F21113030)
Mugdha Arvind (PRN: F21113049)

UNDER THE GUIDANCE OF

Dr. A. P. Kale

TITLE:

Design and Development of a Machine Learning Model for Predicting Titanic Survival

PURPOSE:

The Titanic Survival Prediction Model was developed as a mini project to assist in predicting the survival of passengers on the Titanic using machine learning techniques. This tool utilizes passenger data, including age, socio-economic class, and more, to train a predictive model that estimates survival probabilities. Although it serves as an educational prototype, it can be further refined for more complex analysis and predictions.

HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware Requirements:

- A computer with at least 4GB RAM.
- Disk space of at least 500MB.
- Supported Operating System: Linux, macOS, or Windows.

Software Requirements:

Python 3.x installed on the system, along with the following libraries:

- **pandas:** Used for data manipulation and analysis.
- **numpy**: Facilitates numerical calculations.
- **scikit-learn**: Provides tools for machine learning, including classification algorithms and model evaluation metrics.
- **matplotlib**: Used for data visualization, such as plotting graphs and charts.
- **seaborn**: A data visualization library based on matplotlib that provides a high-level interface for drawing attractive statistical graphics.
- **jupyter**: An open-source web application for creating and sharing documents that contain live code, equations, visualizations, and narrative text.
- These libraries can be installed via pip using the command pip install <library_name>.

WHY THIS TOOL IS NEEDED:

Predicting survival on the Titanic is not only an interesting historical question but also serves as an excellent introduction to machine learning. It demonstrates how to apply data science techniques to real-world problems. Automating the analysis of survival based on various passenger characteristics simplifies the exploration of how different factors affect survival rates.

WHAT THIS TOOL CAN DO:

- Load and preprocess the Titanic dataset.
- Train various machine learning models to predict survival based on passenger features.
- Evaluate model performance using metrics such as accuracy, precision, recall, and F1 score.
- Visualize important features that contribute to survival predictions.

USEFULNESS:

The tool provides insights into the factors that influenced survival on the Titanic, allowing users to understand which attributes were most significant. This can serve educational purposes for students and professionals alike, offering a practical example of applying machine learning to historical data.

FEATURES OF THE TOOL:

Data Loading and Preprocessing:

- Loads the Titanic dataset and handles missing values.
- Encodes categorical variables and normalizes numerical features for better model performance.

Model Training:

- Implements multiple machine learning algorithms (e.g., Logistic Regression, Random Forest) to predict survival.
- Uses cross-validation to ensure the robustness of the models.

Model Evaluation:

- Evaluates models using confusion matrices and classification reports to assess performance.
- Visualizes accuracy and performance metrics for each model.

Feature Importance Analysis:

- Analyzes and visualizes feature importances to understand which factors most significantly impact survival predictions.

Report Generation:

- Generates a report summarizing model performance and feature importance that can be exported in various formats (CSV, JSON).

OUTPUT -

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [3]:
```
test_df = pd.read_csv("./datasets/titanic_test.csv")
train_df = pd.read_csv("./datasets/titanic_train.csv")
train_df.head()
```

Out[3]:

	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

In [4]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 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
```

In [5]:
```
train_df.describe()
```

Out[5]:

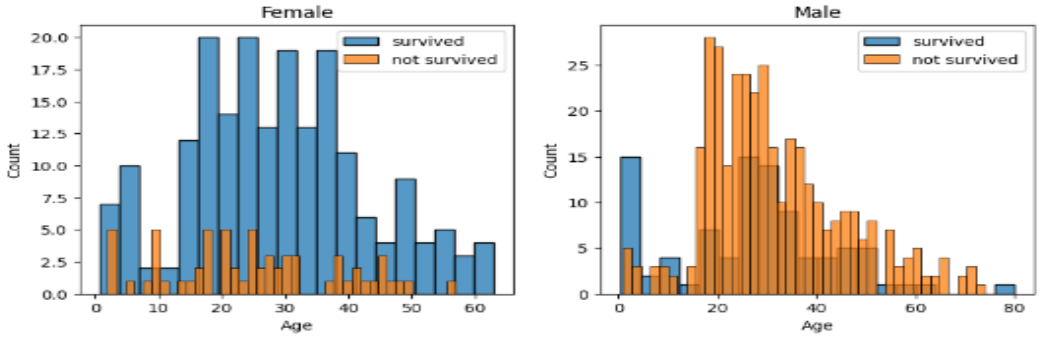
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	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

In [8]:
```
total = train_df.isnull().sum().sort_values(ascending=False)
percent_1 = train_df.isnull().sum()/train_df.isnull().count()*100
percent_2 = (round(percent_1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, percent_2], axis=1, keys=['Total', '%'])
missing_data.head(5)
```

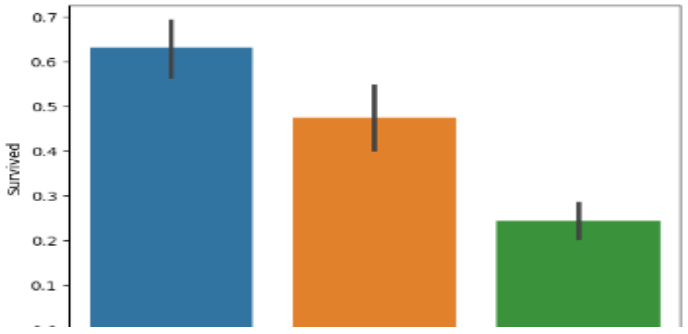
	Total	%
Cabin	687	77.1
Age	177	19.9
Embarked	2	0.2
PassengerId	0	0.0
Survived	0	0.0

In [14]:
```
survived = 'survived'
not_survived = 'not survived'
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10, 4))
women = train_df[train_df['Sex']=='female']
men = train_df[train_df['Sex']=='male']

ax = sns.histplot(women[women['Survived']==1].Age.dropna(), bins=20, label = survived, ax = axes[0], kde =False)
ax = sns.histplot(women[women['Survived']==0].Age.dropna(), bins=40, label = not_survived, ax = axes[0], kde =False)
ax.set_title('Female')
ax.legend()

ax = sns.histplot(men[men['Survived']==1].Age.dropna(), bins=20, label = survived, ax = axes[1], kde = False)
ax = sns.histplot(men[men['Survived']==0].Age.dropna(), bins=40, label = not_survived, ax = axes[1], kde = False)
ax.legend()
ax.set_title('Male')
```

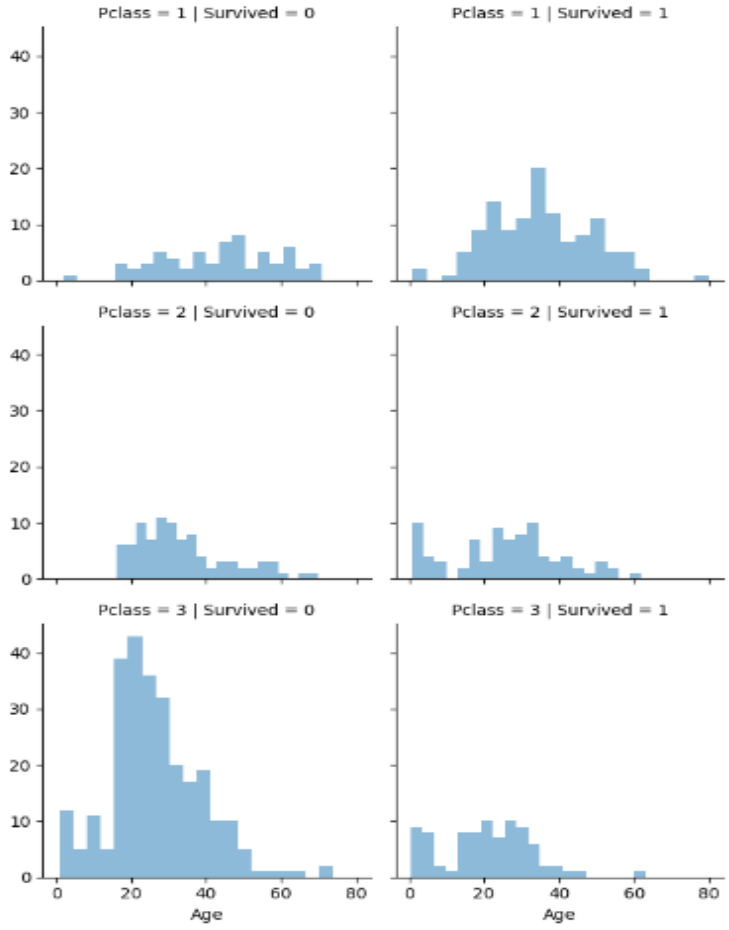
Out[14]: Text(0.5, 1.0, 'Male')



In [15]:
```
sns.barplot(x='Pclass', y='Survived', data=train_df)
plt.show()
```



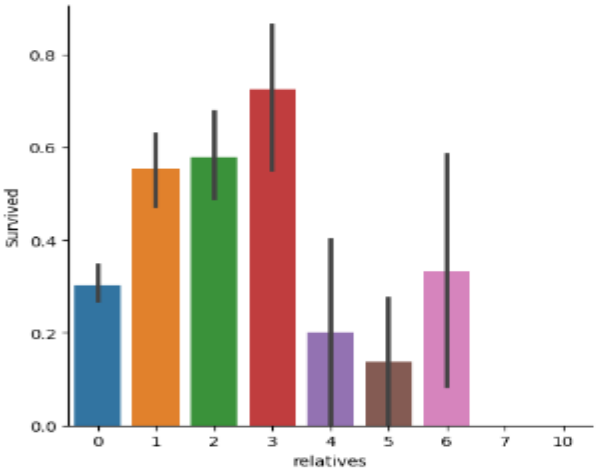
In [11]:
```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass')
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```



In [12]:
```
data = [train_df, test_df]
for dataset in data:
    dataset['relatives'] = dataset['SibSp'] + dataset['Parch']
    dataset.loc[dataset['relatives'] > 0, 'not_alone'] = 0
    dataset.loc[dataset['relatives'] == 0, 'not_alone'] = 1
    dataset['not_alone'] = dataset['not_alone'].astype(int)
train_df['not_alone'].value_counts()
```

Out[12]:
```
1    537
0    354
Name: not_alone, dtype: int64
```

In [178…
```
axes = sns.catplot(x='relatives', y='Survived', data=train_df, kind="bar")
```



In [179…
```
train_df = train_df.drop(['PassengerId'], axis=1)
```

In [180…
```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_df, test_df]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).group())
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)
train_df = train_df.drop(['Cabin'], axis=1)
test_df = test_df.drop(['Cabin'], axis=1)
```

In [181…
```
data = [train_df, test_df]

for dataset in data:
    mean = train_df["Age"].mean()
    std = test_df["Age"].std()
    is_null = dataset["Age"].isnull().sum()
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)
    age_slice = dataset["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    dataset["Age"] = age_slice
    dataset["Age"] = train_df["Age"].astype(int)
train_df["Age"].isnull().sum()
```

Out[181…
```
0
```

In [182…
```
train_df['Embarked'].describe()
```

Out[182…
```
count     889
unique      3
top         S
freq      644
Name: Embarked, dtype: object
```

In [183…
```
common_value = 'S'
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].fillna(common_value)
```

In [184…
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Name        891 non-null    object
 3   Sex         891 non-null    object
 4   Age         891 non-null    int32
 5   SibSp       891 non-null    int64
 6   Parch       891 non-null    int64
 7   Ticket      891 non-null    object
 8   Fare        891 non-null    float64
 9   Embarked    891 non-null    object
 10  relatives   891 non-null    int64
 11  not_alone   891 non-null    int32
 12  Deck        891 non-null    int32
dtypes: float64(1), int32(3), int64(5), object(4)
memory usage: 80.2+ KB
```

In [185…
```
data = [train_df, test_df]

for dataset in data:
    dataset['Fare'] = dataset['Fare'].fillna(0)
    dataset['Fare'] = dataset['Fare'].astype(int)
```

In [186…
```
data = [train_df, test_df]
titles = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

for dataset in data:
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonk
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
    dataset['Title'] = dataset['Title'].map(titles)
    dataset['Title'] = dataset['Title'].fillna(0)
train_df = train_df.drop(['Name'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
```

In [187…
```
genders = {"male": 0, "female": 1}
data = [train_df, test_df]

for dataset in data:
    dataset['Sex'] = dataset['Sex'].map(genders)
```

In [188…
```
train_df['Ticket'].describe()
```

Out[188…
```
count        891
unique       681
top       347082
freq           7
Name: Ticket, dtype: object
```

In [189…
```
train_df = train_df.drop(['Ticket'], axis=1)
test_df = test_df.drop(['Ticket'], axis=1)
```

In [190…
```
ports = {"S": 0, "C": 1, "Q": 2}
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

In [198…

```
ports = {"S": 0, "C": 1, "Q": 2}
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

In [191…

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age'] = dataset['Age'].astype(int)
    dataset.loc[ dataset['Age'] <= 11, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 11) & (dataset['Age'] <= 18), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 18) & (dataset['Age'] <= 22), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 22) & (dataset['Age'] <= 27), 'Age'] = 3
    dataset.loc[(dataset['Age'] > 27) & (dataset['Age'] <= 33), 'Age'] = 4
    dataset.loc[(dataset['Age'] > 33) & (dataset['Age'] <= 40), 'Age'] = 5
    dataset.loc[(dataset['Age'] > 40) & (dataset['Age'] <= 66), 'Age'] = 6
    dataset.loc[ dataset['Age'] > 66, 'Age'] = 6
train_df['Age'].value_counts()
```

Out[191…

```
6    166
4    160
5    150
3    135
2    117
1     95
0     68
Name: Age, dtype: int64
```

In [192…

```
data = [train_df, test_df]

for dataset in data:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[(dataset['Fare'] > 31) & (dataset['Fare'] <= 99), 'Fare']   = 3
    dataset.loc[(dataset['Fare'] > 99) & (dataset['Fare'] <= 250), 'Fare']   = 4
    dataset.loc[ dataset['Fare'] > 250, 'Fare'] = 5
    dataset['Fare'] = dataset['Fare'].astype(int)
```

In [193…

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age_Class']= dataset['Age']* dataset['Pclass']
```

In [194…

```
for dataset in data:
    dataset['Fare_Per_Person'] = dataset['Fare']/(dataset['relatives']+1)
    dataset['Fare_Per_Person'] = dataset['Fare_Per_Person'].astype(int)
train_df.head()
```

Out[194…

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not alone	Deck	Title	Age Class	Fare Per Person
0	0	3	0	2	1	0	0	0	1	0	8	1	6	0
1	1	1	1	5	1	0	3	1	1	0	3	3	5	1
2	1	3	1	3	0	0	0	0	0	1	8	2	9	0
3	1	1	1	5	1	0	3	0	1	0	3	3	5	1
4	0	3	0	5	0	0	1	0	0	1	8	1	15	1

In [195…

```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.drop("PassengerId", axis=1).copy()
```

In [196…

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
```

In [197…

```
logreg = LogisticRegression(max_iter=5000)
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)

acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
```

In [198…

```
linear_svc = SVC()
linear_svc.fit(X_train, Y_train)

Y_pred = linear_svc.predict(X_test)

acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
```

In [199…

```
results = pd.DataFrame({
    'Model': ['Support Vector Machines', 'Logistic Regression', 'Random Forest'],
    'Score': [acc_linear_svc, acc_log, acc_random_forest]
})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df
```

Out[199…

Score	Model
92.26	Random Forest
81.71	Support Vector Machines
81.71	Logistic Regression