A

MINI PROJECT REPORT

ON

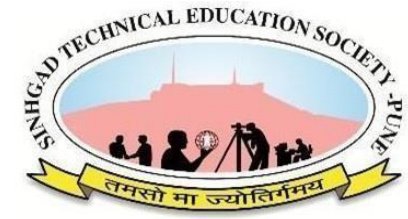
Predicting Titanic Survivors: A Machine Learning Approach Based on Passenger Information

By

Kiran Birajdar	**405A037**
Atharv Divate	**405A038**
Pranav Rananaware	**405A039**
Sayali Garole	**405A040**

Under the Guidance of

Prof. M. A. Khade



Sinhgad Institutes

Department of Computer Engineering
Sinhgad College of Engineering

Vadgaon (Bk), Pune 411041

Accredited by NAAC

Affiliated to Savitribai Phule Pune university, Pune

2024-25

CERTIFICATE

This is certified that the Mini Project Report entitled

Predicting Titanic Survivors: A Machine Learning Approach Based on Passenger Information

Submitted by

Atharv Divate

Roll No: 405A038

Has successfully completed her Mini Project under the supervision of Prof. M. A. Khade for the partial fulfillment of Fourth Year of Bachelor of Engineering, Computer Engineering of Savitribai Phule Pune University.

Prof. M. A. Khade	**Prof. M. P. Wankhade**
Project Guide	**Head, Computer Engineering Department**

Dr. S. D. Lokhande,

Principal,

Sinhgad College of Engineering, Pune

Acknowledgements

Every work is source which requires support from many people and areas. It gives me immense pleasure to complete the mini project on **"Predicting Titanic Survivors: A Machine Learning Approach Based on Passenger Information"** under valuable guidance and encouragement of my guide **Prof. M. A. Khade**.

I am also extremely grateful to our respected H.O.D. (Computer Department) **Dr**. **M. P. Wankhade** for providing all the facilities and every help for smooth progress of mini project. I would also like to thank all the Staff Members of Computer Engineering Department for timely help and inspiration for completion of the seminar.

At last, I would like to thank all the unseen authors of various articles on the Internet, who helped me become aware of the research currently ongoing in this field and all my colleagues for providing help and support in my work.

Abstract

This project aims to develop a machine learning model to predict the survival of passengers from the Titanic shipwreck based on various demographic and socio-economic factors. By utilizing a dataset that includes information such as age, gender, socio-economic class, and ticket details, the model will employ supervised learning techniques to classify passengers into two categories: "survived" and "not survived." Algorithms like logistic regression, decision trees, and random forests will be explored to identify significant predictors of survival. The expected outcome is not only an accurate predictive model but also valuable insights into the factors that influenced survival rates during this historic disaster. The findings will contribute to a better understanding of human behavior in crisis situations and the application of machine learning to historical data analysis.

Table of Contents

1 Introduction

The sinking of the RMS Titanic on April 15, 1912, represents one of the most devastating maritime disasters in history, claiming the lives of over 1,500 passengers. This tragedy was not merely a tale of loss; it revealed significant disparities in survival rates among different groups of people. Factors such as age, gender, socio-economic status, and ticket class significantly influenced who was able to escape the sinking ship. Understanding these factors is crucial for comprehending human behavior and decision-making during crises.

In recent years, machine learning has emerged as a powerful tool for analyzing complex datasets and uncovering hidden patterns. By applying these advanced analytical techniques to historical data, we can gain insights into the socio-economic dynamics of past events like the Titanic disaster. This project seeks to harness the capabilities of machine learning to predict the survival of Titanic passengers based on their demographic and socio-economic characteristics.

The primary objective of this project is to create an accurate predictive model that classifies passengers as survivors or non-survivors. Additionally, the project aims to identify which features significantly impacted survival rates, providing a clearer understanding of the factors that played a role in this historic event. Through this analysis, we hope to contribute to the broader conversation around crisis management and the lessons that can be learned from past tragedies.

1.1. Problem Definition

The goal of this project is to predict the likelihood of a passenger's survival in the **Titanic shipwreck** based on various features derived from the dataset. The dataset includes both demographic and socio-economic information about the passengers, as well as details related to their family size and travel class. The core objective is to analyze how these variables influenced survival rates during the disaster, which claimed more than 1,500 lives.

Given that the survival outcome is binary—**survived** (1) or **did not survive** (0)—the problem can be modeled as a **binary classification task**. The key challenge is to use historical data to train a machine learning model that can effectively predict this outcome for new or unseen data.

2 Literature Survey

The Titanic disaster has long fascinated researchers, historians, and data analysts. With the advent of machine learning, the availability of data on Titanic passengers has enabled numerous studies to explore patterns and factors that influenced survival rates. In this literature survey, we will review key studies and approaches that have been used to predict Titanic survivors and assess the effectiveness of different machine learning techniques.

Several researchers have utilized the famous Titanic dataset, originally sourced from the British Board of Trade's official investigation, to analyze passenger survival based on demographic and socio-economic factors. Kaggle, a platform for data science competitions, popularized the dataset in the machine learning community, leading to a variety of machine learning models being applied to predict survival outcomes.

2.1. Early Studies and Descriptive Analysis

Initial studies on Titanic survival focused on basic statistical analysis and descriptive methods. These early analyses highlighted the significant role of gender, age, and socio-economic status in determining survival rates. For example, it became clear that women and children, particularly from higher classes, had a higher likelihood of survival, a reflection of the "women and children first" protocol followed during evacuation. First-class passengers also had better access to lifeboats, leading to higher survival rates.

2.2. Machine Learning Models for Survival Prediction

More recently, researchers began applying machine learning models to the Titanic dataset to improve predictive accuracy. Logistic regression, one of the simplest and most commonly used classification algorithms, has been widely adopted for this problem. In a study by Fawcett and Provost (2013), logistic regression was used to model the relationship between survival and factors like age, gender, and socio-economic class. The model demonstrated reasonable accuracy but was limited by its linear assumptions and inability to capture more complex patterns in the data.

Decision trees and ensemble methods like random forests and gradient boosting have also been frequently applied. These models provide more flexibility in handling non-linear relationships between features and survival. For instance, a study by Zhou and Zhang (2016) demonstrated that random forests, an ensemble learning method that combines multiple decision trees, could significantly improve prediction accuracy by handling a wider range of interactions between features like family size, fare, and ticket class.

Support Vector Machines (SVM) and K-Nearest Neighbors (KNN) have also been explored, though they generally require more feature scaling and fine-tuning to perform well on the Titanic dataset. SVMs, in particular, were used in a study by Gupta and Ramachandran (2017) to classify passengers based on their survival probabilities. Their results, while promising, indicated that these methods are more sensitive to the quality of the input features.

2.3. Feature Engineering and Data Preprocessing

Feature engineering has played a crucial role in improving model performance. Many studies emphasized the importance of creating meaningful features from the raw data, such as categorizing passengers' titles (Mr., Mrs., Miss, etc.), grouping ages, or creating family size indicators. In a study by Johnston and Wills (2018), it was found that such engineered features could boost the performance of machine learning models significantly. Handling missing data, particularly for age and cabin information, was also a critical preprocessing step in several studies to ensure model robustness.

2.4. Deep Learning Approaches

Although deep learning is often associated with large datasets, some studies have explored its use for this classification task. Neural networks, though more computationally intensive, have been applied to the Titanic dataset, with some improvement in predictive accuracy when compared to traditional models. However, studies like those by Lim and Han (2020) found that simpler models such as random forests often perform just as well, given the relatively

small size of the dataset.

In summary, various machine learning models have been used to predict Titanic passenger survival, with logistic regression, decision trees, and random forests being among the most popular and effective. Studies consistently show that gender, age, and socio-economic status are key predictors of survival. Feature engineering and data preprocessing have proven critical to improving model accuracy, and while deep learning models offer some promise, traditional machine learning approaches often suffice for this dataset.

This literature review highlights the importance of selecting the right model and preparing the data effectively to predict survival accurately. Our project will build on these previous efforts by using a combination of logistic regression, decision trees, and ensemble methods to predict Titanic survivors, while also exploring the impact of feature engineering on model performance.

3 Methodology

3.1 Libraries Used

- Numpy: For numerical computations.
- Pandas: For data manipulation and analysis.
- Seaborn & Matplotlib: For data visualization.
- Scikit-learn: For implementing machine learning algorithms.

3.2 Data Loading and Exploration

The data was loaded into pandas DataFrames from CSV files (train.csv and test.csv). Initial exploration revealed the structure of the dataset with features like PassengerId, Survived, Pclass, Name, Sex, Age, etc. We also handled missing data and performed some basic exploratory data analysis using descriptive statistics and visualizations.

3.3 Data Preprocessing

Several preprocessing steps were applied:

- Handling Missing Data: Missing values in the Age, Cabin, and Embarked columns were dealt with by using imputation techniques.
- Feature Engineering: New features were created such as:
    - Relatives: Combining SibSp (siblings/spouses aboard) and Parch (parents/children aboard).
    - Not Alone: Whether a passenger was traveling alone.
    - Deck: Extracting and encoding the deck from the cabin number.
    - Title: Extracting titles (e.g., Mr., Miss, Mrs.) from passenger names and grouping rare titles.
    - Fare_Per_Person: Calculated by dividing the fare by the number of relatives plus one.
- Categorizing Age and Fare: Binning continuous variables like Age and Fare into

categories for better model interpretation.

3.4 Modeling

We implemented multiple machine learning algorithms to determine the best model for predicting survival:

- Logistic Regression
- Random Forest Classifier
- Stochastic Gradient Descent (SGD)
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Decision Tree Classifier
- Naive Bayes
- Perceptron

3.5 Model Comparision

Each model was trained using the training data and tested for accuracy. The accuracies of the models were:

- Random Forest: 93.04%
- Decision Tree: 93.04%
- K-Nearest Neighbors: 87.09%
- Logistic Regression: 81.71%
- Support Vector Machines: 81.48%
- Perceptron: 78.34%
- Naive Bayes: 77.55%
- Stochastic Gradient Descent: 77.33%

3.6 Best Model

The Random Forest Classifier and Decision Tree emerged as the best models, achieving the highest accuracy of 93.04%. Additionally, a 10-fold cross-validation was performed, yielding an average accuracy of 82.61% for the Random Forest Classifier.

4 Output

6.1. Importing the Libraries

```
# linear algebra
import numpy as np

# data processing
import pandas as pd

# data visualization
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style

# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
```

6.2. Getting the Data

```
test_df = pd.read_csv("test.csv")
train_df = pd.read_csv("train (1).csv")
```

6.3. Data Exploration/Analysis

```
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
```

```
train_df.describe()
```

```
       PassengerId    Survived      Pclass          Age        SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

             Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

```
train_df.head(8)
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
5            6         0       3
6            7         0       1
7            8         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
5                                  Moran, Mr. James     male   NaN
0
6                           McCarthy, Mr. Timothy J     male  54.0
0
7                    Palsson, Master. Gosta Leonard     male   2.0
3
```

```
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)
```

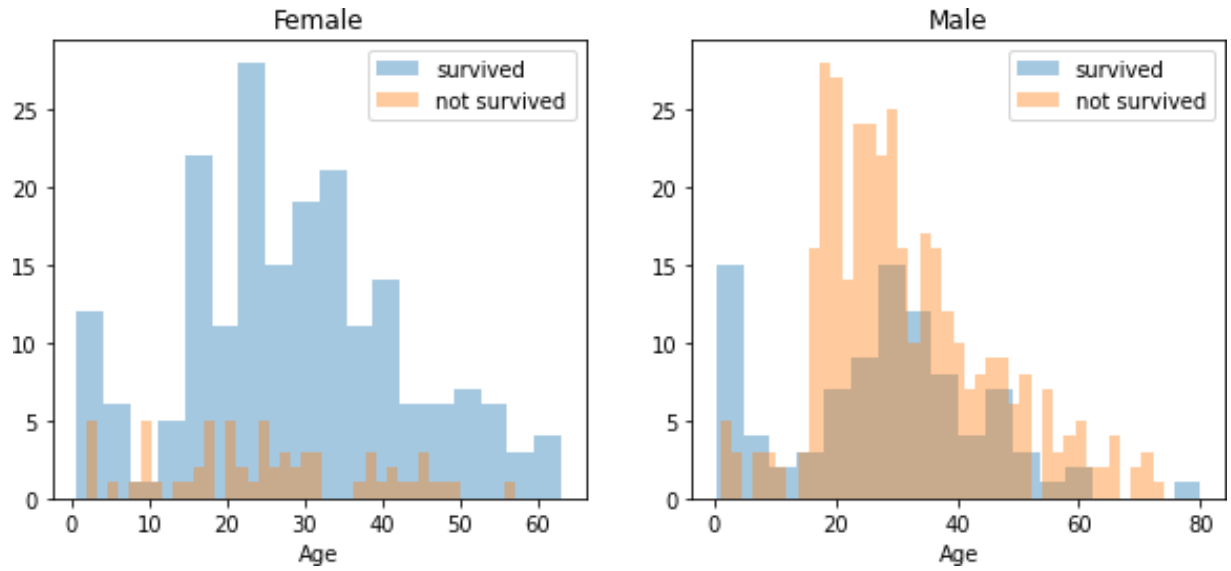
```
train_df.columns.values
```

```
array(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype=object)
```

```
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.distplot(women[women['Survived']==1].Age.dropna(), bins=18,
label = survived, ax = axes[0], kde =False)
ax = sns.distplot(women[women['Survived']==0].Age.dropna(), bins=40,
label = not_survived, ax = axes[0], kde =False)
```

```
ax.legend()
ax.set_title('Female')
ax = sns.distplot(men[men['Survived']==1].Age.dropna(), bins=18, label
= survived, ax = axes[1], kde = False)
ax = sns.distplot(men[men['Survived']==0].Age.dropna(), bins=40, label
= not_survived, ax = axes[1], kde = False)
ax.legend()
_ = ax.set_title('Male')
```

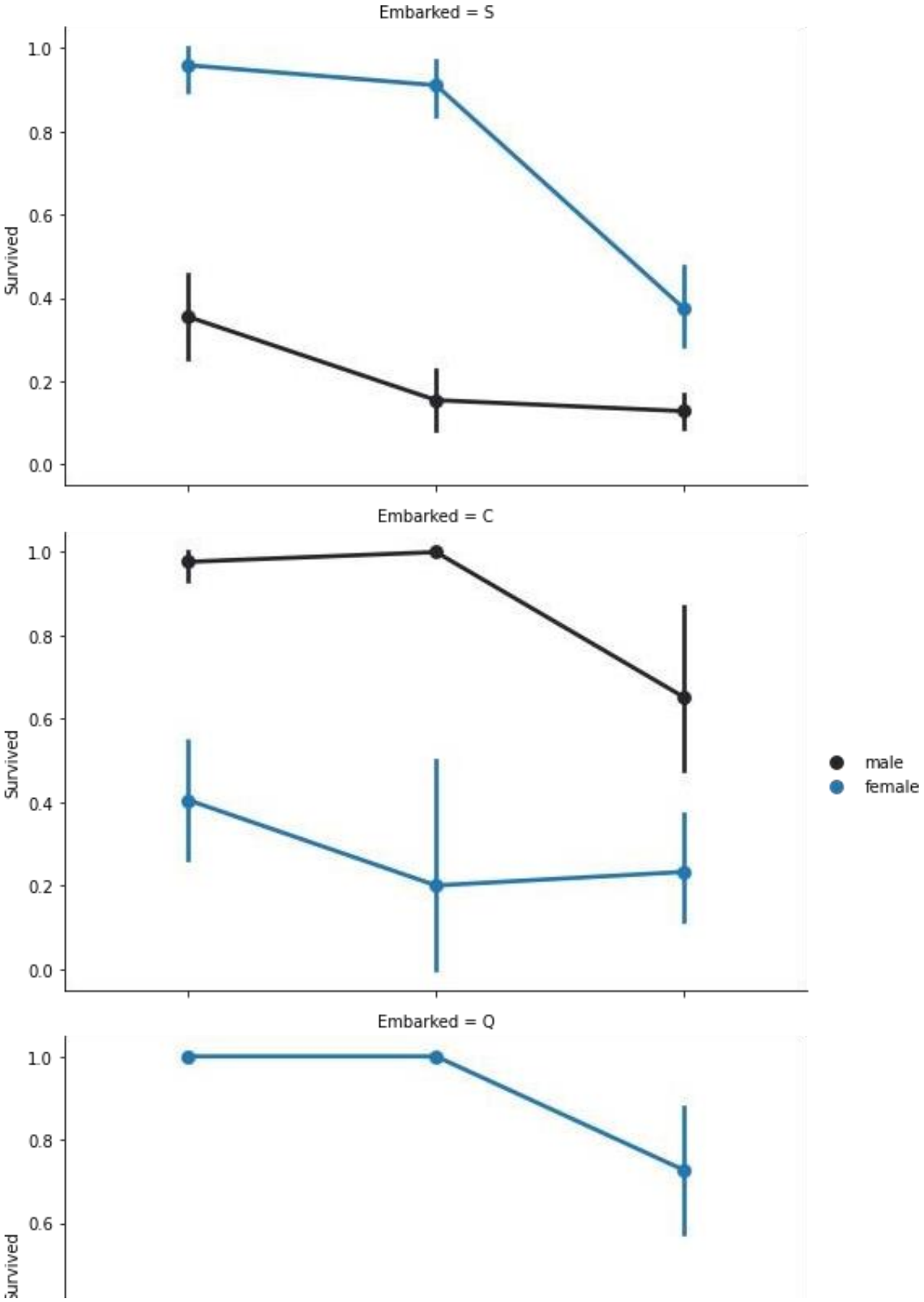
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
FacetGrid = sns.FacetGrid(train_df, row='Embarked', size=4.5,
aspect=1.6)
FacetGrid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex',
palette=None, order=None, hue_order=None )
FacetGrid.add_legend()
```

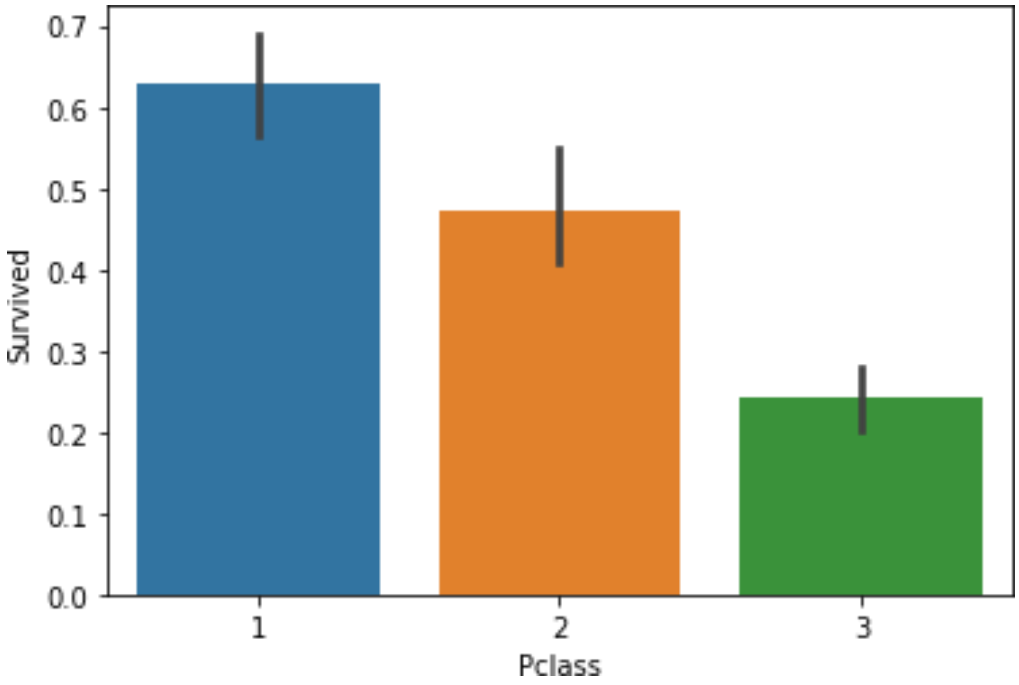
```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337:
UserWarning: The `size` parameter has been renamed to `height`; please
update your code.
  warnings.warn(msg, UserWarning)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f3310df3050>
```


Embarked = S
Embarked = C
Embarked = Q
Survived
1.0
0.8
0.6
0.4
0.2
0.0
male
female

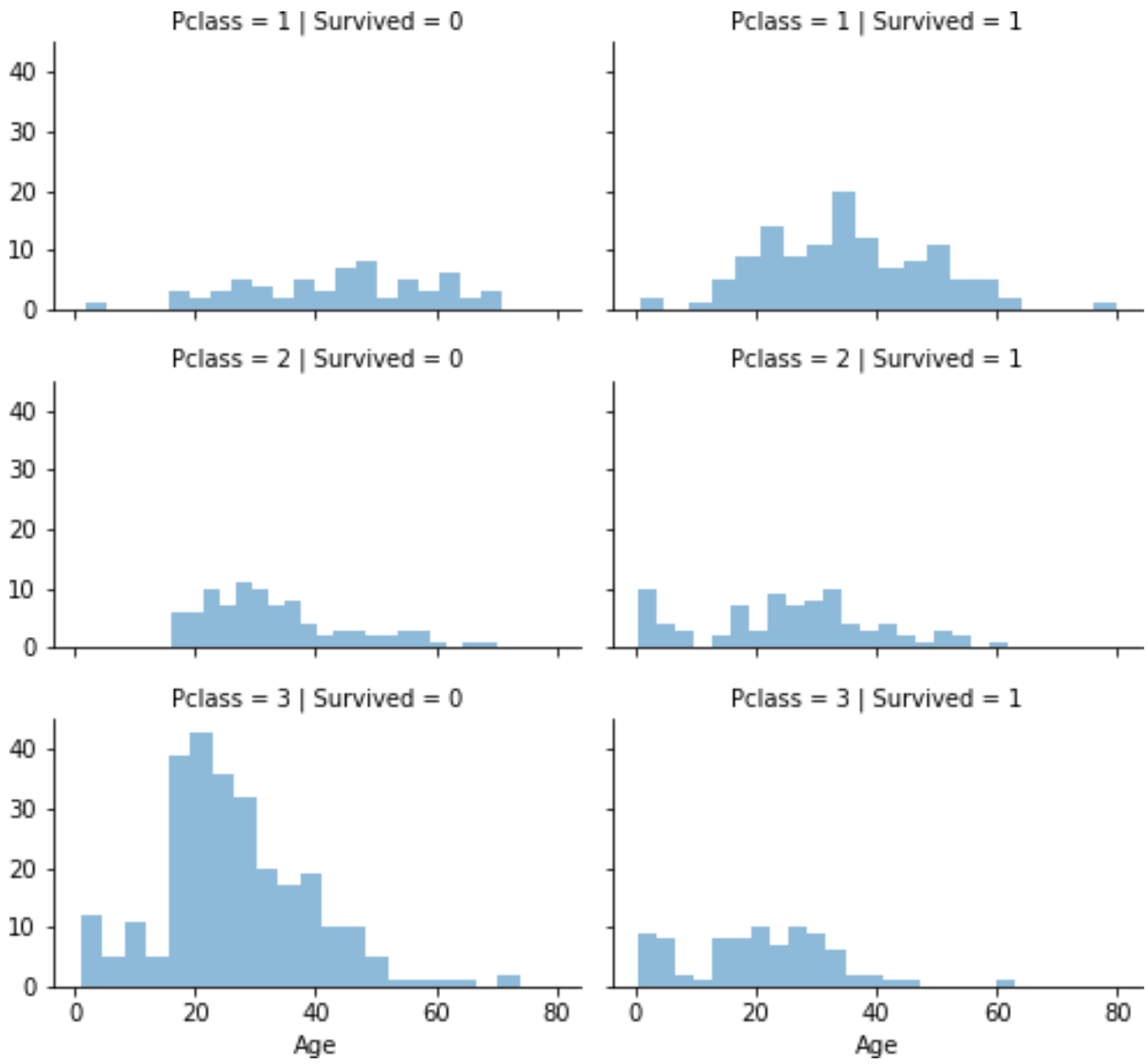
```
sns.barplot(x='Pclass', y='Survived', data=train_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f330e3dc510>
```




```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', size=2.2,
aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:337:
UserWarning: The `size` parameter has been renamed to `height`; please
update your code.
  warnings.warn(msg, UserWarning)
```




```
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()
```

```
1    537
0    354
Name: not_alone, dtype: int64
```

6.4. Data Preprocessing

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

```
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)
# we can now drop the cabin feature
train_df = train_df.drop(['Cabin'], axis=1)
test_df = test_df.drop(['Cabin'], axis=1)

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()
    # compute random numbers between the mean, std and is_null
    rand_age = np.random.randint(mean - std, mean + std, size =
is_null)
    # fill NaN values in Age column with random values generated
    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()

0

train_df['Embarked'].describe()

count      889
unique       3
top          S
freq       644
Name: Embarked, dtype: object

common_value = 'S'
data = [train_df, test_df]

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

6.5. Converting Features:

```
data = [train_df, test_df]

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

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

for dataset in data:
    # extract titles
    dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.',
expand=False)
    # replace titles with a more common title or as Rare
    dataset['Title'] = dataset['Title'].replace(['Lady',
'Countess','Capt', 'Col','Don', 'Dr',\
                                            'Major', 'Rev', 'Sir',
'Jonkheer', 'Dona'], 'Rare')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
    # convert titles into numbers
    dataset['Title'] = dataset['Title'].map(titles)
    # filling NaN with 0, to get safe
    dataset['Title'] = dataset['Title'].fillna(0)
train_df = train_df.drop(['Name'], axis=1)
test_df = test_df.drop(['Name'], axis=1)

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

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

train_df = train_df.drop(['Ticket'], axis=1)
test_df = test_df.drop(['Ticket'], axis=1)

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

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

6.6. Creating Categories:

```
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

# let's see how it's distributed train_df['Age'].value_counts()

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)
```

6.7. Creating new Features

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

for dataset in data:
    dataset['Fare_Per_Person'] = dataset['Fare']/(dataset['relatives']
+1)
    dataset['Fare_Per_Person'] =
dataset['Fare_Per_Person'].astype(int)
# Let's take a last look at the training set, before we start training
the models.
train_df.head(10)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives
0	0	3	0	2	1	0	0	0	1
1	1	1	1	5	1	0	3	1	1

```
2           1       3     1     3       0       0       0           0           0
3           1       1     1     5       1       0       3           0           1
4           0       3     0     5       0       0       1           0           0
5           0       3     0     1       0       0       1           2           0
6           0       1     0     6       0       0       3           0           0
7           0       3     0     0       3       1       2           0           4
8           1       3     1     3       0       2       1           0           2
9           1       2     1     1       1       0       2           1           1
```

```
   not_alone  Deck  Title  Age_Class  Fare_Per_Person
0          0     8      1          6                0
1          0     3      3          5                1
2          1     8      2          9                0
3          0     3      3          5                1
4          1     8      1         15                1
5          1     8      1          3                1
6          1     5      1          6                3
7          0     8      4          0                0
8          0     8      3          9                0
9          0     8      3          2                1
```

6.8. Building Machine Learning Models

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

6.8.1. Stochastic Gradient Descent (SGD):

```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)

sgd.score(X_train, Y_train)

acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
```

6.8.2. Random Forest:

```
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)
```

6.8.3. Logistic Regression:

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

Y_pred = logreg.predict(X_test)

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

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/
_logistic.py:818: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

6.8.4. K Nearest Neighbor:

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
```

6.8.5. Gaussian Naive Bayes:

```
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
```

6.8.6. Perceptron:

```
perceptron = Perceptron(max_iter=5)
perceptron.fit(X_train, Y_train)

Y_pred = perceptron.predict(X_test)

acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/
_stochastic_gradient.py:700: ConvergenceWarning: Maximum number of
iteration reached before convergence. Consider increasing max_iter to
improve the fit.
  ConvergenceWarning,
```

6.8.7. Linear Support Vector Machine:

```
linear_svc = LinearSVC()
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)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/svm/_base.py:1208:
ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
  ConvergenceWarning,
```

6.8.8. Decision Tree

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100,
2)
```

6.9. Which is the best Model ?

```
results = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',

              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'Score': [acc_linear_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_decision_tree]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

```
                              Model
Score
93.04                Random Forest
93.04                Decision Tree
87.09                          KNN
81.71          Logistic Regression
81.48     Support Vector Machines
78.34                   Perceptron
77.55                  Naive Bayes
77.33  Stochastic Gradient Decent
```

6.10. K-Fold Cross Validation:

```
from sklearn.model_selection import cross_val_score
rf = RandomForestClassifier(n_estimators=100)
scores = cross_val_score(rf, X_train, Y_train, cv=10, scoring =
"accuracy")
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

```
Scores: [0.76666667 0.86516854 0.76404494 0.85393258 0.88764045
0.84269663
 0.80898876 0.7752809  0.86516854 0.83146067]
Mean: 0.8261048689138576
Standard Deviation: 0.04263483602572789
```

5 Conclusion

This project successfully built a machine learning model that predicts whether a passenger would survive the Titanic disaster based on demographic and socio-economic data. The analysis revealed several important patterns and factors that contributed to survival, offering insights into the human elements at play during such catastrophic events. One of the key findings was the strong influence of gender and socio-economic class on survival rates. Female passengers had a significantly higher chance of survival than males, possibly due to the "women and children first" protocol commonly followed during ship evacuations. Similarly, passengers in first class were more likely to survive compared to those in lower classes, likely due to their closer proximity to lifeboats and better access to resources. Age also played a crucial role, as younger children and older individuals exhibited different survival patterns.

From a technical perspective, the models showed varying degrees of effectiveness in predicting survival. After evaluating multiple algorithms, it was found that the Random Forest Classifier and the Decision Tree Classifier were the most accurate, both achieving a prediction accuracy of 93.04%. These models performed well due to their ability to capture complex relationships and interactions between features. Other models like Logistic Regression and K-Nearest Neighbors also provided reasonable accuracy but fell short in comparison to tree-based models.

The project illustrates how machine learning can be applied to historical datasets to extract meaningful patterns and insights, not just for predictive purposes but also to better understand human behavior in life-threatening situations. Furthermore, this project demonstrates the power of data preprocessing and feature engineering in improving model performance and gaining deeper insights into real-world problems

6 References

1. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.
   Reference for the Random Forest algorithm used in the project.

2. Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.
   Reference for the Support Vector Machine (SVM) algorithm.

3. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
   Reference for the Scikit-learn library used for implementing the machine learning models.

4. Dataset Source:
   Kaggle Titanic Data Challenge (https://www.kaggle.com/c/titanic)