**Titanic Dataset Analysis Report**

**1. Introduction**

The Titanic dataset is a well-known dataset used for machine learning and data analysis. It contains details about passengers who were on board the Titanic, including information such as age, gender, ticket class, and whether they survived the disaster. The objective of this analysis is to explore the dataset, clean the data, and build a predictive model to determine the likelihood of survival based on passenger attributes.

**2. Data Loading and Exploration**

**2.1 Importing Necessary Libraries**

The first step involves importing essential libraries for data handling and analysis:

import numpy as np

import pandas as pd

* numpy: Used for numerical operations.
* pandas: Used for data manipulation and analysis.

**2.2 Loading the Dataset**

The Titanic dataset is loaded into Pandas dataframes:

train\_data = pd.read\_csv(r'C:\Users\Sami\Downloads\train.csv')

test\_data = pd.read\_csv(r'C:\Users\Sami\Downloads\test.csv')

* train.csv: Contains passenger details along with survival outcomes (used for model training).
* test.csv: Contains passenger details without survival outcomes (used for prediction).

**2.3 Checking Data Structure**

To understand the dataset structure:

train\_data.info()

This provides:

* Number of non-null values in each column.
* Data types (integer, float, object).
* Identifies missing values.

**2.4 Previewing the Dataset**

train\_data.head()

This displays the first few rows of the dataset to get an initial understanding of the data.

**3. Data Preprocessing**

**3.1 Dropping Unnecessary Columns**

Some columns may not contribute to survival prediction and can be removed:

train\_data = train\_data.drop(['PassengerId', 'Name'], axis=1)

* PassengerId: Only an identifier, not useful for prediction.
* Name: Names do not provide any numerical value for modeling.

**3.2 Handling Missing Values**

Missing values must be handled to avoid errors during model training.

* Common strategies:
  + Fill missing Age values with the median age.
  + Fill missing Embarked values with the most frequent port.
  + Drop columns with excessive missing data.

Example:

train\_data['Age'].fillna(train\_data['Age'].median(), inplace=True)

train\_data['Embarked'].fillna(train\_data['Embarked'].mode()[0], inplace=True)

**4. Feature Engineering**

Feature engineering involves converting categorical data into numerical values and creating new meaningful features.

**4.1 Encoding Categorical Variables**

Categorical variables such as Sex and Embarked need to be converted into numerical format:

train\_data['Sex'] = train\_data['Sex'].map({'male': 0, 'female': 1})

train\_data = pd.get\_dummies(train\_data, columns=['Embarked'])

* Sex: Converted into 0 (male) and 1 (female).
* Embarked: One-hot encoded into separate columns.

**4.2 Creating New Features**

New features can be created to enhance model performance:

train\_data['FamilySize'] = train\_data['SibSp'] + train\_data['Parch'] + 1

* FamilySize: Total number of family members onboard.

**5. Model Training**

**5.1 Splitting Data**

The dataset is split into training and validation sets:

from sklearn.model\_selection import train\_test\_split

X = train\_data.drop(['Survived'], axis=1)

y = train\_data['Survived']

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

* X: Independent features.
* y: Target variable (Survived).
* train\_test\_split: Splits the data (80% training, 20% validation).

**5.2 Training a Machine Learning Model**

A classification model is used to predict survival:

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=100, random\_state=0)

model.fit(X\_train, y\_train)

* RandomForestClassifier: A robust ensemble model using decision trees.
* n\_estimators=100: Number of trees in the forest.
* model.fit(): Trains the model on training data.

**5.3 Evaluating Model Performance**

from sklearn.metrics import accuracy\_score

predictions = model.predict(X\_val)

accuracy = accuracy\_score(y\_val, predictions)

print(f'Accuracy: {accuracy}')

* Compares model predictions with actual survival values.
* Accuracy is used as a performance metric.

**6. Making Predictions on Test Data**

Once the model is trained, predictions are made on the test dataset:

test\_data['Sex'] = test\_data['Sex'].map({'male': 0, 'female': 1})

test\_data = pd.get\_dummies(test\_data, columns=['Embarked'])

test\_preds = model.predict(test\_data)

The results are stored in a CSV file:

output = pd.DataFrame({'PassengerId': test\_data['PassengerId'], 'Survived': test\_preds})

output.to\_csv('submission.csv', index=False)

This file can be submitted for evaluation.

**7. Conclusion**

* The Titanic dataset was analyzed, cleaned, and preprocessed.
* A machine learning model (Random Forest Classifier) was trained to predict survival.
* The model was evaluated and used to make predictions.
* Further improvements could be made using feature selection and hyperparameter tuning.

This analysis provides a structured approach to predictive modeling using real-world data.