A COMPREHENSIVE ANALYSIS WITH TITANIC DATASET

EDA, Visualization, Classification &Machine Learning

Masters in Applied Statistics and Data Science under Weekend Program (WM-ASDS)

PROJECT REPORT

SESSION:
Spring-2023
10th Batch: Section-B

SUBMITTED BY:
MD. MASHIUR RAHMAN
Student ID: 20231088

Department of Statistics
Jahangirnagar University, Savar, Dhaka

August 2023

INTRODUCTION

The sinking of the Titanic is one of the most infamous shipwrecks in history. Titanic, launched on May 31, 1911, and set sail on its maiden voyage from Southampton on April 10, 1912, with 2,240 passengers and crew on board. On April 15, 1912, after striking an iceberg, Titanic broke apart and sank to the bottom of the ocean, taking with it the lives of more than 1,500 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. Now we are going to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

DATASET INFORMATION

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use feature engineering to create new features.

The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

We also include gender_submission.csv, a set of predictions that assume all and only female passengers survive, as an example of what a submission file should look like.

Data Source: *https://www.kaggle.com/c/titanic/data*

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Variable Notes

- **pclass** = A proxy for socio-economic status (SES) 1st = Upper 2nd = Middle 3rd = Lower
- **age** = Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5
- **sibsp** = The dataset defines family relations in this way...
- **Sibling** = brother, sister, stepbrother, stepsister
- **Spouse** = husband, wife (mistresses and fiancés were ignored)
- **parch**: The dataset defines family relations in this way...
- **Parent** = mother, father
- **Child** = daughter, son, stepdaughter, stepson
- Some children travelled only with a nanny, therefore **parch=0** for them.
- The output class is **survival**, where we have to predict 0 (No) or 1 (Yes).

Libraries

- pandas
- matplotlib
- seaborn
- scikit-learn

Feature Variables:

- pclass
- sex
- Age
- sibsp
- parch
- ticket
- fare
- cabin
- embarked

Target Variable:

- survival

DESCRIBE DATASET

Before we begin, we require the following libraries and dependencies, which need to be imported into our Python environment. These libraries will make our tasks a lot easier, as they have readily available functions and models that can be used instead of doing that ourselves. This also makes the code more compact and readable.

Import Libraries

Typing the following commands into your Jupyter notebook to import all the required libraries.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

- pandas - used to perform data manipulation and analysis
- numpy - used to perform a wide variety of mathematical operations on arrays
- matplotlib - used for data visualization and graphical plotting
- seaborn - built on top of matplotlib with similar functionalities

These are the fundamental modules required for this project.

Loading Data:

Typing the following commands into your Jupyter notebook to import Insert csv dataset as Pandas Data Frame. There are 891 rows and 12 columns in our training dataset.

```
titanic = pd.read_csv('titanic.csv')
titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	160	0	3	Sage, Master. Thomas Henry	male	NaN	8	2	CA. 2343	69.55	NaN	S
1	181	0	3	Sage, Miss. Constance Gladys	female	NaN	8	2	CA. 2343	69.55	NaN	S
2	202	0	3	Sage, Mr. Frederick	male	NaN	8	2	CA. 2343	69.55	NaN	S
3	325	0	3	Sage, Mr. George John Jr	male	NaN	8	2	CA. 2343	69.55	NaN	S
4	793	0	3	Sage, Miss. Stella Anna	female	NaN	8	2	CA. 2343	69.55	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	467	0	2	Campbell, Mr. William	male	NaN	0	0	239853	0.00	NaN	S
887	482	0	2	Frost, Mr. Anthony Wood "Archie"	male	NaN	0	0	239854	0.00	NaN	S
888	634	0	1	Parr, Mr. William Henry Marsh	male	NaN	0	0	112052	0.00	NaN	S
889	675	0	2	Watson, Mr. Ennis Hastings	male	NaN	0	0	239856	0.00	NaN	S
890	733	0	2	Knight, Mr. Robert J	male	NaN	0	0	239855	0.00	NaN	S

891 rows × 12 columns

Data Information

```
1 titanic.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          715 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
```

```
1 titanic.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            176
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Using the df.info() command to output a comprehensive summary of the data frame's 891 rows and 12 columns as well as showing datatypes for each column.

Using df.isnull() command to displays the count of missing (null) values in each column of the Data Frame. It's a way to identify how many missing values are present in each column. We can see that Age value is missing for many rows. Out of 891 rows, the Age value is present only in 715 rows. Similarly, Cabin values are also missing in many rows. Only 204 out of 891 rows have Cabin values. There are 176 rows with missing Age, 687 rows with missing Cabin and 2 rows with missing Embarked information.

```
1 titanic.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	715.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.839399	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.993128	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	20.250000	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	130.000000	8.000000	6.000000	512.329200

Using df.describe() command to generates a summary of descriptive statistics for the numerical columns in the Data Frame. It includes metrics like mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum values. This summary provides an overview of the distribution and central tendency of the data in those columns.

Keep in mind that these commands are particularly useful during the initial stages of data analysis to understand the structure, quality, and basic statistics of the data within a Data Frame.

EXPLORATORY DATA ANALYSIS (EDA)

There is another amazing course to learn about how to start preprocessing data for any ML project Pre-processing for ML in python. This is amazing for learning how to start with any data science project, as pre-processing is one of the most important and initial steps when solving and ML problem. We'll start by checking out missing data from our data frame and replacing it with useful data.

Data Cleaning:

Data cleaning is an essential step in EDA. We must handle missing values, outliers, and inconsistencies in the dataset. Some common data-cleaning tasks include:

```
titanic.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            176
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Handling Missing Values

This code fills missing values in the 'Age' column with the mean age and fills missing values in the 'Embarked' column using a forward-fill method, with a limit of 2 consecutive missing values. Finally, it prints the count of missing values in each column of the 'titanic' DataFrame. The maximum value is missing on Cabin column. That's why we are drop this column.

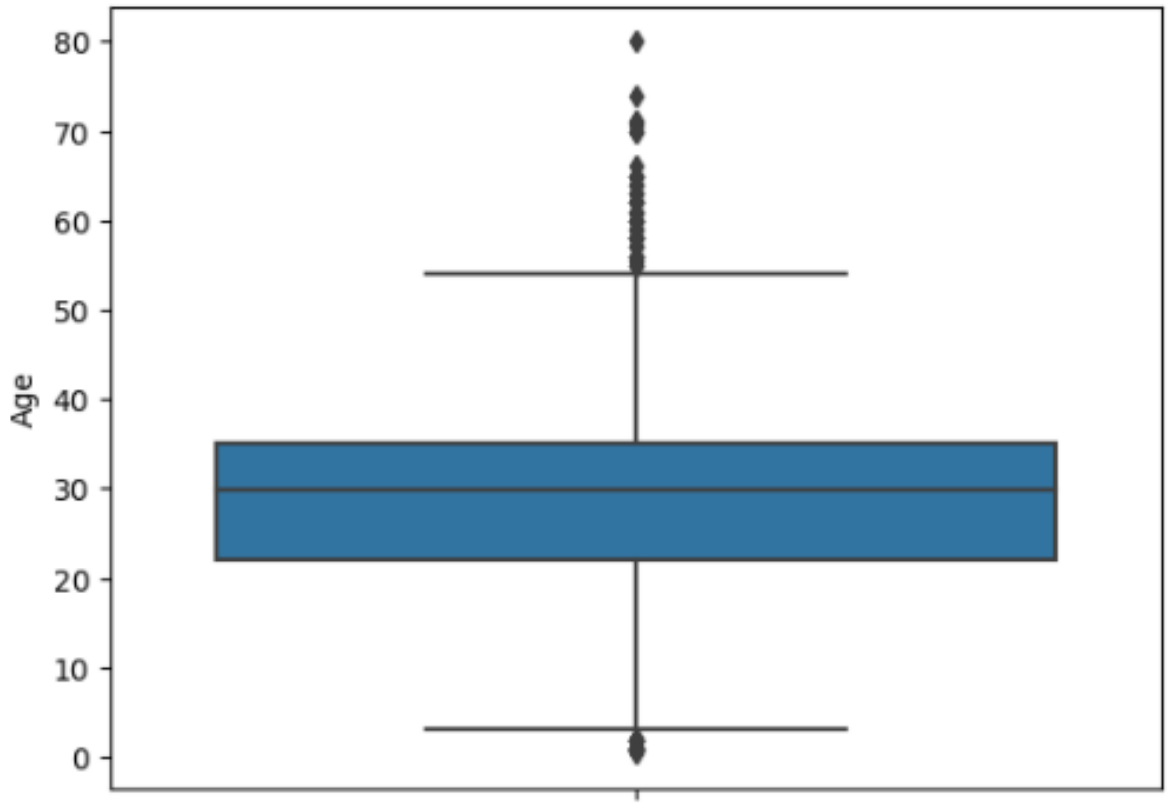
```
titanic.drop('Cabin', axis=1, inplace=True)
titanic['Age'].fillna(titanic['Age'].mean(), inplace=True)
titanic["Embarked"].fillna( method ='ffill', limit = 2, inplace = True)
print(titanic.isnull().sum())
```

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

Removing Outlier

Boxplot is a good method to see outlier

```
mean_age = titanic['Age'].mean()
df = titanic[titanic['Age'] < 81]
sns.boxplot(y='Age', data=df)
plt.show()
```



First calculate the mean age from the 'Age' column. Then create a new DataFrame with rows where age is less than 81.0. Create a boxplot of the 'Age' column using Seaborn. Now this boxplot is showing the Outlier has removed.

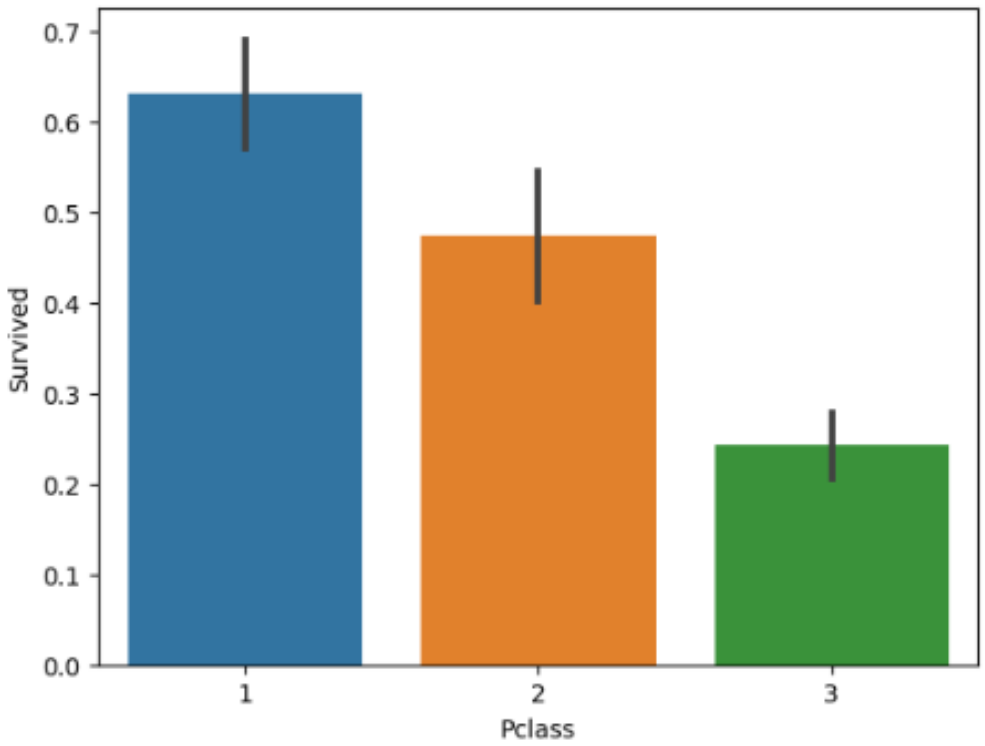
VISUALIZATION

Relationship between Features and Survival

Pclass vs. Survival: Higher class passengers have better survival chance.

```
titanic.Pclass.value_counts()
print(titanic.groupby('Pclass').Survived.value_counts())
sns.barplot(x='Pclass', y='Survived', data=titanic)
```

```
Pclass  Survived
1       1           136
        0            80
2       0            97
        1            87
3       0           372
        1           119
Name: Survived, dtype: int64

<Axes: xlabel='Pclass', ylabel='Survived'>
```

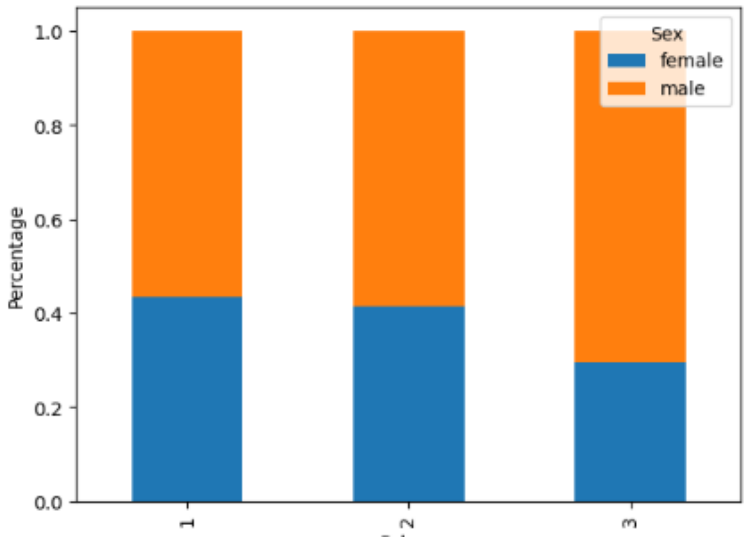


Pclass & Sex vs. Survival

Below, we just find out how many males and females are there in each *Pclass*. We then plot a stacked bar diagram with that information. We found that there are more males among the 3rd Pclass passengers.

```
tab = pd.crosstab(df['Pclass'], df['Sex'])
print (tab)
tab.div(tab.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
plt.xlabel('Pclass')
plt.ylabel('Percentage')
```

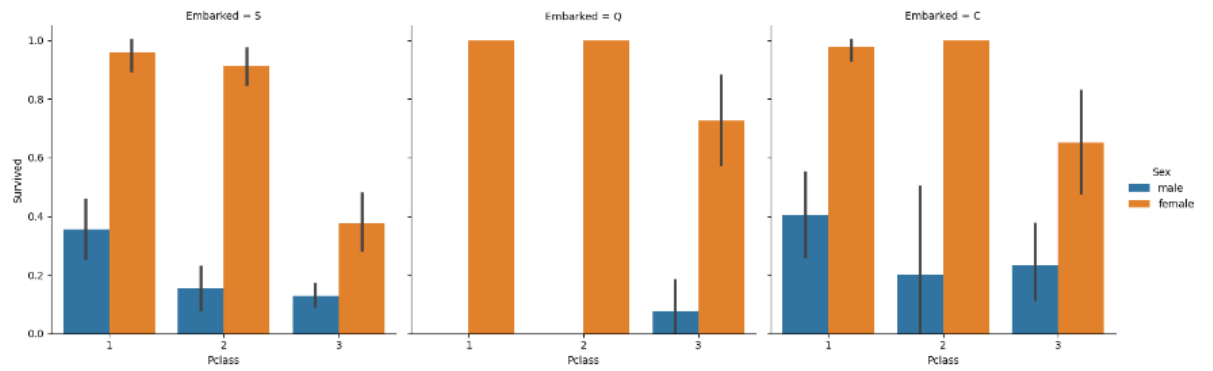
```
Sex     female  male
Pclass
1           94   122
2           76   108
3          144   346

Text(0, 0.5, 'Percentage')
```



Pclass, Sex & Embarked vs. Survival

From the above plot, it can be seen that: Almost all females from Pclass 1 and 2 survived. Females dying were mostly from 3rd Pclass. Males from Pclass 1 only have slightly higher survival chance than Pclass 2 and 3.

```
#Survival vs Pclass by Embarked and Sex
g = sns.catplot(x='Pclass', y='Survived', hue='Sex', col='Embarked', data=df, kind='bar')
plt.show()
```



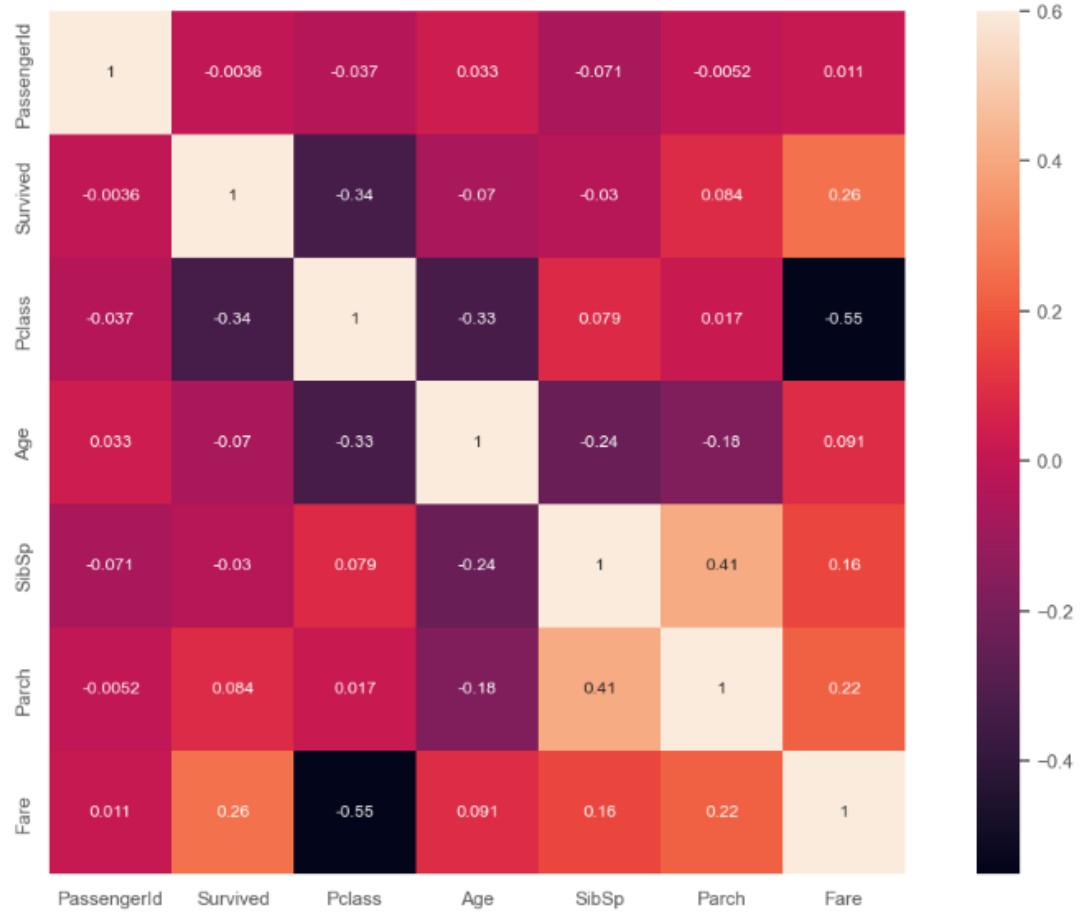
Correlation Matrix

Heatmap of Correlation between different features: A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The value is in the range of -1 to 1. If two variables have a high correlation, we can neglect one variable from those two. Positive numbers = Positive correlation, i.e. increase in one feature will increase the other feature & vice-versa. Negative numbers = Negative correlation, i.e. increase in one feature will decrease the other feature & vice-versa. In our case, we focus on which features have strong positive or negative correlation with the Survived feature

```
1 plt.figure(figsize=(15, 9))
2 sns.heatmap(df.corr(), vmax=0.6, square=True, annot=True)
```

```
C:\Users\GIS-PC\AppData\Local\Temp\ipykernel_23792\1255967912.py:2: FutureWarning: The default value of numeric_only in DataFra
me.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric
_only to silence this warning.
  sns.heatmap(df.corr(), vmax=0.6, square=True, annot=True)
```

```
<Axes: >
```



The 'Fare' shows a negative correlation with Pclass. Additionally, Fare has some level of correlation with all classes. Hence, the Fare column is an essential attribute for this project.

LABEL ENCODING

Label Encoding refers to converting the labels into the numeric form and converting them into the machine-readable form. We will convert the column 'Sex' and 'Embarked'.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
encode = ['Sex', 'Embarked']
label = LabelEncoder()
df[encode] = df[encode].apply(label.fit_transform)
#df.drop(['Name', 'Ticket'], axis=1, inplace=True)
df
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	160	0	3	1	29.839399	8	2	69.55	2
1	181	0	3	0	29.839399	8	2	69.55	2
2	202	0	3	1	29.839399	8	2	69.55	2
3	325	0	3	1	29.839399	8	2	69.55	2
4	793	0	3	0	29.839399	8	2	69.55	2
...	...	...	...	...	...	...	...	...	...
886	467	0	2	1	29.839399	0	0	0.00	2
887	482	0	2	1	29.839399	0	0	0.00	2
888	634	0	1	1	29.839399	0	0	0.00	2
889	675	0	2	1	29.839399	0	0	0.00	2
890	733	0	2	1	29.839399	0	0	0.00	2

890 rows × 9 columns

In column 'Sex', the male is converted to '1' and the female is converted to '0'. Likewise, in 'Embarked' the cities are assigned some defined number. Also remove few unnecessary columns i.e., Name and Ticket columns.

MODEL TRAINING

Now the preprocessing has been done, let's perform the model training and testing. For the train and test dataset completely, the results will be inaccurate. Hence, we will use '**train_test_split**'. We will add **random_state** with the attribute 42 to get the same split upon re-running. If you don't specify a random state, it will randomly split the data upon re-running giving inconsistent results.

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```
X = train.drop(columns=['Survived'], axis=1)
y = train['Survived']
```

```
from sklearn.model_selection import train_test_split, cross_val_score
# classify column
def classify(model):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
    model.fit(x_train, y_train)
    print('Accuracy:', model.score(x_test, y_test))

    score = cross_val_score(model, X, y, cv=5)
    print('CV Score:', np.mean(score))
```

- X contains input attributes and y contains the output attribute.
- We use cross val score() for better validation of the model.
- Here, cv=5 means that the cross-validation will split the data into 5 parts.
- np.abs() will convert the negative score to positive and np.mean() will give the average value of 5 scores.

  Let's train our data with different models.

EVALUATION MODEL PERFORMANCE

We can check precision, recall, f1-score using classification report and also see how accurate is our model for predictions:

Linear Regression: Model Report

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
classify(model)
```

```
Accuracy: 0.3906935655411078
CV Score: 0.3053618164120172
```

Logistic Regression: Model Report

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
classify(model)
```

```
Accuracy: 0.7713004484304933
CV Score: 0.7438202247191011
```

Decision Tree: Model Report

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
classify(model)
```

```
Accuracy: 0.7713004484304933
CV Score: 0.44269662921348313
```

Random Forest: Model Report

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
classify(model)
```

```
Accuracy: 0.8161434977578476
CV Score: 0.4808988764044944
```

Extra Trees: Model Report

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
classify(model)
```

```
Accuracy: 0.7937219730941704
CV Score: 0.5651685393258428
```

SELECTION A BEST MODEL

Among all the models, Logistic Regression shows the highest CV score.

Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes)

```
test = pd.read_csv('test.csv')
encode = ['Sex', 'Embarked']
label = LabelEncoder()
test[encode] = test[encode].apply(label.fit_transform)
test.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
test
```

```
X = train.drop('Survived',axis=1)
y = train['Survived']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
logmodel.score(X_train,y_train)
logmodel.score(X_test,y_test)
```

```
X = train.drop('Survived',axis=1)
y = train['Survived']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

```
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
logmodel.score(X_train,y_train)
```

```
0.8105939004815409
```

```
logmodel.score(X_test,y_test)
```

```
0.7790262172284644
```

Hyperparameter Tuning

By comparing the performance of different model configurations, we can find the best set of hyperparameters that yields the highest accuracy. This approach is a powerful tool for fine-tuning machine learning models and improving their performance. we can beat the results of our best-guess model using the grid search technique!

```
# Define Parameters
max_depth=[2, 8, 16]
n_estimators = [64, 128, 256]
param_grid = dict(max_depth=max_depth, n_estimators=n_estimators)
# Build the grid search
dfrst = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
grid = GridSearchCV(estimator=dfrst, param_grid=param_grid, cv = 5)
grid_results = grid.fit(x_train, y_train)
# Summarize the results in a readable format
print("Best:{0},using{1}".format(grid_results.cv_results_['mean_test_score'],
grid_results.best_params_))
results_df = pd.DataFrame(grid_results.cv_results_)
results_df
```

```
Best: [0.79611613 0.78005161 0.79290323 0.81387097 0.82187097 0.81867097
 0.78818065 0.78816774 0.78498065], using {'max_depth': 8, 'n_estimators': 128}
```

The list above is an overview of the tested model configurations, ranked by their prediction scores. Model number five achieved the best results. The parameters of this model are a maximum depth of 8 and several estimators of 256.

PREDICTION MAKING

```
test_x = test.drop('PassengerId',axis=1)
```

```
predictions = logmodel.predict(test_x)
```

```
final_prediction = pd.DataFrame({'PassengerId':test['PassengerId'],'Survived':predictions})
```

```
final_prediction.head()
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1

```
final_prediction.to_csv('Prediction.csv')
```

The all-predicted values are submitted a csv file

SUBMISSION RESULT

In the last step of the project, we will use the submission template to submit our predicted results. We have to submit the predicted data in PassengerId and Survived column.

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

Main Data

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	1
4	896	0

Predicted Data

CONCLUSION

In our project, we have covered a lot of details about Logistic Regression. You have learned what Logistic Regression is, how to build Logistic regression models, how to visualize the results, how to deal with missing data and some of the theoretical background information. Also, we have covered some basic concepts such as the sigmoid function, confusion matrix, exploratory data analysis, Converting Categorical Features, building logistic regression model. We still can improve our model, but this tutorial is intended to show how we can do some exploratory analysis, clean up data, and implement logistic regression in python.