8213 Shreya Bhattacharjee

MACHINE LEARNING

PROJECT TOPIC

TITANIC DISASTER PREDICTION

SUBMITTED BY:--

SHREYA BHATTACHARJEE

8213 Shreya Bhattacharjee

Mahatma Education Society's

Pillai College of Arts, Commerce & Science

(Autonomous)

Affiliated to University of Mumbai

NAAC Accredited 'A'
grade (3 cycles) Best
College Award by
University of Mumbai
ISO 9001:2015 Certified

CERTIFICATE

This is to certify that Mr. /Miss. ***SHREYA BHATTACHARJEE*** *of* ***T.Y B.Sc. C.S. Semester V|*** *has completed the practical work in the Subject of* ***MACHINE LEARNING*** *during the academic year 2022-23 under the guidance of* ***Mrs. SANJANA BHANGALE*** *being the partial requirement for the fulfilment of the curriculum of* ***Degree of Bachelor of Science in Computer Science, University of Mumbai.***

Exam seat no: 8213

Place:

Date:

Name & Signature of faculty *Name & Signature of external*

Name & Signature of Co-ordinator

ACKNOWLEDGEMENT

I Miss Shreya Bhattacharjee Student of ***Pillai College of Arts, Commerce & Science, New Panvel*** *would like to express my sincere gratitude towards our college's Computer Science Department.*

I would like to thank lecturer ***Mrs. Sanjana Bhangale*** *, and for their constant support during the project. Last but not least I thank*

all my colleagues for being with me throughout the project,

which leads to a successful completion of my project.

The project would not have been completed without creativity

and energy, which our friends provided.

Yours faithfully,

Shreya Bhattacharjee

INTRODUCTION OF PROJECT

In titanic disaster prediction, we are going to build a Logistic Regression model using a training set of samples listing passengers who survived or did not survive the Titanic disaster. Then we check the performance of the model on a test dataset to evaluate how well the model is able to predict whether these passengers of the test dataset survived or not.

While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

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).

Libraries we used :

```
# data analysis
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

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

Adding dataset:

```
[ ] train_df = pd.read_csv('/content/train.csv')
    test_df = pd.read_csv('/content/test.csv')
    combine = [train_df, test_df]
```

Data collection and pre processing

```
print(train_df.columns.values)
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

+ Code + Text

From head() method you can see the top 5 row data in the dataset

```
#preview the data
train_df.head()
```

	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

```
[ ] train_df.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

```
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
```

✓ 1s completed at 19:29

```
train_df.describe()
```

	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

```
[ ] train_df.dtypes

PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object

[ ] train_df[train_df.duplicated()]

   PassengerId  Survived  Pclass  Name  Sex  Age  SibSp  Parch  Ticket  Fare  Cabin  Embarked
```

```
#@title
train_df.corr()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

Analyze by pivoting features:

- **Pclass** We observe significant correlation (>0.5) among Pclass=1 and Survived (classifying #3). We decide to include this feature in our model.
- **Sex** We confirm the observation during problem definition that Sex=female had very high survival rate at 74% (classifying #1).
- **SibSp and Parch** These features have zero correlation for certain values. It may be best to derive a feature or a set of features from these individual features (creating #1)

```
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

```
train_df[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Sex	Survived
0	female	0.742038
1	male	0.188908

```
train_df[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

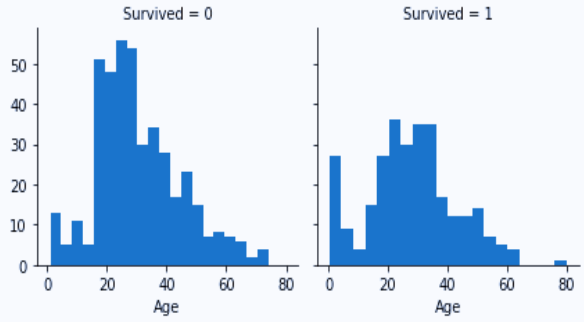
Analyze by visualizing data:

HISTOGRAM

A histogram chart is useful for analyzing continous numerical variables like Age where banding or ranges will help identify useful patterns. The histogram can indicate distribution of samples using automatically defined bins or equally ranged bands.

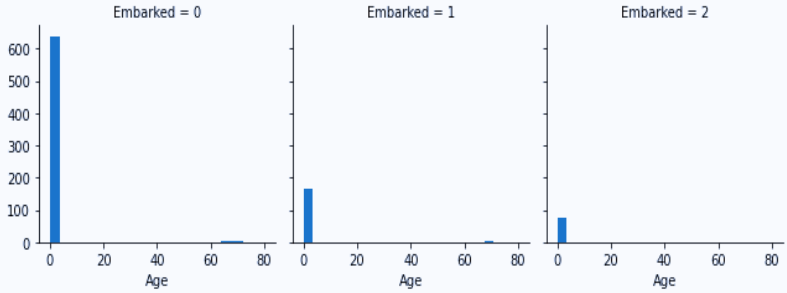
```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f43254b7550>
```



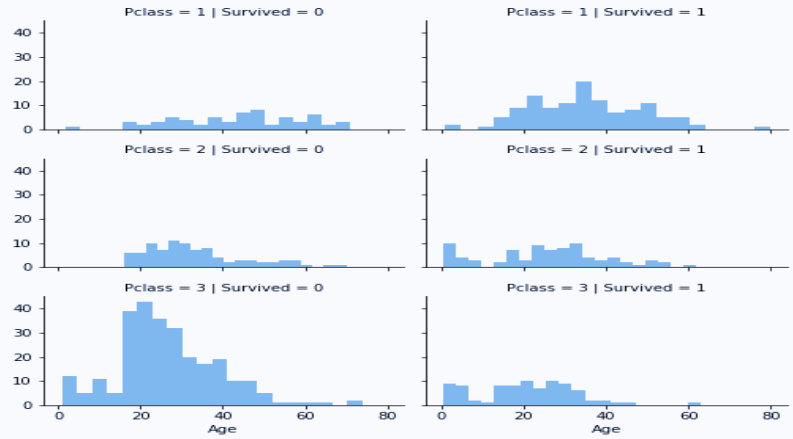

```
g = sns.FacetGrid(train_df, col='Embarked')
g.map(plt.hist, 'Age', bins=20)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f4328893df0>
```



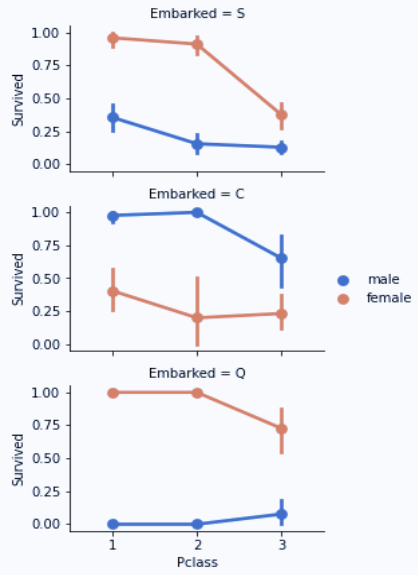

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `size` pa
  warnings.warn(msg, UserWarning)
```



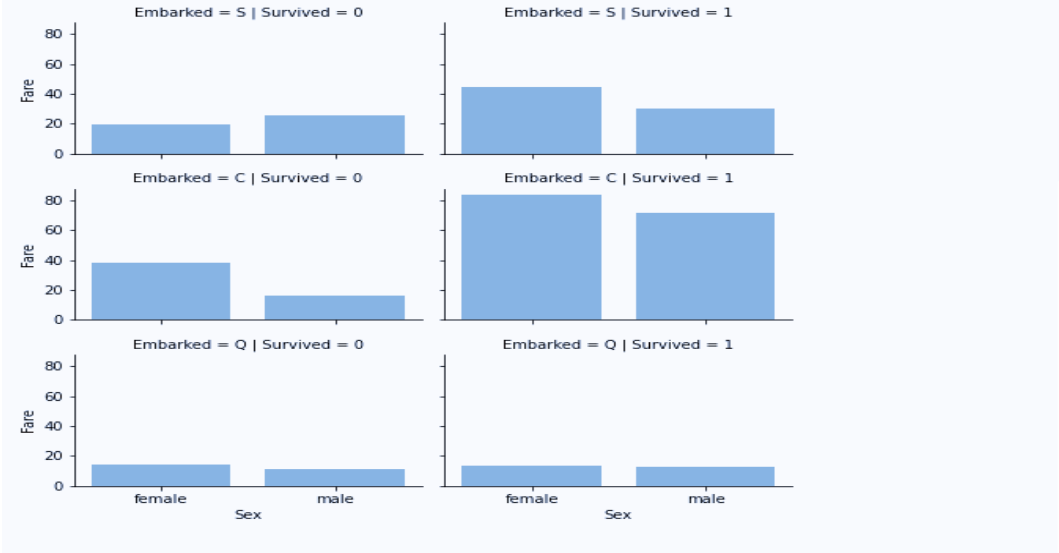

```
grid = sns.FacetGrid(train_df, row='Embarked', size=2.2, aspect=1.6)
grid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette='deep')
grid.add_legend()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `size` parameter has be
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/axisgrid.py:670: UserWarning: Using the pointplot functio
  warnings.warn(warning)
/usr/local/lib/python3.8/dist-packages/seaborn/axisgrid.py:675: UserWarning: Using the pointplot functio
  warnings.warn(warning)
<seaborn.axisgrid.FacetGrid at 0x7f4325503f10>
```

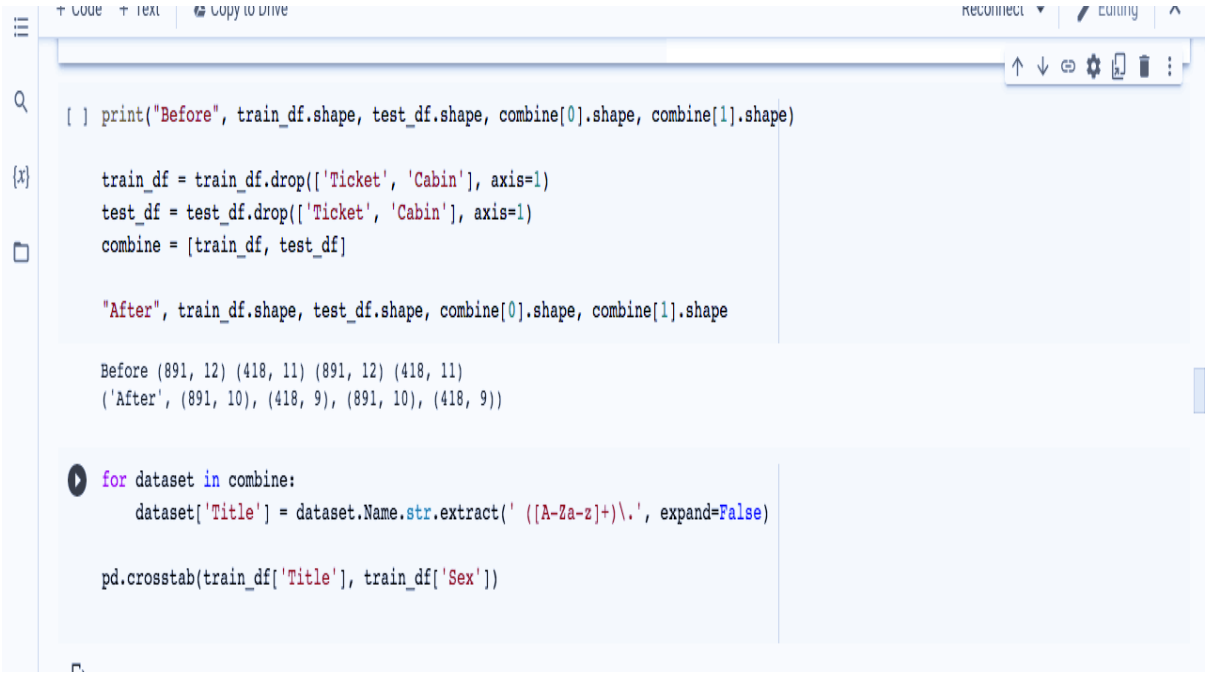


```
grid = sns.FacetGrid(train_df, row='Embarked', col='Survived', size=2.2, aspect=1.6)
grid.map(sns.barplot, 'Sex', 'Fare', alpha=.5, ci=None)
grid.add_legend()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/axisgrid.py:337: UserWarning: The `size` param
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/axisgrid.py:670: UserWarning: Using the barplo
  warnings.warn(warning)
<seaborn.axisgrid.FacetGrid at 0x7f4324f71c10>
```



Correcting by dropping features:



```
[ ] print("Before", train_df.shape, test_df.shape, combine[0].shape, combine[1].shape)

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

    "After", train_df.shape, test_df.shape, combine[0].shape, combine[1].shape

    Before (891, 12) (418, 11) (891, 12) (418, 11)
    ('After', (891, 10), (418, 9), (891, 10), (418, 9))

    for dataset in combine:
        dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

    pd.crosstab(train_df['Title'], train_df['Sex'])
```

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

Creating new feature extracting from existing:

```
for dataset in combine:
    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')

train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

convert the categorical titles to ordinal:

```
title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

train_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C	3
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S	2
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S	3
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S	1

```
train_df = train_df.drop(['Name', 'PassengerId'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
combine = [train_df, test_df]
train_df.shape, test_df.shape
```

```
((891, 9), (418, 9))
```

```
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

Completing a numerical continuous feature :

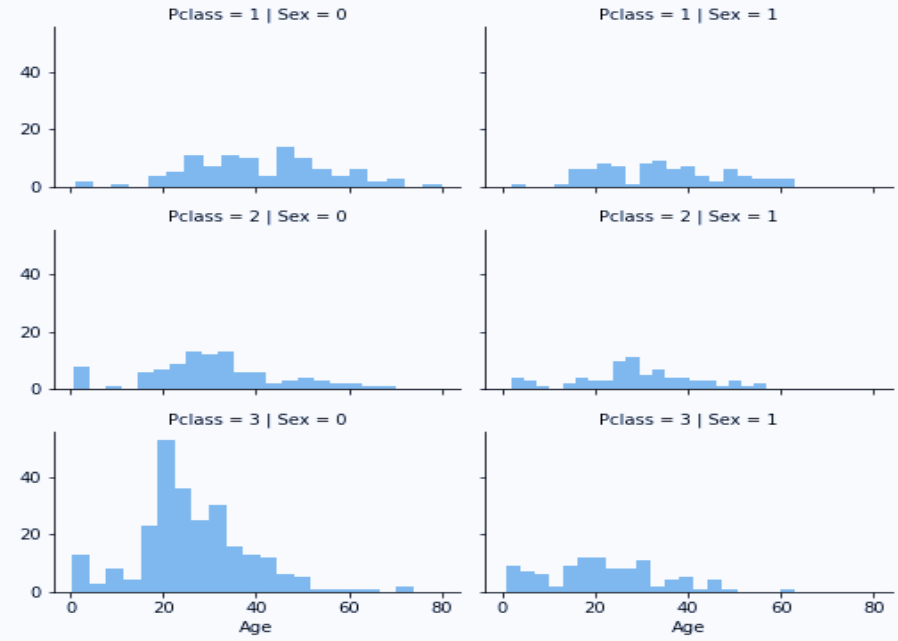
A simple way is to generate random numbers between mean and standard deviation.

More accurate way of guessing missing values is to use other correlated features. In our case we note correlation among Age, Gender, and Pclass. Guess Age values using median values for Age across sets of Pclass and Gender feature combinations. So, median Age for Pclass=1 and Gender=0, Pclass=1 and Gender=1, and so on...

Combine methods 1 and 2. So instead of guessing age values based on median, use random numbers between mean and standard deviation, based on sets of Pclass and Gender combinations.

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

```
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at 0x7f4325ab6430>
```



- preparing an empty array to contain guessed Age values based on Pclass x Gender combinations.

```
guess_ages = np.zeros((2,3))
guess_ages
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = dataset[(dataset['Sex'] == i) & \
                               (dataset['Pclass'] == j+1)]['Age'].dropna()

            # age_mean = guess_df.mean()
            # age_std = guess_df.std()
            # age_guess = rnd.uniform(age_mean - age_std, age_mean + age_std)

            age_guess = guess_df.median()

            # Convert random age float to nearest .5 age
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    for i in range(0, 2):
        for j in range(0, 3):
            dataset.loc[ (dataset.Age.isnull()) & (dataset.Sex == i) & (dataset.Pclass == j+1),\
                    'Age'] = guess_ages[i,j]
dataset['Age'] = dataset['Age'].astype(int)

train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

Create new feature combining existing features

```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1

train_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

- **categorical feature**

```
[ ] freq_port = train_df.Embarked.dropna().mode()[0]
    freq_port
```

```
'S'
```

```
[ ] for dataset in combine:
        dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)

    train_df[['Embarked', 'Survived']].groupby(['Embarked'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

+ Code + Text Copy to Drive

```
[ ] for dataset in combine:
        dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

    train_df.head()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1.0	7.2500	0	1	0	3.0
1	1	1	1	2.0	71.2833	1	3	0	2.0
2	1	3	1	1.0	7.9250	0	2	1	3.0
3	1	1	1	2.0	53.1000	0	3	0	2.0
4	0	3	0	2.0	8.0500	0	1	1	6.0

```
[ ] test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)
    test_df.head()
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	7.8292	2	1	1	6
1	893	3	1	2	7.0000	0	3	0	6
2	894	2	0	3	9.6875	2	1	1	6
3	895	3	0	1	8.6625	0	1	1	3
4	896	3	1	1	12.2875	0	3	0	3

✓ 1s completed at 19:29

```
[ ] train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)
    train_df[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().sort_values(by='FareBand', ascending=True)
```

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

```
for dataset in combine:
    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, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

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

train_df.head(10)
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1.0	0	0	1	0	3.0
1	1	1	1	2.0	3	1	3	0	2.0
2	1	3	1	1.0	1	0	2	1	3.0
3	1	1	1	2.0	3	0	3	0	2.0
4	0	3	0	2.0	1	0	1	1	6.0
5	0	3	0	1.0	1	2	1	1	3.0
6	0	1	0	3.0	3	0	1	1	3.0
7	0	3	0	0.0	2	0	4	0	0.0
8	1	3	1	1.0	1	0	3	0	3.0
9	1	2	1	0.0	2	1	3	0	0.0

✓ 1s completed at 19:29

Prediction of model

- Logistic Regression
- KNN or k-Nearest Neighbors
- Support Vector Machines
- Naive Bayes classifier
- Decision Tree
- Random Forrest
- Perceptron
- RVM or Relevance Vector Machine

Logistic regression

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either

Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.

```
[ ] # 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)
    acc_log

    80.36

    coeff_df = pd.DataFrame(train_df.columns.delete(0))
    coeff_df.columns = ['Feature']
    coeff_df["Correlation"] = pd.Series(logreg.coef_[0])

    coeff_df.sort_values(by='Correlation', ascending=False)
```

	Feature	Correlation
1	Sex	2.202191
5	Title	0.397647
2	Age	0.285899
4	Embarked	0.261853
6	IsAlone	0.125758
3	Fare	-0.087086

✓ 1s completed at 19:29

Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

```
# Support Vector Machines

svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
acc_svc
```

```
78.23
```

K-Nearest Neighbor(KNN) Algorithm

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

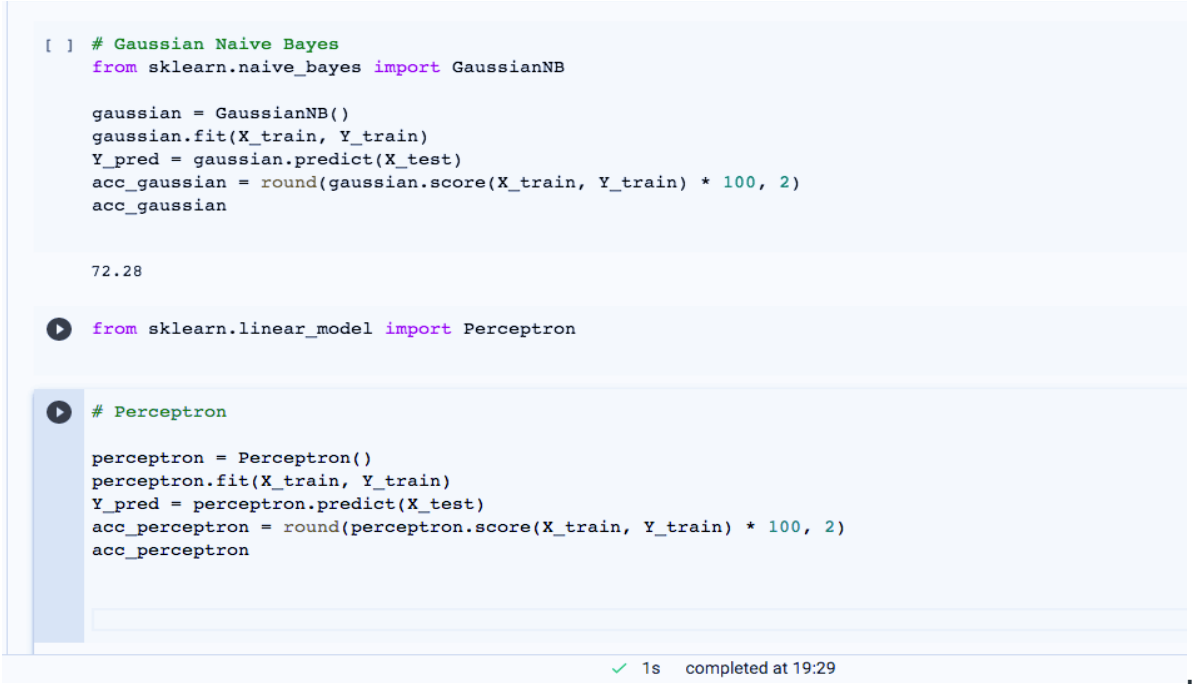
K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

```
#knn
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)
acc_knn
```

```
84.85
```

Naive Bayes Classifiers

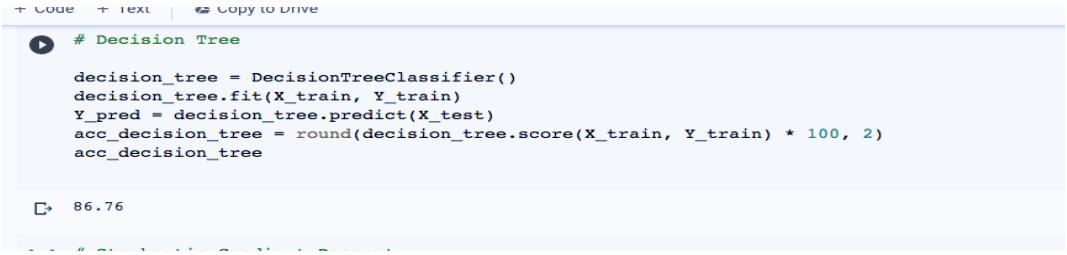
Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other



```
[ ] # Gaussian Naive Bayes
    from sklearn.naive_bayes import GaussianNB

    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)
    acc_gaussian

    72.28

    from sklearn.linear_model import Perceptron

    # Perceptron

    perceptron = Perceptron()
    perceptron.fit(X_train, Y_train)
    Y_pred = perceptron.predict(X_test)
    acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
    acc_perceptron

✓ 1s  completed at 19:29
```

Decision Tree Classification

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome**



```
+ Code  + Text  Copy to Drive

# 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)
acc_decision_tree

86.76
```

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

```
[ ] from sklearn.tree import DecisionTreeClassifier

    # Random Forest

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

    86.76
```

Model evaluation

We will now rank our evaluation of all the models to choose the best one for our problem. While both Decision Tree and Random Forest score the same, we choose to use Random Forest as they correct for decision trees' habit of overfitting to their training set.

```
#model evalution
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

8213 Shreya Bhattacharjee

	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.85
2	Logistic Regression	80.36
6	Stochastic Gradient Decent	80.02
7	Linear SVC	79.12
5	Perceptron	78.68
0	Support Vector Machines	78.23
4	Naive Bayes	72.28