Application of ML in Industries Lab

Submitted By

Harsh Morya

500098302

R2142211493

B.Tech CSE (AIML) NH B6

Submitted To

Dr. Pallabi Sharma

Assignment-1

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

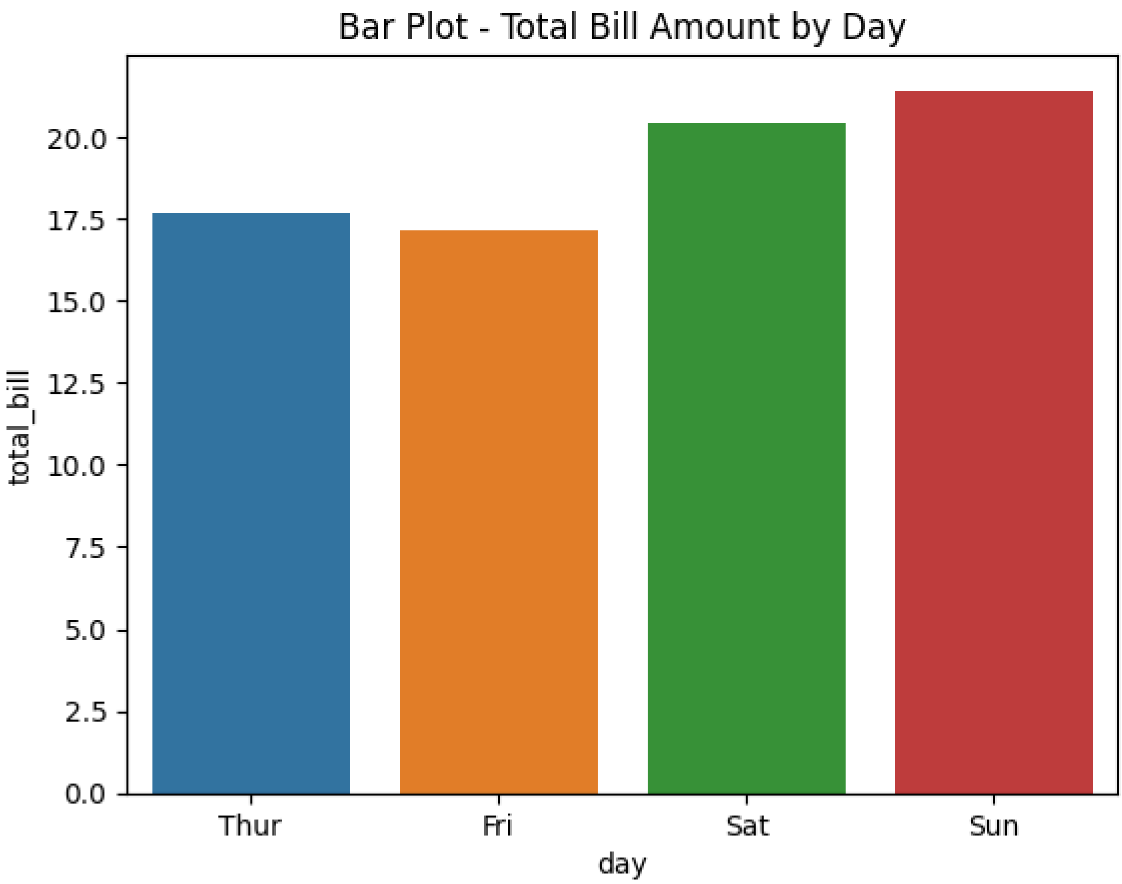
```
# Loading Iris Dataset
df=pd.read_csv("iris.csv")
# Displaying first 5 rows
df.head(5)
# Checking missing values
df.isnull().sum() # (In this Dataset there is no missing value)
# Summary of Dataset
df.describe()  # BY default take numerical column only

# Selecting subset of column using label based indexing
label_based_df=df[["petal_length","sepal_length","species"]]
# Selecting subset of column using position based indexing
index_based_df=df.iloc[:,[0,2,4]]
# Creating new dataframe by filtering rows
new_df=df[df["species"]=="Iris-virginica"]
```

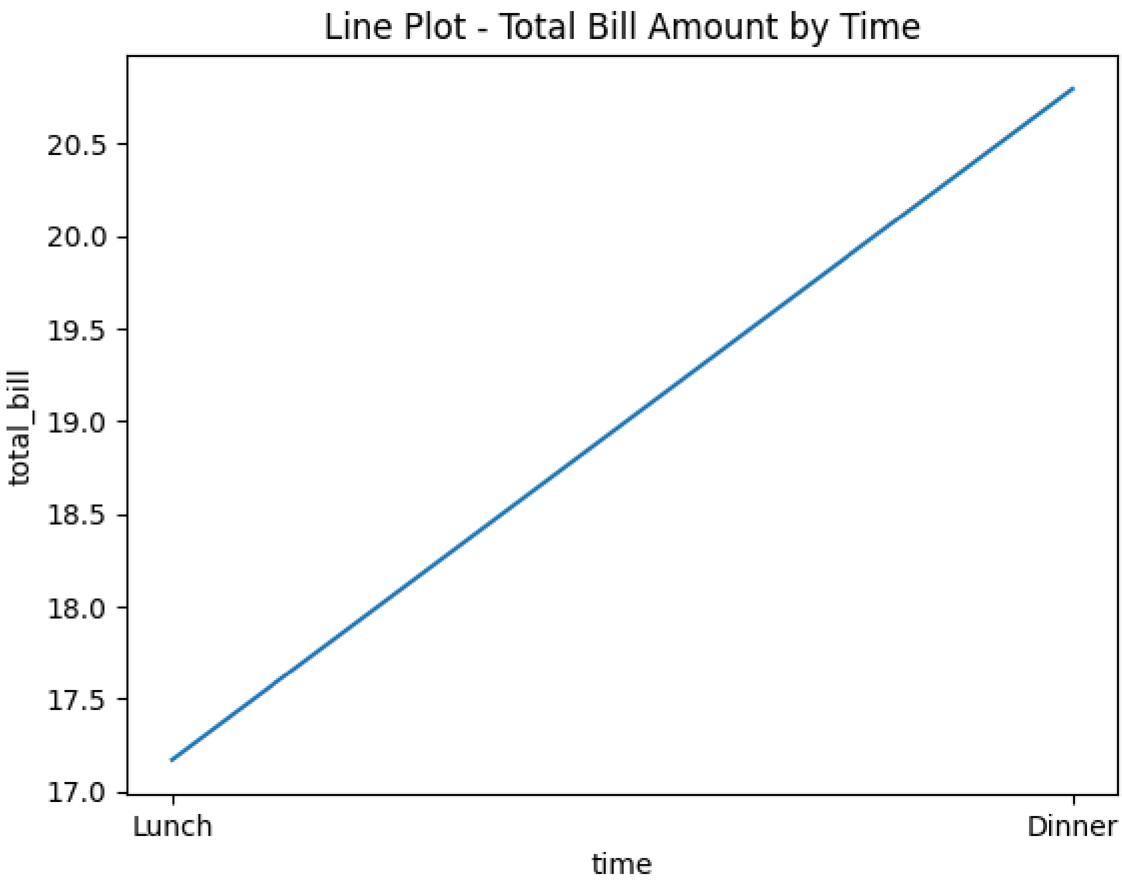
```
# Task-2
df=pd.read_csv("iris.csv")
# Checking missing values
df.isnull().sum() # (In this Dataset there is no missing value)
# Creating a new column by applying mathematical operation
df["Total_length"]=df["sepal_length"]+df["petal_length"]
# Converting cateogrical column into numerical Representation
one_hot_encoded_data = pd.get_dummies(df, columns = ["species"])
# grouping dataset by specific column and applying aggregate function
df_mean=df.groupby("species").mean()
df_count=df.groupby("species").count()
df_sum=df.groupby("species").sum()
# Represnting in meaninful way
df_mean.rename(columns={"petal_length":"petal_length_mean","petal_width":"petal_width_mean","sepal_length":"sepal_length_mean","sepal_width":"sepal_width_mean","Total_length":"Total_length_mean"})
df_count.rename(columns={"petal_length":"petal_length_count","petal_width":"petal_width_count","sepal_length":"sepal_length_count","sepal_width":"sepal_width_count","Total_length":"Total_length_count"})
```

```
df_sum.rename(columns={"petal_length":"petal_length_sum","petal_width":"petal_width_sum","sepal_length":"sepal_length_sum","sepal_width":"sepal_width_sum","Total_length":"Total_length_sum"})
```

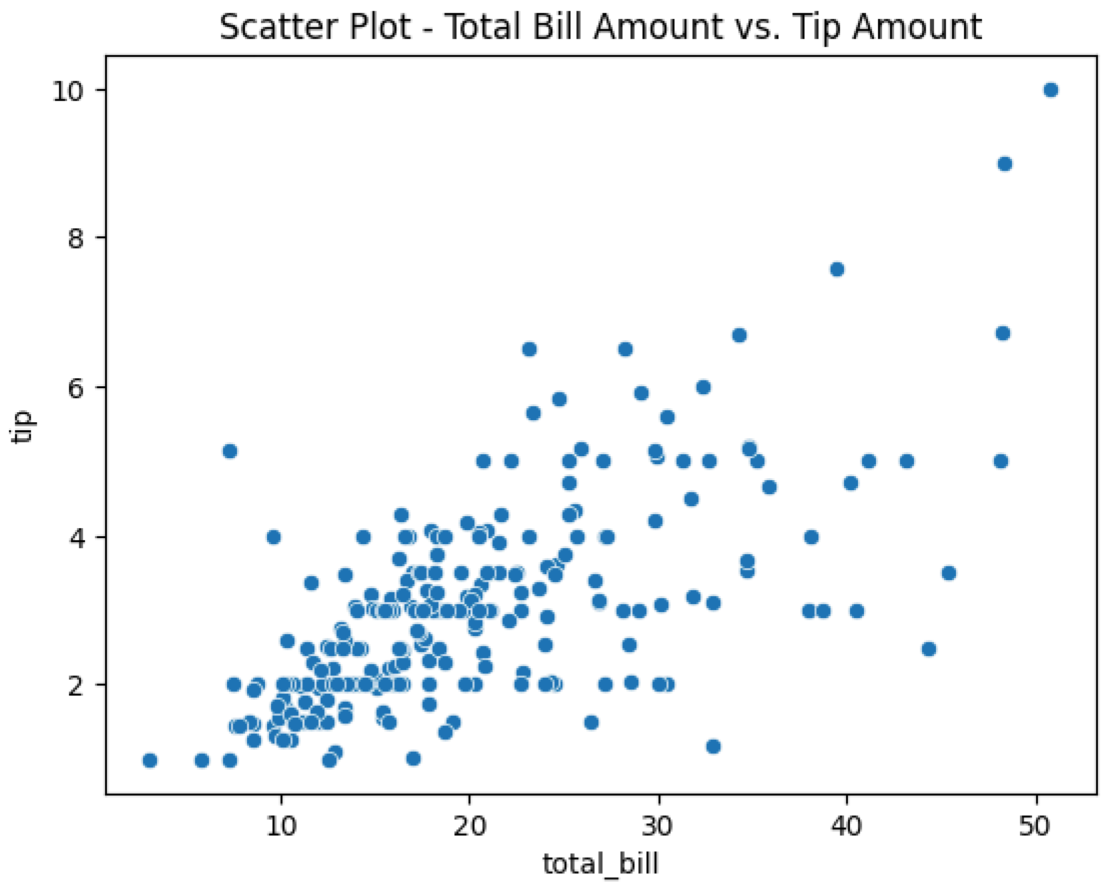
```
# Task-3
# Merge two different dataset using various type of join
df1=pd.read_csv("2017.csv")
df2=pd.read_csv("gapminder_full.csv")
df2.rename(columns={"country":"Country"},inplace=True)
merged_inner=pd.merge(df1,df2,on="Country",how="inner")
merged_left=pd.merge(df1,df2,on="Country",how="left")
merged_right=pd.merge(df1,df2,on="Country",how="right")
merged_outer=pd.merge(df1,df2,on="Country",how="outer")

# Impact of each type of join

# Inner Join: Only the rows with matching country values in both datasets are included in the result. Rows with non-matching country values are excluded.
# Left Join: All rows from the left dataset (df1) are included, and matching rows from the right dataset (df2) are added. If there is no match in the right dataset, NaN values are filled.
# Right Join: All rows from the right dataset (df2) are included, and matching rows from the left dataset (df1) are added. If there is no match in the left dataset, NaN values are filled.
# Outer Join: All rows from both datasets are included. If there is a match, values are filled; otherwise, NaN values are used.
```

```
# Task-4
tips = sns.load_dataset("tips")
sns.barplot(x="day", y="total_bill", data=tips,ci=None)
plt.title('Bar Plot - Total Bill Amount by Day')
plt.show()
sns.lineplot(x="time", y="total_bill", data=tips,ci=None)
plt.title('Line Plot - Total Bill Amount by Time')
plt.show()
sns.scatterplot(x="total_bill", y="tip", data=tips)
plt.title('Scatter Plot - Total Bill Amount vs. Tip Amount')
plt.show()
tips.head()
```


Bar Plot - Total Bill Amount by Day
total_bill
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Thur
Fri
Sat
Sun
day



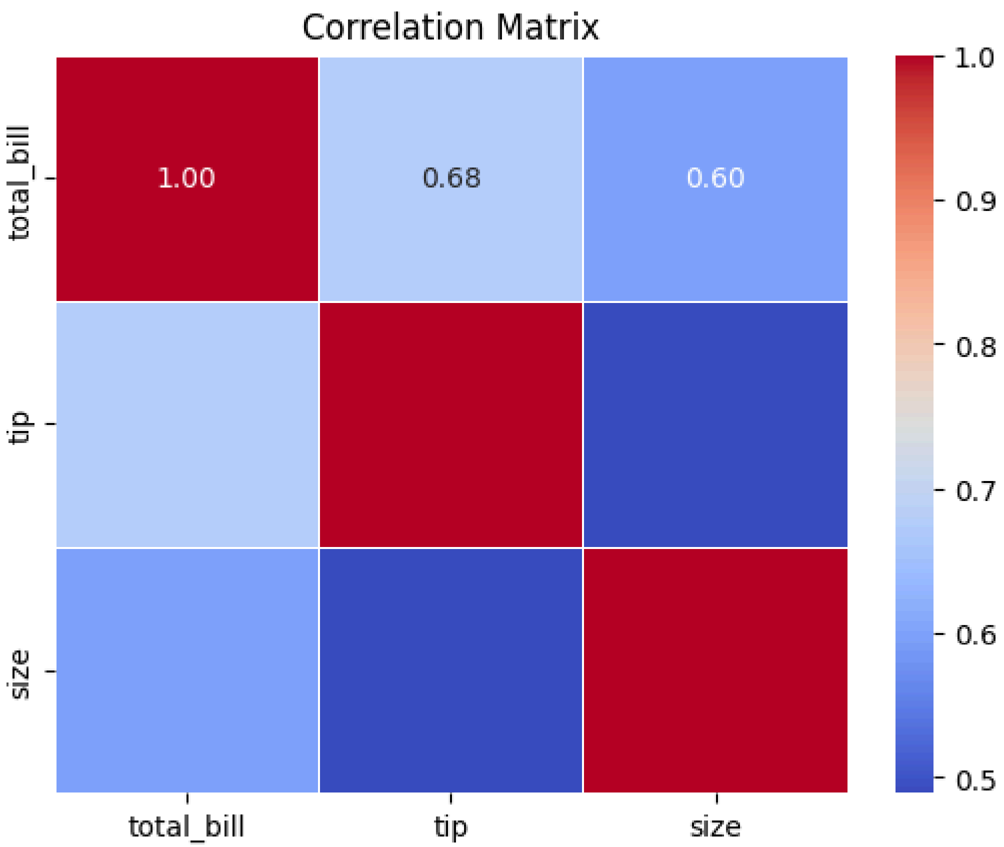
Line Plot - Total Bill Amount by Time
total_bill
20.5
20.0
19.5
19.0
18.5
18.0
17.5
17.0
Lunch
Dinner
time

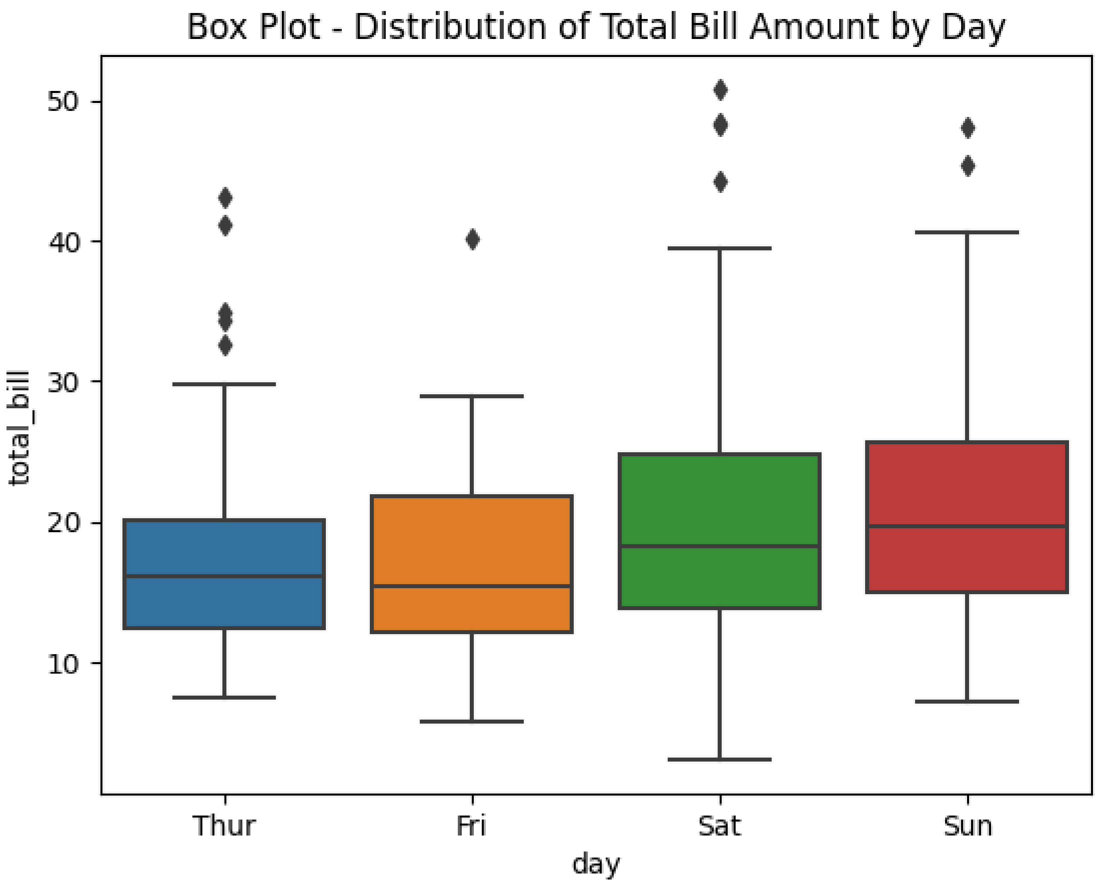


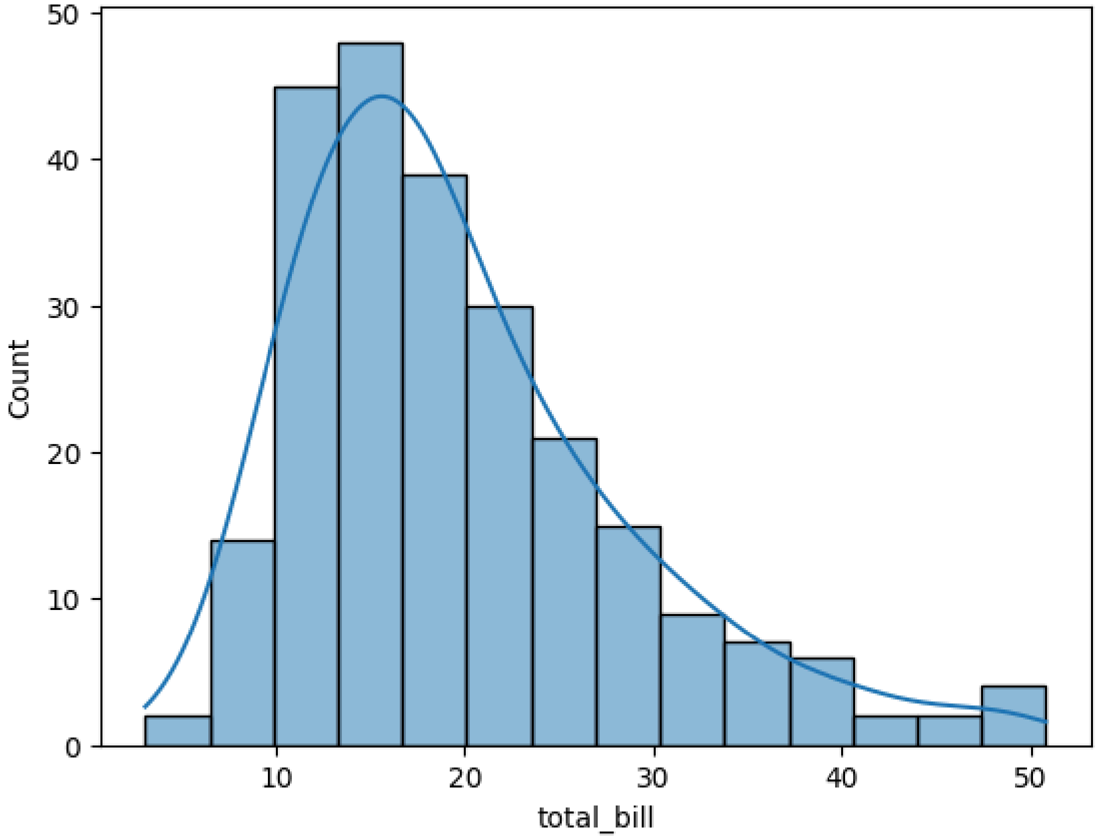
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.5 0	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
correlation_matrix = tips[["total_bill","tip","size"]].corr()
# Ploting the correlation matrix using a heatmap
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
```

```python
plt.title('Correlation Matrix')
plt.show()
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title('Box Plot - Distribution of Total Bill Amount by Day')
plt.show()
sns.histplot(tips['total_bill'], kde=True)
```




Box Plot - Distribution of Total Bill Amount by Day
total_bill
50
40
30
20
10
Thur
Fri
Sat
Sun
day




```python
# Task-5
# Creating a numpy array
arr=np.array([1,2,3,4,5,6,7,8,9,10])
arr2=np.array([11,12,13,14,15,16,17,18,19,20])
# Performing Add,subtract,multiply,and divide on these arrays
add_arr=np.add(arr,arr2)
print(add_arr)
subtract_arr=np.subtract(arr2,arr)
print(subtract_arr)
multiply_arr=np.multiply(arr,arr2)
print(multiply_arr)
divide_arr=np.divide(arr2,arr)
print(divide_arr)
```

```
[12 14 16 18 20 22 24 26 28 30]
[10 10 10 10 10 10 10 10 10 10]
[ 11  24  39  56  75  96 119 144 171 200]
[11.          6.          4.33333333  3.5         3.
2.66666667
  2.42857143  2.25        2.11111111  2.        ]
```

```
# Task-6
# Reshaping array into (2,5)
res_arr=np.reshape(arr,(2,5))
# Transpose Matrix
trans_arr=np.transpose(res_arr)
# Flattening the transposed matrix in 1D array
flatten_arr=trans_arr.flatten()
# Stacking arr ,arr2 vertically
stacked_arr=np.vstack((arr,arr2))  # array dimension/shape should be same
print(stacked_arr)
```

```
[[ 1  2  3  4  5  6  7  8  9 10]
[11 12 13 14 15 16 17 18 19 20]]
```

```
# Task-7
# Calculating mean ,median,standard deviation
mean_of_arr=arr.mean()
median_of_arr=np.median(arr)
standard_dev_of_array=np.std(arr)
# Finding max and min of arr
max_arr=arr.max()
min_arr=arr.min()
# Normalising array
normalised_arr=(arr-mean_of_arr)/standard_dev_of_array



# Task-8
# Creating bool arr for element greater than 5
bool_arr=arr>5
#  Using  boolean indexing to extract elements greater than 5
arr_greater=arr[bool_arr]
# Task-9
# Generate a 3x3 matrix with random values between 0 and 1.
arr3=np.random.rand(3,3)
# Create an array of 10 random integers between 1 and 100.
rand_int_arr=np.random.randint(1,100,size=10)
#Shuffle the elements of 'arr' randomly.
np.random.shuffle(arr)  #Keep in mind that np.random.shuffle modifies the
input array in-place
arr
```

```
array([ 2,  8,  5, 10,  6,  1,  7,  9,  3,  4])
```

```
# Task-10
# Calculating sqrt of each element in arr
square_root_arr=np.sqrt(arr)
```

```
# Calculating exponent of each element of array
exp_arr=np.exp(arr)
# Task-11
# Create a 3x3 matrix 'mat_a' with random values.
mat_a=np.random.rand(3,3)
# Create a 3x1 matrix 'vec_b' with random values.
vec_b=np.random.rand(3,1)
# .Multiply 'mat_a' and 'vec_b' using the dot product.
dot_arr=np.dot(mat_a,vec_b)
# Task-12
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
mean_subtracted_matrix = matrix - np.mean(matrix, axis=1, keepdims=True)
mean_subtracted_matrix
```

```
array([[-1.,  0.,  1.],
[-1.,  0.,  1.],
       [-1.,  0.,  1.]])
```

Assignment-2

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from math import sqrt
from sklearn .preprocessing import MinMaxScaler
import seaborn as sns
from sklearn import preprocessing
```

```
house_prediction_dataset=pd.read_csv("Housing.csv")
print(house_prediction_dataset.shape)
house_prediction_dataset.info()
house_prediction_dataset.mainroad=preprocessing.LabelEncoder().fit_transform(h
ouse_prediction_dataset.mainroad)
```

```
house_prediction_dataset.guestroom=preprocessing.LabelEncoder().fit_transform(house_prediction_dataset.guestroom)
house_prediction_dataset.basement=preprocessing.LabelEncoder().fit_transform(house_prediction_dataset.basement)
house_prediction_dataset.hotwaterheating=preprocessing.LabelEncoder().fit_transform(house_prediction_dataset.hotwaterheating)
house_prediction_dataset.airconditioning=preprocessing.LabelEncoder().fit_transform(house_prediction_dataset.airconditioning)
house_prediction_dataset.prefarea=preprocessing.LabelEncoder().fit_transform(house_prediction_dataset.prefarea)
house_prediction_dataset.furnishingstatus=preprocessing.LabelEncoder().fit_transform(house_prediction_dataset.furnishingstatus)
print(house_prediction_dataset.corr())
house_prediction_dataset.drop(["mainroad","guestroom","basement","hotwaterheating","airconditioning","parking","prefarea","furnishingstatus"],axis=1,inplace=True)

scaler=MinMaxScaler()
scaler.fit(house_prediction_dataset)
scaled=scaler.fit_transform(house_prediction_dataset)
house_prediction_dataset=pd.DataFrame(scaled,columns=house_prediction_dataset.columns)
# print(house_prediction_dataset)
x=np.array(house_prediction_dataset.iloc[:,1:5])
y=np.array(house_prediction_dataset.iloc[:,[0]])
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=1/3,random_state=60)


ln=LinearRegression()
ln.fit(X_train,y_train)
y_pred=ln.predict(X_test)

# Trained the model
print("root mean squared error is",sqrt(mean_squared_error(y_test,y_pred)))
print("mean absolute error is",(mean_absolute_error(y_test,y_pred)))
print("r2 value is",r2_score(y_test,y_pred))
```

Output:

(545, 13)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 545 entries, 0 to 544

Data columns (total 13 columns):

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
                     price      area  bedrooms  bathrooms   stories  mainroad  \
price             1.000000  0.535997  0.366494   0.517545  0.420712  0.296898
area              0.535997  1.000000  0.151858   0.193820  0.083996  0.288874
bedrooms          0.366494  0.151858  1.000000   0.373930  0.408564 -0.012033
bathrooms         0.517545  0.193820  0.373930   1.000000  0.326165  0.042398
stories           0.420712  0.083996  0.408564   0.326165  1.000000  0.121706
mainroad          0.296898  0.288874 -0.012033   0.042398  0.121706  1.000000
guestroom         0.255517  0.140297  0.080549   0.126469  0.043538  0.092337
basement          0.187057  0.047417  0.097312   0.102106 -0.172394  0.044002
hotwaterheating   0.093073 -0.009229  0.046049   0.067159  0.018847 -0.011781
airconditioning   0.452954  0.222393  0.160603   0.186915  0.293602  0.105423
parking           0.384394  0.352980  0.139270   0.177496  0.045547  0.204433
prefarea          0.329777  0.234779  0.079023   0.063472  0.044425  0.199876
furnishingstatus -0.304721 -0.171445 -0.123244  -0.143559 -0.104672 -0.156726
```

	guestroom	basement	hotwaterheating	airconditioning \
price	0.255517	0.187057	0.093073	0.452954
area	0.140297	0.047417	-0.009229	0.222393
bedrooms	0.080549	0.097312	0.046049	0.160603
bathrooms	0.126469	0.102106	0.067159	0.186915
stories	0.043538	-0.172394	0.018847	0.293602
mainroad	0.092337	0.044002	-0.011781	0.105423
guestroom	1.000000	0.372066	-0.010308	0.138179
basement	0.372066	1.000000	0.004385	0.047341
hotwaterheating	-0.010308	0.004385	1.000000	-0.130023
airconditioning	0.138179	0.047341	-0.130023	1.000000
parking	0.037466	0.051497	0.067864	0.159173
prefarea	0.160897	0.228083	-0.059411	0.117382
furnishingstatus	-0.118328	-0.112831	-0.031628	-0.150477

	parking	prefarea	furnishingstatus
price	0.384394	0.329777	-0.304721
area	0.352980	0.234779	-0.171445
bedrooms	0.139270	0.079023	-0.123244
bathrooms	0.177496	0.063472	-0.143559
stories	0.045547	0.044425	-0.104672
mainroad	0.204433	0.199876	-0.156726
guestroom	0.037466	0.160897	-0.118328
basement	0.051497	0.228083	-0.112831
hotwaterheating	0.067864	-0.059411	-0.031628
airconditioning	0.159173	0.117382	-0.150477
parking	1.000000	0.091627	-0.177539
prefarea	0.091627	1.000000	-0.107686
furnishingstatus	-0.177539	-0.107686	1.000000

root mean squared error is 0.11078229163300621 mean absolute error is 0.08509610235995772

r2 value is 0.47501848785800893

Assignment-3

```
import pandas as dm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix,
roc_curve, roc_auc_score

dataset=dm.read_csv('WineQT.csv')
print(dataset.head())
print(dataset.info())
print(dataset.describe().T)
df=dm.DataFrame(dataset)

df['best quality'] = [1 if x > 5 else 0 for x in df.quality]

# dataset.hist(bins=25,figsize=(10,10))
# plt.show()
correlation=dataset.corr()
print(correlation)

#null values
p=dataset.isnull().sum()
print(p)

#spliting dataset
X=df.drop(['quality','best quality'],axis=1)
y=df['best quality']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=
40)

#Normalizing
Nor=MinMaxScaler()
Nor_fit=Nor.fit(X_train)
X_train=Nor_fit.transform(X_train)
X_test=Nor_fit.transform(X_test)
print(X_train)
#models
models=[SVC(kernel='rbf')]
for i in range(1):
```

```
    models[i].fit(X_train,y_train)
    print('Training Accuracy : ', metrics.roc_auc_score(y_train,
models[i].predict(X_train)))
    print('Validation Accuracy : ', metrics.roc_auc_score(y_test,
models[i].predict(X_test)))


y_train_pred = models[0].predict(X_train)
y_test_pred = models[0].predict(X_test)

#Evaluation
# ROC curve and AUC curve
fpr_train, tpr_train, _ = roc_curve(y_train, y_train_pred)
fpr_test, tpr_test, _ = roc_curve(y_test, y_test_pred)
auc_train = roc_auc_score(y_train, y_train_pred)
auc_test = roc_auc_score(y_test, y_test_pred)


plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, label=f"Train ROC Curve (AUC =
{auc_train:.2f})")
plt.plot(fpr_test, tpr_test, label=f"Test ROC Curve (AUC = {auc_test:.2f})")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random
Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

output:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides \
0	7.4	0.70	0.00	1.9	0.076
1	7.8	0.88	0.00	2.6	0.098
2	7.8	0.76	0.04	2.3	0.092
3	11.2	0.28	0.56	1.9	0.075
4	7.4	0.70	0.00	1.9	0.076

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates \
0	11.0	34.0	0.9978	3.51	0.56
1	25.0	67.0	0.9968	3.20	0.68

```
2            15.0                 54.0   0.9970  3.26       0.65
3            17.0                 60.0   0.9980  3.16       0.58
4            11.0                 34.0   0.9978  3.51       0.56

   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
3      9.8        6   3
4      9.4        5   4
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64  
 12  Id                    1143 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
None
```

	count	mean	std	min	25% \
fixed acidity	1143.0	8.311111	1.747595	4.60000	7.10000
volatile acidity	1143.0	0.531339	0.179633	0.12000	0.39250
citric acid	1143.0	0.268364	0.196686	0.00000	0.09000
residual sugar	1143.0	2.532152	1.355917	0.90000	1.90000
chlorides	1143.0	0.086933	0.047267	0.01200	0.07000
free sulfur dioxide	1143.0	15.615486	10.250486	1.00000	7.00000
total sulfur dioxide	1143.0	45.914698	32.782130	6.00000	21.00000
density	1143.0	0.996730	0.001925	0.99007	0.99557
pH	1143.0	3.311015	0.156664	2.74000	3.20500
sulphates	1143.0	0.657708	0.170399	0.33000	0.55000
alcohol	1143.0	10.442111	1.082196	8.40000	9.50000
quality	1143.0	5.657043	0.805824	3.00000	5.00000
Id	1143.0	804.969379	463.997116	0.00000	411.00000

	50%	75%	max
fixed acidity	7.90000	9.100000	15.90000
volatile acidity	0.52000	0.640000	1.58000
citric acid	0.25000	0.420000	1.00000
residual sugar	2.20000	2.600000	15.50000
chlorides	0.07900	0.090000	0.61100
free sulfur dioxide	13.00000	21.000000	68.00000
total sulfur dioxide	37.00000	61.000000	289.00000
density	0.99668	0.997845	1.00369
pH	3.31000	3.400000	4.01000
sulphates	0.62000	0.730000	2.00000
alcohol	10.20000	11.100000	14.90000
quality	6.00000	6.000000	8.00000
Id	794.00000	1209.500000	1597.00000

	fixed acidity	volatile acidity	citric acid \
fixed acidity	1.000000	-0.250728	0.673157

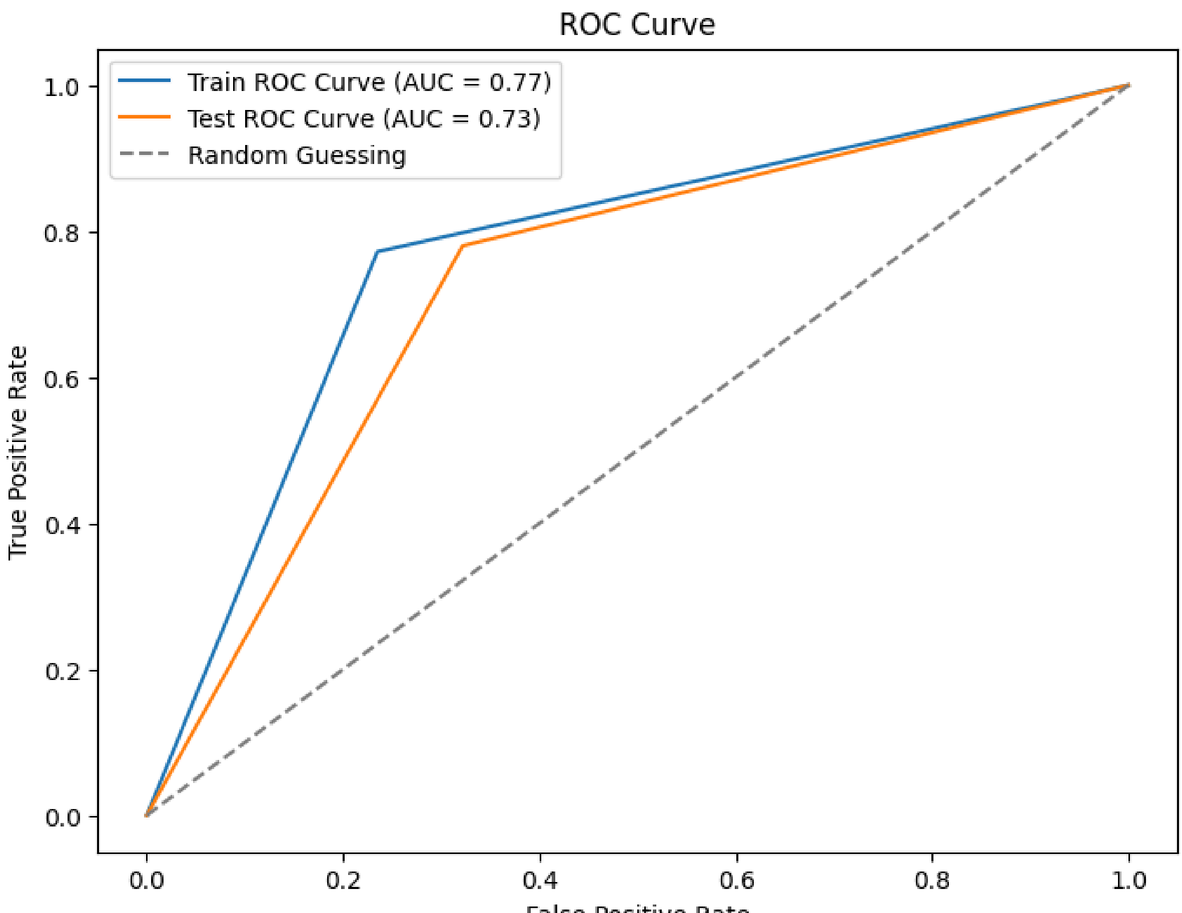
```
volatile acidity       -0.250728        1.000000    -0.544187
citric acid             0.673157       -0.544187     1.000000
residual sugar          0.171831       -0.005751     0.175815
chlorides               0.107889        0.056336     0.245312
free sulfur dioxide    -0.164831       -0.001962    -0.057589
total sulfur dioxide   -0.110628        0.077748     0.036871
density                 0.681501        0.016512     0.375243
pH                     -0.685163        0.221492    -0.546339
sulphates               0.174592       -0.276079     0.331232
alcohol                -0.075055       -0.203909     0.106250
quality                 0.121970       -0.407394     0.240821
Id                     -0.275826       -0.007892    -0.139011

                      residual sugar  chlorides  free sulfur dioxide  \
fixed acidity               0.171831   0.107889            -0.164831
volatile acidity           -0.005751   0.056336            -0.001962
citric acid                 0.175815   0.245312            -0.057589
residual sugar              1.000000   0.070863             0.165339
chlorides                   0.070863   1.000000             0.015280
free sulfur dioxide         0.165339   0.015280             1.000000
total sulfur dioxide        0.190790   0.048163             0.661093
density                     0.380147   0.208901            -0.054150
pH                         -0.116959  -0.277759             0.072804
sulphates                   0.017475   0.374784             0.034445
alcohol                     0.058421  -0.229917            -0.047095
quality                     0.022002  -0.124085            -0.063260
Id                         -0.046344  -0.088099             0.095268

                      total sulfur dioxide   density        pH  sulphates  \
fixed acidity                    -0.110628  0.681501 -0.685163   0.174592
volatile acidity                  0.077748  0.016512  0.221492  -0.276079
```

citric acid	0.036871	0.375243	-0.546339	0.331232
residual sugar	0.190790	0.380147	-0.116959	0.017475
chlorides	0.048163	0.208901	-0.277759	0.374784
free sulfur dioxide	0.661093	-0.054150	0.072804	0.034445
total sulfur dioxide	1.000000	0.050175	-0.059126	0.026894
density	0.050175	1.000000	-0.352775	0.143139
pH	-0.059126	-0.352775	1.000000	-0.185499
sulphates	0.026894	0.143139	-0.185499	1.000000
alcohol	-0.188165	-0.494727	0.225322	0.094421
quality	-0.183339	-0.175208	-0.052453	0.257710
Id	-0.107389	-0.363926	0.132904	-0.103954

	alcohol	quality	Id
fixed acidity	-0.075055	0.121970	-0.275826
volatile acidity	-0.203909	-0.407394	-0.007892
citric acid	0.106250	0.240821	-0.139011
residual sugar	0.058421	0.022002	-0.046344
chlorides	-0.229917	-0.124085	-0.088099
free sulfur dioxide	-0.047095	-0.063260	0.095268
total sulfur dioxide	-0.188165	-0.183339	-0.107389
density	-0.494727	-0.175208	-0.363926
pH	0.225322	-0.052453	0.132904
sulphates	0.094421	0.257710	-0.103954
alcohol	1.000000	0.484866	0.238087
quality	0.484866	1.000000	0.069708
Id	0.238087	0.069708	1.000000

fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0

```
free sulfur dioxide    0
total sulfur dioxide   0
density                0
pH                     0
sulphates              0
alcohol                0
quality                0
Id                     0
dtype: int64
[[0.22727273 0.34246575 0.06       ... 0.08982036 0.20754717 0.87131199]
 [0.66363636 0.30821918 0.5        ... 0.26946108 0.37735849 0.21343377]
 [0.04545455 0.26712329 0.18       ... 0.32335329 0.79245283 0.72379159]
 ...
 [0.32727273 0.60273973 0.09       ... 0.13173653 0.05660377 0.14438167]
 [0.24545455 0.3630137  0.         ... 0.08383234 0.24528302 0.00188324]
 [0.34545455 0.3630137  0.6        ... 0.11377246 0.09433962 0.18832392]]
Training Accuracy :  0.7685715001974192
Validation Accuracy :  0.7 291608391608392
```



Assignment-4

```
import pandas as pd

import numpy as np

from sklearn import preprocessing

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
```

```
air_dataset=pd.read_csv("city_day.csv")

air_dataset=air_dataset.iloc[-20000:]

air_dataset.head(10)


air_dataset.drop(columns=["Date","City","Xylene"],inplace=True)
```

```
mean_df=air_dataset[["PM2.5","PM10","NO","NO2","NOx","NH3"  ,"CO",  "SO2",  "O3",   "Benzene",
 "Toluene","AQI"]].mean(axis=0,skipna=True)

print(mean_df)

air_dataset.fillna(value={"PM2.5":mean_df[0],"PM10":mean_df[1],"NO":mean_df[2],"NO2":mean_df[3],"NOx":
mean_df[4],"NH3":mean_df[5],"CO":mean_df[6],  "SO2":mean_df[7],   "O3":mean_df[8],
 "Benzene":mean_df[9],   "Toluene":mean_df[10],"AQI":mean_df[11]},inplace=True)
```

```
air_dataset.info()

air_dataset.AQI_Bucket=preprocessing.LabelEncoder().fit_transform(air_dataset.AQI_Bucket)

air_dataset.head()

x=np.array(air_dataset.iloc[:,0:12])

y=np.array(air_dataset.iloc[:,12])

X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=1/3,random_state=6)

knn=KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train,y_train)

y_pred=knn.predict(X_test)
```

```
acc=accuracy_score(y_test,y_pred)
print(acc)
```

Output:

0.927703614819259

Assignment-5

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score,classification_report
import seaborn as sns
# 1. Load the dataset


df = pd.read_csv("creditcard.csv")

# 2. Data preprocessing
# Check for missing values
missing_values = df.isnull().sum()
print("Missing values:\n", missing_values)
print(df.info())
# 3. Split the dataset
x=np.array(df.iloc[:,0:29])
y=np.array(df.iloc[:,30])
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=1/3,random_state=6)
```

```
# # 4. Logistic Regression Model
ln=LogisticRegression()
ln.fit(X_train,y_train)
```

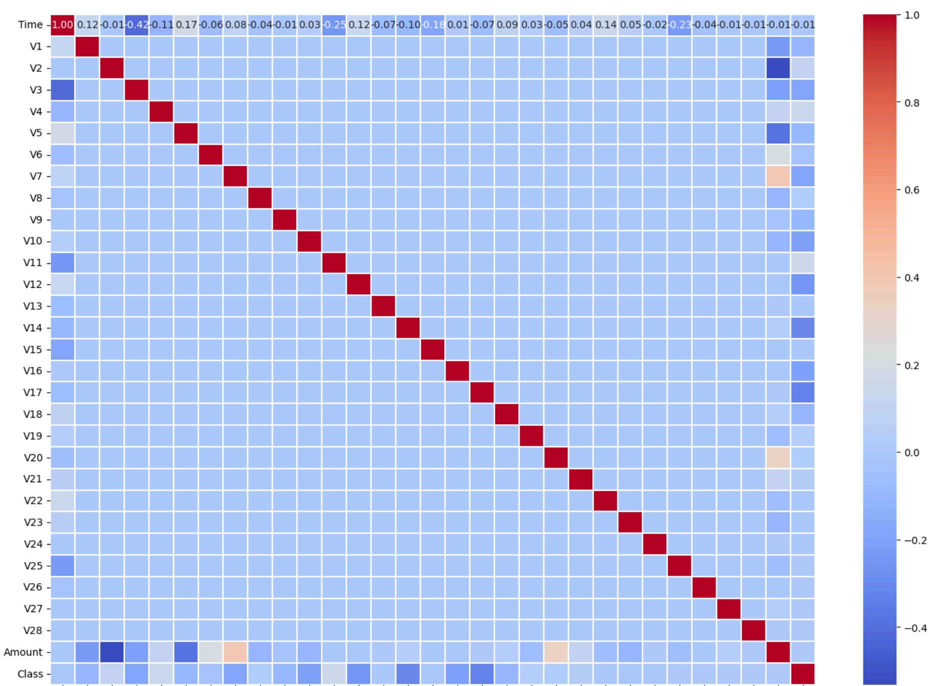
```
y_pred=ln.predict(X_test)

print(y_pred)

cm=confusion_matrix(y_test,y_pred)

print(cm)

acc=accuracy_score(y_test,y_pred)

print(acc)

print(classification_report(y_test,y_pred))
```

```
# # 5. Train the model

# model.fit(X_train, y_train)
```

```
plt.figure(figsize=(17, 12))  # Adjust the width and height according to your preference


# Create the heatmap

sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=.2)
```

```
# Show the plot

plt.show()
```

Output:



Assignment-6

```
import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.api as sm


np.random.seed(0)

group1 = np.random.normal(loc=50, scale=10, size=100)

group2 = np.random.normal(loc=55, scale=10, size=100)

group3 = np.random.normal(loc=60, scale=10, size=100)

data = pd.DataFrame({'Group1': group1, 'Group2': group2, 'Group3': group3})

z_stat, p_value = sm.stats.ztest(group1, group2)
print("Z-test results:")
print(f"Z-statistic: {z_stat}")
print(f"P-value: {p_value}")
if p_value < 0.05:
    print("Reject null hypothesis: There is a significant difference between
means of Group 1 and Group 2.")
else:
    print("Fail to reject null hypothesis: There is no significant difference
between means of Group 1 and Group 2.")

t_stat, p_value = stats.ttest_ind(group1, group3)
print("\nT-test results:")
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")
if p_value < 0.05:
    print("Reject null hypothesis: There is a significant difference between
means of Group 1 and Group 3.")
else:
    print("Fail to reject null hypothesis: There is no significant difference
between means of Group 1 and Group 3.")

f_stat, p_value = stats.f_oneway(group1, group2, group3)
print("\nANOVA test results:")
print(f"F-statistic: {f_stat}")
print(f"P-value: {p_value}")
if p_value < 0.05:
```

```
    print("Reject null hypothesis: There is a significant difference in means between at least two groups.")
else:
    print("Fail to reject null hypothesis: There is no significant difference in means between groups.")
```

Output:

Z-test results:

Z-statistic: -3.597192759749625

P-value: 0.00032167010560191873

Reject null hypothesis: There is a significant difference between means of Group 1 and Group 2.

T-test results:

T-statistic: -6.322395197755704

P-value: 1.674734808649447e-09

Reject null hypothesis: There is a significant difference between means of Group 1 and Group 3.

ANOVA test results:

F-statistic: 19.476212850706954

P-value: 1.1274108620248522e-08

Reject null hypothesis: There is a significant difference in means between at least two groups.

Batch: B6

Roll No.: R2142211493

Lab: 6

Experiment:7

SourceCode:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Preprocess data
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
```

```
])

# Compile the model
model.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10, batch_size=64,
validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the model
model.save('cifar10_model.h5')
```

Screenshot:

```python
# Train the model
history = model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_data=(test_images, test_labels))
```

✓ 2m 4.2s

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 17ms/step - accuracy: 0.3366 - loss: 1.7966 - val_accuracy: 0.5338 - val_loss: 1.3235
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.5516 - loss: 1.2669 - val_accuracy: 0.5950 - val_loss: 1.1668
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.6251 - loss: 1.0699 - val_accuracy: 0.6535 - val_loss: 1.0005
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.6703 - loss: 0.9463 - val_accuracy: 0.6565 - val_loss: 0.9746
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 17ms/step - accuracy: 0.6968 - loss: 0.8755 - val_accuracy: 0.6839 - val_loss: 0.9204
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.7186 - loss: 0.8081 - val_accuracy: 0.7002 - val_loss: 0.8626
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.7342 - loss: 0.7621 - val_accuracy: 0.7125 - val_loss: 0.8431
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.7477 - loss: 0.7220 - val_accuracy: 0.7044 - val_loss: 0.8633
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.7627 - loss: 0.6772 - val_accuracy: 0.7170 - val_loss: 0.8359
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.7824 - loss: 0.6274 - val_accuracy: 0.7177 - val_loss: 0.8264
```

```python
# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

[7] ✓ 1.6s

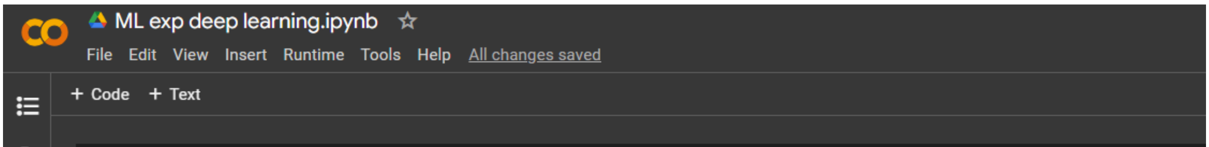
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7218 - loss: 0.8127
Test accuracy: 0.7177000045776367
```

```python
# Save the model
model.save('cifar10_model.h5')
```

[8] ✓ 0.2s

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
```

Assignment-8



```
# Importing libraries
import numpy as np
import tensorflow as tf
from tensorflow import keras

# Generate some dummy data
X_train = np.random.random((1000, 20))  # 1000 samples, 20 features
y_train = np.random.randint(2, size=(1000,))  # Binary labels

# Define the model
simple_model = keras.Sequential([
    keras.layers.Dense(16, input_dim=20, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
simple_model.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=['accuracy'])

# Train the model
simple_model.fit(X_train, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
32/32 [==============================] - 1s 3ms/step - loss: 0.7070 - accuracy: 0.5200
Epoch 2/10
32/32 [==============================] - 0s 2ms/step - loss: 0.7029 - accuracy: 0.5170
Epoch 3/10
32/32 [==============================] - 0s 2ms/step - loss: 0.7010 - accuracy: 0.5240
Epoch 4/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6998 - accuracy: 0.5280
Epoch 5/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6978 - accuracy: 0.5190
Epoch 6/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6966 - accuracy: 0.5240
```

```python
# Compile the model
deep_model.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

# Train the model
deep_model.fit(X_train, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
32/32 [==============================] - 1s 2ms/step - loss: 0.6964 - accuracy: 0.4880
Epoch 2/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6927 - accuracy: 0.5230
Epoch 3/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6897 - accuracy: 0.5200
Epoch 4/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6871 - accuracy: 0.5320
Epoch 5/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6848 - accuracy: 0.5390
Epoch 6/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6817 - accuracy: 0.5450
Epoch 7/10
32/32 [==============================] - 0s 3ms/step - loss: 0.6790 - accuracy: 0.5650
Epoch 8/10
32/32 [==============================] - 0s 3ms/step - loss: 0.6735 - accuracy: 0.5690
Epoch 9/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6701 - accuracy: 0.5690
Epoch 10/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6676 - accuracy: 0.5830
<keras.src.callbacks.History at 0x7dc8259973d0>
```


```
Epoch 9/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6701 - accuracy: 0.5690
Epoch 10/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6676 - accuracy: 0.5830
<keras.src.callbacks.History at 0x7dc8259973d0>
```


```python
# Generate some dummy test data
X_test = np.random.random((500, 20))  # 500 samples for testing
y_test = np.random.randint(2, size=(500,))  # Binary labels for testing

# Evaluate simple model
simple_loss, simple_accuracy = simple_model.evaluate(X_test, y_test)
print("Simple Model - Test Loss:", simple_loss)
print("Simple Model - Test Accuracy:", simple_accuracy)

# Evaluate deep model
deep_loss, deep_accuracy = deep_model.evaluate(X_test, y_test)
print("Deep Model - Test Loss:", deep_loss)
print("Deep Model - Test Accuracy:", deep_accuracy)
```

```
16/16 [==============================] - 1s 5ms/step - loss: 0.7037 - accuracy: 0.4880
Simple Model - Test Loss: 0.703722357749939
Simple Model - Test Accuracy: 0.4880000054836273
16/16 [==============================] - 0s 9ms/step - loss: 0.6906 - accuracy: 0.5320
Deep Model - Test Loss: 0.6905701160430908
Deep Model - Test Accuracy: 0.5320000052452087
```

```python
from sklearn.metrics import classification_report

# Predictions
simple_predictions = (simple_model.predict(X_test) > 0.5).astype("int32")
deep_predictions = (deep_model.predict(X_test) > 0.5).astype("int32")
```