NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

MACHINE LEARNING

CACSC17

Mr. Gaurav Singal

Sneha Gupta

2021UCA1859

ASSIGNMENT 1

Excercise 1

Slice elements from index 1 to index 5 from the following array: [1,2,3,4,5,6,7]

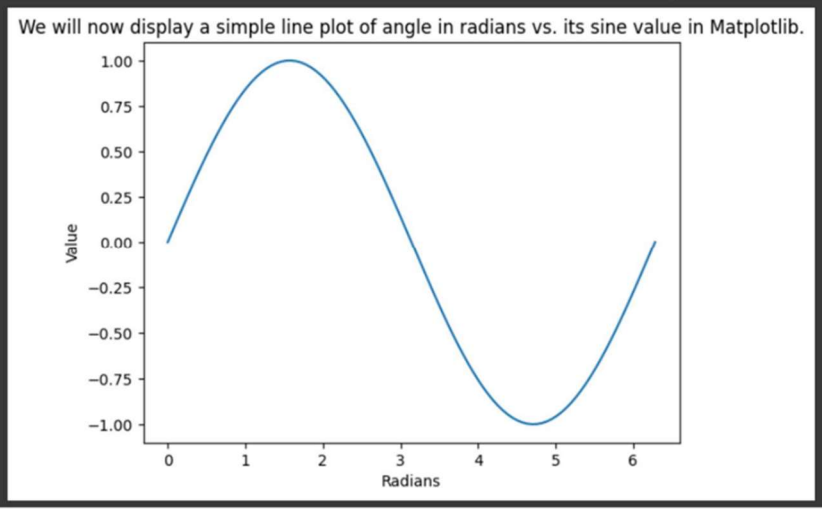
```
# Slice elements from index 1 to index 5 from the following array: [1,2,3,4,5,6,7]
import numpy as np
arr = np.array([1,2,3,4,5,6,7])
sarr = arr[1:6]
sarr
```

```
array([2, 3, 4, 5, 6])
```

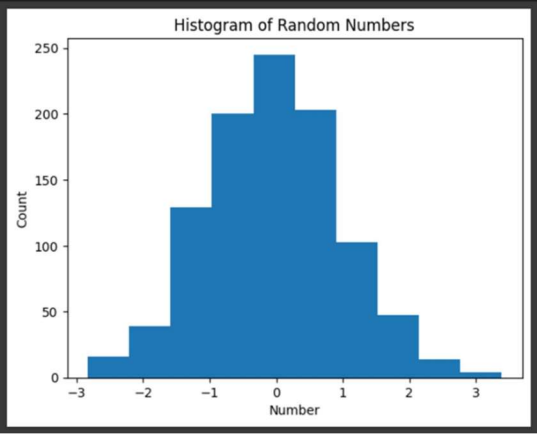
Excercise 2

Draw the sine wave using matplotlib

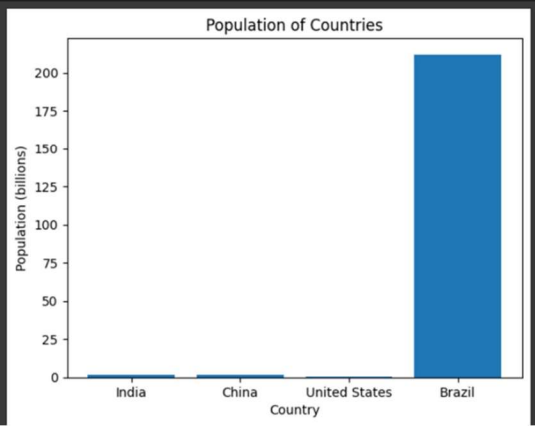
```
#draw the sine wave using matplotlib
import matplotlib.pyplot as plt
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
plt.plot(x, y)
plt.title("We will now display a simple line plot of angle in radians vs. its sine value in Matplotlib.")
plt.xlabel("Radians")
plt.ylabel("Value")
plt.show()
```



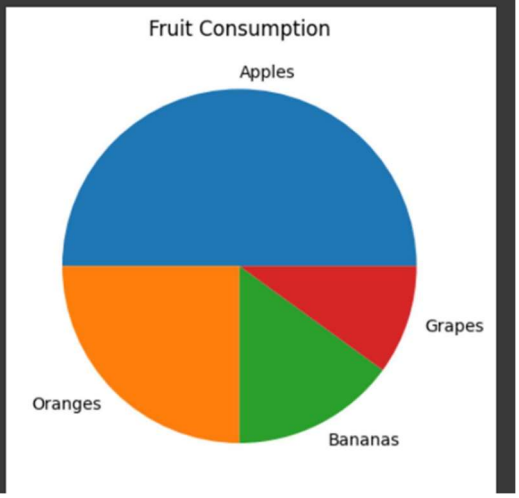
```
[6] data = np.random.randn(1000)
    plt.hist(data)
    plt.title("Histogram of Random Numbers")
    plt.xlabel("Number")
    plt.ylabel("Count")
    plt.show()
```



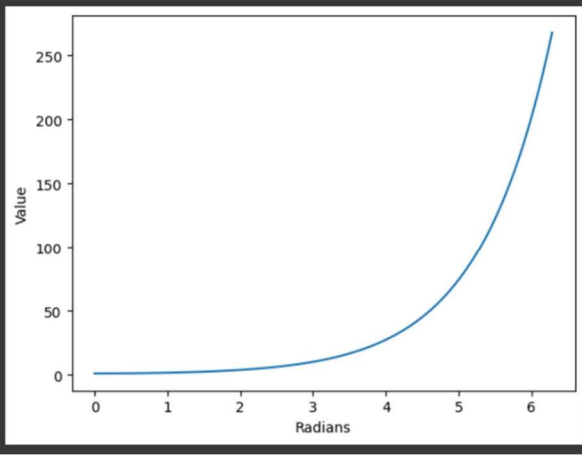

```
[8] countries = ["India", "China", "United States", "Brazil"]
    populations = [1.385, 1.444, 0.332, 212]
    plt.bar(countries, populations)
    plt.title("Population of Countries")
    plt.xlabel("Country")
    plt.ylabel("Population (billions)")
    plt.show()
```




```
[10] fruits = ["Apples", "Oranges", "Bananas", "Grapes"]
     percentages = [50, 25, 15, 10]
     plt.pie(percentages, labels=fruits)
     plt.title("Fruit Consumption")
     plt.show()
```




```
x = np.linspace(0, 2 * np.pi, 100)
y = np.cosh(x)
plt.plot(x, y)
plt.xlabel("Radians")
plt.ylabel("Value")
plt.show()
```



ASSIGNMENT: LAB 2 –

```
pip install pandas
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in c:\users\lenovo\appdata\roaming\python\python311\site-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from pandas) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from pandas) (1.25.2)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
print(pd.__version__)
```

```
2.0.3
```

```
import pandas as pd
df = pd.read_csv('Bias_correction_ucl.csv')
print(df)
```

```
      station        Date  Present_Tmax  Present_Tmin  LDAPS_RHmin  \
0         1.0  2013-06-30          28.7          21.4    58.255688
1         2.0  2013-06-30          31.9          21.6    52.263397
2         3.0  2013-06-30          31.6          23.3    48.690479
3         4.0  2013-06-30          32.0          23.4    58.239788
4         5.0  2013-06-30          31.4          21.9    56.174095
...       ...         ...           ...           ...          ...
```

```
main_data = pd.read_csv('Bias_correction_ucl.csv')
main_data.sample(10)
```

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	...	LDAPS_PPT2	LDAP
355	6.0	2013-07-14	26.5	23.5	65.549965	94.181763	30.221667	23.520221	6.908038	30.766633	...	0.462615	0.
6638	14.0	2017-07-17	28.8	26.0	59.148586	84.882111	31.165211	24.218230	6.149441	15.734958	...	0.000000	0.
2163	14.0	2014-07-24	24.5	19.0	65.626198	92.932800	31.979214	23.568098	7.939121	95.618308	...	0.000000	2.
2452	3.0	2014-08-05	28.7	24.7	70.530167	83.589104	27.932137	24.713666	4.674527	18.714207	...	0.004749	0.

```
main_data= main_data.drop(columns= ['station','Next_Tmin', 'Date','LDAPS_PPT2', 'LDAPS_PPT3', 'LDAPS_PPT4', 'lat',  'lon',  'LDAPS_CC2','LDAPS_CC3','LDAP
main_data
```

	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	LDAPS_CC1	DEM	Slope	rad
0	28.7	21.4	58.255688	91.116364	28.074101	23.006936	6.818887	69.451805	0.233947	212.3350	2.785000	5992.8
1	31.9	21.6	52.263397	90.604721	29.850689	24.035009	5.691890	51.937448	0.225508	44.7624	0.514100	5869.3
2	31.6	23.3	48.690479	83.973587	30.091292	24.565633	6.138224	20.573050	0.209344	33.3068	0.266100	5863.5
3	32.0	23.4	58.239788	96.483688	29.704629	23.326177	5.650050	65.727144	0.216372	45.7160	2.534800	5856.9

```
Y= main_data['LDAPS_Tmin_lapse']
Y
```

```
0       23.006936
1       24.035009
2       24.565633
3       23.326177
4       23.486480
          ...
7747    18.775678
7748    18.733519
7749    18.522965
7750    14.272646
7751    29.619342
Name: LDAPS_Tmin_lapse, Length: 7752, dtype: float64
```

```
main_data.rename(columns = {'Present_Tmax': 'prMax',
                  'Present_Tmin': 'prMin',
                   'LDAPS_RHmin': 'minRelativeH',
                   'LDAPS_RHmax': 'maxRelativeH',
                   'LDAPS_Tmax_lapse': 'maxLapseR',
                  'LDAPS_Tmin_lapse': 'minLapseR',
                   'LDAPS_WS': 'avgWS',
                   'LDAPS_LH': 'avgLHF',
                   'LDAPS_CC1': 'CC'}, inplace=True)
```

```
main_data.columns
```

```
Index(['prMax', 'prMin', 'minRelativeH', 'maxRelativeH', 'maxLapseR',
       'minLapseR', 'avgWS', 'avgLHF', 'CC', 'DEM', 'Slope',
       'Solar radiation'],
      dtype='object')
```

```
main_data.info
```

```
<bound method DataFrame.info of       prMax  prMin  minRelativeH  maxRelativeH  maxLapseR  minLapseR  \
0      28.7   21.4     58.255688     91.116364  28.074101  23.006936
1      31.9   21.6     52.263397     90.604721  29.850689  24.035009
2      31.6   23.3     48.690479     83.973587  30.091292  24.565633
3      32.0   23.4     58.239788     96.483688  29.704629  23.326177
4      31.4   21.9     56.174095     90.155128  29.113934  23.486480
...     ...    ...           ...           ...        ...        ...
7747   23.3   17.1     26.741310     78.869858  26.352081  18.775678
```

```
inputList = ['prMax', 'prMin', 'minRelativeH', 'maxRelativeH','maxLapseR', 'minLapseR', 'avgWS', 'avgLHF', 'CC', 'DEM', 'Slope', 'Solar radiation']
input = main_data[inputList]
```

```
input.isnull().sum()
```

```
prMax               70
prMin               70
minRelativeH        75
maxRelativeH        75
maxLapseR           75
minLapseR           75
avgWS               75
avgLHF              75
CC                  75
DEM                  0
Slope                0
Solar radiation      0
dtype: int64
```

```
output = main_data[['minRelativeH']]
```

```
output = output.fillna(output.median())
output.shape
output.duplicated().sum()
```

```
80
```

```
main_data.isnull()
main_data = main_data.fillna(main_data.median())
main_data
```

	prMax	prMin	minRelativeH	maxRelativeH	maxLapseR	minLapseR	avgWS	avgLHF	CC	DEM	Slope	Solar radiation
0	28.7	21.4	58.255688	91.116364	28.074101	23.006936	6.818887	69.451805	0.233947	212.3350	2.785000	5992.895996
1	31.9	21.6	52.263397	90.604721	29.850689	24.035009	5.691890	51.937448	0.225508	44.7624	0.514100	5869.312500
2	31.6	23.3	48.690479	83.973587	30.091292	24.565633	6.138224	20.573050	0.209344	33.3068	0.266100	5863.555664
3	32.0	23.4	58.239788	96.483688	29.704629	23.326177	5.650050	65.727144	0.216372	45.7160	2.534800	5856.964844
4	31.4	21.9	56.174095	90.155128	29.113934	23.486480	5.735004	107.965535	0.151407	35.0380	0.505500	5859.552246

```
main_data.isnull().sum()
```

```
prMax              0
prMin              0
minRelativeH       0
maxRelativeH       0
maxLapseR          0
minLapseR          0
avgWS              0
avgLHF             0
CC                 0
DEM                0
Slope              0
Solar radiation    0
dtype: int64
```

```
pip install -U scikit-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting scikit-learn
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
x_train, x_test, y_train, y_test= train_test_split(input,output, test_size=0.3, random_state=0)
```

```
x_train
```

	prMax	prMin	minRelativeH	maxRelativeH	maxLapseR	minLapseR	avgWS	avgLHF	CC	DEM	Slope	Solar radiation
6962	32.4	25.1	48.446579	81.101311	33.173575	24.865377	5.352939	62.747479	0.181988	59.8324	2.6865	5434.702148
1031	29.5	26.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12.3700	0.0985	5140.230957
249	26.8	23.9	59.100754	94.579933	30.486747	24.378562	8.567573	29.681216	0.913904	19.5844	0.2713	5785.897949

```
x_test
```

	prMax	prMin	minRelativeH	maxRelativeH	maxLapseR	minLapseR	avgWS	avgLHF	CC	DEM	Slope	Solar radiation
4271	32.1	24.5	63.083488	83.169289	27.043221	24.184606	4.613474	69.023742	0.319488	21.9668	0.1332	4987.023438
971	32.7	24.9	57.288948	89.839920	33.530312	27.405979	8.275806	72.596921	0.299611	21.9668	0.1332	5228.346680
7541	31.7	21.3	81.746681	90.070435	29.039634	23.828240	6.686319	38.842143	0.324799	53.4712	0.6970	4765.267578
4806	26.4	20.5	68.170364	93.259880	25.199747	20.782339	3.917524	63.849919	0.567421	12.3700	0.0985	5799.268066
2048	30.0	25.0	56.149647	91.013557	30.304948	24.716520	6.070194	43.157278	0.139779	17.2956	0.2223	5646.496582

```
y_test
```

	minRelativeH
4271	63.083488
971	57.288948
7541	81.746681
4806	68.170364
2048	56.149647
...	...
4162	31.287195

```
model= LinearRegression()
```

```
import numpy as np
```

```
median_value_y = np.nanmedian(y_train)
y_train_filled = np.nan_to_num(y_train, nan=median_value_y)
```

```
median_value_x = np.nanmedian(x_train)
x_train_filled = np.nan_to_num(x_train, nan=median_value_x)
```

```
x_train_filled = np.array(x_train_filled)
y_train_filled = np.array(y_train_filled)
```

```
model.fit(x_train_filled, y_train_filled)
```

```
▾ LinearRegression
LinearRegression()
```

```
x=model.intercept_
```

```
x
```

```
array([0.29051409])
```

```
y=model.coef_
y
```

```
array([[ 2.15968222e-02, -1.66186976e-02,  9.96584189e-01,
        -7.49305604e-03,  2.52175238e-02, -2.02276710e-02,
        -7.62730126e-03,  6.16989769e-04,  9.49759517e-01,
         2.62210633e-04,  1.42237822e-02, -6.21608689e-05]])
```

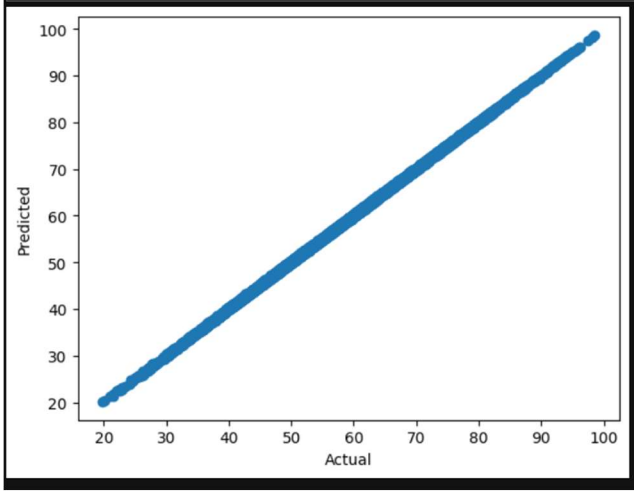
```
y_pred_train = model.predict(x_train_filled)
y_pred_train
```

```
array([[48.46691242],
       [54.8268393 ],
       [59.40827682],
       ...,
       [49.27003592],
       [51.05182706],
       [60.06581354]])
```

```
pip install matplotlib
```

```
import matplotlib.pyplot as plt
```

```
plt.scatter(y_train_filled, y_pred_train)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```




```
from sklearn.metrics import r2_score
```

```
r2_score(y_train_filled, y_pred_train)
```

```
0.999853798463216
```

```
x_test.isnull().sum()
```

```
prMax              19
prMin              19
minRelativeH       21
maxRelativeH       21
maxLapseR          21
minLapseR          21
avgWS              21
avgLHF             21
CC                 21
DEM                 0
Slope               0
Solar radiation     0
dtype: int64
```

```
x_test.isnull()
x_test = x_test.fillna(x_test.median())
x_test
```

	prMax	prMin	minRelativeH	maxRelativeH	maxLapseR	minLapseR	avgWS	avgLHF	CC	DEM	Slope	Solar radiation
4271	32.1	24.5	63.083488	83.169289	27.043221	24.184606	4.613474	69.023742	0.319488	21.9668	0.1332	4987.023438
971	32.7	24.9	57.288948	89.839920	33.530312	27.405979	8.275806	72.596921	0.299611	21.9668	0.1332	5228.346680

```
x_test.isnull().sum()
```

```
prMax              0
prMin              0
minRelativeH       0
maxRelativeH       0
maxLapseR          0
minLapseR          0
avgWS              0
avgLHF             0
CC                 0
DEM                0
Slope              0
Solar radiation    0
dtype: int64
```

```
model.fit(x_train_filled, y_train_filled)
```

▾ LinearRegression

LinearRegression()

```
import warnings
```

```
with warnings.catch_warnings():
    # Suppress the warning
    warnings.simplefilter("ignore")
    model.fit(x_train_filled, y_train_filled)
```

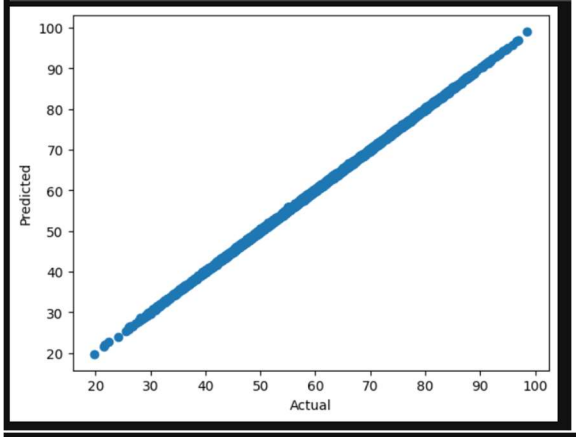
```
x_test.isnull().sum()
```

```
prMax              0
prMin              0
minRelativeH       0
maxRelativeH       0
maxLapseR          0
minLapseR          0
avgWS              0
avgLHF             0
CC                 0
DEM                0
Slope              0
Solar radiation    0
dtype: int64
```

```
x_test
```

	prMax	prMin	minRelativeH	maxRelativeH	maxLapseR	minLapseR	avgWS	avgLHF	CC	DEM	Slope	Solar radiation
4271	32.1	24.5	63.083488	83.169289	27.043221	24.184606	4.613474	69.023742	0.319488	21.9668	0.1332	4987.023438
971	32.7	24.9	57.288948	89.839920	33.530312	27.405979	8.275806	72.596921	0.299611	21.9668	0.1332	5228.346680
7541	31.7	21.3	81.746681	90.070435	29.039634	23.828240	6.686319	38.842143	0.324799	53.4712	0.6970	4765.267578

```
import matplotlib.pyplot as plt
plt.scatter(y_test, y_pred_tes)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```




```
median_value = np.nanmedian(y_test)
y_test_filled = np.nan_to_num(y_test, nan=median_value)
```

```
r2_score(y_test_filled, y_pred_tes)
```

```
0.9998365882695753
```

ASSIGNMENT 3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as pltV
pip install seaborn
import seaborn as sns
import sklearn
import math
```

```
titanic = pd.read_csv("titanic.csv")
titanic.sample(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
218	219	1	1	Bazzani, Miss. Albina	female	32.0	0	0	11813	76.2917	D15	C
672	673	0	2	Mitchell, Mr. Henry Michael	male	70.0	0	0	C.A. 24580	10.5000	NaN	S
496	497	1	1	Eustis, Miss. Elizabeth Mussey	female	54.0	1	0	36947	78.2667	D20	C
633	634	0	1	Parr, Mr. William Henry Marsh	male	NaN	0	0	112052	0.0000	NaN	S

```
print ("Number Of Passengers:" +str(len(titanic.index)))
```

```
Number Of Passengers:891
```

```
titanic.columns
```

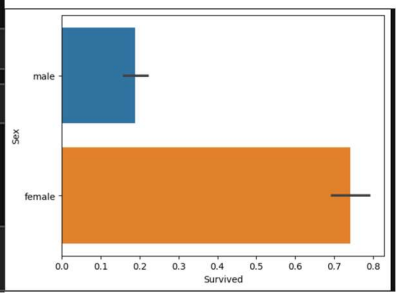
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
titanic=titanic.drop(columns=['SibSp','Parch','Name'])
```

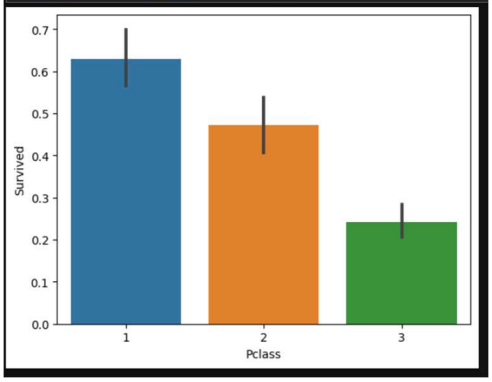
```
titanic.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'Ticket', 'Fare',
       'Cabin', 'Embarked'],
      dtype='object')
```

```
# Sex based survival of people
sns.barplot(x=titanic.Survived, y=titanic.Sex);
```



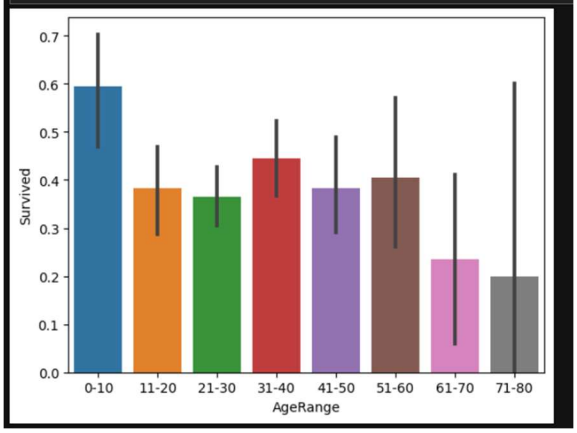

```
# Passenger class based survival of people
sns.barplot(y=titanic.Survived, x=titanic.Pclass);
```



```python
# Assuming you have a DataFrame named 'titanic' with 'Age' and 'Survived' columns

# Define the age ranges
age_ranges = ["0-10", "11-20", "21-30", "31-40", "41-50", "51-60", "61-70", "71-80"]
```

```python
titanic['AgeRange'] = pd.cut(titanic['Age'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80], labels=age_ranges)
```

```python
sns.barplot(x='AgeRange', y='Survived', data=titanic, order=age_ranges);
```




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

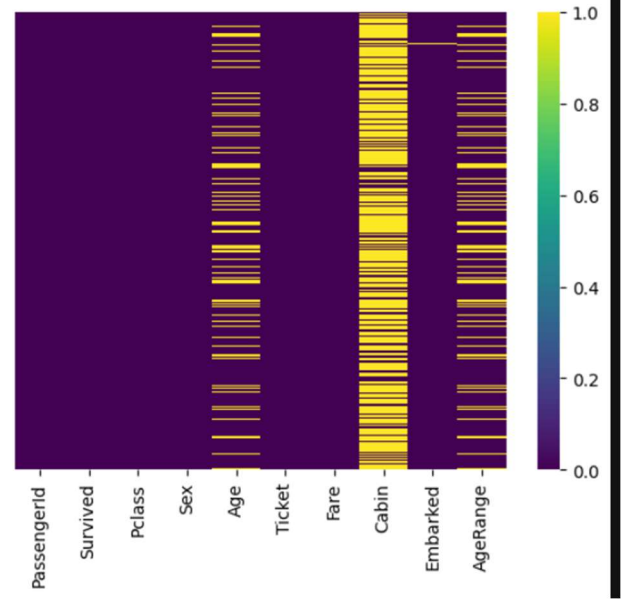
```
PassengerId      0
Survived         0
Pclass           0
Sex              0
Age            177
Ticket           0
Fare             0
Cabin          687
Embarked         2
AgeRange       177
dtype: int64
```

```python
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 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   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   Ticket       891 non-null    object
 6   Fare         891 non-null    float64
 7   Cabin        204 non-null    object
 8   Embarked     889 non-null    object
 9   AgeRange     714 non-null    category
dtypes: category(1), float64(2), int64(3), object(4)
memory usage: 64.0+ KB
```

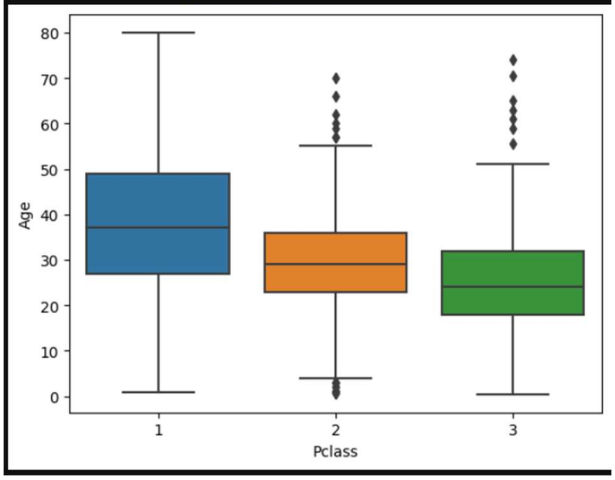
```python
sns.heatmap(titanic.isnull(), yticklabels=False, cmap="viridis")
```

```
<Axes: >
```



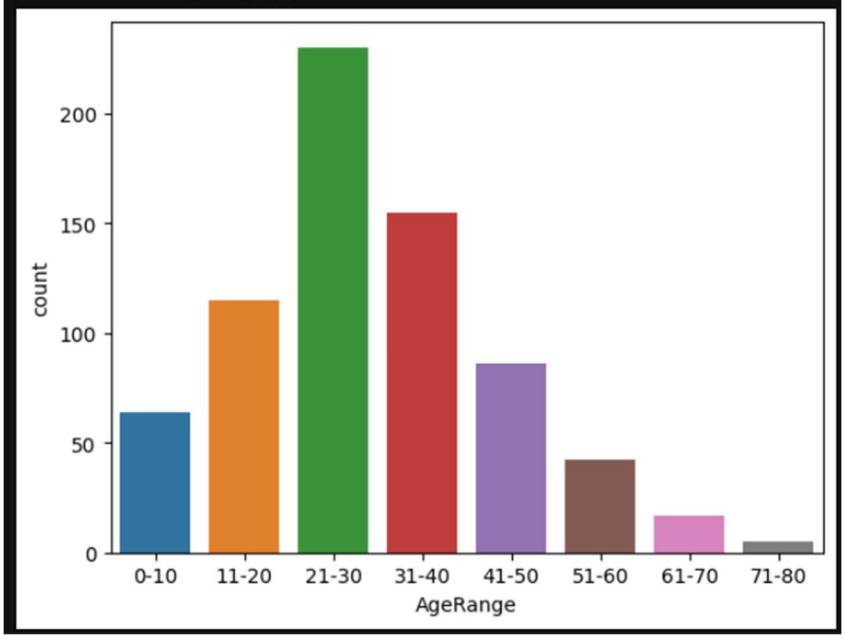

```python
sns.boxplot(x="Pclass", y="Age", data=titanic)
```

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



```
sns.countplot(x='AgeRange', data=titanic)
```

```
<Axes: xlabel='AgeRange', ylabel='count'>
```

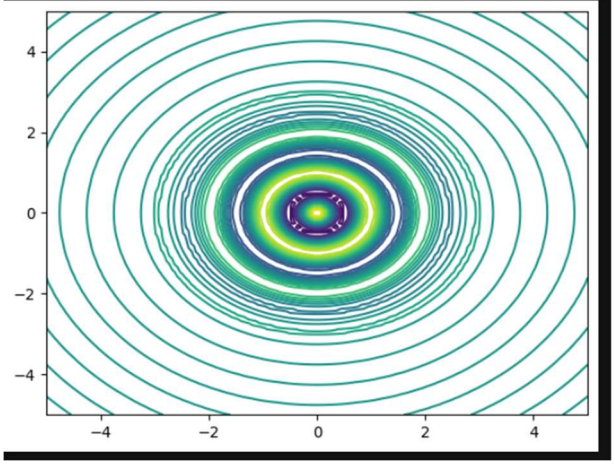



```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
```

```
Z = np.exp(-0.5 * (X**2 + Y**2) / 2) * np.cos(2 * np.pi * np.sqrt(X**2 + Y**2))
```

```
import numpy as py
```

```
# Create a contour plot using matplotlib.pyplot.contour
contour_levels = 20
contour_plot = pltV.contour(X, Y, Z, levels=contour_levels, cmap='viridis')
```




```
sex= pd.get_dummies(titanic['Sex'],drop_first=True)
sex.head(5)
```

	male
0	True
1	False
2	False
3	False
4	True

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 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   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   Ticket       891 non-null    object
 6   Fare         891 non-null    float64
 7   Cabin        204 non-null    object
 8   Embarked     889 non-null    object
 9   AgeRange     714 non-null    category
dtypes: category(1), float64(2), int64(3), object(4)
memory usage: 64.0+ KB
```

```
titanic.drop(['AgeRange', 'Ticket', 'Cabin'], axis = 1, inplace = True)
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 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   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   Fare         891 non-null    float64
 6   Embarked     889 non-null    object
dtypes: float64(2), int64(3), object(2)
memory usage: 48.9+ KB
```

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
titanic['Sex'] = LE.fit_transform(titanic['Sex'])
titanic['Embarked'] = LE.fit_transform(titanic['Embarked'])
```

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 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   Sex          891 non-null    int64
 4   Age          714 non-null    float64
 5   Fare         891 non-null    float64
 6   Embarked     891 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
```

```
X = titanic.drop("Survived", axis=1)
y = titanic["Survived"]
```

```
X.sample(10)
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked
22	23	3	0	15.0	8.0292	1
265	266	2	1	36.0	10.5000	2
177	178	1	0	50.0	28.7125	0
839	840	1	1	NaN	29.7000	0
398	399	2	1	23.0	10.5000	2
308	309	2	1	30.0	24.0000	0
153	154	3	1	40.5	14.5000	2
809	810	1	0	33.0	53.1000	2
300	301	3	0	NaN	7.7500	1
626	627	2	1	57.0	12.3500	1

```
y.sample(10)
```

```
711    0
364    0
559    1
147    0
519    0
378    0
325    1
856    1
61     1
468    0
Name: Survived, dtype: int64
```

```
X_scaled = X.columns.astype(str, int)
```

```
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=1)
X.columns = X.columns.astype(str)
```

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

```
model=LogisticRegression()
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Pclass       891 non-null    int64
 2   Sex          891 non-null    int64
 3   Age          714 non-null    float64
 4   Fare         891 non-null    float64
 5   Embarked     891 non-null    int64
dtypes: float64(2), int64(4)
memory usage: 41.9 KB
```

```
Pcl= pd.get_dummies(titanic['Pclass'],drop_first=True)
Pcl.head(5)
```

	2	3
0	False	True
1	False	False
2	False	True
3	False	False
4	False	True

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Pclass       891 non-null    int64
 2   Sex          891 non-null    int64
 3   Age          714 non-null    float64
 4   Fare         891 non-null    float64
 5   Embarked     891 non-null    int64
dtypes: float64(2), int64(4)
memory usage: 41.9 KB
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 623 entries, 114 to 37
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  623 non-null    int64
 1   Pclass       623 non-null    int64
 2   Sex          623 non-null    int64
 3   Age          496 non-null    float64
 4   Fare         623 non-null    float64
 5   Embarked     623 non-null    int64
dtypes: float64(2), int64(4)
memory usage: 34.1 KB
```

```
median_age = X_train['Age'].median()
X_train['Age'] = X_train['Age'].fillna(median_age)
```

```
X_train.isnull().sum()
```

```
PassengerId    0
Pclass         0
Sex            0
Age            0
Fare           0
Embarked       0
dtype: int64
```

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

```
▾ LogisticRegression
LogisticRegression()
```

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

```
▾ LogisticRegression
LogisticRegression()
```

```
predictions = model.predict(X_train)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy = accuracy_score(y_train, predictions)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.81
```

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_train, predictions))
```

```
              precision    recall  f1-score   support

           0       0.84      0.87      0.85       396
           1       0.75      0.71      0.73       227

    accuracy                           0.81       623
   macro avg       0.80      0.79      0.79       623
weighted avg       0.81      0.81      0.81       623
```

ASSIGNMENT 4

```
import pandas as pd
```

```
import random
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
import seaborn as sns
```

```
from sklearn.preprocessing import LabelEncoder
```

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

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

```
from sklearn. linear_model import LinearRegression
```

```
import matplotlib.pyplot as plt
```

```
md= pd.read_csv('medical.csv')
md
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200

```
md.sample(10)
```

	age	sex	bmi	children	smoker	region	charges
833	58	male	34.39	0	no	northwest	11743.9341
351	50	female	25.60	0	no	southwest	8932.0840
766	47	male	32.30	1	no	southwest	8062.7640
837	56	female	28.31	0	no	northeast	11657.7189
399	18	female	38.17	0	no	southeast	1631.6683
1	18	male	33.77	1	no	southeast	1725.5523

```
print ("Number Of applicants:" +str(len(md.index)))
```

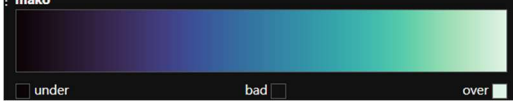
```
Number Of applicants:1338
```

```
md.columns
```

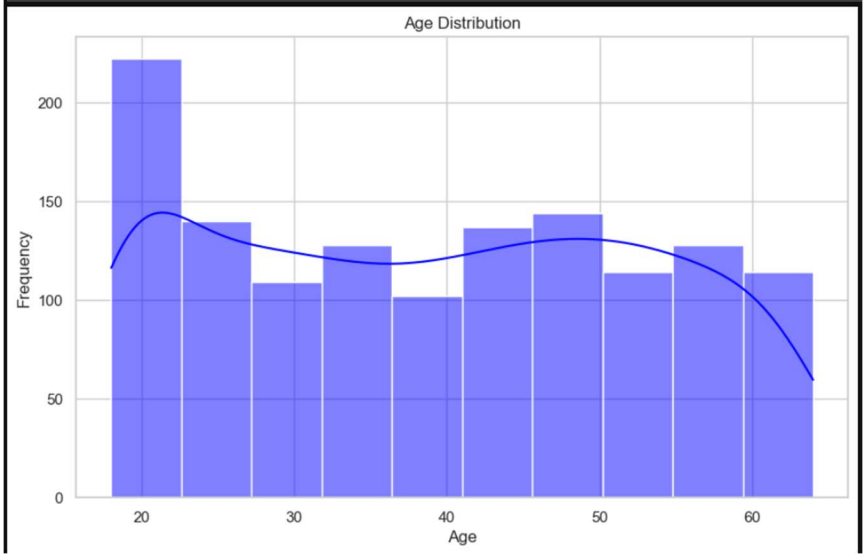
```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

```
sns.color_palette("mako", as_cmap=True)
```

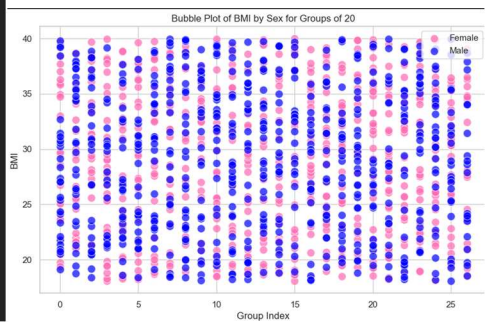
```
mako
```



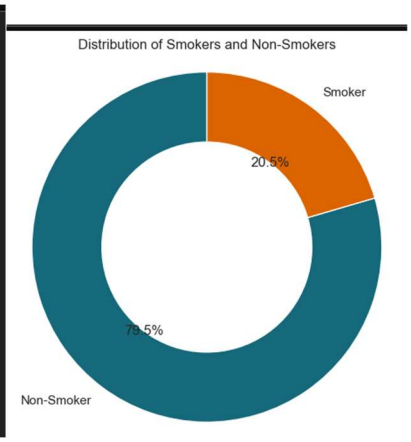

```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))
sns.histplot(data=md, x='age', bins=10, kde=True, color='blue')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
```



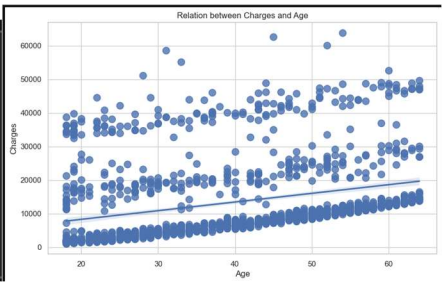
```
sns.color_palette("mako", as_cmap=True)
# Create sample data
np.random.seed(42)
data = {
    'sex': np.random.choice(['female', 'male'], size=1338),
    'bmi': np.random.uniform(18, 40, size=1338)
}

# Create a DataFrame
df = pd.DataFrame(data)

# Calculate group indices
group_size = 50
df['group_index'] = df.index // group_size

# Set the plotting style and size
sns.set(style='whitegrid')
plt.figure(figsize=(10, 6))

# Create separate scatter plots for females and males
female_df = df[df['sex'] == 'female']
male_df = df[df['sex'] == 'male']

sns.scatterplot(x='group_index', y='bmi', data=female_df, color='hotpink', s=100, alpha=0.7, label='Female')
sns.scatterplot(x='group_index', y='bmi', data=male_df, color='blue', s=100, alpha=0.7, label='Male')

# Adding labels and title
plt.xlabel('Group Index')
plt.ylabel('BMI')
plt.title('Bubble Plot of BMI by Sex for Groups of 20')

# Display the legend
plt.legend()

# Display the graph
plt.show()
```



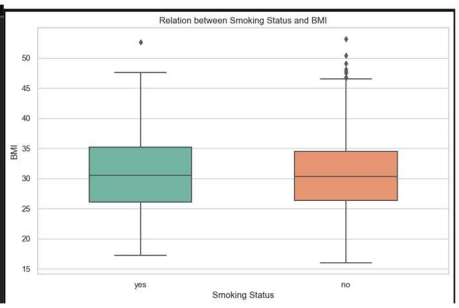

```
# Count the occurrences of smokers and non-smokers
smoker_counts = md['smoker'].value_counts()

# Create a figure and axis
fig, ax = plt.subplots(figsize=(6, 6))

# Colors for the sections
colors = ['#16697A', '#DB6400']

# Create a donut plot
wedges, texts, autotexts = ax.pie(smoker_counts, labels=['Non-Smoker', 'Smoker'], autopct='%1.1f%%',
                                  startangle=90, colors=colors, wedgeprops=dict(width=0.4, edgecolor='w'))

# Adding a circle in the center to create a donut appearance
centre_circle = plt.Circle((0,0),0.30,fc='white')
fig.gca().add_artist(centre_circle)

# Equal aspect ratio ensures that pie is drawn as a circle.
ax.axis('equal')

# Adding title
plt.title('Distribution of Smokers and Non-Smokers')

# Display the plot
plt.show()
```



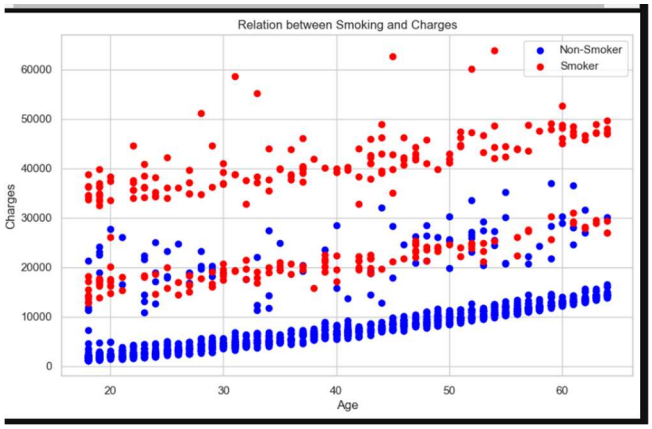

```
plt.figure(figsize=(10, 6))
sns.regplot(x='age', y='charges', data=md, scatter_kws={'s': 100})
plt.xlabel('Age')
plt.ylabel('Charges')
plt.title('Relation between Charges and Age')
plt.grid(True)
plt.show()
```




```
# Create a box plot or violin plot
plt.figure(figsize=(10, 6))
sns.boxplot(x='smoker', y='bmi', data=md, palette='Set2', width=0.5)

# Add labels and title
plt.xlabel('Smoking Status')
plt.ylabel('BMI')
plt.title('Relation between Smoking Status and BMI')

plt.show()
```



```
# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(md[md['smoker'] == 'no']['age'], md[md['smoker'] == 'no']['charges'], label='Non-Smoker', color='bl
plt.scatter(md[md['smoker'] == 'yes']['age'], md[md['smoker'] == 'yes']['charges'], label='Smoker', color='red'

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Charges')
plt.title('Relation between Smoking and Charges')
plt.legend()

plt.show()
```




```
md.isnull().sum()
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

```
md.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
```

```
md['sex'] = LE.fit_transform(md.sex)
md.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   int32
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int32(1), int64(2), object(2)
memory usage: 68.1+ KB
```

```
md['smoker'] = LE.fit_transform(md.smoker)
md['region'] = LE.fit_transform(md.region)
```

```
md.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   int32
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   int32
 5   region    1338 non-null   int32
 6   charges   1338 non-null   float64
dtypes: float64(2), int32(3), int64(2)
memory usage: 57.6 KB
```

```
# Display unique values in the 'region' column
print(md['region'].unique())

# Map integers to new integer labels
int_labels = {0: 0, 1: 1, 2: 2, 3: 3}
md['region_int'] = md['region'].map(int_labels)

# Display the updated DataFrame
print(md.head())
```

```
[3, 2, 1, 0]
Categories (4, int32): [0, 1, 2, 3]
   age  sex     bmi  children  smoker region      charges region_int
0   19    0  27.900         0       1      3  16884.92400          3
1   18    1  33.770         1       0      2   1725.55230          2
2   28    1  33.000         3       0      2   4449.46200          2
3   33    1  22.705         0       0      1  21984.47061          1
4   32    1  28.880         0       0      1   3866.85520          1
```

S

```
print(md['region'].head())
print(md['region'].unique())
```

```
0    3
1    2
2    2
3    1
4    1
Name: region, dtype: category
Categories (4, int32): [0, 1, 2, 3]
[3, 2, 1, 0]
Categories (4, int32): [0, 1, 2, 3]
```

```
md.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1338 non-null   int64
 1   sex         1338 non-null   int32
 2   bmi         1338 non-null   float64
 3   children    1338 non-null   int64
 4   smoker      1338 non-null   int32
 5   region      1338 non-null   category
 6   charges     1338 non-null   float64
 7   region_int  1338 non-null   category
dtypes: category(2), float64(2), int32(2), int64(2)
memory usage: 55.3 KB
```

```
print(md['region'].unique())

# Convert the 'region' column to integers
md['region_int'] = md['region'].cat.codes

# Display the updated DataFrame
print(md.head())
```

```
[3, 2, 1, 0]
Categories (4, int32): [0, 1, 2, 3]
   age  sex     bmi  children  smoker region      charges  region_int
0   19    0  27.900         0       1      3  16884.92400           3
1   18    1  33.770         1       0      2   1725.55230           2
2   28    1  33.000         3       0      2   4449.46200           2
3   33    1  22.705         0       0      1  21984.47061           1
4   32    1  28.880         0       0      1   3866.85520           1
```

```
md.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1338 non-null   int64
 1   sex         1338 non-null   int32
 2   bmi         1338 non-null   float64
 3   children    1338 non-null   int64
 4   smoker      1338 non-null   int32
 5   region      1338 non-null   category
 6   charges     1338 non-null   float64
 7   region_int  1338 non-null   int8
dtypes: category(1), float64(2), int32(2), int64(2), int8(1)
memory usage: 55.2 KB
```

```
print(md.columns)

# Drop the 'region' column
column_to_drop = 'region'
md = md.drop(column_to_drop, axis=1)

# Display the updated DataFrame
print(md.head())
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges',
       'region_int'],
      dtype='object')
   age  sex     bmi  children  smoker      charges  region_int
0   19    0  27.900         0       1  16884.92400           3
1   18    1  33.770         1       0   1725.55230           2
2   28    1  33.000         3       0   4449.46200           2
3   33    1  22.705         0       0  21984.47061           1
4   32    1  28.880         0       0   3866.85520           1
```

```
md.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1338 non-null   int64
 1   sex         1338 non-null   int32
 2   bmi         1338 non-null   float64
 3   children    1338 non-null   int64
 4   smoker      1338 non-null   int32
 5   charges     1338 non-null   float64
 6   region_int  1338 non-null   int8
dtypes: float64(2), int32(2), int64(2), int8(1)
memory usage: 53.7 KB
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

```
from sklearn.metrics import mean_squared_error

# Train a linear regression model
model1 = LinearRegression()
model1.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model1.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

S

```
Mean Squared Error: 33635210.431178406
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Assuming you have performed EDA and prepared X and y
X = md.drop('smoker', axis=1)
y = md['smoker']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy and confusion matrix
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.9402985074626866
Confusion Matrix:
[[207   7]
 [  9  45]]
```

```
from sklearn.linear_model import LinearRegression
model_regression = LinearRegression()
model_regression.fit(X_train, y_train)
```

```
LinearRegression()
```

```
from sklearn.metrics import mean_squared_error, r2_score
y_pred_regression = model_regression.predict(X_test)
mse = mean_squared_error(y_test, y_pred_regression)
r2 = r2_score(y_test, y_pred_regression)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Mean Squared Error: 0.04153118534805607
R-squared: 0.7418712481447924
```

```
from sklearn.linear_model import LogisticRegression
model_classification = LogisticRegression()
model_classification.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
y_pred_classification = model_classification.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_classification)
classification_rep = classification_report(y_test, y_pred_classification)
conf_matrix = confusion_matrix(y_test, y_pred_classification)
print(f'Accuracy: {accuracy*100:.2f}' + ' %')
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 94.03 %
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       214
           1       0.87      0.83      0.85        54

    accuracy                           0.94       268
   macro avg       0.91      0.90      0.91       268
weighted avg       0.94      0.94      0.94       268

Confusion Matrix:
[[207   7]
 [  9  45]]
```

```
from sklearn.linear_model import LogisticRegression

# Assuming you have prepared X_train and y_train for the multi-class classification s
model_multiclass = LogisticRegression(multi_class='ovr')  # 'ovr' or 'multinomial'
model_multiclass.fit(X_train, y_train)  # Use y_train for multi-class, not y_train_mu
```

```
LogisticRegression(multi_class='ovr')
```

ASSIGNMENT 5

OBJECTIVE:: Understand the program to implement the performance parameter (Precision, Recall, Accuracy, log loss, ROC, AUC And Confusion Matrix) of two model for binary, classifications. Dataset overview and preprocessing loading the Heart Disease dataset and performing initial data exploration. The dataset contains 14 features, and the target variable 'target' indicates the presence (1) or absence (0) of heart disease.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```
✓ 1.9s

```
heart_data = pd.read_csv('heart.csv')
print(heart_data.head())
print(heart_data.tail())
print(heart_data.shape)
print(heart_data.info())
print(heart_data.isnull().sum())
```

Splitting the Data into Training data & Test Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

(303, 13) (242, 13) (61, 13)

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
...
ca        0
thal      0
target    0
```

```
heart_data['target'].value_counts()
X = heart_data.drop(columns='target', axis=1)
Y = heart_data['target']
X
```
✓ 0.0s

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2

```
model = LogisticRegression(max_iter=1000)
```
✓ 0.0s

```
# training the LogisticRegression model with Training data
model.fit(X_train, Y_train)
```
✓ 0.1s

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
precision_train = precision_score(Y_train, X_train_prediction)
print('Training data Precision =', precision_train)
precision_test = precision_score(Y_test, X_test_prediction)
print('Test data Precision =', precision_test)
```
✓ 0.0s

```
Training data Precision = 0.8299319727891157
Test data Precision = 0.8181818181818182
```

```
recall_train = recall_score(Y_train, X_train_prediction)
print('Training data Recall =', recall_train)
recall_test = recall_score(Y_test, X_test_prediction)
print('Test data Recall =', recall_test)
```
✓ 0.0s

```
Training data Recall = 0.9242424242424242
Test data Recall = 0.8181818181818182
```

```
from sklearn.metrics import log_loss
# loss on training data
X_train_prediction = model.predict(X_train)
training_data_loss = log_loss(Y_train, X_train_prediction, eps=1e-15)
print(training_data_loss)
```
✓ 0.0s

```
4.995360180816648
```

```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy on Training data : ', round(training_data_accuracy*100, 2), '%')
```

✓ 0.0s

```
Accuracy on Training data :  85.54 %
```

```
f1_score_train = f1_score(Y_train, X_train_prediction)
print('Training data F1 Score =', f1_score_train)
f1_score_test = recall_score(Y_test, X_test_prediction)
print('Test data F1 Score =', f1_score_test)
```

✓ 0.0s

```
Training data F1 Score = 0.8745519713261649
Test data F1 Score = 0.8181818181818182
```

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy on Test data : ', round(test_data_accuracy*100, 2), '%')
```

✓ 0.0s

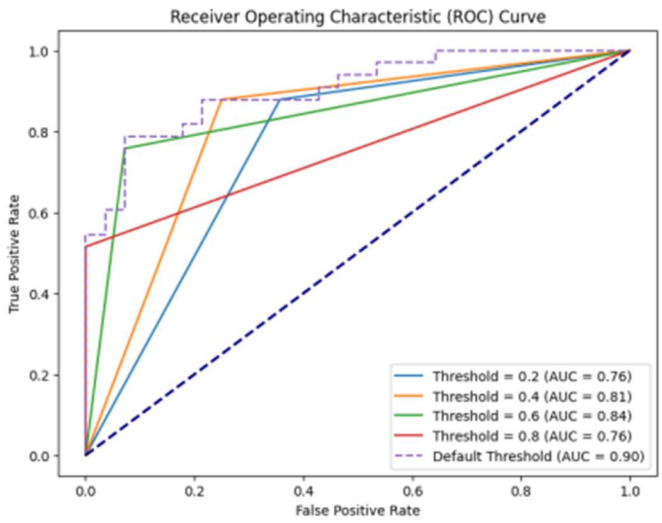
```
Accuracy on Test data :  80.33 %
```

```
cf_matrix = confusion_matrix(Y_test, X_test_prediction)
print(cf_matrix)
```

✓ 0.0s

```
[[22  6]
 [ 6 27]]
```

```
Y_probs = model.predict_proba(X_test)[:, 1]

fpr_default, tpr_default, _ = roc_curve(Y_test, Y_probs)
roc_auc_default = auc(fpr_default, tpr_default)

thresholds = [0.2, 0.4, 0.6, 0.8]

roc_auc_values = []

plt.figure(figsize=(8, 6))

for threshold in thresholds:

    Y_pred_threshold = [1 if prob >= threshold else 0 for prob in Y_probs]


    fpr, tpr, _ = roc_curve(Y_test, Y_pred_threshold)
    roc_auc = auc(fpr, tpr)
    roc_auc_values.append(roc_auc)


    plt.plot(fpr, tpr, label=f'Threshold = {threshold} (AUC = {roc_auc:.2f})')

plt.plot(fpr_default, tpr_default, linestyle='--', label=f'Default Threshold (AUC = {roc_auc_default:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')

plt.show()
```



ASSIGNMENT 6

```
[ ] import numpy as np
    import pandas as pd

[ ] #Import Dataset
    df = pd.read_csv('Churn_Modelling.csv')

    df.shape

    (10000, 14)

    df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geograp
0	1	15634602	Hargrave	619	Fran
1	2	15647311	Hill	608	Sp
2	3	15619304	Onio	502	Fran
3	4	15701354	Boni	699	Fran
4	5	15737888	Mitchell	850	Sp

```
    df.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 10000 entries, 0 to 9999
    Data columns (total 14 columns):
     #   Column           Non-Null Count  Dtype
    ---  ------           --------------  -----
     0   RowNumber        10000 non-null  int64
     1   CustomerId       10000 non-null  int64
     2   Surname          10000 non-null  object
     3   CreditScore      10000 non-null  int64
     4   Geography        10000 non-null  object
     5   Gender           10000 non-null  object
     6   Age              10000 non-null  int64
     7   Tenure           10000 non-null  int64
     8   Balance          10000 non-null  float64
     9   NumOfProducts    10000 non-null  int64
     10  HasCrCard        10000 non-null  int64
     11  IsActiveMember   10000 non-null  int64
     12  EstimatedSalary  10000 non-null  float64
     13  Exited           10000 non-null  int64
    dtypes: float64(2), int64(9), object(3)
    memory usage: 1.1+ MB

[ ] df.drop(columns = ['RowNumber','CustomerId','Su
```

```
    df['Geography'].value_counts()

    France     5014
    Germany    2509
    Spain      2477
    Name: Geography, dtype: int64

[ ] df['Gender'].value_counts()

    Male      5457
    Female    4543
    Name: Gender, dtype: int64

[ ]

[ ] df.drop(columns = ['RowNumber','CustomerId','Surname'],inpla

[ ] df = pd.get_dummies(df,columns=['Geography','Gender'],drop_f
```

```
[ ] df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace=True)

[ ] df = pd.get_dummies(df,columns=['Geography','Gender'],drop_first=True)

[ ] df
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct
0	619	42	2	0.00	1	1	
1	608	41	1	83807.86	1	0	
2	502	42	8	159660.80	3	1	
3	699	39	1	0.00	2	0	
4	850	43	2	125510.82	1	1	
...	...	...	...	...	...	...	
9995	771	39	5	0.00	2	1	
9996	516	35	10	57369.61	1	1	
9997	709	36	7	0.00	1	0	
9998	772	42	3	75075.31	2	1	
9999	792	28	4	130142.79	1	1	

```
X = df.drop(columns=['Exited'])
y = df['Exited'].values

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

```
[ ] X_train.shape
```

```
(8000, 11)
```

```
[ ] from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaledf = scaler.transform(X_test)
```

```
[ ] X_train_scaled
```

```
array([[-0.23082038, -0.94449979, -0.70174202, ...,  1.71490137,
        -0.57273139,  0.91509065],
       [-0.25150912, -0.94449979, -0.35520275, ..., -0.58312392,
        -0.57273139, -1.09278791],
       [-0.3963303 ,  0.77498705,  0.33787579, ...,  1.71490137,
```

```
[ ] import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
[ ] model = Sequential()

model.add(Dense(3,activation='sigmoid',input_dim=11))
model.add(Dense(1,activation='sigmoid'))
model.summary()
```

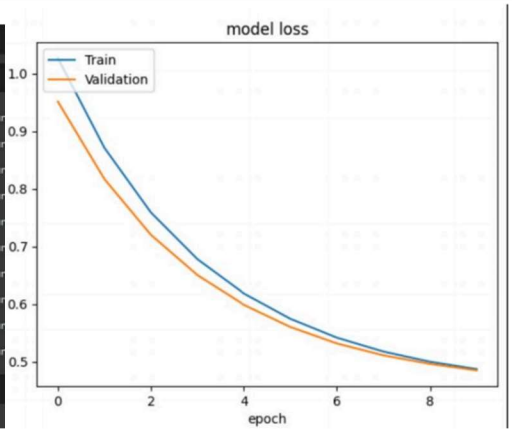
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 3)                 36

 dense_3 (Dense)             (None, 1)                 4

=================================================================
Total params: 40 (160.00 Byte)
Trainable params: 40 (160.00 Byte)
Non-trainable params: 0 (0.00 Byte)
```

```
[ ] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
model.compile(loss='binary_crossentropy', optimizer='Adam',metrics=['accuracy'])
```

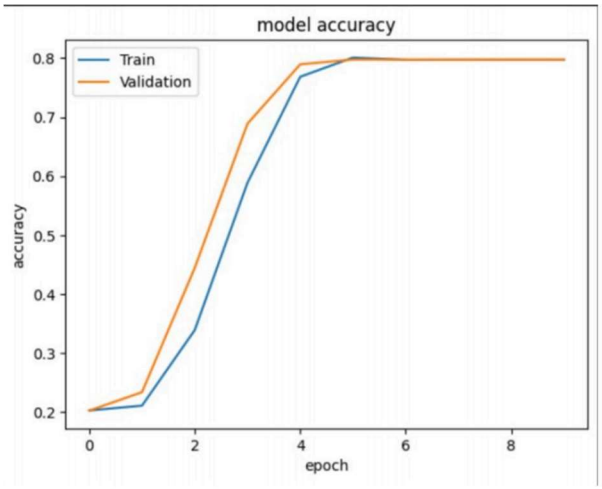
```
[ ] history = model.fit(X_train_scaled, y_train, batch_size=50,epochs=10,verbose=1,validation_split=0.2)

Epoch 1/10
128/128 [==============================] - 1s 2ms/step - loss: 1.0269 - accuracy: 0.2028 - val_loss: 0.9514 - val_accur
Epoch 2/10
128/128 [==============================] - 0s 2ms/step - loss: 0.8711 - accuracy: 0.2109 - val_loss: 0.8169 - val_accur
Epoch 3/10
128/128 [==============================] - 0s 1ms/step - loss: 0.7587 - accuracy: 0.3389 - val_loss: 0.7200 - val_accur
Epoch 4/10
128/128 [==============================] - 0s 1ms/step - loss: 0.6777 - accuracy: 0.5886 - val_loss: 0.6502 - val_accur
Epoch 5/10
128/128 [==============================] - 0s 2ms/step - loss: 0.6184 - accuracy: 0.7683 - val_loss: 0.5987 - val_accur
Epoch 6/10
128/128 [==============================] - 0s 1ms/step - loss: 0.5745 - accuracy: 0.8008 - val_loss: 0.5603 - val_accur
Epoch 7/10
128/128 [==============================] - 0s 1ms/step - loss: 0.5417 - accuracy: 0.7970 - val_loss: 0.5317 - val_accur
Epoch 8/10
128/128 [==============================] - 0s 2ms/step - loss: 0.5176 - accuracy: 0.7972 - val_loss: 0.5110 - val_accur
Epoch 9/10
128/128 [==============================] - 0s 1ms/step - loss: 0.4999 - accuracy: 0.7972 - val_loss: 0.4962 - val_accur
Epoch 10/10
128/128 [==============================] - 0s 2ms/step - loss: 0.4872 - accuracy: 0.7972 - val_loss: 0.4851 - val_accur
```

```
[ ] y_pred = model.predict(X_test)

63/63 [==============================] - 0s 761us/step
```




```
[ ] from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

```
0.7925
```



ASSIGNMENT 7

```
|: import numpy as np
   import pandas as pd
```

1 IRIS dataset

```
|: iris = pd.read_csv('./P7/Iris.csv - Iris.csv.csv')
   iris
```

:		Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	\
0	1	5.1	3.5	1.4	0.2	
1	2	4.9	3.0	1.4	0.2	
2	3	4.7	3.2	1.3	0.2	

```
[3]: cols = iris.columns
     Xcols = cols[:-1]
     Ycols = cols[-1]
```

```
[4]: irisX = iris[Xcols]
     irisX
```

[4]: | | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm |
|---|---|---|---|---|---|
| 0 | 1 | 5.1 | 3.5 | 1.4 | 0.2 |
| 1 | 2 | 4.9 | 3.0 | 1.4 | 0.2 |
| 2 | 3 | 4.7 | 3.2 | 1.3 | 0.2 |
| 3 | 4 | 4.6 | 3.1 | 1.5 | 0.2 |

```
: iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):

   memory usage: 7.2+ KB
```

```
6]: irisX.isna().sum()
```

```
6]: Id               0
    SepalLengthCm    0
    SepalWidthCm     0
    PetalLengthCm    0
    PetalWidthCm     0
    dtype: int64
```

```
']: irisY = pd.DataFrame(iris[Ycols])
    irisY
```

```
']:             Species
    0       Iris-setosa
    1       Iris-setosa
    2       Iris-setosa
    3       Iris-setosa
    4       Iris-setosa
    ..              ...
```

```
8]: from sklearn.preprocessing import OneHotEncoder
    encoder = OneHotEncoder(sparse=False)
    irisYE = pd.DataFrame(encoder.fit_transform(irisY))
```

```
c:\users\avneet\desktop\python module\lib\site-
packages\sklearn\preprocessing\_encoders.py:975: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

```
9]: irisYE.isna().sum()
```

```
9]: 0    0
    1    0
    2    0
    dtype: int64
```

1.0.1 Data split

```
0]: from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
```

```
1]: TrainX , TestX, TrainY, TestY = train_test_split(irisX, irisYE, test_size=0.5,␣
     ↪random_state=42)
```

```
2]: TrainY
```

```
2]:       0    1    2
    84   0.0  1.0  0.0
```

```
TestY
```

```
     0    1    2
73   0.0  1.0  0.0
18   1.0  0.0  0.0
118  0.0  0.0  1.0
78   0.0  1.0  0.0
76   0.0  1.0  0.0
..   …    …    …
```

```
from keras.models import Sequential
from keras.layers import Dense
```

```
model = Sequential()
# Add an input layer
model.add(Dense(16, input_shape=(5,), activation='relu'))
# Add one hidden layer
model.add(Dense(8, activation='relu'))
# Add an output layer
model.add(Dense(3, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
# Train the model
model.fit(TrainX, TrainY, epochs=100, batch_size=10, verbose = False)
```

```
<keras.src.callbacks.History at 0x289cf80c910>
```

```
pred = model.predict(TestX).argmax(axis=1)
pred
```

```
3/3 [==============================] - 0s 4ms/step
```

```
array([2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 2, 2, 2, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 2, 0, 0, 2, 1, 0, 0, 0, 2, 2, 2, 0,
       0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 0, 2, 0,
       2, 2, 0, 2, 2, 0, 2, 2, 2], dtype=int64)
```

```
actual = TestY.values.argmax(axis=1)
actual
```

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0, 1, 2, 2, 1, 2, 1, 2, 1, 0, 2, 1, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0,
       1, 2, 0, 1, 2, 0, 2, 2, 1], dtype=int64)
```

```
accuracy_score(pred, actual)
```

```
0.72
```

```
churn = pd.read_csv('./P7/Churn_Modelling.csv - Churn_Modelling.csv.csv')
churn
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age
0	1	15634602	Hargrave	619	France	Female	42
1	2	15647311	Hill	608	Spain	Female	41
2	3	15619304	Onio	502	France	Female	42
3	4	15701354	Boni	699	France	Female	39
4	5	15737888	Mitchell	850	Spain	Female	43
…	…	…	…	…	…	…	…
9995	9996	15606229	Obijiaku	771	France	Male	39

```
cols = churn.columns
cols
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
Xcols = cols[:-1]
Ycols = cols[-1]
```

```
churnX = churn[Xcols]
churnX.drop(['Surname', 'RowNumber', 'CustomerId', ], axis=1, inplace = True)
churnX
```

```
churnX.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
```

```
churnY = pd.DataFrame(churn[Ycols])
churnY
```

	Exited
0	1
1	0
2	1
3	0
4	0
…	…
9995	0
9996	0
9997	1
9998	1
9999	0

[10000 rows x 1 columns]

```
churnY.isna().sum()
```

```
Exited    0
dtype: int64
```

```
churnX
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts
0	619	France	Female	42	2	0.00	1
1	608	Spain	Female	41	1	83807.86	1
2	502	France	Female	42	8	159660.80	3
3	699	France	Female	39	1	0.00	2
4	850	Spain	Female	43	2	125510.82	1

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False)
toEncodeCols = ['Geography', 'Gender']
dummyDF = churnX[toEncodeCols]
dummyDF
```

	Geography	Gender
0	France	Female
1	Spain	Female
2	France	Female
3	France	Female
4	Spain	Female
…	…	…
9995	France	Male
9996	France	Male
9997	France	Female
9998	Germany	Male
9999	France	Female

```
dummyencoded = pd.DataFrame(encoder.fit_transform(dummyDF))
dummyencoded
```

	0	1	2	3	4
0	1.0	0.0	0.0	1.0	0.0
1	0.0	0.0	1.0	1.0	0.0
2	1.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	1.0	0.0

```
churnX.drop(toEncodeCols, axis=1, inplace=True)
```

```
churnX = pd.concat([churnX, dummyencoded], axis=1)
churnX
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	619	42	2	0.00	1	1
1	608	41	1	83807.86	1	0
2	502	42	8	159660.80	3	1
3	699	39	1	0.00	2	0
4	850	43	2	125510.82	1	1

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
TrainX , TestX, TrainY, TestY = train_test_split(churnX, churnY, test_size=0.3,␣
 ↪random_state=42)
```

```
TrainX
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	\
9069	619	32	4	175406.13	2	1	
2603	643	34	7	160426.07	1	0	
7738	561	33	6	0.00	2	0	
1579	618	41	8	37702.79	1	1	
5058	714	37	9	148466.93	2	0	

```
from keras.models import Sequential
from keras.layers import Dense
```

```
model = Sequential()
# Add an input layer
model.add(Dense(16, input_shape=(13,), activation='relu'))
# Add one hidden layer
model.add(Dense(8, activation='relu'))
# Add an output layer
model.add(Dense(1, activation='sigmoid'))
# Compile the model
```

```
predF = pred>0.50
```

```
predF
```

```
array([[False],
       [False],
       [False],
       …,
       [False],
       [False],
       [False]])
```

```
accuracy_score(predF, TestY)
```

```
0.803
```

```
gre = pd.read_csv('./P7/Gre_Admission.csv - Gre_Admission.csv.csv')
```

```
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
# Train the model
model.fit(TrainX, TrainY, epochs=20, batch_size=10, verbose = False)
```

```
P  LOR   CGPA  \
5   4.5  9.65
```

```
<keras.src.callbacks.History at 0x289d1b0d730>
```

```
pred = model.predict(TestX)
pred
```

```
94/94 [==============================] - 0s 3ms/step
```

```
array([[8.0815299e-10],
       [2.5235298e-16],
       [3.4546113e-06],
```

```
cols = gre.columns
cols
```

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score',
       'LOR ', 'CGPA', 'Research', 'Chance'],
      dtype='object')
```

```
gre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         400 non-null    int64
 1   GRE Score          400 non-null    int64
 2   TOEFL Score        400 non-null    int64

memory usage: 28.2 KB
```

```
Xcols = cols[:-1]
Ycols = cols[-1]
```

```
greX = gre[Xcols]
greX
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	\
0	1	337	118	4	4.5	4.5	9.65	
1	2	324	107	4	4.0	4.5	8.87	
2	3	316	104	3	3.0	3.5	8.00	
3	4	322	110	3	3.5	2.5	8.67	
4	5	314	103	2	2.0	3.0	8.21	
..	…	…	…	…	…	…	…	

```
greY = pd.DataFrame(gre[Ycols])
greY
```

	Chance
0	0.92
1	0.76
2	0.72
3	0.80
4	0.65
..	…
395	0.82

```
from sklearn.model_selection import train_test_split
TrainX , TestX, TrainY, TestY = train_test_split(greX,greY, test_size=0.3,␣
 ↪random_state=42)
```

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
# Add an input layer
model.add(Dense(16, input_shape=(8,), activation='tanh'))
# Add one hidden layer
model.add(Dense(8, activation='tanh'))
# Add an output layer
model.add(Dense(1, activation='relu'))
# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')
# Train the model
model.fit(TrainX, TrainY, epochs=50, batch_size=32, verbose = False)
```

```
<keras.src.callbacks.History at 0x289d3000550>
```

```
pred = model.predict(TestX)
```

```
4/4 [==============================] - 0s 4ms/step
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(TestY, pred)
mse
```

```
0.5273366666666667
```

ASSIGNMENT 8

```
import os
import torch
import torchvision
import tarfile
from torchvision.datasets.utils import download_url
from torch.utils.data import random_split
```

```
project_name='CNN'
```

```
dataset_url = "https://s3.amazonaws.com/fast-ai-imageclas/cifar10.tgz"
download_url(dataset_url, '.')
```

```
# Extract from archive
with tarfile.open('./cifar10.tgz', 'r:gz') as tar:
    tar.extractall(path='./data')
```

```
data_dir = './data/cifar10'

print(os.listdir(data_dir))
classes = os.listdir(data_dir + "/train")
print(classes)
```

```
['test', 'train']
['airplane', 'ship', 'dog', 'horse', 'automobile', 'cat', 'truck', 'frog',
'bird', 'deer']
```

```
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
```

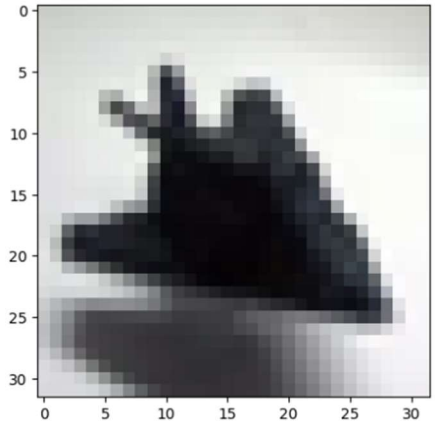
```
dataset = ImageFolder(data_dir+'/train', transform=ToTensor())
```

```
import matplotlib
import matplotlib.pyplot as plt
```

```
%matplotlib inline

matplotlib.rcParams['figure.facecolor'] = '#ffffff'
```

```
]: def show_example(img, label):
    print('Label: ', dataset.classes[label], "("+str(label)+")")
    plt.imshow(img.permute(1, 2, 0))
```

```
]: show_example(*dataset[0])
```




```
: random_seed = 42
  torch.manual_seed(random_seed);
```

```
: val_size = 5000
  train_size = len(dataset) - val_size

  train_ds, val_ds = random_split(dataset, [train_size, val_size])
  len(train_ds), len(val_ds)
```

```
: (45000, 5000)
```

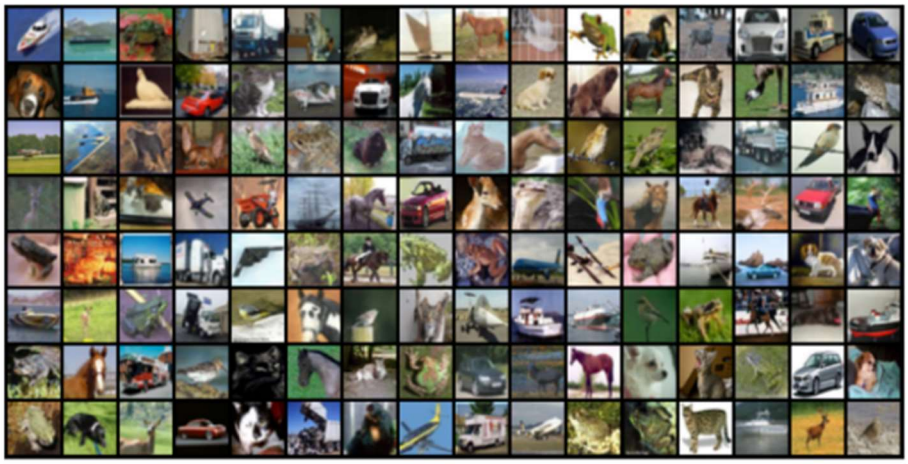
S

```
from torch.utils.data.dataloader import DataLoader
batch_size=128
```

```
train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4,␣
 ↪pin_memory=True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)
```

:
```
from torchvision.utils import make_grid

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 6))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break
```

:
```
show_batch(train_dl)
```



```
simple_model = nn.Sequential(
    nn.Conv2d(3, 8, kernel_size=3, stride=1, padding=1),
    nn.MaxPool2d(2, 2)
)
```

```
for images, labels in train_dl:
    print('images.shape:', images.shape)
    out = simple_model(images)
    print('out.shape:', out.shape)
    break
```

:
```
import torch.nn as nn
import torch.nn.functional as F
```

```
images.shape: torch.Size([128, 3, 32, 32])
out.shape: torch.Size([128, 8, 16, 16])
```

```
class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.
↪4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

```
class Cifar10CnnModel(ImageClassificationBase):
    def __init__(self):
        super().__init__()
```

```
        self.network = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 64 x 16 x 16

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 128 x 8 x 8

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 256 x 4 x 4

            nn.Flatten(),
            nn.Linear(256*4*4, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 10))

    def forward(self, xb):
        return self.network(xb)
```

```
model = Cifar10CnnModel()
model
```

```
: Cifar10CnnModel(
    (network): Sequential(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
      (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU()
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
  ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU()
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU()
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
  ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
for images, labels in train_dl:
    print('images.shape:', images.shape)
    out = model(images)
    print('out.shape:', out.shape)
    print('out[0]:', out[0])
    break
```

```
images.shape: torch.Size([128, 3, 32, 32])
out.shape: torch.Size([128, 10])
out[0]: tensor([ 0.0239, -0.0466,  0.0067,  0.0
```

```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

```
device = get_default_device()
device
```

```
device(type='cuda')
```

```
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
to_device(model, device);
```

```python
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

```python
model = to_device(Cifar10CnnModel(), device)
```

```python
evaluate(model, val_dl)
```

]: {'val_loss': 2.302245855331421, 'val_acc': 0.10039062798023224}

]:
```python
num_epochs = 10
opt_func = torch.optim.Adam
lr = 0.001
```

]:
```python
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
Epoch [0], train_loss: 1.7556, val_loss: 1.4513, val_acc: 0.4541
Epoch [1], train_loss: 1.2430, val_loss: 1.0811, val_acc: 0.6059
Epoch [2], train_loss: 0.9821, val_loss: 0.9124, val_acc: 0.6827
Epoch [3], train_loss: 0.8046, val_loss: 0.8525, val_acc: 0.7020
Epoch [4], train_loss: 0.6658, val_loss: 0.8031, val_acc: 0.7203
Epoch [5], train_loss: 0.5491, val_loss: 0.7401, val_acc: 0.7449
Epoch [6], train_loss: 0.4343, val_loss: 0.7187, val_acc: 0.7646
Epoch [7], train_loss: 0.3390, val_loss: 0.7451, val_acc: 0.7755
Epoch [8], train_loss: 0.2555, val_loss: 0.8065, val_acc: 0.7738
Epoch [9], train_loss: 0.1990, val_loss: 0.9657, val_acc: 0.7625
```