DATA HANDLING AND VISUALIZATION LABSHEETS

NAME: DHANUSHA A
ROLL NO: 20201ISB0005

LABSHEET-1
INTRODUCTION TO NUMPY

```
import numpy as np a=np.array([1,2,3])
b=np.array([1,2,3])

add=np.add(a,b) add
```

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

```
a=np.array([5,10,20])
b=np.array([4,8,10])

sub=np.subtract(a,b) sub
```

```
array([ 1,  2, 10])
```

```
a=np.array([5,10,20])
b=np.array([4,8,10])

sub=np.multiply(a,b) sub
```

```
array([ 20,  80, 200])
```

```
a=np.array([5,7,9])
b=np.array([4,5,6])

sub=np.mod(a,b) sub
```

```
array([1, 2, 3])
```

```
a=np.array([1,2,3])
b=np.array([1,2,3])

add=np.power(a,b) add
```

```
array([ 1,  4, 27])
```

Series creation

```
import pandas as pd import numpy as np
data=np.array(['a','b','c','d']) s=pd.Series(data)
print(s)
```

```
0 a
1 b
2 c
3 d
dtype: object
```

Series with index

```
import pandas as pd import numpy as np
data=np.array(['a','b','c','d'])
s=pd.Series(data,index=[101,102,103,104]) print(s)
```

```
101    a
102    b
103    c
104    d
dtype: object
```

Series with Dictionary

```
import pandas as pd import numpy as np
data={'a': 0.,'b': 1.,'c': 2.}
s=pd.Series(data) print(s)
```

```
a   0.0
b     1.0
c     2.0
dtype: float64
```

Series with Dictionary with index

```
import pandas as pd import numpy as np
data={'a': 0.,'b': 1.,'c': 2.}
s=pd.Series(data,index=['b','c','d','a']) print(s)
```

```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

Create Series from Scalar

```
import pandas as pd import numpy as np
s= pd.Series(5, index=[0,1,2,3]) print(s)
```

```
0    5
1    5
2    5
3    5
dtype: int64
```

Retrieving data from the zeroth position

```
import pandas as pd
s= pd.Series([1,2,3,4,5],index=['a','b','c','d','e']) print(s[0])
1
```

```
import pandas as pd
s=
pd.Series([100,101,102,103,104,105,106,107,108,109,110],index=['a','b','c','d','e','f','g
','h','i','j','k'])
print(s[:3])
```

```
a   100
b     101
c     102
dtype: int64
```

```
import pandas as pd
s=
pd.Series([100,101,102,103,104,105,106,107,108,109,110],index=['a','b','c','d','e','f','g
```

```
','h','i','j','k'])
print(s[2:8])
```

```
c    102
d    103
e    104
f    105
g    106
h    107
dtype: int64
```

Using lable value

```
import pandas as pd
s=
pd.Series([100,101,102,103,104,105,106,107,108,109,110],index=['a','b','c','d','e','f','g
','h','i','j','k'])
print(s['a'])
```

```
100
```

```
import pandas as pd
s=
pd.Series([100,101,102,103,104,105,106,107,108,109,110],index=['a','b','c','d','e','f','g
','h','i','j','k'])
print(s[['a','e','i','d']])
```

```
a    100
e    104
i    108
d    103
dtype: int64
```

Data Frames

```
import pandas as pd
df=pd.read_csv("/content/nyc_weather.csv")
```

Create data frame with empty data

```
import pandas as pd df=pd.DataFrame()
print(df)
```

```
Empty DataFrame Columns: []
Index: []
```

Create data frame from list

```
import pandas as pd data=[1,2,3,4,5]
df=pd.DataFrame(data) print(df)
```

```
    0
0  1
1  2
2  3
3  4
4  5
```

```
import pandas as pd
data=[['Alex',10],['Bob',12],['Clarke',13]] df=pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

```
Name  Age
Alex    10
```

```
Bob      12
Clarke   13
```

```
import pandas as pd
data=[['Dha',21, 10001,'A'],['Sha',23, 10002,'B'],['Dee',22, 10003,'C']]
df=pd.DataFrame(data,columns=['Name','Age','Rollno','Sec'],dtype=float) print(df)
```

```
    Name  Age   Rollno Sec
0  Dha  21.0  10001.0    A
1  Sha  23.0  10002.0    B
2  Dee  22.0  10003.0    C
<ipython-input-31-f22448152035>:3: FutureWarning: Could not cast to float64, falling back
to object. This behavior is deprecated. I
df=pd.DataFrame(data,columns=['Name','Age','Rollno','Sec'],dtype=float)
```

Cretae data frame from Dictionary

```
import pandas as pd
data={'Name':['Tom','Jack','Steve','Ricky'],'Age':[23,25,22,29]}
df=pd.DataFrame(data,index=['rank1','rank2','rank3','rank4'])
print(df)
```

```
         Name  Age
rank1     Tom   23
rank2    Jack   25
rank3   Steve   22
rank4   Ricky   29
```

LABSHEET-2
WORKING WITH PANDAS

```
import pandas as pd


def load_data():
df_all = pd.read_csv('/content/train.csv')
return df_all.loc[:300,['Survived','Pclass','Sex','Cabin','Embarked']].dropna()
df=load_data()
```

```
df.head()
```

	Survived	Pclass	Sex	Cabin	Embarked
0	0	1	male	C30	S
1	1	1	female	D33	C
9	1	3	male	E121	S
10	1	1	female	B22	S
14	0	1	male	B51 B53 B55	S

FINDING DUPLICATE ROWS

```
df.Cabin.duplicated()
```

```
False
False
9      False
10     False
14     False
...
271    False
278    False
286    False
False
False
Name: Cabin, Length: 80, dtype: bool
```

```
df.duplicated()
```

```
False
False
9      False
10     False
14     False
...
271    False
278    False
286    False
False
False
Length: 80, dtype: bool
```

```
df.duplicated(subset=['Survived','Pclass','Sex'])
```

```
False
False
9      False
10     True
14     True
...
271    True
```

```
278    True
286    True
299    True
300    True
Length: 80, dtype: bool
```

COUNTING DUPLICATES AND NON DUPLICATES

```
df.Cabin.duplicated().sum()
```
11

```
df.duplicated().sum()
```
3

```
df.duplicated(subset=['Survived','Pclass','Sex']).sum()
```
70

```
(~df.duplicated()).sum()
```
77

EXTRACTING DUPLICATE ROWS USING LOC

```
df.loc[df.duplicated(), :]
```

	Survived	Pclass	Sex	Cabin	Embarked
138	1	2	female	F33	S
169	1	1	female	B77	S
237	1	1	female	B96 B98	S

USING KEEP

```
df.loc[df.duplicated(keep='first'), :]
```

	Survived	Pclass	Sex	Cabin	Embarked
138	1	2	female	F33	S
169	1	1	female	B77	S
237	1	1	female	B96 B98	S

```
df.loc[df.duplicated(keep='last'), :]
```

	Survived	Pclass	Sex	Cabin	Embarked
36	1	1	female	B77	S
77	1	1	female	B96 B98	S
134	1	2	female	F33	S

```
df.loc[df.duplicated(keep=False), :]
```

	Survived	Pclass	Sex	Cabin	Embarked
36	1	1	female	B77	S
77	1	1	female	B96 B98	S
134	1	2	female	F33	S
138	1	2	female	F33	S
169	1	1	female	B77	S
237	1	1	female	B96 B98	S

DROPPING DUPLICATED ROWS

```
df.drop_duplicates()
```

	Survived	Pclass	Sex	Cabin	Embarked
0	0	1	male	C30	S
1	1	1	female	D33	C
9	1	3	male	E121	S
10	1	1	female	B22	S
14	0	1	male	B51 B53 B55	S
...	...	...	...	...	...
271	1	1	male	C93	S
278	0	1	male	C111	C
286	1	1	male	C148	C
299	1	1	female	D21	S
300	1	2	male	F2	S

77 rows × 5 columns

```
df.drop_duplicates(keep=False)
```

	Survived	Pclass	Sex	Cabin	Embarked
0	0	1	male	C30	S
1	1	1	female	D33	C
9	1	3	male	E121	S
10	1	1	female	B22	S
14	0	1	male	B51 B53 B55	S
...	...	...	...	...	...
271	1	1	male	C93	S
278	0	1	male	C111	C
286	1	1	male	C148	C
299	1	1	female	D21	S
300	1	2	male	F2	S

74 rows × 5 columns

20201ISB0005

LABSHEET-3
DATA CLEANING

```
import pandas as pd import numpy as np

df=pd.read_csv('/content/2,1 dataset titanic.csv')


cols=['Name','Ticket','Cabin'] df=df.drop(cols,axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'> RangeIndex: 891 entries, 0 to 890
Data columns (total 9 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    SibSp        891 non-null    int64
      6    Parch        891 non-null    int64
      7    Fare         891 non-null    float64
      8    Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(2) memory usage: 62.8+ KB
```

```
df=df.dropna() df.info()
```

```
<class 'pandas.core.frame.DataFrame'> Int64Index: 712 entries, 0 to 890
Data columns (total 9 columns):
      #    Column       Non-Null Count  Dtype
      ---  ------       --------------  -----
      0    PassengerId  712 non-null    int64
      1    Survived     712 non-null    int64
      2    Pclass       712 non-null    int64
      3    Sex          712 non-null    object
      4    Age          712 non-null    float64
      5    SibSp        712 non-null    int64
      6    Parch        712 non-null    int64
      7    Fare         712 non-null    float64
      8    Embarked     712 non-null    object
dtypes: float64(2), int64(5), object(2) memory usage: 55.6+ KB
```

```
dummies=[]
cols=['Pclass','Sex','Embarked'] for col in cols:
dummies.append(pd.get_dummies(df[col]))


titanic_dummies= pd.concat(dummies,axis=1)


df= pd.concat((df,titanic_dummies), axis=1)


df= df.drop(['Pclass','Sex','Embarked'],axis=1)


df['Age'] = df['Age'].interpolate() print(df)
```

```
     PassengerId  Survived   Age  SibSp  Parch     Fare  1  2  3  female  \
0              1         0  22.0      1      0   7.2500  0  0  1       0
1              2         1  38.0      1      0  71.2833  1  0  0       1
2              3         1  26.0      0      0   7.9250  0  0  1       1
3              4         1  35.0      1      0  53.1000  1  0  0       1
4              5         0  35.0      0      0   8.0500  0  0  1       0
..       ...       ...   ...    ...    ...      ... .. .. ..     ...
885          886         0  39.0      0      5  29.1250  0  0  1       1
886          887         0  27.0      0      0  13.0000  0  1  0       0
```

```
      887     888         1     19.0    0      0    30.0000  1  0  0    1
      889     890         1     26.0    0      0    30.0000  1  0  0    0
      890     891         0     32.0    0      0    7.7500   0  0  1    0

         male   C  Q  S
      0  1      0  0  1
      1  0      1  0  0
      2  0      0  0  1
      3  0      0  0  1
      4  1      0  0  1
..       ... .. .. ..
      885   0    0  1  0
      886   1    0  0  1
      887   0    0  0  1
      889   1    1  0  0
      890   1    0  1  0

[712 rows x 14 columns]
```

MIN MAX SCALAR STANDARDIZATION

```
from sklearn.preprocessing import MinMaxScaler data=[[-1,2],[-0.5,6],[0,10],[1,18]]
scaler=MinMaxScaler()
print(scaler.fit(data)) print('     ')
MinMaxScaler()
print(scaler.data_max_) print('     ')
print('scaler.transform(data)')
MinMaxScaler() [ 1. 18.]
scaler.transform(data)


from numpy import asarray
from sklearn.preprocessing import StandardScaler data=asarray([[100,0.001],
[8,0.05],
[50,0.005],
[88,0.07],
[4,0.1]])
print(data)
scaler= StandardScaler()
scaled = scaler.fit_transform(data) print(scaled)

[[1.0e+02 1.0e-03]
[8.0e+00 5.0e-02]
[5.0e+01 5.0e-03]
[8.8e+01 7.0e-02]
[4.0e+00 1.0e-01]]
[[ 1.26398112 -1.16389967]
[-1.06174414  0.12639634]
[ 0.  -1.05856939]
[ 0.96062565  0.65304778]
[-1.16286263  1.44302493]]


from sklearn.preprocessing import MinMaxScaler data=[[-1,2],[-0.5,6],[0,10],[1,18]]
scaler=MinMaxScaler()
print(scaler.fit(data)) MinMaxScaler()
print(scaler.data_max_)
print('scaler.transform(data)')

MinMaxScaler() [ 1. 18.]
scaler.transform(data)
```

LABSHEET-4
Z-SCORE NORMALIZATION

```
import numpy as np
data= [1,2,2,2,3,1,1,15,2,2,2,3,1,1,2]
mean= np.mean(data) std= np.std(data)
print("mean of the dataset ids", mean) print("std is", std)
threshold=3 outlier=[]
for i in data:
z=(i-mean)/std if z>threshold:
outlier.append(i)
print("outlier in dataset is", outlier)
```

```
mean of the dataset ids 2.6666666666666665 std is 3.3598941782277745
outlier in dataset is [15]
```

20201ISB0005

LABSHEET-5

OUTLIER DETECTION WITH IQR

```
import numpy as np
import seaborn as sns data=[6,2,3,4,5,1,50]
sort_data=np.sort(data) sort_data
```

```
array([ 1,  2,  3,  4,  5,  6, 50])
```

```
Q1=-np.percentile(data, 25, interpolation = 'midpoint') Q2=-np.percentile(data, 50,
interpolation = 'midpoint') Q3=-np.percentile(data, 75, interpolation = 'midpoint')

print('Q1 25 percentile of the given data is, ', Q1) print('Q2 50 percentile of the given
data is, ', Q2) print('Q3 75 percentile of the given data is, ', Q3)

IQR = Q3 - Q1
print('IQR is', IQR)
```

```
Q1 25 percentile of the given data is,  -2.5 Q2 50 percentile of the given data is,  -
4.0 Q3 75 percentile of the given data is,  -5.5 IQR is -3.0
```

```
low_lim = Q1 - 1.5 * IQR up_lim = Q3 + 1.5 * IQR
```

20201ISB0005

LABSHEET-6

MATPLOTLIB

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


df=pd.read_csv("/content/Toyota.csv", index_col = 0, na_values = ['??','???'])


df.info()
```

```
<class 'pandas.core.frame.DataFrame'> Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Price       1436 non-null   int64
 1   Age         1336 non-null   float64
 2   KM          1421 non-null   float64
 3   FuelType    1336 non-null   object
 4   HP          1436 non-null   object
 5   MetColor    1286 non-null   float64
 6   Automatic   1436 non-null   int64
 7   CC          1436 non-null   int64
 8   Doors       1436 non-null   object
 9   Weight      1436 non-null   int64
dtypes: float64(3), int64(4), object(3) memory usage: 123.4+ KB
```

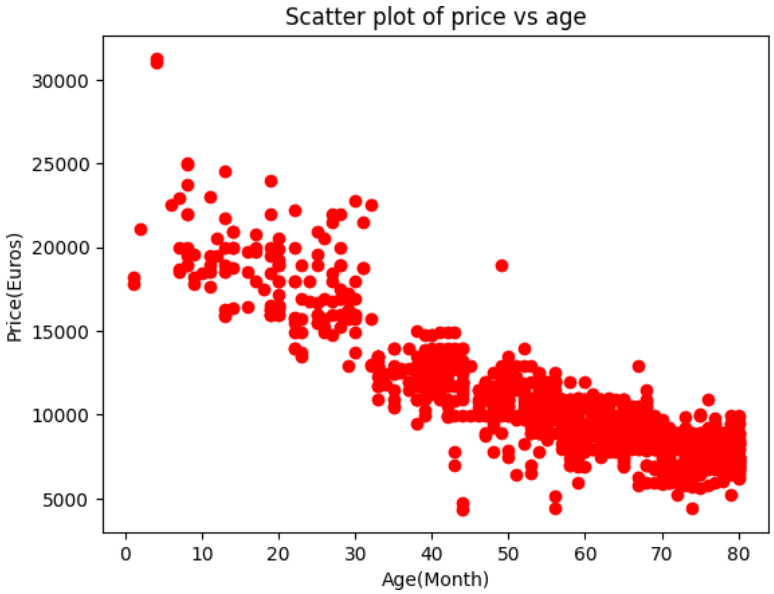
```
df.dropna(axis=0,inplace=True) df
```

	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	13500	23.0	46986.0	Diesel	90	1.0	0	2000	three	1165
1	13750	23.0	72937.0	Diesel	90	1.0	0	2000	3	1165
3	14950	26.0	48000.0	Diesel	90	0.0	0	2000	3	1165
4	13750	30.0	38500.0	Diesel	90	0.0	0	2000	3	1170
5	12950	32.0	61000.0	Diesel	90	0.0	0	2000	3	1170
...	...	...	...	...	...	...	...	...	...	...
1423	7950	80.0	35821.0	Petrol	86	0.0	1	1300	3	1015
1424	7750	73.0	34717.0	Petrol	86	0.0	0	1300	3	1015
1429	8950	78.0	24000.0	Petrol	86	1.0	1	1300	5	1065
1430	8450	80.0	23000.0	Petrol	86	0.0	0	1300	3	1015
1435	6950	76.0	1.0	Petrol	110	0.0	0	1600	5	1114

1099 rows × 10 columns

SCATTER PLOT

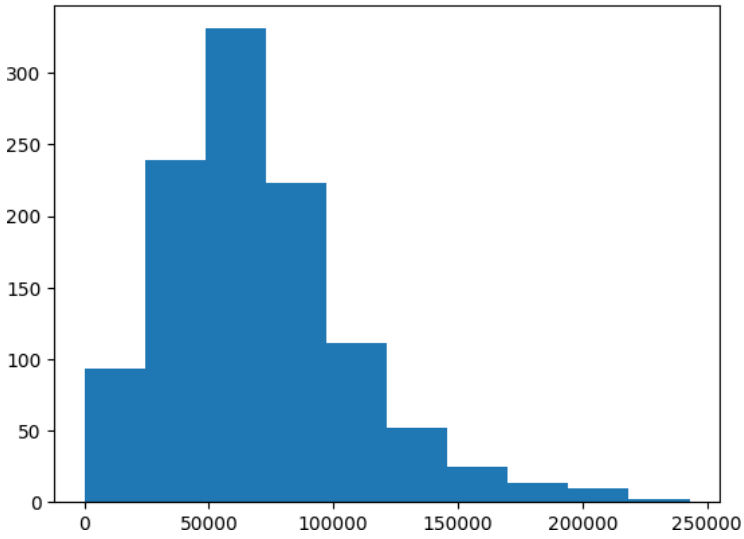
```
plt.scatter(df['Age'], df['Price'], c='red') plt.title('Scatter plot of price vs age')
plt.xlabel('Age(Month)') plt.ylabel('Price(Euros)') plt.show()
```



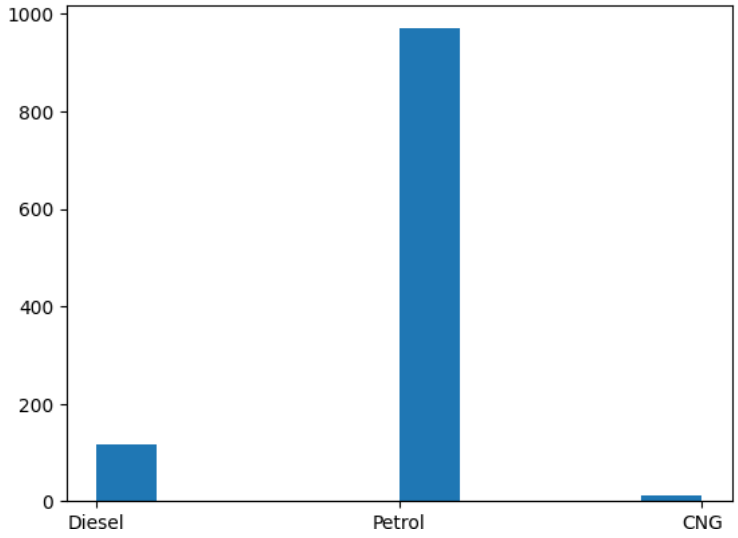
HISTOGRAM

```
plt.hist(df['KM'])
```

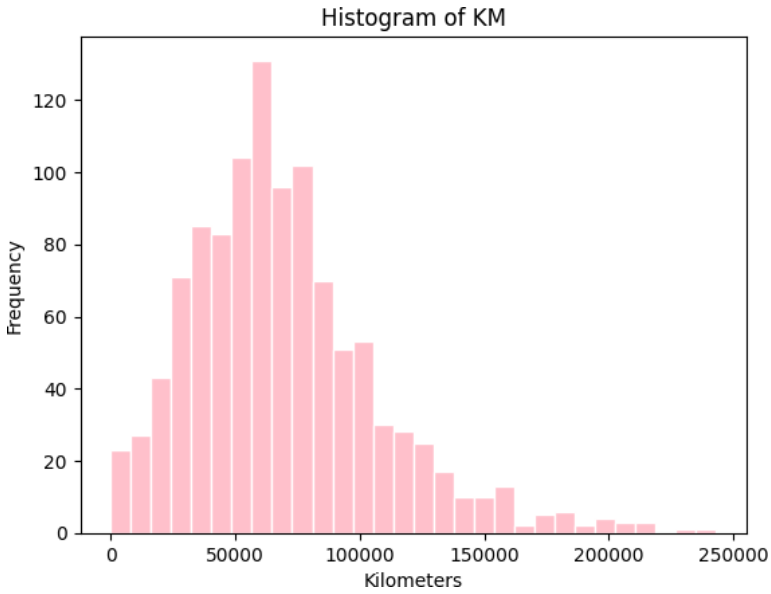
```
(array([ 93., 239., 331., 223., 111.,  52.,  25.,  13.,  10.,   2.]),
array([1.000000e+00, 2.430090e+04, 4.860080e+04, 7.290070e+04,
9.720060e+04, 1.215005e+05, 1.458004e+05, 1.701003e+05,
1.944002e+05, 2.187001e+05, 2.430000e+05]),
<BarContainer object of 10 artists>)
```




```
plt.hist(df['FuelType'])
(array([117.,   0.,   0.,   0.,   0., 970.,   0.,   0.,   0.,  12.]),
array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
<BarContainer object of 10 artists>)
```

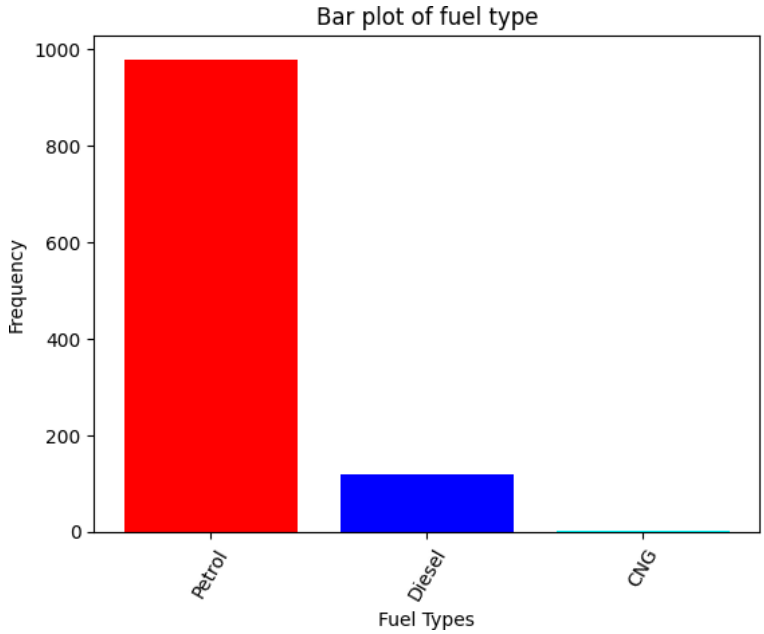



```
plt.hist(df['KM'],color='pink',edgecolor='white',bins=30) plt.title('Histogram of KM')
plt.xlabel('Kilometers') plt.ylabel('Frequency') plt.show()
```



BAR PLOT

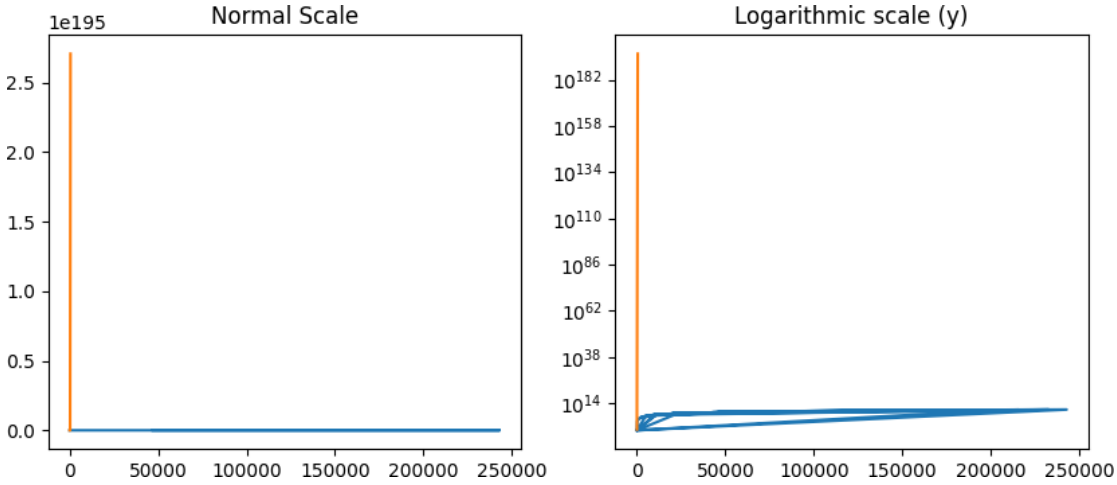
```
counts = [979,120,2]
fueltype= ('Petrol','Diesel','CNG') index= np.arange(len(fueltype))

plt.bar(index,counts,color=['red','blue','cyan']) plt.title('Bar plot of fuel type')
plt.xlabel('Fuel Types') plt.ylabel('Frequency')
plt.xticks(index, fueltype, rotation= 60) plt.show()
```



LINE PLOT

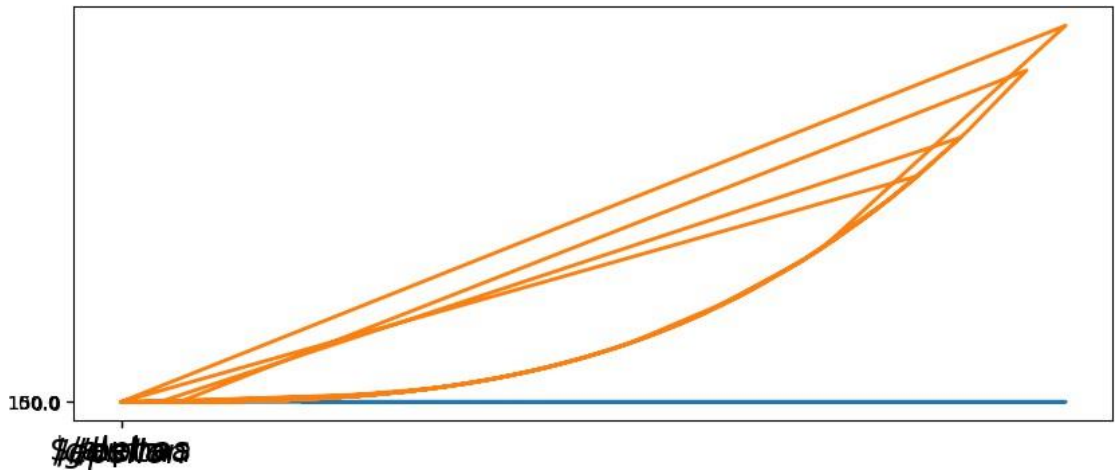
```
fig, axes = plt.subplots(1, 2, figsize=(10,4)) x=df['KM']
axes[0].plot(x, x**2, x, np.exp(x)) axes[0].set_title("Normal Scale")

axes[1].plot(x, x**2, x, np.exp(x)) axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)")
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:396: RuntimeWarning:
overflow encountered in exp result = getattr(ufunc, method)(*inputs, **kwargs)
Text(0.5, 1.0, 'Logarithmic scale (y)')
```

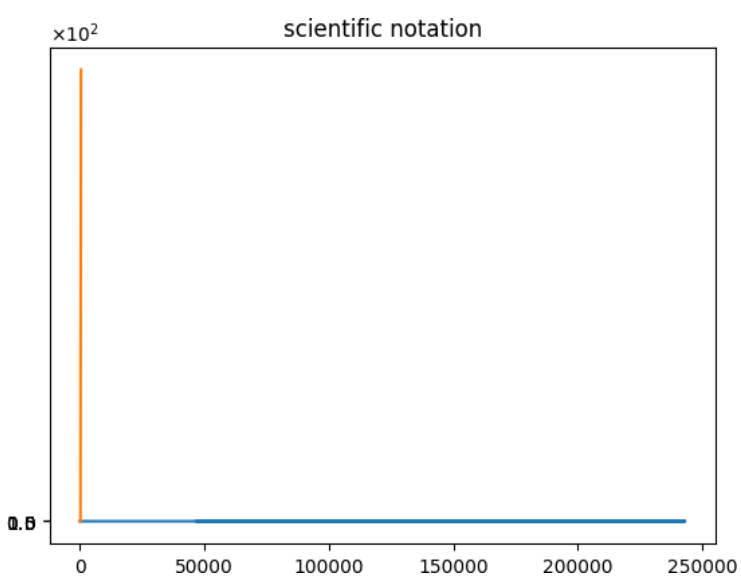



```
fig, ax = plt.subplots(figsize=(10,4)) x=df['KM']
ax.plot(x, x**2,x,x**3, lw=2) ax.set_xticks([1,2,3,4,5])
ax.set_xticklabels([r'$/alphas',r'$/beta$',r'$/gamma$',r'$/delta$', r'$/epsilon$'],
fontsize=18) yticks=[0,50,100,150]
ax.set_yticks(yticks)
ax.set_yticklabels(["$%.1f$" % y for y in yticks])
```

```
[Text(0, 0, '$0.0$'),
Text(0, 50, '$50.0$'),
Text(0, 100, '$100.0$'),
Text(0, 150, '$150.0$')]
```

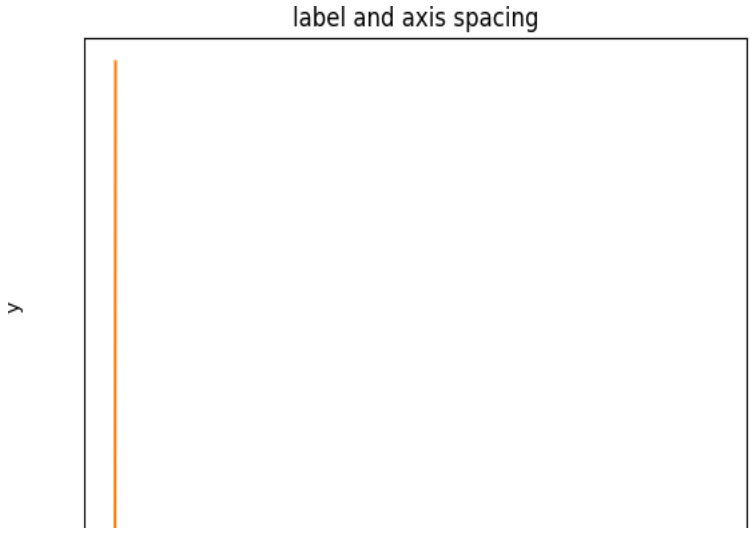



```
fig, ax= plt.subplots(1,1) x=df['KM']
ax.plot(x, x**2, x, np.exp(x))
ax.set_title("scientific notation") ax.set_yticks([0,50,100,150])
from matplotlib import ticker
formatter = ticker.ScalarFormatter(useMathText=True) formatter.set_scientific(True)
formatter.set_powerlimits((-1,1))
ax.yaxis.set_major_formatter(formatter)
```




```
import matplotlib

matplotlib.rcParams['xtick.major.pad'] = 5
matplotlib.rcParams['ytick.major.pad'] = 5

x = df['KM']
fig, ax = plt.subplots(1, 1)
ax.plot(x, x**2, x, np.exp(x)) ax.set_yticks([0, 50, 100, 150])
ax.set_title("label and axis spacing") ax.xaxis.labelpad = 5
ax.yaxis.labelpad = 5 ax.set_ylabel("x")
ax.set_ylabel("y") plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py:396: RuntimeWarning:
overflow encountered in exp result = getattr(ufunc, method)(*inputs, **kwargs)
```




```
import matplotlib

matplotlib.rcParams['xtick.major.pad'] = 3
matplotlib.rcParams['ytick.major.pad'] = 3
```

LABSHEET-7
INTERACTING WITH WEB API

```
import requests
pip install --upgrade 'library' ⇾ Collecting library
Downloading Library-0.0.0.tar.gz (1.4 kB) Preparing metadata (setup.py) ... done
Building wheels for collected packages: library
Building wheel for library (setup.py) ... done
Created wheel for library: filename=Library-0.0.0-py3-none-any.whl size=2054
sha256=33e04a1cd46e5d3b86146af77a7e80978fe44edaeba4a Stored in directory:
/root/.cache/pip/wheels/e0/71/7d/b0e29b944e43374597cd4e3b88c85197001c9bfcd5dce191f4
Successfully built library
Installing collected packages: library Successfully installed library-0.0.0


r = requests.get('https://www.romexchange.com/')


r

⇾ <Response [406]>


r.status_code ⇾ 406
url = 'https://www.romexchange.com/'
headers = {'Content-type': 'application/json'}


url

⇾ 'https://www.romexchange.com/'


headers

⇾ {'Content-type': 'application/json'} r=requests.get(url, headers = headers)
url = 'https://www.romexchange.com/'

headers = {'User-Agent': 'XY','Content-type': 'application/json'} r = requests.get(url,
headers=headers)

url

⇾ 'https://www.romexchange.com/'


headers

⇾ {'User-Agent': 'XY', 'Content-type': 'application/json'}


r

⇾ <Response [200]>


r.status_code ⇾ 200
url = 'https://www.romexchange.com/api?item=mastela&exact=false' headers = {'User-
Agent':'XY','Content-type':'application/json'}
```

```
r= requests.get(url, headers=headers) r.status_code
```

```
500
```

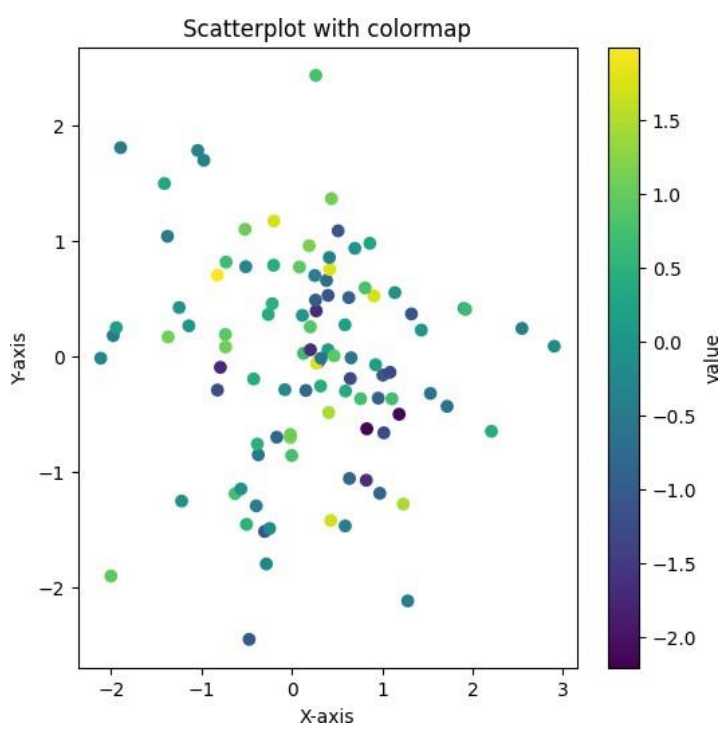
```
r.text
```

```
''
```

20201ISB0005

LABSHEET-8
COLORMAPS

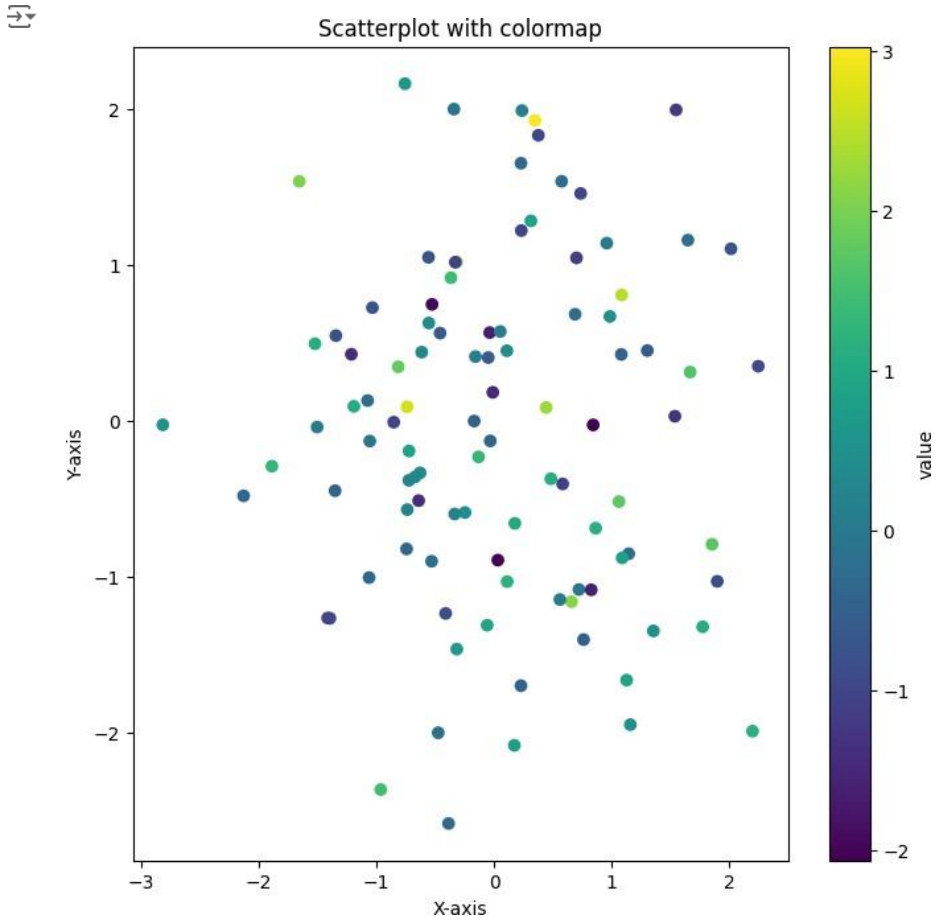
```
import pandas as pd import numpy as np
import matplotlib.pyplot as plt

#sample datafame with multiple columns
data=pd.DataFrame({"x":np.random.randn(100),"y":np.random.randn(100),"value":np.random.ran
dn(100)}) #define the colormap and alpha values
cmap="viridis" alpha=1
#create the scatterplot
plt.figure(figsize=(6,6))
plt.scatter(data["x"],data["y"],c=data["value"],cmap=cmap,alpha=alpha) #customize the
plot(optional)
plt.xlabel("X-axis") plt.ylabel("Y-axis")
plt.title("Scatterplot with colormap") plt.colorbar(label="value")
#show the plot plt.show()
```




```
import pandas as pd import numpy as np
print(np.random.randn(100))
```

```
[-1.65970274e-01 -3.26301492e-01 -6.97091694e-01 5.29185683e-01 1.65900203e-01 -
2.57310809e-01  1.87945887e-01 -1.47856355e+00
1.85465880e+00 -5.74773399e-02 -7.28047219e-01  1.43513290e+00
1.16276640e-01  3.62925427e-01  2.27296732e-01 -4.68725785e-01
-7.20465601e-01  2.31190101e-01  5.47647007e-01  6.14310198e-01
-2.88178116e-01 -2.59650445e-01  7.14726089e-02  2.91407763e-01 7.44199514e-01 -
1.03744520e+00  5.19583750e-02 -1.22315192e+00
2.82553552e-01  9.27484581e-01  4.68496647e-01  3.97669795e-01
-6.15495640e-01 -3.59199216e-01  1.45247374e-01 -1.61267440e-01
-1.08796055e+00  2.03942727e-01  1.33177945e-03  7.08911052e-01 1.92045492e+00 -
1.06460553e+00  9.71054014e-01  8.14301945e-01
1.01645092e-01 -9.38076692e-02  1.33631841e+00  2.55274328e-01
-5.17379367e-01 -1.71773916e+00  9.24194703e-01  1.67657214e-01
-1.72214971e+00  4.27042698e-01 -1.20346437e+00  2.83589309e-01 1.21334367e+00
4.14428011e-02 -1.48913563e+00  4.39560682e-01
```

```
-8.90366916e-01 -9.11298844e-01  3.62446399e-01  5.87632377e-01 1.22152619e+00
7.44396580e-01  1.75575979e+00  3.12178887e-01
-3.40512410e-01 -1.01818680e+00  4.62977518e-02  2.30443390e-01
-3.96879315e-01  1.20713778e+00 -1.20064064e+00 -9.12708432e-01 9.06172668e-01
7.05249075e-02 -9.42170303e-01 -8.52966288e-01
1.96198904e+00  3.61012540e-02  9.66762176e-01 -4.97875528e-01
2.78681896e-01 -1.16708383e+00  7.39087305e-01  1.27038245e+00
7.81304235e-01 -4.62440127e-01  1.00117969e+00 -9.07298230e-02
-1.95950298e-01  1.59291286e+00 -1.22572212e+00 -4.62563405e-01 5.41920487e-01
7.41261996e-01  1.42219990e+00 -9.65150475e-01]

import pandas as pd import numpy as np
import matplotlib.pyplot as plt

#sample datafame with multiple columns
data=pd.DataFrame({"x":np.random.randn(100),"y":np.random.randn(100),"value":np.random.ran
dn(100)}) #define the colormap and alpha values
cmap="viridis" alpha=1
#create the scatterplot
plt.figure(figsize=(8,8))
plt.scatter(data["x"],data["y"],c=data["value"],cmap=cmap,alpha=alpha) #customize the
plot(optional)
plt.xlabel("X-axis") plt.ylabel("Y-axis")
plt.title("Scatterplot with colormap") plt.colorbar(label="value")
#show the plot plt.show()
```

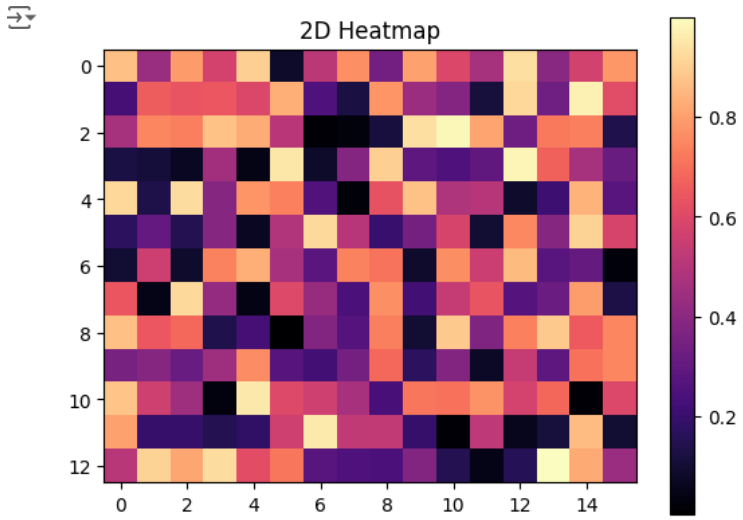


20201ISB0005

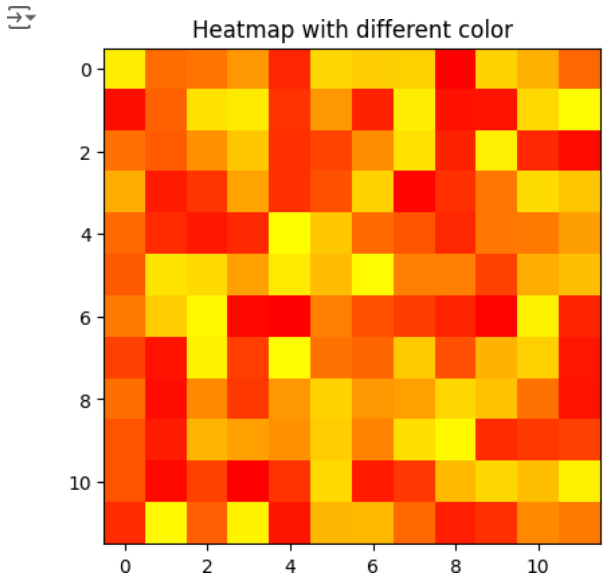
LABSHEET-9

HEATMAPS

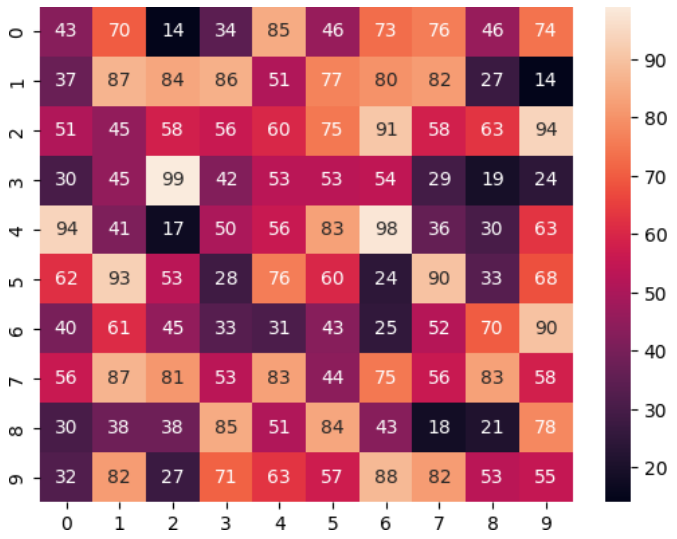
```
import numpy as np
import matplotlib.pyplot as plt data= np.random.random((13,16)) plt.imshow(
data,cmap="magma") plt.title("2D Heatmap")
plt.colorbar() plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt data=np.random.random((12,12)) plt.imshow(data,
cmap='autumn')
plt.title("Heatmap with different color") plt.show()
```

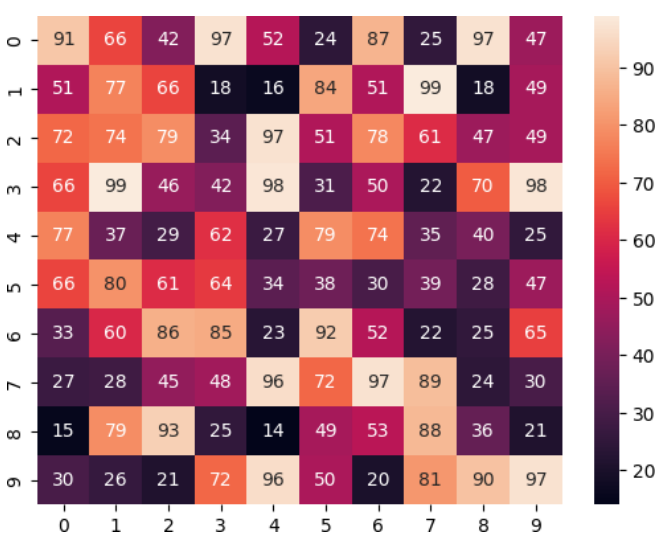



```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
data= np.random.randint(low=14,high=100, size=(10,10)) hm=sns.heatmap(data=data,
annot=True)
plt.show()
```




```
import pandas as pd import numpy as np
df=pd.read_csv('/content/train.csv')
```

```
df= np.random.randint(low=55,
high=60,
size=(8,8))
hm=sns.heatmap(data=data, annot=True) plt.show()
```



20201ISB0005

LABSHEET-10

SEABORN COLOR PALLETTES

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

sns.set(rc={"figure.figsize": (6,6)})
```

BUILDING COLOR PALLETTES

```
current_palette = sns.color_palette() sns.palplot(current_palette)
```

```
sns.palplot(sns.color_palette("hls", 8))
```

```
sns.palplot(sns.color_palette("husl", 8))
```

```
sample_colors = ["windows blue", "amber", "greyish", "faded green", "dusty purple", "pale
red", "medium green", "denim blue"] sns.palplot(sns.xkcd_palette(sample_colors))
```

```
sns.palplot(sns.color_palette("cubehelix", 8))
```

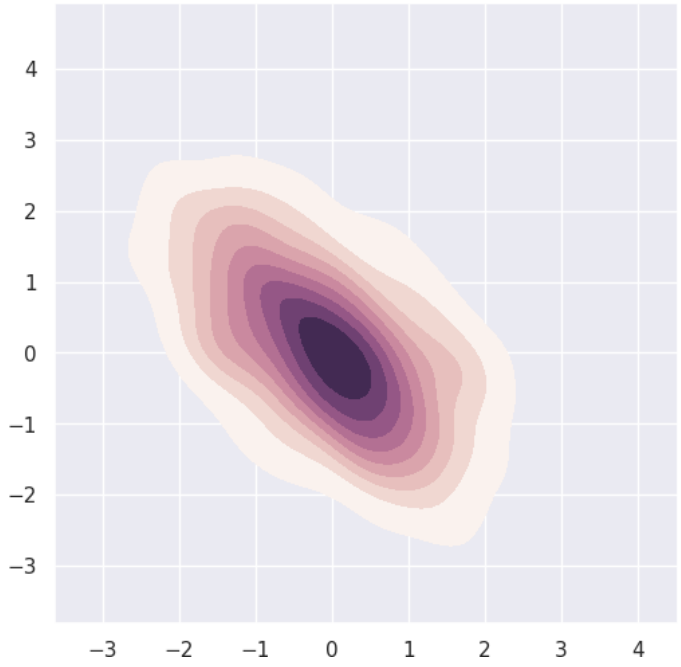
```
sns.palplot(sns.cubehelix_palette(8))
```

```
x,y = np.random.multivariate_normal([0,0], [[1,-.5],[-.5,1]], size=300).T sample_cmap =
sns.cubehelix_palette(light=1, as_cmap=True)
```

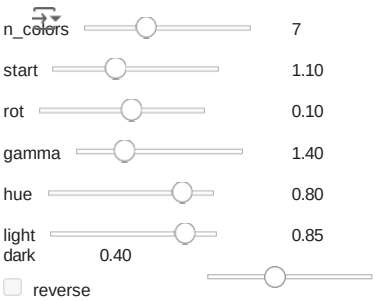
```
sns.kdeplot(x=x,y=y,cmap=sample_cmap, shade=True)
```

```
<ipython-input-16-534ef71d14c3>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
```

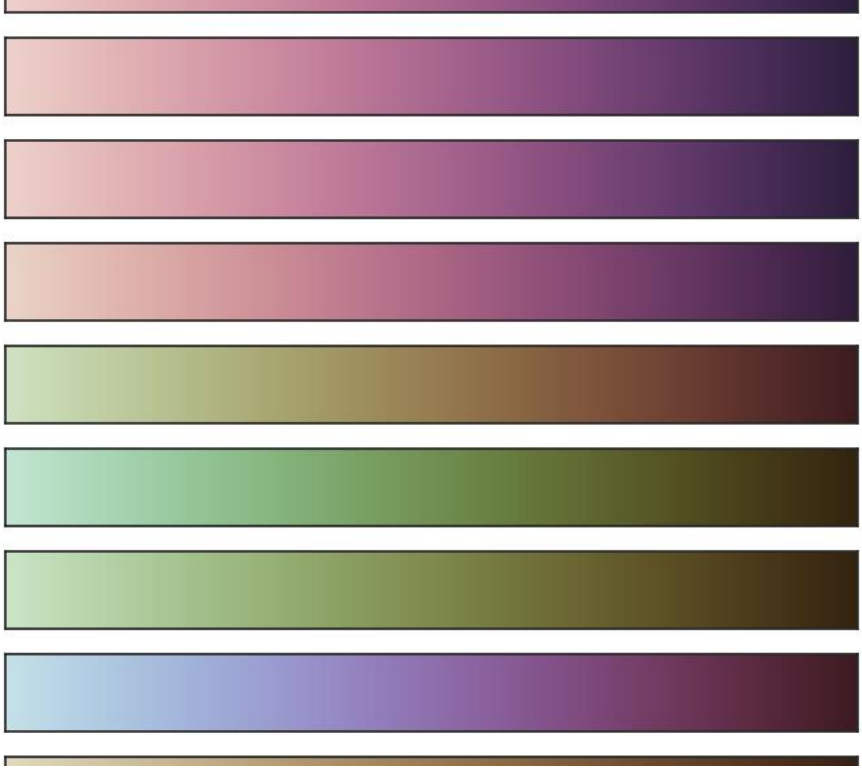
```
sns.kdeplot(x=x,y=y,cmap=sample_cmap, shade=True)
<Axes: >
```




```
sns.choose_cubehelix_palette(as_cmap=True)
```



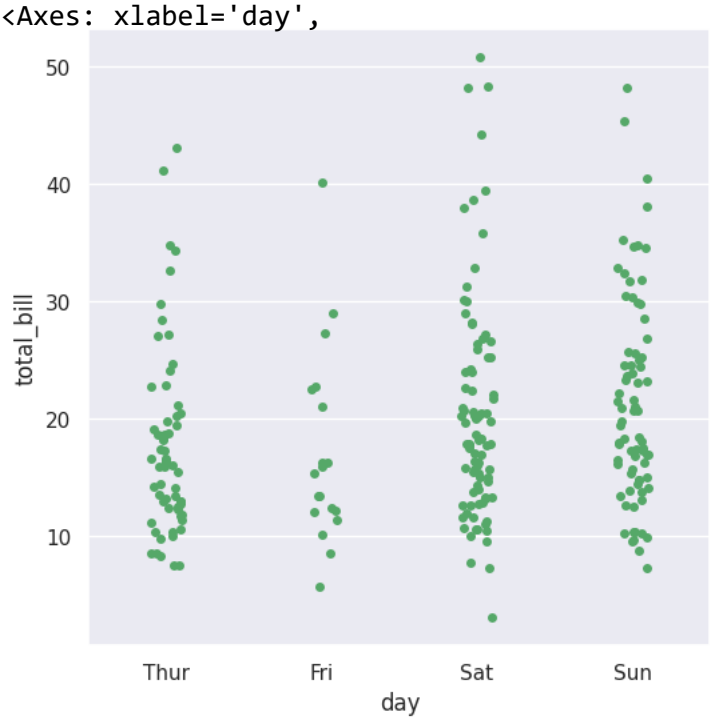
interactive under



```
sns.palplot(sns.cubehelix_palette(n_colors=8, start=1.7, rot=0.2, dark=0, light=.95,
reverse=True))
```

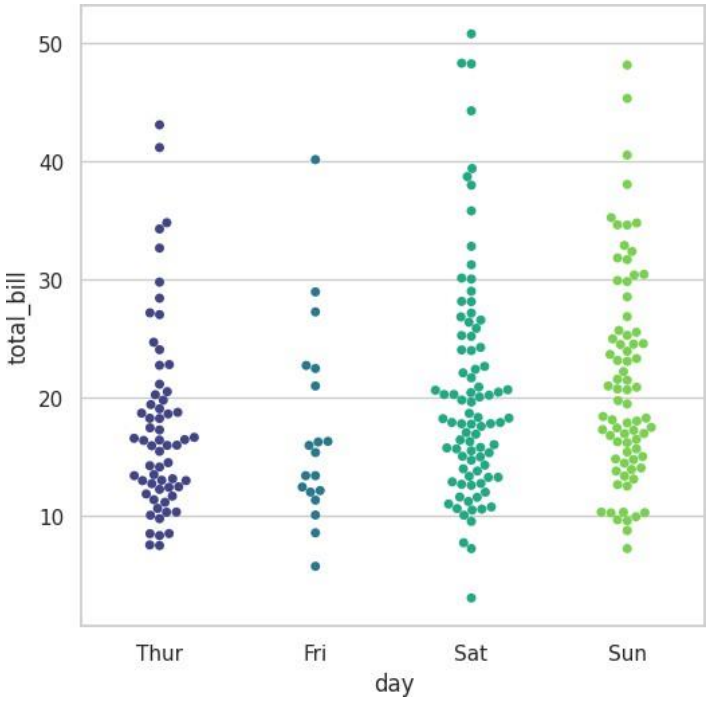
```
#loading built-in dataset

tips=sns.load_dataset("tips")
sns.stripplot(x="day", y="total_bill", data=tips, color="g")
```

```
<Axes: xlabel='day',
```

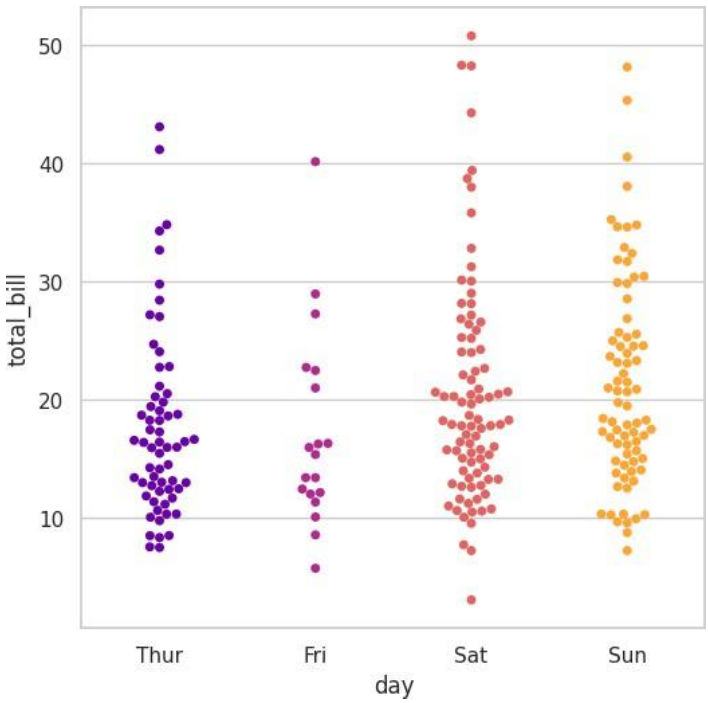



```
sns.set_style('whitegrid')
sns.swarmplot(x="day", y="total_bill", data=tips, palette="viridis")
```

```
<ipython-input-23-1576c2e5eda7>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.swarmplot(x="day", y="total_bill",
data=tips, palette="viridis")
<Axes: xlabel='day', ylabel='total_bill'>
```



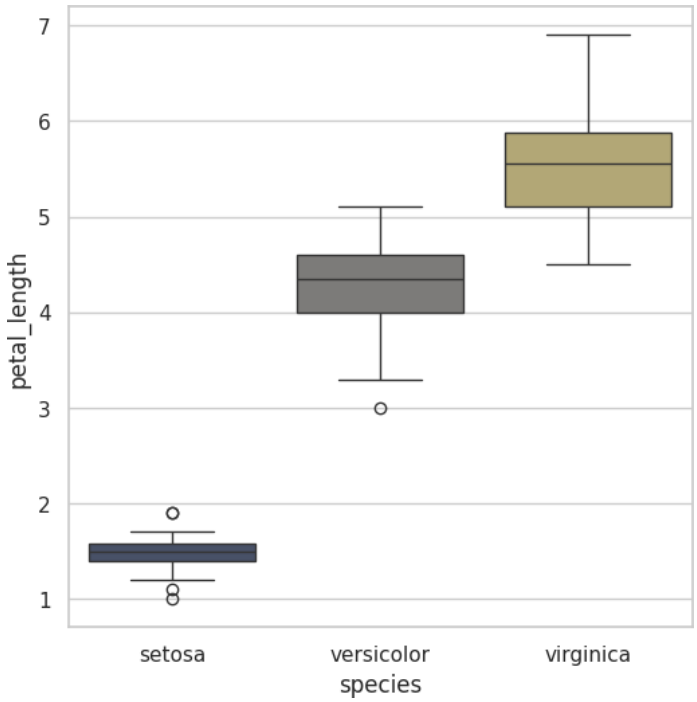

```
sns.set_style('whitegrid')
sns.swarmplot(x="day", y="total_bill", data=tips, palette="plasma")
<ipython-input-24-8931cda8de8a>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.swarmplot(x="day", y="total_bill",
data=tips, palette="plasma")
<Axes: xlabel='day', ylabel='total_bill'>
```




```
iris = sns.load_dataset("iris")
```

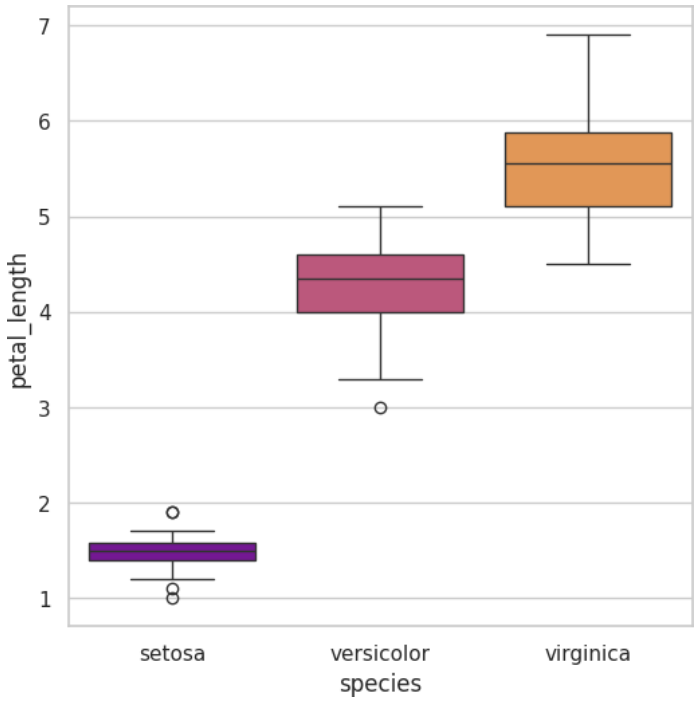
```
sns.boxplot(x="species", y="petal_length", data=iris, palette="cividis")
```

```
<ipython-input-26-cc37ff1b7cba>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.boxplot(x="species", y="petal_length",
data=iris, palette="cividis")
<Axes: xlabel='species', ylabel='petal_length'>
```



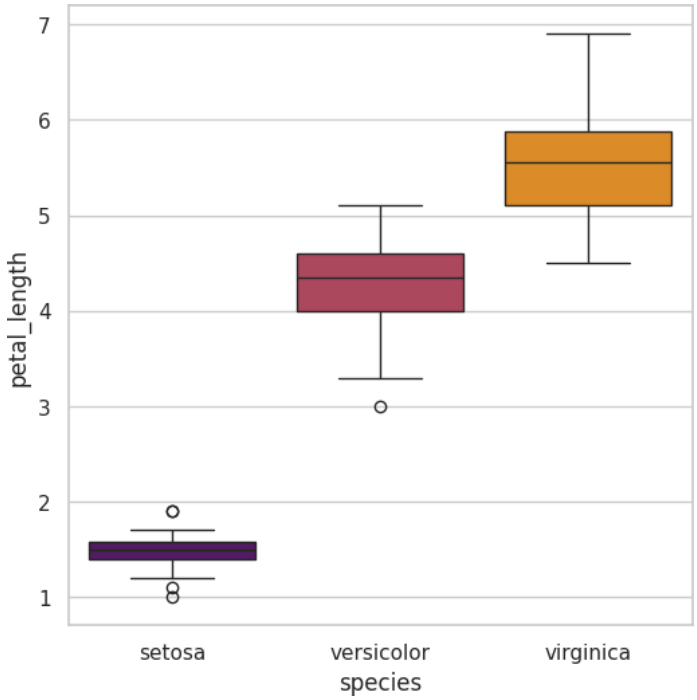

```
iris = sns.load_dataset("iris")
sns.boxplot(x="species", y="petal_length", data=iris, palette="plasma")
```

```
<ipython-input-27-0b4fe890c1f3>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.boxplot(x="species", y="petal_length",
data=iris, palette="plasma")
<Axes: xlabel='species', ylabel='petal_length'>
```



```
iris = sns.load_dataset("iris")
sns.boxplot(x="species", y="petal_length", data=iris, palette="inferno")
```

```
<ipython-input-28-e860428b94f7>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.boxplot(x="species", y="petal_length",
data=iris, palette="inferno")
<Axes: xlabel='species', ylabel='petal_length'>
```




```
iris = sns.load_dataset("iris")
sns.boxplot(x="species", y="petal_length", data=iris, palette="magma")
```

```
<ipython-input-29-ebb177fa7cb5>:2: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.boxplot(x="species", y="petal_length",
data=iris, palette="magma")
<Axes: xlabel='species', ylabel='petal_length'>
```

20201ISB0005

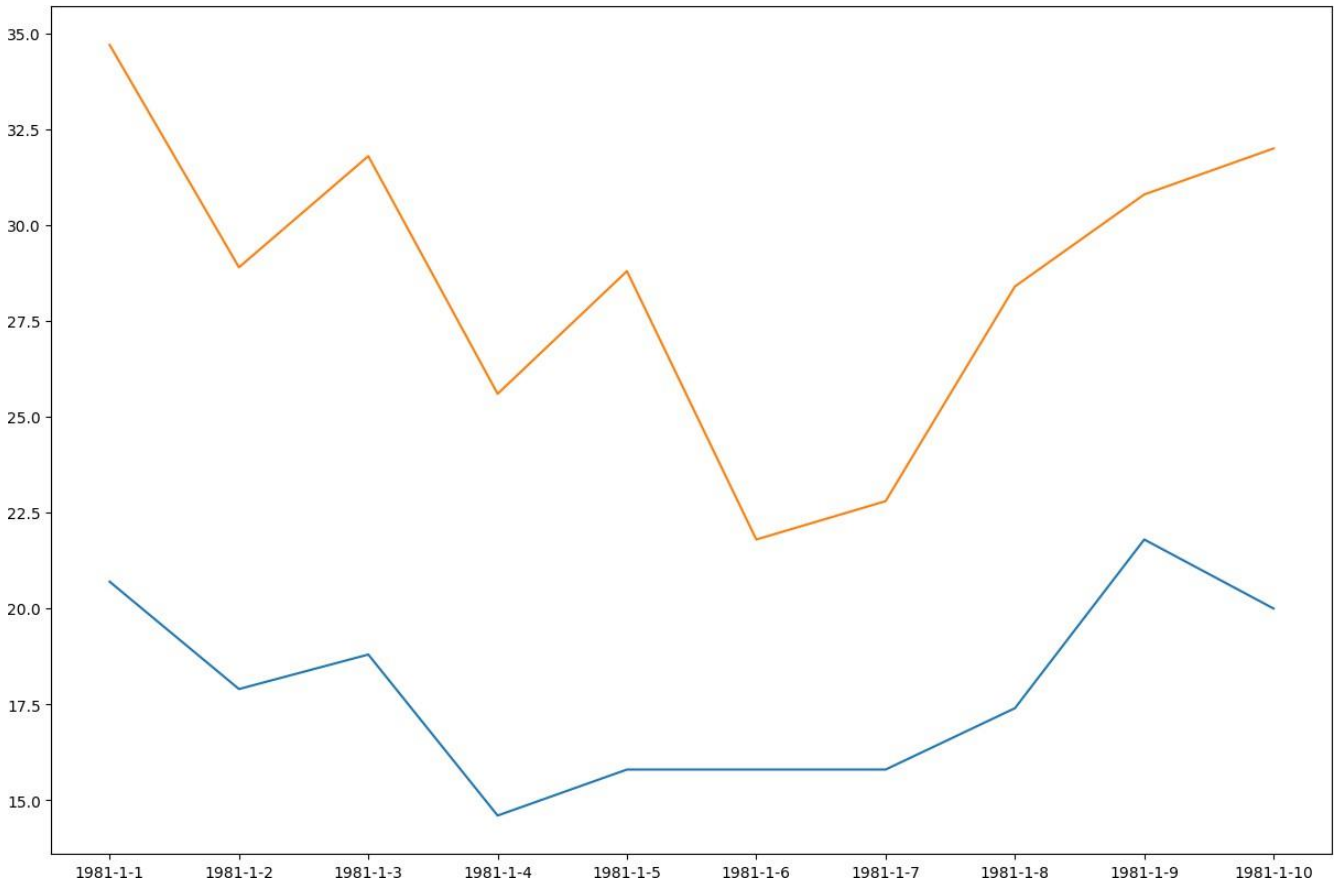
LABSHEET-11
MULTIVARIATE VISUALIZATION

Relational plots: relation b/w two variables categorical plots: categorical values are displayed
distribution plots: examining univariate and bivariate distributions matrix plots: array of scatterplots
Regression plots: emphasixe patterns in dataset during exploratory data analysis

```
import numpy as np import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure import seaborn as sns
%matplotlib inline
```

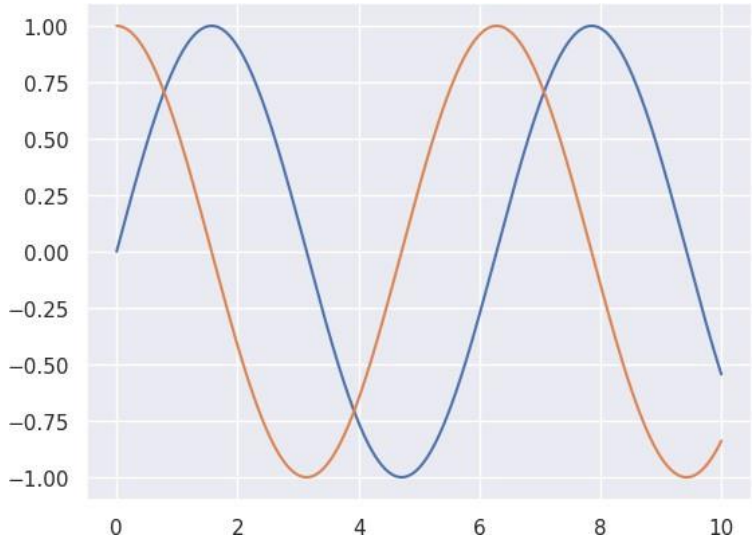
```
dates = ['1981-1-1', '1981-1-2','1981-1-3','1981-1-4','1981-1-5','1981-1-6','1981-1-7','1981-1-8','1981-1-9','1981-1-10']
min_temperature = [20.7,17.9,18.8,14.6,15.8,15.8,15.8,17.4,21.8,20.0]
max_temperature = [34.7,28.9,31.8,25.6,28.8,21.8,22.8,28.4,30.8,32.0]

fig,axes = plt.subplots(nrows=1, ncols=1, figsize=(15,10))
axes.plot(dates,min_temperature, label='Min temperature')
axes.plot(dates,max_temperature, label='Max temperature') axes.legend
```

```
matplotlib.axes._axes.Axes.legend
def legend(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_axes.py
Place a legend on the Axes.

Call signatures::
```



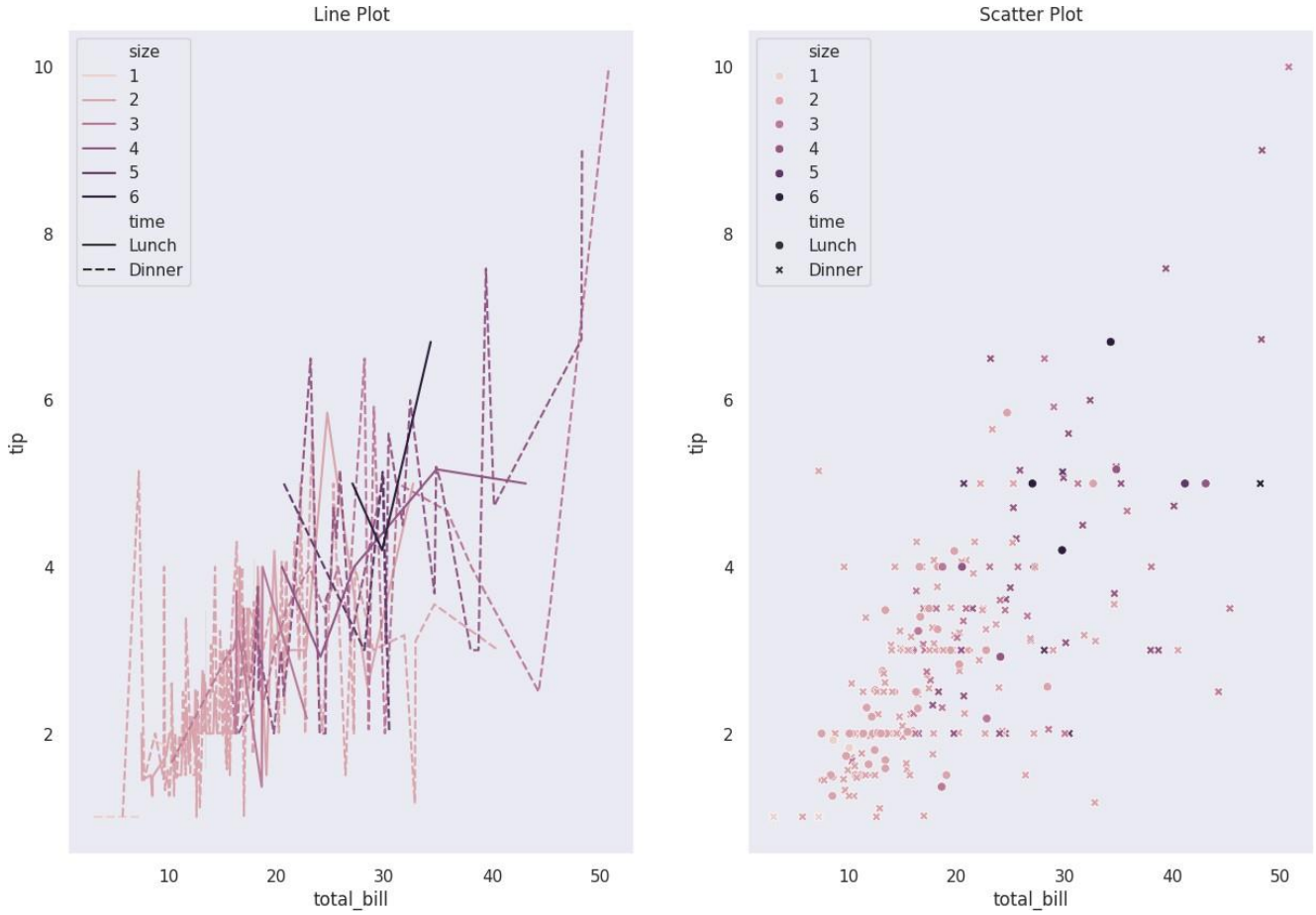
```
sns.set()


x = np.linspace(0,10,1000)
plt.plot(x, np.sin(x), x, np.cos(x))
```

```
[<matplotlib.lines.Line2D at 0x7e3acaaaffa0>,
<matplotlib.lines.Line2D at 0x7e3acaae0040>]
```




```
sns.set(style="dark")
fig, ax = plt.subplots(ncols=2, nrows=1, figsize=(15,10)) df= sns.load_dataset("tips")

print(df.head())

sns.lineplot(x="total_bill", y="tip", hue="size", style= "time",
data=df,ax=ax[0]).set_title("Line Plot")
sct_plt = sns.scatterplot(x="total_bill", y="tip", hue="size", style="time", data=df,
ax=ax[1]).set_title("Scatter Plot") sct_plt.figure.savefig('Scatter_plot1.png')
print('Plot Saved')
```

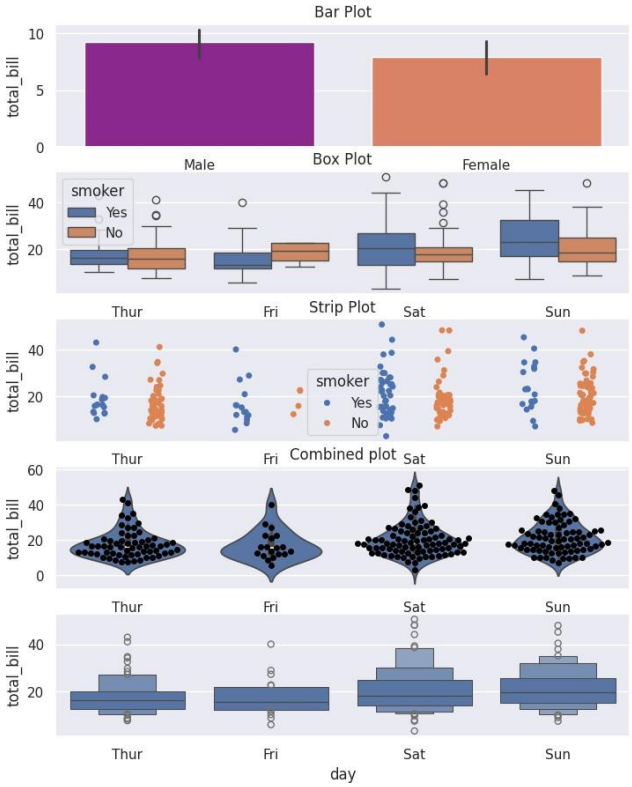
```
    total_bill  tip    sex smoker  day    time  size
16.99  1.01  Female         No  Sun  Dinner    2
10.34  1.66        Male     No  Sun  Dinner    3
21.01  3.50        Male     No  Sun  Dinner    3
23.68  3.31        Male     No  Sun  Dinner    2
24.59  3.61  Female         No  Sun  Dinner    4

Plot Saved
```

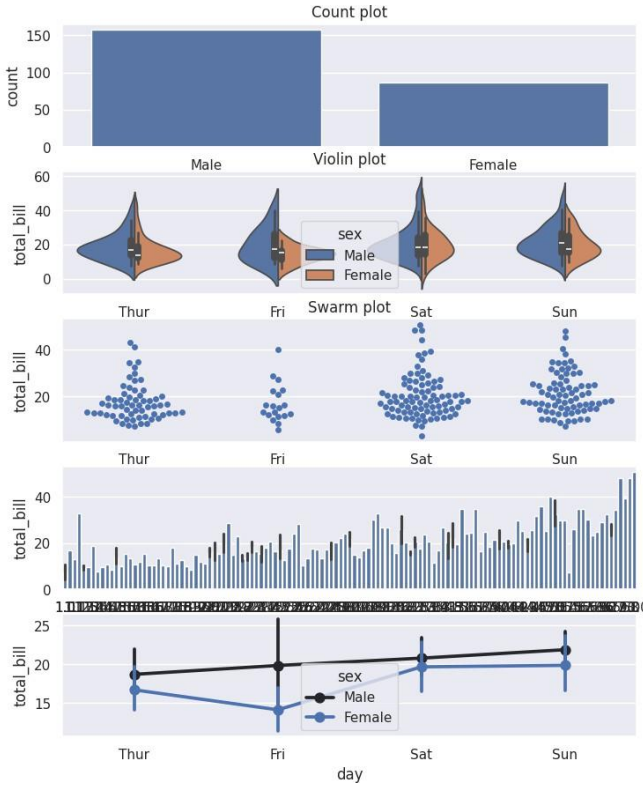


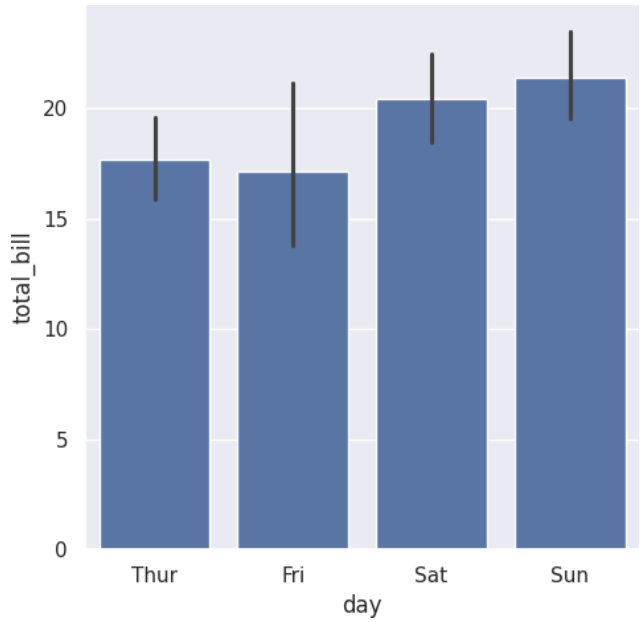

```
sns.set_style('darkgrid')
fig, ax = plt.subplots(nrows=5, ncols=2) fig.set_size_inches(18.5, 10.5)

df=sns.load_dataset('tips')

sns.barplot(x='sex', y='total_bill', data=df, palette='plasma', estimator= np.std,
ax=ax[0,0]).set_title('Bar Plot') sns.countplot(x='sex', data=df,
ax=ax[0,1]).set_title('Count plot')
sns.boxplot(x='day', y='total_bill', data=df, hue='smoker', ax=ax[1,0]).set_title('Box
Plot')

sns.violinplot(x='day', y='total_bill', data=df, hue='sex', split= True,
ax=ax[1,1]).set_title('Violin plot')

sns.stripplot(x='day', y='total_bill', data=df, jitter= True, hue='smoker', dodge=True,
ax=ax[2,0]).set_title('Strip Plot') sns.swarmplot(x='day', y='total_bill', data=df,
ax=ax[2,1]).set_title('Swarm plot')
sns.violinplot(x='day', y='total_bill', data=df, ax=ax[3,0])
sns.swarmplot(x='day',y='total_bill',data=df, color='black',
ax=ax[3,0]).set_title('Combined plot') sns.barplot(x='tip',y='total_bill', data=df,
ax=ax[3,1])
sns.boxenplot(x="day", y="total_bill", color="b", scale="linear", data=df, ax=ax[4,0])
```

```
sns.pointplot(x="day", y="total_bill", color="b", hue="sex", data=df, ax=ax[4,1])
sns.catplot(x='day',y='total_bill',data=df, kind='bar')
```

```
<ipython-input-6-79e72dcff921>:7: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `l sns.barplot(x='sex', y='total_bill', data=df,
palette='plasma', estimator= np.std, ax=ax[0,0]).set_title('Bar Plot')
<ipython-input-6-79e72dcff921>:24: FutureWarning:

The `scale` parameter has been renamed to `width_method` and will be removed in v0.15.
Pass `width_method='linear' for the same eff sns.boxenplot(x="day", y="total_bill",
color="b", scale="linear", data=df, ax=ax[4,0])
<ipython-input-6-79e72dcff921>:26: FutureWarning:
Setting a gradient palette using color= is deprecated and will be removed in v0.14.0. Set
`palette='dark:b'` for the same effect. sns.pointplot(x="day", y="total_bill", color="b",
hue="sex", data=df, ax=ax[4,1])
<seaborn.axisgrid.FacetGrid at 0x7e3ac3b802e0>
```





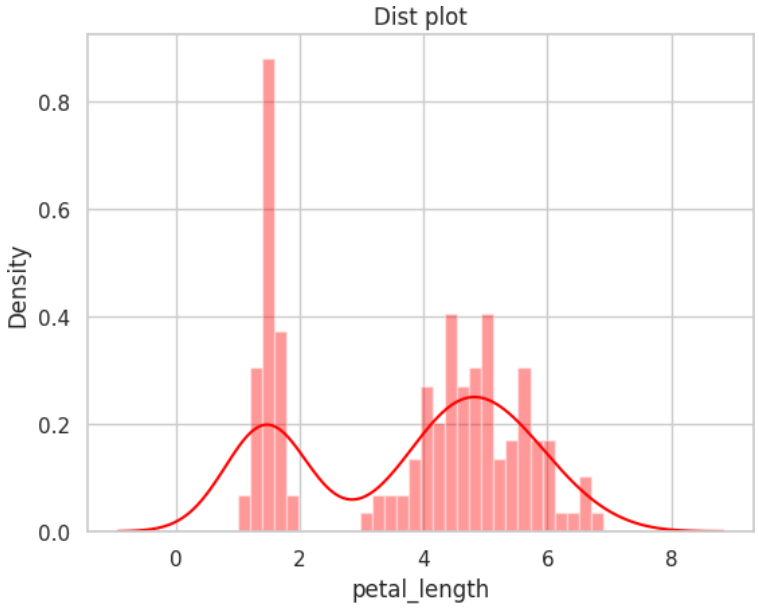


```
sns.set_style('whitegrid')
#loading the dataset directly without any files df=sns.load_dataset('iris')
print(df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0     5.1   3.5   1.4   0.2  setosa
1     4.9   3.0   1.4   0.2  setosa
2     4.7   3.2   1.3   0.2  setosa
3     4.6   3.1   1.5   0.2  setosa
4     5.0   3.6   1.4   0.2  setosa
```

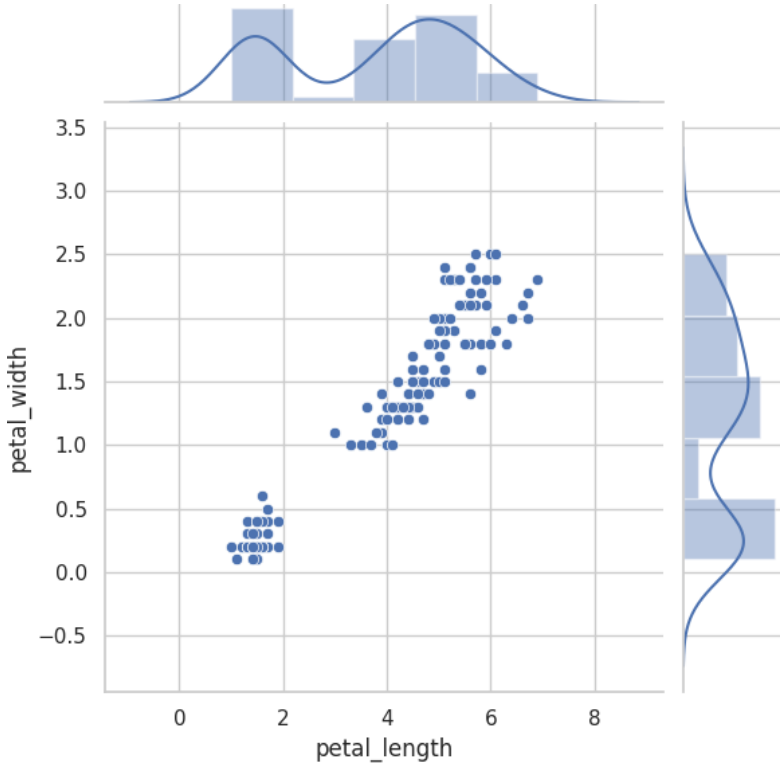
```
sns.distplot(df['petal_length'], kde=True, color='red', bins=30).set_title('Dist plot')
```

```
<ipython-input-8-6c2fae3a6ad9>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

sns.distplot(df['petal_length'], kde=True, color='red', bins=30).set_title('Dist plot')
Text(0.5, 1.0, 'Dist plot')
```



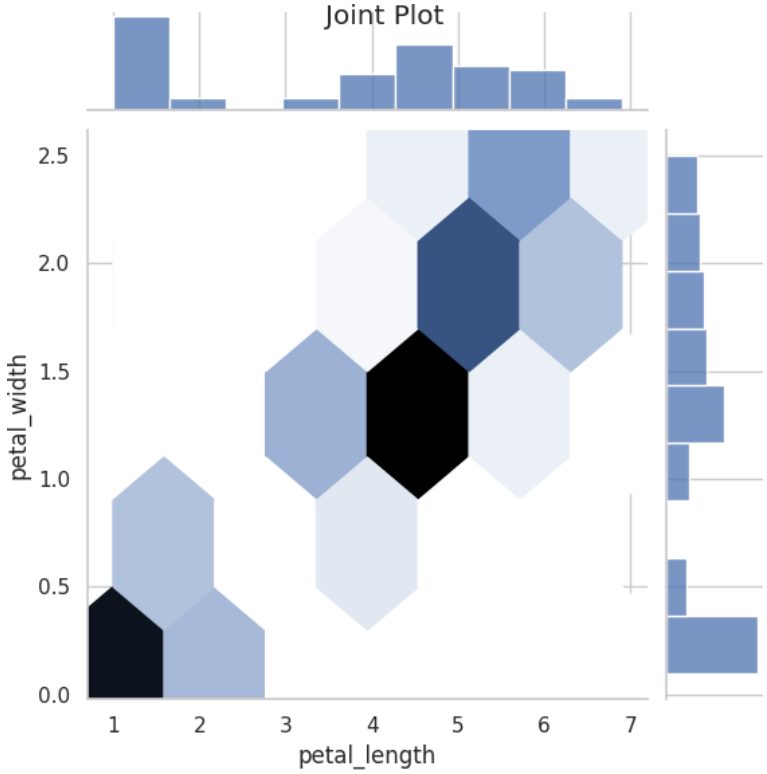

```
jointgrid = sns.JointGrid(x='petal_length', y='petal_width', data=df)
jointgrid.plot_joint(sns.scatterplot)
jointgrid.plot_marginals(sns.distplot)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:1886: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

func(self.x, **orient_kw_x, **kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:1892: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

func(self.y, **orient_kw_y, **kwargs)
<seaborn.axisgrid.JointGrid at 0x7e3b00f8d120>
```




```
g=sns.jointplot(x='petal_length', y= 'petal_width', data=df, kind='hex')
g.fig.suptitle('Joint Plot')
```

```
Text(0.5, 0.98, 'Joint Plot')
```

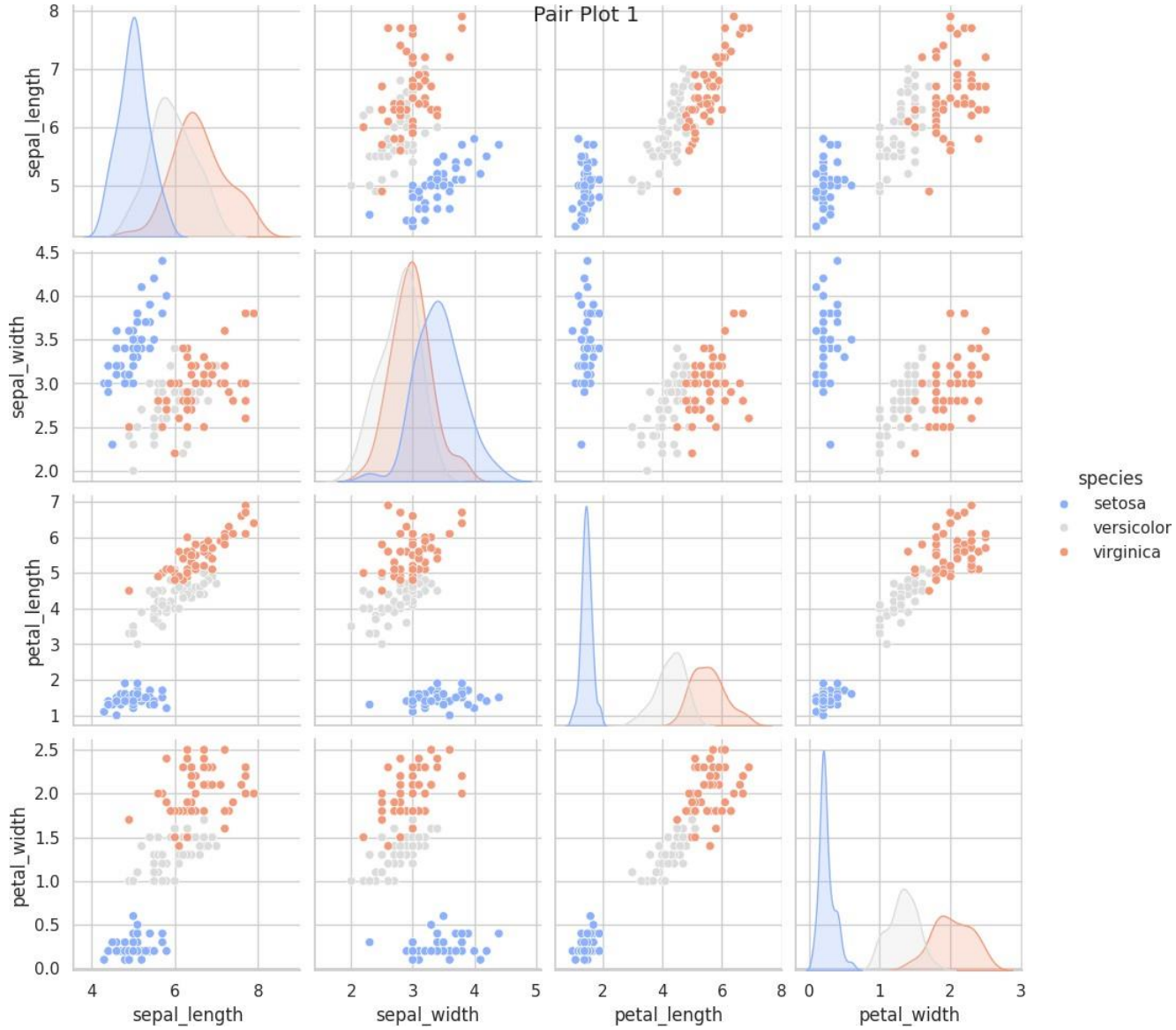



```
g=sns.pairplot(df, hue="species", palette= 'coolwarm') g.fig.suptitle("Pair Plot 1")
g.add_legend
```

```
seaborn.axisgrid.Grid.add_legend
def add_legend(legend_data=None, title=None, label_order=None,
adjust_subtitles=False, **kwargs)
```

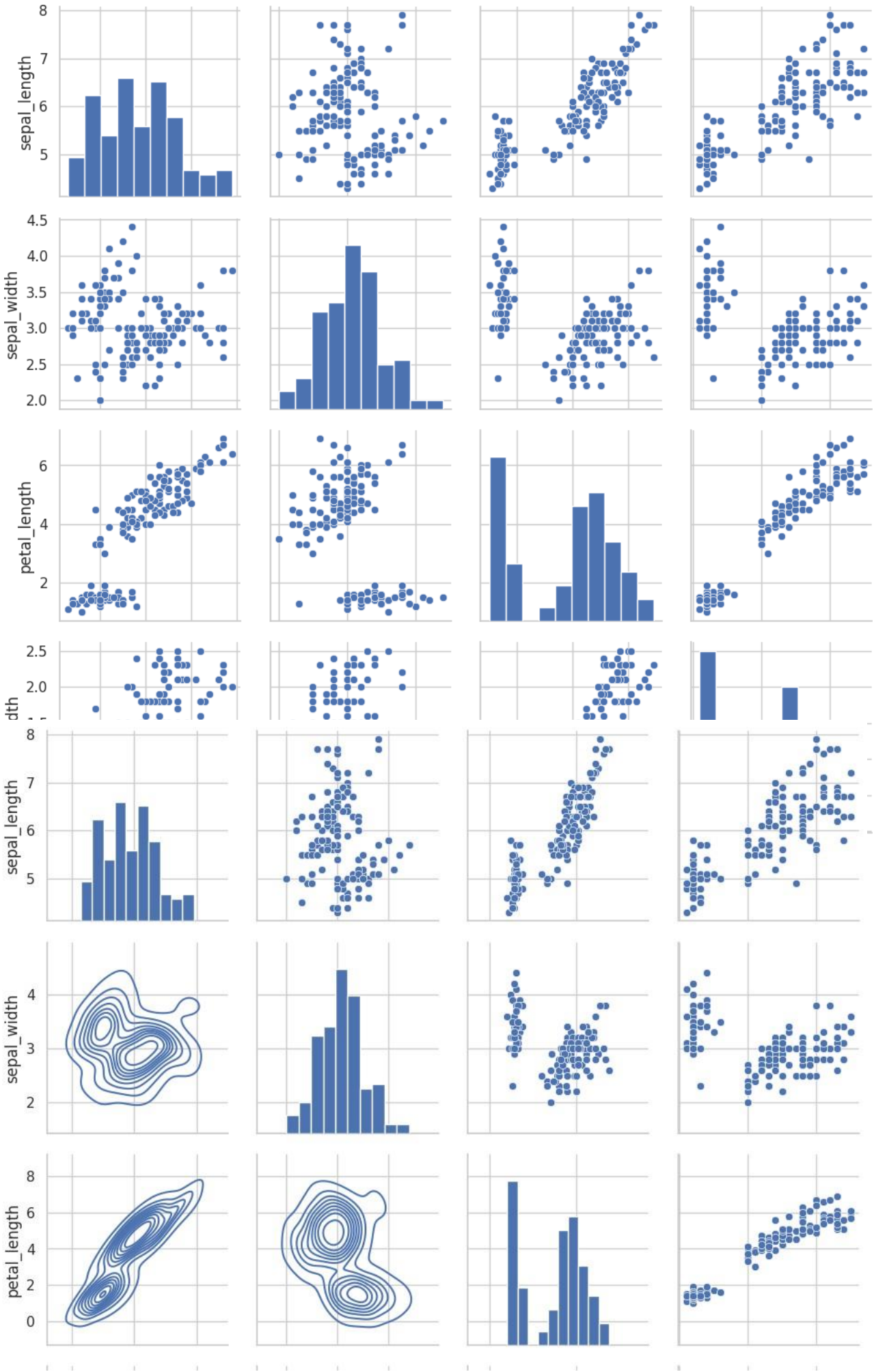
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py

```
Draw a legend, maybe placing it outside axes and resizing the
figure.

Parameters
----------
legend_data : dict
```




```
pairgrid= sns.PairGrid(data=df)
pairgrid= pairgrid.map_offdiag(sns.scatterplot) pairgrid= pairgrid.map_diag(plt.hist)

pairgrid = sns.PairGrid(data=df)
pairgrid = pairgrid.map_upper(sns.scatterplot) pairgrid = pairgrid.map_diag(plt.hist)
pairgrid = pairgrid.map_lower(sns.kdeplot)
```

20201ISB0005



20201ISB0005

LABSHEET-12

TEXT VISUALIZATION

```
import pandas as pd
import matplotlib.pyplot as plt from wordcloud import WordCloud from wordcloud import
STOPWORDS

df= pd.read_csv('/content/netflix titles.csv', usecols=['cast']) df.head()
```

```
cast
NaN
Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...
Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...
NaN
Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...
```

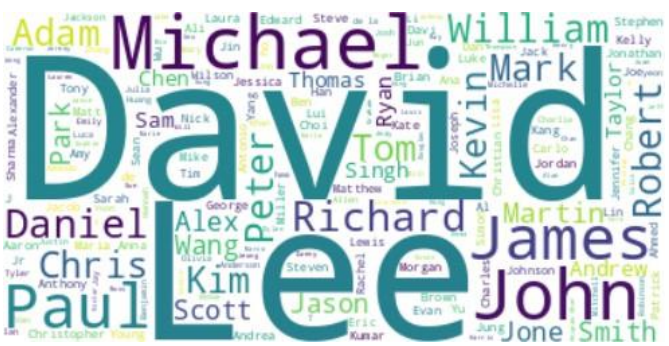
```
ndf=df.dropna() ndf.head()
```

```
cast
Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...
Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...
Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...
Kate Siegel, Zach Gilford, Hamish Linklater, H...
Vanessa Hudgens, Kimiko Glenn, James Marsden, ...
```

```
text= " ".join(item for item in ndf['cast']) print(text)
```

```
Ama Qamata, Khosi Ngema, Gail Mabalane, Thabang Molaba, Dillon Windvogel, Natasha
Thahane, Arno Greeff, Xolile Tshabalala, Getmore
```

```
stopwords = set(STOPWORDS)

wordcloud = WordCloud(background_color="White").generate(text) plt.imshow(wordcloud,
interpolation= 'bilinear')
plt.axis("off")
plt.margins(x=0, y=0) plt.show()
```




```
wordcloud = WordCloud(background_color="White", max_words=100, max_font_size=300, width=
800, height=500, colormap="magma").generate(te plt.figure(figsize=(20,20))
plt.imshow(wordcloud, interpolation= 'bilinear') plt.axis("off")
plt.margins(x=0, y=0) plt.show()
```


Christopher
Kim
Lin
Choi
Ben
Daniel
William
Chen
Charles
Kang
Adam
Andrew
Maria
J
Dan
Amy
Joe
Lee
John
Chris
Tom
Tim
Davi
Morgan
Yang
Jessica
Kevin
Alex
Park
Scott
James
Andrea
Ahmed
Kelly
Evan
David
Peter
Robert
Paul
Jordan
Matthew
Al
Jason
Martin
Brian
Aaron
Han
Taylor
Singh
Johnson
de
Patrick
Sarah
Anthony
Jin
Wu
Ali
Yu
Jennifer
Jone
Mike
Steve
Anna
Jr
George
Young
Sean
Lui
Jonathan
Alexander
Thomas
Mark
Jack
Richard
Ryan
Steven
Smith
ho
Laura
Miller
Brown
Kate
Edward
Carlo
Stephen
Kumar
Sharma
Wang
Christian
Nick
Matt
Jung
Sam
Lewis
Wilson
Tony
Luke

20201ISB0005

LABSHEET-13

TIME SERIES DATA

A time series is the series of data points listed in time order.
A time series is a sequence of successive equal interval points in time.
A time-series analysis consists of methods for analyzing time series data in order to extract meaningful insights and other useful characteristics of data.
For performing time series analysis download stock_data.csv

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


# reading the dataset using read_csv
df = pd.read_csv("/content/stock_data.csv", parse_dates=True, index_col="Date")

# displaying the first five rows of dataset df.head()
```

Date	Open	High	Low	Close	VolumeName	
2006-01-03	39.69	41.22	38.79	40.91	24232729	AABA
2006-01-04	41.22	41.90	40.77	40.97	20553479	AABA
2006-01-05	40.93	41.73	40.85	41.53	12829610	AABA
2006-01-06	42.88	43.57	42.80	43.21	29422828	AABA
2006-01-09	43.10	43.66	42.82	43.42	16268338	AABA

Next steps: Generate code with df | View recommended plots

We have used the 'parse_dates' parameter in the read_csv function to convert the 'Date' column to the DatetimeIndex format. By default, Dates are stored in string format which is not the right format for time series data analysis.

Now, removing the unwanted columns from dataframe i.e. 'Unnamed: 0'.

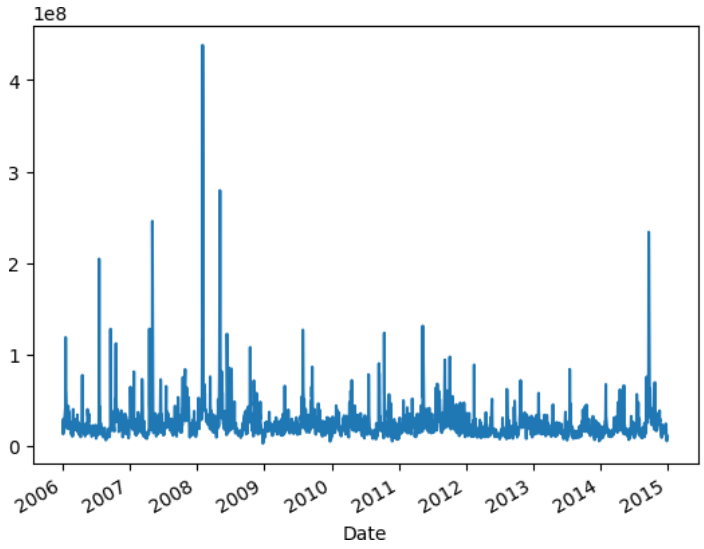
```
# deleting column
df=df.drop(columns='Name') print(df)
```

```
              Open   High    Low  Close    Volume
Date
2006-01-03   39.69  41.22  38.79  40.91  24232729
2006-01-04   41.22  41.90  40.77  40.97  20553479
2006-01-05   40.93  41.73  40.85  41.53  12829610
2006-01-06   42.88  43.57  42.80  43.21  29422828
2006-01-09   43.10  43.66  42.82  43.42  16268338
...            ...    ...    ...    ...       ...
2014-12-23   51.46  51.46  49.93  50.02  15514036
2014-12-24   50.19  50.92  50.19  50.65   5962870
2014-12-26   50.65  51.06  50.61  50.86   5170048
2014-12-29   50.67  51.01  50.51  50.53   6624489
2014-12-30   50.35  51.27  50.35  51.22  10703455

[2263 rows x 5 columns]
```

Example 1: Plotting a simple line plot for time series data.

```
df['Volume'].plot()
```

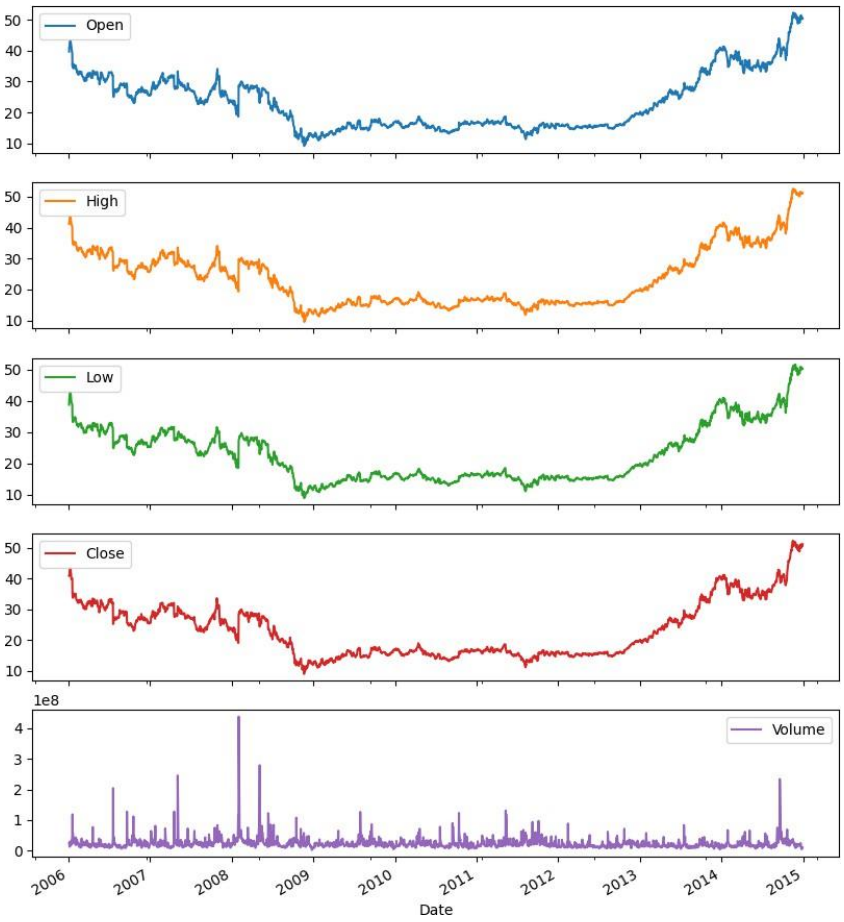
```
<Axes: xlabel='Date'>
```



Example 2: Now let's plot all other columns using subplot.

```
df.plot(subplots=True, figsize=(10, 12))
```

```
array([<Axes: xlabel='Date'>, <Axes: xlabel='Date'>,
<Axes: xlabel='Date'>, <Axes: xlabel='Date'>,
<Axes: xlabel='Date'>], dtype=object)
```

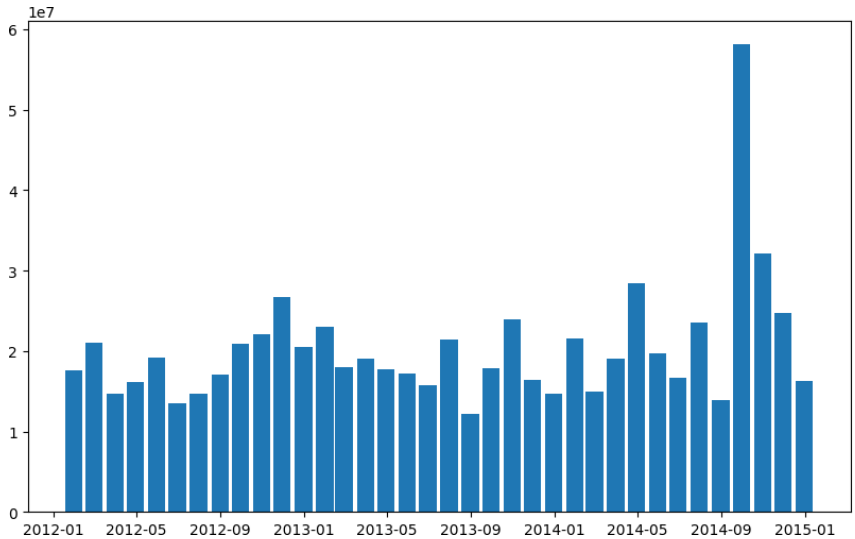


Resampling: Resampling is a methodology of economically using a data sample to improve the accuracy and quantify the uncertainty of a population parameter. Resampling for months or weeks and making bar plots is another very simple and widely used method of finding seasonality. Here we are going to make a bar plot of month data for 2016 and 2017.
Example 3:

```
# Resampling the time series data based on monthly 'M' frequency df_month =
df.resample("M").mean()
print(df_month)

# using subplot
fig, ax = plt.subplots(figsize=(10, 6))

# plotting bar graph
ax.bar(df_month['2012':'2014'].index, df_month.loc['2012':'2014', "Volume"],width=25,
align='center')
```

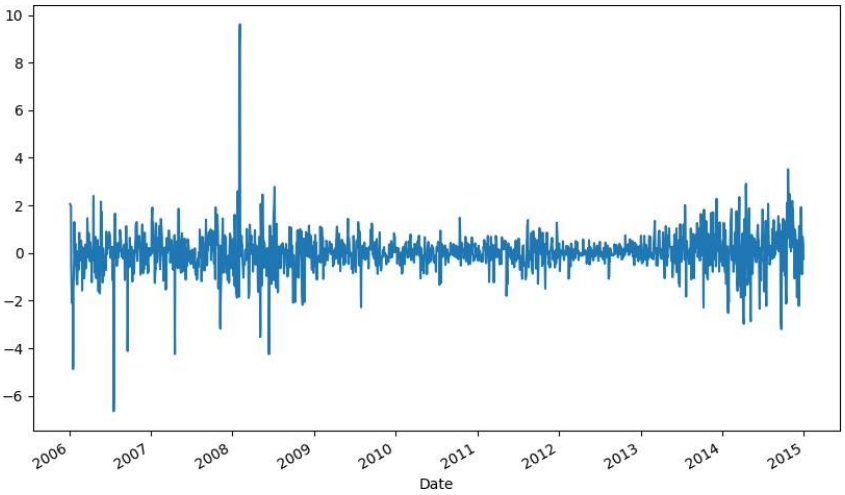
Date	Open	High	Low	Close	Volume	Change
2006-01-31	38.245500	38.694000	37.641500	38.113000	3.400594e+07	0.991442
2006-02-28	33.141579	33.436842	32.627368	32.975789	2.329848e+07	0.996423
2006-03-31	31.333478	31.696957	30.929130	31.218696	2.095522e+07	1.000390
2006-04-30	32.383684	32.790000	31.914737	32.283158	2.200768e+07	1.001098
2006-05-31	31.744545	32.175455	31.171364	31.517273	2.218047e+07	0.998535
...	...	...	...	...	...	...
2014-08-31	36.836190	37.150000	36.545238	36.876667	1.396539e+07	1.003530
2014-09-30	40.662857	41.270000	39.983810	40.671905	5.811769e+07	1.003005
2014-10-31	41.253043	41.886087	40.784783	41.393913	3.210848e+07	1.005501
2014-11-30	49.879474	50.553158	49.440000	50.151579	2.474402e+07	1.006233
2014-12-31	50.359524	50.975714	49.852857	50.331905	1.623090e+07	0.999653

[108 rows x 6 columns]
<BarContainer object of 36 artists>



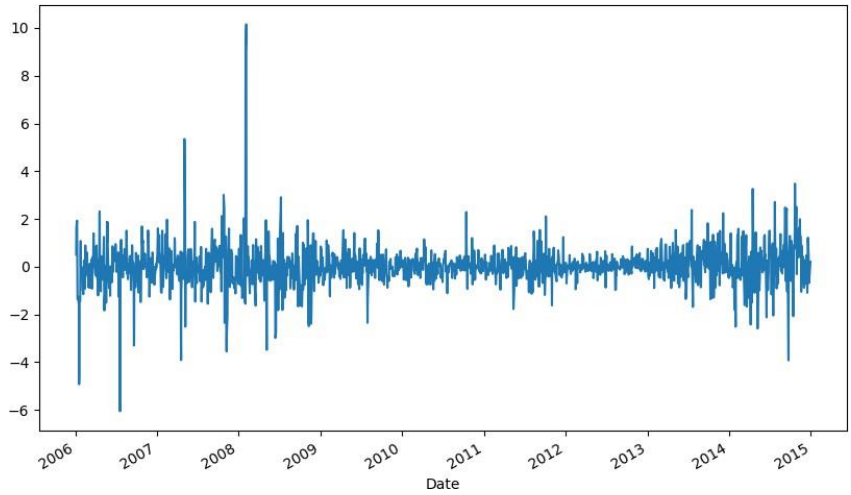
Differencing: Differencing is used to make the difference in values of a specified interval. By default, it's one, we can specify different values for plots. It is the most popular method to remove trends in the data.

```
df.Low.diff(2).plot(figsize=(10, 6))
```




```
df.High.diff(2).plot(figsize=(10, 6))
```

```
<Axes: xlabel='Date'>
```



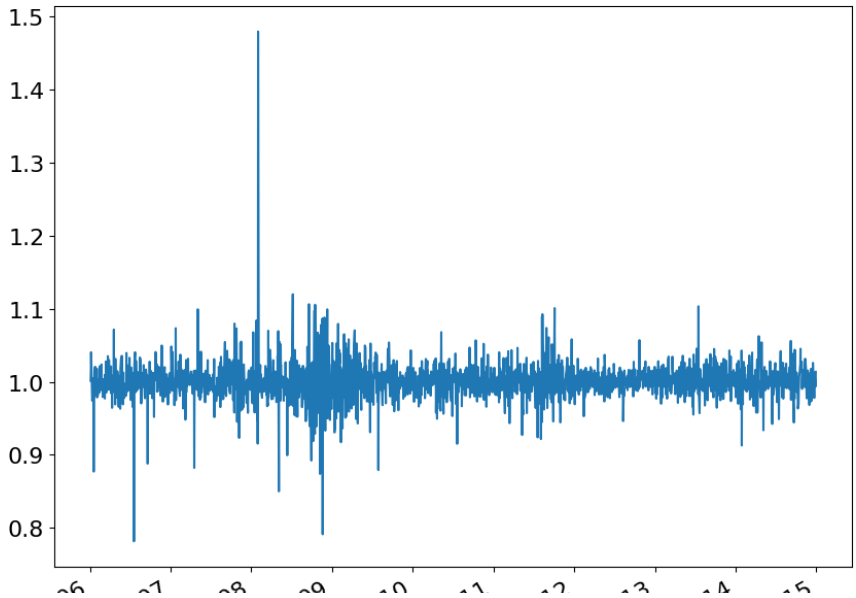
Plotting the Changes in Data

We can also plot the changes that occurred in data over time. There are a few ways to plot changes in data.

Shift: The shift function can be used to shift the data before or after the specified time interval. We can specify the time, and it will shift the data by one day by default. That means we will get the previous day's data. It is helpful to see previous day data and today's data simultaneously side by side.

```
df['Change'] = df.Close.div(df.Close.shift())
```

```
df['Change'].plot(figsize=(10, 8), fontsize=16)
```



.div() function helps to fill up the missing data values. Actually, div() means division.

If we take df. div(6) it will divide each element in df by 6.

We do this to avoid the null or missing values that are created by the 'shift()' operation.

```
df['Change'].plot(figsize=(10, 6))
```

```
<Axes: xlabel='Date'>
```