FOUNDATIONS OF DATA SCIENCE LABORATORY

LABORATORY MANUAL

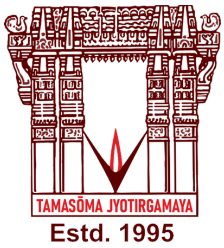
B.TECH

(III YEAR – I SEM)

(2023-2024)

DEPARTMENT OF CSE - (CyS, DS) and AI&DS

(ARTIFICIAL INTELLIGENCE AND DATA SCIENCE)



VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI
INSTITUTE OF ENGINEERING AND TECHNOLOGY

INDEX

INDEX

EXPERIMENT NO: Week 1 & 2

1. Create NumPy arrays from Python Data Structures, Intrinsic NumPy objects and Random Functions.

```
# using 1d list
ar = np.array([1,2,3])
ar
OUTPUT:
array([1, 2, 3])

# using 2d list
ar = np.array([[1,2,3],[4,5,6]])
ar
OUTPUT:
array([[1, 2, 3],
[4, 5, 6]])

# passing Lst
Ist = [1,2,3,-1,0]
np.array(1st)
OUTPUT:
array([ 1, 2, 3, -1, 0])

# using tuple
tup = (1,2,3)
np.array(tup)
OUTPUT:
array([1, 2, 3])
```

```
# using arange keyword
np.arange(1, 20)
```

OUTPUT:

```
array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
# using arange keyword
np.arange(1, 28,2)
```

OUTPUT:

```
array([ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19])
```

```
# using random.randint() function
np.random. randint(10,size=10)
```

OUTPUT:

```
array([8, 8, 4, 4, 6, 6, 1, 0, 5, 4])
```

```
# finding shape of an array
arr = np.array([[1,2,3],[4,5,6]])
arr. shape
```

OUTPUT:

```
(2, 3)
```

```
# creating numpy arrays using zeros() method
aril = np.zeros(3)
ar2 = np.zeros([2,2])
print(ar1)
print (ar2)
```

OUTPUT:

```
[[0. 0. 0.]
```

```
 [0. 0.]
[0. 0.]]

# creating numpy arrays using ones() method
ar1 = np.ones(3)
ar2 = np.ones([3,3])
print (art)
print(ar2)
```

OUTPUT:

```
[1. 1. 1.]
[[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]]
```

2. Manipulation of NumPy arrays- Indexing, Slicing, Reshaping, Joining and Splitting.

```
# Indexing 1D array
ar = np.arange(@, 20,2)
newar = ar[np.array([@, 2, 3])]
print(ar)
print('Elements at indexes @, 2 and 3 are: ')
print (newar)
```

OUTPUT:

```
[0 2 4 6 8 10 12 14 16 18]
Elements at indexes 0, 2 and 3 are:
[0 4 6]
```

```
# Indexing 2D array
```

```
a = np.array([[0 ,1 ,2],[3 ,4 ,5]
[6 ,7 ,8],[9 ,10 ,11]])
print(a)
print( "Indexing 2d array’)
print(a[[0 ,1 ,2 ],[0 ,0 ,1]])
```

OUTPUT:

```
[[0 1 2]
[3 4 5]
[6 7 8]
[ 9 10 11]]
Indexing 2d array
[0 3 7]
```

```
# Slicing 1D array
a = np.arange(20)
print("\n Array is:\n ",a)
# a[start:stop:step]
print("\n a[-8:17:1] = ",a[-8:17:1])
print("\n a[10:] = ",a[10:])
```

OUTPUT:

```
Array is:
[0 1 2 3 4 5 6 7 8 91011 12 13 14 15 16 17 18 19]
a[-8:17:1] = [12 13 14 15 16]
a[10:] = [10 11 12 13 14 15 16 17 18 19]
```

```
# Slicing 2D array
a = np.array([[9, 1, 2, 3, 4, 5],
[6, 7, 8, 9, 18, 11],
(12, 13, 14, 15, 16, 17],
```

```
[18, 19, 20, 21, 22, 23],
[24, 25, 26, 27, 28, 29],
[30, 31, 32, 33, 34, 35]])
print("\n Array is:\n ",a)
# slicing and indexing
print("\n a[@, 3:5] = ",a[@, 3:5])
print("\n a[4:, 4:] = \n")
print(a[4:, 4:])
print("\n a[:, 2] = ",a[:, 2])
```

OUTPUT:

```
Array is:
[01 2 3 4 5]
[6 7 8 9 10 11]
[12 13 14 15 16 17]
[18 19 20 21 22 23]
[24 25 26 27 28 29]
[30 31 32 33 34 35]]
ae, 3:5] = [3 4]
a[4:, 4:] =
[[28 29]
[34 35]]
a[:, 2] = [2 8 14 20 26 32]
```

```
# Reshaping 1D to 2D
ar = np.array([1,2,3,4,5,6,7,8,9])
ar.reshape(3,3)
```

OUTPUT:

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

```
 # Reshaping 1D to 3D
a3 = np.arange(12)
a3.reshape(2, 3, 2)
OUTPUT:
array([[[ 0, 1],
[ 2, 3],
[ 4, 5]],
[[ 6, 7],
[ 8, 9],
[10, 11]]])

# Joining 2-1D numpy arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
OUTPUT:
[1 2 3 4 5 6]

# Joining 2-2D numpy arrays
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
OUTPUT:
[[1 2 5 6]
[3 47 8]]

# Using Stack() method to join 2- 1D arrays to 1- 2D array
arri = np.array([1, 2, 3])
```

```
arr2 = np.array([4, 5, 6])
arr = np.stack((arrl, arr2), axis=1)

print(arr)
```

OUTPUT:

```
[[4 4]
[2 5]
[3 6]]

# Splitting 1D array
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print (newarr)
```

OUTPUT:

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

# Horizontal splitting
a = np.arange(9).reshape(3, 3)
print(a)
ar = np.hsplit(a, 3)
print("Horizontal splitting: ")
print()
for i in ar:
print(i)
print()
```

OUTPUT:

```
[[0 1 2]
[3 4 5]
[6 7 8]]
```

Horizontal splitting:

```
([0]
[3]
[6]]

([4]
[4]
[71]

([2]
[5]
[8]]

print(a)
ar = np.vsplit(a, 3)
print("Vertical splitting: \n")
for i in ar:
print(i)
print()
```

OUTPUT:

```
[[0 1 2]
[3 4 5]
[6 7 8]]

Vertical splitting:

[[0 1 2]]
[[3 4 5]]
[[6 7 8]]
```

3. Computation on NumPy arrays using Universal Functions and Mathematical methods.

```
import statistics as st
arey = np.array([1,2,3,7,5,-1,3,4,6,23,47,-56,2])
print("arr shape:",len(arey))

print("minimum of array:",min(arey))
print("maximum of array:",max(arey))
print("Mean of array:",arey.mean())
print("Median of array:",np.median(arey))
print("Mode of array:",st.mode(arey))
print("Standard deviation of array:",np.std{arey))
print("50% Quantile of array:",np.quantile(arey,@.5))
```

OUTPUT:

```
arr shape: 13
minimum of array: -56
maximum of array: 47
Mean of array: 3.5384615384615383
Median of array: 3.0
Mode of array: 2
Standard deviation of array: 21.240800305853263
50% Quantile of array: 3.0
```

4. Import a CSV file and perform various Statistical and Comparison operations on rows/columns.

```
import pandas as pd
dataset = pd.read_csv('studentper.csv')
```

dataset

OUTPUT:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75
...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95
996	male	group C	high school	free/reduced	none	62	55	55
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

1000 rows × 8 columns

dataset.isna().sum()

OUTPUT:

Gender 0

race/ethnicity 0

parental level of education 0

lunch 0

test preparation course 0

math score 0

reading score 0

writing score 0

dtype: int64

dataset. columns

OUTPUT:

Index(['gender', ‘race/ethnicity’, 'parental level of education’, ‘lunch’, ‘test preparation course’, ‘math score’, ‘reading score’,‘writing score'], dtype="object')

```
dataset[[‘math score’, ‘reading score’, ‘writing score']].mean()
```

OUTPUT:

math score 66.089

reading score 69.169

writing score 68.054

dtype: float64

```
dataset[[‘math score’, ‘reading score', ‘writing score']].mode()
```

OUTPUT:

math score readingscore writing score

0 65 72 74

```
dataset[[‘math score’, ‘reading score', ‘writing score']].median()
```

OUTPUT:

math score 66.0

reading score 70.0

writing score 69.0

dtype: float64

5. Load an image file and do crop and flip operation using NumPy Indexing.

```
#Cropping image
import cv2
import numpy as np
img=cv2.imread('D:/Akshara_Waste/vjsv/Backgrounds/Dr-Sarvepalli-Radhakrishnan.jpeg')
print(img.shape) #prints image shape
reimg=cv2.resize(img,(500,600))
cv2.imshow("original",img) #prints original image
```

```
cv2.imshow("resized",reimg)
cropped_img=img[300:900,500:1800] #cropping image
cv2.imshow("cropped image",cropped_img)
cv2.imwrite("cropped cat.jpg",cropped_img) #to save cropped image
cv2.waitKey(0) #allows users to display a window for given milliseconds or until any key is pressed.If 0 is passed window is open until any key is pressed
cv2.destroyAllWindows()
#allows users to destroy or close all windows at any time after exiting the script
```

OUTPUT:

```
(675, 1200, 3)
```

```
# Flipping image using numpy
from PIL import Image
print (type(img))
# <class ‘numpy.ndarray'>
print (img.shape)
# (225, 400, 3)
img1 = Image. fromarray(np.flipud(crop))
print (img1)
cv2.imshow("img" ,np.flipud(crop))
cv2.waitKey(0)
cv2.destroyAllwindows ()
# Image. fromarray(np.fliplr(crop)).save('tmg2.jpg')
```

Image.fromarray(np.flip(crop, (0, 1))).save('img3.jpg')

OUTPUT:

<class ‘numpy.ndarray'>

(2006, 2000, 3)

<PIL.Image.Image image mode=RGB size=1600x60@ at @x2C9958D3e@De>

EXPERIMENT NO: Week 3, 4, 5 & 6

1. Create Pandas Series and Data Frame from various inputs.

```
#importing pandas library
import pandas as pd
#Creating a list
author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
#Creating a Series by passing list variable to Series() function
auth_series = pd.Series(author)
#Printing Series
print(auth_series)
```

OUTPUT:

```
0 Jitender
1 Purnima
2 Arpit
3 Jyoti
dtype: object
```

```
#Creating Dataframe from multiple Series
#Importing Pandas library
import pandas as pd
#Creating two lists
author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
article = [210, 211, 114, 178]
#Creating two Series by passing lists
auth_series = pd.Series(author)
article_series = pd.Series(article)
```

```
 #Creating a dictionary by passing Series objects as values
frame = { 'Author': auth_series, 'Article': article_series }

#Creating DataFrame by passing Dictionary
result = pd.DataFrame(frame)

#Printing elements of Dataframe
print(result)
```

OUTPUT:

```
Author Article
0 Jitender 210
1 Purnima 211
2 Arpit 114
3 Jyoti 178
```

```
#Importing pandas library
import pandas as pd
#Creating Series
auth_series = pd.Series(['Jitender', 'Purnima', 'Arpit', 'Jyoti'])
article_series = pd.Series([210, 211, 114, 178])

#Creating Dictionary
frame = { 'Author': auth_series, 'Article': article_series }

#Creating Dataframe
result = pd.DataFrame(frame)

#Creating another list
age = [21, 21, 24, 23]

##Creating new column in the dataframe by providing s Series created using list
result['Age'] = pd.Series(age)

#Printing dataframe
print(result)
```

OUTPUT:

Author Article Age
0 Jitender 210 21
1 Purnima 211 21
2 Arpit 114 24
3 Jyoti 178 23

#Creating a Dataframe using dictionary of Series

```
#Importing pandas library
import pandas as pd

#Creating dictionary of Series
dict1={'Auth_Name':pd.Series(['Jitender', 'Purnima', 'Arpit', 'Jyoti']),
'Author_Book_No': pd.Series([210, 211, 114, 178]),
'Age': pd.Series([21, 21, 24, 23]) }

#Creating and Printing Dataframe
df = pd.DataFrame(dict1)
print(df)
```

OUTPUT:

Auth_Name Author_Book_No Age
0 Jitender 210 21
1 Purnima 211 21
2 Arpit 114 24
3 Jyoti 178 23

2. Import any CSV file to Pandas Data Frame and perform the following:

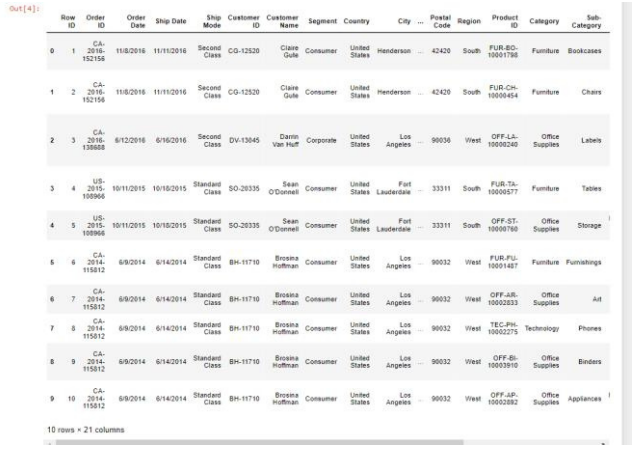
i. Visualize the first and last 10 records.

```
import pandas as pd
sp=pd.read_csv("C:/Users/SINDHU/Downloads/Sample-Superstore-
csv.csv")
sp
```

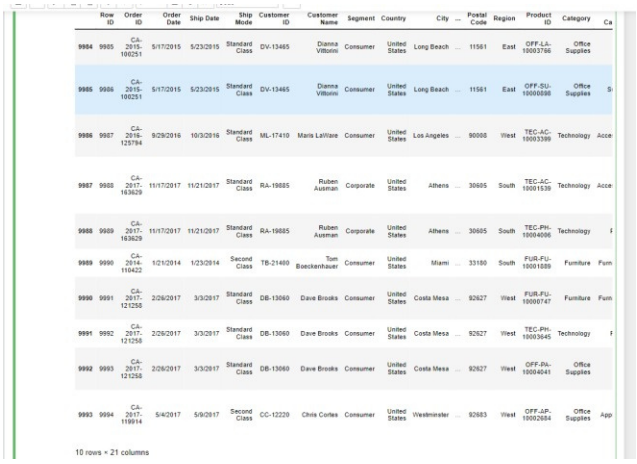
OUTPUT:

```
 # first 10 records
sp.head(10)
```

OUTPUT:



#last 10 records
sp.tail(10)
OUTPUT:



ii. Get the shape, index, and column details.

#shape-It gives number of rows and columns
sp.shape
OUTPUT:
(9994, 21)

sp.index
OUTPUT:
RangeIndex(start=@, stop=9994, step=1)
sp.columns

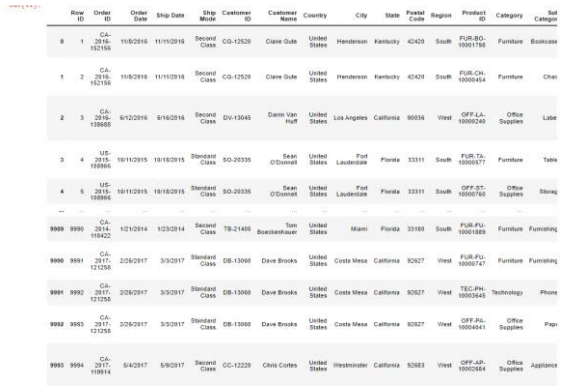
OUTPUT:

Index(["Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode',"Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State',"Postal Code", 'Region', 'Product ID', 'Category', 'Sub-Category',"Product Name', 'Sales', 'Quantity', 'Discount', 'Profit'], dtype='object')

Select/Delete the records(rows)/columns based on conditions. sp.drop(index=1)
#for deleting row or column by specifying the index value and axis

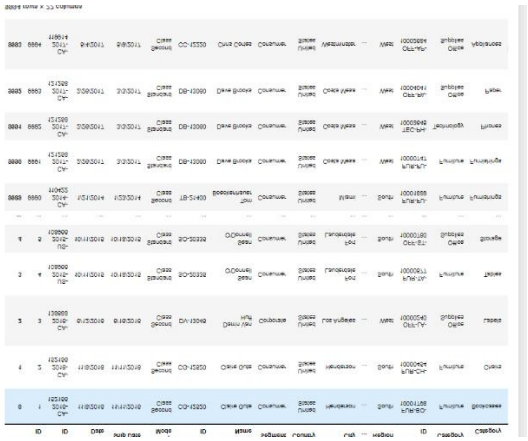
OUTPUT:

sp.drop('Segment' ,axis=1)
OUTPUT:



Perform ranking and sorting operations.
sp["Rank"]=sp["Profit"].rank()
#to create a new coLumn rank and rank the profit

Sp
OUTPUT:



sp[['City'', 'Profit']].sort_values(by="Profit",ascending =True)

OUTPUT:

]:

	City	Profit
7772	Lancaster	-6599.9780
683	Burlington	-3839.9904
9774	San Antonio	-3701.8928
3011	Louisville	-3399.9800
4991	Chicago	-2929.4845
...	...	...
4098	Minneapolis	4630.4755
9039	Detroit	4946.3700
4190	Newark	5039.9856
8153	Seattle	6719.9808
6826	Lafayette	8399.9760

9994 rows × 2 columns

v. Do required statistical operations on the given columns.

```
# mean of the profit
mean=sp["Profit"].mean()
mean
```

OUTPUT:

28 .656896307784802

```
# median of the profit
median=sp["Profit" ].median()
median
```

OUTPUT:

8.6665

```
# mode of the profit
mode=sp[ "Profit" ].mode()
```

mode
OUTPUT:
8 8.0
dtype: float64

#standard deviation std=sp["Profit"].std()
std
OUTPUT:
234. 26010769095757

#minimum value of profit
min=sp["Profit"].min()
min
OUTPUT:
-6599.978

#maximum value of profit max=sp["Profit"].max() max
OUTPUT:
8399 .976

vi.Find the count and uniqueness of the given categorical values.

# To find the total number of columns present in certain category
count=sp["Profit"].count()
count
OUTPUT:
9994
sp["Profit"].unique()

OUTPUT:
array([41.9136, 219.582 , 6.8714, ..., 16.124, 4.1028, 72.948])

vii.Rename single/multiple columns.

# Rename single/multiple columns
sp.rename(columns={'RowID': 'Rowid'},inplace=True)
sp

OUTPUT:

Out[8]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Category	Ca
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-10001798	Furniture	Boo
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-10000454	Furniture	
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-10000240	Office Supplies	
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-10000577	Furniture	
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-10000760	Office Supplies	S
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	1/21/2014	1/23/2014	Second Class	TB-21400	Tom Boeckenhauer	Consumer	United States	Miami	...	33180	South	FUR-FU-10001889	Furniture	Furn
9990	9991	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	...	92627	West	FUR-FU-10000747	Furniture	Furn
9991	9992	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	...	92627	West	TEC-PH-10003645	Technology	P
9992	9993	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	...	92627	West	OFF-PA-10004041	Office Supplies	
9993	9994	CA-2017-119914	5/4/2017	5/9/2017	Second Class	CC-12220	Chris Cortes	Consumer	United States	Westminster	...	92683	West	OFF-AP-10002684	Office Supplies	Appl

9994 rows × 21 columns

EXPERIMENT NO: Week 7, 8, 9 & 10

1. Import any CSV file to Pandas Data Frame and perform the following:

i. Handle missing data by detecting and dropping/ filling missing values.

```
# Loading a Sample Pandas DataFrame

import pandas as pd
df = pd.DataFrame({
'name': ['James', 'Jane', 'Melissa', 'Ed', 'Neil'],
'age': [30, 40, 32, 67, 43],
'score': ['90%', '95%', '100%', '82%', '87%'],
'age_missing_data': [30, 40, 32, 67, None],
'income':[100000, 80000, 55000, 62000, 120000] })
print(df)
```

OUTPUT:

	name	age	score	age_missing_data	income
0	James	30	90%	30.0	100000
1	Jane	40	95%	40.0	80000
2	Melissa	32	100%	32.0	55000
3	Ed	67	82%	67.0	62000
4	Neil	43	87%	NaN	120000

```
#Handle missing data by detecting
df.isnull().sum()
```

OUTPUT:

```
name 0
age 0
score 0
age_missing_data 1
income 0
dtype: int4
```

```
#Handle missing data by detecting and filling missing values.
df.fillna('good',inplace=True)
df.isnull().sum()
```

OUTPUT:

```
name 0
age 0
score 0
age_missing_data 0
income 0
dtype: int64
```

```
df.fillna('good'==78.5,inplace=True)
df
```

OUTPUT:

	Name	age	score	age_missing_data	income
0	James	30	90%	30.0	100000
1	Jane	40	95%	40.0	80000
2	Melissa	32	100%	32.0	55000
3	Ed	67	82%	67.0	62000
4	Neil	43	87%	good	120000

ii. Transform data using apply () and map() method.

```
# Creating a dictionary of genders
genders = {'James': 'Male', 'Jane': 'Female', 'Melissa': 'Female',
'Ed': 'Male'
# Applying a dictionary to the map method
df['gender'] = df['name'].map(genders)
print(df)
```

OUTPUT:

```
Name age score age_missing_data income gender
0 James 30 90% 30.0 100000 Male
1 Jane 40 95% 40.0 80000 Female
2 Melissa 32 100% 32.0 55000 Female
3 Ed 67 82% 67.0 62000 Male
4 Neil 43 87% good 120000 Male
```

```
# Mapping in a custom function

mean_income = df['income'].mean()
def higher_income(x):
return x > mean_income
df['higher_than_avg_income'] = df['income'].map(higher_income)
print(df)
```

OUTPUT:

```
name age score age_missing_data income gender
higher_than_avg_income
0 James 30 90% 30.0 100000 Male True
1 Jane 40 95% 40.0 80000 Female False
2 Melissa 32 100% 32.0 55000 Female False
3 Ed 67 82% 67.0 62000 Male False
4 Neil 43 87% good 120000 Male True
```

```
# Mapping in a Series

last_names = pd.Series(['Doe', 'Miller', 'Edwards', 'Nelson', 'Raul'],
df['Last Name'] = df['name'].map(last_names)
print(df)
```

OUTPUT:

```
name age score age_missing_data income gender
0 James 30 90% 30.0 100000 Male
1 Jane 40 95% 40.0 80000 Female
2 Melissa 32 100% 32.0 55000 Female
3 Ed 67 82% 67.0 62000 Male
4 Neil 43 87% good 120000 Male

higher_than_avg_income Last Name
0 True Doe
1 False Miller
2 False Edwards
```

```
3 False Nelson
4 True Raul

# Applying a function to an entire dataframe

def interview(row):
   return row['age'] < 45 and row['income'] > 75000
   df['interview'] = df.apply(interview, axis=1)
   print(df)

OUTPUT:

   name age score age_missing_data income gender

0 James 30 90% 30.0 100000 Male
1 Jane 40 95% 40.0 80000 Female
2 Melissa 32 100% 32.0 55000 Female
3 Ed 67 82% 67.0 62000 Male
4 Neil 43 87% good 120000 Male

   higher_than_avg_income Last Name interview

0 True Doe True
1 False Miller True
2 False Edwards False
3 False Nelson False
4 True Raul True

# Passing in arguments into an .apply method

def bonus(row, amount, give=False):
   if give:
      return row['income'] / row['age'] * amount
```

```
else:
   return 0
df['bonus'] = df.apply(bonus, args = (0.25,), give = True, axis=1)
print(df)
```

OUTPUT:

```
   name age score age_missing_data income gender

0 James 30 90% 30.0 100000 Male
1 Jane 40 95% 40.0 80000 Female
2 Melissa 32 100% 32.0 55000 Female
3 Ed 67 82% 67.0 62000 Male
4 Neil 43 87% good 120000 Male

   higher_than_avg_income Last Name interview bonus

0 True Doe True
833.333333
1 False Miller True
500.000000
2 False Edwards False
429.687500
3 False Nelson False
231.343284
4 True Raul True
697.674419
```

iii. Detect and filter outliers.

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

OUTPUT:

Out[1]:

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

df.isnull1().sum()

OUTPUT:
Age 0
Sex 0
Cp 0
trestbps 0
chol 0
fbs 0
restecg 0
thalach 0
exang 0
oldpeak 0
slope 0
ca 0
thal 0
target 0
dtype: inte4
df.describe()

OUTPUT:

Out[4]:

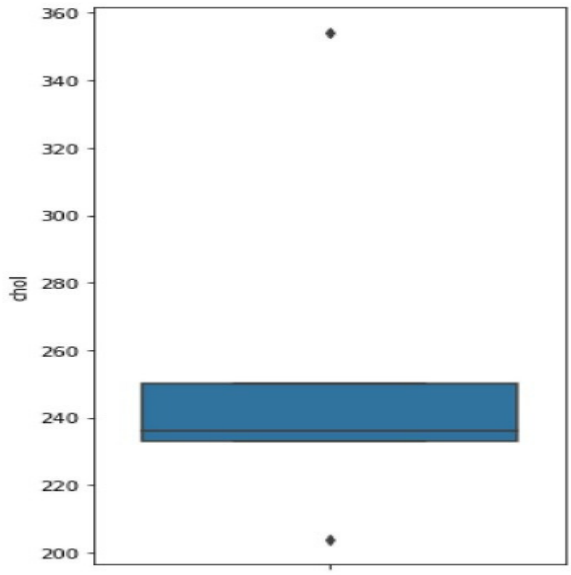
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000	5.000000	5.000000	5.0	5.000000	5.0
mean	50.800000	0.600000	1.400000	129.000000	255.400000	0.200000	0.600000	170.00000	0.200000	1.720000	1.200000	0.0	1.800000	1.0
std	11.189281	0.547723	1.140175	10.246951	57.600347	0.447214	0.547723	14.19507	0.447214	1.194571	1.095445	0.0	0.447214	0.0
min	37.000000	0.000000	0.000000	120.000000	204.000000	0.000000	0.000000	150.00000	0.000000	0.600000	0.000000	0.0	1.000000	1.0
25%	41.000000	0.000000	1.000000	120.000000	233.000000	0.000000	0.000000	163.00000	0.000000	0.800000	0.000000	0.0	2.000000	1.0
50%	56.000000	1.000000	1.000000	130.000000	236.000000	0.000000	1.000000	172.00000	0.000000	1.400000	2.000000	0.0	2.000000	1.0
75%	57.000000	1.000000	2.000000	130.000000	250.000000	0.000000	1.000000	178.00000	0.000000	2.300000	2.000000	0.0	2.000000	1.0
max	63.000000	1.000000	3.000000	145.000000	354.000000	1.000000	1.000000	187.00000	1.000000	3.500000	2.000000	0.0	2.000000	1.0

import pandas as pd

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (4,8))
sns.boxplot(y = df.chol)
```

OUTPUT:

```
Out[5]: <AxesSubplot:ylabel='chol'>
```




```
def out_iqr(df , column):
    global lower,upper
    q25, q75 = np.quantile(df[column], 0.25),
    np.quantile(df[column], 0.75)
    # calculate the IQR
    igr = q75 - q25
    # calculate the outlier cutoff
    cut_off = iqr * 1.5
    # calculate the Lower and upper bound value
    lower, upper = q25 - cut_off, q75 + cut_off
    print('The IQR is',iqr)
    print('The lower bound value is', lower)
    print('The upper bound value is', upper)
    # Calculate the number of records below and above Lower and above bound value
    respectively
    df1 = df[df[column] > upper]
    df2 = df[df[column] < lower]
```

```
return print('Total number of outliers are',
dfi.shape[0]+
df2.shape[0])

out_igqr(df,'chol')
```

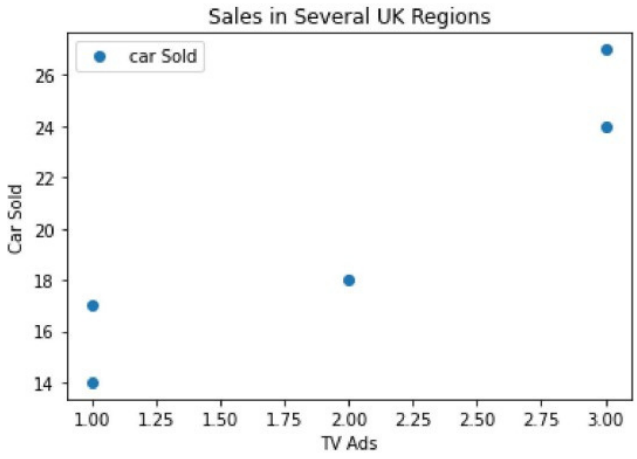
OUTPUT:

```
The IQR is 17.0
The lower bound value is 207.5
The upper bound value is 275.5
Total number of outliers are 2
```

2. Implement regularized Linear regression.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib as mpl
import statsmodels.formula.api as sm

from sklearn.linear_model import LinearRegression
from scipy import stats

tb=pd.read_excel("regr.xlsx")
tb.plot('TV Ads','car Sold',style='0')
plt.ylabel('Car Sold")
plt.title('Sales in Several UK Regions')
plt.show()
```

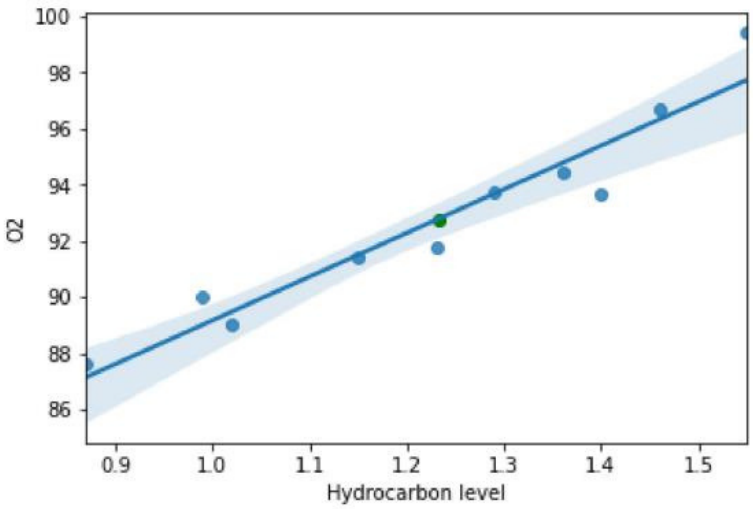
OUTPUT:




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

data=pd.read_excel("reg2.x1sx")


x=data['Hydrocarbon level']
y=data['02']


plt.figure()

sns.regplot(x,y,fit_reg=True)


plt.scatter(np.mean(x) ,np.mean(y), color="green")
```

OUTPUT:

```
<matplotlib.collections.PathCollection at 0x1ff55c73c40>
```




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

from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

df1=pd.read_excel("trucking.xlsx")
```

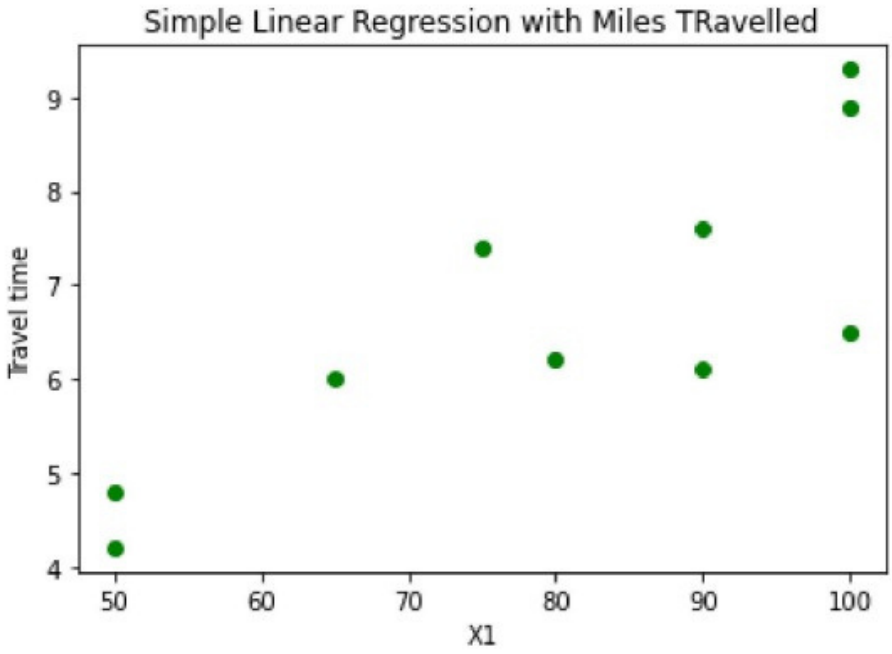
OUTPUT:

	Driving_Assignment	x1	n_of_deliveries	travel_time
0	1	100	4	9.3
1	2	50	3	4.8
2	3	100	4	8.9
3	4	100	2	6.5
4	5	50	2	4.2
5	6	80	2	6.2
6	7	75	3	7.4
7	8	65	4	6.0
8	9	90	3	7.6
9	10	90	2	6.1

```
plt.scatter(df1['x1'],df1['travel_time'],color="green')
plt.ylabel('Travel time')
plt.xlabel('X1")
plt.title('Simple Linear Regression with Miles TRavelled')
```

OUTPUT:

```
Text(0.5, 1.0, 'Simple Linear Regression with Miles TRavelled')
```




```
Regi=ols(formula="travel_time~x1" , data=df1)

Fiti=Reg1.fit()

print (Fiti.summary())
```

OUTPUT:

OLS Regression Results

Dep. Variable:	travel_time	R-squared:	0.664
Model:	OLS	Adj. R-squared:	0.622
Method:	Least Squares	F-statistic:	15.81
Date:	Fri, 25 Nov 2022	Prob (F-statistic):	0.00408
Time:	10:33:24	Log-Likelihood:	-13.092
No. Observations:	10	AIC:	30.18
Df Residuals:	8	BIC:	30.79
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.2739	1.401	0.909	0.390	-1.956	4.504
x1	0.0678	0.017	3.977	0.004	0.028	0.107

Omnibus:	0.694	Durbin-Watson:	1.723
Prob(Omnibus):	0.707	Jarque-Bera (JB):	0.623
Skew:	-0.333	Prob(JB):	0.732
Kurtosis:	1.974	Cond. No.	363.

```
from statsmodels.formula.api import ols

model=ols('travel_time~x1+n_of_deliveries' ,data=df1).fit()

model. summary()
```

OUTPUT:

OLS Regression Results

Dep. Variable:	travel_time	**R-squared:**	0.904
Model:	OLS	**Adj. R-squared:**	0.876
Method:	Least Squares	**F-statistic:**	32.88
Date:	Fri, 25 Nov 2022	**Prob (F-statistic):**	0.000276
Time:	10:34:48	**Log-Likelihood:**	-6.8398
No. Observations:	10	**AIC:**	19.68
Df Residuals:	7	**BIC:**	20.59
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.8687	0.952	-0.913	0.392	-3.119	1.381
x1	0.0611	0.010	6.182	0.000	0.038	0.085
n_of_deliveries	0.9234	0.221	4.176	0.004	0.401	1.446

Omnibus:	0.039	**Durbin-Watson:**	2.515
Prob(Omnibus):	0.981	**Jarque-Bera (JB):**	0.151
Skew:	0.074	**Prob(JB):**	0.927
Kurtosis:	2.418	**Cond. No.**	435.

print(anova_lm(Fit1))

OUTPUT:

	df	sum_sq	mean_sq	F	PR(>F)
x1	1.0	15.871304	15.871304	15.814578	0.00408
Residual	8.0	8.028696	1.003587	NaN	NaN

anova_table=anova_lm({model, type=1)
anova_table

OUTPUT:

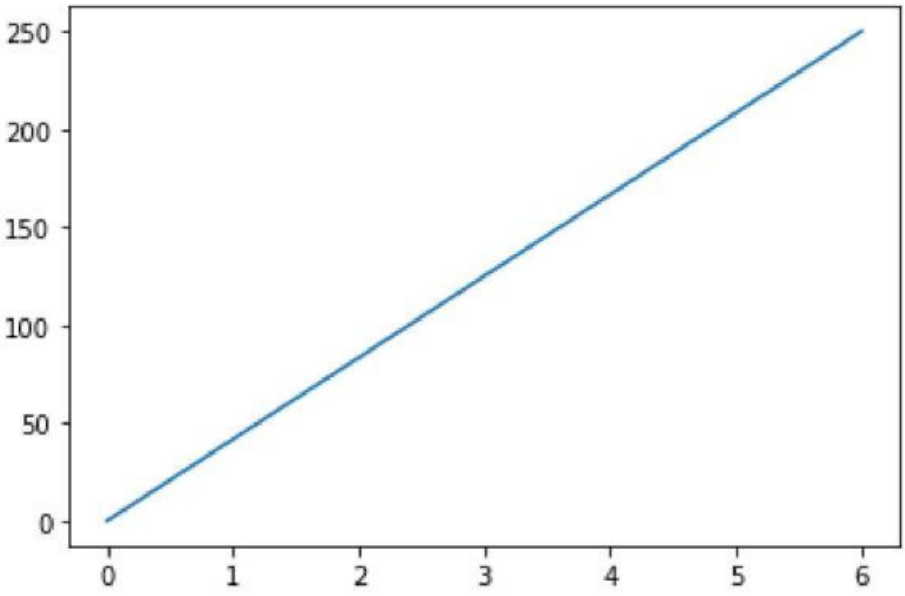
	df	sum_sq	mean_sq	F	PR(>F)
x1	1.0	15.871304	15.871304	48.315660	0.000221
n_of_deliveries	1.0	5.729252	5.729252	17.441075	0.004157
Residual	7.0	2.299443	0.328492	NaN	NaN

EXPERIMENT NO: Week 11, 12, 13 & 14

1. Visualize data using Line Plots, Bar Plots, Histograms, Density Plots and Scatter Plots.

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

xpoints = np.array([@, 6])
ypoints = np.array([@, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

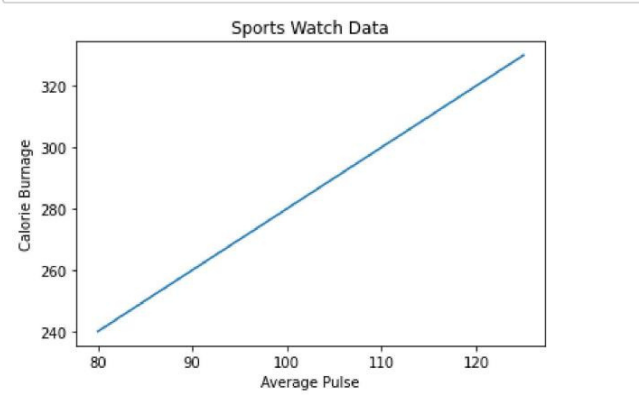
OUTPUT:




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

x = np.array([8@, 85, 90, 95, 100, 105, 110, 115, 12@, 125])
y = np.array([24@, 250, 260, 270, 28@, 290, 300, 310, 320, 33@])

plt.plot(x, y)
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
```

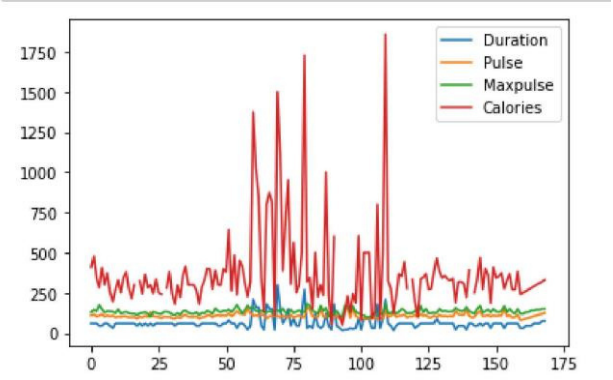
```
plt.ylabel("Calorie     Burnage")
plt.show()
```

OUTPUT:



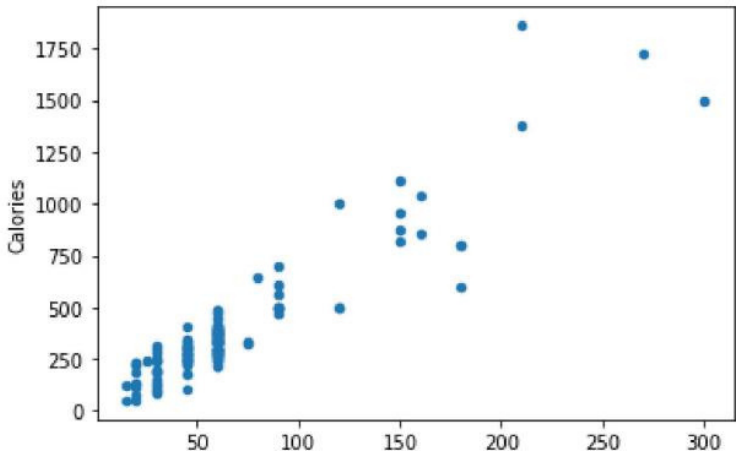

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data_1.csv') df.plot()
plt.show()
```

OUTPUT:




```
import matplotlib.pyplot as plt

df = pd.read_csv(‘data_1.csv')
df.plot(kind = 'scatter', x = ‘Duration’, y = 'Calories')
plt.show()
```

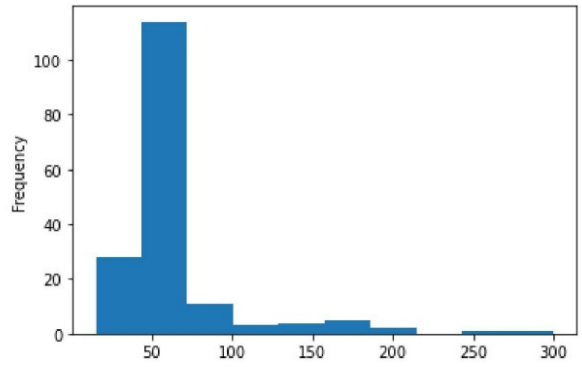
OUTPUT:



df["Duration"].plot(kind = 'hist')

OUTPUT:

```
<AxesSubplot:ylabel='Frequency'>
```



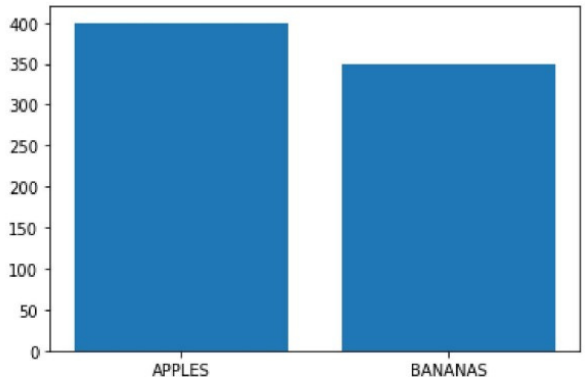

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

OUTPUT:

```
x = ["APPLES", "BANANAS" ]
y=[400,350]

plt.bar(x, y)
```

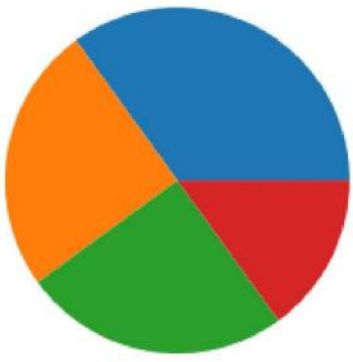
OUTPUT:

```
<BarContainer object of 2 artists>
```




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

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

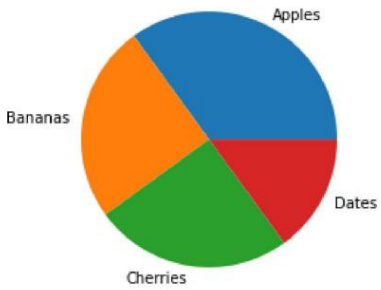
OUTPUT:



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

y = np.array([35, 25, 25, 15])
```

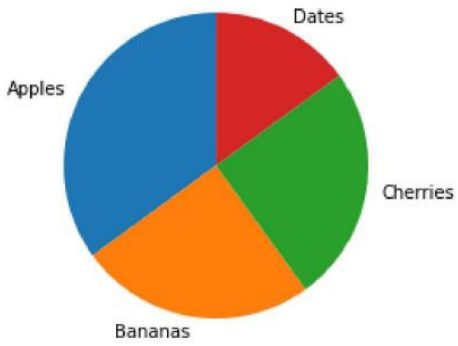
```
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

OUTPUT:




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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle
plt.show()
```

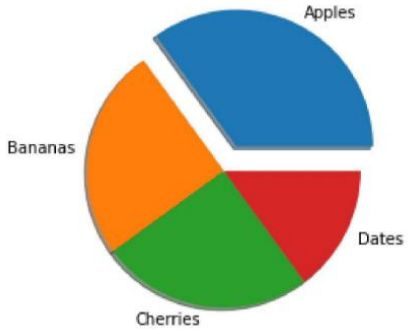
OUTPUT:




```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
```

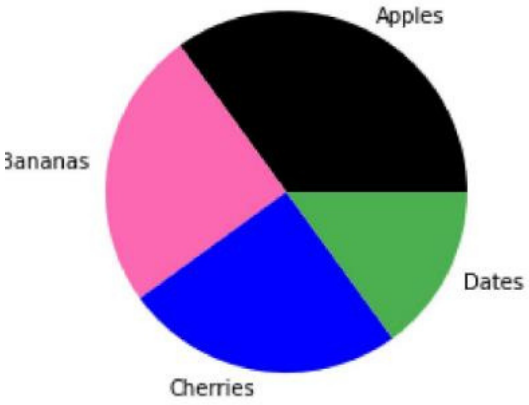
```
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [@.2, 6, ®, @]

plt.pie{y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```

OUTPUT:




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

y = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = [“black", “hotpink", "b”, "#4CAF5@"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

OUTPUT:



2. Use different Colour scales on the dataset.

from urllib.request import urlopen

import json with urlopen

('https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counticounties = json.load(response) counties["features"][0]

OUTPUT:

{'type': 'Feature',
'properties': {'GEO_ID': '0500000US01001',
'STATE': '01',
'COUNTY': '001',
'NAME': 'Autauga',
'LSAD': 'County',
'CENSUSAREA': 594.436},
'geometry': {'type': 'Polygon',
'coordinates': [[[-86.496774, 32.344437],
[-86.717897, 32.402814],
[-86.814912, 32.340803],
[-86.890581, 32.502974],
[-86.917595, 32.664169],
[-86.71339, 32.661732],
[-86.714219, 32.705694],
[-86.413116, 32.707386],
[-86.411172, 32.409937],
[-86.496774, 32.344437]]]},
'id': '01001'}

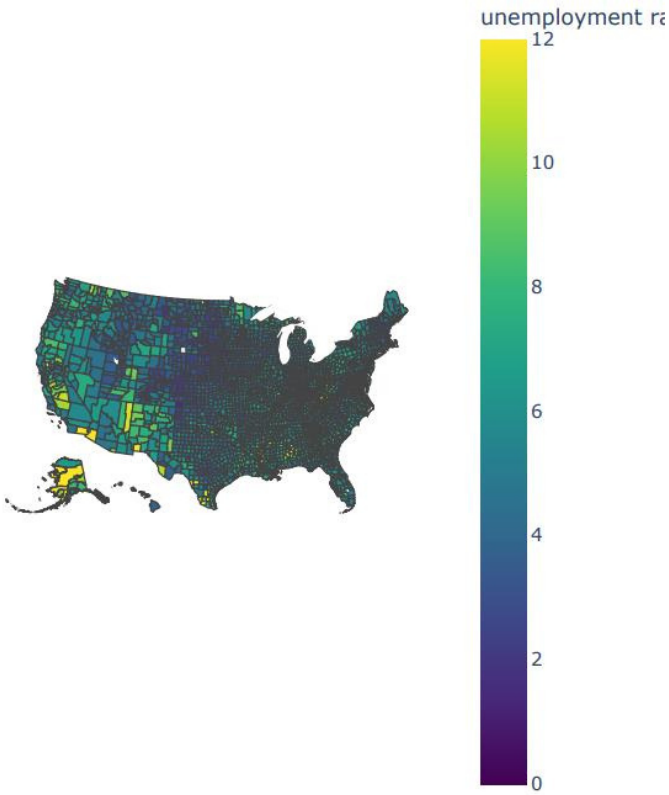
```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-
unempdtype={"fips": str})

df.head()
```

OUTPUT:

	fips	unemp
0	01001	5.3
1	01003	5.4
2	01005	8.6
3	01007	6.6
4	01009	5.5

```
from urllib.request import urlopen

import json
with
urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counticounties = json.load(response)

import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-
unempdtype={"fips": str})

import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
color_continuous_scale="Viridis",
range_color=(0, 12), scope="usa",
labels={'unemp':'unemployment rate'}
)

fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

OUTPUT:




```
import plotly.express as px

df = px.data.election()

geojson = px.data.election_geojson()

print(df["district"][2])
print(geojson["features"][0]["properties"])

df
```

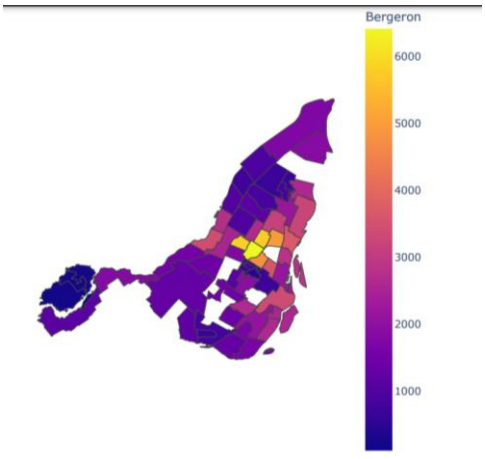
OUTPUT:

```
11-Sault-au-Récollet
{'district': '11-Sault-au-Récollet'}
```

	district	Coderre	Bergeron	Joly	total	winner	result	district_id
0	101-Bois-de-Liesse	2481	1829	3024	7334	Joly	plurality	101
1	102-Cap-Saint-Jacques	2525	1163	2675	6363	Joly	plurality	102
2	11-Sault-au-Récollet	3348	2770	2532	8650	Coderre	plurality	11
3	111-Mile-End	1734	4782	2514	9030	Bergeron	majority	111
4	112-DeLorimier	1770	5933	3044	10747	Bergeron	majority	112
5	113-Jeanne-Mance	1455	3599	2316	7370	Bergeron	plurality	113
6	12-Saint-Sulpice	3252	2521	2543	8316	Coderre	plurality	12
7	121-La Pointe-aux-Prairies	5456	1760	3330	10546	Coderre	majority	121
8	122-Pointe-aux-Trembles	4734	1879	2852	9465	Coderre	majority	122

```
import plotly.express as px
df = px.data.election()
geojson = px.data.election_geojson()
fig = px.choropleth(df, geojson=geojson, color="Bergeron",
locations="district", featureidkey="properties.district",
projection="mercator" )
fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show() ‘
```

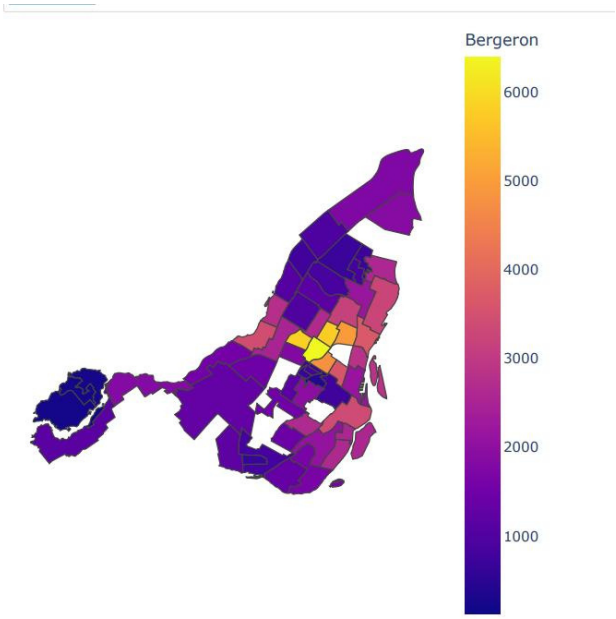
OUTPUT:




```
import plotly.express as px
```

```
df = px.data.election()

geojson = px.data.election_geojson()

fig = px.choropleth(df, geojson=geojson, color="Bergeron",
locations="district", featureidkey="properties.district",
projection="mercator"
)

fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

OUTPUT:



3. For a sales dataset do a Time Series visualization.

```
from pandas import read_csv
from matplotlib import pyplot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

print(series.head())
```

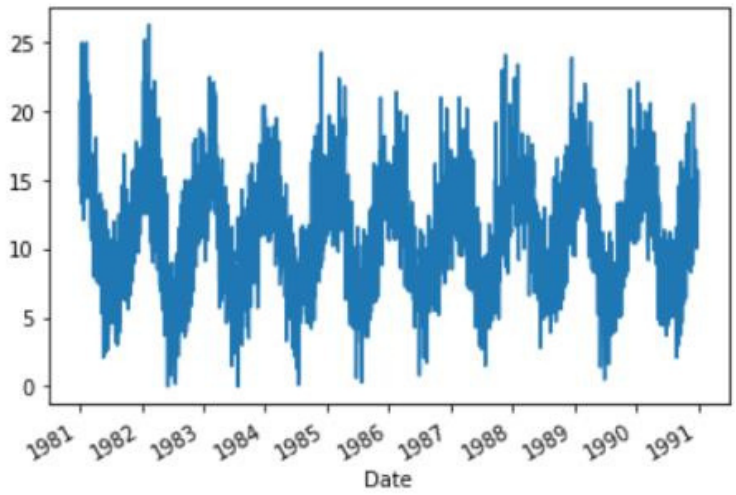
OUTPUT:

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
Date
1981-01-01 20.7
1981-01-02 17.9
1981-01-03 18.8
1981-01-04 14.6
1981-01-05 15.8
Name: Temp, dtype: float64
```

Time Series Line Plot

```
from pandas import read_csv
from matplotlib import pyplot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

series.plot()

pyplot.show()
```

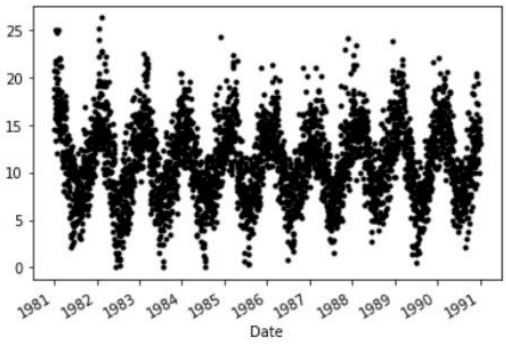
OUTPUT:




```
from pandas import read_csv
from matplotlib import pyplot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates
series.plot(style='k.')

pyplot.show()
```

OUTPUT:

```
  series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

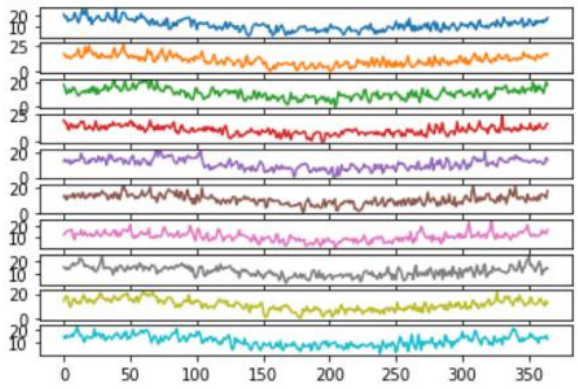



```
from pandas import read_csv from
pandas import DataFrame
```

```
from pandas import Grouper
from matplotlib import pyplot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

groups = series.groupby(Grouper(freq='A'))

years = DataFrame()

for name, group in groups:
    years[name.year] = group.values

years.plot(subplots=True, legend=False)
pyplot.show()
```

OUTPUT:

```
  series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

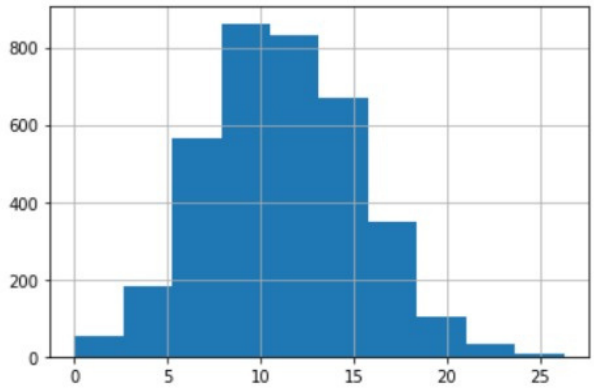


Time Series Histogram and Density Plots

```
from pandas import read_csv
from matplotlib import pyplot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

series.hist()
pyplot.show()
```

OUTPUT:

```
  series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```



from pandas import read_csv
from matplotlib import pyplot

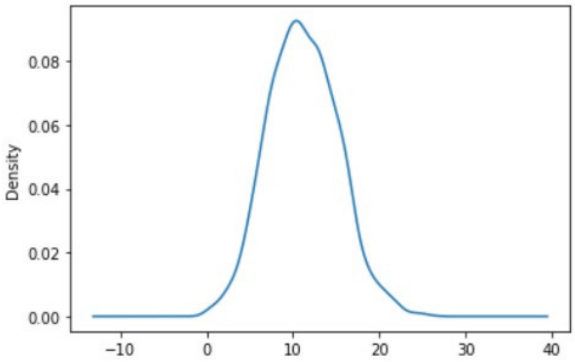
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates)

series.plot(kind='kde')

pyplot.show()

OUTPUT:

```
  series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

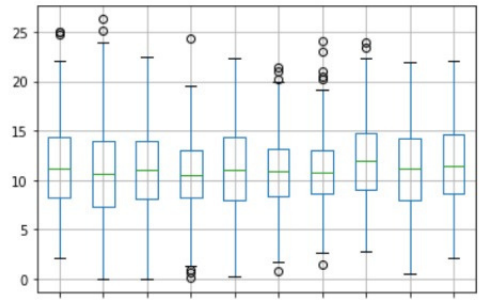


Time Series Box and Whisker Plots by Interval

```
from pandas import read_csv
from pandas import DataFrame
from pandas import Grouper
from matplotlib import pyplot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates)

groups = series.groupby(Grouper(freq='A'))

years = DataFrame()

for name, group in groups:
    years[name.year] = group.values

years.boxplot()

pyplot.show()
```

OUTPUT:

```
  series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

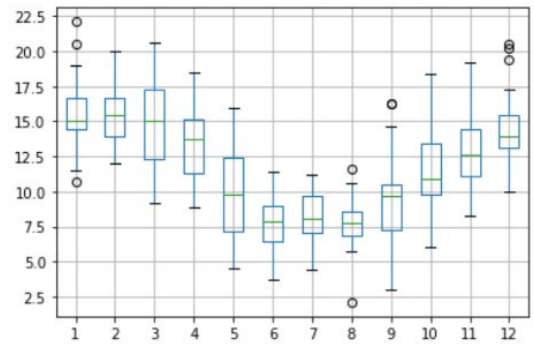


from pandas import read_csv
from pandas import DataFrame
from pandas import Grouper
from matplotlib import pyplot

```
from pandas import concat

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

one_year = series['1990']

groups = one_year.groupby(Grouper(freq='M'))

months = concat([DataFrame(x[1].values) for x in groups], axis=1)
months = DataFrame(months)
months.columns = range(1,13)
months.boxplot()

pyplot.show()
```

OUTPUT:

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

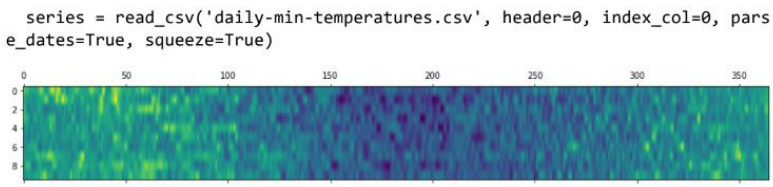


Time Series Heat Maps

```
from pandas import read_csv
from pandas import DataFrame
from pandas import Grouper
from matplotlib import pyplot
```

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

groups = series.groupby(Grouper(freq='A'))

years = DataFrame()

for name, group in groups:
    years[name.year] = group.values

years = years.T

pyplot.matshow(years, interpolation=None, aspect='auto')
pyplot.show()
```

OUTPUT:

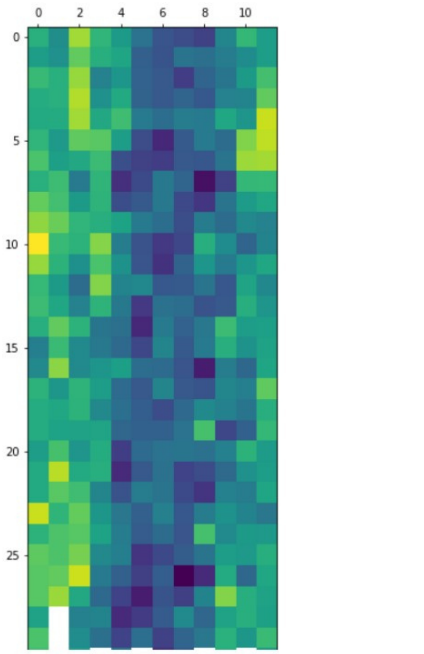



```
from pandas import read_csv
from pandas import DataFrame
from pandas import Grouper
from matplotlib import pyplot
from pandas import concat

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

one_year = series['1990']

groups = one_year.groupby(Grouper(freq='M'))

months = concat([DataFrame(x[1].values) for x in groups], axis=1)
months = DataFrame(months)
```

```
months.columns = range(1,13)

pyplot.matshow(months, interpolation=None, aspect='auto')
pyplot.show()
```

OUTPUT:

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

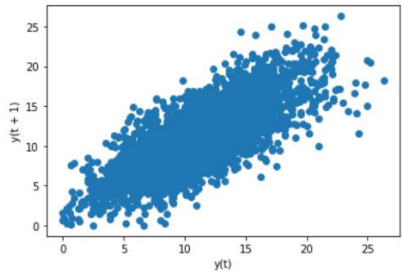


Time Series Lag Scatter Plots

```
from pandas import read_csv
from matplotlib import pyplot
from pandas.plotting import lag_plot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

lag_plot(series)

pyplot.show()
```

OUTPUT:

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```

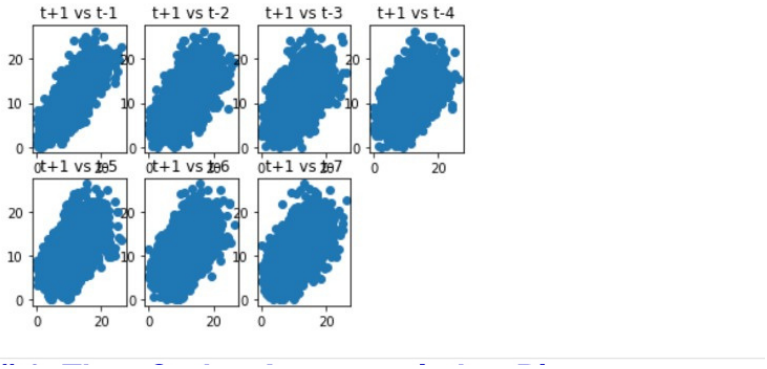



```
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from matplotlib import pyplot
from pandas.plotting import scatter_matrix

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

values = DataFrame(series.values)

lags = 7

columns = [values]

for i in range(1,(lags + 1)):
    columns.append(values.shift(i))

dataframe = concat(columns, axis=1)

columns = ['t+1']

for i in range(1,(lags + 1)):
    columns.append('t-' + str(i))

dataframe.columns = columns

pyplot.figure(1)

for i in range(1,(lags + 1)):
    ax = pyplot.subplot(240 + i)
```

```
ax.set_title('t+1 vs t-' + str(i))

pyplot.scatter(x=dataframe['t+1'].values, y=dataframe['t-'+str(i)].values)
pyplot.show()
```

OUTPUT:

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```



Time Series Autocorrelation Plots

```
from pandas import read_csv
from matplotlib import pyplot
from pandas.plotting import autocorrelation_plot

series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, parse_dates )

autocorrelation_plot(series)

pyplot.show()
```

OUTPUT:

```
series = read_csv('daily-min-temperatures.csv', header=0, index_col=0, pars
e_dates=True, squeeze=True)
```