

Q 1

In [45]: *# Importing necessary libraries*

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import PrecisionRecallDisplay

from scipy.stats import zscore, norm
import pandas as pd
import matplotlib.pyplot as plt

from scipy.stats import zscore, norm
import numpy as np

import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor
```

1. Anomaly Detection (30 points)

Part A (5 Points):

By dividing a data set into quartiles, IQR is used to measure variability. The data is sorted ascending and divided into four equal parts. Q1, Q2, Q3, also known as the first, second, and third quartiles, are the values that separate the four equal parts.

Use the following data points to calculate outliers in the data `data = [11, 3, 8, 10, 12, 5, 1, 50]`

Using a box plot, show the outliers in the box plot.

In [46]:

```

datapoints=np.array([11,3,8,10,12,5,1,50])
sorted=np.sort(datapoints)

q3,q1 = np.percentile(datapoints, [75, 25])

iqr=q3-q1
print("Q1 = ",q1, ", Q3=",q3," , IQR",iqr) #4.5

# Finding the outliers: points outside the range (Q1 - 1.5 * IQR) , (

lb = q1-(1.5*iqr)
ub= q3+(1.5*iqr)

outlier = datapoints[(datapoints < lb) | (datapoints > ub)]

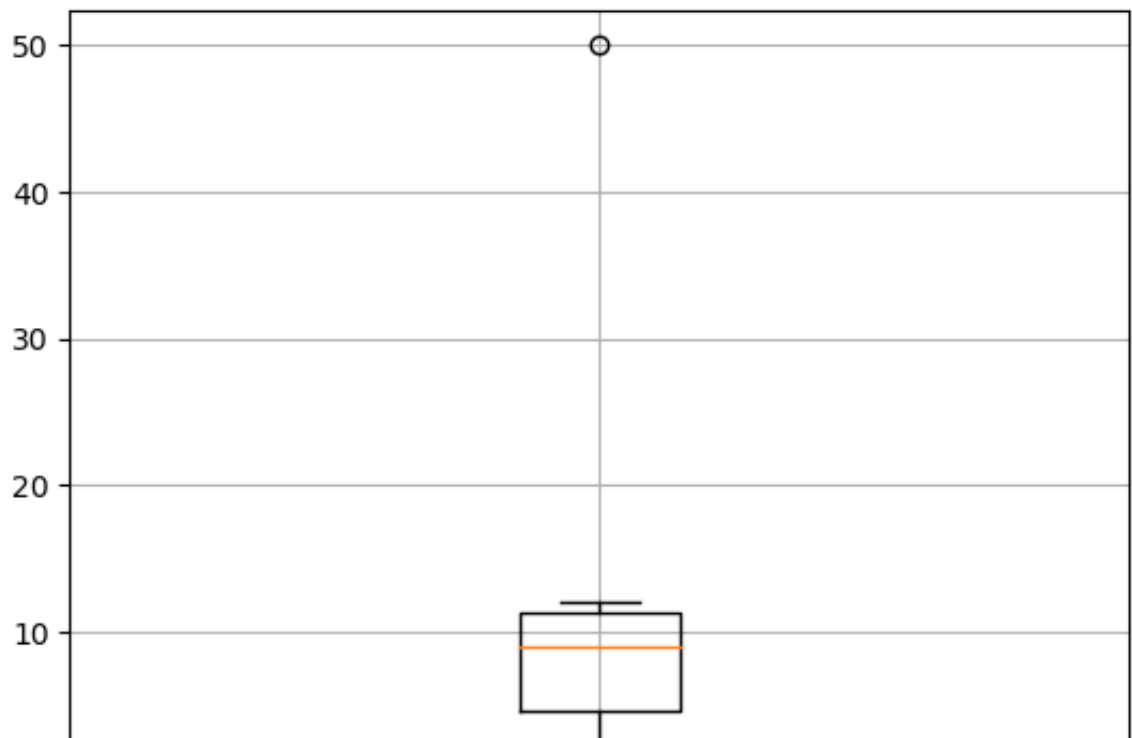
print("Lower Bound is (", lb,") \nUpper Bound is (", ub,")")

print("Outlier is : ", outlier )

plt.boxplot(datapoints)
plt.grid(True)
plt.show()

```

Q1 = 4.5 , Q3= 11.25 , IQR 6.75
 Lower Bound is (-5.625)
 Upper Bound is (21.375)
 Outlier is : [50]



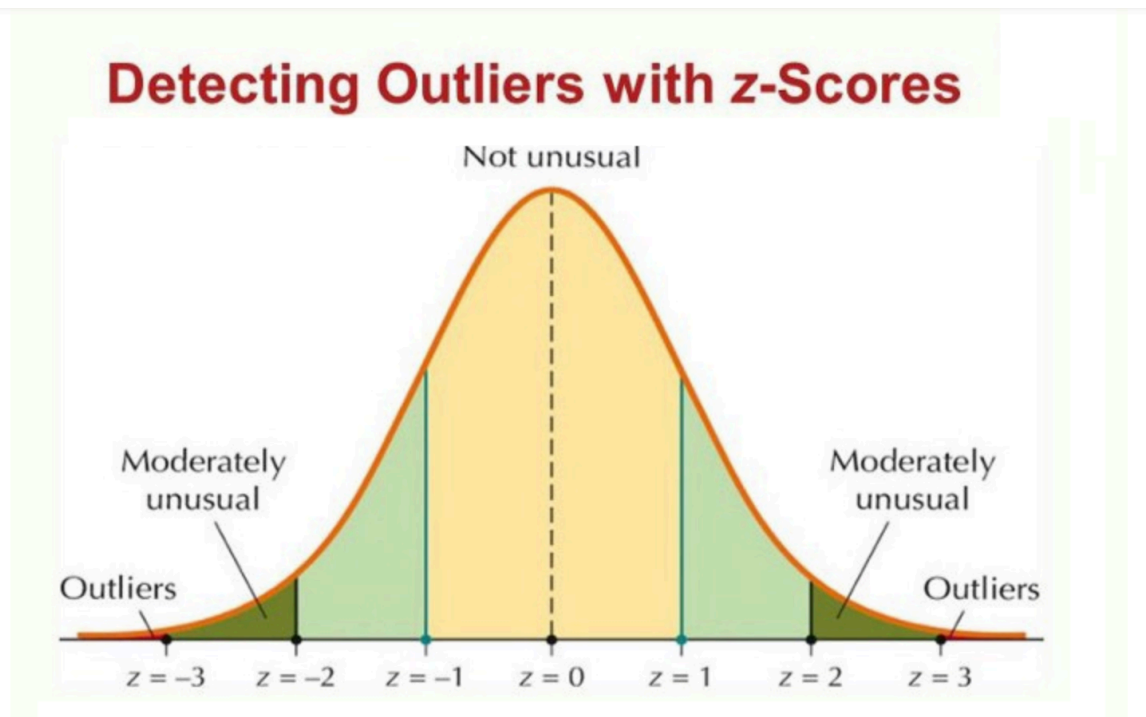


In []:

Part B (5 points):

Using the formula to calculate the Z-score detect outliers in the following data points. data = [6, 3, 9, 6, 9, 20, 3, 10, 3, 50, 6, 5, 9, 9, 3, 6, 3] Using a box plot, show the outliers in the box plot.

Zscore is used to understand how close or far a dpoint is from its mean



Source: Analytics Vidhya

In [85]:

```

dp = np.array([6, 3, 9, 6, 9, 20, 3, 10, 3, 50, 6, 5, 9, 9, 3, 6, 3])

mean = np.mean(dp)
std= np.std(dp)
zs= zscore(dp)

print("Mean : ", np.round(mean,2))
print("Std Dev : ", np.round(std, 2))
print("\nZ Score : \n ", np.round(zs,2))

#Setting a threshold

plt.hist(dp,density=5)

x = np.linspace( (mean - 3 * std), (mean + 3 * std) , 90)
y = norm.pdf(x, mean, std)

plt.axvline(mean, color='red', label='Mean')
plt.axvline(mean - std, color='green' ,label='Mean - Std Dev')
plt.axvline(mean + std, color='green', label='Mean + Std Dev')

plt.plot(x, y, '--', color='black', label='Gaussian Dist')
plt.legend()
plt.show()

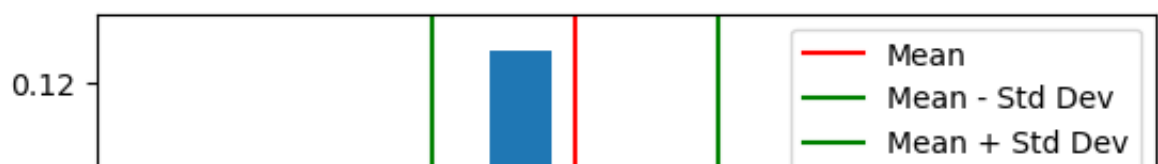
plt.boxplot(dp)
plt.grid(True)
plt.show()

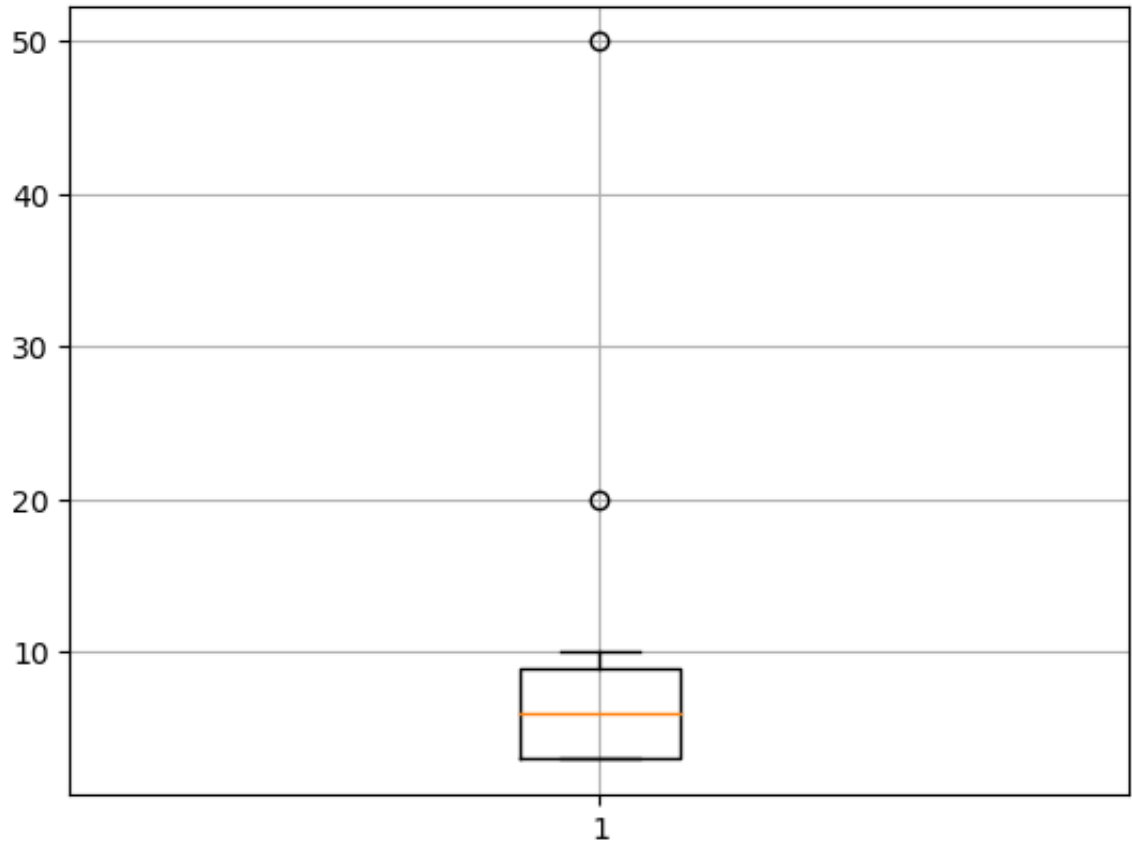
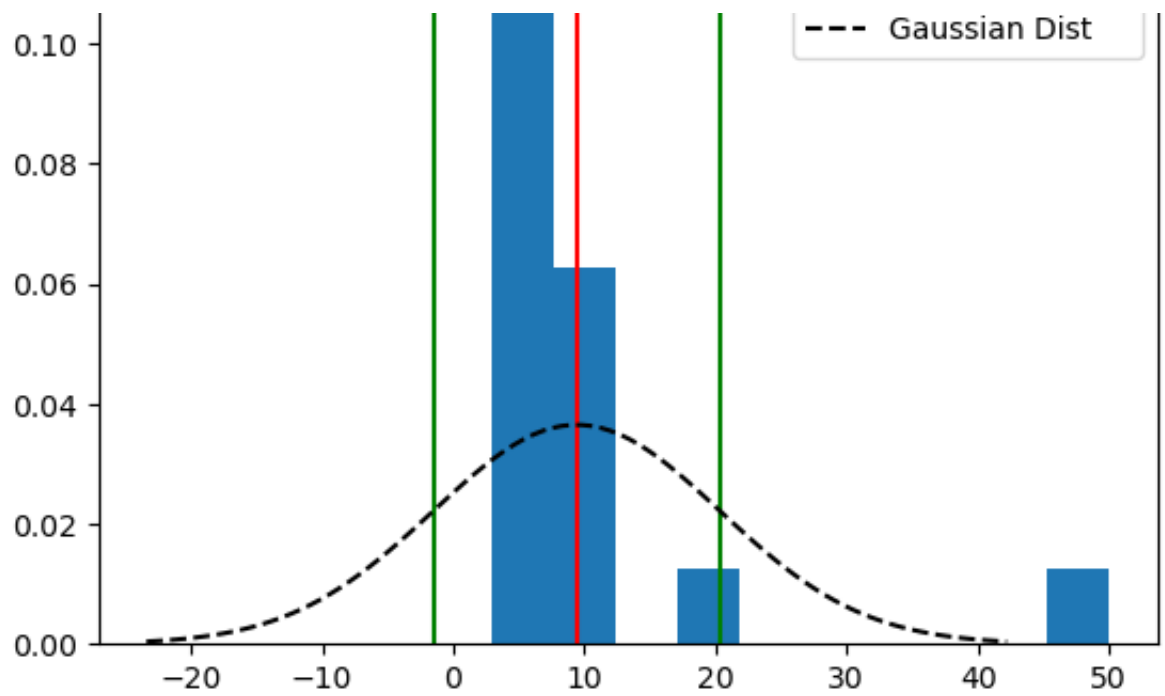
print("Outliers are 20 and 50 ")

```

Mean : 9.41
Std Dev : 10.93

Z Score :
 [-0.31 -0.59 -0.04 -0.31 -0.04 0.97 -0.59 0.05 -0.59 3.71 -0.31
 -0.4
 -0.04 -0.04 -0.59 -0.31 -0.59]





Outliers are 20 and 50

In []:

Part C (20 points):

Use the dataset attached for identifying the outliers using Z-score.

Steps to follow in this question

- Step1(5 points): Show outliers using histograms and scatterplots. Then
- Step2(7 points): Identify the outliers using Z-score for SalePrice column by using atleast 4 different thresholds.
- Step3(4 points): Print the number of outliers removed.
- Step4(4 points): Use LocalOutlierFactor as discussed in the class to plot the outliers from SalePrice and LotArea columns.

STEP 1 - Show outliers using histograms and scatterplots

```
In [98]: data = pd.read_csv("/Users/mumukshapant/Downloads/ML/Assignments/Assignments/Assignment1/Assignment1_data.csv")
data
```

Out[98]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCont
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	
...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	

1460 rows × 81 columns

```
In [ ]:
```

In []:

```
In [108]: y= data["SalePrice"]

#calculating IQR And using it for outlier detection
q3, q1 = np.percentile(y, [75, 25])

iqr = q3 - q1
print("Q1 =", q1, ", Q3 =", q3, ", IQR =", iqr)

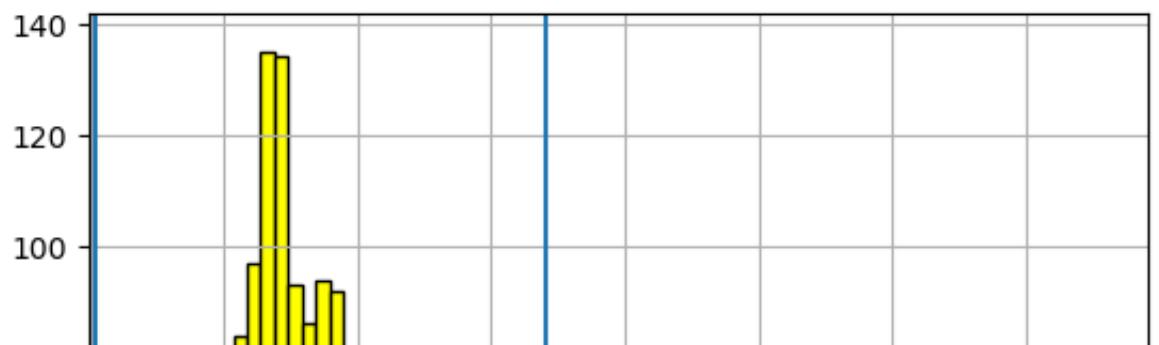
# Finding the outliers: points outside the range (Q1 - 1.5 * IQR), (Q3 + 1.5 * IQR)
lb = q1 - (1.5 * iqr)
ub = q3 + (1.5 * iqr)

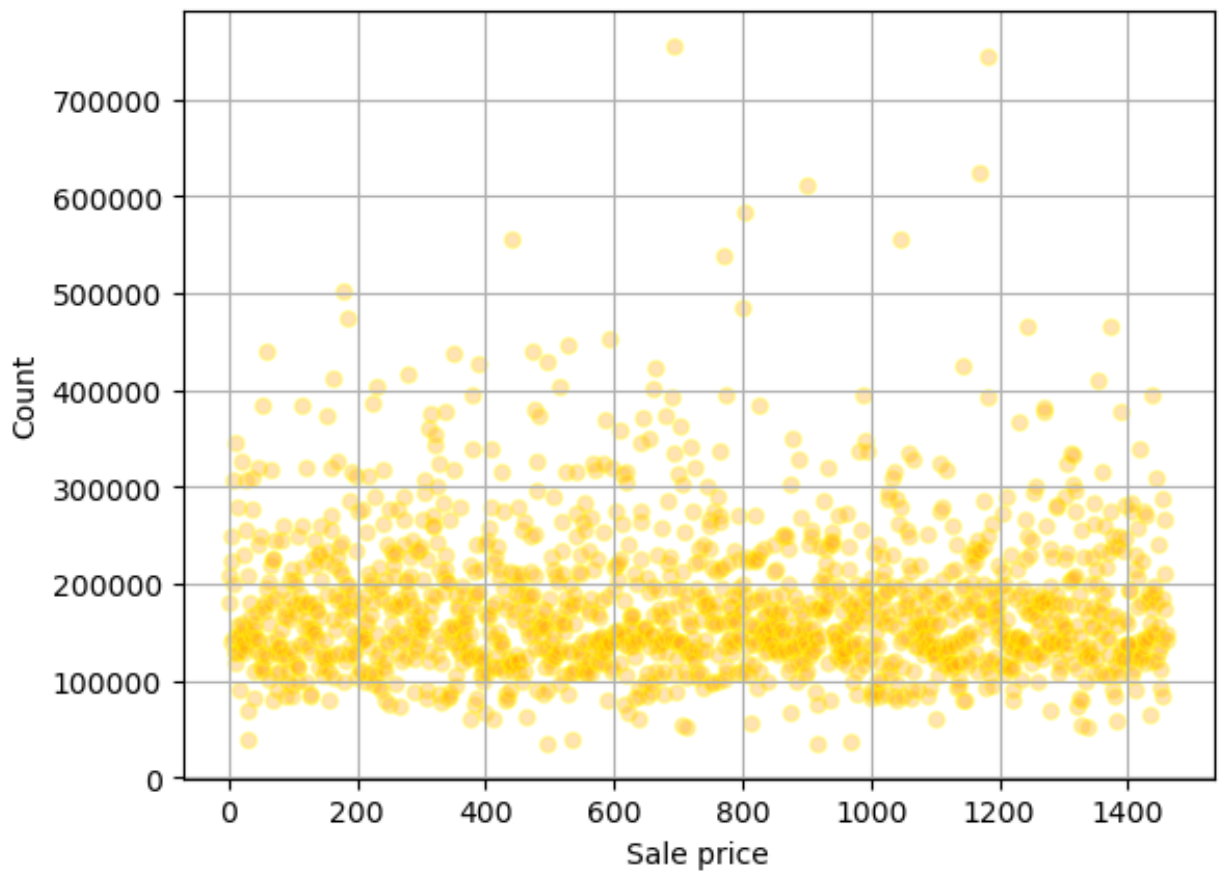
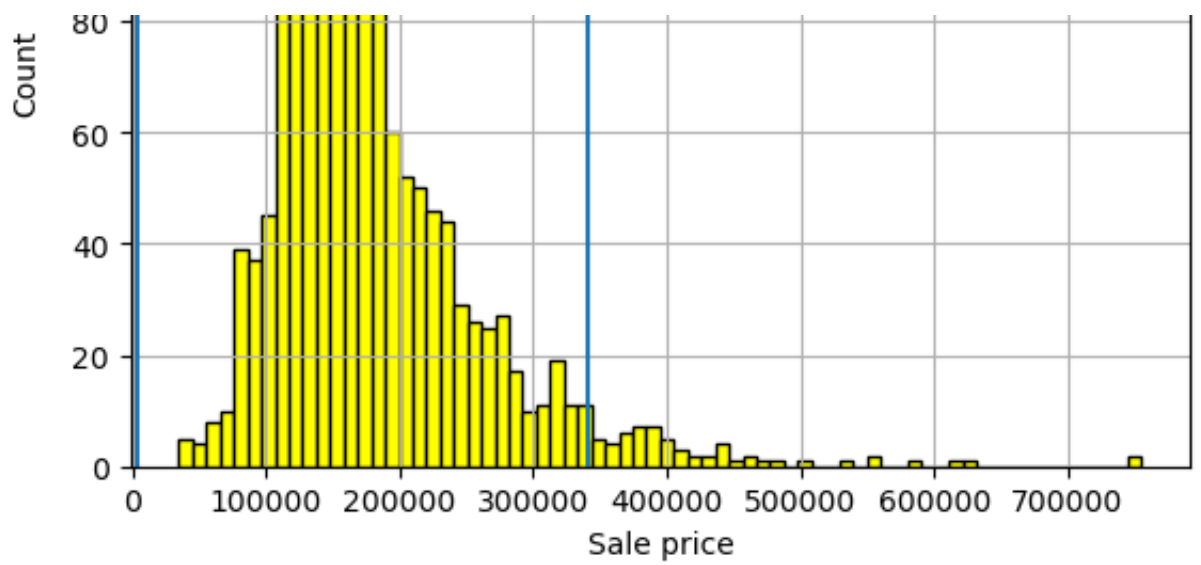
#Histogram
plt.hist(y, edgecolor='black', bins=70, color='yellow')
plt.xlabel("Sale price")
plt.ylabel("Count")

#highlighting the points below and above threshold
plt.axvline(x=lb)
plt.axvline(x=ub)
plt.grid(True)
plt.show()

# Scatter plot
plt.scatter(data['Id'], y, s=30, alpha=0.3, edgecolors='yellow', color='black')
plt.xlabel("Sale price")
plt.ylabel("Count")
plt.grid(True)
plt.show()
```

Q1 = 129975.0 , Q3 = 214000.0 , IQR = 84025.0





STEP 2 , 3 -

Identify the outliers using Z-score for SalePrice column by using atleast 4 different thresholds.

Print the number of outliers removed.

In [50]:


```

# Mean and std dev for Sale Price
y= data['SalePrice']
mean_y = y.mean()
std_y = y.std()

#Zscore
zscore_y = zscore(y)

tholds= [2.5, 3.0, 3.5, 4.0]
colors = ['red']

for t in tholds:

    outliers = y[np.abs(zscore_y) > t]

    print("Threshold",t,"\nOutliers ", outliers)
    # Create a new figure

    # Data points
    plt.scatter(data.index, y, label='Datapoint')

    #outliers
    plt.scatter(outliers.index, outliers, label=f'Outliers w threshold {t}')

    plt.ylabel("Sale Price")
    plt.xlabel("Index")
    plt.legend()
    plt.show()
    print("\n\n\n\n")

```

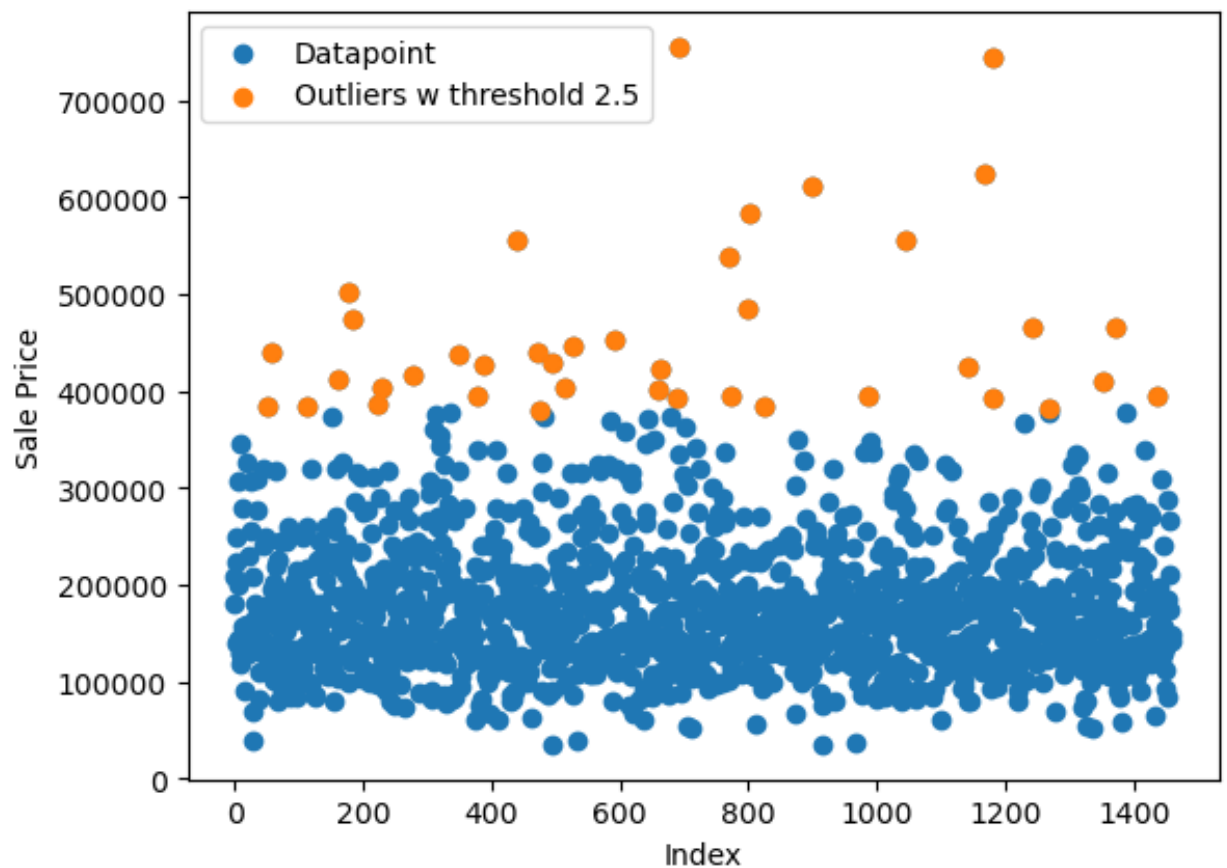
```

Threshold 2.5
Outliers  53      385000
58      438780
112     383970
161     412500
178     501837
185     475000
224     386250
231     403000
278     415298
349     437154
378     394432
389     426000
440     555000
473     440000

```

477	380000
496	430000
515	402861
527	446261
591	451950
661	402000
664	423000
688	392000
691	755000
769	538000
774	395000
798	485000
803	582933
825	385000
898	611657
987	395192
1046	556581
1142	424870
1169	625000
1181	392500
1182	745000
1243	465000
1268	381000
1353	410000
1373	466500
1437	394617

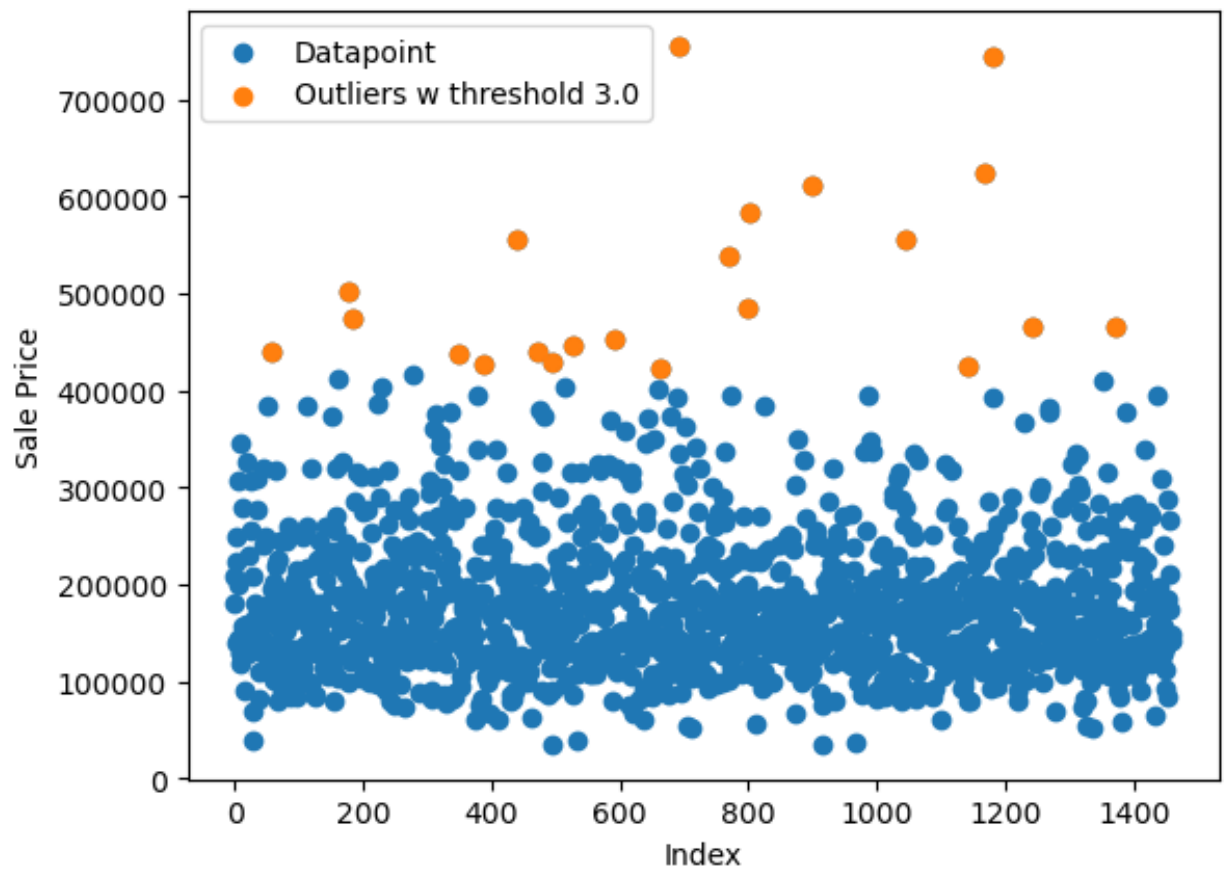
Name: SalePrice, dtype: int64



```

Threshold 3.0
Outliers  58      438780
178      501837
185      475000
349      437154
389      426000
440      555000
473      440000
496      430000
527      446261
591      451950
664      423000
691      755000
769      538000
798      485000
803      582933
898      611657
1046     556581
1142     424870
1169     625000
1182     745000
1243     465000
1373     466500
Name: SalePrice, dtype: int64

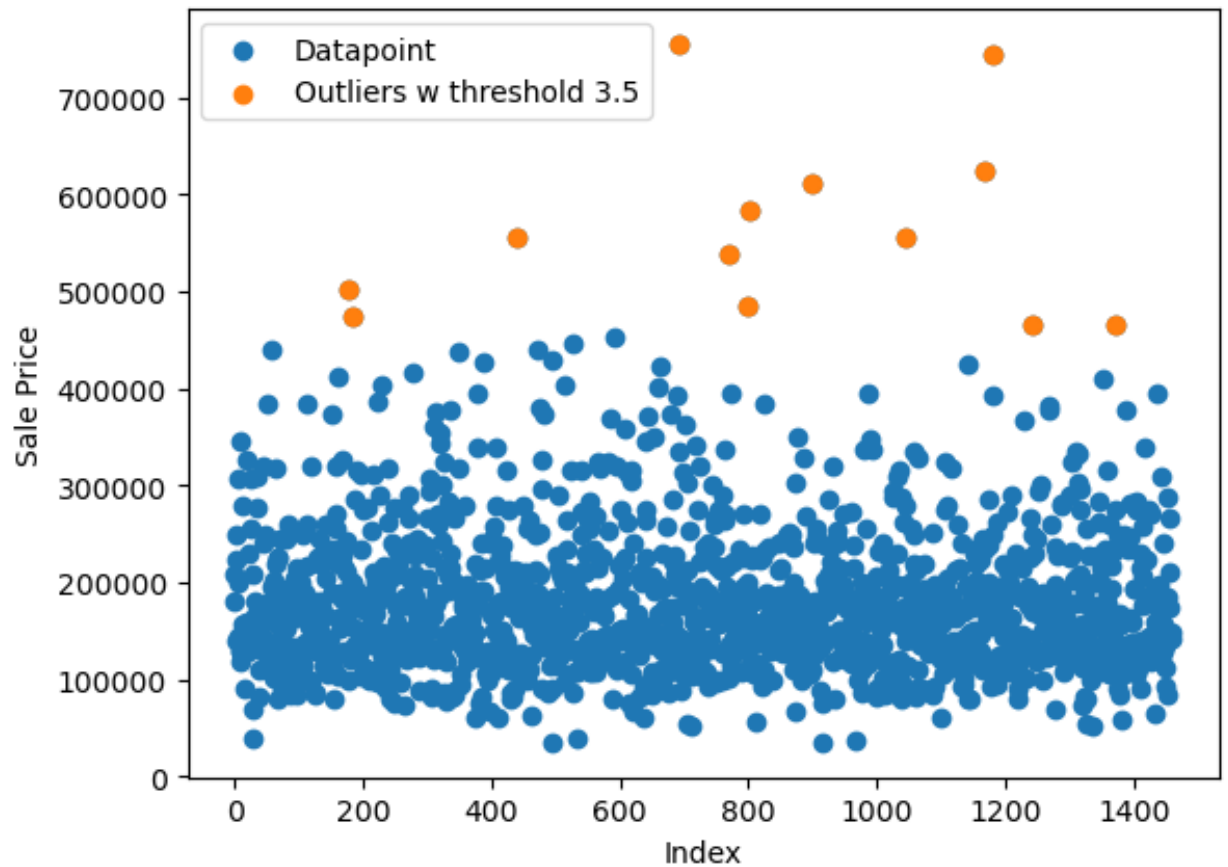
```



```

Threshold 3.5
Outliers 178      501837
185      475000
440      555000
691      755000
769      538000
798      485000
803      582933
898      611657
1046     556581
1169     625000
1182     745000
1243     465000
1373     466500
Name: SalePrice, dtype: int64

```

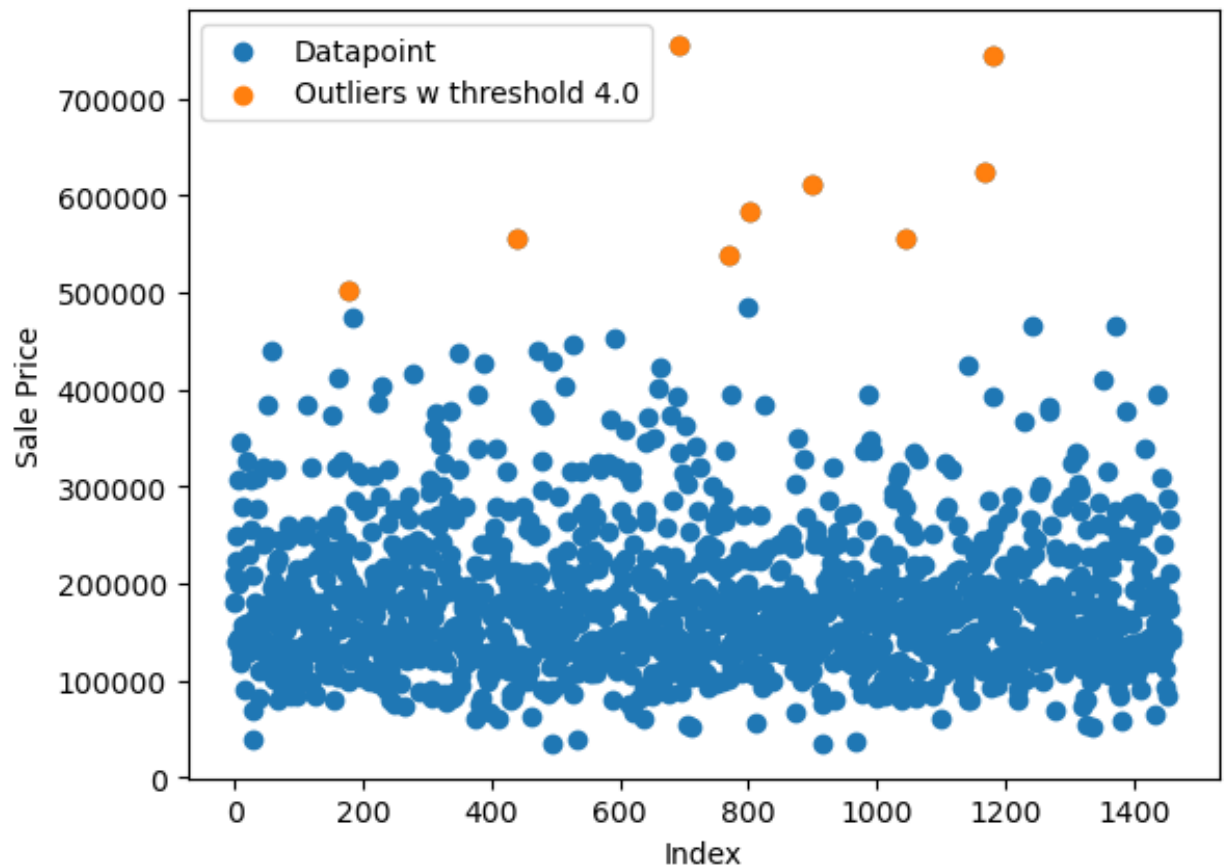


```

Threshold 4.0
Outliers 178      501837
440      555000

```

```
440      555000
691      755000
769      538000
803      582933
898      611657
1046     556581
1169     625000
1182     745000
Name: SalePrice, dtype: int64
```



In []:

Step 4 - Use LocalOutlierFactor as discussed in the class to plot the outliers from SalePrice and LotArea columns.

```
In [88]: CODE
dLot = data[["SalePrice", "LotArea"]].values
```

```

LocalOutlierFactor(n_neighbors=20, contamination=0.1)

fit_predict to compute the predicted labels of the training samples
    = lor.fit_predict(saleAndLot)
r_score = lor.negative_outlier_factor_

of outliers.
of_outlier= (y_pred== -1).sum()

er plot
atter(saleAndLot[:,0], saleAndLot[:,1], color="k", s=3.0, label="Data

e the radius of the circles with outlier scores
    = (outlier_score.max() - outlier_score) / (outlier_score.max() - outl

circles
atter(saleAndLot[:,0], saleAndLot[:,1],
        s=1000*radius,
        edgecolors="r",
        facecolors="none",
        label="Outlier scores",
    )

missing the plot
is("tight")

abel("Lot Area")
abel("Sale Prices")

im((data["SalePrice"].min(), data["SalePrice"].max()))
im((data["LotArea"].min(), data["LotArea"].max()))

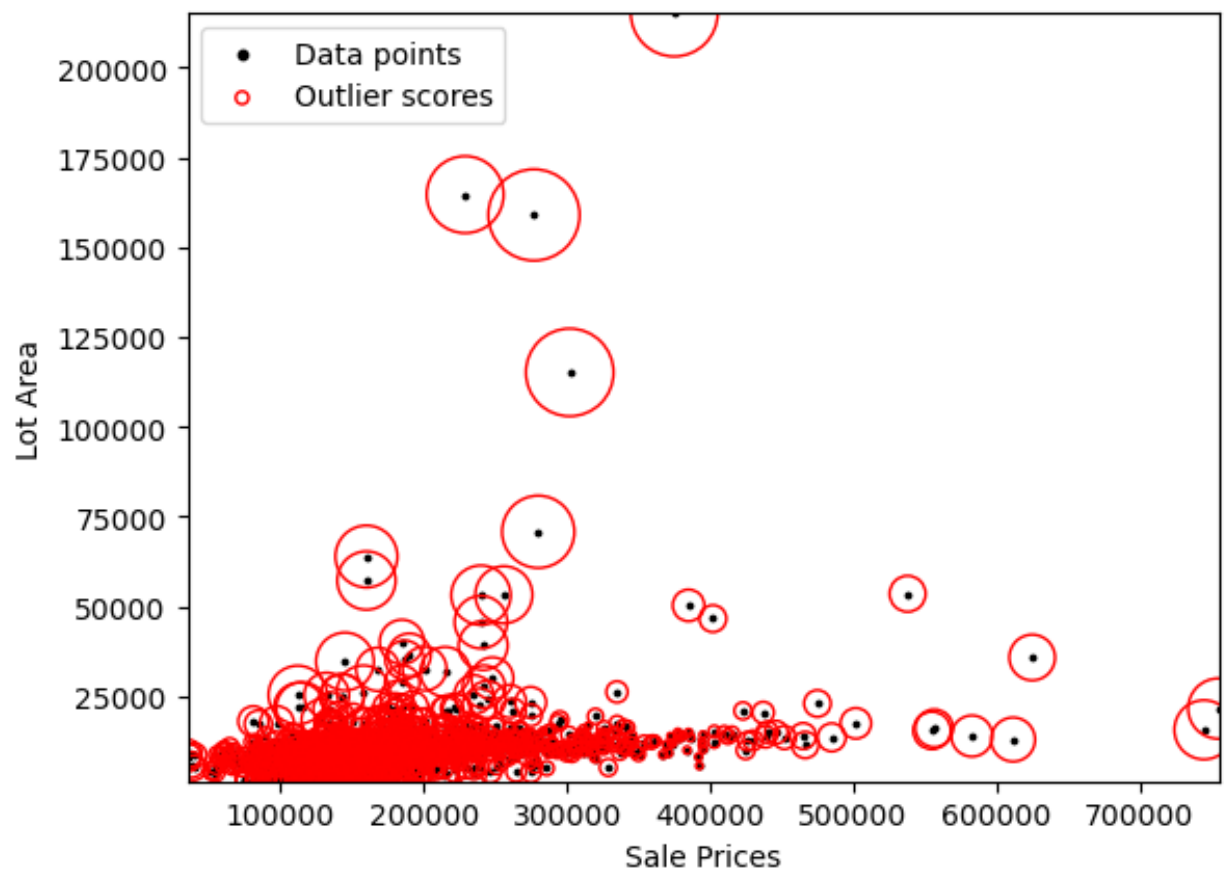
    = plt.legend(loc="upper left")
    .legendHandles[0]._sizes = [10]
    .legendHandles[1]._sizes = [20]

"\nNo. of Outliers are : ", count_of_outlier)

ow()

```

No. of Outliers are : 146



Q 2 - PCA

```
In [11]: # Importing necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

dataset=pd.read_csv("/Users/mumukshapant/Downloads/ML/Assignments/Assi
dataset.shape
```

```
Out[11]: (756, 755)
```

```
In [12]: # To get information about dataset :
info=dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Columns: 755 entries, id to class
dtypes: float64(749), int64(6)
memory usage: 4.4 MB
```

2.1) Seperate and standardize the disease classification dataset.

```
In [13]: X = dataset.drop("class", axis=1) # 1 means to drop columns
y = dataset["class"]
```

Standardise the data

```
In [14]: X_std = preprocessing.scale(X)

#Convert it into dataframe to see it in Tabular Form
df = pd.DataFrame(X_std, columns=X.columns)
print(df.head())
```

	id	gender	PPE	DFA	RPDE	numPulses	\
0	-1.725191	0.968742	0.627644	0.256144	0.605835	-0.846892	
1	-1.725191	0.968742	0.121620	-0.080433	0.368415	-0.907404	
2	-1.725191	0.968742	0.617950	-0.349839	0.733609	-0.927575	
3	-1.711445	-1.032266	-1.980560	1.382279	0.753631	-1.472186	
4	-1.711445	-1.032266	-2.472989	1.398068	0.300123	-0.887233	

	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter
0	-0.842373	0.933328	-0.407251	-0.0549
93	-0.902773	1.040014	-0.426092	-0.1425
1	-0.922907	1.084576	-0.443557	-0.2149
70	-1.466513	2.464215	-0.275316	0.7103
2	-0.882640	0.987044	3.143597	1.1520

	tqwt_kurtosisValue_dec_27	tqwt_kurtosisValue_dec_28	\
0	-0.445877	-0.584822	
1	-0.445730	-0.584895	

2	-0.446030	-0.584767
3	-0.321598	-0.532242
4	-0.300835	-0.475545
	tqwt_kurtosisValue_dec_29	tqwt_kurtosisValue_dec_30 \
0	-0.619412	-0.576762
1	-0.589778	0.193084
2	-0.629033	-0.356261
3	-0.591137	-0.522406
4	-0.521356	-0.490090
	tqwt_kurtosisValue_dec_31	tqwt_kurtosisValue_dec_32 \
0	-0.482286	-0.399331
1	0.016183	-0.067120
2	-0.156055	-0.067593
3	0.008400	-0.449894
4	-0.404833	-0.249678
	tqwt_kurtosisValue_dec_33	tqwt_kurtosisValue_dec_34 \
0	-0.484533	-0.775137
1	-0.175566	-0.526647
2	-0.463462	-0.756063
3	-0.470865	-0.633475
4	-0.042021	-0.419354
	tqwt_kurtosisValue_dec_35	tqwt_kurtosisValue_dec_36
0	-0.814727	-0.366595
1	-0.582972	0.400396
2	-0.804390	-0.780935
3	-0.588387	-0.801583
4	-0.672216	-0.741477

[5 rows x 754 columns]

/Users/mumukshapant/anaconda3/envs/pytorchenv/lib/python3.9/site-packages/sklearn/preprocessing/_data.py:247: UserWarning: Numerical issues were encountered when centering the data and might not be solved. Dataset may contain too large values. You may need to prescale your features.

warnings.warn(

In []:

2.2) Do Eigen decomposition using any LA library of your choice. Display scree plot. (10 points)

Eigen Decomposition

```
In [16]: # Calculating the covariance matrix
cov=df.cov()

#Calculating eigen values
eigen_vals, eigen_vecs = np.linalg.eigh(cov)
```

...

```
In [17]: # print the Eigenvalues
print("Eigenvalues: \n", eigen_vals)
```

```
2.78423314e-20  3.48398474e-20  3.88413213e-20  3.37883383e-20
6.91099254e-20  8.90418970e-20  9.47827857e-20  1.02578701e-19
1.26784647e-19  1.49851309e-19  1.60768492e-19  2.10392652e-19
8.38234165e-15  1.40818189e-14  1.29434226e-13  1.66298199e-13
4.78924545e-13  1.04082078e-12  1.19402347e-12  2.44968536e-12
3.20934483e-12  5.40290178e-12  7.02978091e-12  1.11474666e-11
1.15603719e-11  1.55623515e-11  1.74190240e-11  1.84880452e-11
3.04633034e-11  4.16773206e-11  5.45006969e-11  6.77158577e-11
9.29819670e-11  1.59971259e-10  2.07596053e-10  2.64893692e-10
3.74468240e-10  4.34770285e-10  4.83926499e-10  5.16724558e-10
6.79370932e-10  7.90552682e-10  1.17603580e-09  1.38595516e-09
1.58276466e-09  1.93951874e-09  2.85872929e-09  3.29674530e-09
4.59898598e-09  7.33392866e-09  7.81783889e-09  1.09033326e-08
1.64331798e-08  2.00987644e-08  2.24957799e-08  2.60422079e-08
3.31319381e-08  5.05457671e-08  5.57860993e-08  6.31965364e-08
9.10301134e-08  1.19959698e-07  1.33145463e-07  1.71667431e-07
2.06411489e-07  2.25804289e-07  2.96231254e-07  3.17184620e-07
3.60579165e-07  4.00748115e-07  4.23713646e-07  5.51068554e-07
5.73838742e-07  6.31948399e-07  7.48612555e-07  7.91529456e-07
9.10789400e-07  9.99026416e-07  1.12053755e-06  1.40608615e-06
1.22117100e-06  1.33001700e-06  1.41001100e-06  1.66710000e-06
```

scree plot is a line plot of the eigenvalues of factors . Used for determining no of factors to retain

```
In [28]: # Sorting in DESC order.
eval_sorted = np.flip(eigen_vals)
evec_sorted = np.flip(eigen_vecs, axis=1)
print(eval_sorted)
```

```
2.188387841e-08 1.184331758e-08 1.18383328e-08 7.18178889e-09
7.33392866e-09 4.59898598e-09 3.29674530e-09 2.85872929e-09
1.93951874e-09 1.58276466e-09 1.38595516e-09 1.17603580e-09
7.90552682e-10 6.79370932e-10 5.16724558e-10 4.83926499e-10
4.34770285e-10 3.74468240e-10 2.64893692e-10 2.07596053e-10
1.59971259e-10 9.29819670e-11 6.77158577e-11 5.45006969e-11
4.16773206e-11 3.04633034e-11 1.84880452e-11 1.74190240e-11
1.55623515e-11 1.15603719e-11 1.11474666e-11 7.02978091e-12
5.40290178e-12 3.20934483e-12 2.44968536e-12 1.19402347e-12
1.04082078e-12 4.78924545e-13 1.66298199e-13 1.29434226e-13
1.40818189e-14 8.38234165e-15 2.10392652e-19 1.60768492e-19
1.49851309e-19 1.26784647e-19 1.02578701e-19 9.47827857e-20
8.90418970e-20 6.91099254e-20 5.37805383e-20 3.88415213e-20
3.46398474e-20 2.78423314e-20 1.76820243e-20 1.22952494e-20
6.60989885e-21 2.88411295e-21 -6.71369227e-21 -1.99187807e-20
-2.35991629e-20 -3.18727046e-20 -4.12168792e-20 -4.35013424e-20
-4.81176251e-20 -6.04141537e-20 -7.26039189e-20 -7.48211471e-20
-8.19463246e-20 -9.89039181e-20 -1.12485528e-19 -1.16555034e-19
-1.24006129e-19 -1.29038597e-19 -1.54971347e-19 -1.92552746e-19
-2.74971850e-19 -1.48205413e-16]
```

Explained Variance Ratio

Divide each eigenvalue by the sum of all eigenvalues to get the explained variance ratio for each principal component

```
In [29]: # Explained Variance Ratio forevery eigenvalue
exp_var_ratio = eval_sorted / np.sum(eval_sorted)
```

```
In [30]: # Cumulative of EVR ( This will give us the number of Principal Compon
# The length of "cum_var_exp" is 754 meaning 754 PCs are there.
cum_var_exp = np.cumsum(exp_var_ratio)
len(cum_var_exp)
```

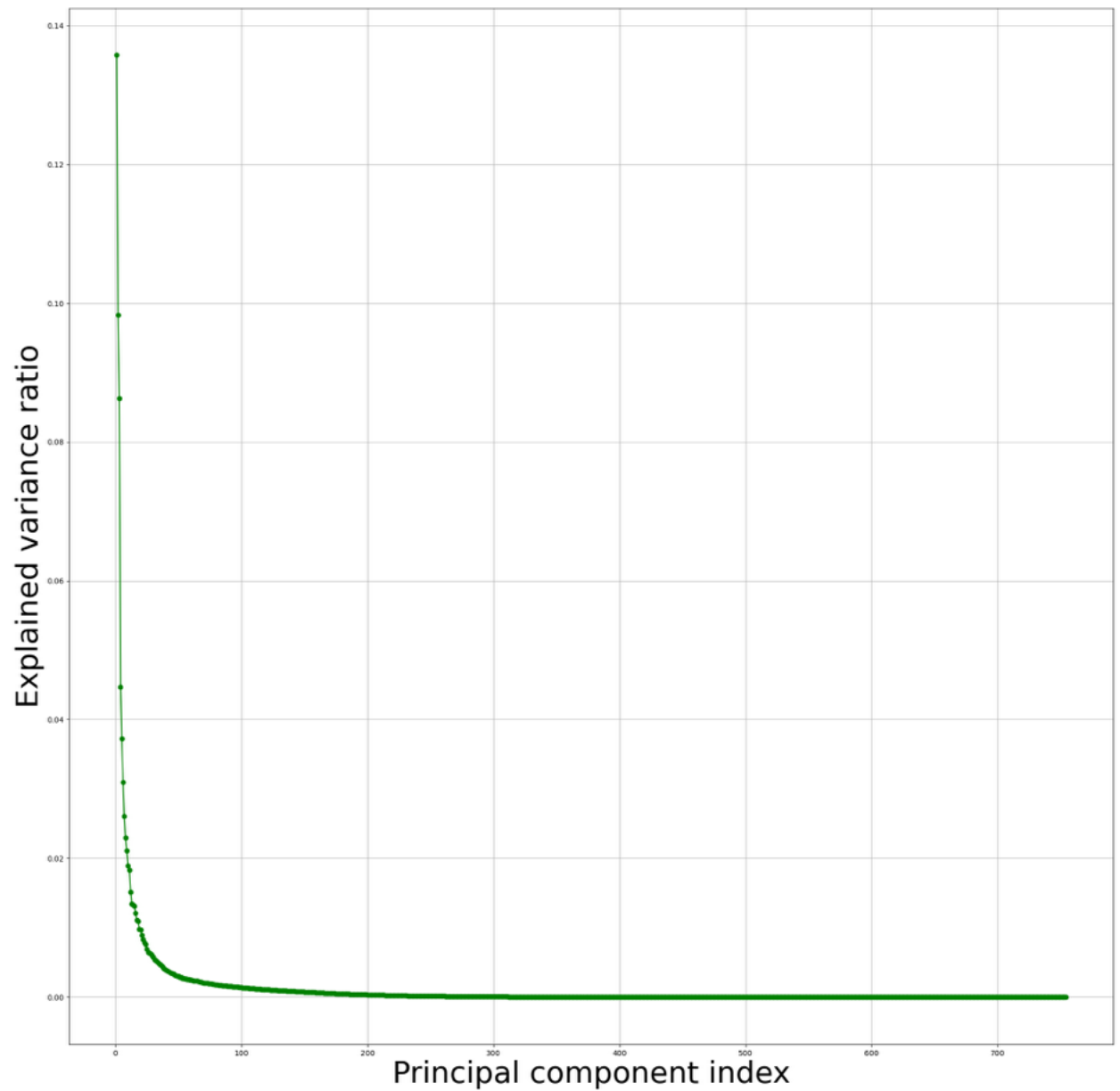
Out[30]: 754

In [53]: *#Now Lets Plot this in Scree Graph*

make a bar plot of the variance associated with each component

```
plt.figure(figsize=(25,25))  
plt.plot(range(1,755), exp_var_ratio, label='Individual Explained Vari'  
plt.grid(True)
```

```
plt.ylabel('Explained variance ratio',fontsize=40)  
plt.xlabel('Principal component index', fontsize=40)  
plt.show()
```



2.3) Primary Component Selection

Primary Component Selection. (Select the first 6 components) (5 points)

```
In [32]: # Primary Component Selection – Select First 6 components  
selected_pc = evec_sorted[:, :6]
```

```
In [67]: selected_pc
```

```
Out[67]: array([[ 0.0073493, -0.00113214,  0.00645179,  0.01483515, -0.022943  
54,          0.02161094],  
          [-0.04502733, -0.04704606, -0.00621151, -0.02455892,  0.010382  
42,          0.00845405],  
          [ 0.01737329, -0.01053272, -0.05773324, -0.01277535, -0.005155  
63,          0.02344555],  
          ...,  
          [-0.01325227,  0.00853193, -0.02178402, -0.01337081,  0.035945  
81,         -0.02532667],  
          [-0.01943189,  0.01712986, -0.02276268, -0.01811131,  0.023486  
2,          -0.02214931],  
          [-0.02903146,  0.03566218, -0.02912439, -0.0245805,  0.014237  
84,         -0.02128807]])
```

2.4) Projected In a New Feature Space

take the dot product of the centered data with the matrix formed by the selected eigenvectors. Each row in the resulting matrix represents the transformed data point in the new feature space. The transformed data in the new feature space is often referred to as the "scores" of the principal components.

```
In [33]: projected = np.dot(X_std, selected_pc)
projected
```

```
Out[33]: array([[ -10.03502652,  -1.48001792,  -6.86079382,   0.74234696,
         3.50698233,  -0.12879346],
        [ -10.65564499,  -1.60059183,  -6.81985852,  -1.38472895,
         3.1615515 ,   1.56188126],
        [ -13.52369673,   1.23889751,  -6.82535545,  -1.43849935,
         2.27903112,   2.45921587],
        ...,
        [   8.29457088,  -2.3699335 ,  -0.91200327,   2.17403474,
         0.98473691,   1.04919861],
        [   4.01690645,  -5.35231374,  -0.82366311,   3.99787378,
        -0.25536571,   0.91921577],
        [   3.99782396,  -6.09710837,  -1.98566314,   1.9253788 ,
        -2.19041036,  -2.47473713]])
```

```
In [ ]:
```

2.5) Principal Component Analysis

```
In [34]: # Cumulative of EVR ( This will give us the number of Principal Compon
# The length of "cum_var_exp" is 754 meaning 754 PCs are there.
cum_var_exp = np.cumsum(exp_var_ratio)
len(cum_var_exp) , exp_var_ratio
```

```
Out[34]: (754,
array([[ 1.35716192e-01,   9.82999394e-02,   8.62218386e-02,   4.4660202
3e-02,
        3.72649780e-02,   3.09793804e-02,   2.60872410e-02,   2.2990511
9e-02,
        2.11165249e-02,   1.89576113e-02,   1.82846515e-02,   1.5167874
1e-02,
        1.34523962e-02,   1.32601329e-02,   1.31622549e-02,   1.2103372
2e-02,
        1.11294553e-02,   1.09146226e-02,   9.77097492e-03,   9.6811587
2e-03,
        8.97427620e-03,   8.29035811e-03,   7.87232824e-03,   7.5847859
2e-03,
        6.91079603e-03,   6.48723155e-03,   6.41414761e-03,   6.2591764
5e-03,
        5.98824403e-03,   5.75738420e-03,   5.38295046e-03,   5.2760154
6e-03,
        5.09703113e-03,   4.91555549e-03,   4.72530032e-03,   4.6894283
7e-03,
        4.41136205e-03,   4.18201415e-03,   4.07027211e-03,   3.87164110
e-03,
        3.71136205e-03,   3.51136205e-03,   3.31136205e-03,   3.11136205
e-03,
        2.91136205e-03,   2.71136205e-03,   2.51136205e-03,   2.31136205
e-03,
        2.11136205e-03,   1.91136205e-03,   1.71136205e-03,   1.51136205
e-03,
        1.31136205e-03,   1.11136205e-03,   9.1136205e-04,   7.1136205e-04,
        5.1136205e-04,   3.1136205e-04,   1.1136205e-04,   9.1136205e-05,
        7.1136205e-05,   5.1136205e-05,   3.1136205e-05,   1.1136205e-05,
        9.1136205e-06,   7.1136205e-06,   5.1136205e-06,   3.1136205e-06,
        1.1136205e-06,   9.1136205e-07,   7.1136205e-07,   5.1136205e-07,
        3.1136205e-07,   1.1136205e-07,   9.1136205e-08,   7.1136205e-08,
        5.1136205e-08,   3.1136205e-08,   1.1136205e-08,   9.1136205e-09,
        7.1136205e-09,   5.1136205e-09,   3.1136205e-09,   1.1136205e-09,
        9.1136205e-10,   7.1136205e-10,   5.1136205e-10,   3.1136205e-10,
        1.1136205e-10,   9.1136205e-11,   7.1136205e-11,   5.1136205e-11,
        3.1136205e-11,   1.1136205e-11,   9.1136205e-12,   7.1136205e-12,
        5.1136205e-12,   3.1136205e-12,   1.1136205e-12,   9.1136205e-13,
        7.1136205e-13,   5.1136205e-13,   3.1136205e-13,   1.1136205e-13,
        9.1136205e-14,   7.1136205e-14,   5.1136205e-14,   3.1136205e-14,
        1.1136205e-14,   9.1136205e-15,   7.1136205e-15,   5.1136205e-15,
        3.1136205e-15,   1.1136205e-15,   9.1136205e-16,   7.1136205e-16,
        5.1136205e-16,   3.1136205e-16,   1.1136205e-16,   9.1136205e-17,
        7.1136205e-17,   5.1136205e-17,   3.1136205e-17,   1.1136205e-17,
        9.1136205e-18,   7.1136205e-18,   5.1136205e-18,   3.1136205e-18,
        1.1136205e-18,   9.1136205e-19,   7.1136205e-19,   5.1136205e-19,
        3.1136205e-19,   1.1136205e-19,   9.1136205e-20,   7.1136205e-20,
        5.1136205e-20,   3.1136205e-20,   1.1136205e-20,   9.1136205e-21,
        7.1136205e-21,   5.1136205e-21,   3.1136205e-21,   1.1136205e-21,
        9.1136205e-22,   7.1136205e-22,   5.1136205e-22,   3.1136205e-22,
        1.1136205e-22,   9.1136205e-23,   7.1136205e-23,   5.1136205e-23,
        3.1136205e-23,   1.1136205e-23,   9.1136205e-24,   7.1136205e-24,
        5.1136205e-24,   3.1136205e-24,   1.1136205e-24,   9.1136205e-25,
        7.1136205e-25,   5.1136205e-25,   3.1136205e-25,   1.1136205e-25,
        9.1136205e-26,   7.1136205e-26,   5.1136205e-26,   3.1136205e-26,
        1.1136205e-26,   9.1136205e-27,   7.1136205e-27,   5.1136205e-27,
        3.1136205e-27,   1.1136205e-27,   9.1136205e-28,   7.1136205e-28,
        5.1136205e-28,   3.1136205e-28,   1.1136205e-28,   9.1136205e-29,
        7.1136205e-29,   5.1136205e-29,   3.1136205e-29,   1.1136205e-29,
        9.1136205e-30,   7.1136205e-30,   5.1136205e-30,   3.1136205e-30,
        1.1136205e-30,   9.1136205e-31,   7.1136205e-31,   5.1136205e-31,
        3.1136205e-31,   1.1136205e-31,   9.1136205e-32,   7.1136205e-32,
        5.1136205e-32,   3.1136205e-32,   1.1136205e-32,   9.1136205e-33,
        7.1136205e-33,   5.1136205e-33,   3.1136205e-33,   1.1136205e-33,
        9.1136205e-34,   7.1136205e-34,   5.1136205e-34,   3.1136205e-34,
        1.1136205e-34,   9.1136205e-35,   7.1136205e-35,   5.1136205e-35,
        3.1136205e-35,   1.1136205e-35,   9.1136205e-36,   7.1136205e-36,
        5.1136205e-36,   3.1136205e-36,   1.1136205e-36,   9.1136205e-37,
        7.1136205e-37,   5.1136205e-37,   3.1136205e-37,   1.1136205e-37,
        9.1136205e-38,   7.1136205e-38,   5.1136205e-38,   3.1136205e-38,
        1.1136205e-38,   9.1136205e-39,   7.1136205e-39,   5.1136205e-39,
        3.1136205e-39,   1.1136205e-39,   9.1136205e-40,   7.1136205e-40,
        5.1136205e-40,   3.1136205e-40,   1.1136205e-40,   9.1136205e-41,
        7.1136205e-41,   5.1136205e-41,   3.1136205e-41,   1.1136205e-41,
        9.1136205e-42,   7.1136205e-42,   5.1136205e-42,   3.1136205e-42,
        1.1136205e-42,   9.1136205e-43,   7.1136205e-43,   5.1136205e-43,
        3.1136205e-43,   1.1136205e-43,   9.1136205e-44,   7.1136205e-44,
        5.1136205e-44,   3.1136205e-44,   1.1136205e-44,   9.1136205e-45,
        7.1136205e-45,   5.1136205e-45,   3.1136205e-45,   1.1136205e-45,
        9.1136205e-46,   7.1136205e-46,   5.1136205e-46,   3.1136205e-46,
        1.1136205e-46,   9.1136205e-47,   7.1136205e-47,   5.1136205e-47,
        3.1136205e-47,   1.1136205e-47,   9.1136205e-48,   7.1136205e-48,
        5.1136205e-48,   3.1136205e-48,   1.1136205e-48,   9.1136205e-49,
        7.1136205e-49,   5.1136205e-49,   3.1136205e-49,   1.1136205e-49,
        9.1136205e-50,   7.1136205e-50,   5.1136205e-50,   3.1136205e-50,
        1.1136205e-50,   9.1136205e-51,   7.1136205e-51,   5.1136205e-51,
        3.1136205e-51,   1.1136205e-51,   9.1136205e-52,   7.1136205e-52,
        5.1136205e-52,   3.1136205e-52,   1.1136205e-52,   9.1136205e-53,
        7.1136205e-53,   5.1136205e-53,   3.1136205e-53,   1.1136205e-53,
        9.1136205e-54,   7.1136205e-54,   5.1136205e-54,   3.1136205e-54,
        1.1136205e-54,   9.1136205e-55,   7.1136205e-55,   5.1136205e-55,
        3.1136205e-55,   1.1136205e-55,   9.1136205e-56,   7.1136205e-56,
        5.1136205e-56,   3.1136205e-56,   1.1136205e-56,   9.1136205e-57,
        7.1136205e-57,   5.1136205e-57,   3.1136205e-57,   1.1136205e-57,
        9.1136205e-58,   7.1136205e-58,   5.1136205e-58,   3.1136205e-58,
        1.1136205e-58,   9.1136205e-59,   7.1136205e-59,   5.1136205e-59,
        3.1136205e-59,   1.1136205e-59,   9.1136205e-60,   7.1136205e-60,
        5.1136205e-60,   3.1136205e-60,   1.1136205e-60,   9.1136205e-61,
        7.1136205e-61,   5.1136205e-61,   3.1136205e-61,   1.1136205e-61,
        9.1136205e-62,   7.1136205e-62,   5.1136205e-62,   3.1136205e-62,
        1.1136205e-62,   9.1136205e-63,   7.1136205e-63,   5.1136205e-63,
        3.1136205e-63,   1.1136205e-63,   9.1136205e-64,   7.1136205e-64,
        5.1136205e-64,   3.1136205e-64,   1.1136205e-64,   9.1136205e-65,
        7.1136205e-65,   5.1136205e-65,   3.1136205e-65,   1.1136205e-65,
        9.1136205e-66,   7.1136205e-66,   5.1136205e-66,   3.1136205e-66,
        1.1136205e-66,   9.1136205e-67,   7.1136205e-67,   5.1136205e-67,
        3.1136205e-67,   1.1136205e-67,   9.1136205e-68,   7.1136205e-68,
        5.1136205e-68,   3.1136205e-68,   1.1136205e-68,   9.1136205e-69,
        7.1136205e-69,   5.1136205e-69,   3.1136205e-69,   1.1136205e-69,
        9.1136205e-70,   7.1136205e-70,   5.1136205e-70,   3.1136205e-70,
        1.1136205e-70,   9.1136205e-71,   7.1136205e-71,   5.1136205e-71,
        3.1136205e-71,   1.1136205e-71,   9.1136205e-72,   7.1136205e-72,
        5.1136205e-72,   3.1136205e-72,   1.1136205e-72,   9.1136205e-73,
        7.1136205e-73,   5.1136205e-73,   3.1136205e-73,   1.1136205e-73,
        9.1136205e-74,   7.1136205e-74,   5.1136205e-74,   3.1136205e-74,
        1.1136205e-74,   9.1136205e-75,   7.1136205e-75,   5.1136205e-75,
        3.1136205e-75,   1.1136205e-75,   9.1136205e-76,   7.1136205e-76,
        5.1136205e-76,   3.1136205e-76,   1.1136205e-76,   9.1136205e-77,
        7.1136205e-77,   5.1136205e-77,   3.1136205e-77,   1.1136205e-77,
        9.1136205e-78,   7.1136205e-78,   5.1136205e-78,   3.1136205e-78,
        1.1136205e-78,   9.1136205e-79,   7.1136205e-79,   5.1136205e-79,
        3.1136205e-79,   1.1136205e-79,   9.1136205e-80,   7.1136205e-80,
        5.1136205e-80,   3.1136205e-80,   1.1136205e-80,   9.1136205e-81,
        7.1136205e-81,   5.1136205e-81,   3.1136205e-81,   1.1136205e-81,
        9.1136205e-82,   7.1136205e-82,   5.1136205e-82,   3.1136205e-82,
        1.1136205e-82,   9.1136205e-83,   7.1136205e-83,   5.1136205e-83,
        3.1136205e-83,   1.1136205e-83,   9.1136205e-84,   7.1136205e-84,
        5.1136205e-84,   3.1136205e-84,   1.1136205e-84,   9.1136205e-85,
        7.1136205e-85,   5.1136205e-85,   3.1136205e-85,   1.1136205e-85,
        9.1136205e-86,   7.1136205e-86,   5.1136205e-86,   3.1136205e-86,
        1.1136205e-86,   9.1136205e-87,   7.1136205e-87,   5.1136205e-87,
        3.1136205e-87,   1.1136205e-87,   9.1136205e-88,   7.1136205e-88,
        5.1136205e-88,   3.1136205e-88,   1.1136205e-88,   9.1136205e-89,
        7.1136205e-89,   5.1136205e-89,   3.1136205e-89,   1.1136205e-89,
        9.1136205e-90,   7.1136205e-90,   5.1136205e-90,   3.1136205e-90,
        1.1136205e-90,   9.1136205e-91,   7.1136205e-91,   5.1136205e-91,
        3.1136205e-91,   1.1136205e-91,   9.1136205e-92,   7.1136205e-92,
        5.1136205e-92,   3.1136205e-92,   1.1136205e-92,   9.1136205e-93,
        7.1136205e-93,   5.1136205e-93,   3.1136205e-93,   1.1136205e-93,
        9.1136205e-94,   7.1136205e-94,   5.1136205e-94,   3.1136205e-94,
        1.1136205e-94,   9.1136205e-95,   7.1136205e-95,   5.1136205e-95,
        3.1136205e-95,   1.1136205e-95,   9.1136205e-96,   7.1136205e-96,
        5.1136205e-96,   3.1136205e-96,   1.1136205e-96,   9.1136205e-97,
        7.1136205e-97,   5.1136205e-97,   3.1136205e-97,   1.1136205e-97,
        9.1136205e-98,   7.1136205e-98,   5.1136205e-98,   3.1136205e-98,
        1.1136205e-98,   9.1136205e-99,   7.1136205e-99,   5.1136205e-99,
        3.1136205e-99,   1.1136205e-99,   9.1136205e-100,   7.1136205e-100,
        5.1136205e-100,   3.1136205e-100,   1.1136205e-100,   9.1136205e-101,
        7.1136205e-101,   5.1136205e-101,   3.1136205e-101,   1.1136205e-101,
        9.1136205e-102,   7.1136205e-102,   5.1136205e-102,   3.1136205e-102,
        1.1136205e-102,   9.1136205e-103,   7.1136205e-103,   5.1136205e-103,
        3.1136205e-103,   1.1136205e-103,   9.1136205e-104,   7.1136205e-104,
        5.1136205e-104,   3.1136205e-104,   1.1136205e-104,   9.1136205e-105,
        7.1136205e-105,   5.1136205e-105,   3.1136205e-105,   1.1136205e-105,
        9.1136205e-106,   7.1136205e-106,   5.1136205e-106,   3.1136205e-106,
        1.1136205e-106,   9.1136205e-107,   7.1136205e-107,   5.1136205e-107,
        3.1136205e-107,   1.1136205e-107,   9.1136205e-108,   7.1136205e-108,
        5.1136205e-108,   3.1136205e-108,   1.1136205e-108,   9.1136205e-109,
        7.1136205e-109,   5.1136205e-109,   3.1136205e-109,   1.1136205e-109,
        9.1136205e-110,   7.1136205e-110,   5.1136205e-110,   3.1136205e-110,
        1.1136205e-110,   9.1136205e-111,   7.1136205e-111,   5.1136205e-111,
        3.1136205e-111,   1.1136205e-111,   9.1136205e-112,   7.1136205e-112,
        5.1136205e-112,   3.1136205e-112,   1.1136205e-112,   9.1136205e-113,
        7.1136205e-113,   5.1136205e-113,   3.1136205e-113,   1.1136205e-113,
        9.1136205e-114,   7.1136205e-114,   5.1136205e-114,   3.1136205e-114,
        1.1136205e-114,   9.1136205e-115,   7.1136205e-115,   5.1136205e-115,
        3.1136205e-115,   1.1136205e-115,   9.1136205e-116,   7.1136205e-116,
        5.1136205e-116,   3.1136205e-116,   1.1136205e-116,   9.1136205e-117,
        7.1136205e-117,   5.1136205e-117,   3.1136205e-117,   1.1136205e-117,
        9.1136205e-118,   7.1136205e-118,   5.1136205e-118,   3.1136205e-118,
        1.1136205e-118,   9.1136205e-119,   7.1136205e-119,   5.1136205e-119,
        3.1136205e-119,   1.1136205e-119,   9.1136205e-120,   7.1136205e-120,
        5.1136205e-120,   3.1136205e-120,   1.1136205e-120,   9.1136205e-121,
        7.1136205e-121,   5.1136205e-121,   3.1136205e-121,   1.1136205e-121,
        9.1136205e-122,   7.1136205e-122,   5.1136205e-122,   3.1136205e-122,
        1.1136205e-122,   9.1136205e-123,   7.1136205e-123,   5.1136205e-123,
        3.1136205e-123,   1.1136205e-123,   9.1136205e-124,   7.1136205e-124,
        5.1136205e-124,   3.1136205e-124,   1.1136205e-124,   9.1136205e-125,
        7.1136205e-125,   5.1136205e-125,   3.1136205e-125,   1.1136205e-125,
        9.1136205e-126,   7.1136205e-126,   5.1136205e-126,   3.1136205e-126,
        1.1136205e-126,   9.1136205e-127,   7.1136205e-127,   5.1136205e-127,
        3.1136205e-127,   1.1136205e-127,   9.1136205e-128,   7.1136205e-128,
        5.1136205e-128,   3.1136205e-128,   1.1136205e-128,   9.1136205e-129,
        7.1136205e-129,   5.1136205e-129,   3.1136205e-129,   1.1136205e-129,
        9.1136205e-130,   7.1136205e-130,   5.1136205e-130,   3.1136205e-130,
        1.1136205e-130,   9.1136205e-131,   7.1136205e-131,   5.1136205e-131,
        3.1136205e-131,   1.1136205e-131,   9.1136205e-132,   7.1136205e-132,
        5.1136205e-132,   3.1136205e-132,   1.1136205e-132,   9.
```

In []:

2.6) Compare the presision and recall for the data using logistic regression before and after PCA.

```
In [35]: ### Split Data  
from sklearn.model_selection import train_test_split  
  
X_train ,X_test, y_train, y_test = train_test_split(X,y, test_size=0.3
```

Logistic Regression

```
In [36]: # instance of a model
logR = LogisticRegression()
logR.fit(X_train, y_train)
y_predicted= logR.predict(X_test)

accuracy = accuracy_score(y_test, y_predicted)
precision = precision_score(y_test, y_predicted)
recall= recall_score(y_test, y_predicted)

print("Logistic Regression Performance Metrics without PCA")
print("Accuracy is", accuracy)
print("Precision is", precision)
print("Recall is ", recall)
```

```
Logistic Regression Performance Metrics without PCA
Accuracy is 0.73568281938326
Precision is 0.7644230769230769
Recall is 0.9352941176470588
```

```
/Users/mumukshapant/anaconda3/envs/pytorchenv/lib/python3.9/site-pack
ages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

```
https://scikit-learn.org/stable/modules/preprocessing.html
(https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options
:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
(https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression)
n_iter_i = _check_optimize_result(
```

PCA

```
In [37]: #Split data into Training and Testing
X_train_pca, X_test_pca, _, _ = train_test_split(projected,y , test_size
```



```
In [38]: logR_pca= LogisticRegression()
logR_pca.fit(X_train_pca, y_train)
y_predicted_pca= logR_pca.predict(X_test_pca)

accuracy_pca= accuracy_score(y_test, y_predicted_pca)
precision_pca = precision_score(y_test, y_predicted_pca)
recall_pca= recall_score(y_test, y_predicted_pca)

print("After PCA :\n ")
print("Accuracy is ", accuracy_pca)
print("Precision is ", precision_pca)
print("Recall is ", recall_pca)
```

After PCA :

```
Accuracy is  0.8105726872246696
Precision is  0.8324607329842932
Recall is    0.9352941176470588
```

Q3 . EM Algorithm (35 points)

Estimate the probability distribution in a 1-dimensional dataset There are two Normal distributions $N(\mu_1, \sigma_1^2)$ and $N(\mu_2, \sigma_2^2)$. There are 5 parameters to estimate: $\theta = (w, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2)$ where w is the probability that the data comes from the first normal probability distribution and $(1-w)$ comes from the second normal probability distribution. The probability density function (PDF) of the mixture model is: $f(x|\theta) = w f_1(x | \mu_1, \sigma_1^2) + (1-w) f_2(x | \mu_2, \sigma_2^2)$ Your goal is to best fit a given probability density by finding $\theta = (w, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2)$ through EM iterations.

Using the following way to produce data:

In [93]:

```
In [60]: # Generating data
random_seed = 36784765
np.random.seed(random_seed)

Mean1 = 9.0
Standard_dev1 = 5.0
Mean2 = 2.0
Standard_dev2 = 2.0

# generate data
y1 = np.random.normal(Mean1, Standard_dev1, 500)
y2 = np.random.normal(Mean2, Standard_dev2, 2000)
data = np.append(y1, y2)

class Gaussian:
    "Model univariate Gaussian"
    def __init__(self, mu, sigma):
        # Mean and standard deviation
        self.mu = mu
        self.sigma = sigma

    def pdf(self, datum):
        "Probability of a data point given the current parameters"
        u = (datum - self.mu) / abs(self.sigma)
        y = (1 / (np.sqrt(2 * np.pi) * abs(self.sigma))) * np.exp(-u *
        return y

# Fitting Gaussian to the data
mu_est = data.mean()
std_est = data.std()
g_est = Gaussian(mu_est, std_est)
```

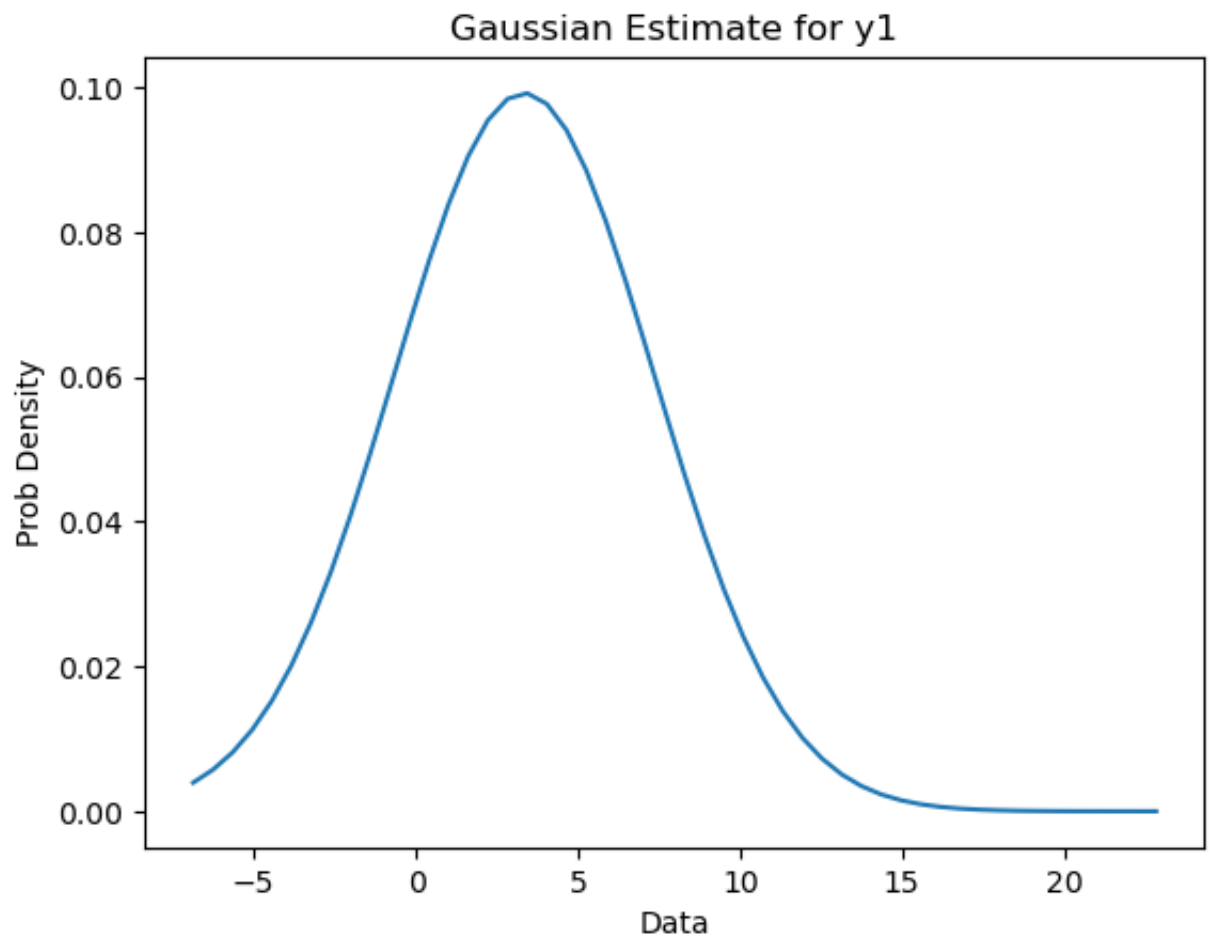
In []:

```
In [61]: # y1
x_vals = np.linspace(min(y1), max(y1))

plt.plot(x_vals, g_est.pdf(x_vals))

plt.xlabel('Data')
plt.ylabel('Prob Density')
plt.title('Gaussian Estimate for y1')

plt.show()
```

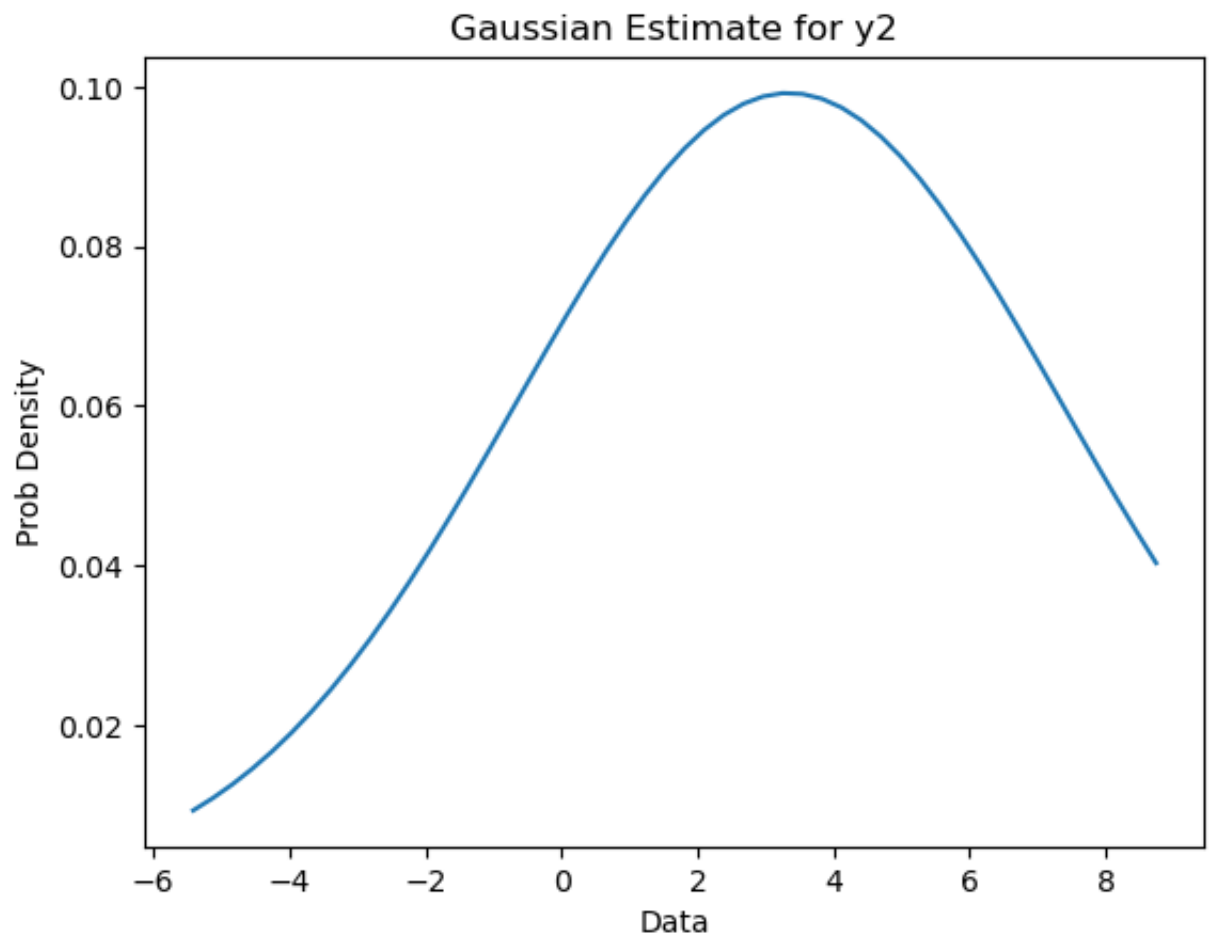


```
In [62]: # y2
x_vals = np.linspace(min(y2), max(y2))

plt.plot(x_vals, g_est.pdf(x_vals))

plt.xlabel('Data')
plt.ylabel('Prob Density')
plt.title('Gaussian Estimate for y2')

plt.show()
```



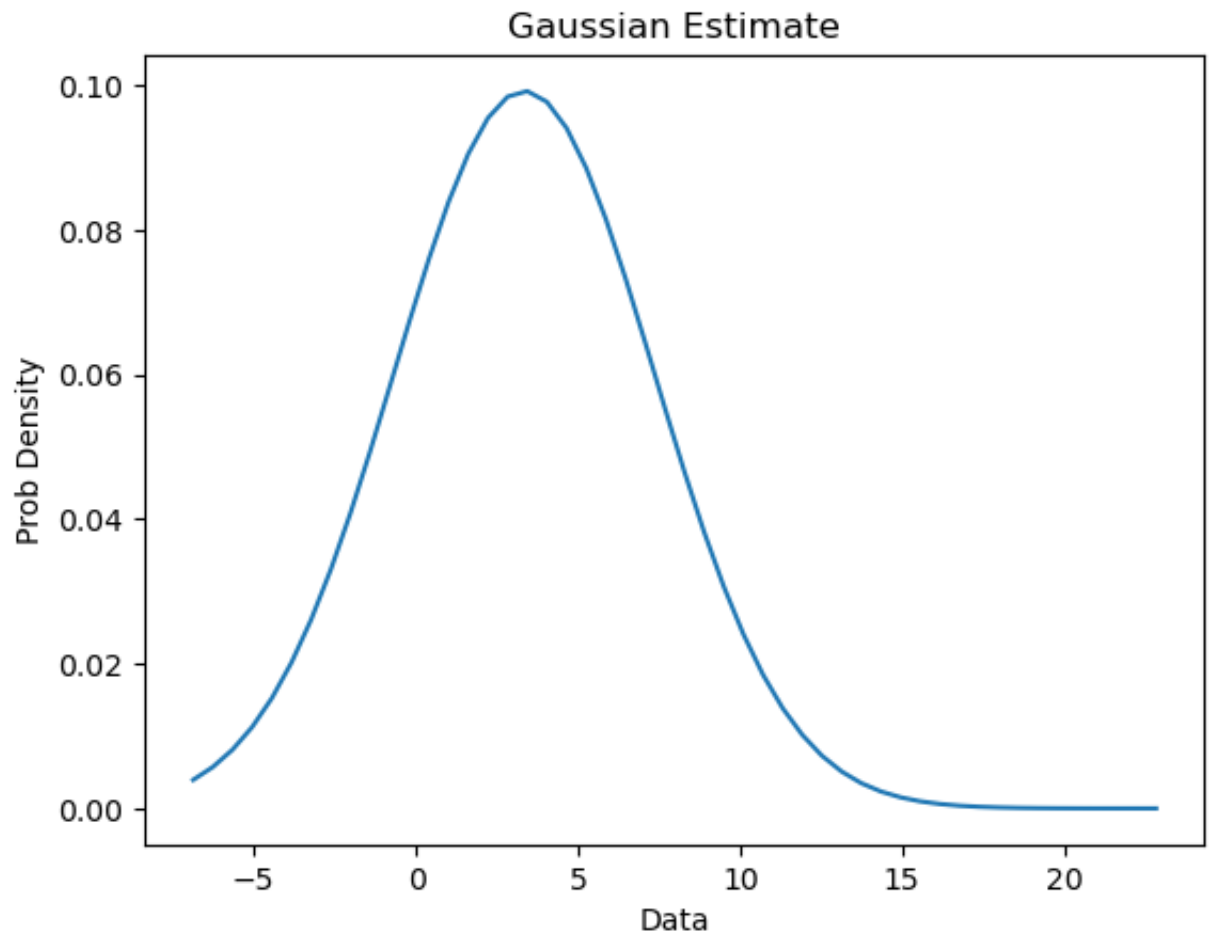
```
In [63]: # combined y1 and y2

x_vals = np.linspace(min(data), max(data))

plt.plot(x_vals, g_est.pdf(x_vals))

plt.xlabel('Data')
plt.ylabel('Prob Density')
plt.title('Gaussian Estimate')

plt.show()
```



```
In [ ]:
```

```
In [76]: # ( 2 PART )
class GaussianMixture_self:

    def __init__(self, data, mu_min=None, mu_max=None, sigma_min=1, si
        # Data points
        self.data = data
```

```

        # Initialize two Gaussian components with specified min/max va
        self.g1 = Gaussian(mu_min if mu_min is not None else min(data)
        self.g2 = Gaussian(mu_max if mu_max is not None else max(data)

        self.mix = mix

    def Estep(self):
        "Perform an E(stimation)-step"
        # Calculate the probability of each data point belonging to Ga
        pdf1 = self.g1.pdf(self.data)
        pdf2 = self.g2.pdf(self.data)

        # Responsibilities of both Gaussian for each dpoint
        res1 = pdf1 * self.mix
        res2 = pdf2 * (1 - self.mix)

        # Normalise the responsibilities
        self.wt = res1 / (res1 + res2)

    def Mstep(self):
        "Perform an M(aximization)-step"

        # Updating mu & sigma basis wts.
        self.g1.mu = np.sum(self.wt * self.data) / np.sum(self.wt)
        self.g2.mu = np.sum((1 - self.wt) * self.data) / np.sum(1 - se

        self.g1.sigma = np.sqrt(np.sum(self.wt * (self.data - self.g1.
        self.g2.sigma = np.sqrt(np.sum((1 - self.wt) * (self.data - se

    def iterate(self, N=1, verbose=False):
        "Perform N iterations, then compute log-likelihood"
        for i in range(N):
            # E-step
            self.Estep()

            # M-step
            self.Mstep()

    def pdf(self, x):
        # Probability density function of the mixture model
        return self.mix * self.g1.pdf(x) + (1 - self.mix) * self.g2.pc

class Gaussian:
    "Univariate Gaussian distribution"

    def __init__(self, mu, sigma):
        self.mu = mu
        self.sigma = sigma

    def pdf(self, x):
        # Probability density function of the Gaussian distribution
        u = (x - self.mu) / abs(self.sigma)

```

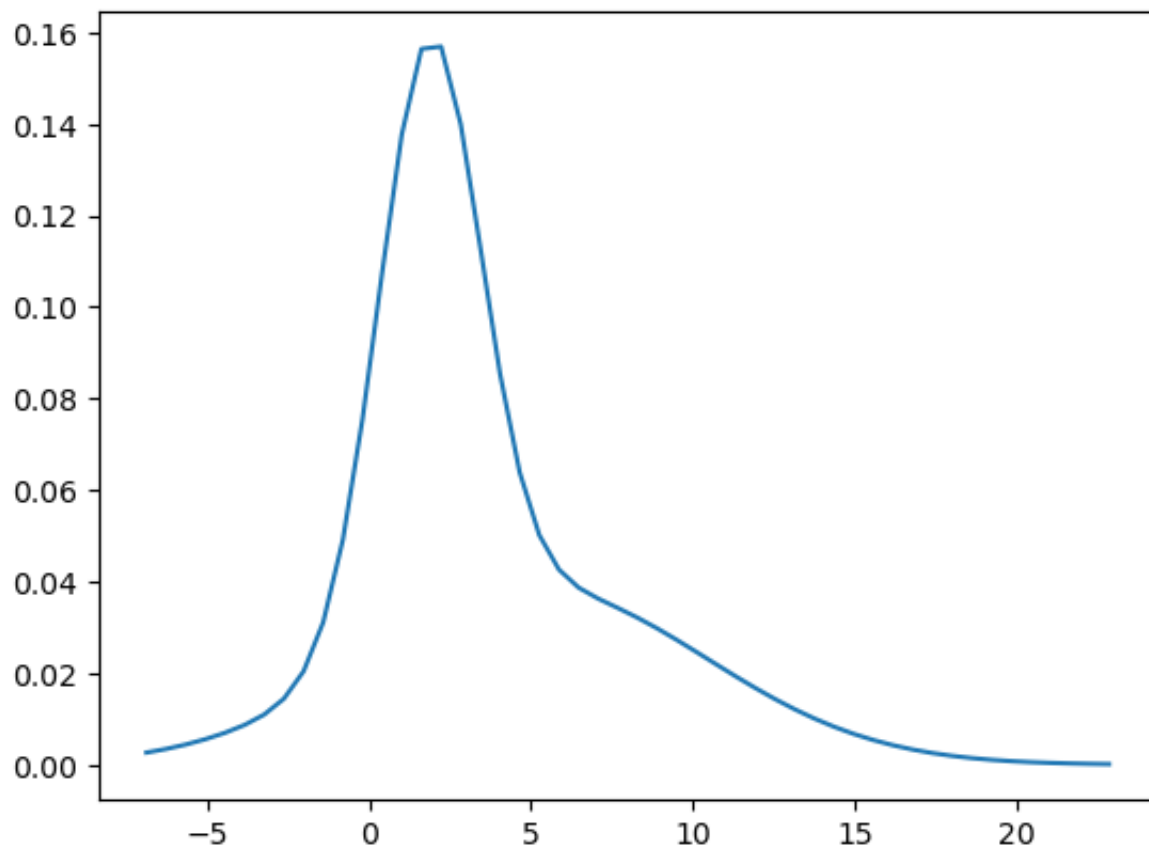
```
y = (1 / (np.sqrt(2 * np.pi) * abs(self.sigma))) * np.exp(-u *  
return y
```

```
In [81]: # Instantiate class with data  
gmm = GaussianMixture_self(data)  
  
# EM iterations  
gmm.iterate(N=100)
```

```
In [83]: # Plot graph
x = np.linspace(min(data), max(data))

plt.plot(x, gmm.pdf(x))

plt.show()
```



In []:

In []: