Homework 1

Name: Abdullah Moizuddin

Class: CS 422

'Recitation Exercises'

1. #### Exercise 1 **A.** Dividing customers of a company according to their gender could be a data preprocessing task. I think a query would accomplish this.

   B. Dividing customers of a company according to their profitability could be a data preprocessing task. Like separating gender, this can be accomplished by a query with specified thresholds that define what is ‘profitable’ vs ‘not as profitable’

   C. Computing the total sales of a company is more of a task that can be accomplished by a few lines of code. I don’t think its a data mining task as its just basic computation.

   D. Sorting a student database based off of student ID numbers is more of a data preprocessing task. As the only operation being done on this is a sort so that can be done using an SQL query.

   E. Predicting the outcomes of a fair dice toss is a simple probability. If the dice wasn’t fair then predicting its outcome could be a data mining task. But because it’s fair, there is an equal chance for it to land on each side.

   F. I think predicting future stock prices is a data mining task. Using a dataset of historical records to predict which stock will rise and will plummet could help people invest in stocks which will make them money.

   G. Monitoring a patient’s heart rate could also be a data mining activity. By monitoring heart rate and looking for abnormalities, you can use that information to identify or predict signs of heart disease or diabetes.

   H. Monitoring seismic waves could also be a data mining activity. We could monitor this activity and use a model to classify what is normal vs abnormal activity.

   I. Extracting frequencies of a sound wave isn’t a data mining task. It’s more of a processing task than anything else.

2. #### Exercise 2 **A.** Binary, Qualitative, Nominal

   B. Continuous, Quantitative, Ratio

   C. Discrete, Qualitative, Ordinal

   D. Continuous, Quantitative, Ratio

   E. Discrete, Qualitative, Ordinal

3. #### Exercise 7 Daily rainfall is dependant on the amount of water that evaporated in the days prior. So if there is a very hot day there might be a lot of rain in the next few days due to evaporation. Rainfall is inherently more scattered. However daily temperatures are more closely related. If it is the middle of summer its likely that there won’t be snow anytime soon. So, daily temperature shows more temporal autocorrelation.

4. #### Exercise 15 **A.** This is proportional sampling. The sample is formed by each of the k groups. Thus, you have a 'representative' from each group and get a sample that is proportional to the the dataset. It is beneficial to use this sampling scheme if there isnt much diversity in each group. It is easier to maintian diverity of the dataset if there is a representative from each group. Also, the variance from the data set to the sample is maintained when you do proportional sampling.

   B. Random sampling, even though its easier, randomly selects n elements from the dataset without any regard to its grouping. Which isn't a good thing because it doesnt maintain the variance of the dataset which is what we need.

5. #### Exercise 16 $tf\prime_{ij} = tf_{ij} * log\frac{m}{df_i}$

   Where,
   df_i is the number of documents the i^{th} term appears.

   $tf\prime_{ij}$ is the frequencey of the i^{th} term in the the j^{th} document.

   m is the number of documents.

   A. If a term occurs in one document, the formula becomes, $tf\prime_{ij} = tf_{ij} * log\frac{m}{1}$ which is the maximum value. and the value of $tf\prime_{ij}$ will be high. This the i^{th} term is significant and it effects the measure of simmilarity between documents. If the term is in every documents then, $log\frac{m}{df_i}$ will go towards 0. This means that the term is common and it can be found in every document.

   B. I think the purpose of this transformation is to see how important the term is. The common terms will be eliminated from the simmilarity calculation and only the important terms wil remain.

6. #### Exercise 17 **A.** The interval in which x^* has a linear relationship to attribute y is (a, b).
   So, x^* in terms of x is $\sqrt{x} = x^*$.
   Then, x in terms of x^* is $(x^*)^2 = x$ and the interval in terms of x is (a^2, b^2)

   B. $y = x^2 + C$

7. #### Exercise 18 **A.**
   $x = 0101010001$
   $y = 0100011000$

   Hamming Distance = 3
   Jaccard Measure = $\frac{2}{5}$
   B.
   Hamming Distance is simmilar to Simple Matching Coefficient because simple matching coefficient = hamming distance/# of data points Jaccard Measure is simmilar to cosine simmilarity as both measures ignore 0-0 measures.

   C.
   I think Jaccard would be better suited for this because we want to see the genes that the organisms share and jaccard ignores the 0-0 measures which would make it a suitable measure.

D.

For this I think Hamming would be better as we are trying to see how the genes of the organisms are different. If we were to use Jaccard, we would just see that the two organisms have a simmilarity score very close to 1

8. #### Exercise 19

In [113]:
```
class Vectors(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def corr(self):
        toInt = lambda s: int(s)
        def std_dev(vector):
            size = len(vector)
            mean = sum(map(toInt, vector)) / size
            squareDiff = lambda x : (x - mean)**2
            return ((1 / (size-1)) * (sum (map(squareDiff, (map(toInt
, vector)))))) ** (1 / 2)

        def covar(x, y):
            sizeX = len(x)
            sizeY = len(y)
            meanX = sum(map(toInt, x)) / sizeX
            meanY = sum(map(toInt, y)) / sizeY

            func = lambda tup : (tup[0] - meanX) * (tup[1] - meanY)

            return (1 / (sizeX-1)) * sum(map(func, zip(map(toInt, x),
map(toInt, y))))

        try:
            return covar(self.x, self.y) / (std_dev(self.x) * std_dev
(self.y))
        except:
            return 'undefined'

    def cosSim(self):
        dotProduct = sum(map(lambda tup: int(tup[0]) * int(tup[1]), z
ip(self.x, self.y)))
        magnX = (sum(map(lambda n : int(n)**2, self.x))) ** (1/2)
        magnY = (sum(map(lambda n: int(n)**2, self.y))) ** (1/2)
        return dotProduct / (magnX * magnY)

    def euclidDist(self):
        return sum(map(lambda tup: (tup[0] - tup[1])**2  ,zip(self.x,
self.y)))

    def jaccard(self):
        num = sum(map(lambda tup: 1 if tup[0] == 1 and tup[1]== 1 els
e 0 ,zip(self.x, self.y)))
        denom = sum(map(lambda tup: 1 if tup[0] != 0 or tup[1] != 0 e
lse 0, zip(self.x, self.y)))
        return num / denom
```

A.

Cosine, Correlation, Euclidean

$x = (1, 1, 1, 1)$

$y = (2, 2, 2, 2)$

In [114]:
```
v = Vectors((1, 1, 1, 1),(2, 2, 2, 2))
print("Correlation:", v.corr())
print("Cosine Similarity:", v.cosSim())
print("Euclidean Distance:",v.euclidDist())
```

```
Correlation: undefined
Cosine Similarity: 1.0
Euclidean Distance: 4
```

B.

Cosine, Correlation, Euclidean, Jaccard

$x = (0, 1, 0, 1)$

$y = (1, 0, 1, 0)$

In [115]:
```
v = Vectors((0, 1, 0, 1),(1, 0, 1, 0))
print("Cosine Similarity:", v.cosSim())
print("Correlation:", v.corr())
print("Euclidean Distance:",v.euclidDist())
print("Jaccard:", v.jaccard())
```

```
Cosine Similarity: 0.0
Correlation: -1.0
Euclidean Distance: 4
Jaccard: 0.0
```

C.

Cosine, Correlation, Euclidean

$x = (0, -1, 0, 1)$

$y = (1, 0, -1, 0)$

In [116]:
```
v = Vectors((0, -1, 0, 1),(1, 0, -1, 0))
print("Cosine Similarity:", v.cosSim())
print("Correlation:", v.corr())
print("Euclidean Distance:",v.euclidDist())
```

```
Cosine Similarity: 0.0
Correlation: 0.0
Euclidean Distance: 4
```

D.

Cosine, Correlation, Jaccard

$x = (1, 1, 0, 1, 0, 1)$

$y = (1, 1, 1, 0, 0, 1)$

```
In [117]: v = Vectors((1, 1, 0, 1, 0, 1),(1, 1, 1, 0, 0, 1))
          print("Cosine Similarity:", v.cosSim())
          print("Correlation:", v.corr())
          print("Jaccard:",v.jaccard())
```

```
Cosine Similarity: 0.75
Correlation: 0.25
Jaccard: 0.6
```

E.

Cosine, Correlation

$x = (2, -1, 0, 2, 0, -3)$

$y = (-1, 1, -1, 0, 0, -1)$

```
In [118]: v = Vectors((2, -1, 0, 2, 0, 3),(-1, 1, -1, 0, 0, -1))
          print("Cosine Similarity:", v.cosSim())
          print("Correlation:", v.corr())
```

```
Cosine Similarity: -0.7071067811865476
Correlation: -0.6324555320336757
```

Practicum Problems

Problem 2.1

```
In [119]: import numpy as np
          import seaborn as sn
          import matplotlib.pyplot as plt
          import seaborn as sn
          import pandas as pd
          from sklearn.impute import SimpleImputer as Imputer
          %matplotlib inline
```

Visualization of the dataset

In [120]:
```
ds  = sn.load_dataset('titanic')
ds
```

Out[120]:

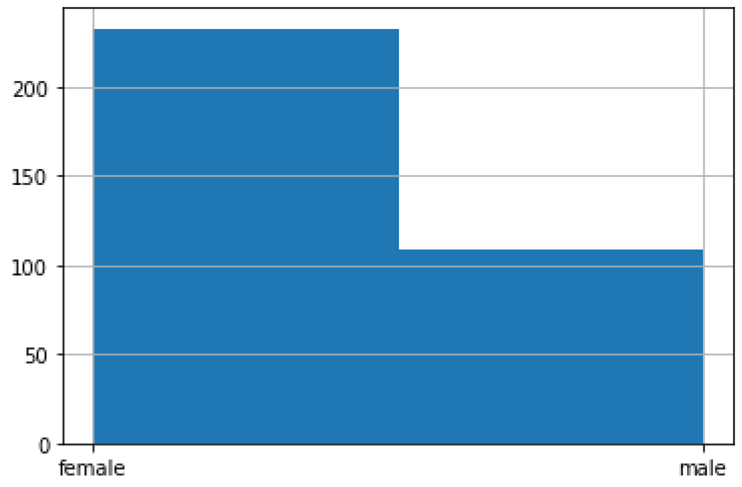
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_r
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	F
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	F
3	1	1	female	35.0	1	0	53.1000	S	First	woman	F
4	0	3	male	35.0	0	0	8.0500	S	Third	man	
...	...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	0	0	13.0000	S	Second	man	
887	1	1	female	19.0	0	0	30.0000	S	First	woman	F
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	F
889	1	1	male	26.0	0	0	30.0000	C	First	man	
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	

891 rows × 15 columns

Gender Distribution of the survivors

In [121]:
```
survivors = ds[ds.survived == 1]
survivors['sex'].hist(bins=2)
```

Out[121]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff90afd99d0>
```

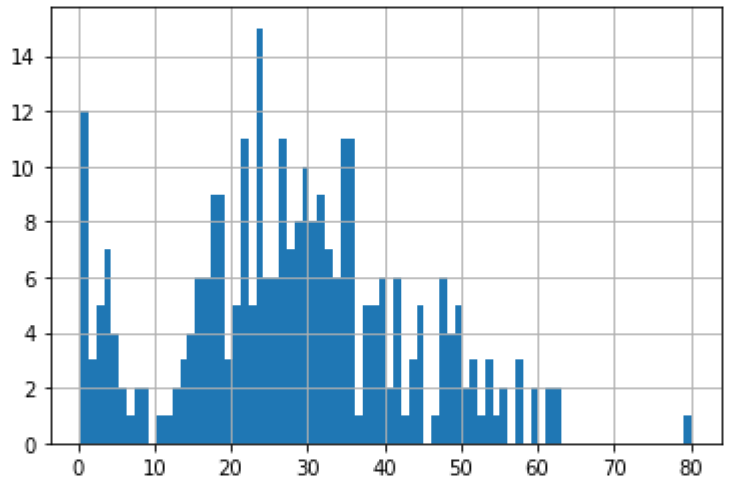


From this we can see that the count of males that survived were significantly lower thatn the females that survived. This seems accurate because they were prioritizing women and children.

Age distribution of the survivors

```
In [122]: survivors['age'].hist(bins=80)
```

```
Out[122]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff90a1821d0>
```



From this we can see that most of the survivors were between 15-35. Another thing to notice is that there was also a priority in saving children which is why children 5 and under have a high count of being saved.

Problem 2.2

I created a mapping towards the different datasets that I am going to use. I'm read_csving to each dataset object because copying it from a main dataset ended up giving me the same values after I imputed.

In [139]:
```
path = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto
-mpg/auto-mpg.data'
features = ['mpg', 'cylinders','displacement','horsepower','weight',
            'acceleration','model year', 'origin','car name']

datasets = {"normal":None,
            "mean":None,
            "median":None,
            "mode":None}

for dataset in datasets:
    datasets[dataset] = pd.read_csv(path, delim_whitespace=True, na_v
alues=['?'],names = features)
    datasets[dataset] = datasets[dataset].replace('?',np.nan)
datasets["normal"].describe()
```

Out[139]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model yea
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.00000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.01005
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.69762
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.00000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.00000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.00000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.00000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.00000

Mean Imputation Strategy

In [140]:
```
imp = Imputer(missing_values = np.nan, strategy = "mean")
datasets["mean"].horsepower = imp.fit_transform(datasets["mean"][["ho
rsepower"]])
datasets["mean"]["horsepower"].var()
```

Out[140]:
```
1459.1779160026776
```

Median Imputation Strategy

In [141]:
```
imp = Imputer(missing_values = np.nan, strategy = "median")
datasets["median"].horsepower = imp.fit_transform(datasets["median"]
[["horsepower"]])
datasets["median"]["horsepower"].var()
```

Out[141]:
```
1460.96905180816
```

Mode Imputation Strategy

```
In [142]: imp = Imputer(missing_values = np.nan, strategy="most_frequent")
          datasets["mode"].horsepower = imp.fit_transform(datasets["mode"][["ho
          rsepower"]])
          datasets["mode"]["horsepower"].var()

Out[142]: 1490.0361252104324
```

From this we can see that when the missing values are replaced with the mean the variance is less. This is because variance is a measure of how far data points deviate from the mean. If the missing data points are replaced with the mean, then they are going to be closer to the mean. I think the ideal method of imputing values would be to predict what the horsepower will be based off of the cylinders and displacement of each car. As seen in the statical summary, as cylinder count and displacement increase horsepower increases also. Thus, we can come up with a more accurate estimate of the horsepower a car will have.

Problem 2.3

```
In [243]: from sklearn.decomposition import PCA
          from sklearn.metrics import explained_variance_score
          path = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iri
          s/iris.data'
          features = ['sepal_len','sepal_width','petal_len', 'petal_width','cla
          ss']
          iris_ds = pd.read_csv(path,names = features)
          featurelist = ['sepal_len','sepal_width','petal_len', 'petal_width']
          featureMat = iris_ds[featurelist]
```

PCA for 3 Principal Components

```
In [232]: pca3 = PCA(n_components=3)
          pca3.fit(featureMat[['sepal_len','sepal_width','petal_len', 'petal_wi
          dth']])
          for(x,var)in enumerate(pca3.explained_variance_ratio_):
              print("PC"+str(x+1), "variance ratio:", var )

          PC1 variance ratio: 0.9246162071742684
          PC2 variance ratio: 0.053015567850534955
          PC3 variance ratio: 0.017185139525006779
```

PCA for 2 Principal Compnents

In [233]:
```
pca2 = PCA(n_components=2)
pca2.fit(featureMat[['sepal_len','sepal_width','petal_len', 'petal_wi
dth']])
for(x,var)in enumerate(pca2.explained_variance_ratio_):
    print("PC"+str(x+1), "variance ratio:", var )
```

```
PC1 variance ratio: 0.9246162071742684
PC2 variance ratio: 0.053015567850534955
```

PCA for 1 Principal Compnents

In [235]:
```
pca1 = PCA(n_components=1)
pca1.fit(featureMat[['sepal_len','sepal_width','petal_len', 'petal_wi
dth']])
for(x,var)in enumerate(pca1.explained_variance_ratio_):
    print("PC"+str(x+1), "variance ratio:", var )
```

```
PC1 variance ratio: 0.9246162071742684
```

Variance for Default Features

In [236]:
```
totalVar =0
for feature in featurelist:
    totalVar += iris_ds[feature].var()
for feature in featurelist:
    print(feature,"variance ratio:", iris_ds[feature].var()/totalVar)
```

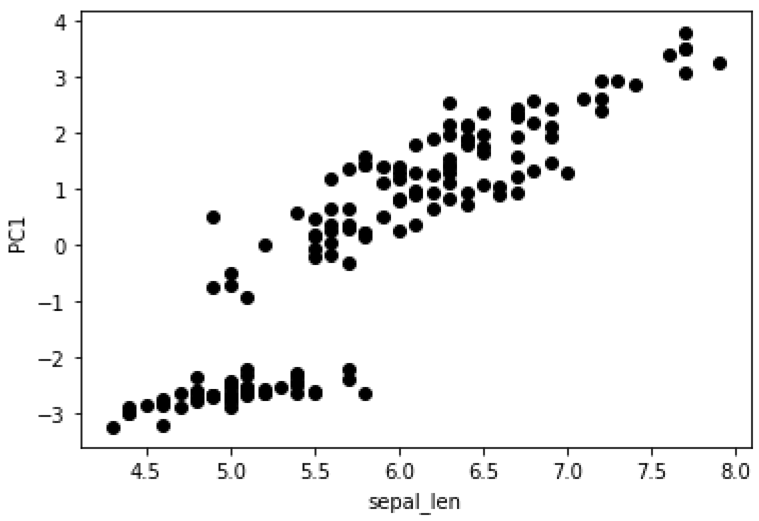
```
sepal_len variance ratio: 0.15006561652804068
sepal_width variance ratio: 0.04114511759566788
petal_len variance ratio: 0.6813265407839865
petal_width variance ratio: 0.1274627250923049
```

As seen from the data, the first principal component has a very high variance score at $.92$. As opposed to the original features which have relativly low variance ratios. The feature with the greatest variance is petal length and that still doesn't have have as much variablilty as the first principal component.

Problem 2.4

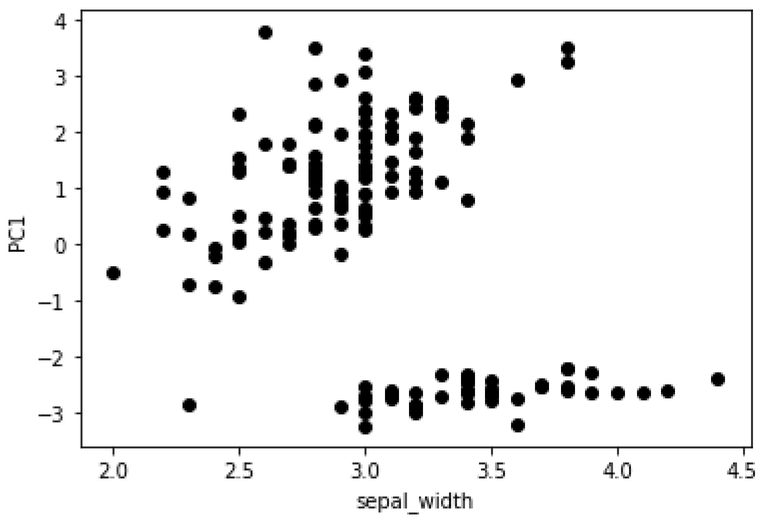
In [250]:
```
PC1 = pca1.fit_transform(featureMat)
PC1= pd.DataFrame(PC1)
for feature in featurelist:
    x = featureMat[feature]
    y = PC1[0]
    plt.scatter(x, y, c='black')
    plt.ylabel('PC1')
    plt.xlabel(feature)
    print(feature, "vs", "PC1")
    plt.show()
    print("correlation coefficient", feature, "vs", "PC1:", np.corrco
ef(x, y)[0][1])
    print("\n")
```

```
sepal_len vs PC1
```



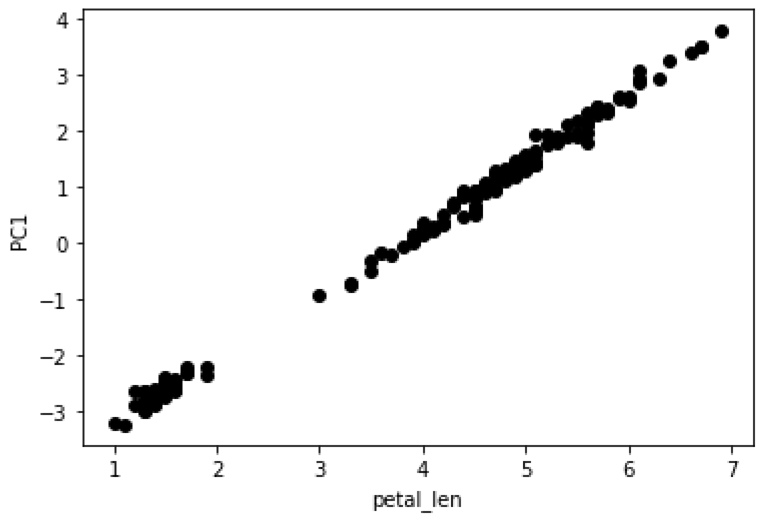
```
correlation coefficient sepal_len vs PC1: 0.8975448849407616
```

```
sepal_width vs PC1
```



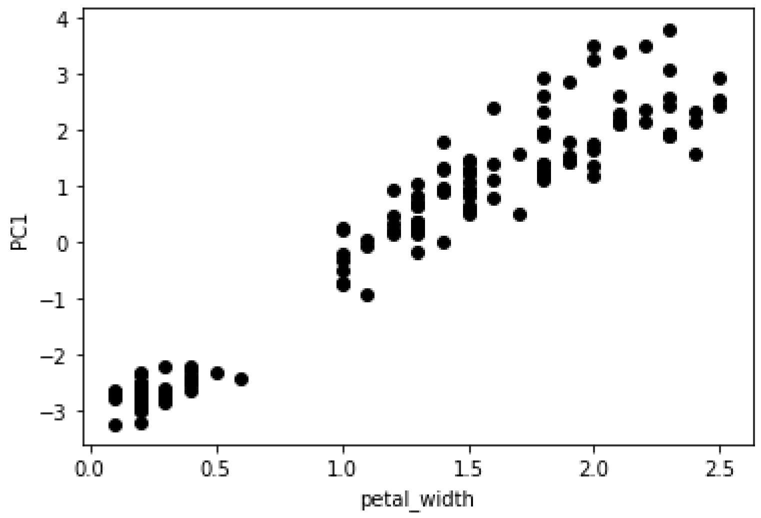
```
correlation coefficient sepal_width vs PC1: -0.3899933790475055
```

```
petal_len vs PC1
```



```
correlation coefficient petal_len vs PC1: 0.9978540506354597
```

```
petal_width vs PC1
```



```
correlation coefficient petal_width vs PC1: 0.9664841831537923
```

As seen from the plots, the feature that correlation to PC1 is petal length. This is understandable as petal length had the greatest variation ratio. Yes, the results agree with the visuals.

Problem 2.5

In [257]:
```
pca = PCA(n_components=4)
dataset_pca = pd.DataFrame(pca.fit_transform(featureMat), columns=['P
C1', 'PC2', 'PC3', 'PC4'])
pca_tot_var = 0

for feature in featurelist:
    print(feature,"variance:", iris_ds[feature].var())
print("feature total variance:", totalVar)

print("\n")

for PC in dataset_pca:
    print("Variance of", PC, "=", dataset_pca[PC].var())
    pca_tot_var += dataset_pca[PC].var()

print("PCA total variance:", pca_tot_var)

print("\n","Variance ratio captured by PC 1 and 2:",(dataset_pca["PC
1"].var() + dataset_pca["PC2"].var()) /totalVar)
```

```
sepal_len variance: 0.6856935123042505
sepal_width variance: 0.18800402684563763
petal_len variance: 3.1131794183445156
petal_width variance: 0.5824143176733784
feature total variance: 4.5692912751677826


Variance of PC1 = 4.224840768320109
Variance of PC2 = 0.24224357162751495
Variance of PC3 = 0.07852390809415456
Variance of PC4 = 0.02368302712600192
PCA total variance: 4.569291275167781

 Variance ratio captured by PC 1 and 2: 0.9776317750248029
```

PC1 captures 92% of the variance by itself. So aftwer we add PC2, we get 97% of variance captured and so we need to reduce the dataset to 2 features to capture atleast 95% of the variance.