Solution 1:

1. Dividing the customers of a company according to their gender.
   No, this is not a Data Mining task it seems like SQL task or Database task.
2. Dividing the customers of a company according to their profitability.
   No, this is not a Data Mining task it seems like SQL task or Database task.
3. Computing the total sales of a company.
   No, this is not a Data Mining task is just aggregation.
4. Sorting a student database based on student identification numbers.
   No, this is a SQL task not a Data mining task.
5. Predicting the outcomes of tossing a (fair) pair of dice.
   No, this is not a Data mining task. Just a probability task.
6. Predicting the future stock price of a company using historical records.
   Yes, this is a Data Mining task because it is involving the prediction based on previous data.
7. Monitoring the heart rate of a patient for abnormalities.
   Yes, by using the data mining techniques the abnormalities in the patient's heart rate are recorded by comparing them with the normal heart rate observation.
8. Monitoring seismic waves for earthquake activities.
   Yes, because we are analysis the seismic waves and make decision based on activities for earthquake.
9. Extracting the frequencies of a sound wave.
   No, this is just variable defining and preprocessing on data before data mining task.

Solution 2.2:

1. Time in terms of AM or PM.
   Binary, Qualitative, Ordinal.
2. Brightness as measured by a light meter.
   Continuous, Quantitative, Ratio
3. Brightness as measured by people's judgments.
   Discrete, Qualitative, Ordinal
4. Angles as measured in degrees between 0 and 360.
   Continuous, Quantitative, Ratio
5. **Discrete, Qualitative, Ordinal.** # Bronze, Silver, and Gold medals as awarded at the Olympics.
6. **Continuous, Quantitative, Ratio.** # Height above sea level.
7. **Discrete, Quantitative, Ratio** # Number of patients in a hospital.
8. **Discrete, Qualitative, Nominal** # ISBN numbers for books.
9. **Discrete, Qualitative, Ordinal.** # Ability to pass light in
10. **Discrete, Qualitative, Ordinal.** # Military rank.
11. **Continuous, Quantitative, Ratio** # Distance from the center of campus.
12. **Discrete, Quantitative, Ratio** # Density of a substance in grams per cubic centimeter
13. **Discrete, Qualitative, Nominal** # Coat check number

Solution 2.7:

Daily Temperature show more temporal auto correlation because regular or every day temperature are near locations like Chicago and Naperville almost same in term of increasing and decreasing in temperature. When compared to location far away like Chicago and San Diego we can see no strong correlation between them. Rainfall can vary every hour and would be different between the two location.

Solution 2.15:

1. From this first scheme (n*mi)/m our result will have equal number of elements or objects from all K groups. This is an example of stratified sampling.
2. For this 2nd scheme the number of objects from K groups will not be same and equal, there are possible cases where we do not have any objects from some groups. This is a random sampling example and high chance of overfitting.

Solution 2.16:

1. If the team occur in multiple documents or every document, then dfi will be larger and having weight = 0. While those that occur in single document have maximum weight log(m).
2. This normalization reflects the observation that teams that occur in every document do not have power to distinguish one document from other.

Solution 2.17:

1. What is the corresponding interval (*A*, *B*) in terms of *x* ?:
   x = (a^2, b^2)
2. Give an equation that relates *y* to *x*.
   Y = X^2

Solution 2.18:

1. x=0101010001
   y=010**0**011**1**00**0**
   Hamming Distance = number of different bits => **3**

   Jaccard similarity = no. of 1-1 matches/(no. of bits – no. of 0-0 matches)
   = 2/(10-5) => **0.4**

2. The hamming distance is like the SMC Simple Matching Coefficient
   SMC = Hamming Distance/no. of bits

   The Jaccard distance is similar to the cosine measure because both ignore 0-0 matches.

3. Jaccard similarity would suit this. To find the similar genes. We need to find the like genes and omit unlike genes.

4. Since the genetic makeup of two human being have > 99.9% of same shared genes, we need to focus on their differences. Hence the hamming distance is more appropriate in this situation.

Solution 2.19:

1. x=(1, 1, 1, 1), y=(2, 2, 2, 2) cosine, correlation, Euclidean

   cosine: cos(x,y) = x.y/(|x|.|y|) = 8/(2*4) = 1

   correlation: corr(x,y) = covariance(x,y)/sd(x)*sd(y)
   0/0*0 => 0/0 (undefined)

   Euclidean: d(x,y) = square_root[(x1-y1)^2+…+(xn-yn)^2]

   square_root[4] => 2

2. x=(0, 1, 0, 1), y=(1, 0, 1, 0) cosine, correlation, Euclidean, Jaccard

   cosine: cos(x,y) = 0
   correlation: -1
   Euclidean: 2
   Jaccard: 0

3. x=(0, −1, 0, 1), y=(1, 0, −1, 0) cosine, correlation, Euclidean

   cosine: 0
   Correlation: cos(x,y) = 0
   Euclidean: 2

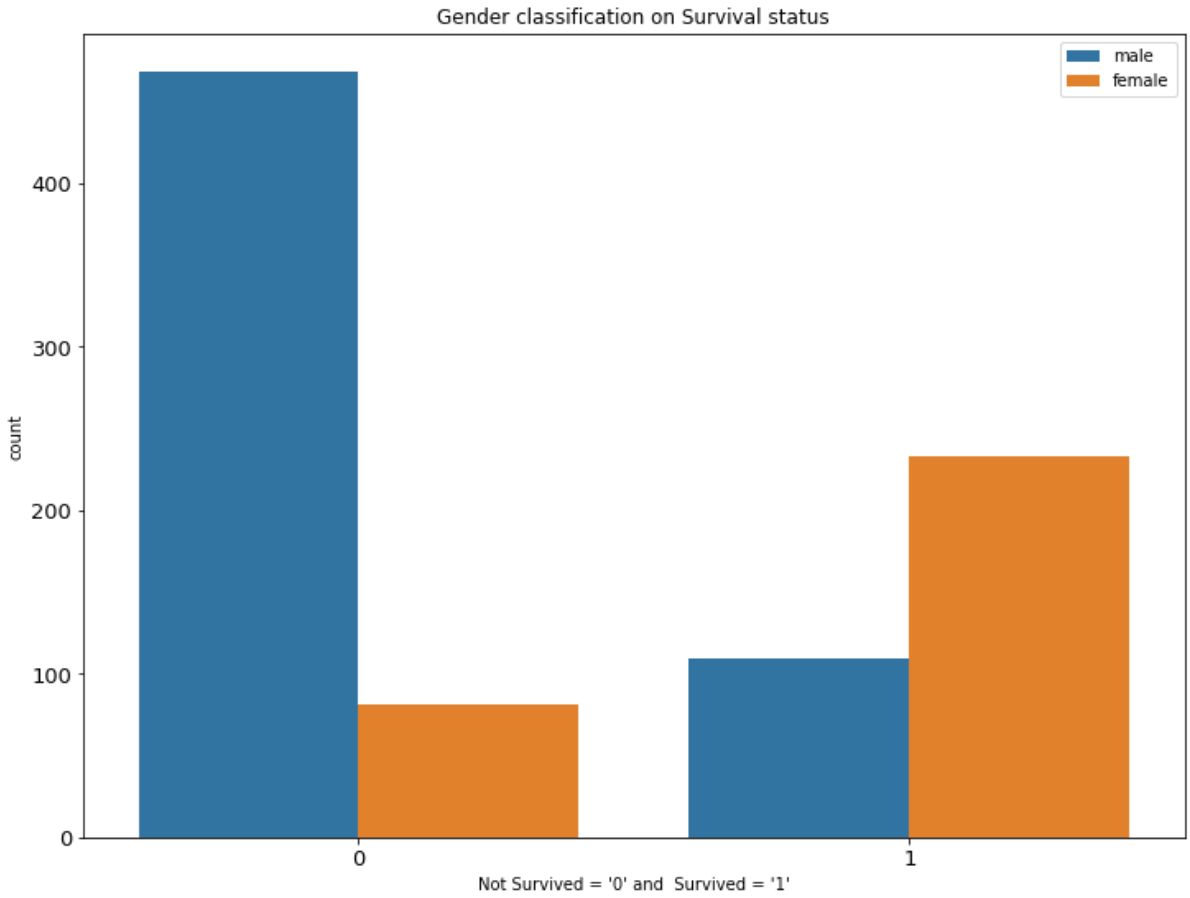
4. x=(1, 1, 0, 1, 0, 1), y=(1, 1, 1, 0, 0, 1) cosine, correlation, Jaccard

   cosine: cos(x,y) = ¾ = 0.75
   correlation: correlation: = corr(x,y) = 0.25
   Jaccard: 0.6

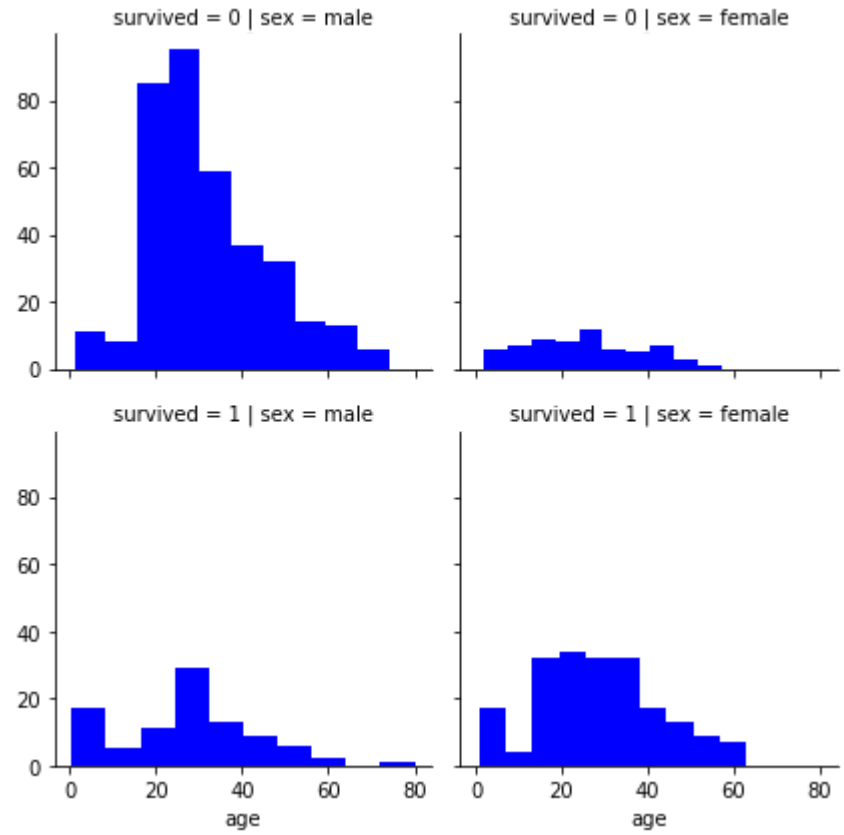
5. x=(2, −1, 0, 2, 0, −3), y=(−1, 1, −1, 0, 0, −1) cosine, correlation
   cosine: cos(x,y) = 0
   correlation: corr(x,y) = 0
   xbar = 0 and ybar = -0.66.

In [1]:
```
#Problem # 2.1
import seaborn as sb
import numpy as np
import matplotlib.pyplot as plt
```

In [30]:
```
#data
titanic_dataset = sb.load_dataset('titanic')
plt.figure(figsize=(12,9))
sb.countplot(x='survived', hue='sex', data=titanic_dataset)
plt.xticks(size=13)
plt.yticks(size=13)
plt.xlabel("Not Survived = '0' and  Survived = '1'")
plt.title("Gender classification on Survival status")
plt.legend(loc='best')
plt.show()
```



In [32]:
```
grid = sb.FacetGrid(titanic_dataset,col='sex',row='survived')
grid.map(plt.hist,'age',color='blue')
plt.show()
```



From the above visualization we can say survival condition that women and children servived more than men. Men under the age of 10 had better survival chance than the older men. Women between the age range 20-50 have survival better tan men, in that age-range.

In [ ]:
```
#Problem # 2.2
```

In [5]:
```
import seaborn as sb
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
```

In [7]:
```
#data
dataset = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data",
                      delim_whitespace = True, header=None,
                      names = ['mpg','cylinders','displacement', 'horsepower', 'weight','acceleartion','model', 'origin', 'car_name'])
```

In [8]:
```
dataset['horsepower'] = dataset['horsepower'].replace('?',np.nan)

dataset_mean = dataset.copy()
dataset_median = dataset.copy()
dataset_mode = dataset.copy()
```

In [25]:
```
## Imputer##
#mean
imr = SimpleImputer(missing_values=np.nan, strategy='mean')
imr = imr.fit(dataset[['horsepower']])
dataset_mean['horsepower'] = imr.transform(dataset_mean[['horsepower']]).ravel
()

#median
imr = SimpleImputer(missing_values=np.nan, strategy='median')
imr = imr.fit(dataset[['horsepower']])
dataset_median['horsepower'] = imr.transform(dataset_median[['horsepower']]).r
avel()

#mode
imr = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imr = imr.fit(dataset[['horsepower']])
dataset_mode['horsepower'] = imr.transform(dataset_mode[['horsepower']]).ravel
()
dataset_mode['horsepower'] = dataset_mode['horsepower'].astype(float)
```

In [26]:
```
print('*****MEAN*****')
print(dataset_mean.var())

print('*****MEDAIAN*****')
print(dataset_median.var())

print('*****MODE*****')
print(dataset_mode.var())
```

```
*****MEAN*****
mpg                 61.089611
cylinders            2.893415
displacement     10872.199152
horsepower        1459.177916
weight          717140.990526
acceleartion         7.604848
model               13.672443
origin               0.643292
dtype: float64
*****MEDAIAN*****
mpg                 61.089611
cylinders            2.893415
displacement     10872.199152
horsepower        1460.969052
weight          717140.990526
acceleartion         7.604848
model               13.672443
origin               0.643292
dtype: float64
*****MODE*****
mpg                 61.089611
cylinders            2.893415
displacement     10872.199152
horsepower        1490.036125
weight          717140.990526
acceleartion         7.604848
model               13.672443
origin               0.643292
dtype: float64
```

In [27]:
```
dataset_mean['horsepower'].var()
```

Out[27]: 1459.1779160026776

In [28]:
```
dataset_median['horsepower'].var()
```

Out[28]: 1460.96905180816

In [29]:
```
dataset_mode['horsepower'].var()
```

Out[29]: 1490.0361252104324

Here we can see when we replaced the missing values with mean. The whole set of data is centralized and moves towards the mean. The variance is more then replace with median and mode. That means replacing missing values with MEAN is the optimum way to minimize variance or distance from center.

Is there a different method of imputing values that would match the distribution more accurately? I believe we can use build regression model to predict the missing value in the horsepower and fill the missing values to predicted values. In this case we may get better distribution and variance in horsepower.

In [ ]:
```
## PROBLEM #3
```

In [10]:
```
import seaborn as sb
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
# from sklearn.preprocessing import Imputer
from sklearn.decomposition import PCA
from sklearn import datasets
from sklearn import decomposition
```

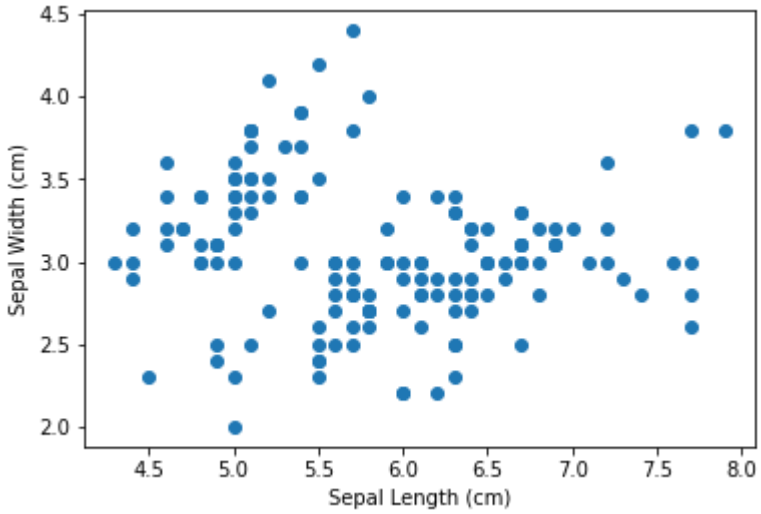
In [70]:
```
#dataset = datasets.load_iris() #importing iris datasets
#dataset_df = pd.DataFrame(dataset.data, columns=dataset.feature_names)
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.dat
a"
# load dataset into Pandas DataFrame
dataset_df = pd.read_csv(url, names=['sepal length','sepal width','petal lengt
h','petal width','target'])
```

In [71]:
```
dataset_df.head(3)
```

Out[71]:

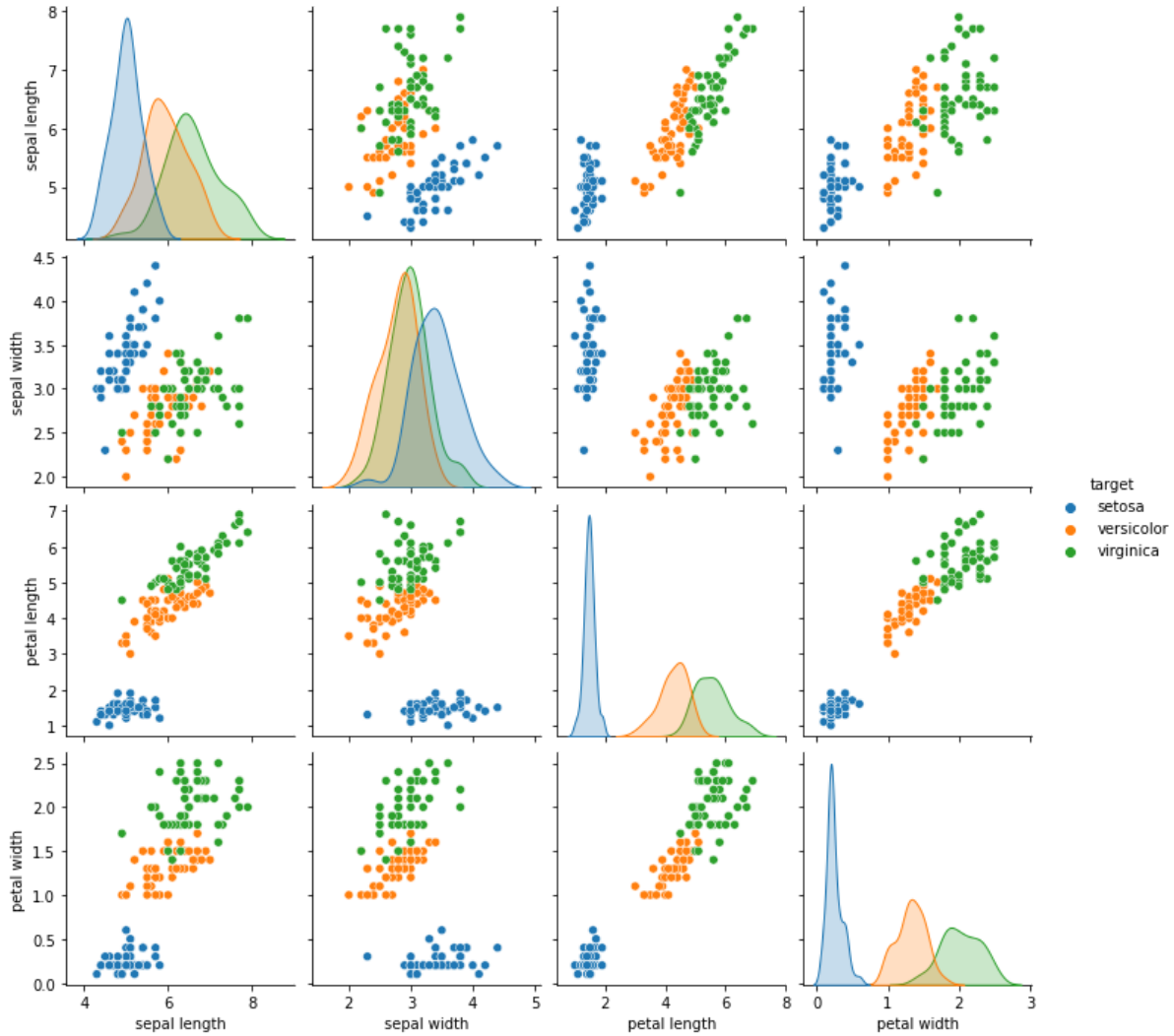
	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

In [73]:
```
plt.scatter(dataset_df['sepal length'], dataset_df['sepal width']);
plt.xlabel('Sepal Length (cm)');
plt.ylabel('Sepal Width (cm)');
```



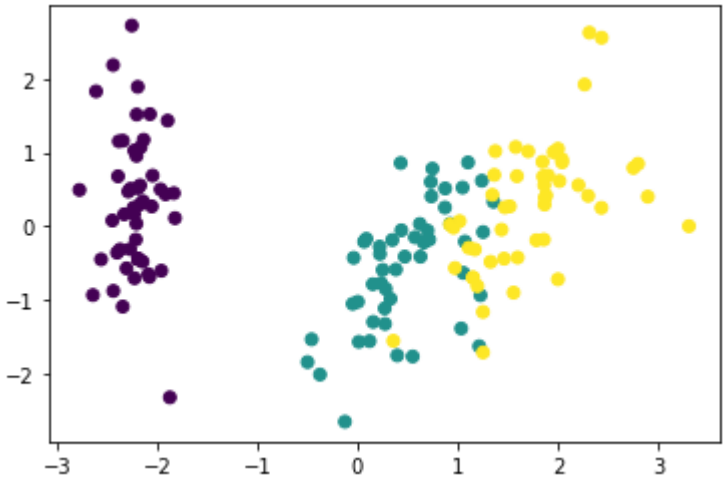
In [75]:
```
dataset_df['target'] = dataset.target_names[dataset.target]
sb.pairplot(dataset_df, hue='target')
```

Out[75]: <seaborn.axisgrid.PairGrid at 0x7f1c71e323c8>



In [76]:
```
from sklearn.preprocessing import StandardScaler
# Separating out the features
X = dataset_df.iloc[:, 0:4]
# Separating out the target
Y = dataset_df.iloc[:,4]
# Standardizing the features
X_std = StandardScaler().fit_transform(X)
```

In [79]:
```
from sklearn import decomposition
pca = decomposition.PCA(n_components=4)##finding 4 PCA

X = pca.fit_transform(X_std)
plt.scatter(X[:,0], X[:,1], c=dataset.target);
```



In [80]:
```
print("Variance of the Principal components",pca.explained_variance_ratio_)
```

```
Variance of the Principal components [0.72770452 0.23030523 0.03683832 0.0051
5193]
```

In [81]:
```
print("Variance of Each Original feature are:\n", dataset_df.var())
```

```
Variance of Each Original feature are:
 sepal length    0.685694
sepal width     0.188004
petal length    3.113179
petal width     0.582414
dtype: float64
```

In [84]:
```
#sepal length','sepal width','petal length','petal width','target'
variance1 = dataset_df.loc[:,"sepal length"].var()
variance2 = dataset_df.loc[:,"sepal width"].var()
variance3 = dataset_df.loc[:,"petal length"].var()
variance4 = dataset_df.loc[:,"petal width"].var()

total_variance = variance1 +variance2 + variance3 + variance4
print("Total Variance of original features is",total_variance)
```

```
Total Variance of original features is 4.5692912751677826
```

In [85]:
```
percentage_variance1 = (variance1 / total_variance) * 100
percentage_variance2 = (variance2 / total_variance) * 100
percentage_variance3 = (variance3 / total_variance) * 100
percentage_variance4 = (variance4 / total_variance) * 100

print ("Percentage of Variance explained by each of the original features are"
,"\n"
      , percentage_variance1 , percentage_variance2, percentage_variance3,  pe
rcentage_variance4)
```

```
Percentage of Variance explained by each of the original features are
 15.006561652804068 4.1145117595667875 68.13265407839864 12.746272509230492
```

The percentge of variance covered by **PCI (72.77%) and PC2(23.03%)** is around 95%. THe PC1 and PC2 explains most of the variance, while the third and fourth components can be dropped without losing much information.

When calculating the percentage of variance of each of the 4 features of dataset Percentage of Variance of sepal length = 15% Percentage of Variance of sepal width = 4.11% Percentage of Variance of **petal length**= 68.13% Percentage of Variance of petal width = 12.74%

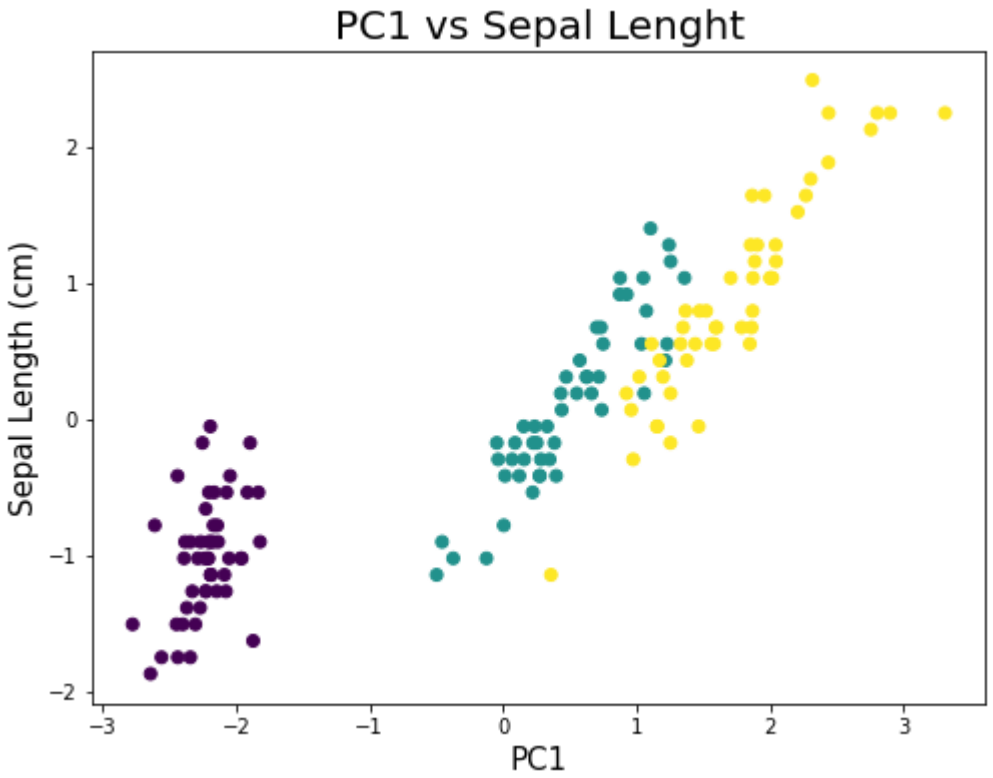
We find the number of dimensions where variance is spread is 3 in the case of feature variance. And using PCA we can clearly conclude that the number of dimensions have been reduced to 2 covering 95% of variance

In [86]:
```
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

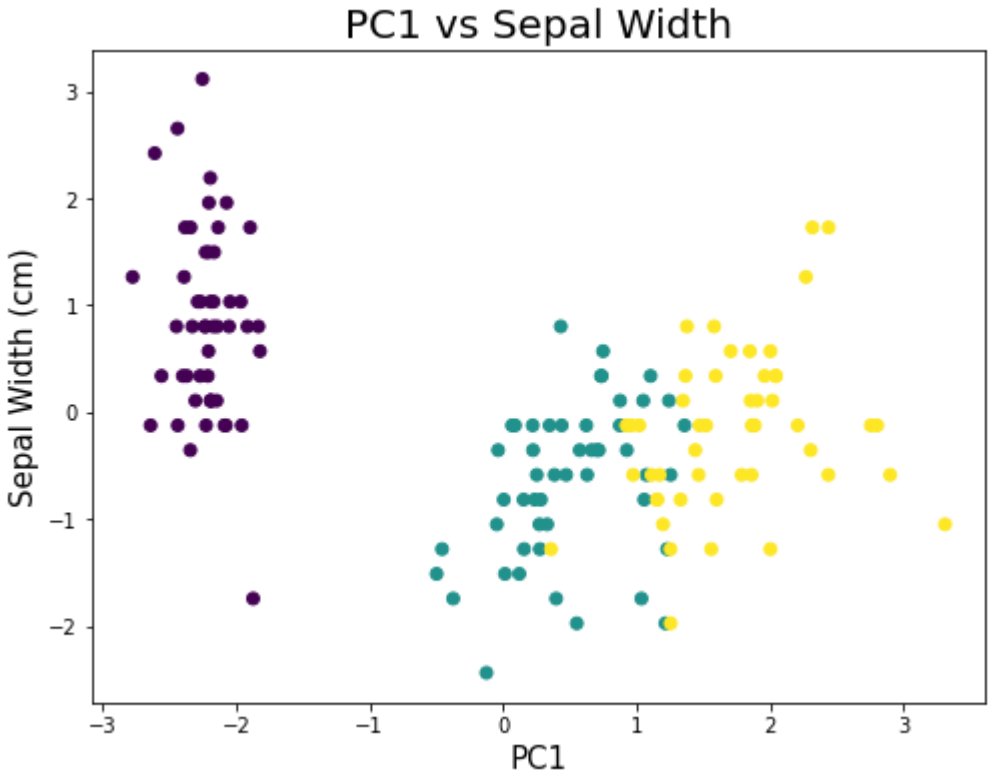
```
Covariance matrix
[[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
Eigenvectors
[[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

Eigenvalues
[2.93035378 0.92740362 0.14834223 0.02074601]
```

In [ ]:
```
## Problem 4
```

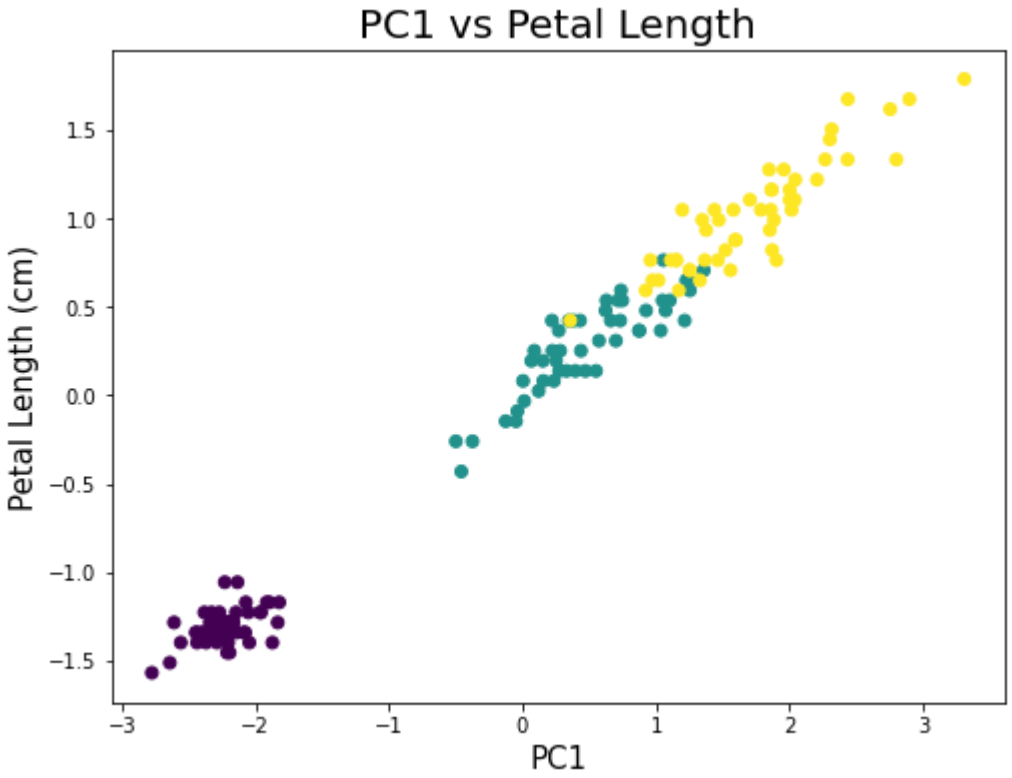
In [157]:
```
#PCA Plotting

plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X_std[:,0], c=dataset.target);
plt.xlabel('PC1',fontsize = 15);
plt.ylabel('Sepal Length (cm)',fontsize = 15);
plt.title("PC1 vs Sepal Lenght", fontsize = 20)
plt.show()
```



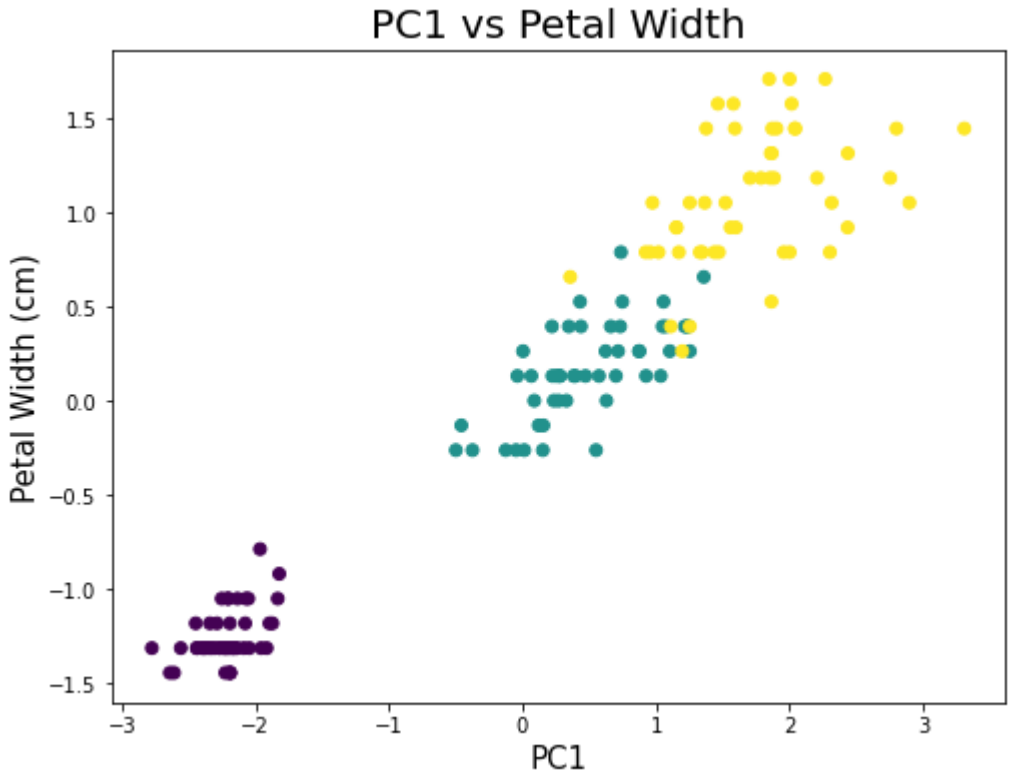
In [158]:
```
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X_std[:,1], c=dataset.target);
plt.xlabel('PC1',fontsize = 15);
plt.ylabel('Sepal Width (cm)',fontsize = 15);
plt.title("PC1 vs Sepal Width", fontsize = 20)
plt.show()
```



In [159]:
```
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X_std[:,2], c=dataset.target);
plt.xlabel('PC1',fontsize = 15);
plt.ylabel('Petal Length (cm)',fontsize = 15);
plt.title("PC1 vs Petal Length", fontsize = 20)
plt.show()
```



In [129]:
```
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X_std[:,3], c=dataset.target);
plt.xlabel('PC1',fontsize = 15);
plt.ylabel('Petal Width (cm)',fontsize = 15);
plt.title("PC1 vs Petal Width", fontsize = 20)
plt.show()
```



In [135]:
```
#Cosine distances and Correlation coefficient:
```

In [132]:
```
from scipy.spatial import distance
PC_sepallegth = distance.cosine(X[:,0], X_std[:,0])
PC_sepalwidth = distance.cosine(X[:,0], X_std[:,1])
PC_petallength = distance.cosine(X[:,0], X_std[:,2])
PC_petalwidth = distance.cosine(X[:,0], X_std[:,3])
```

In [134]:
```
print("Cosine distances Sepal Legth",PC_sepallegth)
print("Cosine distances Sepal Width",PC_sepalwidth)
print("Cosine distances Petal Legth",PC_petallength)
print("Cosine distances Petal Width",PC_petalwidth)
```

```
Cosine distances Sepal Legth 0.10877552110664224
Cosine distances Sepal Width 1.4493129756580254
Cosine distances Petal Legth 0.008315578401502655
Cosine distances Petal Width 0.03500421252862451
```

In [141]:
```
print("Correlation coefficient of Sepal Legth:", np.corrcoef(X[:,0], X_std[:,0
])[0,1])
print("Correlation coefficient of Sepal Width:", np.corrcoef(X[:,0], X_std[:,1
])[0,1])
print("Correlation coefficient of Petal Legth:", np.corrcoef(X[:,0], X_std[:,2
])[0,1])
print("Correlation coefficient of Petal Width:", np.corrcoef(X[:,0], X_std[:,3
])[0,1])
```

```
Correlation coefficient of Sepal Legth: 0.8912244788933584
Correlation coefficient of Sepal Width: -0.4493129756580255
Correlation coefficient of Petal Legth: 0.9916844215984977
Correlation coefficient of Petal Width: 0.9649957874713762
```

PC1 is closer to **Petal length** and **Petal width** since it has less cosine distance and high correlation coefficient.

Yes, the results agree with the visual inspection.

In [142]:
```
## Problem 5
```

In [152]:
```
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)

#sepal length','sepal width','petal length','petal width','target'
variance1 = dataset_df.loc[:,"sepal length"].var()
variance2 = dataset_df.loc[:,"sepal width"].var()
variance3 = dataset_df.loc[:,"petal length"].var()
variance4 = dataset_df.loc[:,"petal width"].var()

total_variance = variance1 +variance2 + variance3 + variance4
print("\nTotal Variance of original features :",total_variance)


print("\nTotal Variance of the four eigenvectors :",(eig_vecs[0].var()+eig_vec
s[1].var()+eig_vecs[2].var()+eig_vecs[3].var()))
percentage_variance1 = (variance1 / total_variance) * 100
percentage_variance2 = (variance2 / total_variance) * 100
percentage_variance3 = (variance3 / total_variance) * 100
percentage_variance4 = (variance4 / total_variance) * 100

print ("\nPercentage of Variance explained by each of the original features :"
,"\n"
      , percentage_variance1 , percentage_variance2, percentage_variance3,  pe
rcentage_variance4)
```

```
Covariance matrix
[[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
Eigenvectors
[[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

Eigenvalues
[2.93035378 0.92740362 0.14834223 0.02074601]

Total Variance of original features : 4.5692912751677826

Total Variance of the four eigenvectors : 0.7499999999999998

Percentage of Variance explained by each of the original features :
 15.006561652804068 4.1145117595667875 68.13265407839864 12.746272509230492
```

Total Variance of the four eigenvectors: **0.74**

Total Variance of original features : **4.56**

In [153]:
```
print("Variance of the Principal components",pca.explained_variance_ratio_)
```

```
Variance of the Principal components [0.72770452 0.23030523 0.03683832 0.0051
5193]
```

In [156]:
```
print("Cumulative Variance of the Principal components in % :",np.cumsum(pca.e
xplained_variance_ratio_)*100)
```

```
Cumulative Variance of the Principal components in % : [ 72.77045209  95.8009
7536  99.48480732 100.        ]
```

Here we can say we are getting more than 95% after 2 principal components. We should select **2 PCA** in this problem statement and we are reducing dimension from 4 to 2 using PCA.