Chanadana-MiniProject-SLC

March 8, 2020

0.1 K-Nearest-Neighbors

KNN falls in the supervised learning family of algorithms. Informally, this means that we are given a labelled dataset consiting of training observations (x,y) and would like to capture the relationship between x and y. More formally, our goal is to learn a function h:X→Y so that given an unseen observation x, h(x) can confidently predict the corresponding output y.

In this module we will explore the inner workings of KNN, choosing the optimal K values and using KNN from scikit-learn.

0.2 Problem statement

0.2.1 Dataset

The data set we'll be using is the Iris Flower Dataset which was first introduced in 1936 by the famous statistician Ronald Fisher and consists of 50 observations from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals.

Source: https://archive.ics.uci.edu/ml/datasets/Iris

Train the KNN algorithm to be able to distinguish the species from one another given the measurements of the 4 features.

0.3 Question 1

Read the iris.csv file

[156]:
```
#Data setup
import pandas as pd

df = pd.read_csv('iris.csv', skiprows=0)
df.sample(10)
```

[156]:
```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
102  103            7.1           3.0            5.9           2.1
38    39            4.4           3.0            1.3           0.2
93    94            5.0           2.3            3.3           1.0
80    81            5.5           2.4            3.8           1.1
14    15            5.8           4.0            1.2           0.2
30    31            4.8           3.1            1.6           0.2
113  114            5.7           2.5            5.0           2.0
```

```
139  140          6.9          3.1          5.4          2.1
65    66          6.7          3.1          4.4          1.4
138  139          6.0          3.0          4.8          1.8

             Species
102   Iris-virginica
38       Iris-setosa
93   Iris-versicolor
80   Iris-versicolor
14       Iris-setosa
30       Iris-setosa
113   Iris-virginica
139   Iris-virginica
65   Iris-versicolor
138   Iris-virginica
```

0.4 Data Pre-processing

0.5 Question 2 - Estimating missing values

Its not good to remove the records having missing values all the time. We may end up loosing some data points. So, we will have to see how to replace those missing values with some estimated values (median)

[109]:
```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values='NaN', strategy='median', axis=0)
imputer = imputer.fit(df.iloc[:,:-1])
imputed_data = imputer.transform(df.iloc[:,:-1].values)
df.iloc[:,:-1] = imputed_data

iris = df
```

```
/home/edwin/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:58: DeprecationWarning: Class Imputer is
deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22.
Import impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

0.6 Question 3 - Dealing with categorical data

Change all the classes to numericals (0to2).

[110]:
```
iris.iloc[:,5].unique()
```

[110]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

[111]:
```
iris.head()
```

[111]:
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0  1.0            5.1           3.5            1.4           0.2  Iris-setosa
1  2.0            4.9           3.0            1.4           0.2  Iris-setosa
2  3.0            4.7           3.2            1.3           0.2  Iris-setosa
3  4.0            4.6           3.1            1.5           0.2  Iris-setosa
4  5.0            5.0           3.6            1.4           0.2  Iris-setosa
```

[112]:
```
from sklearn.preprocessing import LabelEncoder
class_label_encoder = LabelEncoder()

iris.iloc[:,-1] = class_label_encoder.fit_transform(iris.iloc[:,-1])
```

[113]:
```
iris.head()
```

[113]:
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0  1.0            5.1           3.5            1.4           0.2        0
1  2.0            4.9           3.0            1.4           0.2        0
2  3.0            4.7           3.2            1.3           0.2        0
3  4.0            4.6           3.1            1.5           0.2        0
4  5.0            5.0           3.6            1.4           0.2        0
```

0.7 Question 4

Observe the association of each independent variable with target variable and drop variables from feature set having correlation in range -0.1 to 0.1 with target variable.

[12]:
```
iris.corr()
```

[12]:
```
                     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  \
Id             1.000000       0.702734     -0.392693       0.872346
SepalLengthCm  0.702734       1.000000     -0.109369       0.871120
SepalWidthCm  -0.392693      -0.109369      1.000000      -0.420713
PetalLengthCm  0.872346       0.871120     -0.420713       1.000000
PetalWidthCm   0.890676       0.815986     -0.356510       0.962043
Species        0.942753       0.775061     -0.417318       0.944477

               PetalWidthCm   Species
Id                 0.890676  0.942753
SepalLengthCm      0.815986  0.775061
SepalWidthCm      -0.356510 -0.417318
PetalLengthCm      0.962043  0.944477
PetalWidthCm       1.000000  0.952513
Species            0.952513  1.000000
```

0.8 Question 5

Observe the independent variables variance and drop such variables having no variance or almost zero variance(variance < 0.1). They will be having almost no influence on the classification.

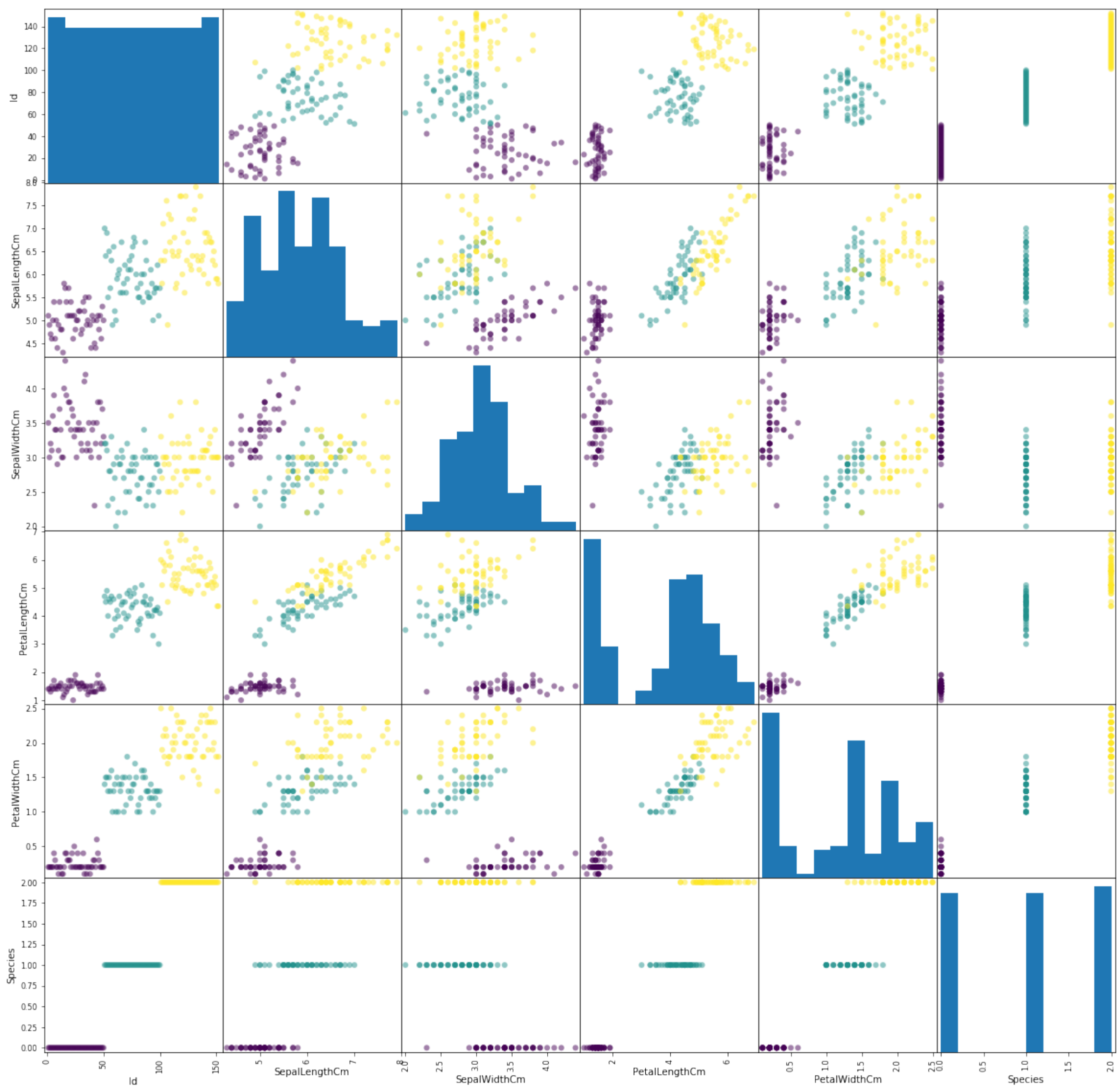
[13]:
```
iris.var()
```

[13]:
```
Id               1938.000000
SepalLengthCm       0.676645
SepalWidthCm        0.185552
PetalLengthCm       3.076516
PetalWidthCm        0.577141
Species             0.675322
dtype: float64
```

0.9 Question 6

Plot the scatter matrix for all the variables.

[15]:
```
splt = pd.plotting.scatter_matrix(iris, c=iris.iloc[:,-1], figsize=(20, 20),␣
↪marker='o')
```



0.10 Split the dataset into training and test sets

0.11 Question 7

Split the dataset into training and test sets with 80-20 ratio.

[19]:
```
import numpy as np
from sklearn.model_selection import train_test_split

# Transform data into features and target
X = np.array(iris.ix[:, 1:5])
y = np.array(iris['Species'])

# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=7)
```

[20]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(121, 4)
(121,)
```

[21]:
```
print(X_test.shape)
print(y_test.shape)
```

```
(31, 4)
(31,)
```

0.12 Question 8 - Model

Build the model and train and test on training and test sets respectively using ***scikit-learn****. Print the Accuracy of the model with different values of* ***k=3,5,9****.*

Hint: For accuracy you can check **accuracy_score()** in scikit-learn

[22]:
```
# loading library
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# instantiate learning model (k = 3)
knn = KNeighborsClassifier(n_neighbors = 3)

# fitting the model
knn.fit(X_train, y_train)

# predict the response
```

```
y_pred = knn.predict(X_test)

# evaluate accuracy
print(accuracy_score(y_test, y_pred))

# instantiate learning model (k = 5)
knn = KNeighborsClassifier(n_neighbors=5)

# fitting the model
knn.fit(X_train, y_train)

# predict the response
y_pred = knn.predict(X_test)

# evaluate accuracy
print(accuracy_score(y_test, y_pred))

# instantiate learning model (k = 9)
knn = KNeighborsClassifier(n_neighbors=9)

# fitting the model
knn.fit(X_train, y_train)

# predict the response
y_pred = knn.predict(X_test)

# evaluate accuracy
print(accuracy_score(y_test, y_pred))
```

```
0.9354838709677419
0.967741935483871
0.9032258064516129
```

0.13 Question 9 - Cross Validation

Run the KNN with no of neighbours to be 1,3,5..19 and *Find the **optimal number of neighbours** from the above list using the Mis classification error

Hint:

Misclassification error (MSE) = 1 - Test accuracy score. Calculated MSE for each model with neighbours = 1,3,5...19 and find the model with lowest MSE

[23]:
```
# creating odd list of K for KNN
myList = list(range(1,20))

# subsetting just the odd ones
neighbors = list(filter(lambda x: x % 2 != 0, myList))
```

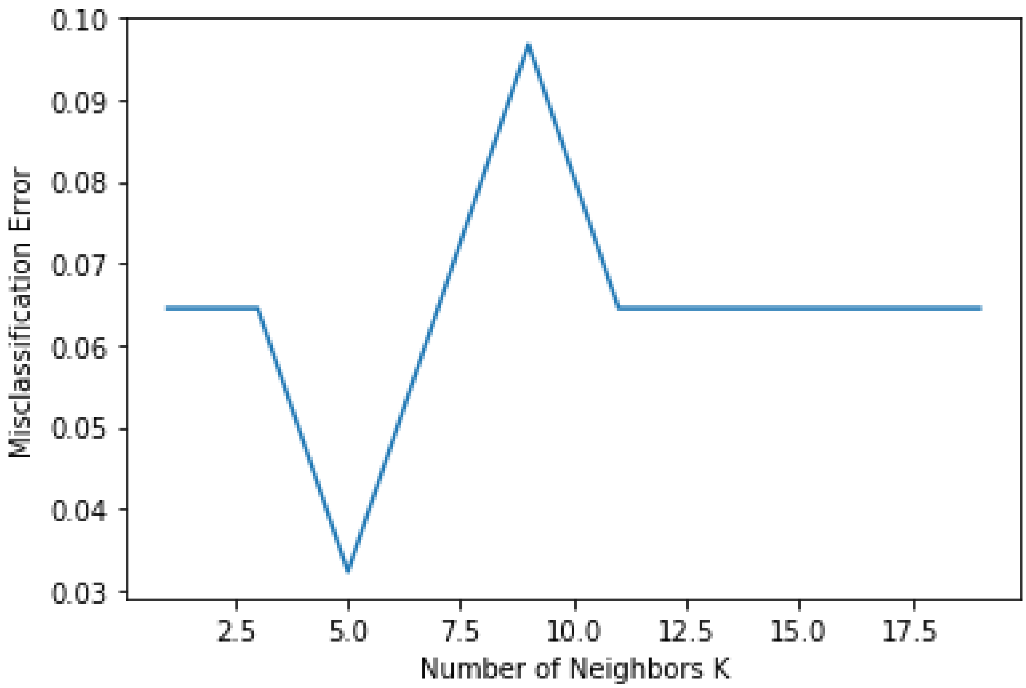
[24]:
```
# empty list that will hold accuracy scores
ac_scores = []

# perform accuracy metrics for values from 1,3,5....19
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    # predict the response
    y_pred = knn.predict(X_test)
    # evaluate accuracy
    scores = accuracy_score(y_test, y_pred)
    ac_scores.append(scores)

# changing to misclassification error
MSE = [1 - x for x in ac_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d" % optimal_k)
```

```
The optimal number of neighbors is 5
```

0.14 Question 10

Plot misclassification error vs k (with k value on X-axis) using matplotlib.

[33]:
```
import matplotlib.pyplot as plt
# plot misclassification error vs k
plt.plot(neighbors, MSE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
```



1 Naive Bayes

[25]:
```
#Load all required library
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
```

1.0.1 Question 1

Import Iris.csv

[85]:
```
# Load using input file
iris=pd.read_csv("iris.csv")
iris.head(5)
```

[85]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa

```
4   5           5.0          3.6          1.4          0.2  Iris-setosa
```

[86]:
```
# Check dimension of data
iris.shape
```

[86]:
```
(152, 6)
```

[87]:
```
#Check shape of data
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152 entries, 0 to 151
Data columns (total 6 columns):
Id                 152 non-null int64
SepalLengthCm      151 non-null float64
SepalWidthCm       150 non-null float64
PetalLengthCm      150 non-null float64
PetalWidthCm       151 non-null float64
Species            152 non-null object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

[88]:
```
# check for missing values
```

[89]:
```
iris.isna().sum()
```

[89]:
```
Id                0
SepalLengthCm     1
SepalWidthCm      2
PetalLengthCm     2
PetalWidthCm      1
Species           0
dtype: int64
```

[90]:
```
iris = iris.dropna()
```

[91]:
```
iris.isna().sum()
```

[91]:
```
Id                0
SepalLengthCm     0
SepalWidthCm      0
PetalLengthCm     0
PetalWidthCm      0
Species           0
dtype: int64
```

1.0.2 Question 2

Slice data set for Independent variables and dependent variables

Please note 'Species' is my dependent variables, name it y and independent set data as X

[92]:
```
X=iris.iloc[:,:4].values
y=iris['Species'].values
```

[93]:
```
#Check the dataset
print(y)
print(X)
```

```
['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica']
```

```
[[  1.    5.1   3.5   1.4]
 [  2.    4.9   3.    1.4]
 [  3.    4.7   3.2   1.3]
 [  4.    4.6   3.1   1.5]
 [  5.    5.    3.6   1.4]
 [  6.    5.4   3.9   1.7]
 [  7.    4.6   3.4   1.4]
 [  8.    5.    3.4   1.5]
 [  9.    4.4   2.9   1.4]
 [ 10.    4.9   3.1   1.5]
 [ 11.    5.4   3.7   1.5]
 [ 12.    4.8   3.4   1.6]
 [ 13.    4.8   3.    1.4]
 [ 14.    4.3   3.    1.1]
 [ 15.    5.8   4.    1.2]
 [ 16.    5.7   4.4   1.5]
 [ 17.    5.4   3.9   1.3]
 [ 18.    5.1   3.5   1.4]
 [ 19.    5.7   3.8   1.7]
 [ 20.    5.1   3.8   1.5]
 [ 21.    5.4   3.4   1.7]
 [ 22.    5.1   3.7   1.5]
 [ 23.    4.6   3.6   1. ]
 [ 24.    5.1   3.3   1.7]
 [ 25.    4.8   3.4   1.9]
 [ 26.    5.    3.    1.6]
 [ 27.    5.    3.4   1.6]
 [ 28.    5.2   3.5   1.5]
 [ 29.    5.2   3.4   1.4]
 [ 30.    4.7   3.2   1.6]
 [ 31.    4.8   3.1   1.6]
 [ 32.    5.4   3.4   1.5]
 [ 33.    5.2   4.1   1.5]
 [ 34.    5.5   4.2   1.4]
 [ 35.    4.9   3.1   1.5]
 [ 36.    5.    3.2   1.2]
 [ 37.    5.5   3.5   1.3]
 [ 38.    4.9   3.1   1.5]
 [ 39.    4.4   3.    1.3]
 [ 40.    5.1   3.4   1.5]
 [ 41.    5.    3.5   1.3]
 [ 42.    4.5   2.3   1.3]
 [ 43.    4.4   3.2   1.3]
 [ 44.    5.    3.5   1.6]
 [ 45.    5.1   3.8   1.9]
 [ 46.    4.8   3.    1.4]
 [ 47.    5.1   3.8   1.6]
 [ 48.    4.6   3.2   1.4]
```

```
[ 49.    5.3   3.7   1.5]
[ 50.    5.    3.3   1.4]
[ 51.    7.    3.2   4.7]
[ 52.    6.4   3.2   4.5]
[ 53.    6.9   3.1   4.9]
[ 54.    5.5   2.3   4. ]
[ 55.    6.5   2.8   4.6]
[ 56.    5.7   2.8   4.5]
[ 57.    6.3   3.3   4.7]
[ 58.    4.9   2.4   3.3]
[ 59.    6.6   2.9   4.6]
[ 60.    5.2   2.7   3.9]
[ 61.    5.    2.    3.5]
[ 62.    5.9   3.    4.2]
[ 63.    6.    2.2   4. ]
[ 64.    6.1   2.9   4.7]
[ 65.    5.6   2.9   3.6]
[ 66.    6.7   3.1   4.4]
[ 67.    5.6   3.    4.5]
[ 68.    5.8   2.7   4.1]
[ 69.    6.2   2.2   4.5]
[ 70.    5.6   2.5   3.9]
[ 71.    5.9   3.2   4.8]
[ 72.    6.1   2.8   4. ]
[ 73.    6.3   2.5   4.9]
[ 74.    6.1   2.8   4.7]
[ 75.    6.4   2.9   4.3]
[ 76.    6.6   3.    4.4]
[ 77.    6.8   2.8   4.8]
[ 78.    6.7   3.    5. ]
[ 79.    6.    2.9   4.5]
[ 80.    5.7   2.6   3.5]
[ 81.    5.5   2.4   3.8]
[ 82.    5.5   2.4   3.7]
[ 83.    5.8   2.7   3.9]
[ 84.    6.    2.7   5.1]
[ 85.    5.4   3.    4.5]
[ 86.    6.    3.4   4.5]
[ 87.    6.7   3.1   4.7]
[ 88.    6.3   2.3   4.4]
[ 89.    5.6   3.    4.1]
[ 90.    5.5   2.5   4. ]
[ 91.    5.5   2.6   4.4]
[ 92.    6.1   3.    4.6]
[ 93.    5.8   2.6   4. ]
[ 94.    5.    2.3   3.3]
[ 95.    5.6   2.7   4.2]
[ 96.    5.7   3.    4.2]
```

```
[ 97.    5.7   2.9   4.2]
[ 98.    6.2   2.9   4.3]
[ 99.    5.1   2.5   3. ]
[100.    5.7   2.8   4.1]
[101.    6.3   3.3   6. ]
[102.    5.8   2.7   5.1]
[103.    7.1   3.    5.9]
[104.    6.3   2.9   5.6]
[105.    6.5   3.    5.8]
[106.    7.6   3.    6.6]
[107.    4.9   2.5   4.5]
[108.    7.3   2.9   6.3]
[109.    6.7   2.5   5.8]
[110.    7.2   3.6   6.1]
[111.    6.5   3.2   5.1]
[112.    6.4   2.7   5.3]
[113.    6.8   3.    5.5]
[114.    5.7   2.5   5. ]
[115.    5.8   2.8   5.1]
[116.    6.4   3.2   5.3]
[117.    6.5   3.    5.5]
[118.    7.7   3.8   6.7]
[119.    7.7   2.6   6.9]
[120.    6.    2.2   5. ]
[121.    6.9   3.2   5.7]
[122.    5.6   2.8   4.9]
[123.    7.7   2.8   6.7]
[124.    6.3   2.7   4.9]
[125.    6.7   3.3   5.7]
[126.    7.2   3.2   6. ]
[127.    6.2   2.8   4.8]
[128.    6.1   3.    4.9]
[129.    6.4   2.8   5.6]
[130.    7.2   3.    5.8]
[131.    7.4   2.8   6.1]
[132.    7.9   3.8   6.4]
[133.    6.4   2.8   5.6]
[134.    6.3   2.8   5.1]
[135.    6.1   2.6   5.6]
[136.    7.7   3.    6.1]
[137.    6.3   3.4   5.6]
[138.    6.4   3.1   5.5]
[139.    6.    3.    4.8]
[140.    6.9   3.1   5.4]
[141.    6.7   3.1   5.6]
[142.    6.9   3.1   5.1]
[143.    5.8   2.7   5.1]
[144.    6.8   3.2   5.9]
```

```
 [145.     6.7    3.3    5.7]
 [146.     6.7    3.     5.2]
 [147.     6.3    2.5    5. ]
 [148.     6.5    3.     5.2]
 [149.     6.2    3.4    5.4]
 [150.     5.9    3.     5.1]]
```

1.1 Question 3

Find the distribution of target variable (Class)

And, Plot the distribution of target variable using histogram

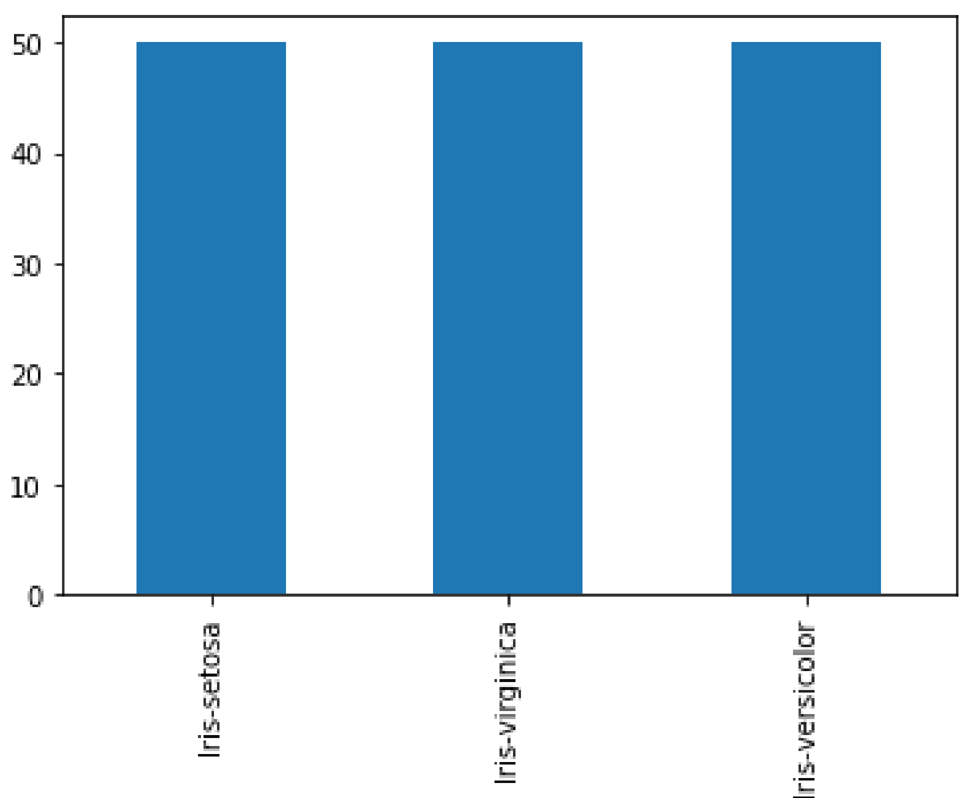
[94]:
```python
iris["Species"].value_counts()
```

[94]:
```
Iris-setosa        50
Iris-virginica     50
Iris-versicolor    50
Name: Species, dtype: int64
```

1.1.1 Plot the distribution of target variable using histogram

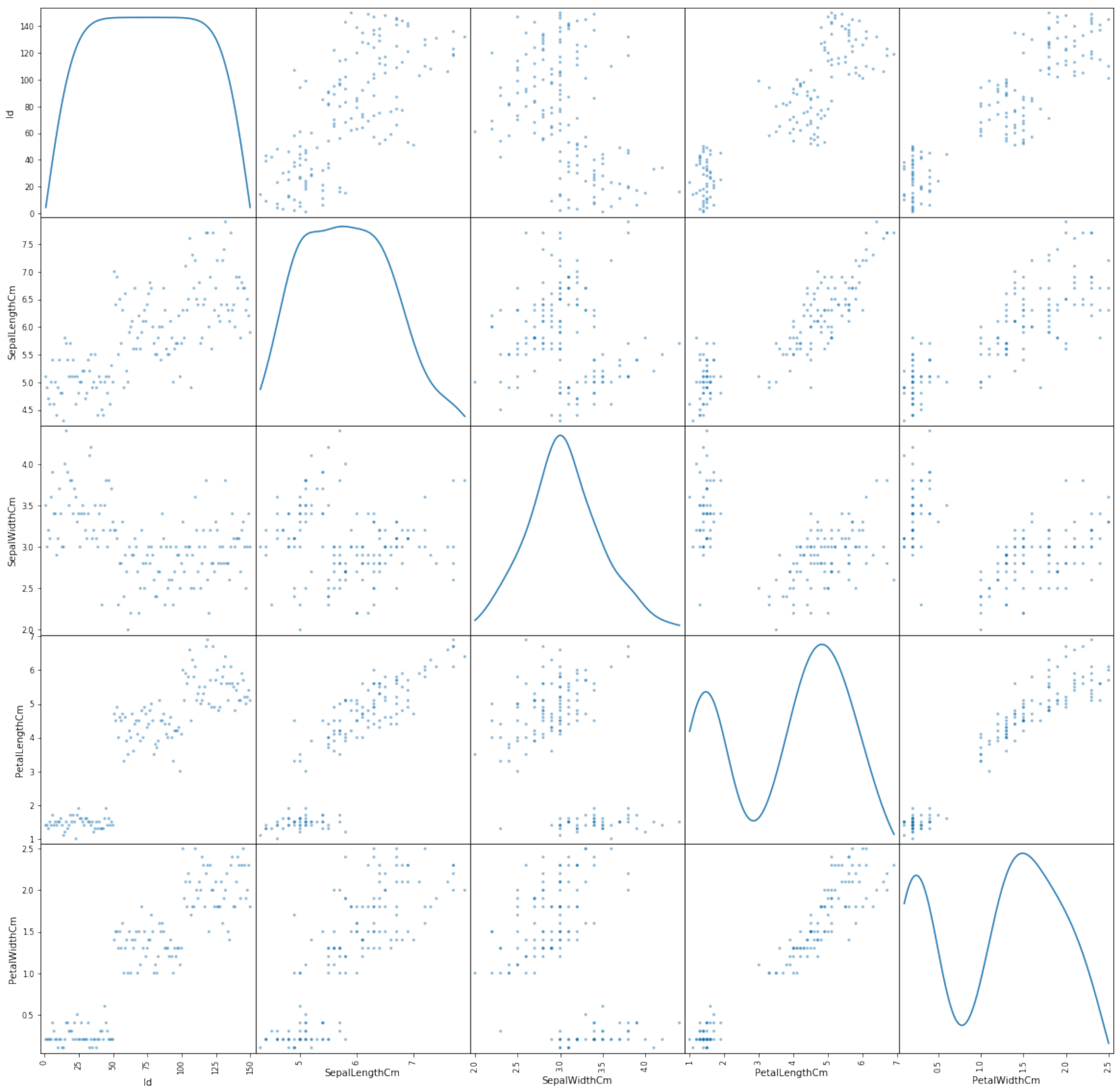
[95]:
```python
pd.value_counts(iris["Species"]).plot(kind="bar")
```

[95]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2d4366c438>
```



1.1.2 Plot Scatter Matrix to understand the distribution of variables and give insights from it(1 Marks)

[96]:
```
spd = pd.plotting.scatter_matrix(iris, figsize=(20,20), diagonal="kde")
```



1.1.3 Question 3

Find Correlation among all variables and give your insights

[97]:
```
corr = iris.corr()
corr
```

```
#Please note, it's Require to remove correlated features because they are voted␣
↪twice in the model and it can lead to over inflating importance.We will␣
↪ignore it here
```

[97]:

```
                     Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  \
Id             1.000000       0.716676     -0.397729       0.882747
SepalLengthCm  0.716676       1.000000     -0.109369       0.871754
SepalWidthCm  -0.397729      -0.109369      1.000000      -0.420516
PetalLengthCm  0.882747       0.871754     -0.420516       1.000000
PetalWidthCm   0.899759       0.817954     -0.356544       0.962757

               PetalWidthCm
Id                 0.899759
SepalLengthCm      0.817954
SepalWidthCm      -0.356544
PetalLengthCm      0.962757
PetalWidthCm       1.000000
```

[98]:

```
iris
```

[98]:

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
5      6            5.4           3.9            1.7           0.4
6      7            4.6           3.4            1.4           0.3
7      8            5.0           3.4            1.5           0.2
8      9            4.4           2.9            1.4           0.2
9     10            4.9           3.1            1.5           0.1
10    11            5.4           3.7            1.5           0.2
11    12            4.8           3.4            1.6           0.2
12    13            4.8           3.0            1.4           0.1
13    14            4.3           3.0            1.1           0.1
14    15            5.8           4.0            1.2           0.2
15    16            5.7           4.4            1.5           0.4
16    17            5.4           3.9            1.3           0.4
17    18            5.1           3.5            1.4           0.3
18    19            5.7           3.8            1.7           0.3
19    20            5.1           3.8            1.5           0.3
20    21            5.4           3.4            1.7           0.2
21    22            5.1           3.7            1.5           0.4
22    23            4.6           3.6            1.0           0.2
23    24            5.1           3.3            1.7           0.5
24    25            4.8           3.4            1.9           0.2
25    26            5.0           3.0            1.6           0.2
```

26	27	5.0	3.4	1.6	0.4
27	28	5.2	3.5	1.5	0.2
28	29	5.2	3.4	1.4	0.2
29	30	4.7	3.2	1.6	0.2
..	…	…	…	…	…
120	121	6.9	3.2	5.7	2.3
121	122	5.6	2.8	4.9	2.0
122	123	7.7	2.8	6.7	2.0
123	124	6.3	2.7	4.9	1.8
124	125	6.7	3.3	5.7	2.1
125	126	7.2	3.2	6.0	1.8
126	127	6.2	2.8	4.8	1.8
127	128	6.1	3.0	4.9	1.8
128	129	6.4	2.8	5.6	2.1
129	130	7.2	3.0	5.8	1.6
130	131	7.4	2.8	6.1	1.9
131	132	7.9	3.8	6.4	2.0
132	133	6.4	2.8	5.6	2.2
133	134	6.3	2.8	5.1	1.5
134	135	6.1	2.6	5.6	1.4
135	136	7.7	3.0	6.1	2.3
136	137	6.3	3.4	5.6	2.4
137	138	6.4	3.1	5.5	1.8
138	139	6.0	3.0	4.8	1.8
139	140	6.9	3.1	5.4	2.1
140	141	6.7	3.1	5.6	2.4
141	142	6.9	3.1	5.1	2.3
142	143	5.8	2.7	5.1	1.9
143	144	6.8	3.2	5.9	2.3
144	145	6.7	3.3	5.7	2.5
145	146	6.7	3.0	5.2	2.3
146	147	6.3	2.5	5.0	1.9
147	148	6.5	3.0	5.2	2.0
148	149	6.2	3.4	5.4	2.3
149	150	5.9	3.0	5.1	1.8

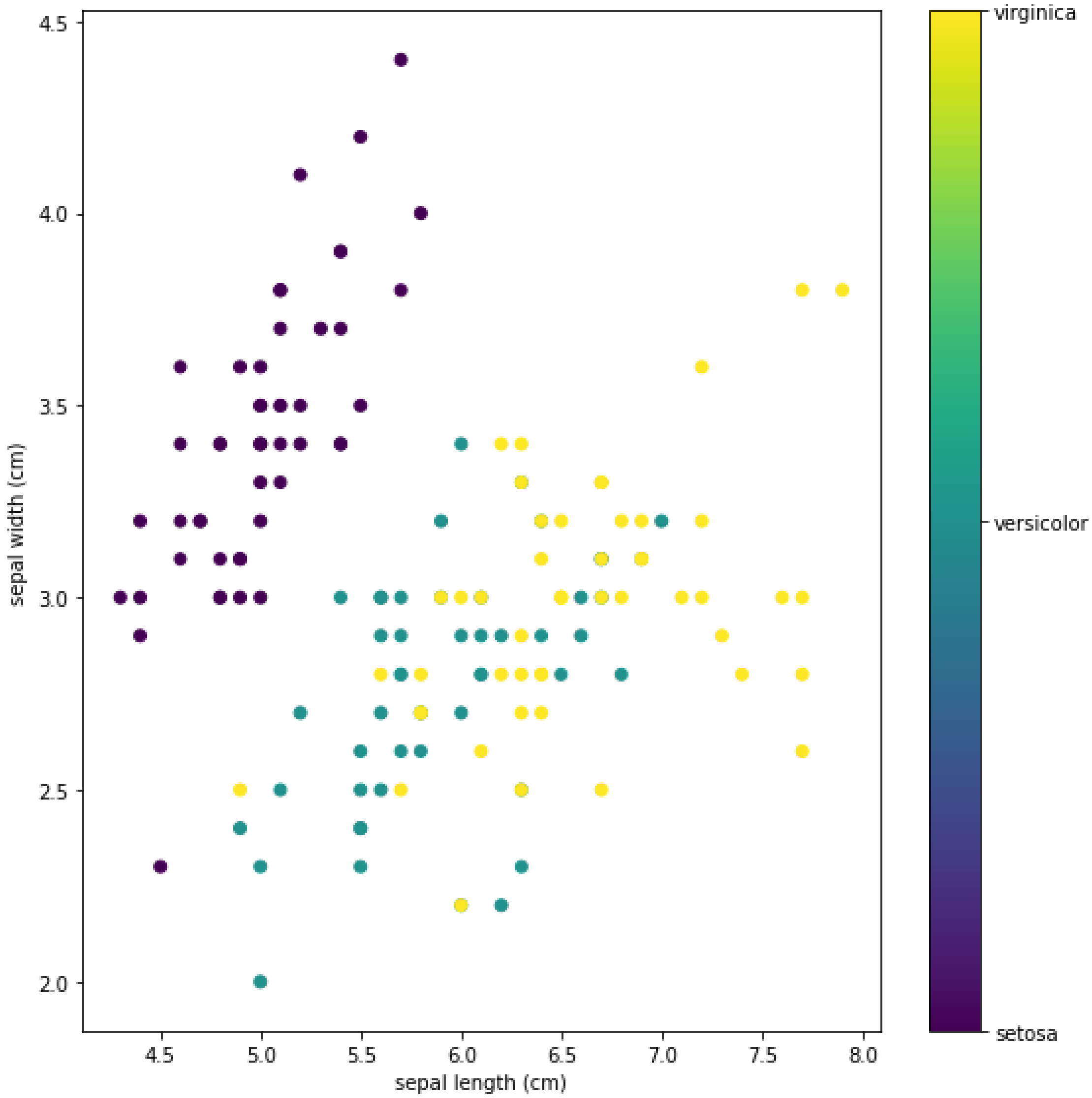
	Species
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa
5	Iris-setosa
6	Iris-setosa
7	Iris-setosa
8	Iris-setosa
9	Iris-setosa

```
10       Iris-setosa
11       Iris-setosa
12       Iris-setosa
13       Iris-setosa
14       Iris-setosa
15       Iris-setosa
16       Iris-setosa
17       Iris-setosa
18       Iris-setosa
19       Iris-setosa
20       Iris-setosa
21       Iris-setosa
22       Iris-setosa
23       Iris-setosa
24       Iris-setosa
25       Iris-setosa
26       Iris-setosa
27       Iris-setosa
28       Iris-setosa
29       Iris-setosa
..               …
120   Iris-virginica
121   Iris-virginica
122   Iris-virginica
123   Iris-virginica
124   Iris-virginica
125   Iris-virginica
126   Iris-virginica
127   Iris-virginica
128   Iris-virginica
129   Iris-virginica
130   Iris-virginica
131   Iris-virginica
132   Iris-virginica
133   Iris-virginica
134   Iris-virginica
135   Iris-virginica
136   Iris-virginica
137   Iris-virginica
138   Iris-virginica
139   Iris-virginica
140   Iris-virginica
141   Iris-virginica
142   Iris-virginica
143   Iris-virginica
144   Iris-virginica
145   Iris-virginica
```

```
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

[99]:

```
from sklearn.datasets import load_iris
iris = load_iris()

from matplotlib import pyplot as plt

# The indices of the features that we are plotting
x_index = 0
y_index = 1

# this formatter will label the colorbar with the correct target names
formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])

plt.figure(figsize=(8, 8))
plt.scatter(iris.data[:, x_index], iris.data[:, y_index], c=iris.target)
plt.colorbar(ticks=[0, 1, 2], format=formatter)
plt.xlabel(iris.feature_names[x_index])
plt.ylabel(iris.feature_names[y_index])

plt.tight_layout()
plt.show()
```



1.1.4 Question 4

Split data in Training and Validation in 80:20

[100]:
```
### SPLITTING INTO TRAINING AND TEST SETS
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
↪20,random_state=22)
```

1.1.5 Question 5

Do Feature Scaling

[101]:
```python
### NORMALIZTION / FEATURE SCALING
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

1.1.6 Question 6

Train and Fit NaiveBayes Model

[102]:
```python
### WE WILL FIT THE THE CLASSIFIER TO THE TRAINING SET
naiveClassifier=GaussianNB()
naiveClassifier.fit(X_train,y_train)
```

[102]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[103]:
```python
y_pred = naiveClassifier.predict(X_test)
```

[104]:
```python
#Keeping the actual and predicted value side by side
y_compare = np.vstack((y_test,y_pred)).T
#Actual->LEFT
#predicted->RIGHT
#Number of values to be print
y_compare[:20,:]
```

[104]:
```
array([['Iris-setosa', 'Iris-setosa'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-setosa', 'Iris-setosa'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-setosa', 'Iris-setosa'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-versicolor', 'Iris-versicolor'],
       ['Iris-virginica', 'Iris-virginica'],
       ['Iris-versicolor', 'Iris-versicolor']], dtype=object)
```

1.1.7 Question 7

Print Accuracy and Confusion Matrix and Conclude your findings

[157]:
```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
↪---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call␣
↪last)

    <ipython-input-157-4356c19c44e3> in <module>
      1 # Making the Confusion Matrix
      2 from sklearn.metrics import confusion_matrix
----> 3 cm = confusion_matrix(y_test, y_pred)
      4 print(cm)


    ~/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.
↪py in confusion_matrix(y_true, y_pred, labels, sample_weight)
    251
    252     """
--> 253     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    254     if y_type not in ("binary", "multiclass"):
    255         raise ValueError("%s is not supported" % y_type)


    ~/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.
↪py in _check_targets(y_true, y_pred)
     69     y_pred : array or indicator matrix
     70     """
---> 71     check_consistent_length(y_true, y_pred)
     72     type_true = type_of_target(y_true)
     73     type_pred = type_of_target(y_pred)


    ~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in␣
↪check_consistent_length(*arrays)
    233     if len(uniques) > 1:
    234         raise ValueError("Found input variables with inconsistent␣
↪numbers of"
--> 235                          " samples: %r" % [int(l) for l in lengths])
    236
    237
```

```
        ValueError: Found input variables with inconsistent numbers of samples:␣
 ↪[192, 30]
```

[106]:
```
#finding accuracy from the confusion matrix.
a = cm.shape
correctPrediction = 0
falsePrediction = 0

for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            correctPrediction +=cm[row,c]
        else:
            falsePrediction += cm[row,c]
print('Correct predictions: ', correctPrediction)
print('False predictions', falsePrediction)
print ('\n\nAccuracy of the Naive Bayes Clasification is: ', correctPrediction/
 ↪(cm.sum()))
```

```
Correct predictions:  30
False predictions 0


Accuracy of the Naive Bayes Clasification is:  1.0
```

[107]:
```
from sklearn import metrics
print(metrics.classification_report(y_pred, y_test))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         6
Iris-versicolor       1.00      1.00      1.00        10
 Iris-virginica       1.00      1.00      1.00        14

      micro avg       1.00      1.00      1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

2 SVM

[158]:
```
#Import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within

ve years.

Fields description follow:

preg = Number of times pregnant

plas = Plasma glucose concentration a 2 hours in an oral glucose tolerance test

pres = Diastolic blood pressure (mm Hg)

skin = Triceps skin fold thickness (mm)

test = 2-Hour serum insulin (mu U/ml)

mass = Body mass index (weight in kg/(height in m)^2)

pedi = Diabetes pedigree function

age = Age (years)

class = Class variable (1:tested positive for diabetes, 0: tested negative for diabetes)

2.0.1 Question 1

Read the input file 'Diabetes.csv' using Pandas and check it's column names(1 Marks)

[159]:
```
diabetes = pd.read_csv('pima-indians-diabetes.csv')
print(diabetes.columns)
```

```
Index(['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'],
dtype='object')
```

[160]:
```
# Eye ball the imported dataset
diabetes.head()
```

[160]:

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

2.0.2 Question 2

Check the dimensions of dataset

[161]:
```
print("dimension of diabetes data: {}".format(diabetes.shape))
#The diabetes dataset consists of 768 data points, with 9 features
```

```
dimension of diabetes data: (768, 9)
```

2.0.3 Question 3

Check distribution of dependent variable 'class' and plot it

[162]:
```
print(diabetes.groupby('class').size())
```

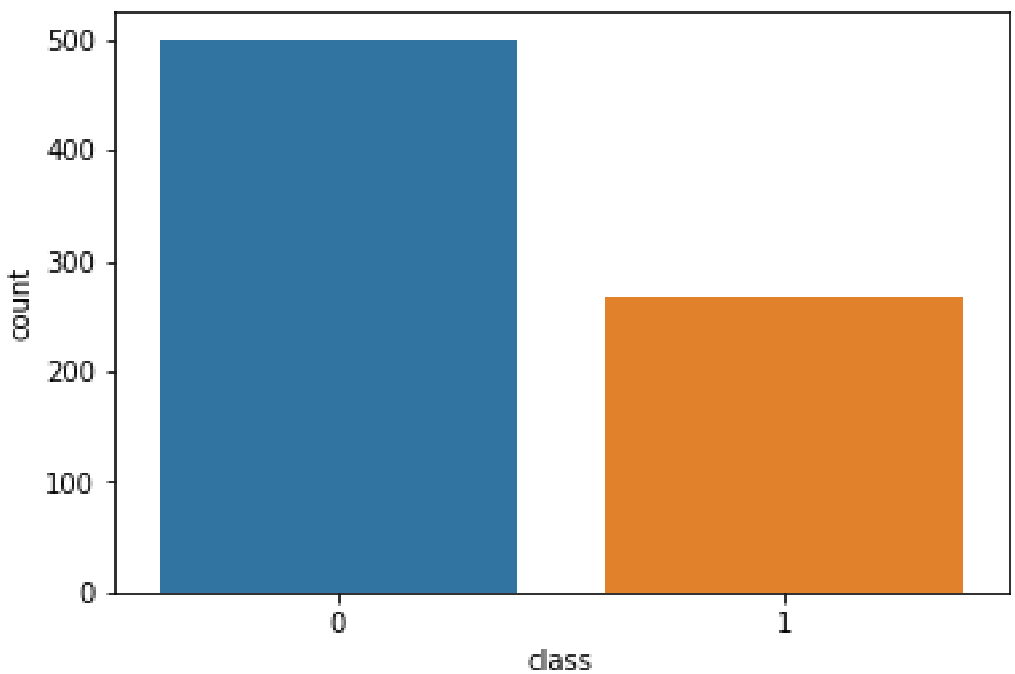
```
class
0    500
1    268
dtype: int64
```

2.0.4 Out of 768 data points, 500 are labeled as 0 and 268 as 1.

2.0.5 Class 0 means No diabetes, outcome 1 means diabetes

[163]:
```
import seaborn as sns

sns.countplot(diabetes['class'],label="Count")
```

[163]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d41287e10>



[164]:
```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
preg     768 non-null int64
```

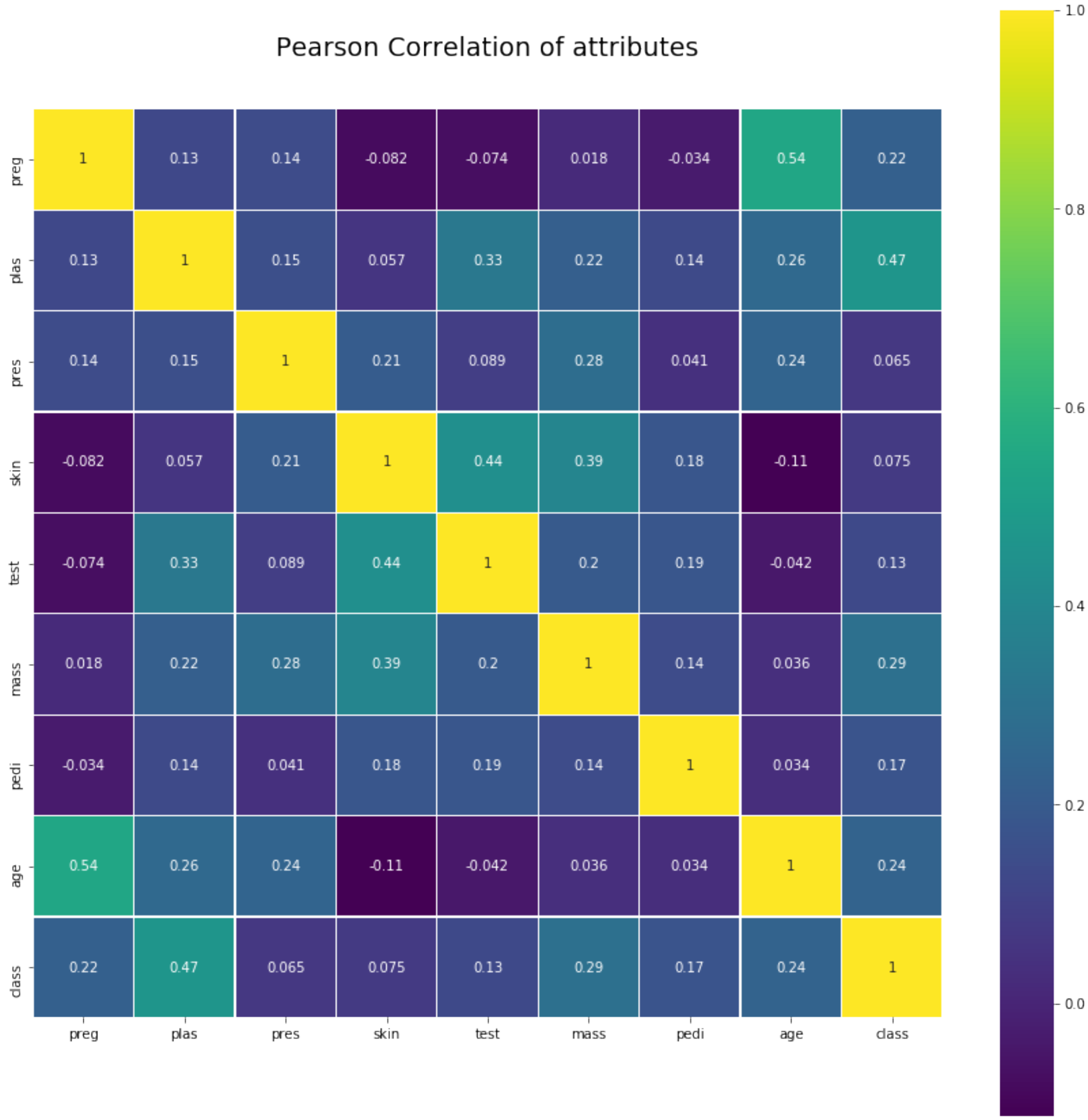
```
plas     768 non-null int64
pres     768 non-null int64
skin     768 non-null int64
test     768 non-null int64
mass     768 non-null float64
pedi     768 non-null float64
age      768 non-null int64
class    768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

2.0.6 Question 4

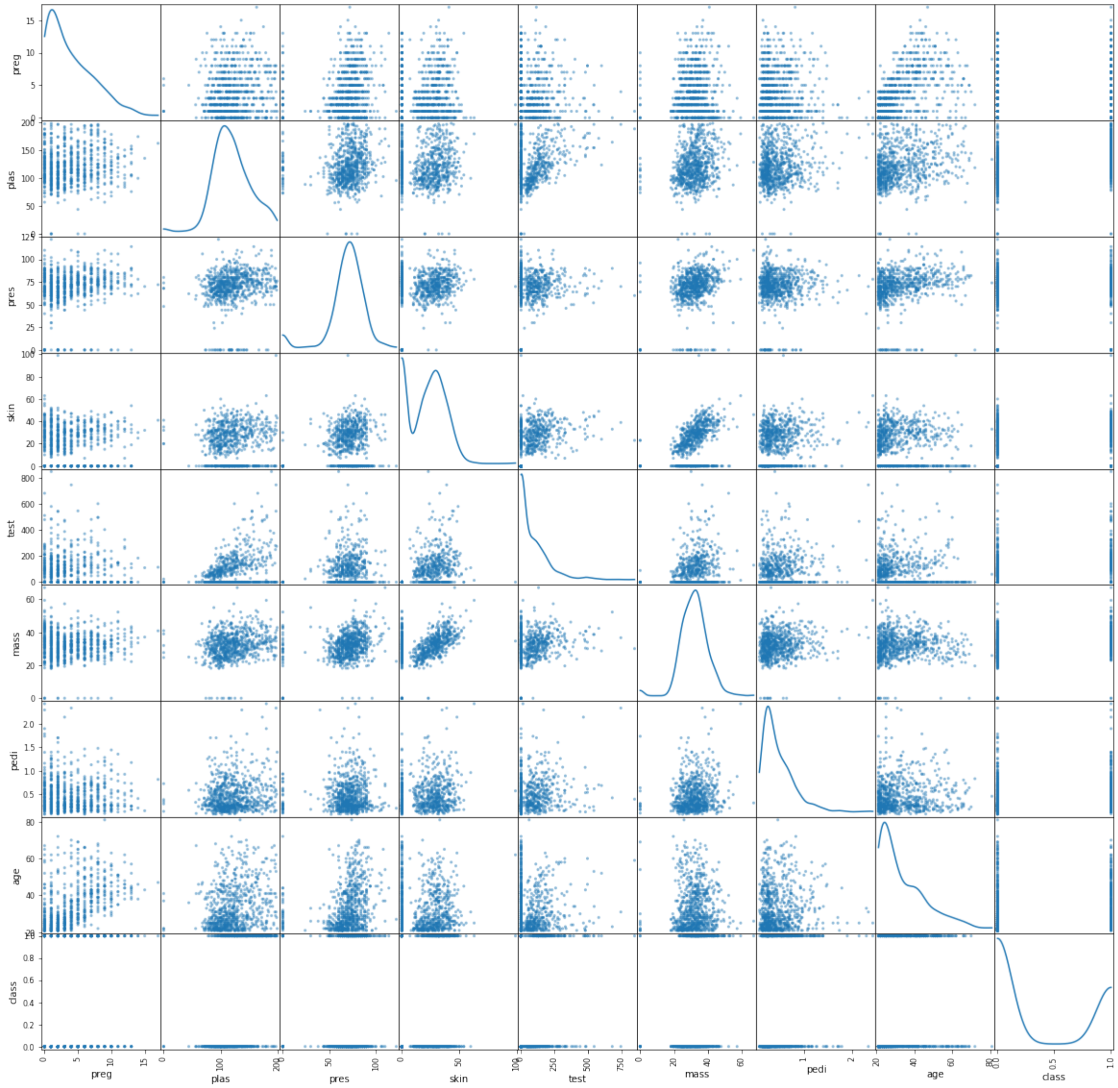
Do correlation analysis and bivariate viualization with Insights

[165]:
```
colormap = plt.cm.viridis # Color range to be used in heatmap
plt.figure(figsize=(15,15))
plt.title('Pearson Correlation of attributes', y=1.05, size=19)
sns.heatmap(diabetes.corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
#There is no strong correlation between any two variables.
#There is no strong correlation between any independent variable and class␣
 ↪variable.
```

[165]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2d386d70f0>



[166]:
```
spd = pd.plotting.scatter_matrix(diabetes, figsize=(20,20), diagonal="kde")
```



2.0.7 Question 5

Do train and test split with stratify sampling on Outcome variable to maintain the distribution of dependent variable

[167]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(diabetes.loc[:, diabetes.
 ↪columns != 'class'], diabetes['class'], stratify=diabetes['class'],␣
 ↪random_state=11)
```

[168]:
```
X_train.shape
```

[168]: (576, 8)

2.0.8 Question 6

Train Support Vector Machine Model

[169]:
```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

```
Accuracy on training set: 1.00
Accuracy on test set: 0.65
```

[170]:
```
#The model overfits substantially with a perfect score on the training set and␣
↪only 65% accuracy on the test set.

#SVM requires all the features to be on a similar scale. We will need to␣
↪rescale our data that all the features are approximately on the same scale␣
↪and than see the performance
```

2.0.9 Question 7

Scale the data points using MinMaxScaler

[171]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

```
/home/edwin/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:323: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
/home/edwin/anaconda3/lib/python3.7/site-
packages/sklearn/preprocessing/data.py:323: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

2.0.10 Question 8

Fit SVM Model on scaled data and give your observation

[172]:
```
svc = SVC()
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train_scaled,␣
↪y_train)))
```

```
print("Accuracy on test set: {:.2f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.76
Accuracy on test set: 0.79
```

2.0.11 Question 9

Try improving the model accuracy using C=1000

[173]:
```
svc = SVC(C=1000)
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.812
Accuracy on test set: 0.755
```