Boosting

68_Adnan Shaikh

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import pandas as pd
df=pd.read_csv('./apples_and_oranges.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Weight	Size	Class
0	69	4.39	orange
1	69	4.21	orange
2	65	4.09	orange
3	72	5.85	apple
4	67	4.70	orange

In [4]:
```
df.shape
```

Out[4]: (40, 3)

In [5]:
```
x=df.drop("Class",axis="columns")
y=df.Class
```

In [6]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

In [7]:
```
x_test.shape
```

Out[7]: (8, 2)

In [8]:
```
x_train.shape
```

Out[8]: (32, 2)

In [9]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [10]:
```
ada=AdaBoostClassifier(n_estimators=100,base_estimator=None,learning_rate=1,random_state=1)
ada.fit(x_train,y_train)
```

Out[10]: AdaBoostClassifier(learning_rate=1, n_estimators=100, random_state=1)

In [11]:
```
y_pred=ada.predict(x_test)
```

In [12]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[3 0]
 [0 5]]
```

In [13]:
```
accuracy=float(cm.diagonal().sum())/len(y_test)
```

In [14]:
```
print("Accuracy of Adaboost for Given Data set :",accuracy)
```

```
Accuracy of Adaboost for Given Data set : 1.0
```

In [15]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00         3
      orange       1.00      1.00      1.00         5

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8
```

Gradient Boosting

In [16]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [17]:
```
gb = GradientBoostingClassifier(n_estimators=100)

gb.fit(x_train,y_train)
```

Out[17]:
```
GradientBoostingClassifier()
```

In [18]:
```
y_pred = gb.predict(x_test)
```

In [19]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00         3
      orange       1.00      1.00      1.00         5

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8
```

Xtreme Gradient Boosting

In [20]:
```
from xgboost import XGBClassifier
```

In [21]:
```
xgb = XGBClassifier(n_estimators=200,reg_alpha=1)
```

In [22]:
```
xgb.fit(x_train,y_train)
```

```
[15:02:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:10
95: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logis
tic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore th
e old behavior.
```

Out[22]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=200, n_jobs=12, num_parallel_tree=1, random_state=0,
              reg_alpha=1, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [23]:
```
y_pred = xgb.predict(x_test)
```

In [24]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

       apple       0.60      1.00      0.75         3
      orange       1.00      0.60      0.75         5

    accuracy                           0.75         8
   macro avg       0.80      0.80      0.75         8
weighted avg       0.85      0.75      0.75         8
```

K Means

68_Adnan Shaikh

In [1]:
```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_excel("./income.xlsx")
```

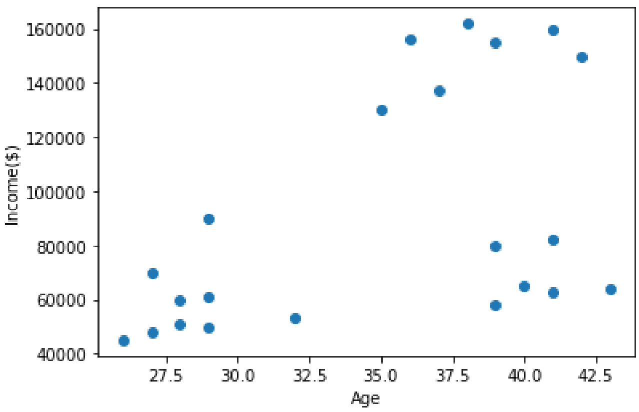
In [3]:
```
df.head()
```

Out[3]:

	Name	Age	Income($)	Unnamed: 3
0	Rob	27	70000	NaN
1	Michael	29	90000	NaN
2	Mohan	29	61000	NaN
3	Ismail	28	60000	NaN
4	Kory	42	150000	NaN

In [4]:
```
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
```

Out[4]: Text(0, 0.5, 'Income($)')



In [5]:
```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
```

Out[5]: array([2, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0])

In [6]:
```
df['cluster']=y_predicted
df
```

Out[6]:

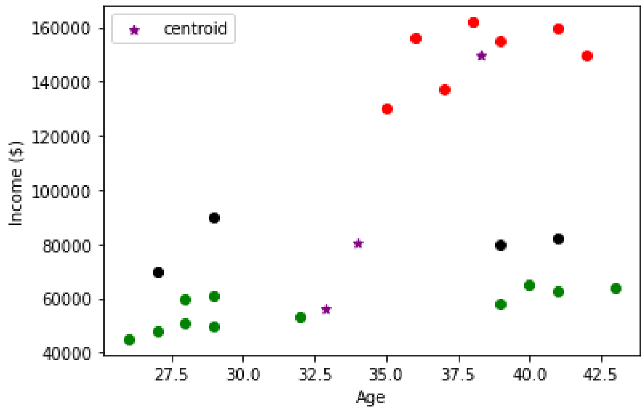
	Name	Age	Income($)	Unnamed: 3	cluster
0	Rob	27	70000	NaN	2
1	Michael	29	90000	NaN	2
2	Mohan	29	61000	NaN	0
3	Ismail	28	60000	NaN	0
4	Kory	42	150000	NaN	1
5	Gautam	39	155000	NaN	1
6	David	41	160000	NaN	1
7	Andrea	38	162000	NaN	1
8	Brad	36	156000	NaN	1
9	Angelina	35	130000	NaN	1
10	Donald	37	137000	NaN	1
11	Tom	26	45000	NaN	0
12	Arnold	27	48000	NaN	0
13	Jared	28	51000	NaN	0
14	Stark	29	49500	NaN	0
15	Ranbir	32	53000	NaN	0
16	Dipika	40	65000	NaN	0
17	Priyanka	41	63000	NaN	0
18	Nick	43	64000	NaN	0
19	Alia	39	80000	NaN	2
20	Sid	41	82000	NaN	2
21	Abdul	39	58000	NaN	0

In [7]:
```
km.cluster_centers_
```

Out[7]:
```
array([[3.29090909e+01, 5.61363636e+04],
       [3.82857143e+01, 1.50000000e+05],
       [3.40000000e+01, 8.05000000e+04]])
```

In [8]:
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='cen
troid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
```

Out[8]: <matplotlib.legend.Legend at 0x22d95fdda60>



Preprocessing using min max scaler

In [9]:
```
scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])
```

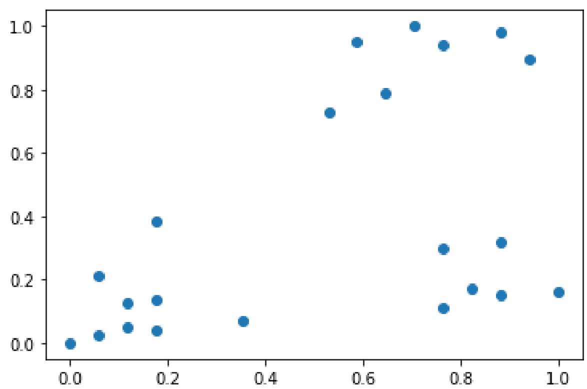
In [10]:
```
df.head()
```

Out[10]:

	Name	Age	Income($)	Unnamed: 3	cluster
0	Rob	0.058824	0.213675	NaN	2
1	Michael	0.176471	0.384615	NaN	2
2	Mohan	0.176471	0.136752	NaN	0
3	Ismail	0.117647	0.128205	NaN	0
4	Kory	0.941176	0.897436	NaN	1

In [11]:
```
plt.scatter(df.Age,df['Income($)'])
```

Out[11]: <matplotlib.collections.PathCollection at 0x22d96069340>



In [12]:
```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
```

Out[12]:
```
array([1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0])
```

In [13]:
```
df['cluster']=y_predicted
df.head()
```

Out[13]:

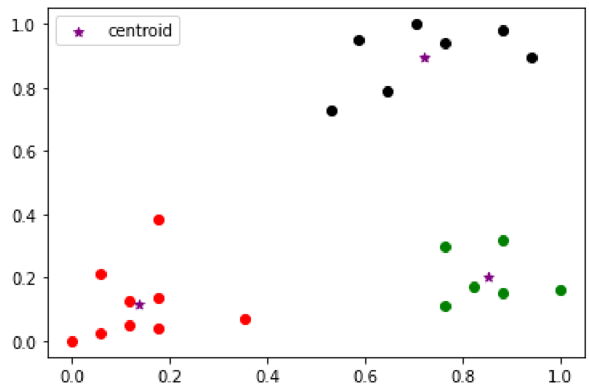
	Name	Age	Income($)	Unnamed: 3	cluster
0	Rob	0.058824	0.213675	NaN	1
1	Michael	0.176471	0.384615	NaN	1
2	Mohan	0.176471	0.136752	NaN	1
3	Ismail	0.117647	0.128205	NaN	1
4	Kory	0.941176	0.897436	NaN	2

In [14]:
```
km.cluster_centers_
```

Out[14]:
```
array([[0.85294118, 0.2022792 ],
       [0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ]])
```

In [15]:
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='cen
troid')
plt.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x22d962d5b20>
```

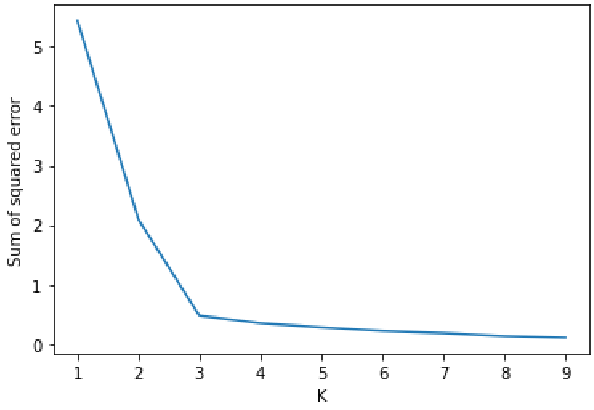


Elbow Plot

In [16]:
```
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)
```

In [17]:
```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Out[17]: [<matplotlib.lines.Line2D at 0x22d96368610>]



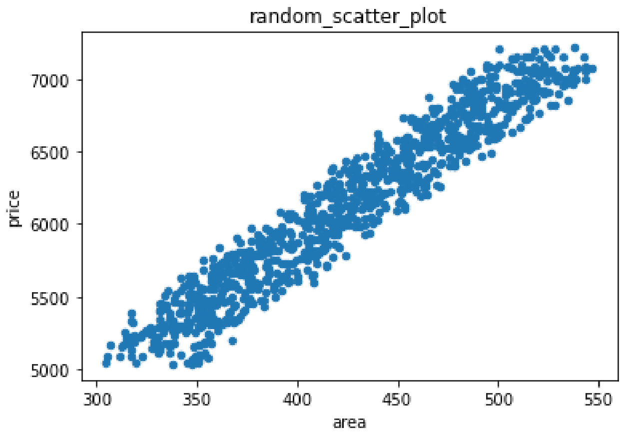
Linear Regression

68_Adnan Shaikh

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
obj = pd.DataFrame({"x":np.arange(300,500,0.2)+np.random.uniform(low=0,high=50,size=1000), "y":np.arange(5000,7000,2)+np.random.uniform(low=0,high=250,size=1000)})
```

In [3]:
```
fig,axis = plt.subplots()
obj.plot(x="x",y="y",ax=axis,kind="scatter",title="random_scatter_plot",xlabel="area",ylabel="price",layout=(10,9))
plt.show()
```



In [4]:
```
def minmax_normalize(data,new_max,new_min):
  mini = data.min()
  maxi = data.max()
  normalize_data = []
  for x in data:
    normalize_data.append((x-mini)/(mini-maxi)*(new_max-new_min)+new_min)
  return np.array(normalize_data)
```

In [5]:
```
class LinearModel:
  def fit(self,x,y):
    x_mean,y_mean = x.mean(), y.mean()
    x_norm, y_norm = x.apply(lambda k: k-x_mean), y.apply(lambda k: k-y_mean)
    sum_x = x_norm.sum()
    sum_y = y_norm.sum()
    xy_sum = (x_norm*y_norm).sum()
    sum_x_sq =  (x_norm**2).sum()
    intercept = xy_sum/sum_x_sq
    slope = y_mean-intercept*x_mean
    self.intercept, self.slope = intercept,slope
    return intercept,slope

  def predict(self,x):
    result = []
    for data in x:
      result.append(self.slope+self.intercept*data)
    return np.array(result)

  def average_error(self,x,y):

    predicted_y = self.predict(x)
    rmse = np.sqrt(np.sum(np.square(predicted_y-y))/x.shape[0])
    return rmse
```

In [6]:
```
model = LinearModel()
```

In [7]:
```
model.fit(obj.x,obj.y)
```

Out[7]:
```
(9.379816850089613, 2143.430221913603)
```

In [8]:
```
model.average_error(obj.x,obj.y)
```

Out[8]:
```
157.9563099222744
```

In [9]:
```
obj.head()
```

Out[9]:

	x	y
0	329.923993	5172.183623
1	330.659897	5156.647979
2	338.970577	5248.960807
3	313.505977	5168.992688
4	304.432691	5046.956522

In [10]:
```
model.predict([325])
```

Out[10]:
```
array([5191.87069819])
```

Logistic Regression

68_Adnan Shaikh

In [3]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
digits = load_digits()
```

In [4]:
```
x_train,x_test,y_train,y_test=train_test_split(digits.data,digits.target,test_size=0.3)
```

In [5]:
```
lr=LogisticRegression(solver='liblinear',multi_class='ovr')
lr.fit(x_train,y_train)
lr.score(x_test,y_test)
```

Out[5]:
```
0.9611111111111111
```

In [6]:
```
svm= SVC(gamma='auto')
svm.fit(x_train,y_train)
svm.score(x_test,y_test)
```

Out[6]:
```
0.32222222222222224
```

In [7]:
```
rf=RandomForestClassifier(n_estimators=40)
rf.fit(x_train,y_train)
rf.score(x_test,y_test)
```

Out[7]:
```
0.9648148148148148
```

K fold cross Validation

In [8]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits=3)
kf
```

Out[8]:
```
KFold(n_splits=3, random_state=None, shuffle=False)
```

In [9]:
```
for train_index ,test_index in kf.split([1,2,3,4,5,6,7,8,9]):
  print(train_index,test_index)
```

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

In [10]:
```
def get_score(model, x_train,x_test,y_train,y_test):
  model.fit(x_train,y_train)
  return model.score(x_test,y_test)
```

In [11]:
```
from sklearn.model_selection import StratifiedKFold
fold = StratifiedKFold(n_splits=3)

score_logistic=[]
score_svm=[]
score_rf=[]

for train_index,test_index in fold.split(digits.data,digits.target):
  x_train,x_test,y_train,y_test=digits.data[train_index],digits.data[test_index],digits.target[train_index],digits.target[test_index]
  score_logistic.append(get_score(LogisticRegression(solver='liblinear',multi_class='ovr'),x_train,x_test,y_train,y_test))
  score_svm.append(get_score(SVC(gamma='auto'),x_train,x_test,y_train,y_test))
  score_rf.append(get_score(RandomForestClassifier(n_estimators=40),x_train,x_test,y_train,y_test))
```

In [12]:
```
score_logistic
```

Out[12]:
```
[0.8948247078464107, 0.9532554257095158, 0.9098497495826378]
```

In [13]:
```
score_rf
```

Out[13]:
```
[0.9332220367278798, 0.9599332220367279, 0.9315525876460768]
```

In [14]:
```
score_svm
```

Out[14]:
```
[0.3806343906510851, 0.41068447412353926, 0.5125208681135225]
```

Cross_val_Score Function

In [15]:
```
from sklearn.model_selection import cross_val_score
```

In [16]:
```
cross_val_score(LogisticRegression(solver='liblinear',multi_class='ovr'),digits.data,digits.target,cv=3)
```

Out[16]:
```
array([0.89482471, 0.95325543, 0.90984975])
```

In [17]:
```
score1=cross_val_score(RandomForestClassifier(n_estimators=5),digits.data,digits.target,cv=3)
np.average(score1)
```

Out[17]:
```
0.8503060656649972
```

In [18]:
```
score2=cross_val_score(RandomForestClassifier(n_estimators=20),digits.data,digits.target,cv=3)
np.average(score2)
```

Out[18]:
```
0.9254312743461325
```

In [19]:
```
score3=cross_val_score(RandomForestClassifier(n_estimators=30),digits.data,digits.target,cv=3)
np.average(score3)
```

Out[19]:
```
0.9304396215915415
```

In [20]:
```
score4=cross_val_score(RandomForestClassifier(n_estimators=40),digits.data,digits.target,cv=3)
np.average(score4)
```

Out[20]:
```
0.9304396215915415
```

Multi-Variate Linear Regression

68_Adnan Shaikh

In [9]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
```

In [10]:
```
#Lambda function to create linearly dependent feature using linearly independent features
dep_feat = lambda X: np.sum([np.random.uniform(10,10)+np.random.randint(0,2)*np.random.randint(-10
,10)*np.random.rand()*x for x in X],axis=0)
```

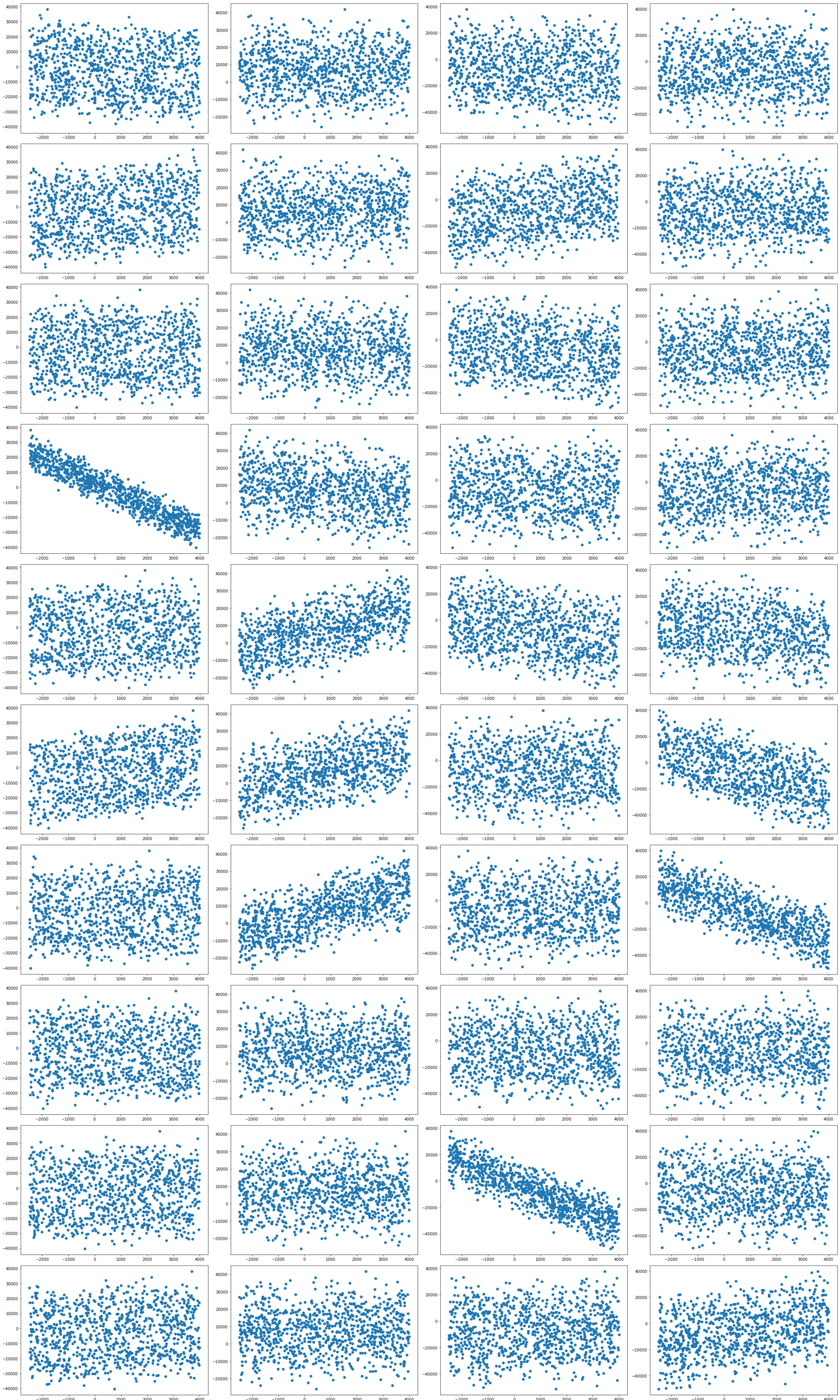
In [11]:
```
#Initializing constants N <- No. of records, NO_INDEP <- No. of Independent Features, NO_DEP <- N
o. of Dependent Features
N = 1000
NO_INDEP = 10
NO_DEP = 4
```

In [12]:
```
#Initializing Independent Features X and Dependent Features Y
X = np.array([np.random.uniform(low=-2500,high=4001,size=N) for _ in range(NO_INDEP)])
Y = np.transpose([dep_feat(X) for _ in range(NO_DEP)])
X = np.transpose(X)
print(f"Shape of Independent Array: {X.shape},Shape of Dependent Array: {Y.shape}")
print(f"Sample of X data from each features:\n {X[:10,:]}")
print(f"Sample of Y data from each features:\n {Y[:10,:]}")
```

```
Shape of Independent Array: (1000, 10),Shape of Dependent Array: (1000, 4)
Sample of X data from each features:
 [[ 3517.36411745  3242.55487782  1504.67607976  3226.63227622
   -920.1981244  -1707.33163588  3706.33546492  2932.22514714
   3569.83653402  -365.01638012]
 [ 2970.65588953  -398.40570039  2685.21420735  -755.71512461
   1410.56656655  2561.16992899  3298.43661622  3678.38940818
   -825.84615784  3853.50737534]
 [ 2264.11644431  1723.10709214  2575.55049952 -1545.96619944
  -1310.78850035  3785.76949599  3492.19283486 -1782.50668535
    724.47400094  1112.58514722]
 [ 1826.44220667 -2028.40216093  1547.78294526 -1125.21689097
   -518.83851474  -545.89148162 -1966.03802717  1491.70816542
   3274.73803414    62.60049293]
 [  684.06862355  -790.12721043  2459.11760112  3316.94509863
   3648.94615902  1570.81760048   103.78725663 -1978.66985567
   -564.87640845 -1221.29854422]
 [  875.10627978  2961.91361651  1300.35312827  -434.08739244
  -1097.70175763  -291.25786239 -2119.14918139  1786.16295187
   -353.02453889  1235.46064586]
 [  125.29622962  2506.00256549  1463.22622442 -1898.50438102
    196.46293069  3240.44440033   957.4538656  -1301.082368
    977.84967921   867.99298017]
 [-1170.80852259  -402.22742058  1805.49815175  2384.89282736
   -322.50930537  -640.78597159 -1037.2555477  -2273.88239981
   2052.57617831  -990.12846267]
 [-2309.07704444  1409.13580929  1517.52149903  -438.0430452
  -1118.0855794   2396.95264855  -961.24801975   -45.24891748
  -1948.04437598  1837.55807605]
 [ 2295.18705355 -1644.35723336  3257.59362967 -1348.72718286
  -2079.70400979  -372.83631433  -279.54406207  1099.88192433
  -2333.76306898  -678.09158362]]
Sample of Y data from each features:
 [[-30006.32284031   1797.11074228 -21365.7146995  -10003.18125475]
 [ 13145.29345795  27641.28417799  -3758.05432889 -25619.70098495]
 [ 23497.50496002  22732.26919346  -1048.03173066 -33701.56351273]
 [  4205.92218009  -9402.66895759 -33746.2089082   14332.16639925]
 [-25192.2736726   13613.87713678 -15578.31542618 -17673.47704061]
 [  6457.05888182 -12372.42860218  13302.0848313   24417.43081554]
 [ 27305.11625887  17149.78818647  -2069.22452284 -17498.41658928]
 [-21270.96301299 -10930.79251929 -22242.86025724   8213.42668203]
 [ 14689.71168783    128.67904649  20720.707765     3595.33946789]
 [  5949.04474399  -7617.30647249  13666.99309792   3514.0923618 ]]
```

In [13]:
```
#Scatter plot between each dependent and independent variable
fig,axes = plt.subplots(NO_INDEP,NO_DEP,figsize=(30,50),tight_layout=True)
for i in range(NO_INDEP):
  for j in range(NO_DEP):
    axes[i,j].scatter(x=X[:,i],y=Y[:,j])

plt.show()
```



In [14]:
```
class MultiVariateRegression:

  def fit_model(self,X,Y):
    # Estimator Beta[i,j] calculation using matrices (Generalization of Multiple Linear Regression to Multivariate Regression)
    self.X,self.Y =  np.concatenate((np.ones((len(X),1)),X),axis=1),np.array(Y)
    self.X_transpose = self.X.T
    self.compose_mat = np.matmul(self.X_transpose,self.X) #Making square matrix that contains all the required summation of (XiXj)
    self.compose_inverse = np.linalg.inv(self.compose_mat)
    self.Beta = np.matmul(np.matmul(self.compose_inverse,self.X_transpose),Y) #Calculating Beta Estimators

  def predict(self,X):
    return np.matmul(np.concatenate(([1],X)),self.Beta) #Predicting Single Sample


  def predict_many(self,indep_feat):
    #Predicting Multiple Sample
    Y = []
    for x in indep_feat:
      Y.append(self.predict(x))
    return np.array(Y)

  def residual(self,Y,predicted_Y):
    #Error calculation using SSR and averaging errors of all the dependent variables
    sq_of_res = np.square(np.subtract(Y,predicted_Y))
    np.round(sq_of_res,2)
    ssr = np.sum(np.transpose(sq_of_res),axis=1)
    return np.round(np.average(ssr),2)
```

In [15]:
```
#Model Demonstration
mvr = MultiVariateRegression()
X_train, X_test, Y_train,Y_test = train_test_split(X,Y,test_size=0.33,random_state=69,shuffle=True) # Splitting in training and test set
mvr.fit_model(X_train,Y_train)
Y_predicted = mvr.predict_many(X_test)
print(f"Residual: {mvr.residual(Y_test,Y_predicted)}")
#Residual is zero since each dependent vector is a linear combination of independent vectors (refer block 2) it shows model working as required
#It can also be used to check relation between independent variables
```

```
Residual: 0.0
```

Random Forest

68_Adnan Shaikh

In [1]:
```
import pandas as pd
from sklearn.datasets import load_digits
digits = load_digits()
```

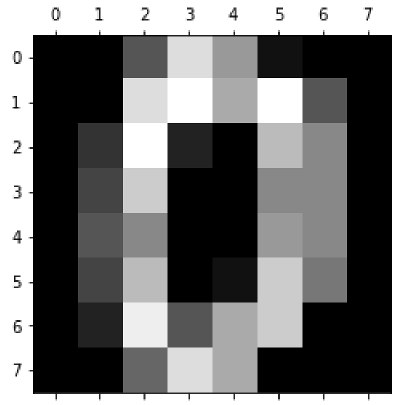
In [2]:
```
dir(digits)
```

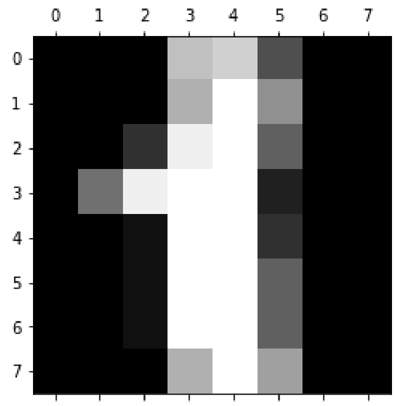
Out[2]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

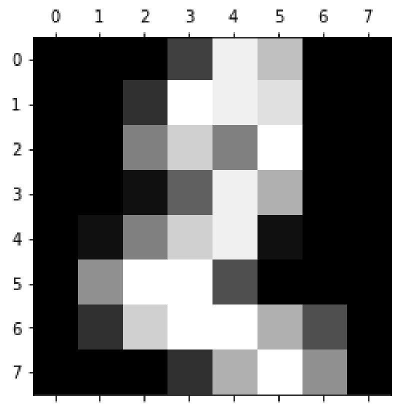
In [3]:
```
import matplotlib.pyplot as plt
```

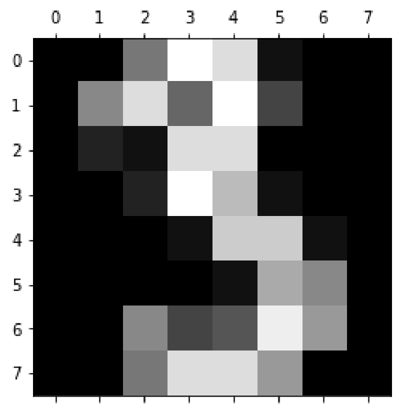
In [4]:
```python
plt.gray()
for i in range(4):
    plt.matshow(digits.images[i])
```

```
<Figure size 432x288 with 0 Axes>
```









In [5]:
```
df = pd.DataFrame(digits.data)
df.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0

5 rows × 64 columns

In [6]:
```
df['target'] = digits.target
```

In [7]:
```
df[0:12]
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4
5	0.0	0.0	12.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	9.0	16.0	16.0	10.0	0.0	0.0	5
6	0.0	0.0	0.0	12.0	13.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	9.0	15.0	11.0	3.0	0.0	6
7	0.0	0.0	7.0	8.0	13.0	16.0	15.0	1.0	0.0	0.0	...	0.0	0.0	0.0	13.0	5.0	0.0	0.0	0.0	0.0	7
8	0.0	0.0	9.0	14.0	8.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	11.0	16.0	15.0	11.0	1.0	0.0	8
9	0.0	0.0	11.0	12.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	9.0	12.0	13.0	3.0	0.0	0.0	9
10	0.0	0.0	1.0	9.0	15.0	11.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	10.0	13.0	3.0	0.0	0.0	0
11	0.0	0.0	0.0	0.0	14.0	13.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	13.0	16.0	1.0	0.0	1

12 rows × 65 columns

Train and the model and prediction

In [8]:
```
X = df.drop('target',axis='columns')
y = df.target
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

In [10]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)
```

Out[10]:
```
RandomForestClassifier(n_estimators=20)
```

In [11]:
```
model.score(X_test, y_test)
```

Out[11]:
```
0.975
```

In [12]:
```
y_predicted = model.predict(X_test)
```

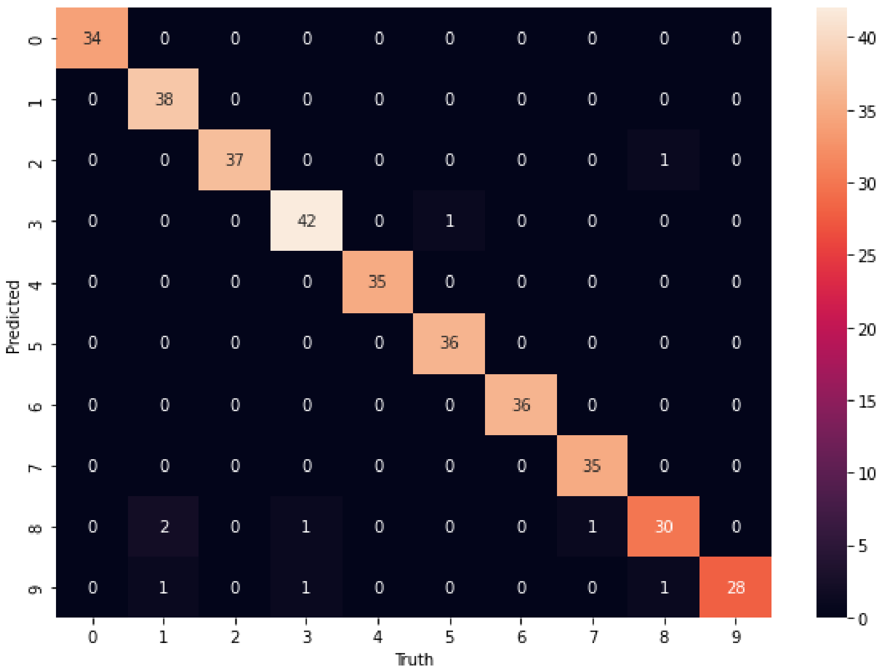
Confusion Matrix

In [13]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

Out[13]:
```
array([[34,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 38,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 37,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0,  0, 42,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 35,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 36,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 36,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 35,  0,  0],
       [ 0,  2,  0,  1,  0,  0,  0,  1, 30,  0],
       [ 0,  1,  0,  1,  0,  0,  0,  0,  1, 28]], dtype=int64)
```

In [14]:
```
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Truth')
plt.ylabel('Predicted')
```

Out[14]:
```
Text(69.0, 0.5, 'Predicted')
```



SVM

68_Adnan Shaikh

In [3]:
```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

In [4]:
```
iris.feature_names
```

Out[4]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [5]:
```
dir(iris)
```

Out[5]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

In [6]:
```
iris.target_names
```

Out[6]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [7]:
```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [8]:
```
df['target'] = iris.target
df.head()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [9]:
```
df[df.target==1].head()
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

In [10]:
```
df[df.target==2].head()
```

Out[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

In [11]:
```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [12]:
```
df[45:55]
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

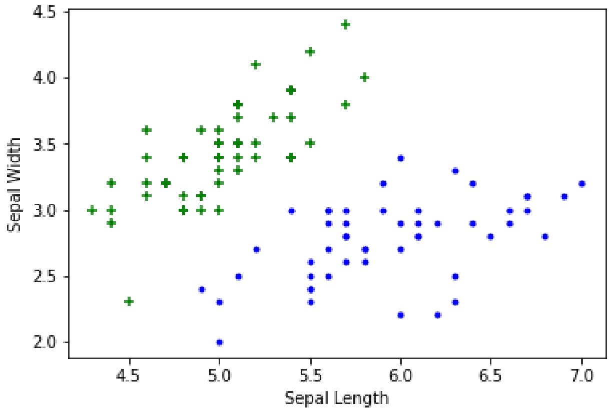
In [13]:
```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

In [14]:
```
import matplotlib.pyplot as plt
```

Sepal length vs Sepal Width (Setosa vs Versicolor)

In [15]:
```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

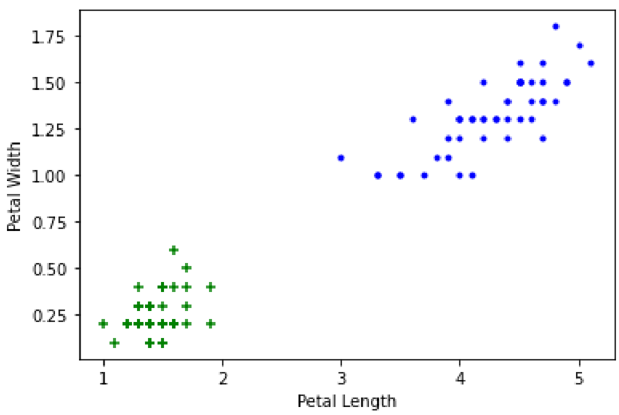
Out[15]: <matplotlib.collections.PathCollection at 0x18b41d9bd60>



Petal length vs Pepal Width (Setosa vs Versicolor)

In [16]:
```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

Out[16]: <matplotlib.collections.PathCollection at 0x18b42550400>



Train Using Support Vector Machine (SVM)

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [20]:
```
len(X_train)
```

Out[20]: 120

In [21]:
```
len(X_test)
```

Out[21]: 30

In [22]:
```
from sklearn.svm import SVC
model = SVC()
```

In [23]:
```
model.fit(X_train, y_train)
```
Out[23]: `SVC()`

In [24]:
```
model.score(X_test, y_test)
```
Out[24]: `0.8666666666666667`

In [25]:
```
model.predict([[4.8,3.0,1.5,0.3]])
```
```
C:\Users\adnan\anaconda3\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid
feature names, but SVC was fitted with feature names
  warnings.warn(
```
Out[25]: `array([0])`

Tune parameters

1. Regularization (C)

In [26]:
```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```
Out[26]: `0.8666666666666667`

In [27]:
```
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```
Out[27]: `0.9333333333333333`

1. Gamma

In [28]:
```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```
Out[28]: `0.9333333333333333`

1. Kernel

In [29]:
```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
```
Out[29]: `SVC(kernel='linear')`

In [30]:
```
model_linear_kernal.score(X_test, y_test)
```
Out[30]: `0.9666666666666667`