Practical No. 4

Aim: Practical of Decision tree.

Code:

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

In [2]:
```
dataset = pd.read_csv('C:\\Users\\Student\\Downloads\\petrol_consumption.csv')
```

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

Out[3]:

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

In [4]:
```
X = dataset.drop('Petrol_Consumption', axis=1)
```

In [5]:
```
y = dataset['Petrol_Consumption']
```

In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [8]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [9]:
```
regressor = DecisionTreeRegressor()
```

In [10]:
```
regressor.fit(X_train, y_train)
```

Out[10]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

In [11]:
```
y_pred = regressor.predict(X_test)
```

In [12]:
```
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
```

In [13]:
```
df
```

Out[13]:

	Actual	Predicted
29	534	541.0
4	410	414.0
26	577	574.0
30	571	554.0
32	577	631.0
37	704	644.0
34	487	628.0
40	587	649.0
7	467	414.0
10	580	510.0

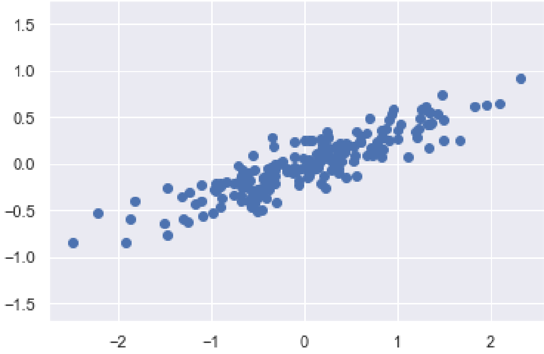
Practical No. 3

Aim: Practical of Principal Component Analysis.

Code:

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

In [4]:
```
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');
```



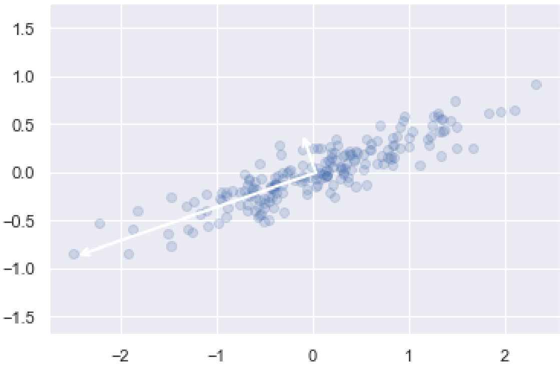
In [5]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

Out[5]:
```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [7]:
```
print(pca.explained_variance_)
```

```
[0.7625315 0.0184779]
```

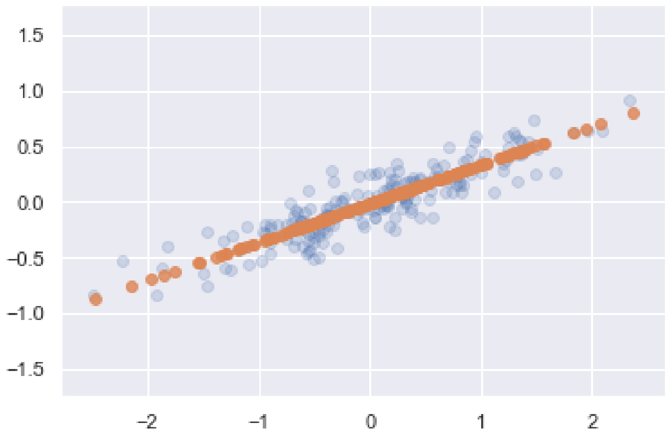
In [11]:
```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->', linewidth=2, shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



In [12]:
```
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:  ", X.shape)
print("transformed shape:", X_pca.shape)
```

```
original shape:   (200, 2)
transformed shape: (200, 1)
```

In [13]:
```
X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
```



Practical No. 1

Aim: Practical of K-Nearest Neighbour.

Code:

In [1]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

In [2]:
```
shu11 = ["Dhoni","Dubey"]
rish11= ["Kohli","Pandya"]
dip11 = ["Rishabh","Rohit"]
```

In [3]:
```
s11=[[50.58,87.56],[9,150]]
r11=[[60.02,93.82],[29.91,115.50]]
d11=[[97.27,98.04],[48.46,88.44]]
```

In [4]:
```
common11=[]
for i in s11:
    common11.append(i)
for j in r11:
    common11.append(j)
for k in d11:
    common11.append(k)
```

In [5]:
```
common11
```

Out[5]:
```
[[50.58, 87.56],
 [9, 150],
 [60.02, 93.82],
 [29.91, 115.5],
 [97.27, 98.04],
 [48.46, 88.44]]
```

In [6]:
```
common_name11=[]
for i in shu11:
    common_name11.append(i)
for j in rish11:
    common_name11.append(j)
```

In [7]:
```
common_name11
```

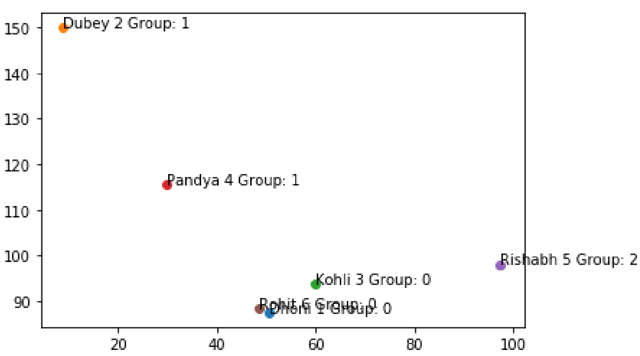
Out[7]:
```
['Dhoni', 'Dubey', 'Kohli', 'Pandya', 'Rishabh', 'Rohit']
```

In [8]:
```
model=KMeans(n_clusters=3)
```

In [9]:
```
model.fit(common11)
```

Out[9]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [10]:
```
counter=0
for i in common11:
    plt.scatter(i[0],i[1])
    plt.text(i[0],i[1],common_name11[counter] + " " + str(counter+1) + " Group: " + str(model.labels_[counter]))
    counter += 1
```



Practical No. 2

Aim: Practical of Clustering using KMeans.

Code:

In [1]:
```
import pandas as pd
```

In [2]:
```
from sklearn.cluster import KMeans
```

In [3]:
```
from sklearn.linear_model import LinearRegression
```

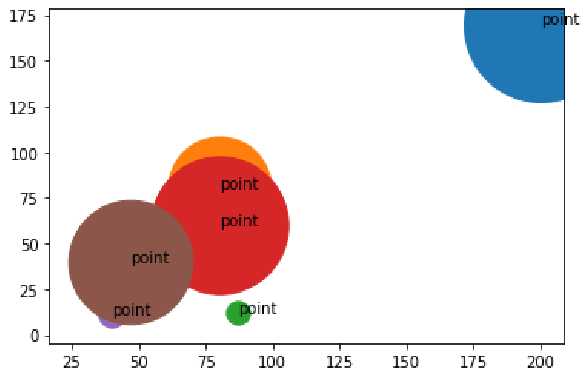
In [4]:
```
import matplotlib.pyplot as plt
```

In [5]:
```
import numpy as np
```

In [6]:
```
x=[[200,170,10000],[80,80,4500],[87,12,233],[80,60,8000],[40,11,300],[47,40,6500]]
```

In [7]:
```
y=[[200,170,10000],[80,80,4500],[87,12,233],[80,60,8000],[40,11,300],[47,40,6500]]
```

In [8]:
```
for i in x:
    plt.scatter(i[0],i[1],i[2])
    plt.text(i[0],i[1],"point")
```



In [9]:
```
num_clusters=3
model = KMeans
```

In [10]:
```
regression_model = LinearRegression()
```

In [11]:
```
regression_model.fit(x,y)
```

Out[11]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [12]:
```
model = KMeans(n_clusters=3)
```

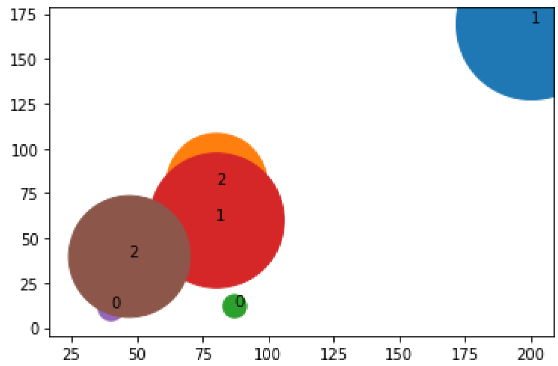
In [13]:
```
model.fit(x)
```

Out[13]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [14]:
```
model.labels_
```

Out[14]:
```
array([1, 2, 0, 1, 0, 2])
```

In [15]:
```
counter=0
for i in x:
    plt.scatter(i[0],i[1],i[2])
    plt.text(i[0],i[1],model.labels_[counter])
    counter += 1
```



In [16]:
```
def getStringFromList(lst):
    result=""
    for i in lst:
        result = result + str(i)+","
    return result[0:len(result)-1]
```

In [17]:
```
for i in range(len(x)):
    print("Element " + getStringFromList(x[i]) + " belongs to the group " + str(model.labels_[i]))
```

```
Element 200,170,10000 belongs to the group 1
Element 80,80,4500 belongs to the group 2
Element 87,12,233 belongs to the group 0
Element 80,60,8000 belongs to the group 1
Element 40,11,300 belongs to the group 0
Element 47,40,6500 belongs to the group 2
```

Practical No. 5

Aim: Practical of Hypothesis Testing.

Code:

```
In [1]: from scipy.stats import ttest_rel #t_test testing
```

```
In [2]: d1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
        d2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075, -0.169]
        stat, p = ttest_rel(d1, d2)
        print('stat=%.3f, p=%.3f' % (stat, p))

        stat=-0.334, p=0.746
```

```
In [3]: if p > 0.05:
            print('Probably the same distribution')
        else:
            print('Probably different distributions')

        Probably the same distribution
```

```
In [4]: from scipy.stats import chi2_contingency  #chisquare testing
```

```
In [5]: table = [[10, 20, 30],[6,  9,  17]]
        stat, p, dof, expected = chi2_contingency(table)
        print('stat=%.3f, p=%.3f' % (stat, p))

        stat=0.272, p=0.873
```

```
In [6]: if p > 0.05:
            print('Probably independent')
        else:
            print('Probably dependent')

        Probably independent
```

Practical No. 6

Aim: Practical of Time series forecasting.

Code:

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
dataset= pd.read_csv('C:\\Users\\Student\\OneDrive\\DS\\AirPassengers.csv')
```

In [4]:
```
print(dataset.head())
print(dataset.dtypes)
```

```
     Month  #Passengers
0  1949-01          112
1  1949-02          118
2  1949-03          132
3  1949-04          129
4  1949-05          121
Month          object
#Passengers     int64
dtype: object
```

In [7]:
```
from datetime import datetime
con= dataset['Month']
dataset['Month']=pd.to_datetime(dataset['Month'])
dataset.set_index('Month',inplace=True)
dataset.index
```

Out[7]:
```
DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',
               '1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',
               '1949-09-01', '1949-10-01',
               ...
               '1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',
               '1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',
               '1960-11-01', '1960-12-01'],
              dtype='datetime64[ns]', name='Month', length=144, freq=None)
```

In [8]:
```
ts=dataset['#Passengers']
```

In [9]:
```
ts.head(10)
```

Out[9]:
```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
1949-06-01    135
1949-07-01    148
1949-08-01    148
1949-09-01    136
1949-10-01    119
Name: #Passengers, dtype: int64
```

In [10]:
```
ts['1949-01-01']
```

Out[10]:
```
112
```

In [11]:
```
ts['1949-01-01':'1949-05-01']
```

Out[11]:
```
Month
1949-01-01    112
1949-02-01    118
1949-03-01    132
1949-04-01    129
1949-05-01    121
Name: #Passengers, dtype: int64
```

Practical No. 7

Aim: Practical of Linear Regression.

Code:

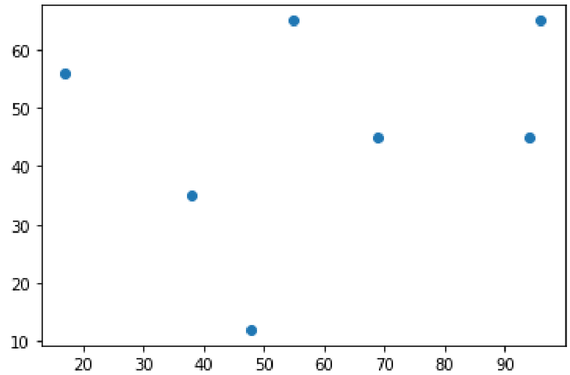
In [1]:
```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

In [2]:
```
model = LinearRegression()
```

In [3]:
```
x1=[55,96,17,94,38,48,69]
x=[[55],[96],[17],[94],[38],[48],[90]]
y=[65,65,56,45,35,12,45]
```

In [4]:
```
plt.scatter(x1,y)
```

Out[4]: <matplotlib.collections.PathCollection at 0x26a30b43e08>



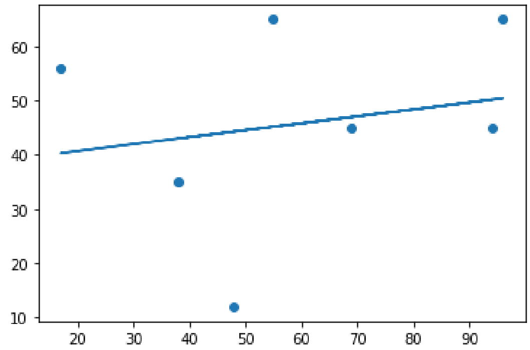
In [5]:
```
model.fit(x,y)
```

Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [6]:
```
y_pred=[]
for i in x1:
    y_pred.append(model.predict([[i]]))
```

In [7]:
```
plt.scatter(x1,y)
plt.plot(x1, y_pred)
```

Out[7]: [<matplotlib.lines.Line2D at 0x26a30f35848>]



In [8]:
```
model.predict([[45]])
```

Out[8]: array([43.89661534])

Practical No. 8

Aim: Practical of Logistic Regression.

Code:

```
In [1]: from sklearn.linear_model import LogisticRegression

In [2]: import matplotlib as plt

In [3]: model = LogisticRegression()

In [4]: x=[[60],[84],[25]]
        y=["average","smart","noob"]

In [5]: model.fit(x,y)

Out[5]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                           intercept_scaling=1, l1_ratio=None, max_iter=100,
                           multi_class='auto', n_jobs=None, penalty='l2',
                           random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                           warm_start=False)

In [6]: print(model.predict([[90]]))

        ['smart']
```