Data Science Manual

Program-1:-

Preliminary Information

```
import pandas as pd

df=pd.read_csv('IPL IMB381IPL2013.csv')

pd.set_option('display.max_columns',10)

df.head()

cols = list(df.columns)

cols

df.shape

df.info()
```

1).Grouping and Aggregating

```
df.loc[0:9]
df[['PLAYER NAME','COUNTRY']][0:9]
df.iloc[4:9,1:4]
```

2).Adding and Removing attributes

```
df['PREMIUM']=df['SOLD PRICE']-df['BASE PRICE']
df.head()

df.drop(‘ECON’,axis=1)
```

3).Grouping and Aggregating

```
sold_price_by_frame = df.groupby('AGE')['SOLD PRICE'].mean().reset_index()

sold_price_by_frame

sold_price_by_frame_role = df.groupby(['AGE','PLAYING ROLE'])['SOLD PRICE'].mean().reset_index()
```

```
sold_price_by_frame_role
```

4).Joining Dataframes

```
soldprice_comparison=sold_price_by_frame_role.merge(sold_price_by_frame,on = 'AGE',how = 'outer')

soldprice_comparison

soldprice_comparison.rename(columns = {'SOLD PRICE_x':'SOLD_PRICE_AGE_ROLE','SOLD PRICE_y':'SOLD_PRICE_AGE'},inplace = True)

soldprice_comparison
```

5).Filtering The Data

```
soldprice_comparison['CHANGE']=soldprice_comparison.apply(lambda x:(x.SOLD_PRICE_AGE_ROLE-x.SOLD_PRICE_AGE)/x.SOLD_PRICE_AGE,axis = 1)

soldprice_comparison

soldprice_comparison[soldprice_comparison.CHANGE > 0]
```

6).Handling NULL Values

```
soldprice_comparison[soldprice_comparison.CHANGE.isnull()]

soldprice_comparison[soldprice_comparison.CHANGE < 0]= None

soldprice_comparison

soldprice_comparison[soldprice_comparison.CHANGE.isnull()]

soldprice_comparison = soldprice_comparison.dropna(subset = ['CHANGE'])

soldprice_comparison[soldprice_comparison.CHANGE.isnull()]
```

Program-2:-

Preliminary Information

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

ipldf = pd.read_csv('IPL IMB381IPL2013.csv')

ipldf.head()
```

1).Bar Chart and Histogram

```
sns.barplot(x='AGE',y='SOLD PRICE',data = ipldf)

sns.barplot(x='AGE',y='SOLD PRICE',data = ipldf,hue='PLAYING ROLE')

plt.hist(ipldf['SOLD PRICE'])

plt.hist(ipldf['SOLD PRICE'],bins = 20)
```

2).Comparing Distribution

```
sns.distplot(ipldf[ipldf['CAPTAINCY EXP']==1]['SOLD PRICE'],color='y',label = 'Captaincy Experience')

sns.distplot(ipldf[ipldf['CAPTAINCY EXP']==0]['SOLD PRICE'],color='r',label = 'No Captaincy Experience')

plt.legend()
```

3).Box Plot and Mention Quartiles

```
sns.boxplot(x='PLAYING ROLE',y='SOLD PRICE',data = ipldf)
```

4).Correlation using pairplot and heatmap

```
infl_features = ['SR-B','AVE','SIXERS','SOLD PRICE']
sns.pairplot(ipldf[infl_features],size = 2)
ipldf[infl_features].corr()
sns.heatmap(ipldf[infl_features].corr(),annot = True)
```

Program-3:-

Preliminary Information

```
import pandas as pd
import numpy as np
df=pd.read_excel('IBM-313 Marks.xlsx')
print(df.head())
print(df.columns)
```

1).Central tendency

```
import scipy
from scipy import stats
data = df['Total']
print('MEAN = ',np.mean(df['Total']))
print('MEDIAN = ',np.median(df['Total']))
print('MEAN = ',scipy.mean(data))
print('MEDIAN = ',scipy.median(data))
print('MODE = ',stats.mode(data))
from scipy import stats
x=df['Total']
```

```
y=np.array(x)
print('Percentile = ',np.percentile(y,30))
```

2).Dispersion And Distribution

```
range=max(y)-min(y)
print("RANGE = ",range)
Q1 = np.percentile(y,25)
Q3 = np.percentile(y,75)
print("IQR = ",Q3-Q1)
print("VARIANCE = ",np.var(y))
import statistics
print("POPULATION STANDARD DEVIATION = ",statistics.pstdev(y))
print("SAMPLE STANDARD DEVIATION = ",statistics.stdev(y))
from scipy.stats import skew
print(skew(y))
```

3).ANOVA

```
anova_df = pd.read_excel('discounts.xlsx')
anova_df.head()
import seaborn as sns
import matplotlib.pyplot as plt
sns.distplot(anova_df['discount_0'],label = 'No Discount')
sns.distplot(anova_df['discount_10'],label = '10% Discount')
sns.distplot(anova_df['discount_20'],label = '20% Discount')
plt.legend()
from scipy.stats import f_oneway
f_oneway(anova_df['discount_0'],anova_df['discount_10'],anova_df['discount_20'])
```

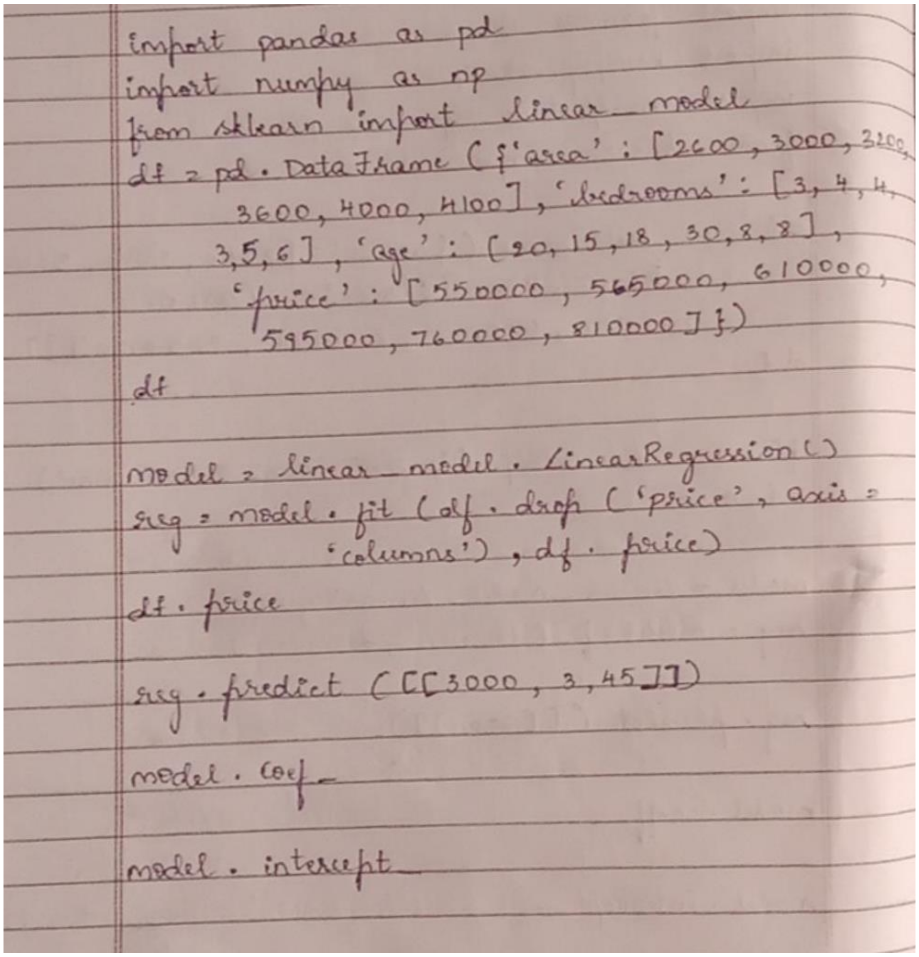
4).Hypothesis Testing

```
pp_df=pd.read_excel('passport.xlsx')

pp_df.head()

print(list(pp_df.processing_time))

import math

def z_test(p_mean,p_std,sample):

    z_score = (sample.mean() - p_mean)/(p_std/math.sqrt(len(sample)))

    return z_score,stats.norm.cdf(z_score)

z_test(30,12.5,pp_df.processing_time)
```

Program-4:-

```
import numpy as np

import matplotlib.pyplot as plt

def estimate_coef(x, y):

    n = np.size(x)

    m_x, m_y = np.mean(x), np.mean(y)

    SS_xy = np.sum(y*x) - n*m_y*m_x

    SS_xx = np.sum(x*x) - n*m_x*m_x

    b_1 = SS_xy / SS_xx

    b_0 = m_y - b_1*m_x

    return(b_0, b_1)

def plot_regression_line(x, y, b):

    plt.scatter(x, y, color = "m", marker = "o", s = 30)

    y_pred = b[0] + b[1]*x

    plt.plot(x, y_pred, color = "g")

    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
def main():
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \nb_1 = {}".format(b[0], b[1]))
    plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()
```

Program-5:-



```
import pandas as pd
import numpy as np
from sklearn import linear_model
df = pd.DataFrame({'area': [2600, 3000, 3200,
        3600, 4000, 4100], 'bedrooms': [3, 4, 4,
       3,5,6], 'age': [20, 15, 18, 30, 8, 8],
        'price': [550000, 565000, 610000,
         595000, 760000, 810000]})

df

model = linear_model.LinearRegression()
reg = model.fit(df.drop('price', axis =
              'columns'), df.price)
df.price

reg.predict([[3000, 3, 45]])

model.coef_

model.intercept_
```

Program-6:-

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix
x=np.arange(10).reshape(-1,1)
y=np.array([0,1,0,0,1,1,1,1,1,1])
model=LogisticRegression(solver='liblinear',C=10.0,random_state=0)
model.fit(x,y)
p_pred=model.predict_proba(x)
y_pred=model.predict(x)
score_=model.score(x,y)
conf_m=confusion_matrix(y,y_pred)
report=classification_report(y,y_pred)
print('x:',x,sep='\n')
print('y:',x,sep='\n',end='\n\n')
print('intercept:',model.intercept_)
print('coef:',model.coef_,end='\n\n')
print('p_pred:',p_pred,sep='\n',end='\n\n')
print('y_pred:',y_pred,end='\n\n')
print('score_:',score_,end='\n\n')
print('conf_m:',conf_m,sep='\n',end='\n\n')
print('report:',report,sep='\n')
```

Program-7:-

```
from sklearn.datasets import load.iris()
iris=load.iris()
x=iris.data
y=iris.target


from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=0)

from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
y_pred=gnb.fit(X_train,y_train).predict(x_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(“Accuracy=”,accuracy_score(y_test,y_pred)

print(“Confusion Matrix=”,confusion_matrix(y_test,y_pred)

print(“Classification Report=”,classification_report(y_test,y_pred)
```

Program-8:-

```
from sklearn.datasets import load.iris()
iris=load.iris()
x=iris.data
y=iris.target


from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=0)

from sklearn.tree import DecisionTreeClassifier
t= DecisionTreeClassifier()
y_pred=t.fit(X_train,y_train).predict(x_test)

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(“Accuracy=”,accuracy_score(y_test,y_pred)

print(“Confusion Matrix=”,confusion_matrix(y_test,y_pred)

print(“Classification Report=”,classification_report(y_test,y_pred)
```

Program-9:-

```
from sklearn.datasets import load.iris()
iris=load.iris()
x=iris.data
y=iris.target


from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=0)
```

```
from sklearn.ensemble import RandomForestClassifier
t= RandomForestClassifier ()
y_pred=t.fit(X_train,y_train).predict(x_test)

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(“Accuracy=”,accuracy_score(y_test,y_pred)

print(“Confusion Matrix=”,confusion_matrix(y_test,y_pred)

print(“Classification Report=”,classification_report(y_test,y_pred)
```

Program-10:-

```
from sklearn.datasets import load.iris()
iris=load.iris()
x=iris.data
y=iris.target


from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=0)

from sklearn.neighbors import KNeighborsClassifier
t= KNeighborsClassifier ()
y_pred=t.fit(X_train,y_train).predict(x_test)

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print(“Accuracy=”,accuracy_score(y_test,y_pred)

print(“Confusion Matrix=”,confusion_matrix(y_test,y_pred)

print(“Classification Report=”,classification_report(y_test,y_pred)
```

Program-11:-

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

X = np.array([[5,3],
    [10,15],
    [15,12],
    [24,10],
    [30,45],
    [85,70],
    [71,80],
    [60,78],
    [55,52],
    [80,91],])

plt.scatter(X[:,0],X[:,1], label='True Position')
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

Program-12:-

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np

X = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])

clustering = AgglomerativeClustering(n_clusters = 2).fit(X)

print(clustering.labels_)
```