**Pratical No:- 1**

**Pratical Name:- Calculate the mean and standard deviation.**

# Roll No:-

import numpy as np # Example data

data = [10, 20, 30, 40, 50]

# Calculate mean mean = np.mean(data)

# Calculate standard deviation std\_dev = np.std(data)

print("Mean:", mean) print("Standard Deviation:", std\_dev)

**Pratical No:- 2**

# Pratical Name:- Read the CSV file. Roll No:-

import csv

# Open the CSV file

with open('your\_file.csv', mode='r') as file: csv\_reader = csv.reader(file)

# Skip the header row if necessary next(csv\_reader) # If there's a header in your CSV

# Read and print each row for row in csv\_reader:

print(row) import pandas as pd

# Read the CSV file into a DataFrame df = pd.read\_csv('your\_file.csv')

# Display the first few rows print(df.head())

-------------------------------------------------------------------------------------------------

import pandas as pd  
  
file\_path="C:\\Users\\Sandeep Bonde\\OneDrive\\Desktop\\Book1.csv"  
  
sales\_data=pd.read\_csv(file\_path)  
  
print("Display the rows of the dataset")  
print(sales\_data)

**Pratical No:- 3**

# Pratical Name:- Perform data filtering, and calculate aggregate statistics Roll No:-

import pandas as pd

# Load the CSV file into a DataFrame df = pd.read\_csv('your\_file.csv')

filtered\_data = df[df['Age'] > 30]

filtered\_data = df[(df['Age'] > 30) & (df['Salary'] > 50000)] mean\_salary = filtered\_data['Salary'].mean()

print(f"Mean Salary: {mean\_salary}")

sum\_salary = filtered\_data['Salary'].sum() print(f"Total Salary: {sum\_salary}")

row\_count = filtered\_data.shape[0] print(f"Number of rows: {row\_count}")

# Group by 'Department' and calculate the mean of numeric columns grouped\_data = df.groupby('Department').mean()

print(grouped\_data)

aggregate\_stats = df.groupby('Department').agg({ 'Salary': ['mean', 'sum'],

'Age': 'mean'

})

print(aggregate\_stats) import pandas as pd # Load the CSV file

df = pd.read\_csv('your\_file.csv')

# Filter rows where 'Age' > 30 and 'Salary' > 50000 filtered\_data = df[(df['Age'] > 30) & (df['Salary'] > 50000)]

# Calculate aggregate statistics

mean\_salary = filtered\_data['Salary'].mean() total\_salary = filtered\_data['Salary'].sum() row\_count = filtered\_data.shape[0]

print(f"Mean Salary: {mean\_salary}") print(f"Total Salary: {total\_salary}") print(f"Number of rows: {row\_count}")

# Group by 'Department' and calculate mean salary and age grouped\_data = df.groupby('Department').agg({

'Salary': ['mean', 'sum'], 'Age': 'mean'

})

print(grouped\_data)

-------------------------------------------------------------------------------------------------

import numpy as np  
import pandas as pd  
df=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],[np.nan],[np.nan],[np.nan]],columns=['A','B','C'])  
print(df)  
  
r1=df.agg(['sum','min','max'])  
print(r1)  
  
r2=df.agg({'A':['sum','min','max'],'C':['sum','min','max']})  
print(r2)  
  
  
r3=df.agg(x=('A','max'),y=('B','max'),z=('C','max'))  
print(r3)  
  
r4=df.agg("mean",axis=1)  
print(r4)  
  
r5=df.agg("mean",axis=0)  
print(r5)

**Pratical No:- 4**

# Pratical Name:- Calculate total sales by month. Roll No:-

import pandas as pd

df = pd.read\_csv('your\_sales\_data.csv')

df['Date'] = pd.to\_datetime(df['Date'])

df['YearMonth'] = df['Date'].dt.to\_period('M')

total\_sales\_by\_month = df.groupby('YearMonth')['Sales'].sum().reset\_index()

print(total\_sales\_by\_month)

import pandas as pd

df = pd.read\_csv('your\_sales\_data.csv')

df['Date'] = pd.to\_datetime(df['Date'])

df['YearMonth'] = df['Date'].dt.to\_period('M')

total\_sales\_by\_month = df.groupby('YearMonth')['Sales'].sum().reset\_index()

print(total\_sales\_by\_month)

import pandas as pd  
  
data={  
 "Date":["2023-01-15","2023-01-20","2023-02-10","2023-02-15","2023-03-01"],  
 "Sales":[200,150,300,250,400],  
}  
df=pd.DataFrame(data)  
  
df['Date']=pd.to\_datetime(df['Date'])  
  
df['YearMonth']=df['Date'].dt.to\_period('M')  
  
monthly\_sales=df.groupby('YearMonth')['Sales'].sum()  
  
print(monthly\_sales)

**Pratical No:- 5**

# Pratical Name:- Implement the Clustering using K-means. Roll No:-

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans from sklearn.datasets import make\_blobs

X, y = make\_blobs(n\_samples=300, centers=4, random\_state=42)

kmeans = KMeans(n\_clusters=4)

kmeans.fit(X)

centroids = kmeans.cluster\_centers\_

labels = kmeans.labels\_

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50) plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', c='red', s=200, label='Centroids')

plt.title('K-Means Clustering') plt.legend()

plt.show()

-------------------------------------------------------------------------------------------------

import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn.datasets import make\_blobs  
  
X,y=make\_blobs(n\_samples=300,centers=4,cluster\_std=0.6,random\_state=42)  
  
plt.scatter(X[:,0],X[:,1],s=50,c='gray',marker='o')  
plt.title("Raw Data")  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.show()  
  
kmeans=KMeans(n\_clusters=4,random\_state=42)  
kmeans.fit(X)  
  
labels=kmeans.labels\_  
centroids=kmeans.cluster\_centers\_  
  
plt.scatter(X[:,0],X[:,1],c=labels,cmap='viridis',s=50)  
plt.scatter(centroids[:,0],centroids[:,1],s=200,c='red',marker='X',label='Centroids')  
plt.title("K-means Clustering")  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.legend()  
plt.show()

**Pratical No:- 6**

# Pratical Name:- Classification using Random Forest. Roll No:-

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.datasets import load\_iris

from sklearn.metrics import classification\_report, accuracy\_score

data = load\_iris()

X = data.data

y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

y\_pred = rf\_classifier.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred)) print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

importances = rf\_classifier.feature\_importances\_

plt.figure(figsize=(8, 6)) plt.barh(data.feature\_names, importances)

plt.xlabel("Feature Importance") plt.title("Random Forest Feature Importance") plt.show()

-------------------------------------------------------------------------------------------------

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.datasets import load\_iris  
from sklearn.metrics import classification\_report,accuracy\_score  
  
data=load\_iris()  
  
X=data.data  
y=data.target  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
rf\_classifier.fit(X\_train, y\_train)  
  
y\_pred = rf\_classifier.predict(X\_test)  
  
print("Accuracy:", accuracy\_score(y\_test, y\_pred))  
print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  
  
import matplotlib.pyplot as plt  
  
importances = rf\_classifier.feature\_importances\_  
  
plt.figure(figsize=(8, 6))  
plt.barh(data.feature\_names, importances)  
  
plt.xlabel("Feature Importance")  
plt.title("Random Forest Feature Importance")  
plt.show()

**Pratical No:- 7**

# Pratical Name:- Regression Analysis using Linear Regression. Roll No:-

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

np.random.seed(42)

X = 2.5 \* np.random.randn(1000) + 25 # Feature (square footage)

y = 0.5 \* X + np.random.randn(1000) \* 5 + 15 # Target (house price)

X = X.reshape(-1, 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

y\_pred = lin\_reg.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}") print(f"R-squared: {r2}")

plt.figure(figsize=(8, 6))

plt.scatter(X\_test, y\_test, color='blue', label='Actual data')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Regression line')

plt.title('Linear Regression: House Price Prediction') plt.xlabel('Square Footage')

plt.ylabel('Price') plt.legend() plt.show()

-------------------------------------------------------------------------------------------------

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error,r2\_score  
  
data={  
 'YearsExperience':[1,2,3,4,5,6,7,8,9,10],  
 'Salary':[40000,42000,44000,46000,48000,50000,52000,54000,56000,58000],  
}  
  
df=pd.DataFrame(data)  
  
X=df[['YearsExperience']]  
y=df[['Salary']]  
  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=42)  
  
model=LinearRegression()  
model.fit(X\_train,y\_train)  
  
y\_pred=model.predict(X\_test)  
  
mse=mean\_squared\_error(y\_test,y\_pred)  
print(f"Mean Squared Error :{mse}")  
  
r2=r2\_score(y\_test,y\_pred)  
print(f"R-squared : {r2}")  
  
plt.scatter(X\_test,y\_test,color='blue',label='actual data')  
plt.plot(X\_test,y\_pred,color='red',label='Fitted line')  
plt.xlabel('Years of Experience ')  
plt.ylabel('salary')  
plt.title('Linear Regression :Salary vs Experience')  
plt.legend()  
plt.show()  
  
new\_data=np.array([[11]])  
predicted\_salary=model.predict(new\_data)  
  
print(f"predicted salary for 11 years of experience : ${predicted\_salary[0]:,.2f}")

**Pratical No:- 8**

# Pratical Name:- Association Rule Mining using Apriori. Roll No:-

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

data = {

'Milk': [1, 1, 1, 0, 1],

'Bread': [1, 1, 1, 1, 1],

'Butter': [0, 1, 1, 1, 1],

'Beer': [0, 0, 1, 1, 1],

'Diapers': [1, 1, 0, 1, 1],

}

df = pd.DataFrame(data)

frequent\_itemsets = apriori(df, min\_support=0.6, use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.0)

print("Frequent Itemsets:\n", frequent\_itemsets) print("\nAssociation Rules:\n", rules)

print("\nFormatted Association Rules:")

for index, rule in rules.iterrows():

print(f"Rule: {', '.join(list(rule['antecedents']))} -> {', '.join(list(rule['consequents']))}")

print(f"Support: {rule['support']}, Confidence: {rule['confidence']}, Lift:

{rule['lift']}") print("="\*50)

-------------------------------------------------------------------------------------------------

import pandas as pd  
from mlxtend.frequent\_patterns import apriori, association\_rules  
  
*# Sample data (Transaction format as one-hot encoded)*dataset = [  
 ['milk', 'bread', 'nuts'],  
 ['milk', 'bread'],  
 ['milk', 'bread', 'nuts', 'apple'],  
 ['bread', 'nuts'],  
 ['milk', 'apple']  
]  
  
*# Convert to DataFrame*from mlxtend.preprocessing import TransactionEncoder  
te = TransactionEncoder()  
te\_data = te.fit(dataset).transform(dataset)  
df = pd.DataFrame(te\_data, columns=te.columns\_)  
  
*# Step 1: Find frequent itemsets*frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)  
  
*# Step 2: Generate rules*rules = association\_rules(frequent\_itemsets, metric='confidence', min\_threshold=0.6)  
  
*# Display results*print("Frequent Itemsets:\n", frequent\_itemsets)  
print("\nAssociation Rules:\n", rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

-------------------------------------------------------------------------------------------------

import pandas as pd  
from mlxtend.frequent\_patterns import apriori,association\_rules  
  
data={  
 'Milk':[1,1,0,1,1],  
 'Bread':[1,1,1,1,0],  
 'Butter':[0,1,1,1,1],  
 'Cheese':[1,0,1,1,1]  
}  
df=pd.DataFrame(data)  
  
frequent\_itemsets=apriori(df,min\_support=0.6,use\_colnames=True)  
  
rules=association\_rules(frequent\_itemsets,metric="confidence",min\_threshold=0.7)  
  
print("frequent Itemsets :")  
print(frequent\_itemsets)  
  
print("\nAssociation Rules :")  
print(rules)

**Pratical No:- 9**

# Pratical Name:- Visualize the result of the clustering and compare. Roll No:-

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans from sklearn.datasets import make\_blobs

from sklearn.metrics import adjusted\_rand\_score

X, y = make\_blobs(n\_samples=300, centers=3, random\_state=42)

kmeans = KMeans(n\_clusters=3) kmeans.fit(X)

labels = kmeans.labels\_

centroids = kmeans.cluster\_centers\_

ari = adjusted\_rand\_score(y, labels)

print(f"Adjusted Rand Index (ARI) between true labels and predicted clusters:

{ari}")

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', s=50) plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', c='red', s=200, label='Centroids')

plt.title('KMeans Clustering (Predicted Labels)') plt.xlabel('Feature 1')

plt.ylabel('Feature 2') plt.legend()

# Visualize the true labels (if available) plt.subplot(1, 2, 2)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o', s=50) plt.title('True Labels')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.tight\_layout() plt.show()

---------------------------------------------------------------------------------------------

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.datasets import make\_blobs  
from sklearn.cluster import KMeans,DBSCAN  
  
X,y=make\_blobs(n\_samples=300,centers=3,cluster\_std=1.0,random\_state=42)  
  
kmeans=KMeans(n\_clusters=3,random\_state=42)  
kmeans\_labels=kmeans.fit\_predict(X)  
  
dbscan=DBSCAN(eps=0.8,min\_samples=5)  
dbscan\_labels=dbscan.fit\_predict(X)  
  
fig,axes=plt.subplots(1,2,figsize=(12,5))  
  
axes[0].scatter(X[:,0],X[:,1],c=kmeans\_labels,cmap='viridis',marker='o',edgecolor='k')  
axes[0].scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1],c='red',marker='X',s=200,label="centroids")  
axes[0].set\_title("K-Means Clustering")  
axes[0].legend()  
  
axes[1].scatter(X[:,0],X[:,1],c=dbscan\_labels,cmap='viridis',marker='o',edgecolor='k')  
axes[1].set\_title("DBSCAN clustering")  
  
plt.show()

**Pratical No:- 10**

# Pratical Name:- Visualize the correlation matrix using a pseudocolor plot Roll No:-

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

# Generate some sample data

data = np.random.rand(10, 5) # 10 rows, 5 columns

df = pd.DataFrame(data, columns=[f'Feature {i+1}' for i in range(5)])

# Compute the correlation matrix corr\_matrix = df.corr()

# Plotting the correlation matrix using a heatmap plt.figure(figsize=(8, 6))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Matrix Heatmap') plt.show()

-------------------------------------------------------------------------------------------------

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
np.random.seed(42)  
data=pd.DataFrame(np.random.rand(10,5),columns=['A','B','C','D','E'])  
  
corr\_matrix=data.corr()  
  
plt.figure(figsize=(8,6))  
plt.pcolormesh(corr\_matrix,cmap='coolwarm',edgecolor='k')  
plt.colorbar(label='Correlation Coefficient')  
  
plt.xticks(np.arange(0.5,len(corr\_matrix.columns),1),corr\_matrix.columns)  
plt.yticks(np.arange(0.5,len(corr\_matrix.index),1),corr\_matrix.index)  
plt.title('Correlation Matrix Heatmap')  
  
plt.show()

**Pratical No:- 11**

# Pratical Name:- Use of degrees distribution of a network. Roll No:-

import networkx as nx

import matplotlib.pyplot as plt import numpy as np

# Create a sample graph (e.g., Erdos-Renyi random graph)

G = nx.erdos\_renyi\_graph(n=100, p=0.05) # 100 nodes, probability of edge creation = 0.05

# Get the degree of each node

degrees = [G.degree(n) for n in G.nodes()]

# Compute the frequency of each degree degree\_count = np.bincount(degrees)

# Plot the degree distribution plt.figure(figsize=(8, 6))

plt.bar(range(len(degree\_count)), degree\_count, width=0.8, color='b', alpha=0.7)

plt.xlabel('Degree') plt.ylabel('Frequency')

plt.title('Degree Distribution of the Network') plt.show()

# Create a Barabási–Albert scale-free network

G = nx.barabasi\_albert\_graph(n=100, m=2) # 100 nodes, each new node connects to 2 existing nodes

# Get the degree of each node and calculate the degree distribution as before degrees = [G.degree(n) for n in G.nodes()]

degree\_count = np.bincount(degrees) # Plot the degree distribution plt.figure(figsize=(8, 6))

plt.bar(range(len(degree\_count)), degree\_count, width=0.8, color='r', alpha=0.7) plt.xlabel('Degree')

plt.ylabel('Frequency')

plt.title('Degree Distribution of Barabási–Albert Network') plt.show()

-------------------------------------------------------------------------------------------------

import networkx as nx  
import matplotlib.pyplot as plt  
  
G=nx.erdos\_renyi\_graph(n=100,p=0.05)  
  
degree\_sequence=[G.degree(node) for node in G.nodes() ]  
  
degree\_count={}  
for degree in degree\_sequence:  
 degree\_count[degree]=degree\_count.get(degree,0)+1  
  
degrees=list(degree\_count.keys())  
counts=list(degree\_count.values())  
  
plt.figure(figsize=(8,6))  
plt.bar(degrees,counts,color='b')  
plt.xlabel('Degree(k)')  
plt.ylabel('Number of Nodes (Count)')  
plt.title('Degree Distribution of the Network ')  
plt.show()  
  
plt.figure(figsize=(8,6))  
plt.loglog(degrees,counts,marker='o',color='r')  
plt.xlabel('Log(Degree)')  
plt.ylabel('Log(Count)')  
plt.title('Log-Log Degree Distribution ')  
plt.show()

**Pratical No:- 12**

# Pratical Name:- Graph visualization of a network using maximum, minimum, median, first quartile and third quartile

**Roll No:-**

import networkx as nx

import matplotlib.pyplot as plt import numpy as np

# Create a random graph (e.g., Erdos-Renyi random graph)

G = nx.erdos\_renyi\_graph(n=100, p=0.05) # 100 nodes, probability 0.05

# Compute the degree of each node degrees = [G.degree(n) for n in G.nodes()]

# Compute degree statistics degree\_min = np.min(degrees) degree\_max = np.max(degrees) degree\_median = np.median(degrees) degree\_q1 = np.percentile(degrees, 25) degree\_q3 = np.percentile(degrees, 75)

# Print degree statistics

print(f"Minimum degree: {degree\_min}") print(f"Maximum degree: {degree\_max}") print(f"Median degree: {degree\_median}") print(f"First Quartile (Q1): {degree\_q1}") print(f"Third Quartile (Q3): {degree\_q3}")

# Classify nodes based on degree node\_colors = []

for node in G.nodes(): degree = G.degree(node) if degree == degree\_min:

node\_colors.append('blue') # Min degree elif degree == degree\_max:

node\_colors.append('red') # Max degree elif degree == degree\_median:

node\_colors.append('green') # Median degree elif degree <= degree\_q1:

node\_colors.append('purple') # First Quartile (Q1) elif degree >= degree\_q3:

node\_colors.append('orange') # Third Quartile (Q3) else:

node\_colors.append('gray') # Other degrees

# Visualize the network plt.figure(figsize=(10, 8))

pos = nx.spring\_layout(G) # Layout for node positioning

# Draw the graph with colored nodes based on degree

nx.draw(G, pos, node\_size=50, node\_color=node\_colors, with\_labels=False, edge\_color='lightgray')

# Add a title

plt.title('Network Visualization Based on Degree Statistics') plt.show()

import networkx as nx  
import matplotlib.pyplot as plt  
import numpy as np  
  
G=nx.erdos\_renyi\_graph(n=100,p=0.1)  
  
degree\_sequence=[G.degree(node) for node in G.nodes() ]  
  
max\_degree=np.max(degree\_sequence)  
min\_degree=np.min(degree\_sequence)  
median\_degree=np.median(degree\_sequence)  
q1=np.percentile(degree\_sequence,25)  
q3=np.percentile(degree\_sequence,75)  
  
print(f"Maximum Degree:{max\_degree}")  
print(f"Minimum Degree:{min\_degree}")  
print(f"Median Degree:{median\_degree}")  
print(f"First Quartile(Q1):{q1}")  
print(f"Thirs Quartile(Q3):{q3}")  
  
node\_colors=[]  
for degree in degree\_sequence:  
 if degree==max\_degree:  
 node\_colors.append('red')  
 elif degree==min\_degree:  
 node\_colors.append('blue')  
 elif degree<=q1:  
 node\_colors.append('green')  
 elif degree<=median\_degree:  
 node\_colors.append('yellow')  
 elif degree<=q3:  
 node\_colors.append('orange')  
 else:  
 node\_colors.append('purple')  
  
plt.figure(figsize=(10,8))  
pos=nx.spring\_layout(G,seed=42)  
nx.draw(G,pos,with\_labels=True,node\_size=300,node\_color=node\_colors,font\_size=10,font\_weight='bold',edge\_color='gray')  
  
import matplotlib.lines as mlines  
  
max\_label=mlines.Line2D([],[],color='red',marker='o',markersize=10,label=f"Max Degree({max\_degree})")  
min\_label=mlines.Line2D([],[],color='blue',marker='o',markersize=10,label=f"min Degree({min\_degree})")  
q1\_label=mlines.Line2D([],[],color='green',marker='o',markersize=10,label=f"Q1(<={q1})")  
median\_label=mlines.Line2D([],[],color='yellow',marker='o',markersize=10,label=f"Median(<={median\_degree})")  
q3\_label=mlines.Line2D([],[],color='orange',marker='o',markersize=10,label=f"Q3(<={q3})")  
above\_q3\_label=mlines.Line2D([],[],color='purple',marker='o',markersize=10,label=f"Above Q3")  
  
plt.legend(handles=[max\_label,min\_label,q1\_label,median\_label,q3\_label,above\_q3\_label],loc="upper left")  
  
plt.title("Network Visualization with Degree statistics coloring")  
plt.show()