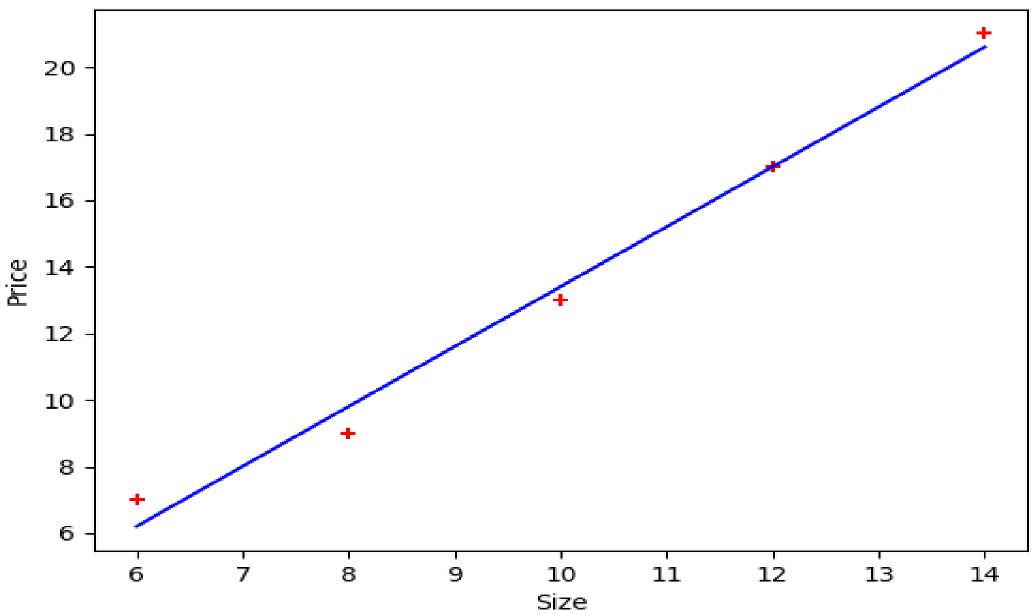
1.Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model

df = pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\pizza.csv")

plt.xlabel('Size')
plt.ylabel('Price')
plt.scatter(df['size'], df['price'], color='red', marker='+')

reg = linear_model.LinearRegression()
reg.fit(df[['size']], df['price'])

plt.plot(df['size'], reg.predict(df[['size']]), color='blue')
plt.show()

predicted_price = reg.predict(np.array([[15]]))
print("Predicted price for size 15 pizza:", predicted_price[0])
```

Output:

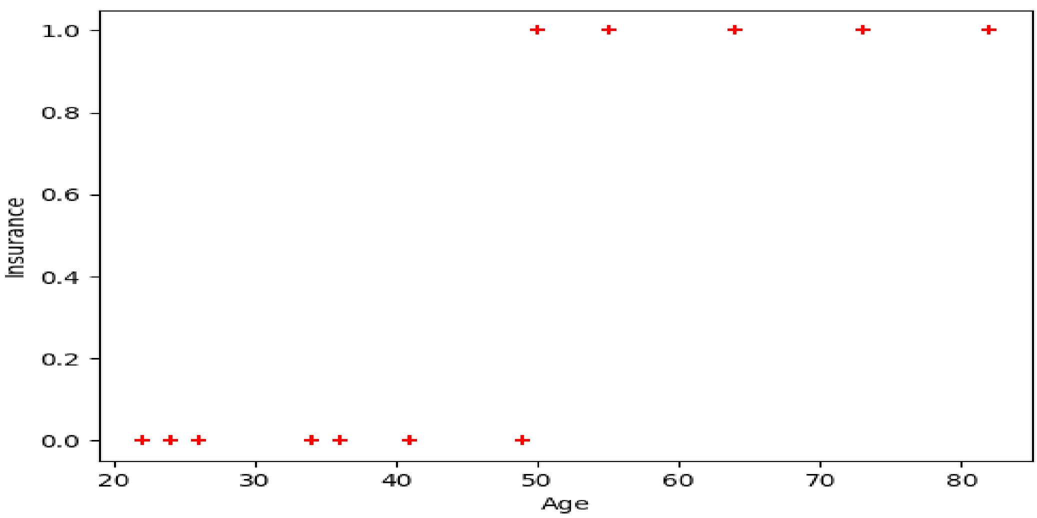


Predicted price for size 15 pizza: 22.4

2. Logistic Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

df = pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\insurance.csv")  #
Removed space in filename

plt.xlabel('Age')
plt.ylabel('Insurance')
plt.scatter(df['age'], df['insurance'], color='red', marker='+')
plt.show()

x_train, x_test, y_train, y_test = train_test_split(df[['age']], df['insurance'], test_size=0.2)

model = LogisticRegression()
model.fit(x_train, y_train)

inputdata = np.array([45, 23, 56, 66, 77, 88, 99, 12])
inputdata = inputdata.reshape(-1, 1)
predictions = model.predict(inputdata)
print("Predictions:",predictions)
```

Output:



Predictions: [0 0 1 1 1 1 1 0]

3. ID3 Algorithm

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data = pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\iris.csv")

column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
data.columns = column_names

print(data.isnull().sum())

X = data.drop('class', axis=1)
y = data['class']

print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"Training set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")

model = DecisionTreeClassifier(criterion='entropy', random_state=42)
model.fit(X_train, y_train)

train_accuracy = model.score(X_train, y_train)
test_accuracy = model.score(X_test, y_test)

print(f"Train Accuracy: {train_accuracy:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

new_sample = [1.3, 1.3, 4.5, 8.0]
print(f"Sample length: {len(new_sample)}")

prediction = model.predict([new_sample])
print(f"Predicted Class: {prediction[0]}")
```

Output:

```
sepal_length    0
sepal_width     0
```

petal_length   0

petal_width   0

class   0

dtype: int64

Features shape: (150, 4)

Target shape: (150,)

Training set size: 120

Test set size: 30

Train Accuracy: 1.0000

Test Accuracy: 1.0000

Sample length: 4

Predicted Class: Virginica

4.Navie Bayes

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
data = pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\iris.csv")
x = data.drop(["variety"], axis=1)
y = data["variety"]
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2,
random_state=42)
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(y_pred)
```

Output:

['Versicolor' 'Setosa' 'Virginica' 'Versicolor' 'Versicolor' 'Setosa'

'Versicolor' 'Virginica' 'Versicolor' 'Versicolor' 'Virginica' 'Setosa'

'Setosa' 'Setosa' 'Setosa' 'Versicolor' 'Virginica' 'Versicolor'

'Versicolor' 'Virginica' 'Setosa' 'Virginica' 'Setosa' 'Virginica'
'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Setosa' 'Setosa']

5. KNN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
df=pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\knn.csv")
x=df[['weight','height']].values
y=df['class'].values
le=LabelEncoder()
y_encoded=le.fit_transform(y)
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(x,y_encoded)
sample=[[23,56]] #Example height and weight
prediction=knn.predict(sample)
predicted_table=le.inverse_transform(prediction)[0]
print(f"Predicted status for {sample[0]} (height,weight):{predicted_table}")
```

Output:

Predicted status for [23, 56] (height,weight):underweight

6. K Means

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

df = pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\kmeans.csv")

x= df[['Height', 'Weight']]
Kmean = KMeans(n_clusters=3, random_state=30)
Kmean.fit(x)

x['cluster'] = Kmean.predict(x)

print(x)
```

Output:

	Height	Weight	cluster
0	185	72	0
1	170	56	1
2	168	60	1
3	179	68	0
4	182	72	0
5	188	77	2
6	189	71	2

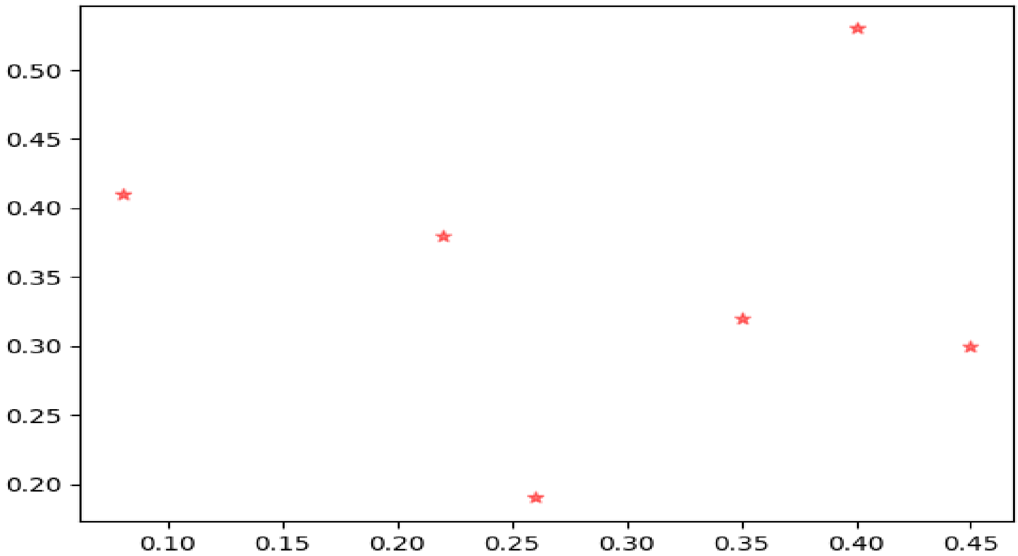
7. Hierarchical Clustering

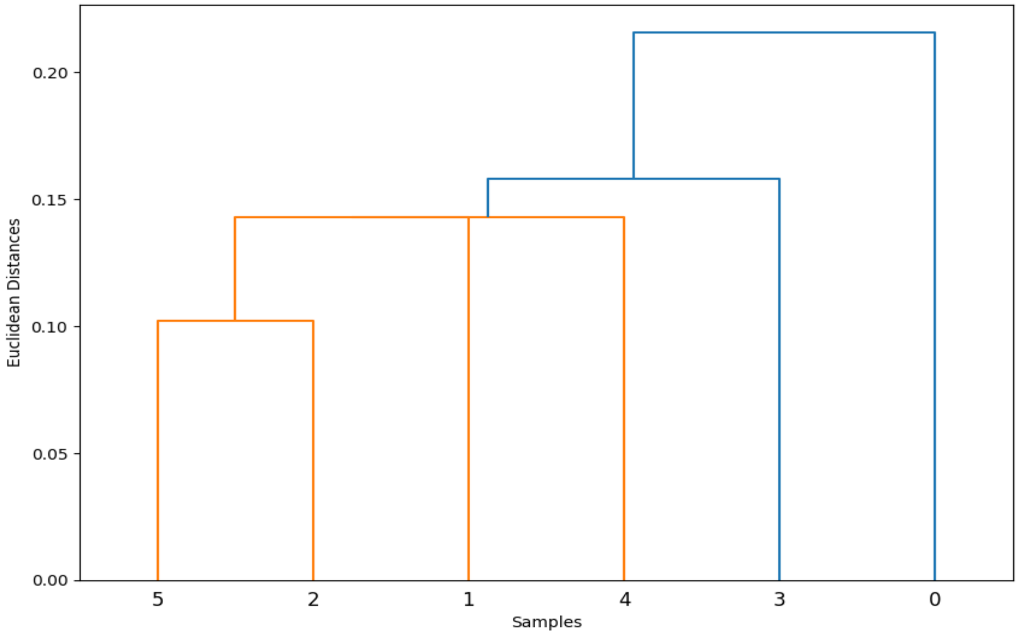
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
from sklearn.decomposition import PCA

data = pd.read_csv("C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data
Set\\hierarchial.csv")
print(data)

x = data['x']
y = data['y']
n = range(1, 8)

fig, ax = plt.subplots()
ax.scatter(x, y, marker='*', c='red', alpha=0.5)

linked = linkage(data[['x', 'y']], 'single')

plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.xlabel('Samples')
plt.ylabel('Euclidean Distances')
plt.show()

model = AgglomerativeClustering(n_clusters=5, metric='euclidean', linkage='single')
model.fit(data[['x', 'y']])
```

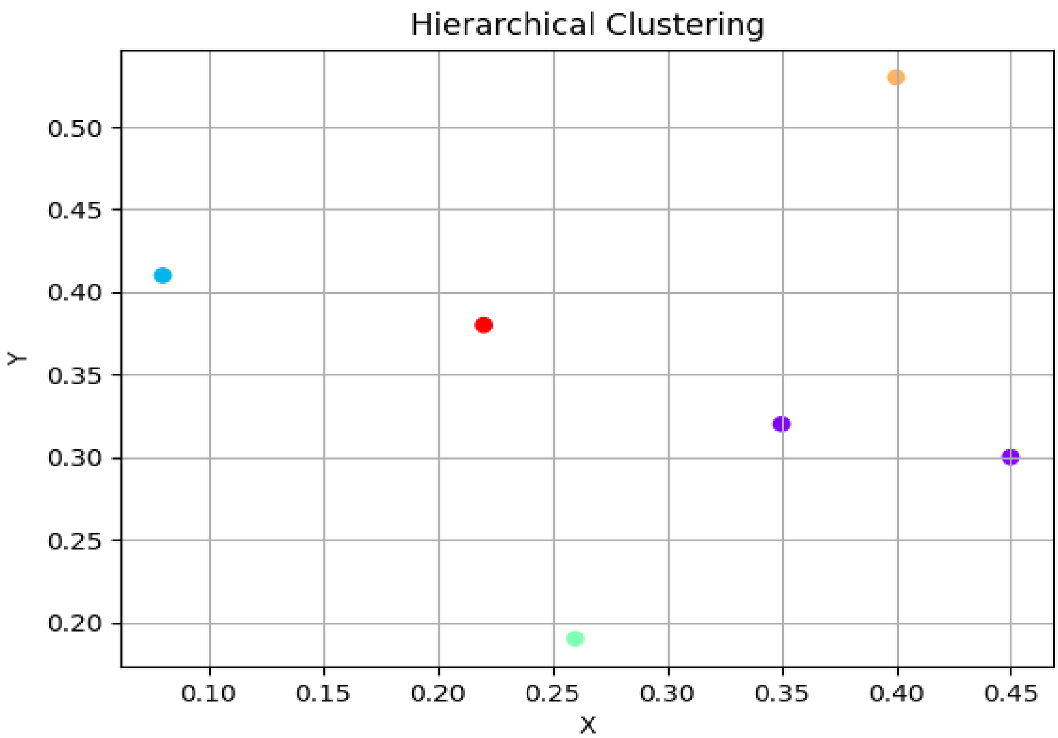
```
x = data['x']
y = data['y']
n = range(1, 8)

fig, ax = plt.subplots()
ax.scatter(x, y, c=model.labels_, cmap='rainbow')
plt.grid()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Hierarchical Clustering')
plt.show()
```

Output:

Unnamed: 0		x	y
0	p1	0.40	0.53
1	p2	0.22	0.38
2	p3	0.35	0.32
3	p4	0.26	0.19
4	p5	0.08	0.41
5	p6	0.45	0.30




0.20
0.15
0.10
0.05
0.00
Euclidean Distances
5
2
1
4
3
0
Samples

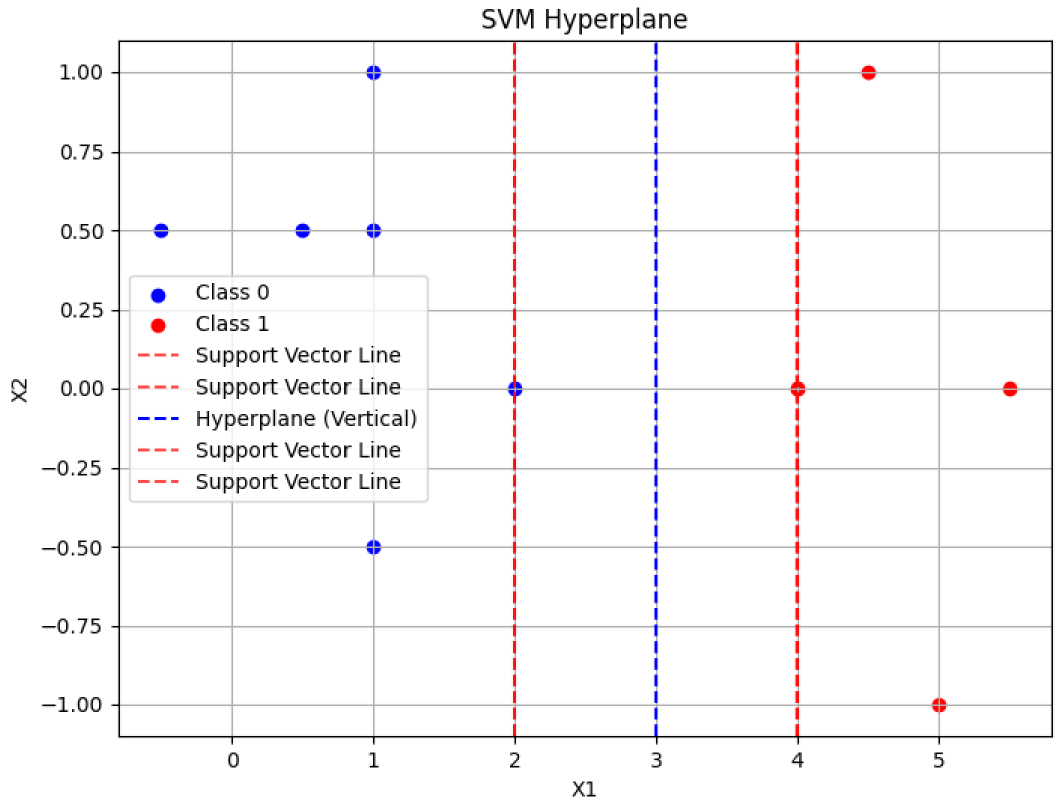

Hierarchical Clustering
0.50
0.45
0.40
0.35
0.30
0.25
0.20
Y
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
X

8. Support Vector Machine

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.ma.extras import unique
from sklearn.svm import SVC
x=np.array([(1,0.5),(1,1),(1,-0.5),(-0.5,0.5),(0.5,0.5),(2,0),(4,0),(4.5,1),(5,-1),(5.5,0)])
y=np.array([-1,-1,-1,-1,-1,-1,1,1,1,1])

x_coords=x[:,0]
y_coords=x[:,1]
plt.figure(figsize=(8,6))
plt.scatter(x_coords[y==-1],y_coords[y==-1],color='blue',label='Class 0')
plt.scatter(x_coords[y==1],y_coords[y==1],color='red',label='Class 1')

clf=SVC(kernel='linear',C=1.0)
clf.fit(x,y)

support_vectors=clf.support_vectors_
for sv in support_vectors:
   plt.axvline(x=sv[0],color='red',linestyle='--',alpha=0.7,label='Support Vector Line')

handles, labels=plt.gca().get_legend_handles_labels()
unique=dict(zip(labels,handles))
plt.legend(unique.values(),unique.keys())

w=clf.coef_[0]
b=clf.intercept_[0]
if np.isclose(w[1],0):
   x_hyperplane=-b/w[0]
   plt.axvline(x=x_hyperplane,color='blue',linestyle='--',label='Hyperplane (Vertical)')
else:
   x_vals=np.linspace(-1,7,200)
   y_vals=(w[0]*x_vals+b)/w[1]
   plt.plot(x_vals,y_vals,'k--',label='Hyperplane')

plt.scatter(clf.support_vectors_[:,0],clf.support_vectors_[:,1],s=120,facecolors='none',edgecol
ors='blue',label='Support Vectors')
   support_vectors=clf.support_vectors_
for sv in support_vectors:
   plt.axvline(x=sv[0],color='red',linestyle='--',alpha=0.7,label='Support Vector Line')

plt.xlabel("X1")
plt.ylabel("X2")
plt.title("SVM Hyperplane")
plt.legend()
```

```
plt.grid(True)
plt.show()
```

Output:



9. Random Forest

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import AgglomerativeClustering

x_real = np.array([
    [1.0,2.0],
    [1.1,2.1],
    [8.0,9.0]
])

n_samples, n_features = x_real.shape
```

```
x_synthetic=np.zeros_like(x_real)
for i in range(n_features):
    x_synthetic[:,i]=np.random.permutation(x_real[:,i])

x_combined=np.vstack([x_real,x_synthetic])
y_labels=np.array([1]*n_samples+[0]*n_samples)

rf=RandomForestClassifier(n_estimators=10,max_depth=3,random_state=42)
rf.fit(x_combined,y_labels)

leaf_indices=rf.apply(x_real)

proximity=np.zeros((n_samples,n_samples))
for tree_leaf in leaf_indices.T:
    for i in range(n_samples):
        for j in range(n_samples):
            if tree_leaf[i]==tree_leaf[j]:
                proximity[i,j]+=1

proximity/=rf.n_estimators

distance=1-proximity

clustering=AgglomerativeClustering(n_clusters=2,metric='precomputed',linkage='average')
labels=clustering.fit_predict(distance)

print("Real Data Points:")
print(x_real)

print("\nProximity Matrix:")
print(np.round(proximity,2))

print("\nDistance Matrix:")
print(np.round(distance,2))

print("\nCluster Labels:")
print(labels)
```

Output:

```
Real Data Points:
[[1.  2. ]
 [1.1 2.1]
 [8.  9. ]]
```

Proximity Matrix:

```
[[1.  0.4 0. ]
 [0.4 1.  0. ]
 [0.  0.  1. ]]
```

Distance Matrix:

```
[[0.  0.6 1. ]
 [0.6 0.  1. ]
 [1.  1.  0. ]]
```

Cluster Labels:

```
[0 0 1]
```

10. LOF

```
import numpy as np
from sklearn.neighbors import NearestNeighbors

data=np.array([1,2,2.5,3,3.5,10]).reshape(-1,1)
k=2

nbrs=NearestNeighbors(n_neighbors=k+1)
nbrs.fit(data)
distances, indices = nbrs.kneighbors(data)

lrd=[]
lof=[]

for i in range(len(data)):
    reach_dists=[]
    for j in range(1,k+1):
        neighbor_idx=indices[i][j]
        neighbor_k_dist=distances[neighbor_idx][k]
        actual_dist=distances[i][j]
        reach_dist=max(neighbor_k_dist,actual_dist)
        reach_dists.append(reach_dist)
    lrd_i=1/(np.mean(reach_dists))
    lrd.append(lrd_i)
```

```
for i in range(len(data)):
    lrd_ratios=[]
    for j in range(1,k+1):
        neighbor_idx=indices[i][j]
        lrd_ratios.append(lrd[neighbor_idx]/lrd[i])
    lof_i=np.mean(lrd_ratios)
    lof.append(lof_i)

print(f"{'Point':>6} | {'LRD':>8} | {'LOF':>8} | Outlier?")
print("-"*36)
for i in range(len(data)):
    outlier='YES' if lof[i]>1.5 else 'NO'
    print(f"{data[i][0]:>6} |    {lrd[i]:>2.2f} |  {lof[i]:>8.2f} | {outlier}")
```

Output:

```
Point |      LRD |      LOF | Outlier?
------------------------------------
   1.0 |    0.80 |      1.46 | NO
   2.0 |    1.00 |      1.07 | NO
   2.5 |    1.33 |      0.88 | NO
   3.0 |    1.33 |      1.00 | NO
   3.5 |    1.33 |      1.00 | NO
  10.0 |    0.15 |      9.00 | YES
```

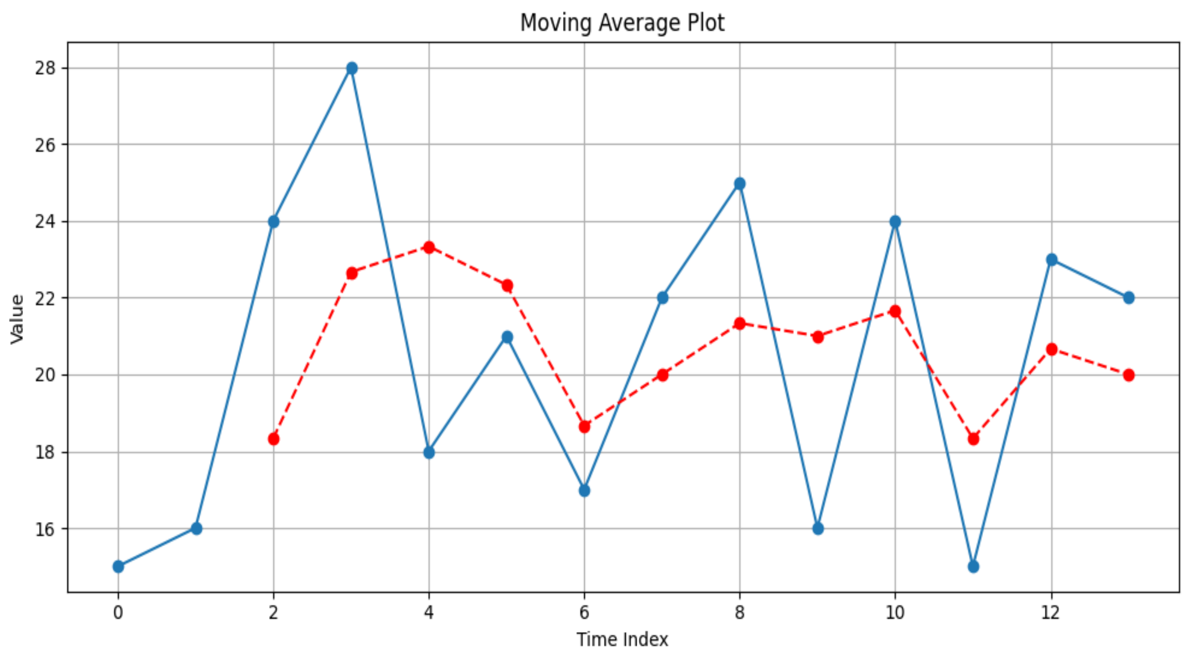
11. Moving Average

```
import matplotlib.pyplot as plt

input=[15,16,24,28,18,21,17,22,25,16,24,15,23,22]
def moving_avg(input, window_size):
    result=[]
    moving_sum=sum(input[:window_size])
    result.append(moving_sum/window_size)
    for i in range(len(input)-window_size):
        moving_sum=moving_sum+(input[i+window_size]-input[i])
        result.append(moving_sum/window_size)
    return result
window_size=3
print(moving_avg(input,window_size))
```

```
ma=moving_avg(input,window_size)

x_ma=list(range(window_size-1,len(input)))
plt.figure(figsize=(10,5))
plt.plot(input,label='Original Data',marker='o')

plt.plot(x_ma,ma,label='f{window_size}-Point Moving
Average',color='red',marker='o',linestyle='--')
plt.title('Moving Average Plot')
plt.xlabel('Time Index')
plt.ylabel('Value')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

[18.333333333333332, 22.666666666666668, 23.333333333333332, 22.333333333333332, 18.666666666666668, 20.0, 21.333333333333332, 21.0, 21.666666666666668, 18.333333333333332, 20.666666666666668, 20.0]



12. ARIMA

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
file_path="C:\\Users\\ASUS\\OneDrive\\Documents\\ML Data Set\\airline_passengers.csv"
df=pd.read_csv(file_path,parse_dates=['Month'],index_col='Month')
df.index.freq='MS'

model=ARIMA(df['Passengers'],order=(2,1,2))
model_fit=model.fit()
forecast=model_fit.forecast(steps=1).iloc[0]
print(f"Forecasted next value using ARIMA:{forecast:.2f}")

plt.plot(df['Passengers'],label='Original Data')
plt.plot(model_fit.fittedvalues,label='Fitted values')
plt.legend()
plt.title('ARIMA Forecasting on Airline Passengers')
plt.show()
```

Output:

Forecasted next value using ARIMA:439.85