## Apriori 1

```
import pandas as pd

import mlxtend

from mlxtend.frequent_patterns import apriori, fpgrowth,association_rules

# dataset

data = pd.DataFrame({

    'A': [1, 0, 1, 0],

    'B': [0, 1, 1, 1],

    'C': [1, 1, 1, 0],

    'E': [0, 1, 1, 1]

})

# Convert to Boolean

data = data.astype(bool)

# Apply Apriori Algorithm

frequent_itemsets = apriori(data, min_support=0.5, use_colnames=True)

print(frequent_itemsets)

# Generate Rules

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)

print(rules[["antecedents", "consequents", "support", "confidence"]])

rules = rules[rules["support"] >= 0.6]

print(rules[["antecedents", "consequents", "support", "confidence"]])

print(rules[["antecedents", "consequents", "support", "confidence"]])
```

# fpgrowth 2

```
import pandas as pd

import mlxtend

from mlxtend.frequent_patterns import apriori, fpgrowth,association_rules

# dataset

data = pd.DataFrame({

    'A': [1, 0, 1, 0],

    'B': [0, 1, 1, 1],

    'C': [1, 1, 1, 0],

    'E': [0, 1, 1, 1]

})

# Convert to Boolean

data = data.astype(bool)

# Apply fpgrowth Algorithm

frequent_itemsets = fpgrowth (data, min_support=0.5, use_colnames=True)

print(frequent_itemsets)

# Generate Rules

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)

print(rules[["antecedents", "consequents", "support", "confidence"]])

rules = rules[rules["support"] >= 0.6]

print(rules[["antecedents", "consequents", "support", "confidence"]])

print(rules[["antecedents", "consequents", "support", "confidence"]])
```

## Cluster 3

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

# Data points

data = np.array([

    [0, 0],

    [0, 5],

    [5, 0],

    [5, 5],

    [2, 2],

    [3, 3],

    [1, 4],

    [4, 1]

])

# K-Means clustering with k=2

kmeans = KMeans(n_clusters=2, n_init=10, random_state=0)

kmeans.fit(data)

labels = kmeans.labels_

centroids = kmeans.cluster_centers_

# Define manual colors

colors = ['cyan', 'orange']

# Plot points with color per label

for i in range(len(data)):

   plt.scatter(data[i, 0], data[i, 1], color=colors[labels[i]], edgecolors='black', s=100)

# Plot centroids

plt.scatter(centroids[:, 0], centroids[:, 1], color='red', marker='X', s=200)

plt.title("K-Means Clustering (k=2)")

plt.grid(True)
```

```
plt.show()
```

## logistic 4

```
import numpy as np

from sklearn.linear_model import LogisticRegression

# Data: X = hours of study, y = pass/fail (0/1)

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)

y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Model

model = LogisticRegression()

model.fit(X, y)

# Prediction

print("Prediction for 6 hours:", "Pass" if model.predict([[6]]) == 1 else "Fail")
```

## linear 5

```
import numpy as np

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

# Input Data (hours of study and corresponding scores)

X = np.array([1, 2, 3, 4, 5,6,7,8]).reshape(-1, 1)  # employee experience years

y = np.array([40,43,49,54,58,65,70,75])  # salary in 1000$

# Create a linear regression model

model = LinearRegression()

# Fit the model

model.fit(X, y)

# Get the slope (coefficient) and intercept of the regression line

slope = model.coef_[0]

intercept = model.intercept_

# Print the linear equation
```

```python
print(f"Linear Equation: y = {intercept:.2f} + {slope:.2f}x")

# Predictions for new data points

new_data = np.array([[6.5], [9], [10]])  # New data for prediction

predictions = model.predict(new_data)

# Output the predictions

for i, prediction in enumerate(predictions, start=1):

    print(f"Prediction for {new_data[i-1][0]} experience years is : {prediction:.2f}")

# Evaluate the model

y_pred = model.predict(X)

mse = mean_squared_error(y, y_pred)  # Mean Squared Error

print(f"Mean Squared Error: {mse:.2f}")
```