Practical 1: To handle and preprocess missing data in a dataset using Pandas library functions such as detecting, removing, and imputing missing values to improve data quality for further analysis or machine learning tasks.
Note: Take appropriate dataset for implementation.

id	age	salary	city
1	24	45000	Mumbai
2	27	54000	Delhi
3	NaN	50000	Bangalore
4	22	NaN	Mumbai
5	25	61000	NaN
6	29	58000	Delhi
7	31	NaN	Pune
8	NaN	47000	Delhi
9	23	46000	Mumbai
10	26	NaN	Hyderabad
11	NaN	52000	Pune
12	30	60000	Mumbai
13	28	58000	NaN
14	22	NaN	Chennai
15	NaN	49000	Kolkata
16	27	53000	Pune
17	25	NaN	Mumbai
18	26	52000	Delhi
19	NaN	56000	Bangalore
20	24	48000	NaN

Code:

```
import pandas as pd


df = pd.read_csv("data.csv")

df
```

```
print("\nMissing Values Count:\n", df.isnull().sum())

df_removed = df.dropna()
print("\nAfter Removing Missing Values:\n", df_removed)

df['age'] = df['age'].fillna(df['age'].mean())
df['salary'] = df['salary'].fillna(df['salary'].mean())

df['city'] = df['city'].fillna(df['city'].mode()[0])

print("\nAfter Imputation:\n", df)

print("\nMissing Values After Imputation:\n", df.isnull().sum())
```

Practical 2: To convert categorical data into numerical values using appropriate encoding techniques such as Label Encoding, One-Hot Encoding, and Ordinal Encoding, enabling machine learning models to process and analyze categorical features effectively. (Take a suitable dataset as input as we have performed in classroom teaching)

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, OrdinalEncoder
data = { 'name': ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T'],
```

```
'gender': ['Male', 'Female', 'Male', 'Female', 'Male',
        'Female', 'Male', 'Male', 'Female', 'Female',
        'Male', 'Female', 'Male', 'Female', 'Male',
        'Female', 'Male', 'Female', 'Male', 'Female'],

'city': ['Mumbai', 'Delhi', 'Pune', 'Chennai', 'Delhi',
      'Mumbai', 'Pune', 'Delhi', 'Chennai', 'Mumbai',
      'Pune', 'Delhi', 'Mumbai', 'Chennai', 'Delhi',
      'Pune', 'Mumbai', 'Delhi', 'Pune', 'Chennai'],

'education': ['School', 'Diploma', 'Graduate', 'Postgraduate',
          'School', 'Graduate', 'Diploma', 'School',
          'Graduate', 'Postgraduate', 'School', 'Diploma',
          'Graduate', 'Postgraduate', 'School', 'Graduate',
          'Diploma', 'Postgraduate', 'Graduate', 'School']


}

df = pd.DataFrame(data) print("Original Data:\n", df)



le = LabelEncoder() df['gender_encoded'] = le.fit_transform(df['gender'])

 print("\nAfter Label Encoding (gender):\n", df)



ohe = pd.get_dummies(df['city'], prefix='city')

 df_ohe = pd.concat([df, ohe], axis=1)

print("\nAfter One-Hot Encoding (city):\n", df_ohe)



education_order = [['School', 'Diploma', 'Graduate', 'Postgraduate']]

ordinal = OrdinalEncoder(categories=education_order)

df['education_encoded'] = ordinal.fit_transform(df[['education']])
```

```
print("\nAfter Ordinal Encoding (education):\n", df)
```

Practical 3: To implement the Decision Tree algorithm for classifying data by learning decision rules from features and representing them in a hierarchical tree structure for accurate prediction and interpretation.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd from sklearn.model_selection

import train_test_split from sklearn.tree

import DecisionTreeClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score

data = { 'age': [22, 25, 27, 30, 35, 40, 45, 50, 23, 29, 31, 38, 42, 48, 26, 33, 37, 41, 46, 52],

'income': ['Low', 'Medium', 'Medium', 'High', 'High', 'Medium', 'Low', 'Medium', 'Low', 'High',
    'Low', 'High', 'Medium', 'High', 'Medium', 'Low', 'High', 'Medium', 'Low', 'High'],

'student': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
    'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'],

'buylaptop': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
    'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']


}
```

```
df = pd.DataFrame(data) print("Dataset:\n", df)

le = LabelEncoder() df['income'] = le.fit_transform(df['income'])

df['student'] = le.fit_transform(df['student'])

df['buylaptop'] = le.fit_transform(df['buylaptop'])

print("\nAfter Encoding:\n", df)

X = df[['age', 'income', 'student']] y = df['buylaptop']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )

print("\nTraining Data:\n", X_train)

print("\nTesting Data:\n", X_test)

model = DecisionTreeClassifier() model.fit(X_train, y_train)

y_pred = model.predict(X_test)

 print("\nPredictions:", y_pred)

accuracy = accuracy_score(y_test, y_pred)

print("\nAccuracy:", accuracy)

print("\nFeature Importance:")

for name, score in zip(X.columns, model.feature_importances_):

print(name, ":", score)
```

Practical 4: To implement the Random Forest algorithm for accurate classification or regression by constructing an ensemble of decision trees and aggregating their outputs to improve prediction performance and reduce overfitting.

Code:

```
import pandas as pd from sklearn.model_selection

import train_test_split from sklearn.preprocessing

import LabelEncoder from sklearn.ensemble

import RandomForestClassifier

from sklearn.metrics import accuracy_score

data = { 'age': [22, 25, 27, 30, 35, 40, 45, 50, 23, 29, 31, 38, 42, 48, 26, 33, 37, 41, 46, 52],

'income': ['Low', 'Medium', 'Medium', 'High', 'High', 'Medium', 'Low', 'Medium', 'Low', 'High',
        'Low', 'High', 'Medium', 'High', 'Medium', 'Low', 'High', 'Medium', 'Low', 'High'],

'student': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
         'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'],

'buylaptop': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
          'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']


}

df = pd.DataFrame(data) print("Dataset:\n", df)

le = LabelEncoder() df['income'] = le.fit_transform(df['income'])

df['student'] = le.fit_transform(df['student'])

df['buylaptop'] = le.fit_transform(df['buylaptop'])

print("\nAfter Encoding:\n", df)

X = df[['age', 'income', 'student']] y = df['buylaptop']
```

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )

model = RandomForestClassifier(n_estimators=10, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nPredictions:", y_pred)

accuracy = accuracy_score(y_test, y_pred) print("\nAccuracy:", accuracy)

print("\nFeature Importance:")

for name, score in zip(X.columns, model.feature_importances_):

print(name, ":", score)
```

Practical 5: To implement the K-Nearest Neighbors (KNN) algorithm for classifying data samples based on the majority class of their nearest neighbors using a distance-based similarity measure.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

 from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score

data = { 'age': [22, 25, 27, 30, 35, 40, 45, 50, 23, 29, 31, 38, 42, 48, 26, 33, 37, 41, 46, 52],

'income': ['Low', 'Medium', 'Medium', 'High', 'High', 'Medium', 'Low', 'Medium', 'Low', 'High',
        'Low', 'High', 'Medium', 'High', 'Medium', 'Low', 'High', 'Medium', 'Low', 'High'],

'student': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
         'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'],

'buylaptop': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
          'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']


}

df = pd.DataFrame(data)

print("Dataset:\n", df)

le = LabelEncoder()

df['income'] = le.fit_transform(df['income'])

df['student'] = le.fit_transform(df['student'])

df['buylaptop'] = le.fit_transform(df['buylaptop'])

print("\nAfter Encoding:\n", df)

X = df[['age', 'income', 'student']]

y = df['buylaptop']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )

model = KNeighborsClassifier(n_neighbors=3)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nPredictions:", y_pred)
```

```
accuracy = accuracy_score(y_test, y_pred)

print("\nAccuracy:", accuracy)
```

Practical 6: To implement the Support Vector Machine (SVM) algorithm for classifying data by identifying the optimal hyper plane that maximizes the margin between different classes.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

data = { 'age': [22, 25, 27, 30, 35, 40, 45, 50, 23, 29, 31, 38, 42, 48, 26, 33, 37, 41, 46, 52],

'income': ['Low', 'Medium', 'Medium', 'High', 'High', 'Medium', 'Low', 'Medium', 'Low', 'High',
        'Low', 'High', 'Medium', 'High', 'Medium', 'Low', 'High', 'Medium', 'Low', 'High'],

'student': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
         'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'],

'buylaptop': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
          'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']


}
```

```
df = pd.DataFrame(data)

print("Dataset:\n", df)

le = LabelEncoder()

df['income'] = le.fit_transform(df['income'])

df['student'] = le.fit_transform(df['student'])

df['buylaptop'] = le.fit_transform(df['buylaptop'])

print("\nAfter Encoding:\n", df)

X = df[['age', 'income', 'student']] y = df['buylaptop']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )

model = SVC(kernel='linear')

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

 print("\nPredictions:", y_pred)

accuracy = accuracy_score(y_test, y_pred) print("\nAccuracy:", accuracy)
```

Practical 7: To implement the Naïve Bayes Classifier algorithm for predicting the class of data samples by applying Bayes' Theorem with the assumption of conditional independence among features.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd from sklearn.model_selection

import train_test_split from sklearn.preprocessing

import LabelEncoder from sklearn.naive_bayes

import GaussianNB from sklearn.metrics

import accuracy_score


data = { 'age': [22, 25, 27, 30, 35, 40, 45, 50, 23, 29, 31, 38, 42, 48, 26, 33, 37, 41, 46, 52],

'income': ['Low', 'Medium', 'Medium', 'High', 'High', 'Medium', 'Low', 'Medium', 'Low', 'High',
    'Low', 'High', 'Medium', 'High', 'Medium', 'Low', 'High', 'Medium', 'Low', 'High'],

'student': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
    'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'],

'buylaptop': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
    'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']


}


df = pd.DataFrame(data)

print("Dataset:\n", df)

le = LabelEncoder() df['income'] = le.fit_transform(df['income'])

df['student'] = le.fit_transform(df['student'])

df['buylaptop'] = le.fit_transform(df['buylaptop'])

print("\nAfter Encoding:\n", df)

X = df[['age', 'income', 'student']]

y = df['buylaptop']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )
```

```
model = GaussianNB() model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nPredictions:", y_pred)

accuracy = accuracy_score(y_test, y_pred)

print("\nAccuracy:", accuracy)
```

Practical 8: To implement the Simple Linear Regression algorithm to predict a continuous output variable based on a single input feature by modeling their linear relationship.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

data = { 'area': [2600, 3000, 3200, 3600, 4000, 2200, 2800, 3100, 3300, 3500, 2500, 2700,
3400, 3800, 3900, 2950, 3050, 3150, 3250, 3450],

'price': [550000, 565000, 610000, 680000, 725000, 480000, 540000, 590000,
    630000, 660000, 510000, 545000, 615000, 700000, 715000,
```

```
        560000, 575000, 600000, 620000, 650000]

}

df = pd.DataFrame(data)

 print("Dataset:\n", df)

X = df[['area']] # independent variable y = df['price']

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)

print("\nPredictions:\n", y_pred)

print("\nModel Performance:")

print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))

print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))

print("R² Score:", r2_score(y_test, y_pred))

print("\nSlope (Coefficient):", model.coef_[0])

print("Intercept:", model.intercept_)
```

Practical 9: To implement the Multiple Linear Regression algorithm to predict a continuous output variable based on multiple input features by modeling their linear relationship.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

 from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


data = { 'area': [2600, 3000, 3200, 3600, 4000, 2200, 2800, 3100, 3300, 3500, 2500, 2700,
3400, 3800, 3900, 2950, 3050, 3150, 3250, 3450],

'bedrooms':  [3, 4, 3, 5, 4, 2, 3, 4, 3, 4,
          3, 2, 4, 5, 4, 3, 3, 4, 3, 4],

'age':      [10, 8, 6, 5, 4, 12, 11, 7, 6, 5,
          9, 10, 5, 3, 4, 8, 7, 6, 5, 5],

'price':    [550000, 565000, 610000, 680000, 725000, 480000, 540000, 590000,
          630000, 660000, 510000, 545000, 615000, 700000, 715000,
          560000, 575000, 600000, 620000, 650000]


}

df = pd.DataFrame(data)

print("Dataset:\n", df)

X = df[['area', 'bedrooms', 'age']]

y = df['price']

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)

print("\nPredicted Prices:\n", y_pred)

print("\nModel Performance:")

print("MSE:", mean_squared_error(y_test, y_pred))

print("MAE:", mean_absolute_error(y_test, y_pred))

print("R² Score:", r2_score(y_test, y_pred))

print("\nCoefficients (slopes):", model.coef_)

print("Intercept:", model.intercept_)
```

Practical 10: To implement the Logistic Regression algorithm for classifying data into two categories by modeling the probability of a given input belonging to a specific class using the sigmoid function.
Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

data = { 'age': [22, 25, 27, 30, 35, 40, 45, 50, 23, 29, 31, 38, 42, 48, 26, 33, 37, 41, 46, 52],
```

```
'income': ['Low', 'Medium', 'Medium', 'High', 'High', 'Medium', 'Low', 'Medium', 'Low', 'High',
         'Low', 'High', 'Medium', 'High', 'Medium', 'Low', 'High', 'Medium', 'Low', 'High'],

'student': ['Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
          'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No'],

'buylaptop': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
            'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']


}

df = pd.DataFrame(data)

print("Dataset:\n", df)

le = LabelEncoder()

df['income'] = le.fit_transform(df['income'])

df['student'] = le.fit_transform(df['student'])

 df['buylaptop'] = le.fit_transform(df['buylaptop'])

print("\nAfter Encoding:\n", df)

X = df[['age', 'income', 'student']]

y = df['buylaptop']

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=42 )

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nPredictions:", y_pred)

accuracy = accuracy_score(y_test, y_pred)

print("\nAccuracy:", accuracy)
```

Practical 11: To implement the K-Means clustering algorithm to group similar data points into clusters by minimizing the dissimilarity between points and their representative centroids.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd from sklearn.cluster import KMeans

data = { 'X': [1, 2, 3, 4, 5, 6, 7, 8, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 1.5, 2.2, 3.8, 4.8, 5.8],

'Y': [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 1.2, 2.2,
    3.2, 4.2, 4.8, 5.0, 5.5, 1.3, 1.6, 2.8, 3.8, 4.6]


}

df = pd.DataFrame(data)

 print("Dataset:\n", df)

kmeans = KMeans(n_clusters=3, random_state=42) df['cluster'] =
kmeans.fit_predict(df[['X', 'Y']])

print("\nCluster Labels:\n", df)

print("\nCluster Centers:\n", kmeans.cluster_centers_)
```

Practical 12: To implement the K-Medoids clustering algorithm to group similar data points into clusters by minimizing the dissimilarity between points and their representative medoids.

Note: Take appropriate dataset for implementation.

Code:

```
import pandas as pd

from sklearn_extra.cluster import KMedoids

data = { 'X': [2, 3, 4, 5, 6, 7, 8, 9, 3.2, 4.1, 5.3, 6.2, 7.1, 8.4, 2.3, 3.5, 4.7, 5.8, 6.6, 7.9],

'Y': [1, 1.8, 2.5, 3, 3.6, 4.2, 4.9, 5.4, 1.5, 2.7,
    3.3, 4.1, 4.9, 5.6, 1.2, 1.9, 2.9, 3.7, 4.4, 5.1]


}

df = pd.DataFrame(data)

print("Dataset:\n", df)

kmedoids = KMedoids(n_clusters=3, random_state=42)

df['cluster'] = kmedoids.fit_predict(df[['X', 'Y']])

print("\nCluster Labels:\n", df)

print("\nMedoid Centers (Actual Data Points Used As Centers):\n",
kmedoids.cluster_centers_)
```

Practical 13: To implement the Hierarchical Clustering algorithm to group similar data points into clusters based on their distance/similarity, and visualize the clustering structure using a dendrogram.
You can take appropriate dataset as well as per the classroom practical's.

Point	X	Y

P1	2	3
P2	3	4
P3	5	6
P4	8	8

Code:

```
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

data = {
   'X': [2, 3, 5, 8],
   'Y': [3, 4, 6, 8]
}

df = pd.DataFrame(data, index=['P1', 'P2', 'P3', 'P4'])
print("Dataset:\n", df)
linked = linkage(df, method='single')
plt.figure(figsize=(6, 4))
dendrogram(linked, labels=df.index.tolist())
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Points")
plt.ylabel("Distance")
plt.show()
```

Practical 14: To implement the Apriori algorithm for discovering frequent itemsets and generating association rules from a transactional dataset.

Example Dataset (Sample Transactions):

Transaction ID	**Items Purchased**
T1	Bread, Milk
T2	Bread, Diaper, Beer, Eggs
T3	Milk, Diaper, Beer, Coke
T4	Bread, Milk, Diaper, Beer
T5	Bread, Milk, Diaper, Coke

Code:

```
import pandas as pd

data = { 'TID': range(1, 21), 'Items': [ "Bread, Milk", "Bread, Diaper, Beer, Eggs", "Milk, Diaper, Beer, Coke", "Bread, Milk, Diaper, Beer", "Bread, Milk, Diaper, Coke", "Milk, Eggs", "Bread, Eggs", "Beer, Chips", "Milk, Chips", "Bread, Milk, Chips", "Coke, Chips", "Diaper, Beer", "Bread, Beer", "Milk, Diaper", "Bread, Coke", "Milk, Bread, Diaper", "Milk, Beer", "Bread, Diaper, Chips", "Beer, Coke", "Bread, Milk, Eggs" ] }

df = pd.DataFrame(data) print("Original Dataset:\n")

print(df)
```

```
df['Items'] = df['Items'].apply(lambda x: x.split(', '))

print("\nConverted to List Format:\n")

print(df)

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder() te_array = te.fit(df['Items']).transform(df['Items'])

df_encoded = pd.DataFrame(te_array, columns=te.columns_)

print("\nOne-Hot Encoded Data:\n") print(df_encoded)

from mlxtend.frequent_patterns import apriori

frequent_itemsets = apriori(df_encoded, min_support=0.3,  use_colnames=True)

print("\nFrequent Itemsets (Support >= 0.3):\n")

print(frequent_itemsets)

from mlxtend.frequent_patterns import association_rules

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
print("\nAssociation Rules:\n")

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