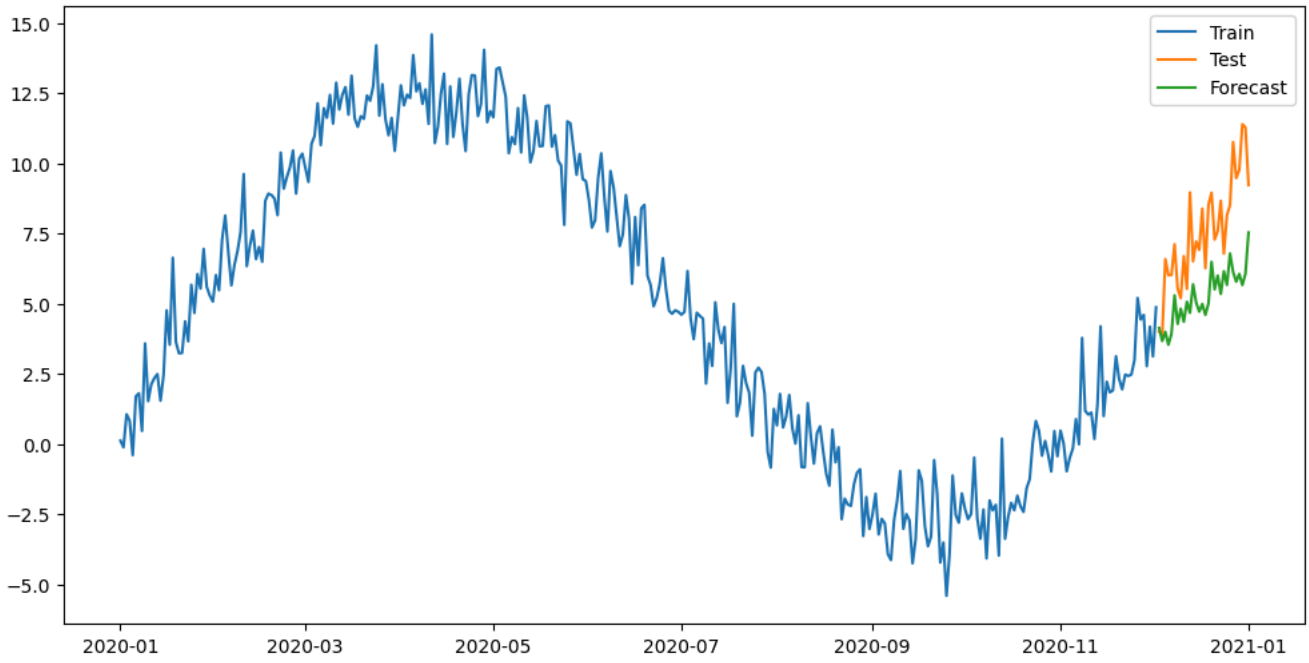
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from textblob import TextBlob
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
date_rng = pd.date_range(start='1/1/2020', end='1/01/2021', freq='D')
df_time_series = pd.DataFrame(date_rng, columns=['date'])
df_time_series['data'] = np.random.randn(len(df_time_series))
# Adding trend and seasonality
df_time_series['data'] = df_time_series['data'] + np.linspace(0, 10, len(df_time_series))
df_time_series['data'] += 10 * np.sin(2 * np.pi * df_time_series.index / 365)
df_time_series.set_index('date', inplace=True)
# Splitting the data into train and test sets
train = df_time_series.iloc[:-30]
test = df_time_series.iloc[-30:]
# Fitting the ARIMA model
model = SARIMAX(train['data'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
model_fit = model.fit(disp=False)
# Forecasting
forecast = model_fit.forecast(steps=len(test))
test['forecast'] = forecast.values
# Plotting
plt.figure(figsize=(12, 6))
plt.plot(train['data'], label='Train')
plt.plot(test['data'], label='Test')
plt.plot(test['forecast'], label='Forecast')
plt.legend()
plt.show()
# Sentiment Analysis or Text Mining on Unstructured Data
# Sample text data
data = {'text': [
    "I love this product! It's amazing.",
    "This is the worst service I've ever experienced.",
    "Pretty good, could be better.",
    "I'm extremely happy with the results.",
    "Not satisfied, it's very disappointing."
]}
df_text = pd.DataFrame(data)
# Function to calculate sentiment
def get_sentiment(text):
    blob = TextBlob(text)
    return blob.sentiment.polarity
# Applying sentiment analysis
df_text['sentiment'] = df_text['text'].apply(get_sentiment)
print(df_text)
# Clustering or Classification Techniques for Segmentation and Pattern Recognition
# Generating sample data
data = {
    'feature1': np.random.rand(100),
    'feature2': np.random.rand(100)
}
df_clustering = pd.DataFrame(data)
# Applying K-Means clustering
kmeans = KMeans(n_clusters=3)
df_clustering['cluster'] = kmeans.fit_predict(df_clustering)
# Plotting the clusters
plt.figure(figsize=(8, 6))
plt.scatter(df_clustering['feature1'], df_clustering['feature2'], c=df_clustering['cluster'], cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-Means Clustering')
plt.show()
# Generating a binary target variable for classification
df_clustering['target'] = (df_clustering['feature1'] + df_clustering['feature2'] > 1).astype(int)
# Splitting the data
X = df_clustering[['feature1', 'feature2']]
y = df_clustering['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Applying Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)
# Predictions
y_pred = model.predict(X_test)
# Evaluation
print(classification_report(y_test, y_pred))
```

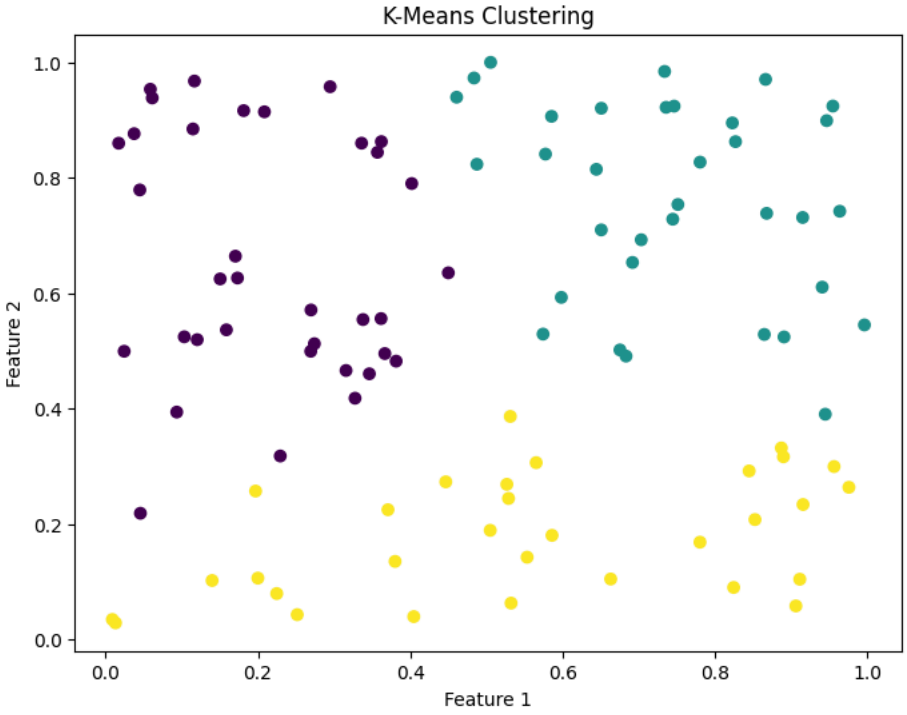
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
<ipython-input-1-71474e607760>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  test['forecast'] = forecast.values
```



	text	sentiment
0	I love this product! It's amazing.	0.612500
1	This is the worst service I've ever experienced.	-0.100000
2	Pretty good, could be better.	0.483333
3	I'm extremely happy with the results.	0.800000
4	Not satisfied, it's very disappointing.	-0.515000

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```



	precision	recall	f1-score	support
0	0.88	1.00	0.93	7
1	1.00	0.92	0.96	13
accuracy			0.95	20
macro avg	0.94	0.96	0.95	20
weighted avg	0.96	0.95	0.95	20