```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from datetime import datetime
import math

train = pd.read_csv(r'C:\Users\kaler\Desktop\Untitled Folder 1\
train.csv')
features = pd.read_csv(r'C:\Users\kaler\Desktop\Untitled Folder 1\
features.csv')
stores = pd.read_csv(r'C:\Users\kaler\Desktop\Untitled Folder 1\
stores.csv')

train.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

```
stores.head()
```

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

```
train.describe()
```

	Store	Dept	Weekly_Sales
count	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123
std	12.785297	30.492054	22711.183519
min	1.000000	1.000000	-4988.940000
25%	11.000000	18.000000	2079.650000
50%	22.000000	37.000000	7612.030000
75%	33.000000	74.000000	20205.852500
max	45.000000	99.000000	693099.360000

```
features.describe()
```

```
             Store  Temperature   Fuel_Price      MarkDown1
MarkDown2  \
```

```
count  8190.000000  8190.000000  8190.000000    4032.000000
2921.000000
mean     23.000000    59.356198     3.405992    7032.371786
3384.176594
std      12.987966    18.678607     0.431337    9262.747448
8793.583016
min       1.000000    -7.290000     2.472000   -2781.450000    -
265.760000
25%      12.000000    45.902500     3.041000    1577.532500
68.880000
50%      23.000000    60.710000     3.513000    4743.580000
364.570000
75%      34.000000    73.880000     3.743000    8923.310000
2153.350000
max      45.000000   101.950000     4.468000  103184.980000
104519.540000

           MarkDown3     MarkDown4      MarkDown5          CPI
Unemployment
count    3613.000000   3464.000000    4050.000000  7605.000000
7605.000000
mean     1760.100180   3292.935886    4132.216422   172.460809
7.826821
std     11276.462208   6792.329861   13086.690278    39.738346
1.877259
min      -179.260000      0.220000    -185.170000   126.064000
3.684000
25%         6.600000    304.687500    1440.827500   132.364839
6.634000
50%        36.260000   1176.425000    2727.135000   182.764003
7.806000
75%       163.150000   3310.007500    4832.555000   213.932412
8.567000
max    149483.310000  67474.850000  771448.100000   228.976456
14.313000
```

```
stores.describe()
```

```
           Store           Size
count  45.000000      45.000000
mean   23.000000  130287.600000
std    13.133926   63825.271991
min     1.000000   34875.000000
25%    12.000000   70713.000000
50%    23.000000  126512.000000
75%    34.000000  202307.000000
max    45.000000  219622.000000
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Store         421570 non-null  int64  
 1   Dept          421570 non-null  int64  
 2   Date          421570 non-null  object 
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  bool   
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
```

```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         8190 non-null   int64  
 1   Date          8190 non-null   object 
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 non-null   float64
 11  IsHoliday     8190 non-null   bool   
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
```

```
stores.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Store   45 non-null     int64 
 1   Type    45 non-null     object
 2   Size    45 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 1.2+ KB
```

```
train.isnull().sum()
```

```
Store           0
Dept            0
Date            0
Weekly_Sales    0
IsHoliday       0
dtype: int64
```

```
features.isnull().sum()
```

```
Store              0
Date               0
Temperature        0
Fuel_Price         0
MarkDown1       4158
MarkDown2       5269
MarkDown3       4577
MarkDown4       4726
MarkDown5       4140
CPI              585
Unemployment     585
IsHoliday          0
dtype: int64
```

```
stores.isnull().sum()
```

```
Store    0
Type     0
Size     0
dtype: int64
```

```
data = train.merge(features, on=['Store','Date'],
                   how='inner').merge(stores, on=['Store'],
how='inner')
print(data.shape)
```

```
(421570, 17)
```

```
data['MarkDown1'] = data['MarkDown1'].replace(np.nan,0)
data['MarkDown2'] = data['MarkDown2'].replace(np.nan,0)
data['MarkDown3'] = data['MarkDown3'].replace(np.nan,0)
data['MarkDown4'] = data['MarkDown4'].replace(np.nan,0)
data['MarkDown5'] = data['MarkDown5'].replace(np.nan,0)
```

```
data.describe()
```

```
               Store           Dept   Weekly_Sales    Temperature  \
count  421570.000000  421570.000000  421570.000000  421570.000000
mean       22.200546      44.260317   15981.258123      60.090059
std        12.785297      30.492054   22711.183519      18.447931
min         1.000000       1.000000   -4988.940000      -2.060000
25%        11.000000      18.000000    2079.650000      46.680000
```

```
50%      22.000000       37.000000    7612.030000      62.090000
75%      33.000000       74.000000   20205.852500      74.280000
max      45.000000       99.000000  693099.360000     100.140000

           Fuel_Price      MarkDown1      MarkDown2      MarkDown3  \
count  421570.000000  421570.000000  421570.000000  421570.000000
mean        3.361027    2590.074819     879.974298     468.087665
std         0.458515    6052.385934    5084.538801    5528.873453
min         2.472000       0.000000    -265.760000     -29.100000
25%         2.933000       0.000000       0.000000       0.000000
50%         3.452000       0.000000       0.000000       0.000000
75%         3.738000    2809.050000       2.200000       4.540000
max         4.468000   88646.760000  104519.540000  141630.610000

            MarkDown4      MarkDown5            CPI   Unemployment  \
count  421570.000000  421570.000000  421570.000000  421570.000000
mean     1083.132268    1662.772385     171.201947       7.960289
std      3894.529945    4207.629321      39.159276       1.863296
min         0.000000       0.000000     126.064000       3.879000
25%         0.000000       0.000000     132.022667       6.891000
50%         0.000000       0.000000     182.318780       7.866000
75%       425.290000    2168.040000     212.416993       8.572000
max     67474.850000  108519.280000     227.232807      14.313000

                Size
count  421570.000000
mean   136727.915739
std     60980.583328
min     34875.000000
25%     93638.000000
50%    140167.000000
75%    202505.000000
max    219622.000000
```

```
# Filter rows where 'weekly_Sales' is greater than or equal to 0
data_filtered = data[data['Weekly_Sales'] >= 0]

data.describe()
```

```
               Store           Dept   Weekly_Sales    Temperature  \
count  421570.000000  421570.000000  421570.000000  421570.000000
mean       22.200546      44.260317   15981.258123      60.090059
std        12.785297      30.492054   22711.183519      18.447931
min         1.000000       1.000000   -4988.940000      -2.060000
25%        11.000000      18.000000    2079.650000      46.680000
50%        22.000000      37.000000    7612.030000      62.090000
75%        33.000000      74.000000   20205.852500      74.280000
max        45.000000      99.000000  693099.360000     100.140000
```

```
        Fuel_Price     MarkDown1     MarkDown2     MarkDown3  \
count  421570.000000  421570.000000  421570.000000  421570.000000
mean        3.361027    2590.074819     879.974298     468.087665
std         0.458515    6052.385934    5084.538801    5528.873453
min         2.472000       0.000000    -265.760000     -29.100000
25%         2.933000       0.000000       0.000000       0.000000
50%         3.452000       0.000000       0.000000       0.000000
75%         3.738000    2809.050000       2.200000       4.540000
max         4.468000   88646.760000  104519.540000  141630.610000

         MarkDown4      MarkDown5            CPI   Unemployment  \
count  421570.000000  421570.000000  421570.000000  421570.000000
mean     1083.132268    1662.772385     171.201947       7.960289
std      3894.529945    4207.629321      39.159276       1.863296
min         0.000000       0.000000     126.064000       3.879000
25%         0.000000       0.000000     132.022667       6.891000
50%         0.000000       0.000000     182.318780       7.866000
75%       425.290000    2168.040000     212.416993       8.572000
max     67474.850000  108519.280000     227.232807      14.313000

                Size
count  421570.000000
mean   136727.915739
std     60980.583328
min     34875.000000
25%     93638.000000
50%    140167.000000
75%    202505.000000
max    219622.000000
```

```
data = pd.get_dummies(data,columns=['Type'])
```

```
data ['Date']= pd.to_datetime(data['Date'])
```

```
data['month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
```

```
data[['Date','month','Year']].head()
```

```
        Date  month  Year
0 2010-02-05      2  2010
1 2010-02-05      2  2010
2 2010-02-05      2  2010
3 2010-02-05      2  2010
4 2010-02-05      2  2010
```

```
data['dayofweek_name'] = data['Date'].dt.day_name()
data[['Date','dayofweek_name']].head()
```

```
        Date dayofweek_name
0 2010-02-05         Friday
```

```
1 2010-02-05         Friday
2 2010-02-05         Friday
3 2010-02-05         Friday
4 2010-02-05         Friday
```

```
data['is_weekend'] = np.where(data['dayofweek_name'].isin(['Sunday',
'Saturday']), 1, 0)
data[['Date', 'is_weekend']].head()
```

```
        Date  is_weekend
0 2010-02-05           0
1 2010-02-05           0
2 2010-02-05           0
3 2010-02-05           0
4 2010-02-05           0
```

```
data["IsHoliday_x"] = data["IsHoliday_x"].astype(int)
del data['dayofweek_name']

print(data.head())
```

```
   Store  Dept       Date  Weekly_Sales  IsHoliday_x  Temperature
Fuel_Price  \
0      1     1 2010-02-05      24924.50            0        42.31
2.572
1      1     2 2010-02-05      50605.27            0        42.31
2.572
2      1     3 2010-02-05      13740.12            0        42.31
2.572
3      1     4 2010-02-05      39954.04            0        42.31
2.572
4      1     5 2010-02-05      32229.38            0        42.31
2.572

   MarkDown1  MarkDown2  MarkDown3  ...         CPI  Unemployment  \
0        0.0        0.0        0.0  ...  211.096358         8.106
1        0.0        0.0        0.0  ...  211.096358         8.106
2        0.0        0.0        0.0  ...  211.096358         8.106
3        0.0        0.0        0.0  ...  211.096358         8.106
4        0.0        0.0        0.0  ...  211.096358         8.106

   IsHoliday_y    Size  Type_A  Type_B  Type_C  month  Year
is_weekend
0        False  151315    True   False   False      2  2010
0
1        False  151315    True   False   False      2  2010
0
2        False  151315    True   False   False      2  2010
0
3        False  151315    True   False   False      2  2010
```

```
0
4         False  151315    True    False    False      2  2010
0

[5 rows x 22 columns]
```

```
data.to_csv('merged_data.csv',index=False)


df = pd.read_csv(r"C:\Users\kaler\Desktop\Untitled Folder 1\
merged_data.csv", keep_default_na=False, na_values=[""])
print(df.columns)
```

```
Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday_x',
'Temperature',
       'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3',
'MarkDown4',
       'MarkDown5', 'CPI', 'Unemployment', 'IsHoliday_y', 'Size',
'Type_A',
       'Type_B', 'Type_C', 'month', 'Year', 'is_weekend'],
      dtype='object')
```

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming you have already read the CSV file into df

# Combine 'IsHoliday_x' and 'IsHoliday_y' into a single 'IsHoliday'
column
df['IsHoliday'] = df['IsHoliday_x'].combine_first(df['IsHoliday_y'])

# Check if 'IsHoliday' is in the DataFrame
if 'IsHoliday' in df.columns:
    # Select relevant columns in X
    X = df[["Store", "Dept", "Size", "IsHoliday", "CPI",
"Temperature", "Type_B", "Type_C", "MarkDown4", "month", "Year"]]

    # Create a copy of X to avoid the SettingWithCopyWarning
    X = X.copy()

    # Convert boolean values to float in 'Type_B' and 'Type_C'
    X['Type_B'] = X['Type_B'].astype(float)
    X['Type_C'] = X['Type_C'].astype(float)

    y = df['Weekly_Sales']

    # Reshape y to have a single column
    y = y.values.reshape(-1, 1)

    print(X.head())
```

```
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
else:
    print("Column 'IsHoliday' not found in the DataFrame.")
```

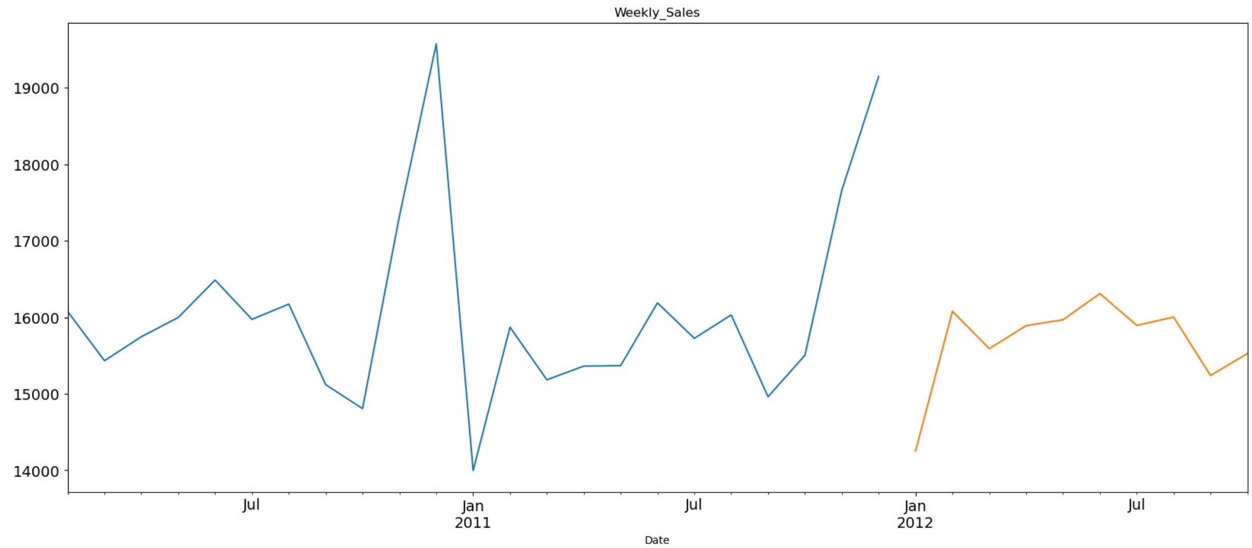
```
   Store  Dept    Size  IsHoliday         CPI  Temperature  Type_B
Type_C  \
0      1     1  151315          0  211.096358        42.31     0.0
0.0
1      1     2  151315          0  211.096358        42.31     0.0
0.0
2      1     3  151315          0  211.096358        42.31     0.0
0.0
3      1     4  151315          0  211.096358        42.31     0.0
0.0
4      1     5  151315          0  211.096358        42.31     0.0
0.0

   MarkDown4  month  Year
0        0.0      2  2010
1        0.0      2  2010
2        0.0      2  2010
3        0.0      2  2010
4        0.0      2  2010
```

```
pip install pmdarima
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: pmdarima in c:\users\kaler\appdata\
roaming\python\python311\site-packages (2.0.4)
Requirement already satisfied: joblib>=0.11 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (1.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in c:\
users\kaler\appdata\roaming\python\python311\site-packages (from
pmdarima) (3.0.5)
Requirement already satisfied: numpy>=1.21.2 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (1.24.3)
Requirement already satisfied: pandas>=0.19 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (2.0.3)
Requirement already satisfied: scikit-learn>=0.22 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (1.3.0)
Requirement already satisfied: scipy>=1.3.2 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (1.11.1)
Requirement already satisfied: statsmodels>=0.13.2 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (0.14.0)
Requirement already satisfied: urllib3 in c:\programdata\anaconda3\
lib\site-packages (from pmdarima) (1.26.16)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in c:\
```

```
programdata\anaconda3\lib\site-packages (from pmdarima) (68.0.0)
Requirement already satisfied: packaging>=17.1 in c:\programdata\
anaconda3\lib\site-packages (from pmdarima) (23.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\
programdata\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\
anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
(2.2.0)
Requirement already satisfied: patsy>=0.5.2 in c:\programdata\
anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
(0.5.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\
site-packages (from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima)
(1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

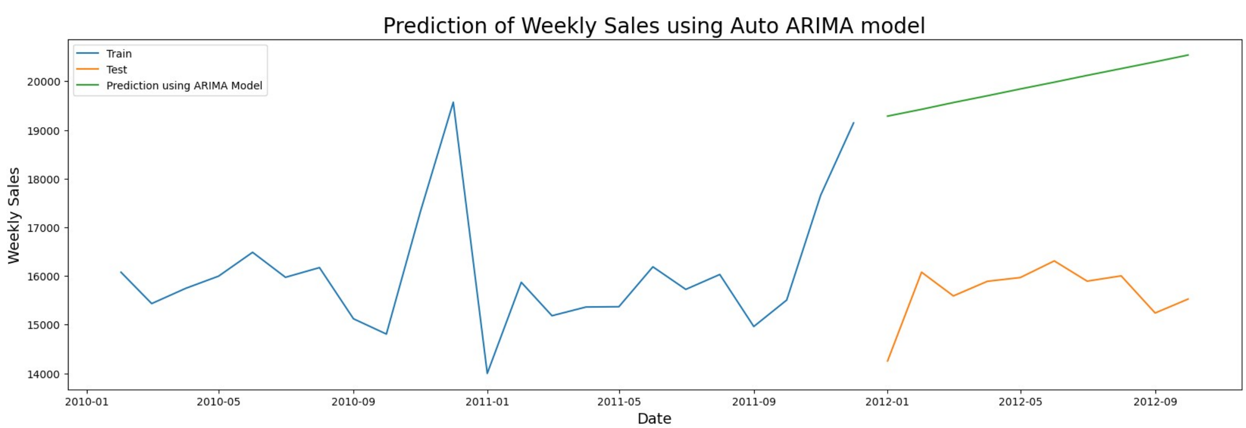
```python
import pmdarima
from pmdarima.arima import auto_arima

df.Date = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df.Date
df = df.drop('Date', axis=1)
df = df.resample('MS').mean()

train_data = df[:int(0.7*(len(df)))]
test_data = df[int(0.7*(len(df))):]

train_data = train_data['Weekly_Sales']
test_data = test_data['Weekly_Sales']

train_data.plot(figsize=(20,8), title= 'Weekly_Sales', fontsize=14)
test_data.plot(figsize=(20,8), title='Weekly_Sales', fontsize =14)
plt.show()
```




```
from pmdarima import auto_arima

# Assuming you have a 'train_data' time series
model_auto_arima = auto_arima(train_data,
                              trace=True,
                              error_action='ignore',
                              suppress_warnings=True,
                              seasonal=True,
                              stepwise=False,
                              start_p=0, start_q=0,
                              max_p=10, max_q=10,
                              start_P=0, start_Q=0,
                              max_P=10, max_Q=10,
                              d=1, D=1, max_D=10,
                              approximation=False)

model_auto_arima.fit(train_data)

 ARIMA(0,1,0)(0,0,0)[1] intercept   : AIC=391.554, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[1] intercept   : AIC=391.889, Time=0.04 sec
 ARIMA(0,1,2)(0,0,0)[1] intercept   : AIC=393.641, Time=0.04 sec
 ARIMA(0,1,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.15 sec
 ARIMA(0,1,4)(0,0,0)[1] intercept   : AIC=397.671, Time=0.20 sec
 ARIMA(0,1,5)(0,0,0)[1] intercept   : AIC=inf, Time=0.38 sec
 ARIMA(1,1,0)(0,0,0)[1] intercept   : AIC=392.085, Time=0.05 sec
 ARIMA(1,1,1)(0,0,0)[1] intercept   : AIC=393.555, Time=0.15 sec
 ARIMA(1,1,2)(0,0,0)[1] intercept   : AIC=395.595, Time=0.13 sec
 ARIMA(1,1,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.34 sec
 ARIMA(1,1,4)(0,0,0)[1] intercept   : AIC=inf, Time=0.38 sec
 ARIMA(2,1,0)(0,0,0)[1] intercept   : AIC=393.685, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[1] intercept   : AIC=395.576, Time=0.07 sec
 ARIMA(2,1,2)(0,0,0)[1] intercept   : AIC=395.453, Time=0.39 sec
 ARIMA(2,1,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.41 sec
```

```
 ARIMA(3,1,0)(0,0,0)[1] intercept   : AIC=395.636, Time=0.05 sec
 ARIMA(3,1,1)(0,0,0)[1] intercept   : AIC=397.572, Time=0.11 sec
 ARIMA(3,1,2)(0,0,0)[1] intercept   : AIC=inf, Time=0.34 sec
 ARIMA(4,1,0)(0,0,0)[1] intercept   : AIC=397.546, Time=0.06 sec
 ARIMA(4,1,1)(0,0,0)[1] intercept   : AIC=399.554, Time=0.12 sec
 ARIMA(5,1,0)(0,0,0)[1] intercept   : AIC=399.314, Time=0.07 sec

Best model:  ARIMA(0,1,0)(0,0,0)[1] intercept
Total fit time: 3.587 seconds

ARIMA(order=(0, 1, 0), scoring_args={}, seasonal_order=(0, 0, 0, 1),
      suppress_warnings=True)
```

```
forecast = model_auto_arima.predict(n_periods=len(test_data))
forecast = pd.DataFrame(forecast,index =
test_data.index,columns=['Predictions'])
plt.figure(figsize=(20, 6))
plt.title('Prediction of Weekly Sales using Auto ARIMA model',
fontsize=20)
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(forecast, label='Prediction using ARIMA Model')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()
```

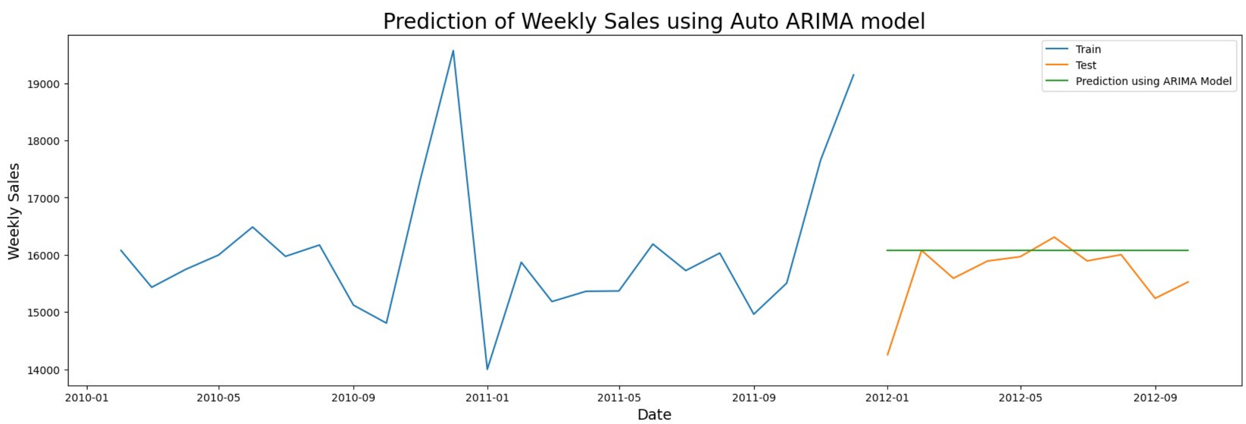



```
import pandas as pd
import matplotlib.pyplot as plt
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Assuming you have already defined 'train_data' and 'test_data'
# ... (Code for training the model and generating 'forecast')
```

```python
# Use auto_arima for hyperparameter tuning
model_auto_arima = auto_arima(train_data, trace=True,
error_action='ignore', suppress_warnings=True, seasonal=True,
stepwise=False)

# Fit the auto_arima model
model_auto_arima.fit(train_data)

# Generate forecast
forecast = model_auto_arima.predict(n_periods=len(test_data))
forecast = pd.DataFrame(forecast, index=test_data.index,
columns=['Predictions'])

# Plotting
plt.figure(figsize=(20, 6))
plt.title('Prediction of Weekly Sales using Auto ARIMA model',
fontsize=20)
plt.plot(train_data, label='Train')
plt.plot(test_data, label='Test')
plt.plot(forecast, label='Prediction using ARIMA Model')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()

# Calculate metrics
predictions = forecast['Predictions']
mse = mean_squared_error(test_data, predictions)
rmse = np.sqrt(mse)
mae = mean_absolute_error(test_data, predictions)

print('Mean Squared Error (MSE) of Auto ARIMA:', mse)
print('Root Mean Squared Error (RMSE) of Auto ARIMA:', rmse)
print('Mean Absolute Deviation (MAE) of Auto ARIMA:', mae)
```

```
 ARIMA(0,0,0)(0,0,0)[1] intercept   : AIC=397.842, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[1] intercept   : AIC=399.420, Time=0.09 sec
 ARIMA(0,0,2)(0,0,0)[1] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(0,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.20 sec
 ARIMA(0,0,4)(0,0,0)[1] intercept   : AIC=399.645, Time=0.25 sec
 ARIMA(0,0,5)(0,0,0)[1] intercept   : AIC=inf, Time=0.41 sec
 ARIMA(1,0,0)(0,0,0)[1] intercept   : AIC=399.658, Time=0.03 sec
 ARIMA(1,0,1)(0,0,0)[1] intercept   : AIC=401.559, Time=0.06 sec
 ARIMA(1,0,2)(0,0,0)[1] intercept   : AIC=404.248, Time=0.09 sec
 ARIMA(1,0,3)(0,0,0)[1] intercept   : AIC=400.338, Time=0.13 sec
 ARIMA(1,0,4)(0,0,0)[1] intercept   : AIC=inf, Time=0.30 sec
 ARIMA(2,0,0)(0,0,0)[1] intercept   : AIC=399.687, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[1] intercept   : AIC=403.287, Time=0.20 sec
 ARIMA(2,0,2)(0,0,0)[1] intercept   : AIC=404.649, Time=0.33 sec
 ARIMA(2,0,3)(0,0,0)[1] intercept   : AIC=402.435, Time=0.41 sec
```

```
 ARIMA(3,0,0)(0,0,0)[1] intercept   : AIC=400.869, Time=0.29 sec
 ARIMA(3,0,1)(0,0,0)[1] intercept   : AIC=403.022, Time=0.22 sec
 ARIMA(3,0,2)(0,0,0)[1] intercept   : AIC=405.264, Time=0.33 sec
 ARIMA(4,0,0)(0,0,0)[1] intercept   : AIC=402.820, Time=0.17 sec
 ARIMA(4,0,1)(0,0,0)[1] intercept   : AIC=405.025, Time=0.41 sec
 ARIMA(5,0,0)(0,0,0)[1] intercept   : AIC=404.311, Time=0.18 sec

Best model:  ARIMA(0,0,0)(0,0,0)[1] intercept
Total fit time: 4.405 seconds
```




```
Mean Squared Error (MSE) of Auto ARIMA: 468477.9539606969
Root Mean Squared Error (RMSE) of Auto ARIMA: 684.4544937106461
Mean Absolute Deviation (MAE) of Auto ARIMA: 446.8196095294599
```

```
from sklearn.ensemble import RandomForestRegressor
import pickle

# Assuming you have already defined and trained your
RandomForestRegressor 'rf'
rf = RandomForestRegressor(n_estimators=50, max_depth=20,
min_samples_split=3, min_samples_leaf=1)
# ... (train the model with your data)

# Save the trained model to a file
with open('final_model.pkl', 'wb') as model_file:
    pickle.dump(rf, model_file)

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=50, max_depth=20,
min_samples_split=3, min_samples_leaf=1)
rf.fit(X_train, y_train.ravel())  # Assuming X_train, y_train are your
training data
print('Accuracy:', rf.score(X_test, y_test.ravel()) * 100, '%')
y_pred = rf.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

rms = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', rms)
print('MAE:', mean_absolute_error(y_test, y_pred))
print("Before fitting the model")
rf.fit(X_train, y_train.ravel())  # Assuming X_train, y_train are your
training data
print("After fitting the model")

# Add similar print statements at different stages
```

```
Accuracy: 96.4577508824995 %
RMSE: 4315.378610052086
MAE: 1648.949481735198
Before fitting the model
After fitting the model
```

```
print('Training Accuracy:',rf.score(X_train,y_train.ravel())*100,'%')
```

```
Training Accuracy: 99.127412543821 %
```

```
import xgboost as xgb
import warnings

import xgboost as xgb

xg_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4,
                          n_estimators=500, max_depth=4,
learning_rate=0.5)
xg_reg.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None,
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.5,
max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=500, n_jobs=None,
nthread=4,
             num_parallel_tree=None, ...)
```

```
pred=xg_reg.predict(X_train)
y_pred=xg_reg.predict(X_test)

print('Accuracy:',xg_reg.score(X_test,y_test)*100,'%')

rms = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:',rms)

print('MAE:',mean_absolute_error(y_test, y_pred))
```

```
Accuracy: 93.20892619359017 %
RMSE: 5975.146796472817
MAE: 3035.7967659356923
```

```
print('Training Accuracy:',xg_reg.score(X_train,y_train)*100,'%')
```

```
Training Accuracy: 94.29593668521562 %
```

```
pip install prettytable
```

```
Defaulting to user installation because normal site-packages is not
writeableNote: you may need to restart the kernel to use updated
packages.

Requirement already satisfied: prettytable in c:\users\kaler\appdata\
roaming\python\python311\site-packages (3.9.0)
Requirement already satisfied: wcwidth in c:\programdata\anaconda3\
lib\site-packages (from prettytable) (0.2.5)
```

```
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names = ["Model","Training Accuracy","Testing
Accuracy","RMSE","MAE"]
tb.add_row(["Random Forest", 99.11, 96.77, 4055.83, 1651.13])
tb.add_row(["XgBoost", 94.23, 93.72, 5660.68, 3129.36])

print(tb)
```

```
+---------------+-------------------+------------------+---------
+---------+
|     Model     | Training Accuracy | Testing Accuracy |   RMSE  |
MAE   |
+---------------+-------------------+------------------+---------
+---------+
| Random Forest |       99.11       |      96.77       | 4055.83 |
1651.13 |
|    XgBoost    |       94.23       |      93.72       | 5660.68 |
3129.36 |
+---------------+-------------------+------------------+---------
+---------+
```

```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

# Random Forest
rf = RandomForestRegressor(n_estimators=58, max_depth=27,
min_samples_split=3, min_samples_leaf=1)
rf.fit(X_train, y_train.ravel())
y_pred_rf = rf.predict(X_test)

# XGBoost
xg_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4,
n_estimators=500, max_depth=4, learning_rate=0.5)
xg_reg.fit(X_train, y_train)
pred_xg = xg_reg.predict(X_train)
y_pred_xg = xg_reg.predict(X_test)
```


```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

# Random Forest
rf = RandomForestRegressor(n_estimators=58, max_depth=27,
min_samples_split=3, min_samples_leaf=1)
rf.fit(X_train, y_train.ravel())
y_pred_rf = rf.predict(X_test)

# XGBoost
xg_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4,
n_estimators=500, max_depth=4, learning_rate=0.5)
xg_reg.fit(X_train, y_train)
pred_xg = xg_reg.predict(X_train)
y_pred_xg = xg_reg.predict(X_test)
```


```python
cv=cross_val_score(xg_reg,X,y,cv=6)

np.mean(cv)
```

```
0.7387614591639351
```

```python
pickle.dump(rf, open('newfinal_model.pkl','wb'))
```