**Phase-4**

**Development Part2**

| **Date** | **29-10-2023** |
| --- | --- |
| **Team ID** | **3909** |
| **Project Name** | **Product Demand using Machine learnings** |

**Table of Contents**

| **S.No** | **Title** |
| --- | --- |
| 1 | Code |
| 2 | Feature extraction |
| 3 | ARIMA Model |
| 4 | Evaluation |
| 5 | Conclusion |

**Code:**

#Importing Libraries

*# Data Manipulation*  
**import** numpy **as** np  
**import** pandas **as** pd  
  
*# Data Visualization*  
**import** matplotlib.pyplot **as** plt  
%matplotlib inline  
**import** seaborn **as** sns  
**import** plotly.express **as** px  
**import** bokeh  
  
**import** seaborn **as** sns  
color\_pal = sns.color\_palette()  
plt.style.use('fivethirtyeight')  
  
*# For Analysis and Forecasting*  
**from** scipy **import** stats  
  
*# Others*  
**import** datetime  
**import** os  
**import** pickle  
**import** requests

#Importing Data

df = pd.read\_csv(r"/content/Historical Product Demand.csv")

df.head()

Product\_Code Warehouse Product\_Category Date Order\_Demand  
0 Product\_0993 Whse\_J Category\_028 2012/7/27 100.0  
1 Product\_0979 Whse\_J Category\_028 2012/1/19 500.0  
2 Product\_0979 Whse\_J Category\_028 2012/2/3 500.0  
3 Product\_0979 Whse\_J Category\_028 2012/2/9 500.0  
4 Product\_0979 Whse\_J Category\_028 2012/3/2 500.0

df.tail()

Product\_Code Warehouse Product\_Category Date Order\_Demand  
84885 Product\_1241 Whse\_J Category\_019 2012/8/17 1000.0  
84886 Product\_1239 Whse\_J Category\_019 2012/8/21 200.0  
84887 Product\_1241 Whse\_J Category\_019 2012/9/21 20000.0  
84888 Product\_1239 Whse\_J Category\_019 2012/9/13 1000.0  
84889 Product\_1241 Whse\_J NaN NaN NaN

#DATA CLEANING AND PREPROCESSING

df.describe()

Order\_Demand  
count 8.488900e+04  
mean 8.810712e+03  
std 4.283116e+04  
min 0.000000e+00  
25% 1.000000e+02  
50% 1.000000e+03  
75% 5.000000e+03  
max 4.000000e+06

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 84890 entries, 0 to 84889  
Data columns (total 5 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Product\_Code 84890 non-null object   
 1 Warehouse 84890 non-null object   
 2 Product\_Category 84889 non-null object   
 3 Date 84888 non-null object   
 4 Order\_Demand 84889 non-null float64  
dtypes: float64(1), object(4)  
memory usage: 3.2+ MB

print("Number of attributes with null vaules: ", df.isnull().any().sum())  
print("Percentage of missing values: ",df.isnull().any(axis=1).sum()/len(df)\*100)

Number of attributes with null vaules: 3  
Percentage of missing values: 0.0023559901048415596

df.dropna(axis=0, how="any", inplace=True)

df["Date"] = pd.to\_datetime(df['Date'])  
*# df["Order\_Demand"] = df["Order\_Demand"].str.replace("(", "")*  
*# df["Order\_Demand"] = df["Order\_Demand"].str.replace(")", "")*  
*#Changing the datatype to float*  
df["Order\_Demand"] = df["Order\_Demand"].astype(float)

df = df.sort\_values(by=['Date', 'Product\_Code'])  
df = df.set\_index('Date')  
df.head()

Product\_Code Warehouse Product\_Category Order\_Demand  
Date   
2011-01-08 Product\_0965 Whse\_A Category\_006 2.0  
2011-05-31 Product\_1724 Whse\_A Category\_003 108.0  
2011-06-24 Product\_1521 Whse\_S Category\_019 85000.0  
2011-06-24 Product\_1521 Whse\_S Category\_019 7000.0  
2011-09-02 Product\_1507 Whse\_C Category\_019 1250.0

category\_yearly\_demand = df.groupby([df.index.year, 'Product\_Category'])['Order\_Demand'].mean()

padded\_category\_data = {}  
**for** category, category\_data **in** category\_yearly\_demand.groupby(level = 'Product\_Category'):  
*# print(f"Category: {category}")*  
 padded\_category\_data[category] = [0 **for** \_ **in** range(7)]  
 **for** year, total\_demand **in** category\_data.items():  
 index = ((year[0] - 2010) % 7) - 1  
 padded\_category\_data[category][index] = total\_demand  
  
fig = plt.figure(figsize=(12, 25))  
rows, cols = 11, 3  
x = [2011, 2012, 2013, 2014, 2015, 2016, 2017]  
  
**for** title, data **in** padded\_category\_data.items():  
 *# Create subplots in the grid*  
 ax = fig.add\_subplot(rows, cols, int(title[-2:]))  
 *# Plotting data on the current subplot*  
 ax.plot(x, data)  
 ax.set\_title(title)  
  
plt.tight\_layout()  
fig.suptitle("Yearly Average Demand for all the Product Categories", y=1.02)

Text(0.5, 1.02, 'Yearly Average Demand for all the Product Categories')

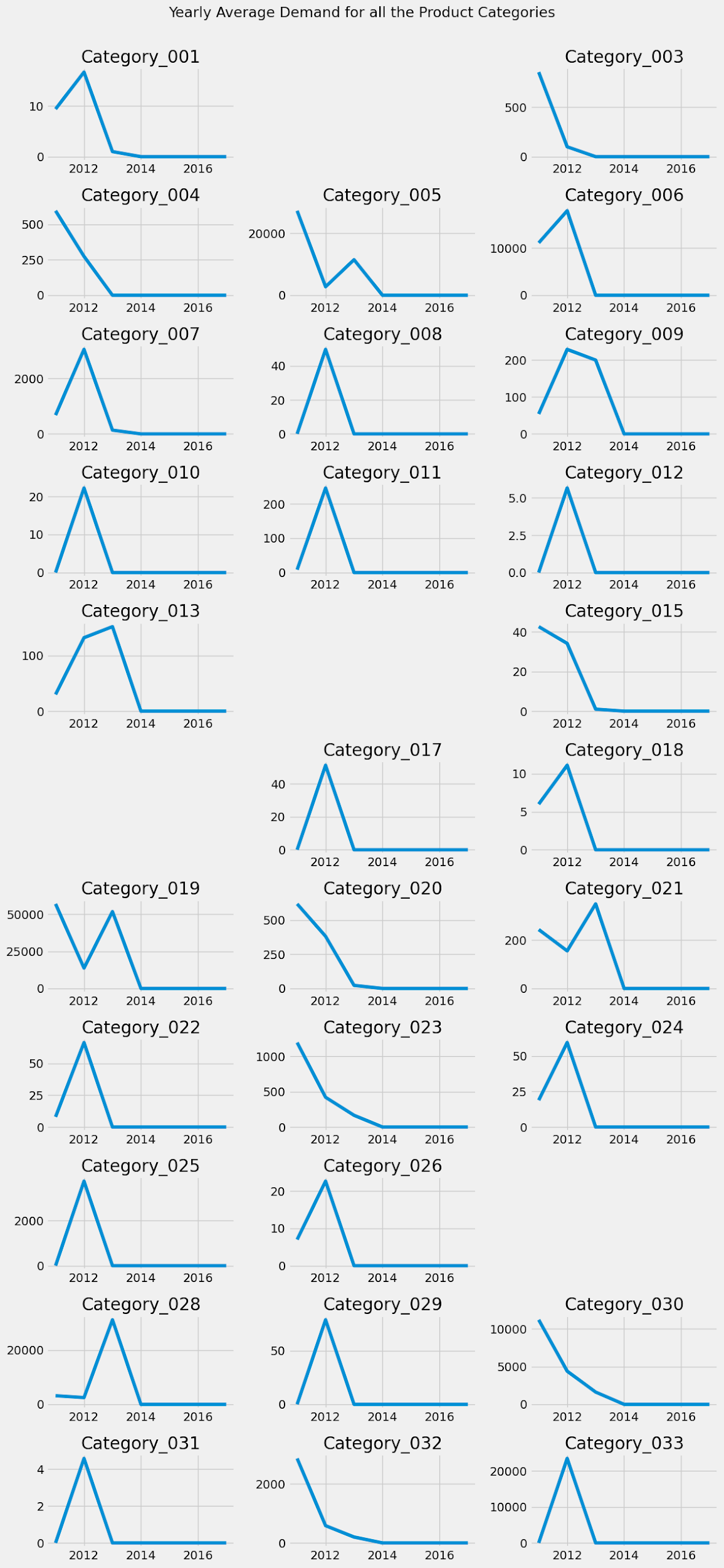


Figure 1: Category wise product demand

padded\_yearly\_categories = {}  
**for** year, year\_data **in** category\_yearly\_demand.groupby(level = 'Date'):  
*# print(f"Category: {category}")*  
 padded\_yearly\_categories[year] = [0 **for** \_ **in** range(33)]  
 **for** category, total\_demand **in** year\_data.items():  
*# print(category)*  
 index = (int(category[1][-2:]) % 33) - 1  
 padded\_yearly\_categories[year][index] = total\_demand  
  
x = [i+1 **for** i **in** range(33)]  
rows = len(padded\_yearly\_categories)  
cols = 1  
  
fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(10, 20))  
  
colors = plt.cm.viridis(np.linspace(0, 1, len(x)))  
  
**for** i, (year, data) **in** enumerate(padded\_yearly\_categories.items()):  
 *# Calculate the row and column indices for the subplot*  
  
 *# Create a bar plot in the current subplot*  
 bars = axes[i].bar(x, data, color=colors)  
 axes[i].bar\_label(bars, labels=x, fontsize = 8)  
  
 *# Set the category title as the subplot title*  
 axes[i].set\_title(year)  
  
 *# Hide only the y-axis scales (ticks)*  
*# axes[row\_idx, col\_idx].get\_yaxis().set\_visible(False)*  
  
plt.tight\_layout()  
fig.suptitle("Yearwise Average Demand of all Product Categories", y=1.01)  
plt.show()

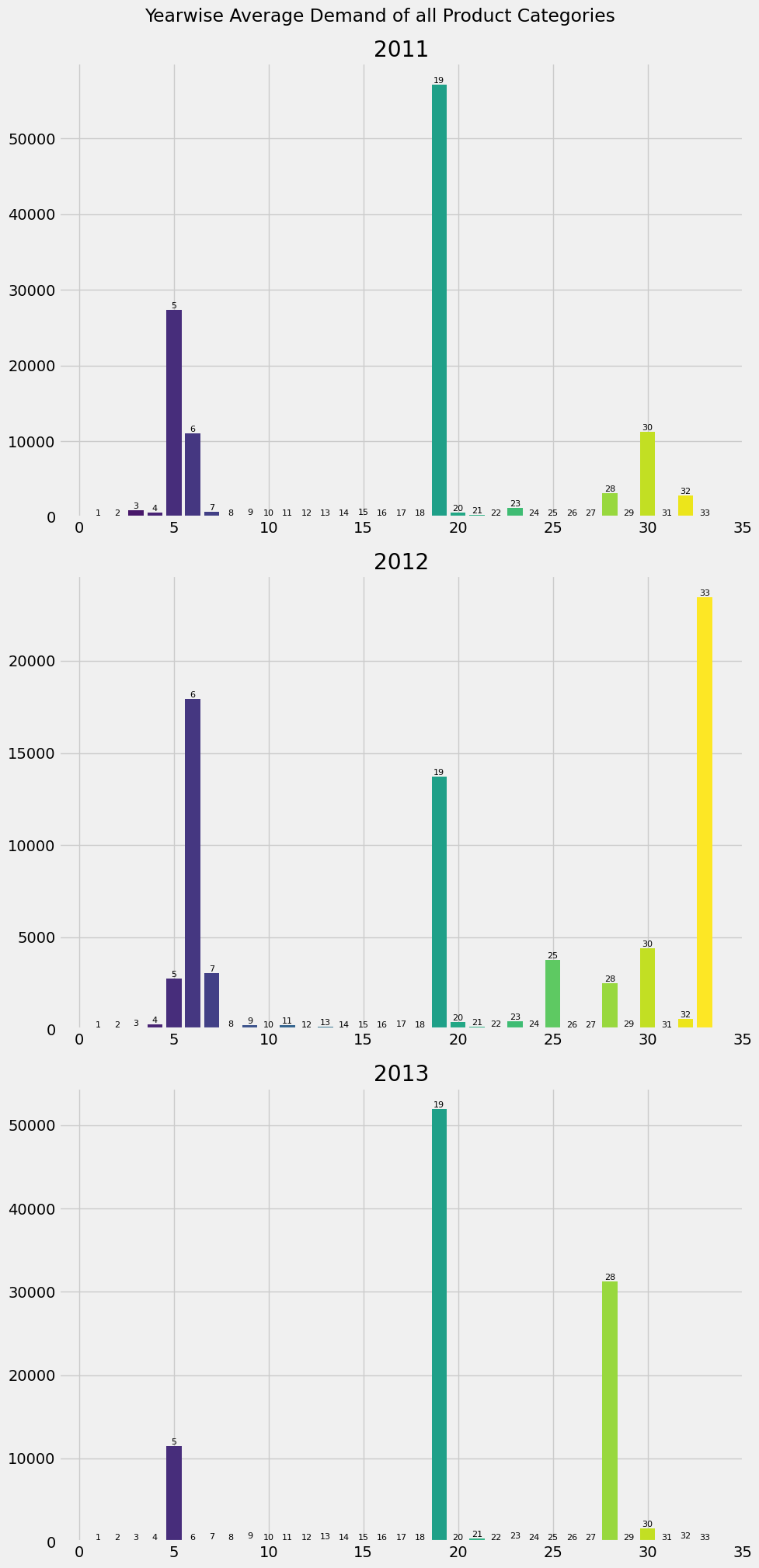


Figure 2: Year Wise average demand of all product categories

warehouse\_yearly\_demand = df.groupby([df.index.year, 'Warehouse'])['Order\_Demand'].mean()

demand\_data = {}  
warehouses = []  
years = [2011 + i **for** i **in** range(7)]  
**for** warehouse, warehouse\_data **in** warehouse\_yearly\_demand.groupby(level='Warehouse'):  
 warehouses.append(warehouse)  
 demand\_data[warehouse] = [0 **for** i **in** range(7)]  
 **for** year, year\_data **in** warehouse\_data.items():  
 index = ((year[0] - 2010) % 7) - 1  
 demand\_data[warehouse][index] = year\_data  
*# print(demand\_data)*  
  
*# Determine the number of warehouses and the number of years*  
num\_warehouses = len(warehouses)  
num\_years = len(years)  
  
*# Set the width of the bars*  
bar\_width = 0.15  
  
*# Create a figure*  
fig, ax = plt.subplots(figsize=(12, 8))  
  
*# Define the index for the x-axis*  
x = np.arange(num\_years)  
  
*# Create a grouped bar chart*  
**for** i, warehouse **in** enumerate(warehouses):  
 *# Offset the x-position for each warehouse*  
 x\_pos = x + i \* bar\_width  
  
 *# Plot the demand values for the current warehouse*  
 ax.bar(x\_pos, demand\_data[warehouse], width=bar\_width, label=warehouse)  
  
*# Set x-axis labels and tick positions*  
ax.set\_xticks(x + (num\_warehouses - 1) \* bar\_width / 2)  
ax.set\_xticklabels(years)  
  
*# Set labels and title*  
ax.set\_xlabel('Year')  
ax.set\_ylabel('Demand')  
ax.set\_title('Yearly Average Demand by Warehouse')  
  
*# Add a legend to distinguish the warehouses*  
ax.legend()  
  
*# Show the chart*  
plt.show()

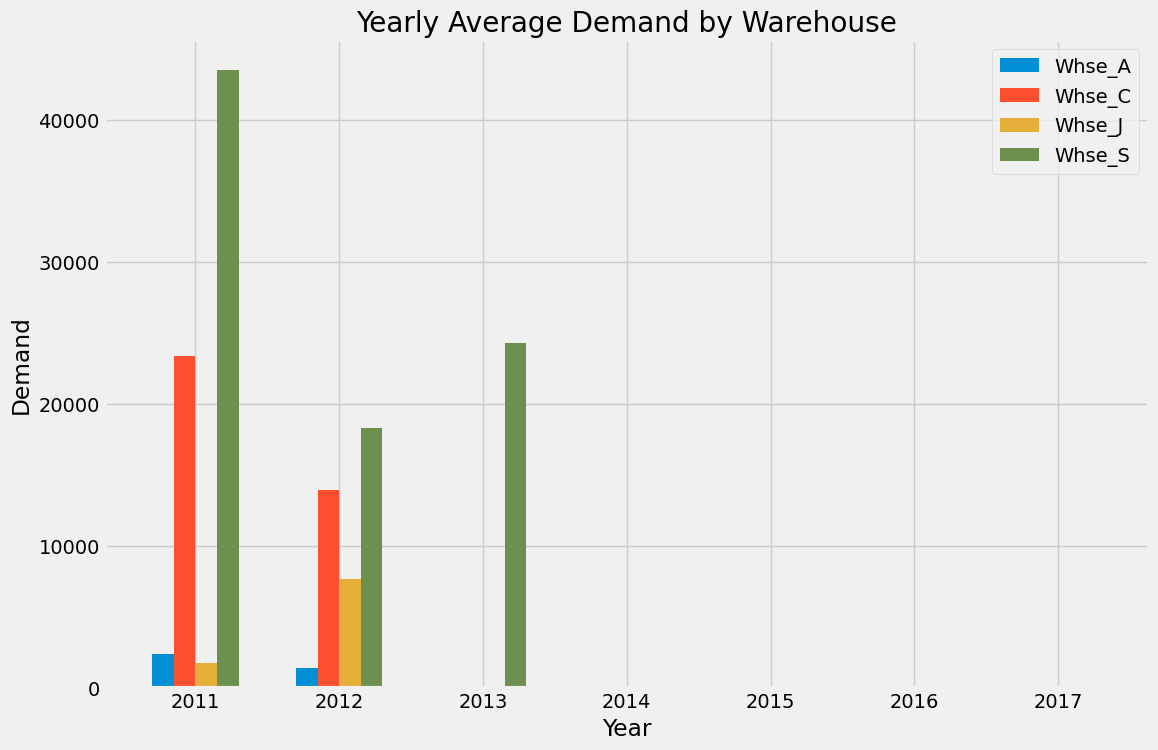


Figure 3: Yearly average demand by warehouse

**#MODEL & EVALUATION**

*# Rename the columns*  
df = df.rename(columns = {'Product\_Code': 'Code',  
 'Product\_Category':'Category',  
 'Order\_Demand':'Demand'})  
df.head()

Code Warehouse Category Demand  
Date   
2011-01-08 Product\_0965 Whse\_A Category\_006 2.0  
2011-05-31 Product\_1724 Whse\_A Category\_003 108.0  
2011-06-24 Product\_1521 Whse\_S Category\_019 85000.0  
2011-06-24 Product\_1521 Whse\_S Category\_019 7000.0  
2011-09-02 Product\_1507 Whse\_C Category\_019 1250.0

df\_new = df.loc[df.index >= '01-01-2012']  
df\_new.plot(kind = 'line',figsize=(15, 5),color = color\_pal[0], title = 'Order Demand' )  
plt.show()

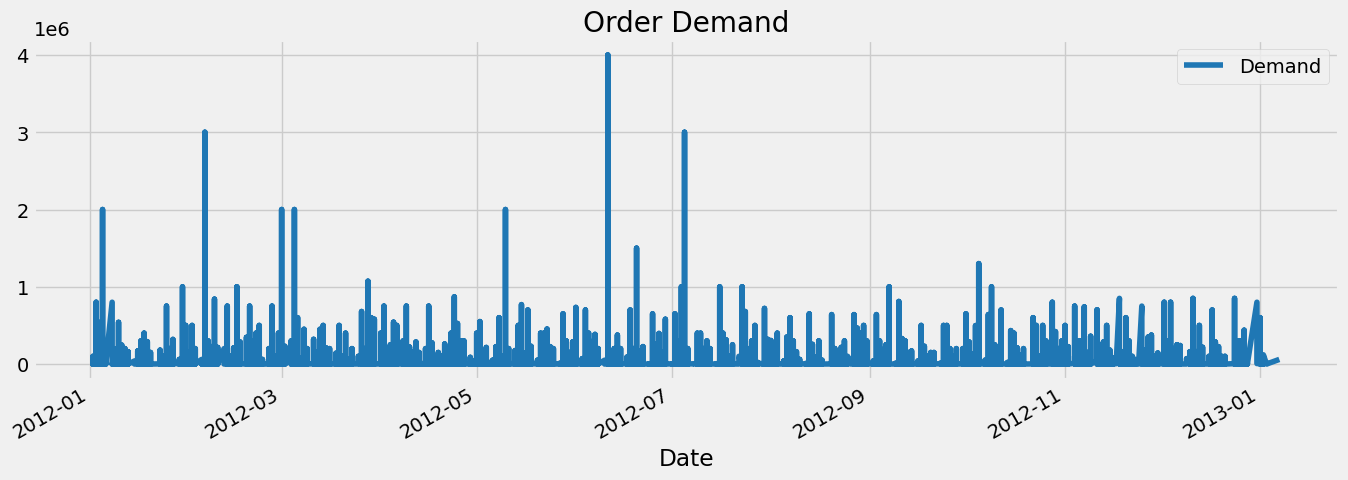


Figure 4: Order Demand

*" Function that create time series features using the index columns"*  
**def** create\_feature(dataframe):  
 dataframe = dataframe.copy()  
 dataframe['day\_of\_the\_week'] = dataframe.index.dayofweek  
 dataframe['Quarter'] = dataframe.index.quarter  
 dataframe['Month'] = dataframe.index.month  
 dataframe['Year'] = dataframe.index.year  
 dataframe['Week'] = dataframe.index.isocalendar().week.astype(int)  
 **return** dataframe  
  
df = create\_feature(df\_new)  
df.dtypes

Code object  
Warehouse object  
Category object  
Demand float64  
day\_of\_the\_week int64  
Quarter int64  
Month int64  
Year int64  
Week int64  
dtype: object

*# features, Target variable*  
Features = ['day\_of\_the\_week', 'Quarter','Month', 'Year', 'Week']  
target = ['Demand']

*# Split the data in Train/ Test set*  
df\_train = df.loc[df.index <= '2012-11-01'].copy()  
df\_test = df.loc[df.index > '2012-11-01'].copy()  
  
*#plot train/test*  
fig, ax = plt.subplots(figsize = (15,5))  
df\_train.Demand.plot(ax = ax, label = 'train set',legend = 'train set', title = 'Train/Test set')  
df\_test.Demand.plot(ax = ax, legend = 'train set', label = 'test set')  
plt.show()

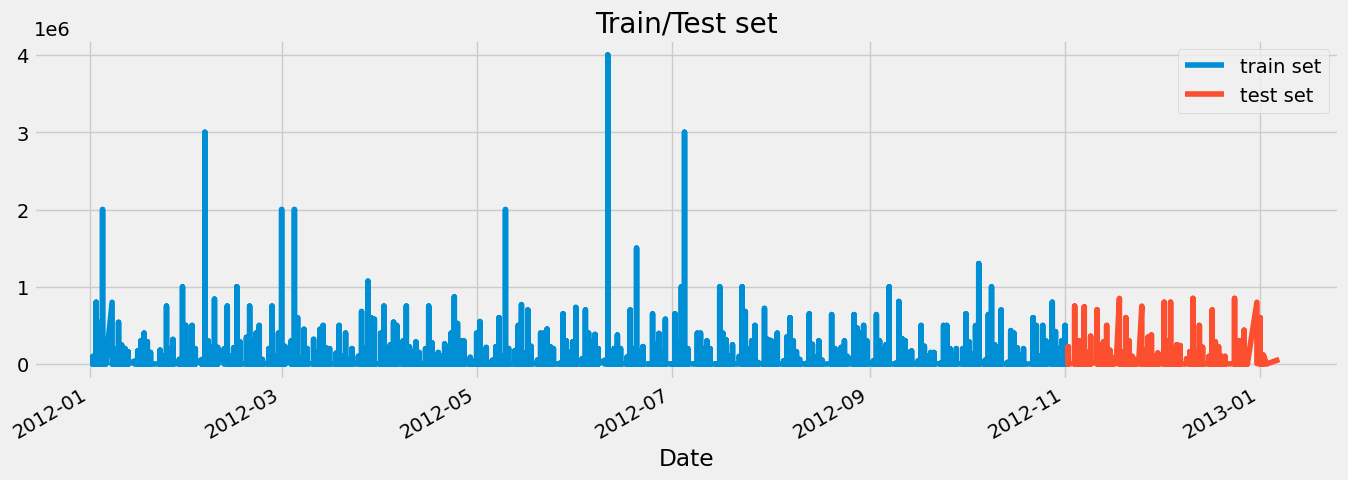


Figure 5: Train/Test Set

X\_train = df\_train[Features]  
X\_test = df\_test[Features]  
  
y\_train = df\_train[target]  
y\_test = df\_test[target]

**from** xgboost **import** XGBRegressor  
model = XGBRegressor(n\_estimators = 1000,  
 early\_stopping\_rounds = 50,  
 learning\_rate = 0.01).fit(X\_train, y\_train,  
 eval\_set = [(X\_train,y\_train), (X\_test,y\_test)],  
 verbose = 50)

[0] validation\_0-rmse:43755.20432 validation\_1-rmse:33950.47798  
[50] validation\_0-rmse:43697.64467 validation\_1-rmse:33839.96202  
[100] validation\_0-rmse:43670.79389 validation\_1-rmse:33806.24955  
[150] validation\_0-rmse:43656.35298 validation\_1-rmse:33809.45442  
[174] validation\_0-rmse:43651.85164 validation\_1-rmse:33815.04443

*# Features Importance*  
fea\_Imp = pd.DataFrame(model.feature\_importances\_, index = Features, columns = ['feature\_importance'])  
*#plot the features importance*  
fea\_Imp.feature\_importance.sort\_values().plot(kind = 'barh', figsize =(5,3))  
plt.show()

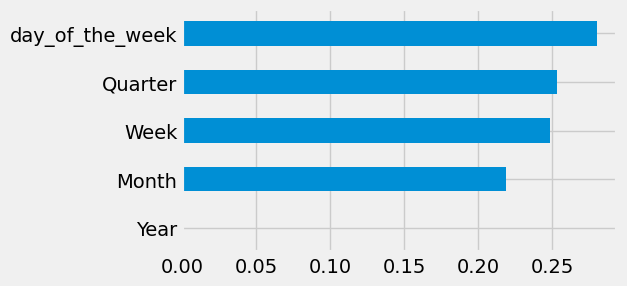
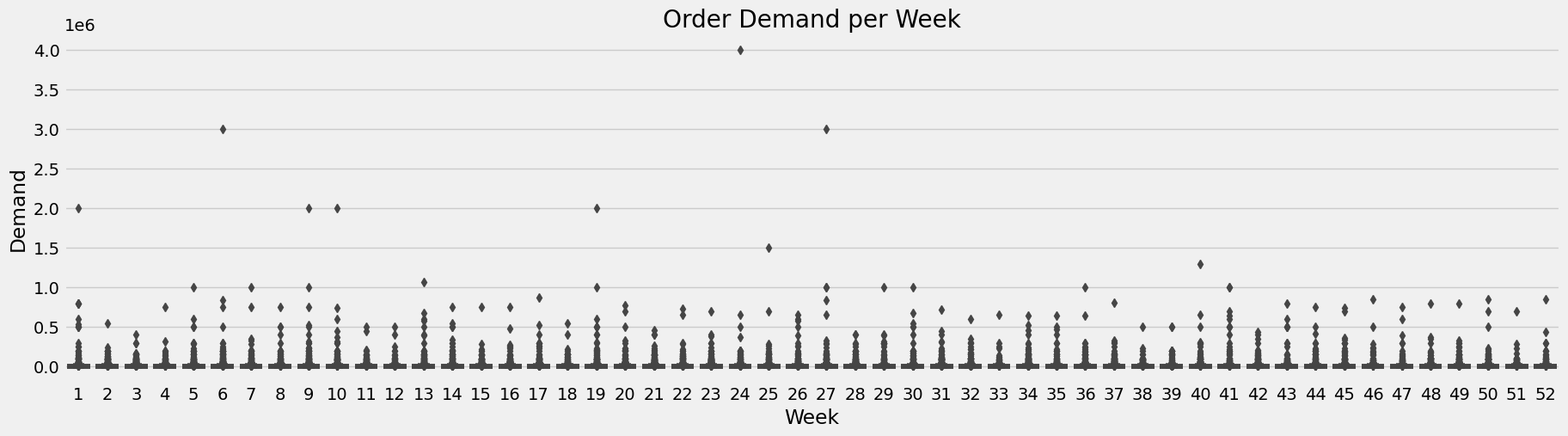


Figure 6: Feature extraction

**#TIME SERIES ANALYSIS**

fig, ax = plt.subplots(figsize=(20, 5))  
sns.boxplot(data=df, x='Week', y='Demand')  
ax.set\_title('Order Demand per Week')  
plt.show()

  
Figure 7: Order Demand Per Week

df\_week = df.resample('W').mean()  
df\_week['Demand'].plot(figsize = (15,4), title = 'Weekly Order demand')  
plt.show()

<ipython-input-28-ff46de005cc5>:1: FutureWarning: The default value of numeric\_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric\_only will default to False. Either specify numeric\_only or select only columns which should be valid for the function.  
 df\_week = df.resample('W').mean()

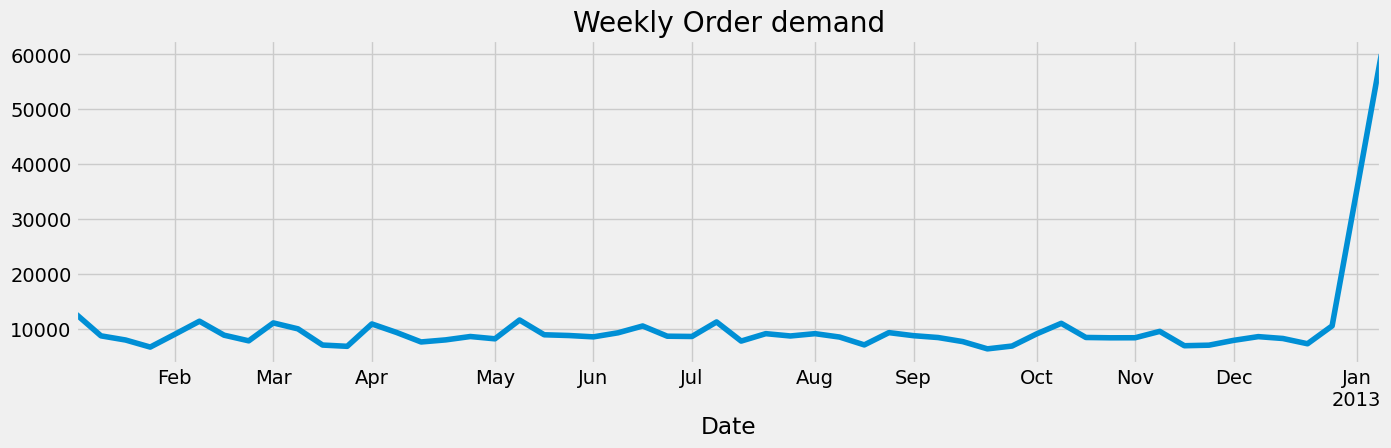


Figure 8: Weekly Order Demand

df\_month = df.resample('MS').mean()  
df\_month.Demand.plot(figsize = (25,5), title = 'Monthly Order Demand', color = color\_pal[2])  
plt.ylabel('Demand')  
plt.show()

<ipython-input-29-f9bbe684cd85>:1: FutureWarning: The default value of numeric\_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric\_only will default to False. Either specify numeric\_only or select only columns which should be valid for the function.  
 df\_month = df.resample('MS').mean()



Figure 9: Monthly order Demand

**#ARIMA MODEL**

**from** statsmodels.graphics.tsaplots **import** plot\_acf, plot\_pacf

df\_diff = df\_month.Demand.diff()[1:]  
df\_diff.plot(figsize=(12,6), color = color\_pal[1])  
plt.axhline(0, linestyle='--', color='k', alpha=0.3)

<matplotlib.lines.Line2D at 0x780e3468f5e0>

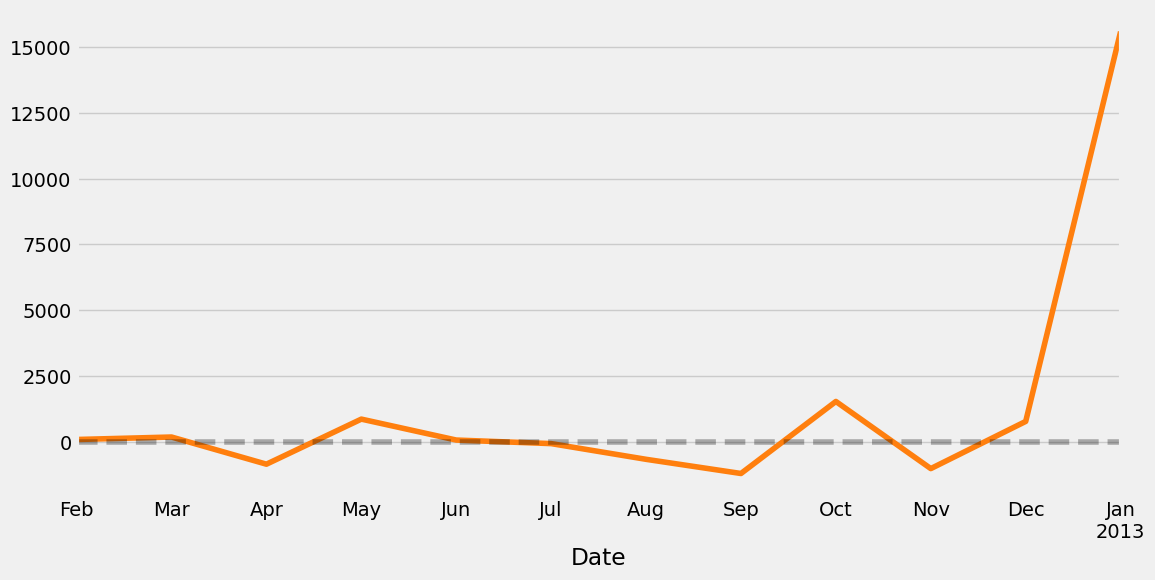


Figure 10: Monthly order demand

**#TRAIN & TEST SPLIT**

**from** datetime **import** datetime  
**from** datetime **import** timedelta  
train\_end = datetime(2012,11,1)  
test\_end = datetime(2013,1,1)  
  
df\_train = df\_diff[:train\_end]  
df\_test = df\_diff[train\_end:test\_end]  
  
*# plot train and test dataset on the same graph*  
fig, ax = plt.subplots(figsize=(15, 5))  
df\_train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')  
df\_test.plot(ax=ax, label='Test Set')  
ax.axvline('01-01-2016', color='black', ls='--')  
ax.legend(['Training Set', 'Test Set','Split Point'])  
plt.show()

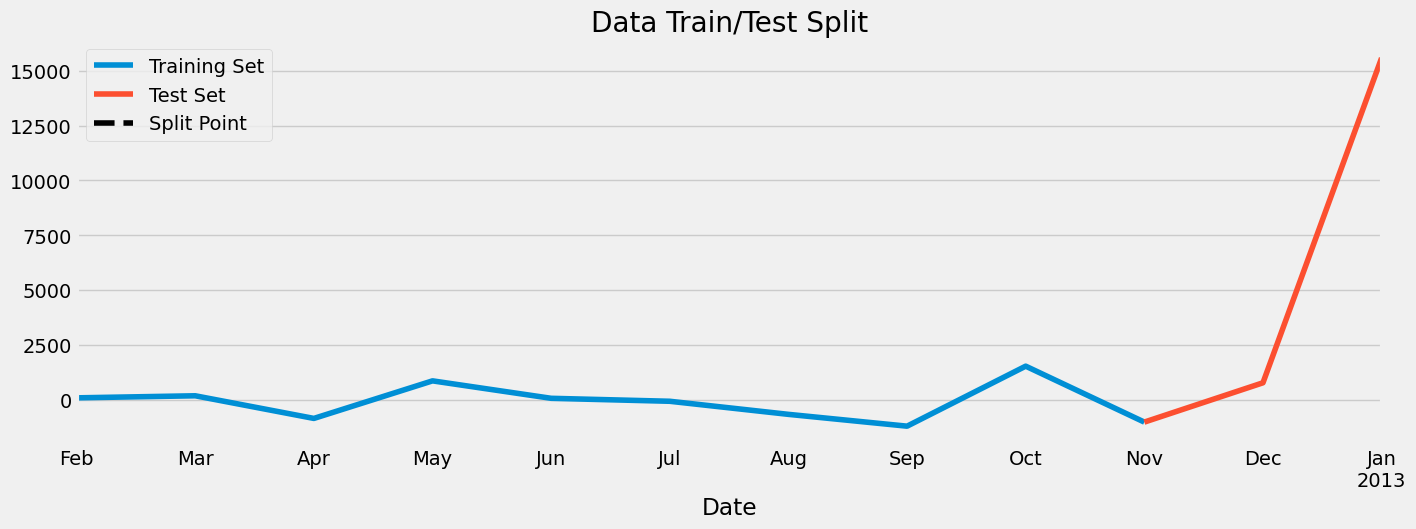


Figure 11: Data Train/Test Split

**from** statsmodels.tsa.arima.model **import** ARIMA  
model\_arima = ARIMA(df\_train, order = (2,1,1)).fit()

/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle\_retvals  
 warnings.warn("Maximum Likelihood optimization failed to "

print(model\_arima.summary())

SARIMAX Results   
==============================================================================  
Dep. Variable: Demand No. Observations: 10  
Model: ARIMA(2, 1, 1) Log Likelihood -73.174  
Date: Sun, 29 Oct 2023 AIC 154.349  
Time: 13:57:39 BIC 155.138  
Sample: 02-01-2012 HQIC 152.646  
 - 11-01-2012   
Covariance Type: opg   
==============================================================================  
 coef std err z P>|z| [0.025 0.975]  
------------------------------------------------------------------------------  
ar.L1 -0.6329 0.455 -1.391 0.164 -1.525 0.259  
ar.L2 -0.2624 0.853 -0.308 0.758 -1.934 1.409  
ma.L1 -0.9997 0.644 -1.553 0.120 -2.261 0.262  
sigma2 4.899e+05 1.31e-06 3.73e+11 0.000 4.9e+05 4.9e+05  
===================================================================================  
Ljung-Box (L1) (Q): 0.35 Jarque-Bera (JB): 0.63  
Prob(Q): 0.55 Prob(JB): 0.73  
Heteroskedasticity (H): 4.17 Skew: -0.58  
Prob(H) (two-sided): 0.27 Kurtosis: 2.45  
===================================================================================  
  
Warnings:  
[1] Covariance matrix calculated using the outer product of gradients (complex-step).  
[2] Covariance matrix is singular or near-singular, with condition number 2.57e+27. Standard errors may be unstable.

*# predict Demand for the next year*  
future = model\_arima.predict(start = '2013-01-01', end = '2013-12-01')  
*# Getting only the value of zeroth index since the diff() operation looses first value.*  
future.iloc[0] = df\_month.query('index == "2013-01-01"')['Demand']  
*#cumsum Return cumulative sum over a DataFrame or Series axis*  
future = future.cumsum()

**#Predicting Next Year Demand**

fig, ax = plt.subplots(figsize=(15, 5))  
df\_month.Demand.plot(ax=ax, label='Product Demand', title='Product Demand/ Future prediction')  
future.plot(ax=ax, label='Future')  
ax.legend(['Product demand', 'Future prediction'])  
plt.show()

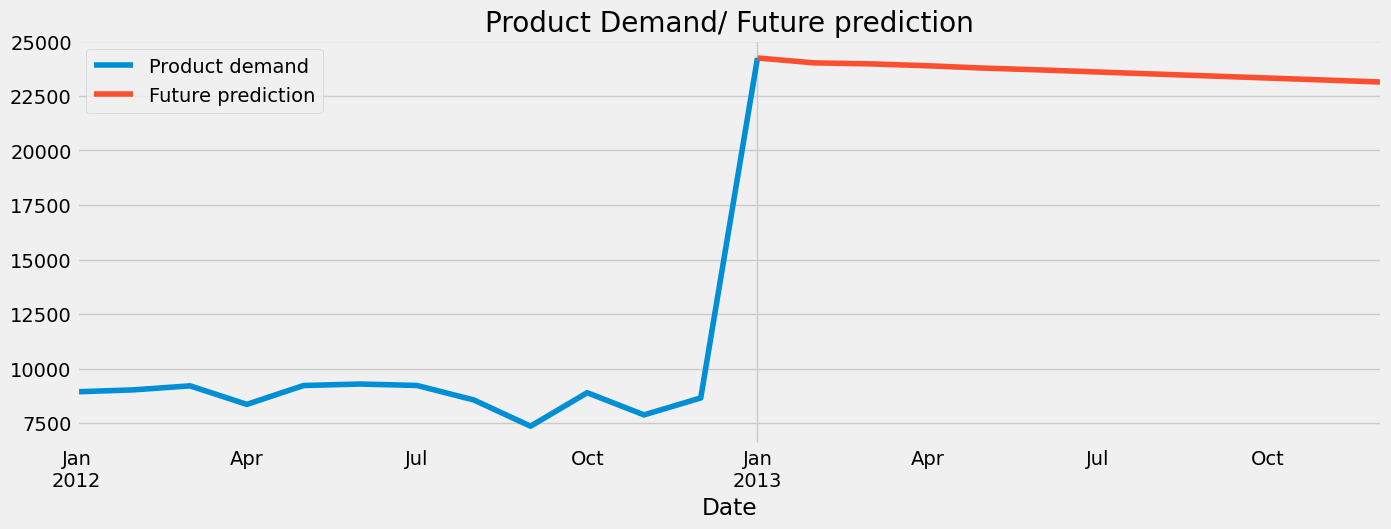


Figure 12: Future Prediction

**Feature extraction:**

Feature extraction is the process of transforming raw data into features that are more informative and predictive. This is an important step in building any machine learning model, including ARIMA models.

Some common feature extraction techniques for product demand forecasting include:

* Trend features: These features capture the overall trend in the data, such as the linear slope or the exponential growth rate.
* Seasonal features: These features capture the cyclical nature of the data, such as the weekly or monthly seasonality.
* Event features: These features capture one-time events that may have affected the data, such as a marketing campaign or a natural disaster.
* Holiday features: These features capture the impact of holidays on demand.
* Economic features: These features capture the impact of economic factors on demand, such as GDP growth and unemployment rates.

**ARIMA model**

ARIMA (Autoregressive Integrated Moving Average) is a statistical forecasting model that is often used for product demand forecasting. ARIMA models are based on the assumption that future demand can be predicted from past demand and historical factors.

ARIMA models are characterized by three parameters:

* Autoregressive (AR) order: This parameter specifies the number of previous demand values that are used to predict future demand.
* Integrated (I) order: This parameter specifies the number of times the data needs to be differenced to make it stationary (i.e., to remove any trend or seasonality).
* Moving average (MA) order: This parameter specifies the number of previous forecast errors that are used to predict future forecast errors.

The ARIMA model is fitted to the historical demand data to estimate the AR, I, and MA parameters. Once the model is fitted, it can be used to forecast future demand by recursively applying the following equation:

F\_t = c + AR\_1 \* F\_t-1 + AR\_2 \* F\_t-2 + ... + AR\_p \* F\_t-p - MA\_1 \* e\_t-1 - MA\_2 \* e\_t-2 - ... - MA\_q \* e\_t-q

where:

* F\_t is the forecast for demand at time t
* c is a constant term
* AR\_1, AR\_2, ..., AR\_p are the autoregressive parameters
* MA\_1, MA\_2, ..., MA\_q are the moving average parameters
* e\_t is the forecast error at time t

**Evaluation:**

Once the ARIMA model has been fitted and used to forecast future demand, it is important to evaluate the model's performance. This can be done by calculating various evaluation metrics, such as:

* **Mean absolute error (MAE):** This metric measures the average absolute difference between the actual demand and the forecasted demand.
* **Mean squared error (MSE):** This metric measures the average squared difference between the actual demand and the forecasted demand.
* **Root mean squared error (RMSE):** This metric is the square root of the MSE. It is a good measure of the overall forecasting accuracy.
* **Mean absolute percentage error (MAPE):** This metric measures the average absolute error divided by the actual demand. It is a good measure of the forecasting accuracy relative to the size of the demand.

The evaluation metrics can be used to compare the performance of different ARIMA models or to compare the performance of the ARIMA model to other forecasting models.

**Conclusion:**

ARIMA models are a popular and effective tool for product demand forecasting. By carefully extracting features from the historical demand data and fitting the ARIMA model to the data, businesses can obtain accurate forecasts of future demand. This information can be used to make better decisions about inventory management, production planning, and marketing campaigns.