**Project Title: Product Demand Predictor with Machine Learning**

**Introduction:**

In this phase we are document the project codes and we prepared for submission. A product company plans to offer discounts on its product during the upcoming holiday season. The company wants to find the price at which its product can be a better deal compared to its competitors. For this task, the company provided a dataset of past changes in sales based on price changes. You need to train a model that can predict the demand for the product in the market with different price segments.

**Project Overview:**

Develop a machine learning model to predict product demand based on historical sales data, economic indicators, and other relevant factors. This project aims to help businesses optimize inventory, reduce overstock and understock situations, and improve customer satisfaction by ensuring products are available when needed.

**Project Steps**:

Data Collection:

Gather historical sales data for the products you want to predict demand for.

Collect relevant data sources such as economic indicators (e.g., GDP, inflation, unemployment), seasonality, promotions, and any other factors that might influence demand.

Data Preprocessing:

Clean and preprocess the data, handling missing values, outliers, and data transformations.

Create features like time-based features (e.g., day of the week, month, year), lag variables, and any other relevant features.

Data Exploration:

Perform exploratory data analysis (EDA) to understand the distribution of data, correlations, and trends.

Visualize the data to gain insights into historical demand patterns and any relationships with external factors.

Feature Engineering:

Engineer relevant features based on the insights gained during EDA.

Create a target variable (demand) and define the prediction horizon (e.g., daily, weekly, monthly).

Model Selection:

Choose an appropriate machine learning algorithm for demand prediction. Common choices include:

Time series models (e.g., ARIMA, SARIMA, Prophet)

Regression models (e.g., Linear Regression, Random Forest, XGBoost)

Deep learning models (e.g., LSTM, GRU)

Experiment with multiple models and compare their performance.

Model Training:

Split the data into training, validation, and test sets.

Train the selected model on the training data.

Tune hyperparameters using the validation set to optimize model performance.

Model Evaluation:

Evaluate the model's performance using appropriate metrics (e.g., Mean Absolute Error, Root Mean Squared Error, Mean Absolute Percentage Error).

Compare the model's performance against a baseline or existing methods.

Deployment:

Deploy the trained model in a production environment where it can be used to make real-time predictions.

Implement a mechanism to regularly update the model with new data.

Monitoring and Maintenance:

Continuously monitor the model's performance in the production environment.

Retrain the model periodically to adapt to changing demand patterns and external factors.

Visualization and Reporting:

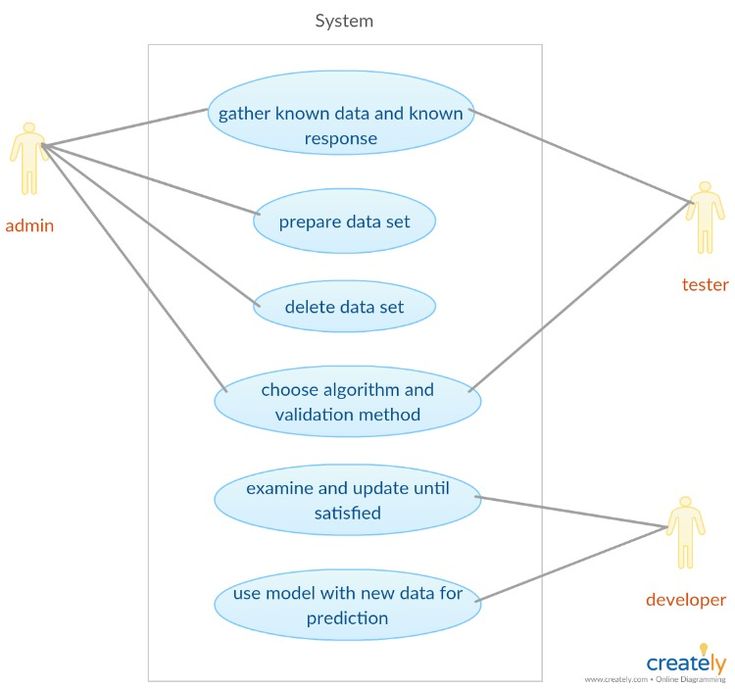
Create dashboards or reports to visualize the predicted demand, actual sales, and any relevant external factors.

Provide actionable insights to stakeholders.

Future Enhancements:

Consider future enhancements, such as incorporating additional data sources, improving the model's forecasting horizon, and adding anomaly detection features.

**Use Case Diagram :**



**Step by step Procedure:**

Step 1:

**Python Codes:**

import pandas as pd

from fbprophet import Prophet

import matplotlib.pyplot as plt

data = pd.DataFrame({

'ds': pd.date\_range(start='2022-01-01', periods=365, freq='D'),

'y': [100 + i + 10 \* i \*\* 0.5 + 5 \* (i % 7 == 0) for i in range(365)]})

model = Prophet()

model.fit(data)

future = model.make\_future\_dataframe(periods=30)

forecast = model.predict(future)

fig = model.plot(forecast)

plt.title('Product Demand Forecast')

plt.xlabel('Date')

plt.ylabel('Demand')

plt show()

**Output:**

**ID Store ID Total Price Base Price Units Sold**

**0 1 8091 99.0375 111.8625 20**

**1 2 8091 99.0375 99.0375 28**

**2 3 8091 133.9500 133.9500 19**

**3 4 8091 133.9500 133.9500 44**

1. **5 8091 141.0750 141.0750 52**
2. Now let’s have a look at whether this dataset contains any null values or not:

1

data.isnull().sum();

**ID 0**

**Store ID 0**

**Total Price 1**

**Base Price 0**

**Units Sold 0**

**dtype: int64**

1. So the dataset has only one missing value in the **Total Price** column, I will remove that entire row for now:

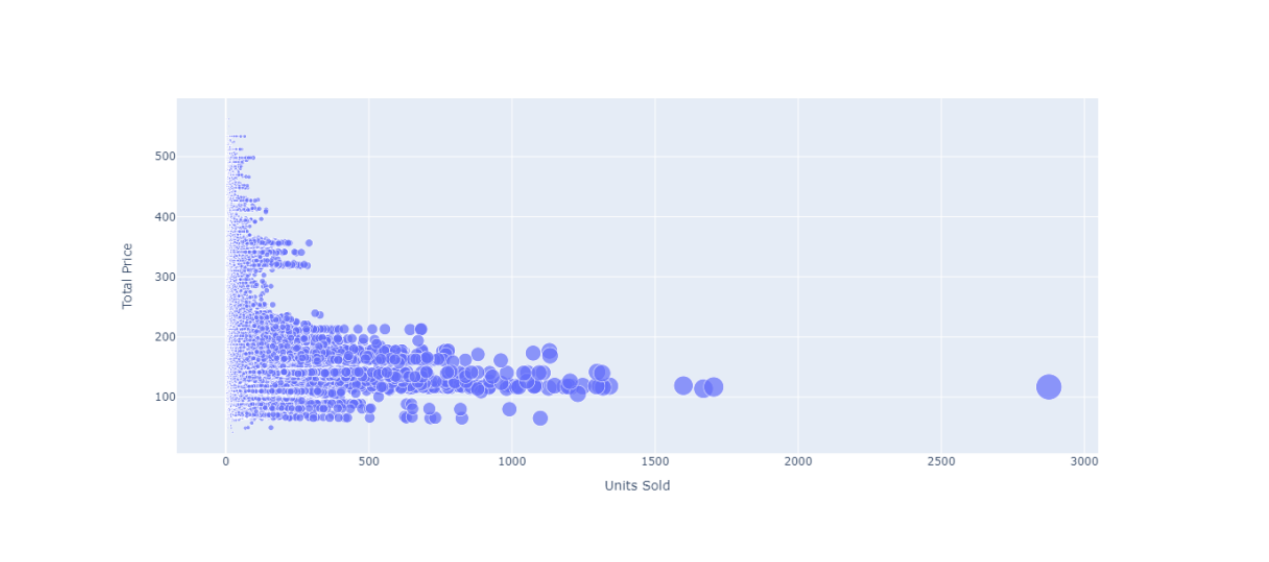
Let us now analyze the relationship between the price and the demand for the product. Here I will use a [**scatter plot**](https://thecleverprogrammer.com/2020/12/20/scatter-plot-with-python/) to see how the demand for the product varies with the price change:

fig = px.scatter(data, x="Units Sold", y="Total Price",

size='Units Sold')

fig.show()

**Output:**



1. We can see that most of the data points show the sales of the product is increasing as the price is decreasing with some exceptions. Now let’s have a look at the correlation between the features of the dataset:

print(data.corr())

**Output:**

**ID Store ID Total Price Base Price Units Sold**

**ID 1.000000 0.007464 0.008473 0.018932 -0.010616**

**Store ID 0.007464 1.000000 -0.038315 -0.038848 -0.004372**

**Total Price 0.008473 -0.038315 1.000000 0.958885 -0.235625**

**Base Price 0.018932 -0.038848 0.958885 1.000000 -0.140032**

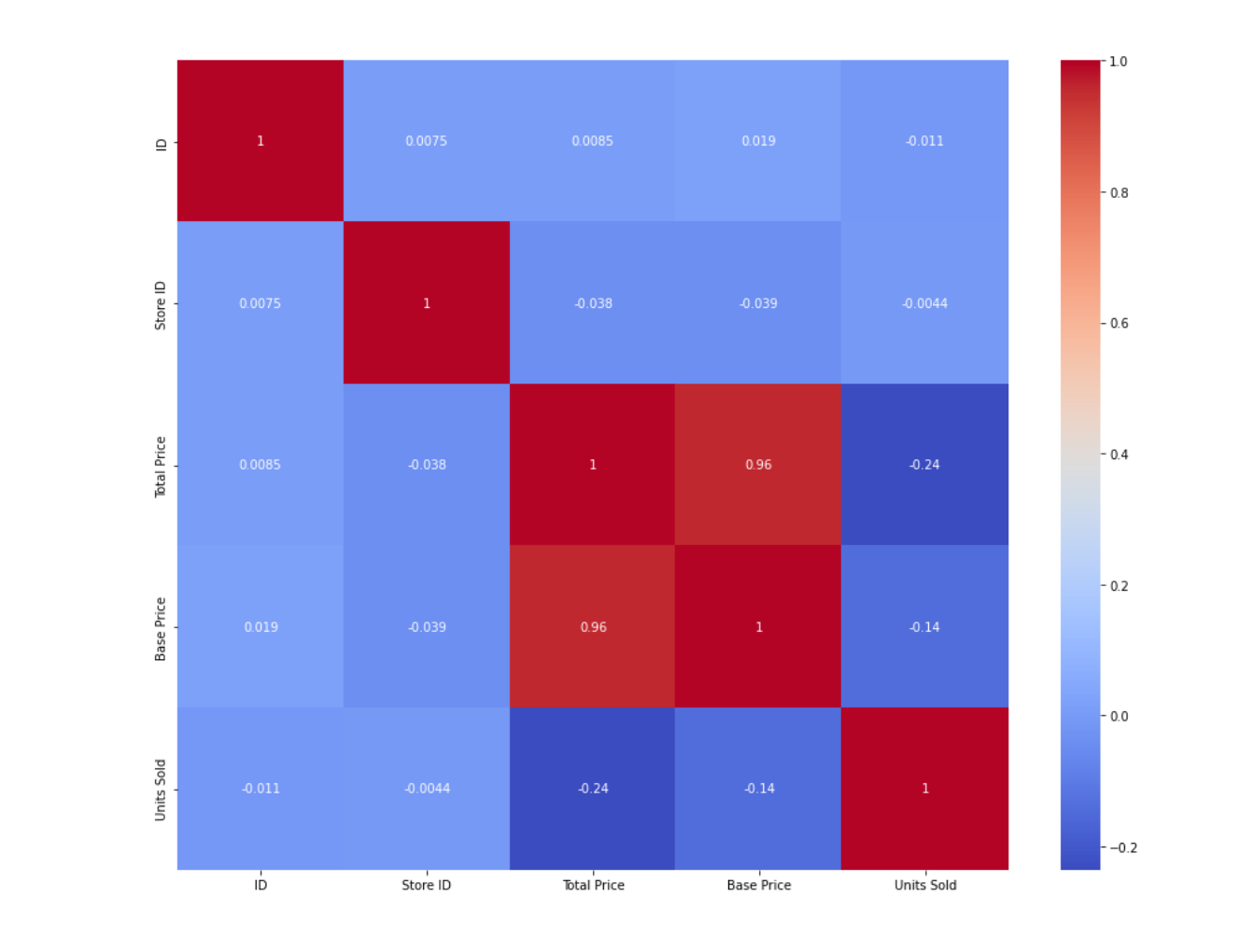
**Units Sold -0.010616 -0.004372 -0.235625 -0.140032 1.000000**

correlations = data.corr(method='pearson')

plt.figure(figsize=(15, 12))

sns.heatmap(correlations, cmap="coolwarm", annot=True)

plt.show()



Now let’s move to the task of training a machine learning model to predict the demand for the product at different prices. I will choose the **Total Price** and the **Base Price** column as the features to train the model, and the **Units Sold** column as labels for the model:

x = data[["Total Price", "Base Price"]]

y = data["Units Sold"]

Now let’s split the data into training and test sets and use the decision tree regression algorithm to train our model:

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2,

random\_state=42)

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(xtrain, ytrain)

Now let’s input the features **(Total Price, Base Price)** into the model and predict how much quantity can be demanded based on those values:

#features = [["Total Price", "Base Price"]]

features = np.array([[133.00, 140.00]])

model.predict(features)

**Step 2:**

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

df.head(5)

**Step 3:**

Data Cleaning and Preprocessing

| Product\_Code | Warehouse | Product\_Category | Date | Order\_Demand |
| --- | --- | --- | --- | --- |
| count | 1048575 | 1048575 | 1048575 | 1037336 | 1048575 |
| unique | 2160 | 4 | 33 | 1729 | 3828 |
| top | Product\_1359 | Whse\_J | Category\_019 | 2013/9/27 | 1000 |
| freq | 16936 | 764447 | 481099 | 2075 | 112682 |

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1048575 entries, 0 to 1048574

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Product\_Code 1048575 non-null object

1 Warehouse 1048575 non-null object

2 Product\_Category 1048575 non-null object

3 Date 1037336 non-null object

4 Order\_Demand 1048575 non-null object

dtypes: object(5)

memory usage: 40.0+ 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: 1

Percentage of missing values: 1.0718355863910547

As missing values are only 1% of the data and that too in just one attribute it is safe to remove them

linkcode

*# Dropping the missing values*

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

**Step3:**

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)

In [8]:

linkcode

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

df = df.set\_index('Date')

df.head()

**Step4:**

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)

**Yearly Average Demand for all the Product**

**Categories:**

****

****

****

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

bars = axes[i].bar(x, data, color=colors)

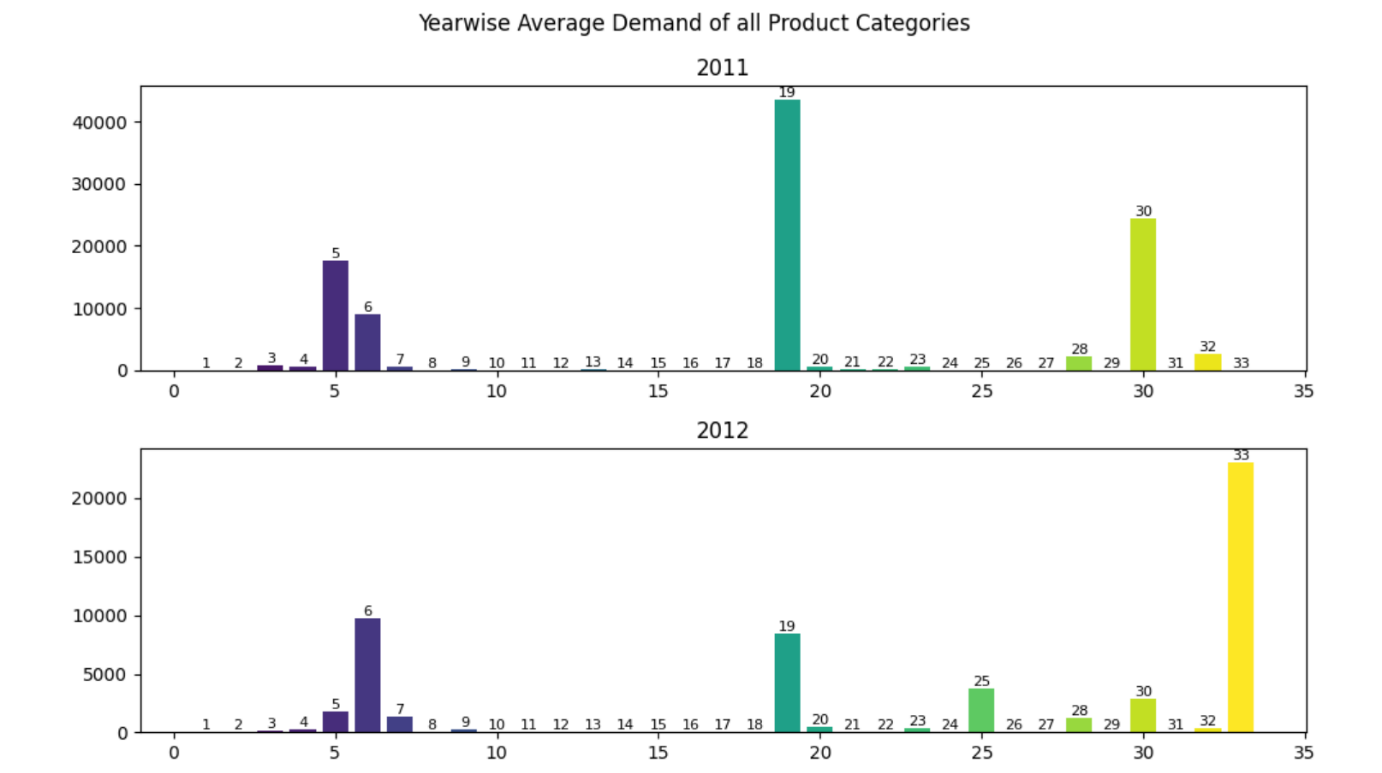
axes[i].bar\_label(bars, labels=x, fontsize = 8)

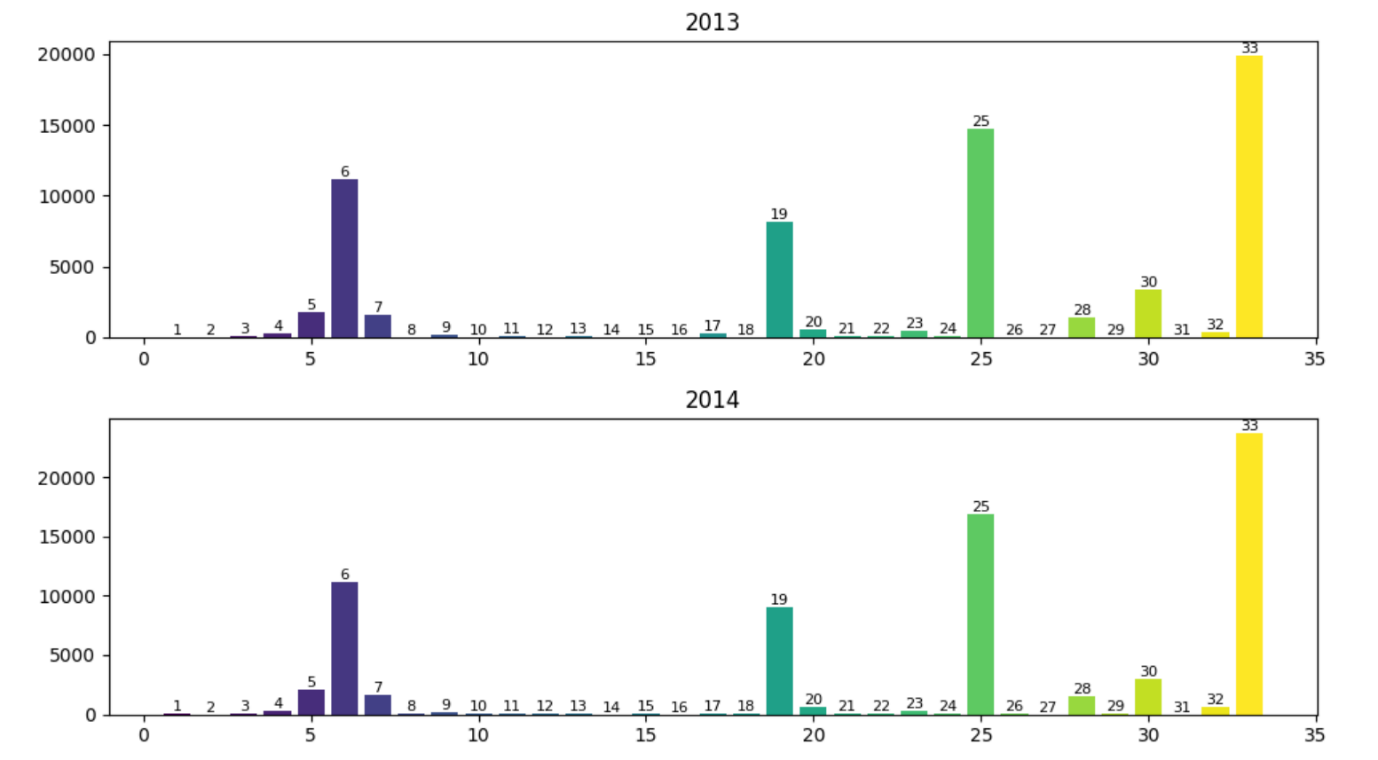
axes[i].set\_title(year)

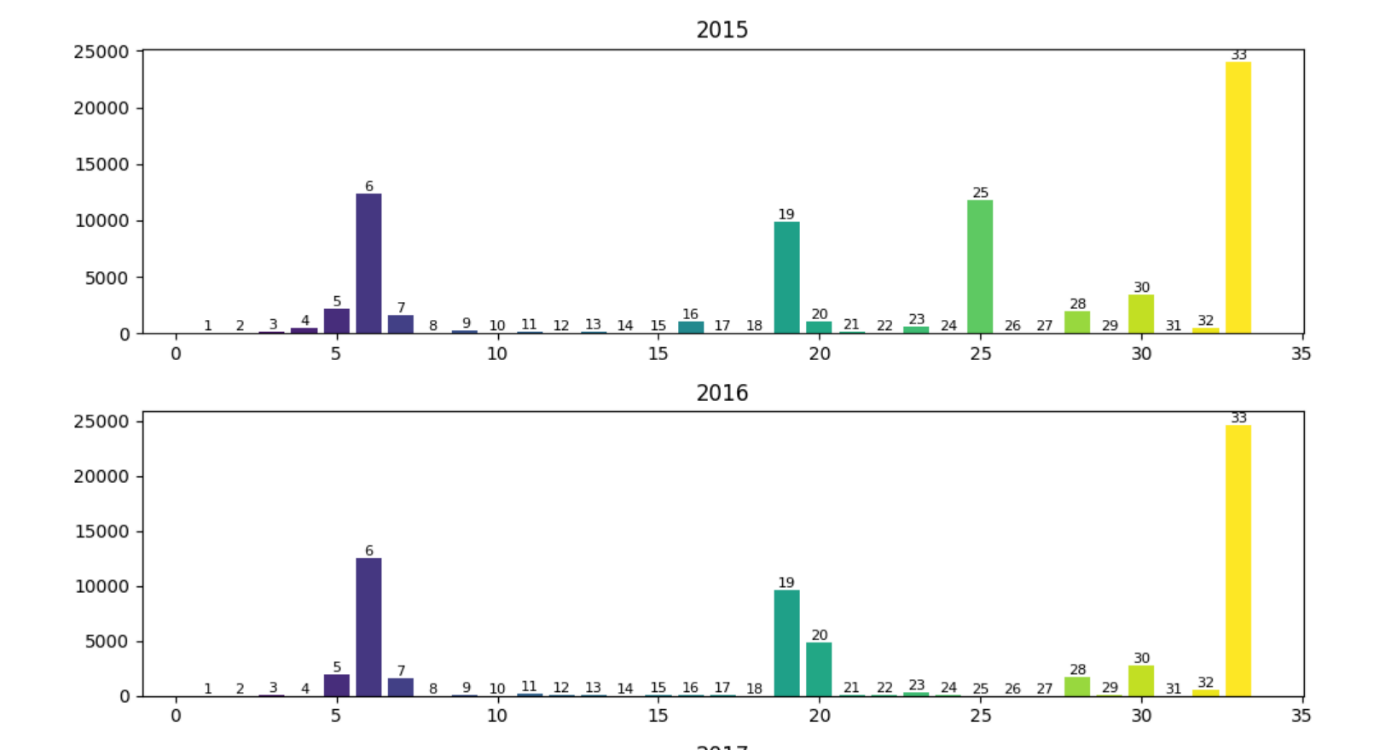
plt.tight\_layout()

fig.suptitle("Yearwise Average Demand of all Product Categories", y=1.01)

plt.show()

****

****

****

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):

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')

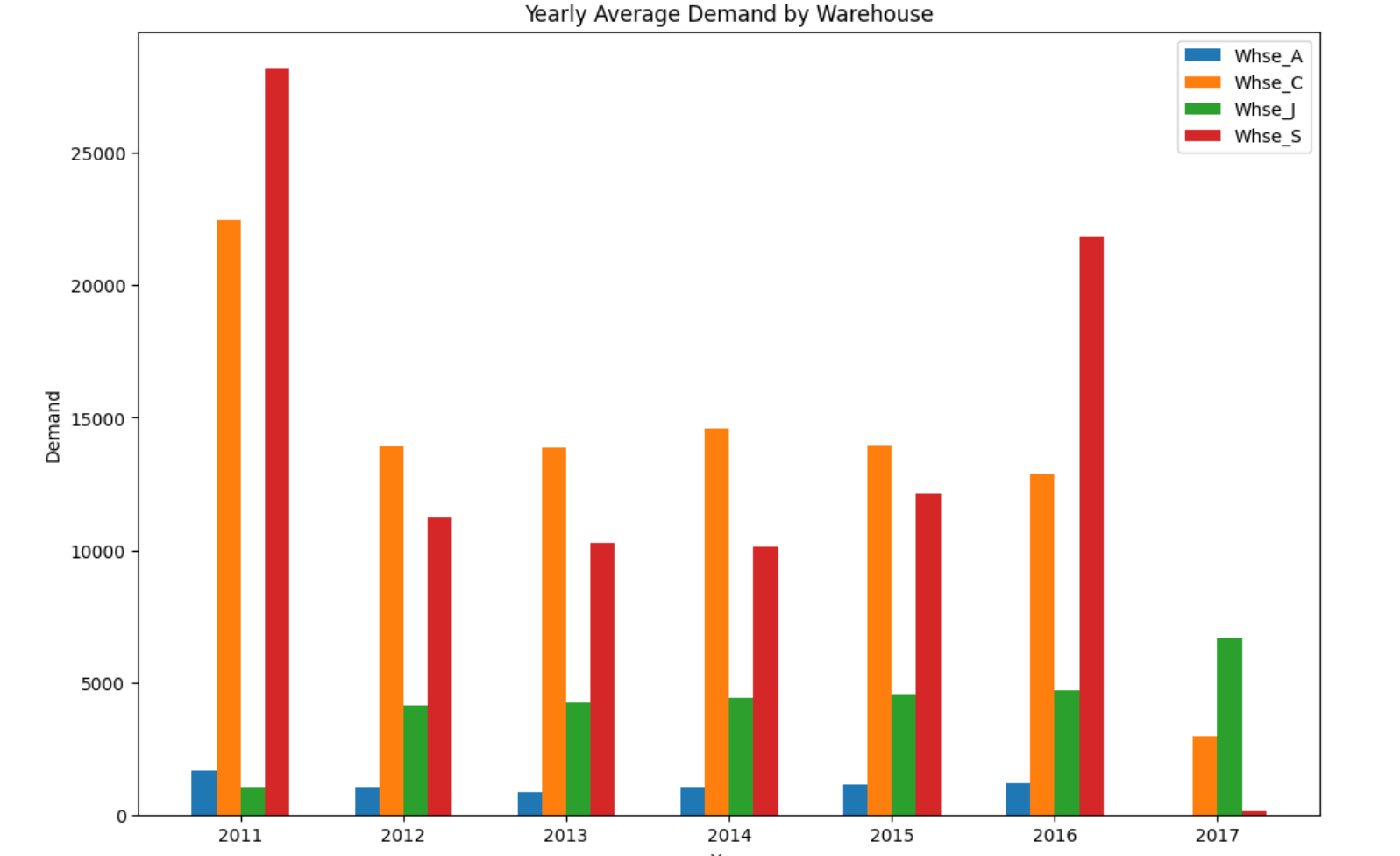
ax.set\_title('Yearly Average Demand by Warehouse')

*# Add a legend to distinguish the warehouses*

ax.legend()

*# Show the chart*

plt.show()

****

**Conclusion:**

In conclusion, a product demand predictor is a critical tool for businesses, enabling them to forecast demand, optimize inventory, enhance customer satisfaction, and make data-driven decisions. Despite its challenges, an effective demand predictor can provide a competitive advantage in an evolving marketplace, making it an invaluable asset for organizations seeking operational efficiency and success.