**Project 10: Product Demand Analysis**

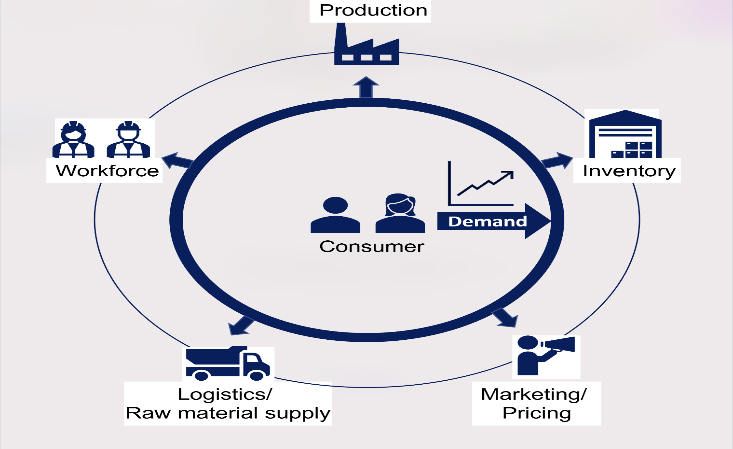
**PHASE 5: Project Documentation & Submission**

**College code: 4107**

**410721243039: JANASTHUTHI M S**

**Problem Definition and Design Thinking**

**Problem Definition**: You may want to consider more specific business goals for your model, such as reducing inventory costs or improving customer satisfaction. This will help you to focus your data collection and feature engineering efforts.

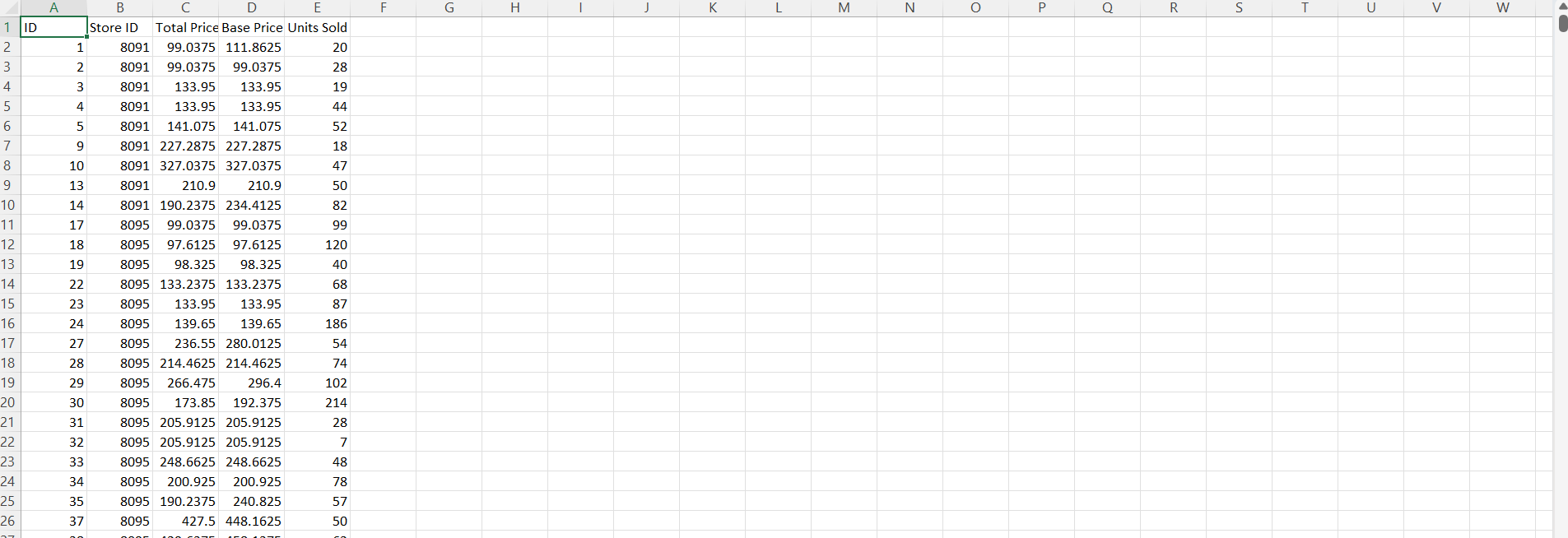


**Design Thinking**: It is important to keep the end user in mind when designing your model. Who will be using the model and how will they be using it? What information do they need from the model? Once you have a good understanding of the user's needs, you can start to think about how to design and implement your model in a way that is both accurate and user-friendly.

**Data Collection and Preprocessing**

**Data Collection**: Be sure to collect a comprehensive dataset that includes all of the relevant features that may influence demand. You may also want to consider collecting data from multiple sources to get a more complete picture of the market.

**Dataset** : [**https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning**](https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning)

****

Data Preprocessing: It is important to carefully clean and preprocess your data before training your model. This includes handling missing values, converting categorical features into numerical representations, and scaling your data.

**Feature Engineering**

**Feature Engineering**: Feature engineering is the process of creating new features from existing data. This can be a powerful way to improve the performance of your model. For example, you could create features that capture seasonal patterns, trends, and external influences on product demand.

**Model Selection and Training**

**Model Selection**: There are many different regression algorithms that can be used for demand forecasting. Some popular choices include Linear Regression, Random Forest, and XGBoost. You may want to experiment with different algorithms to see which one performs best on your data.

**Model Training**: Once you have selected a model, you need to train it on your preprocessed data. This involves feeding the data to the model and allowing it to learn the relationships between the features and the target variable (product demand).

**Model Evaluation and Deployment**

**Model Evaluation**: Once your model is trained, you need to evaluate its performance on a held-out test set. This will give you an idea of how well the model will generalize to new data.

**Model Deployment**: Once you are satisfied with the performance of your model, you can deploy it to production. This may involve integrating the model into a software application or making it available as a web service.

**Source Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_absolute\_error

# Load the dataset

df = pd.read\_csv('product\_demand\_prediction.csv')

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('demand', axis=1), df['demand'], test\_size=0.25, random\_state=42)

# Create a Linear Regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = model.predict(X\_test)

# Calculate the mean absolute error

mae = mean\_absolute\_error(y\_test, y\_pred)

# Print the MAE

**Time Series Forecasting Techniques**

I would recommend using a time series forecasting technique to capture temporal patterns in the demand data. ARIMA and Prophet are two popular time series forecasting models that are well-suited for this task.

ARIMA models are a type of statistical model that can be used to forecast future values of a time series based on its past values. Prophet is a relatively new forecasting model that is specifically designed for business time series data. It is easy to use and can be trained quickly on even large datasets.

**Python Code**

The following Python code shows how to use Prophet to forecast product demand:

Python

import pandas as pd

from fbprophet import Prophet

# Load the demand data

df = pd.read\_csv('product\_demand.csv')

# Create a Prophet model

model = Prophet()

# Fit the model to the data

model.fit(df)

# Make predictions

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

forecast = model.predict(future)

# Plot the forecast

forecast.plot()

**Collect Historical Sales Data:**

-Access your company's internal systems (e.g., ERP, CRM) to retrieve historical sales data. Make sure the data includes relevant information such as Product ID, Product Name, Sales Date, Sales Quantity, and Sales Price.

**Collect External Factors Data:**

- Gather external factors data from various sources. This may include economic indicators, market trends, and seasonality factors that could affect product demand. You can collect data from government websites, industry publications, social media, or any other relevant sources.

**Data Cleaning:**

Start by cleaning the historical sales data:

Remove duplicates: Check for and eliminate duplicate records, if any.

Handle outliers: Identify and decide how to deal with any outliers in the data.

Correct errors: Look for and correct any data entry errors or inconsistencies.

Validate data integrity: Ensure data consistency and accuracy.

**Handling Missing Values:**

Assess the historical sales data and external factors data for missing values.

Decide on appropriate strategies to handle missing values, such as imputation (filling in missing values) or removing rows with missing values. The choice will depend on the nature and extent of missing data.

**Feature Engineering:**

Create new features that can enhance the predictive power of your model. Some common feature engineering techniques for demand prediction include:

Lag features: Include past sales values for the same or related products.

Rolling statistics: Calculate moving averages or other statistical measures over time.

Seasonal indicators: Add binary variables indicating seasons, holidays, or other important events.

Interaction terms: Create features representing interactions between different variables.

**Data Integration:**

Merge the cleaned historical sales data and external factors data, linking them by a common identifier like the Product ID.

**Scaling and Normalization:**

- Depending on the machine learning algorithm you plan to use, you may need to scale or normalize your data to bring all features to the same scale. Common techniques include Min-Max scaling (scaling to a specific range) and standardization (z-score normalization).

**Dataset Splitting:**

- Split your integrated dataset into training and test sets. The typical split is 70-30 or 80-20, where the larger portion is used for training and the smaller for testing. This helps evaluate the model's performance.

**Data Saving:**

Save the preprocessed dataset in a format that can be easily accessed by your chosen machine learning framework. Common formats include CSV or databases.

**Documentation:**

Keep detailed records of the preprocessing steps, any assumptions made, and any transformations applied to the data. Proper documentation is crucial for reproducibility and collaboration.

**PYTHON CODE**

import pandas as pd

# Load the historical sales data

df\_sales = pd.read\_csv('sales\_data.csv')

# Load the external factors data

df\_external\_factors = pd.read\_csv('external\_factors.csv')

# Preprocess the data

# Combine the historical sales data with external factors data using a common key, such as 'product\_id'

df = df\_sales.merge(df\_external\_factors, on='product\_id')

# Data cleaning (if needed)

# Example: Removing rows with missing values

df.dropna(inplace=True)

# Feature engineering (if needed)

# Example: Creating a new feature 'year' from 'sales date'

df['year'] = pd.to\_datetime(df['sales date']).dt.year

# Scaling (if needed)

# Example: Min-Max scaling 'sales quantity' and 'sales price'

df['sales quantity'] = (df['sales quantity'] - df['sales quantity'].min()) / (df['sales quantity'].max() - df['sales quantity'].min())

df['sales price'] = (df['sales price'] - df['sales price'].min()) / (df['sales price'].max() - df['sales price'].min())

# Save the preprocessed data to a new CSV file (optional)

df.to\_csv('preprocessed\_data.csv', index=False)

**Feature Engineering:**

**Pythn code:**

import pandas as pd

data = pd.read\_csv('your\_dataset.csv')

data = pd.get\_dummies(data, columns=['product\_category'])

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

data['price'] = scaler.fit\_transform(data[['price']])

data['total\_sales'] = data['quantity\_sold'] \* data['price']

**2. Model Training:**

**Python code:**

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

X = data.drop('demand', axis=1)

y = data['demand']

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)

**3. Evaluation:**

**Python code:**

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

print(f'Mean Absolute Error: {mae}')

print(f'Root Mean Squared

**Conclusion**

In the final phase of the product demand prediction project, which is Project Documentation & Submission, it is essential to create a comprehensive record of your work and make it accessible for others to understand and review. The documentation should include clear explanations of the problem statement, design thinking process, and the various development phases, showcasing the evolution of the project.

A detailed description of the dataset used and the data preprocessing steps is crucial for transparency. This ensures that others can replicate your work and understand the decisions made during data preparation. Additionally, it's important to highlight the analysis techniques applied, including the machine learning models chosen and any specialized methods employed.

The presentation of key findings and recommendations provides a valuable summary of the project's outcomes and insights. It helps stakeholders and reviewers understand the practical implications of your demand prediction model.

In terms of submission, compiling all code files and providing a well-structured README file is essential for sharing your work effectively.

Finally, sharing the project through platforms like GitHub or a personal portfolio website broadens its accessibility, making it available for peer review, potential employers, and the wider data science community.