# Chapter 1
# Introduction

## 1.1) ABSTRACT:

This report's main goal is to provide an overview of the process of creating a sales prediction model and offer insights into the possible benefits it may have for businesses. The purpose of this report is to emphasize the value of applying predictive analytics in the sales domain and to provide an approach for building and developing an accurate prediction model.

The project's theme centres around empowering businesses with a comprehensive toolset that will help them in making wise decisions. Its goal is to create a sales prediction model using a variety of machine-learning techniques and historical data.

The project will involve data preprocessing, feature engineering, and training of different machine learning algorithms to identify the most accurate prediction model.

The model's performance will be assessed using evaluation metrics such as mean squared error or mean absolute error.

A customizable dashboard with real-time monitoring and customizable analytics will be used to display these insights.

The project will provide valuable insights into sales prediction, helping businesses optimize their operations and improve profitability. The findings can be applied across various industries, from retail and e-commerce to manufacturing and finance. The ultimate objective is to create a robust and reliable prediction model that can support data-driven decision-making for sales forecasting.

## 1.2) INTRODUCTION:

The project "Prediction Model for Sales" is an innovative project that aimed at providing a solution for businesses to accurately forecast future sales. It helps in optimizing sales and help in decision-making processes. This project involved examining sales data, market trends, and other factors to create a machine-learning algorithm that is capable of predicting future sales accurately. The model aims to identify patterns, trends, and relationships in data to make predictions by taking into account variables such as previous sales performance, market conditions, customer behavior, marketing campaigns, pricing strategies, and any other relevant predictors and provide insights for sales performance. The prediction model can be used to forecast sales for different time horizons, such as monthly, quarterly, or yearly. The outcome of the project is the dashboard app providing real-time information, predictions, and notifications to optimize sales planning and to help businesses and organizations make informed decisions regarding production, inventory management, resource allocation, and revenue forecasting. Overall, this model provides valuable insights, promotes growth, and enables companies to make data-driven decisions that eventually increase sales and stay competitive in the ever-changing market landscape.

## 1.3) PROBLEM STATEMENT:

Sales forecasting is a critical component of business planning and decision-making. The company's current approach to sales forecasting mainly relies on the use of manual forecasting methods and historical sales data which is time-consuming and is likely to cause human errors. This method is unreliable and cannot capture the complexities and patterns involved in sales trends. Therefore, the company is facing trouble in planning its production, managing its inventory, and meeting client requests that's why the development of a robust and reliable predictive sales model becomes essential to enhance decision-making processes and improve overall business performance.

## 1.4) OBJECTIVES:

The objective of this project is to build a robust and accurate sales prediction model.

- It helps the organization in planning and in improving sales activities. It helps them to project future sales demand, identify market opportunities, and allocate resources accordingly.
- It will help companies in optimizing their sales processes and improve their overall efficiency.
- Businesses can optimize their inventory to ensure that products are available when customers desire them.
- Predictive sales forecasting provides insights and information that can help with strategic decision-making.

- Build advanced machine learning models that produce precise forecasts of future sales patterns by evaluating market trends, past sales data, and outside variables. This involves data preprocessing, feature engineering, and algorithm selection to ensure the reliability of the forecast.
- Designing a dashboard that would be easy to navigate and allow users to customize their analytics views.

## 1.5) BENEFITS:

- Companies can maximize their inventory management systems by precisely projecting future sales.
- Businesses can better target their marketing efforts to attract the right audience and increase conversion rates by knowing the behavior and preferences of their customers. Businesses can gain important insights to make well-informed decisions by using accurate sales forecasts that help in improved decision-making.
- A prediction model can make accurate forecasts and predictions this improves the accuracy of sales projections and helps organizations make better decisions.
- By accurately predicting sales, companies can efficiently allocate resources such as workforce, marketing budgets, and inventory. This helps in cost optimization by avoiding excess or insufficient allocation of resources.
- A prediction model can automate the sales forecasting process, saving time and effort. This allows sales teams to focus on customer engagement and relationship building, rather than spending excessive time on manual forecasting.

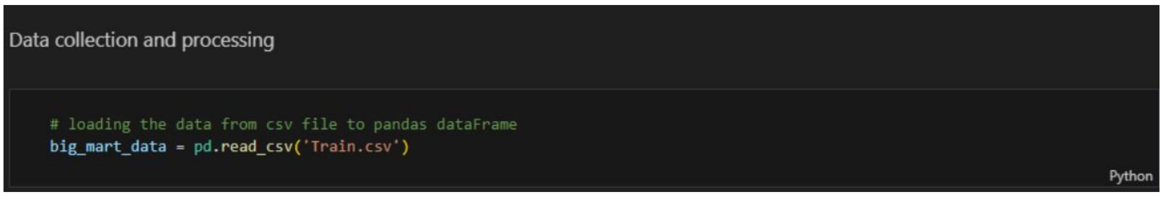
# Chapter 2

# Model Development

## 2.1) METHODOLOGY:

To develop the prediction model the methodology consisted of several steps:

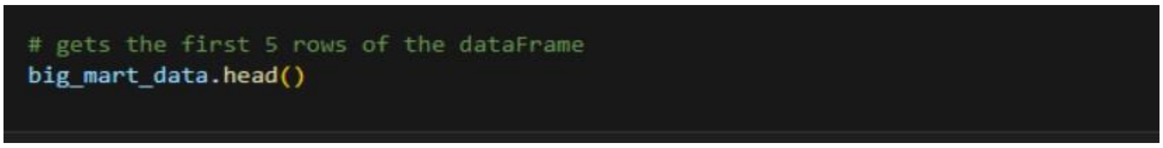
**2.1.1) Data Collection:** It is the process of gathering information or data from various sources to create a dataset that will be used to build a model. It involves collecting relevant and accurate data that is necessary to train and test the model effectively. The historical sales data that we gathered from Kaggle consists of multiple files or tables.Each file or table contains different columns that provide various features of the sales data. We load the data from a CSV file into a pandas DataFrame called "big_mart_data", which represents a dataset of Big Mart sales.



Data collection and processing

```python
# loading the data from csv file to pandas dataFrame
big_mart_data = pd.read_csv('Train.csv')
```

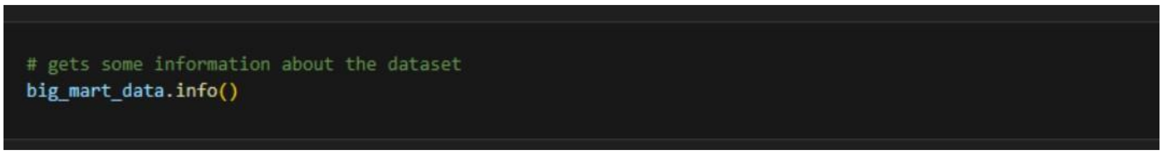
By using the .head()  function, only the first 5 rows of the DataFrame are displayed, allowing the user to get a quick overview of the data without having to view the entire dataset with each row representing a data point and each column representing a feature of that data point. It will display the values for each attribute (column) in the data.



```python
# gets the first 5 rows of the dataFrame
big_mart_data.head()
```

The .info()  function in Python is commonly used to retrieve information about a DataFrame, which is a two-dimensional labeled data structure from the pandas library.

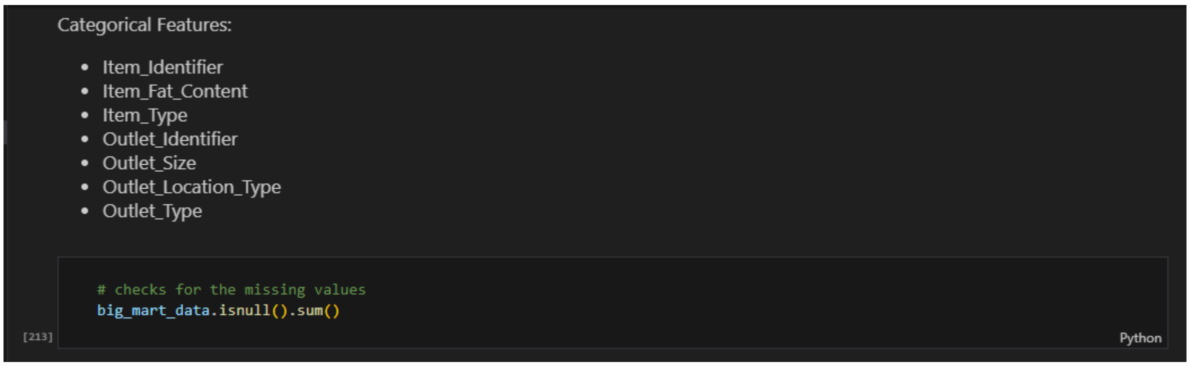
The output of the .info()  function returns a concise summary of the DataFrame, including the number of rows and columns, total memory usage, data types of each column, and the amount of non-null values.



```python
# gets some information about the dataset
big_mart_data.info()
```

**2.1.2) Preprocessing and Feature Selection:** We performed data preprocessing techniques to clean the data and prepare for analysis. This included identifying and correcting errors, handling any missing values or duplicates, handling noisy data, and conducting feature engineering, such as creating new variables. We used feature selection methods like step-wise regression, correlation analysis, and variable importance analysis to improve model performance and decrease dimensionality. The goal of this is to find the best set of features that allows one to build optimized models.

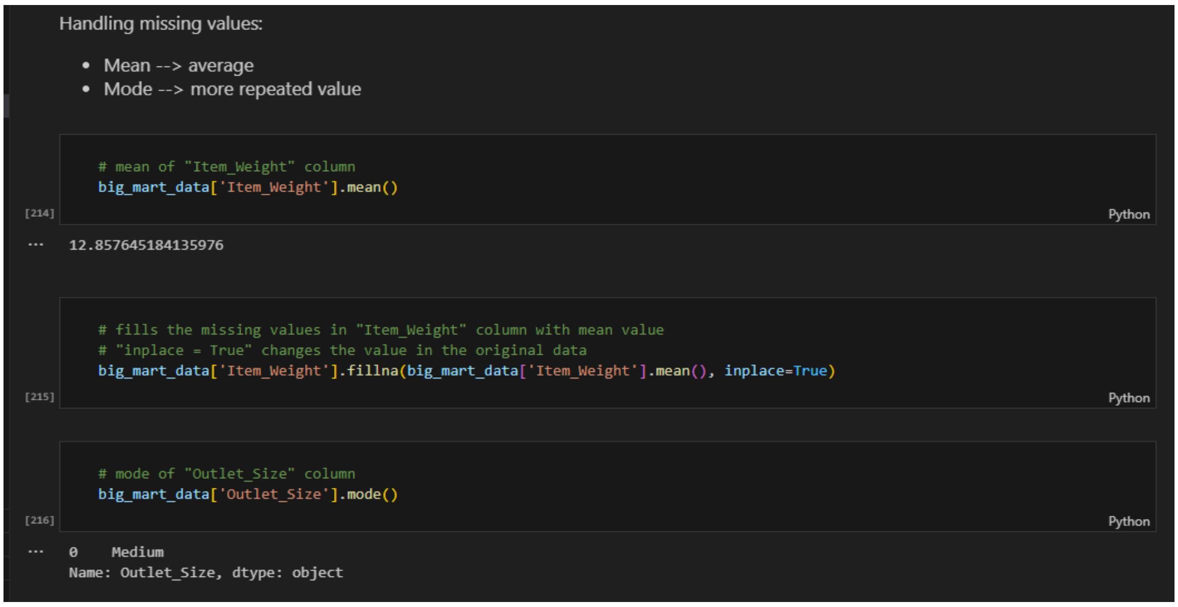
The code below checks for missing or null values in the data frame big_mart_data and then sums up the count of null values for each column. It returns a Series object containing the count of null values for each column in the data frame.


Categorical Features:

- Item_Identifier
- Item_Fat_Content
- Item_Type
- Outlet_Identifier
- Outlet_Size
- Outlet_Location_Type
- Outlet_Type

```
[213]
# checks for the missing values
big_mart_data.isnull().sum()
```

This fills missing values in item_weight column of the data frame big_mart_data with the mean value of that column. The inplace=True parameter ensures that the changes are appliead directly to the data frame.


Handling missing values:

- Mean --> average
- Mode --> more repeated value

```
[214]
# mean of "Item_Weight" column
big_mart_data['Item_Weight'].mean()
```

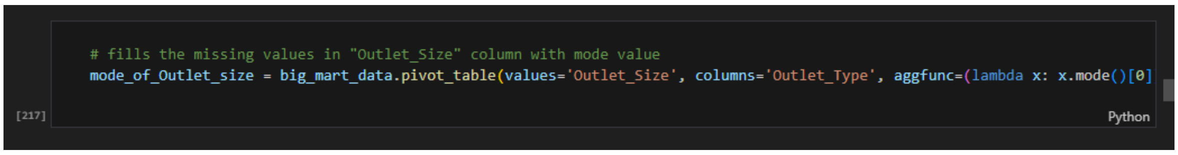
```
12.857645184135976
```

```
[215]
# fills the missing values in "Item_Weight" column with mean value
# "inplace = True" changes the value in the original data
big_mart_data['Item_Weight'].fillna(big_mart_data['Item_Weight'].mean(), inplace=True)
```

```
[216]
# mode of "Outlet_Size" column
big_mart_data['Outlet_Size'].mode()
```

```
0    Medium
Name: Outlet_Size, dtype: object
```

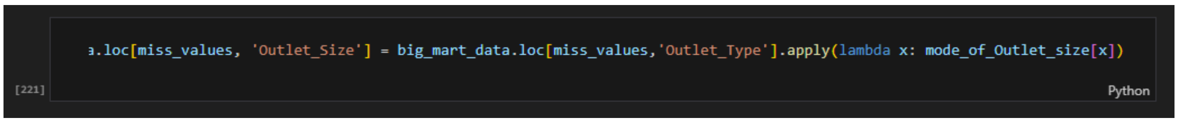
This creates a pivot table from the big_mart_data data frame, where the values are taken from the outlet_size column and grouped by outlet_type. The aggregation function used here is a lambda function that calculates the mode of each group and [0] is used to extract the first mode in case of multiple modes.



```
# fills the missing values in "Outlet_Size" column with mode value
mode_of_Outlet_size = big_mart_data.pivot_table(values='Outlet_Size', columns='Outlet_Type', aggfunc=(lambda x: x.mode()[0]
```

[217] Python

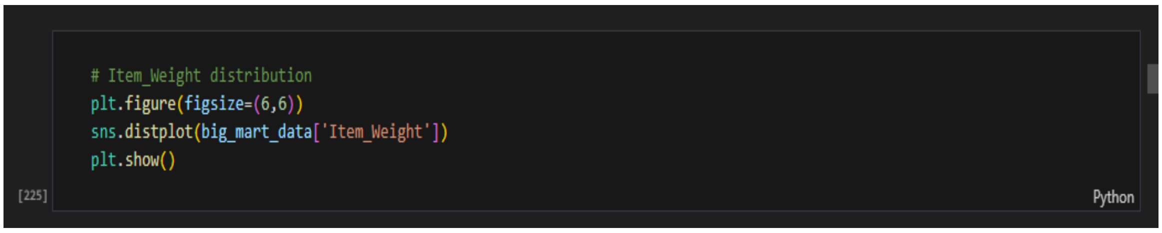
This line of code uses the loc indexer to locate rows in the outlet_size column of the data frame big_mart_data where missing values are present, based on the Boolean mask miss_values. It then assigns values to these rows using the loc indexer again to access the corresponding values from the outlet_type column. The apply method is used to apply a lambda function that maps each outlet_type value to its corresponding mode of outlet_size, as determined previously by the mode_of_outlet_size pivot table.



```
a.loc[miss_values, 'Outlet_Size'] = big_mart_data.loc[miss_values,'Outlet_Type'].apply(lambda x: mode_of_Outlet_size[x])
```

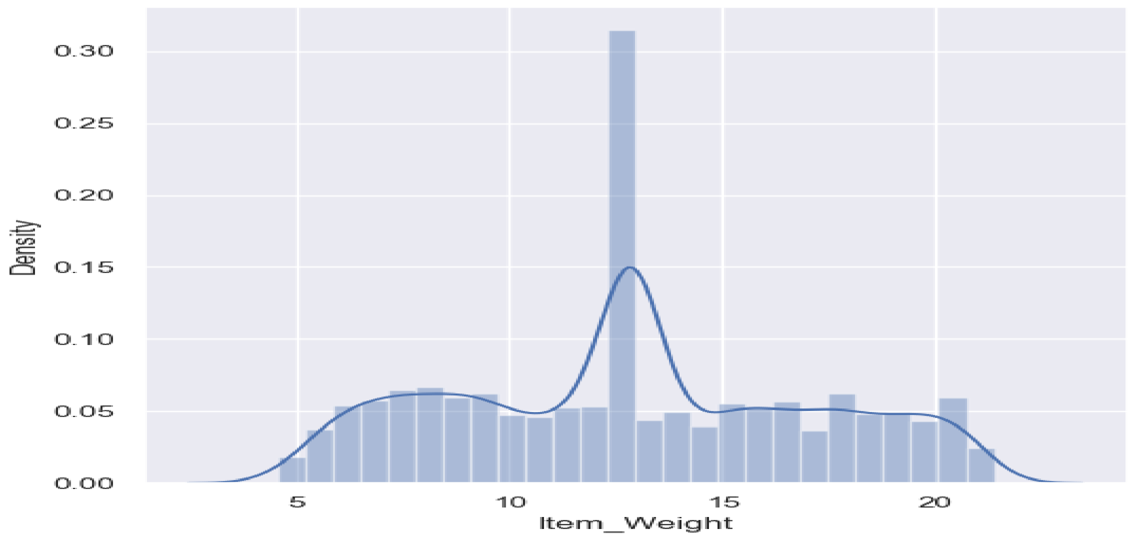
[221] Python

The resulting graph visually represents the distribution of weights of items in the dataset. The x-axis represents the range of item_weight, and the y-axis represents the density (probability) of occurrence for each weight. The shape of the curve provides insights into the distribution pattern, such as whether it is normally distributed, skewed or has multiple peaks.

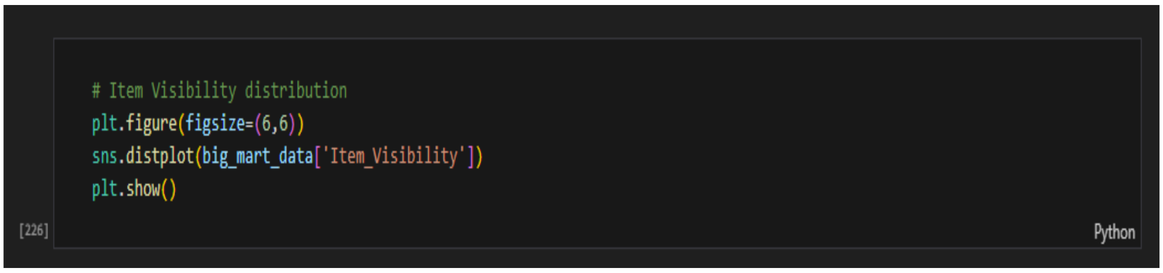


```
# Item_Weight distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Weight'])
plt.show()
```

[225] Python

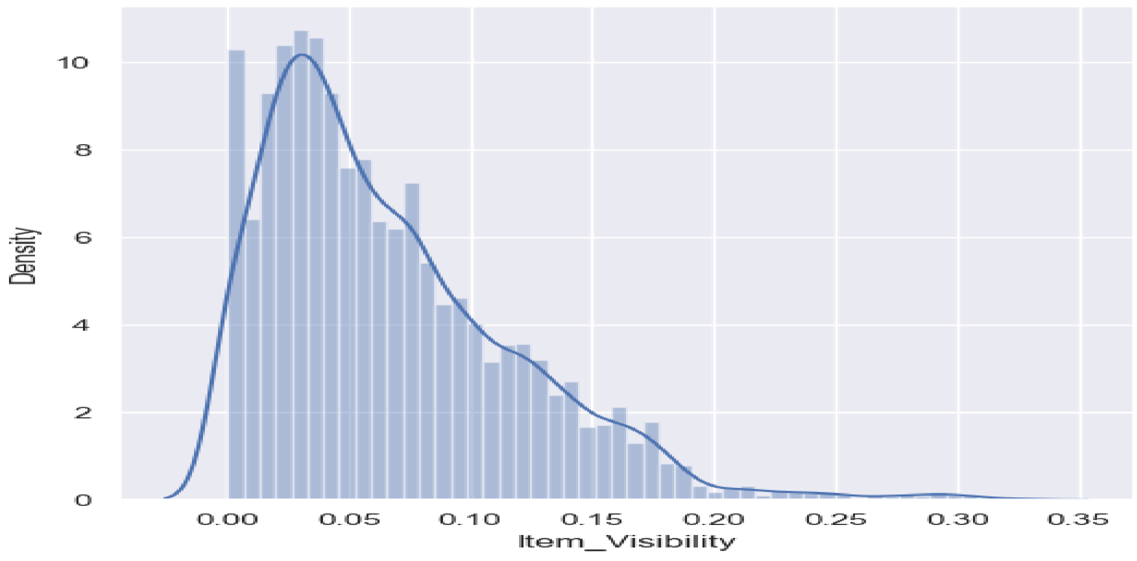


The resulting graph visually represents the distribution of visibility of items in the dataset. The x-axis represents the range of item_visibility, and the y-axis represents the density (probability) of occurrence for each visibility value. The shape of the curve provides insights into the distribution pattern, such as whether it is normally distributed, skewed or has multiple peaks.

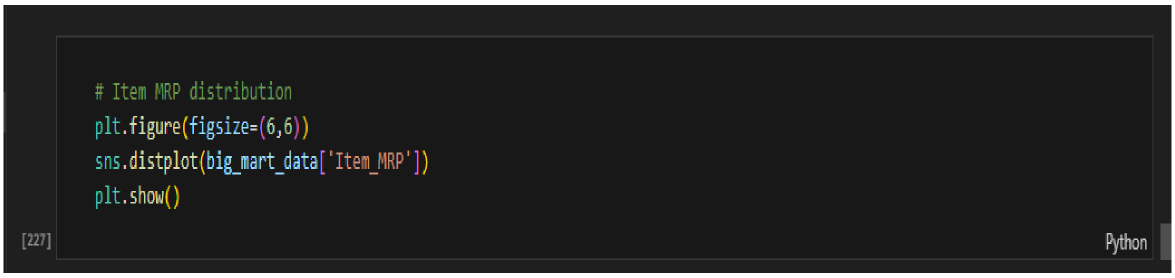


```
# Item Visibility distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Visibility'])
plt.show()
```

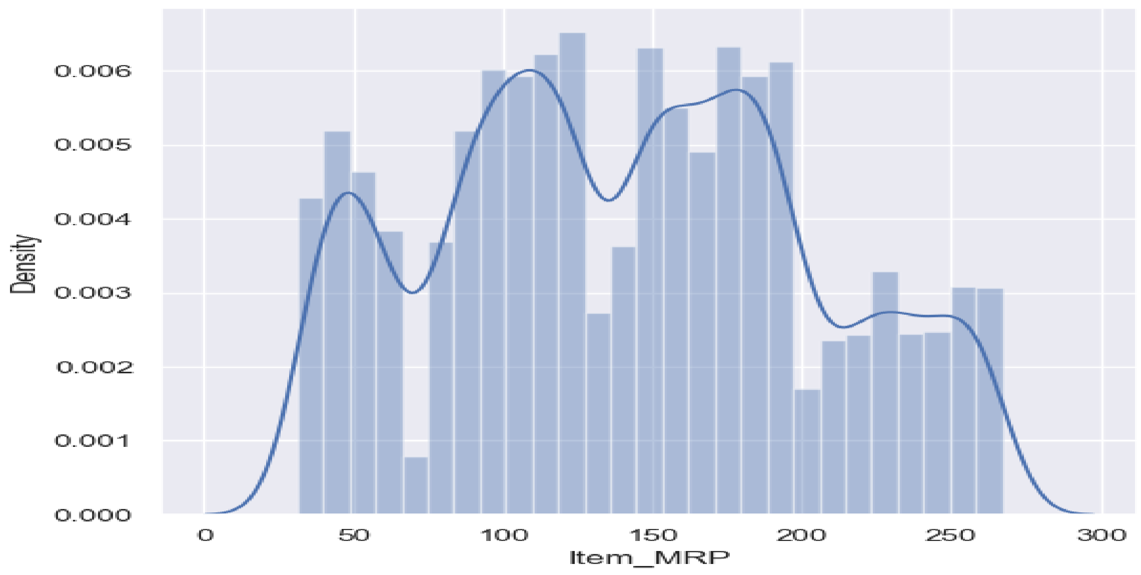
[226] Python



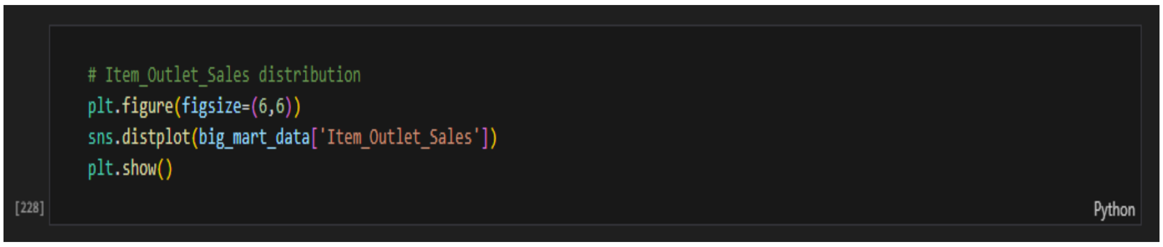
The resulting graph visually represents the distribution of the maximum retail price (MRP) of items in the dataset. The x-axis represents the range of MRP values, and the y-axis represents the density (probability) of occurrence for each MRP value. The shape of the curve provides insights into the distribution pattern, such as whether it is normally distributed, skewed, or has multiple peaks.



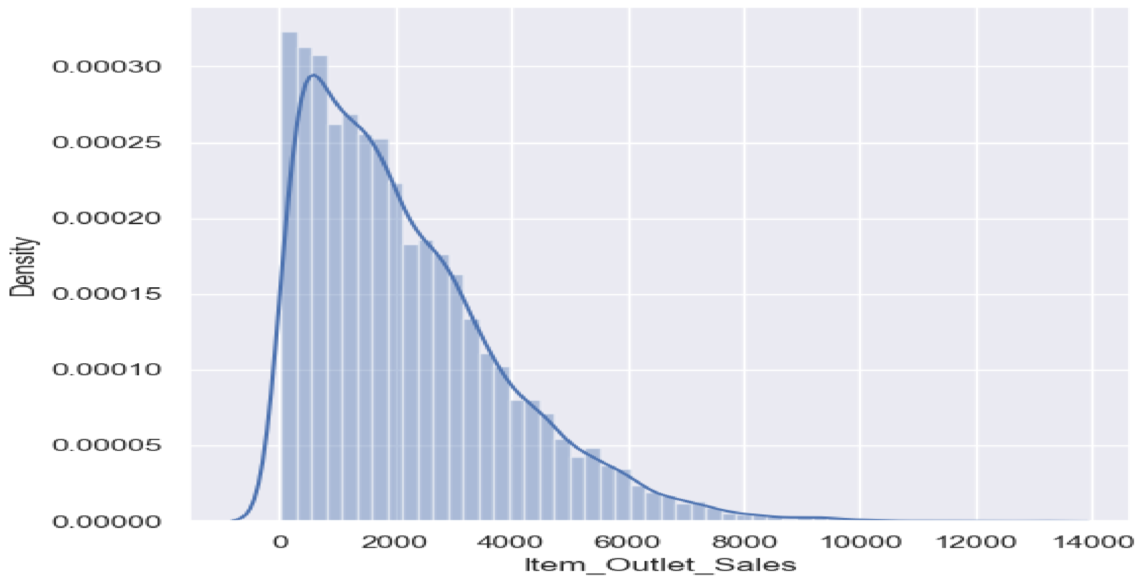
```
# Item MRP distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_MRP'])
plt.show()
```



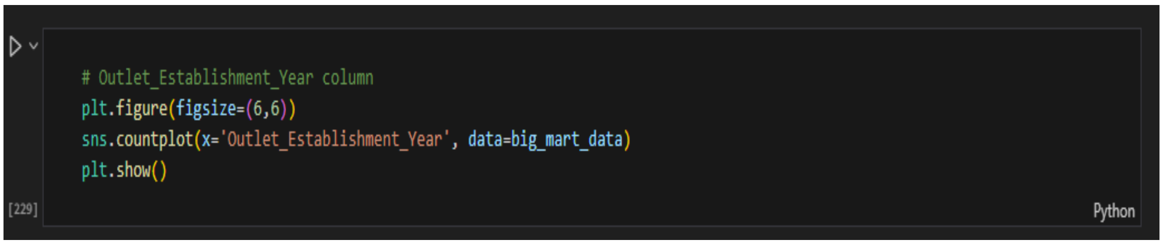
The resulting graph visually represents the distribution of sales of items in the dataset. The x-axis represents the range of sales values, and the y-axis represents the density (probability) of occurrence for each sales value. The shape of the curve provides insights into the distribution pattern, such as whether it is normally distributed, skewed, or has multiple peaks.



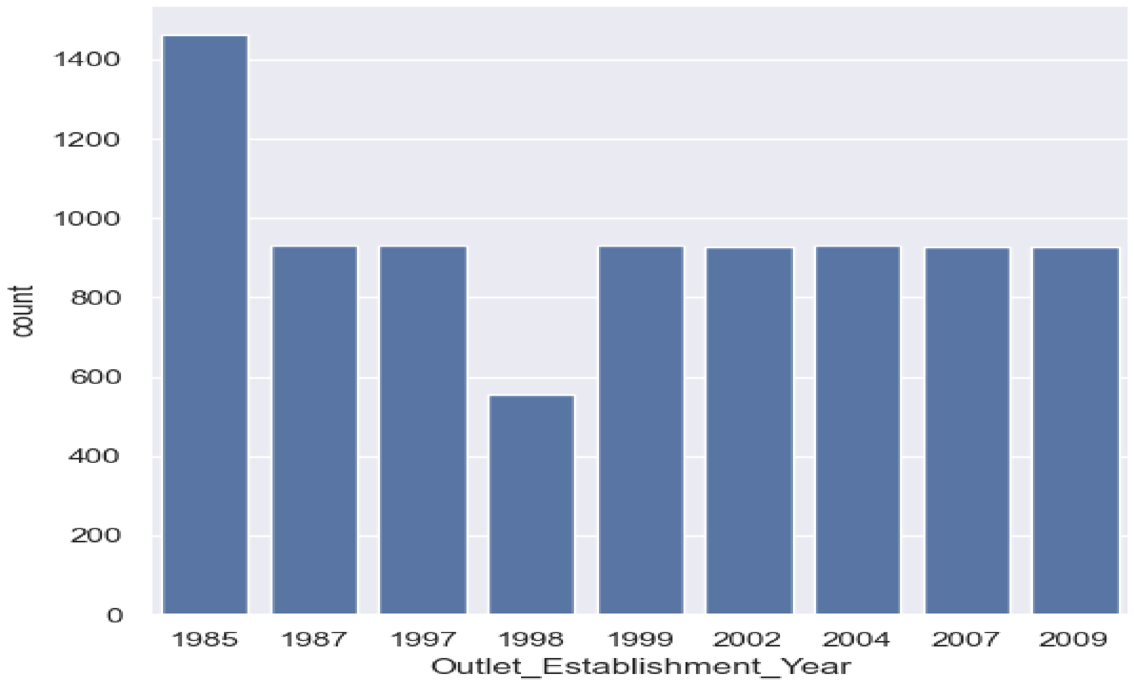
```
# Item_Outlet_Sales distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Outlet_Sales'])
plt.show()
```



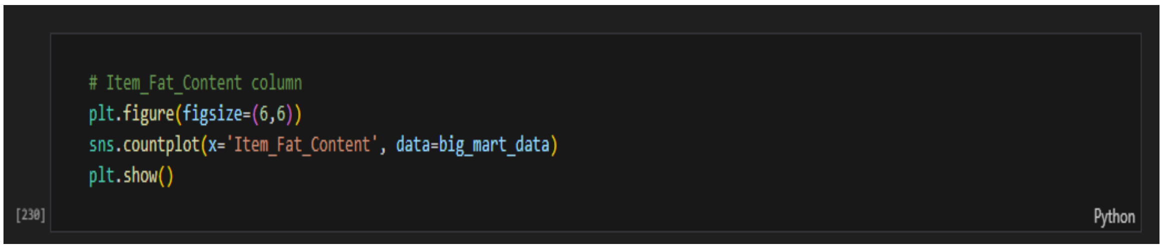
The resulting graph visually represents the distribution of establishment years of outlet in the dataset. Each bar represents the count of oultets established in a specific year. This plot helps in understanding the distribution of outlets over different years of establishment.



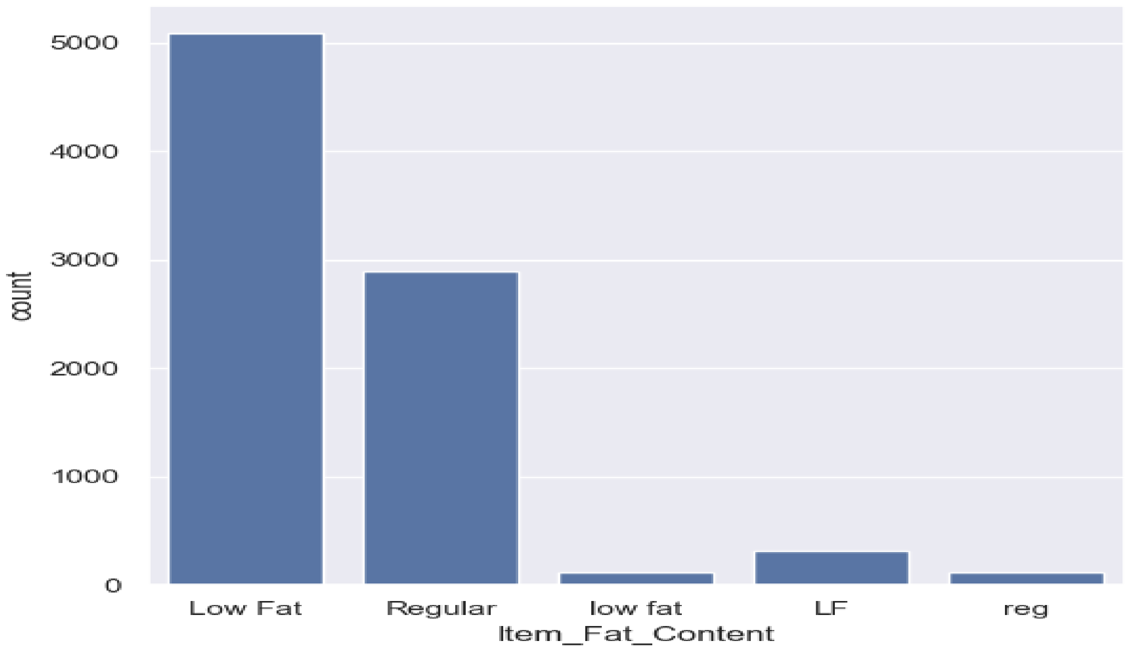
```
# Outlet_Establishment_Year column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Establishment_Year', data=big_mart_data)
plt.show()
```



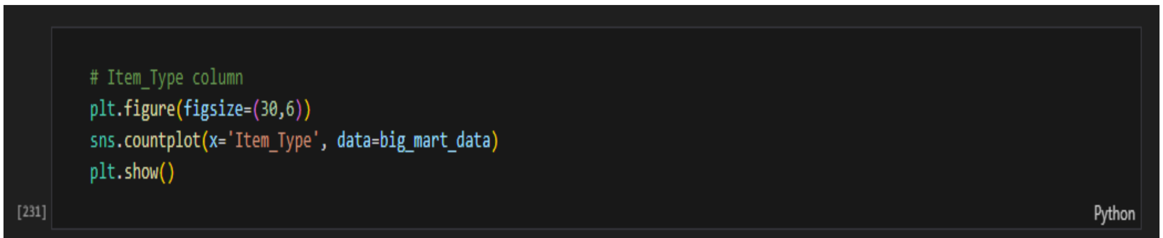
The resulting graph visually represents the distribution of item fat content in the dataset. Each bar represents the count of items belonging to a specific fat content category. This plot helps in understanding the frequency of occurance of different fat content categories among the items in the dataset.



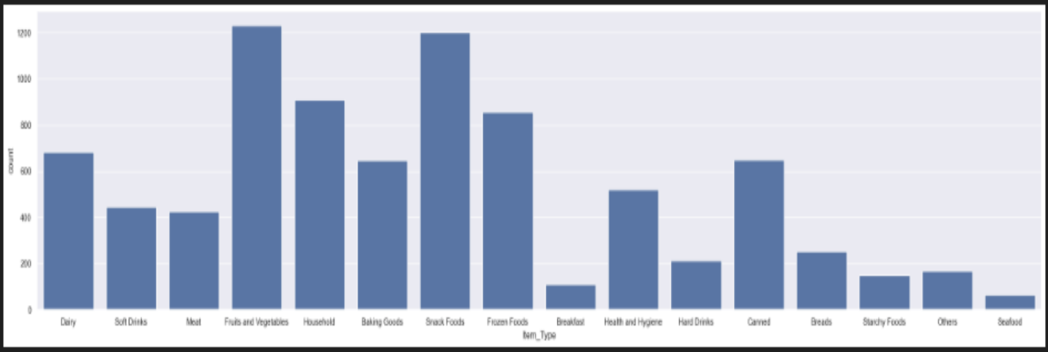
```
# Item_Fat_Content column
plt.figure(figsize=(6,6))
sns.countplot(x='Item_Fat_Content', data=big_mart_data)
plt.show()
```



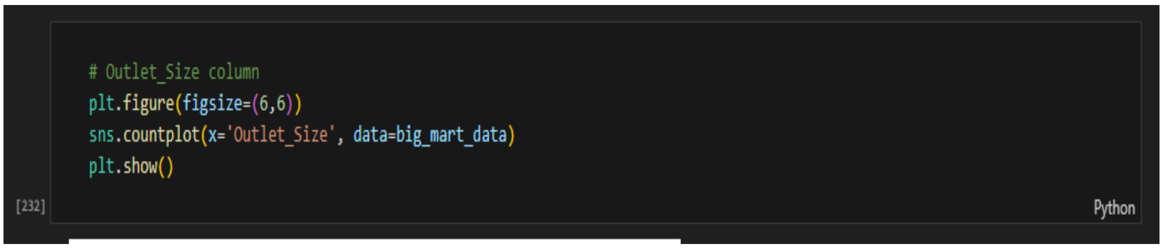
The resulting graph visually represents the distribution of different item types in the dataset. Each bar represents the count of items belonging to a specific item type category. This plot helps in understanding the frequency of occurrence of different item types among the items in the dataset, potentially highlighting which types of items are more prevlent.



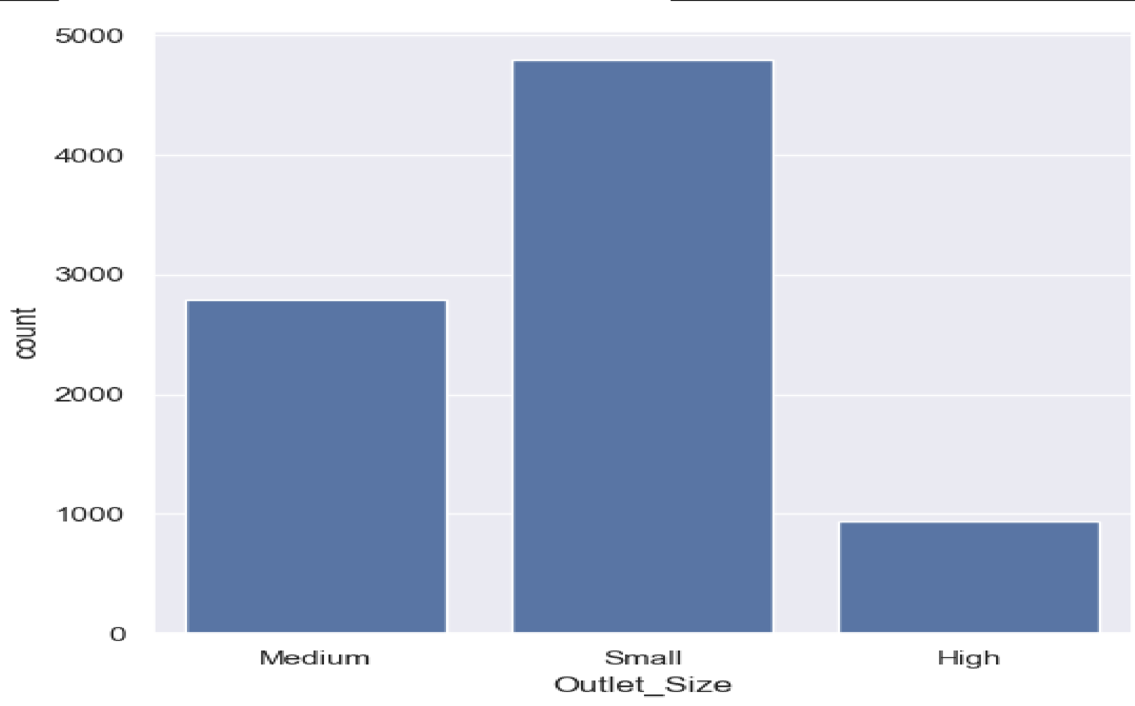
```
# Item_Type column
plt.figure(figsize=(30,6))
sns.countplot(x='Item_Type', data=big_mart_data)
plt.show()
```



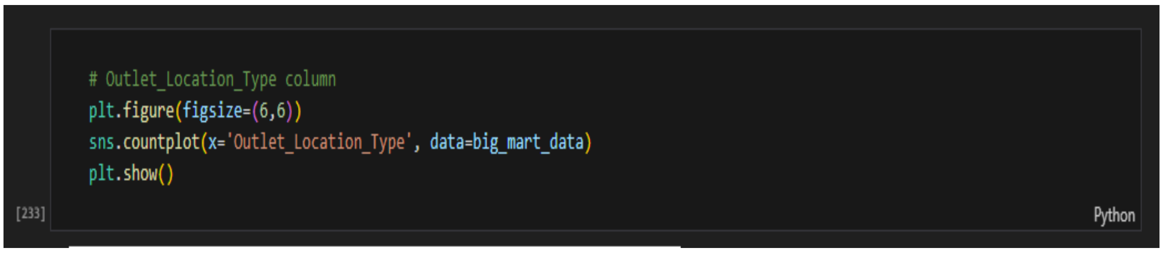
The resulting graph visually represents the distribution of different outlet sizes in the dataset. Each bar represents the count of outlets belonging to a specific size category. This plot helps in understanding the frequency of occurrence of different outlet sizes among the items in the dataset, providing insights into the distribution of outlet sizes.



```
# Outlet_Size column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Size', data=big_mart_data)
plt.show()
```

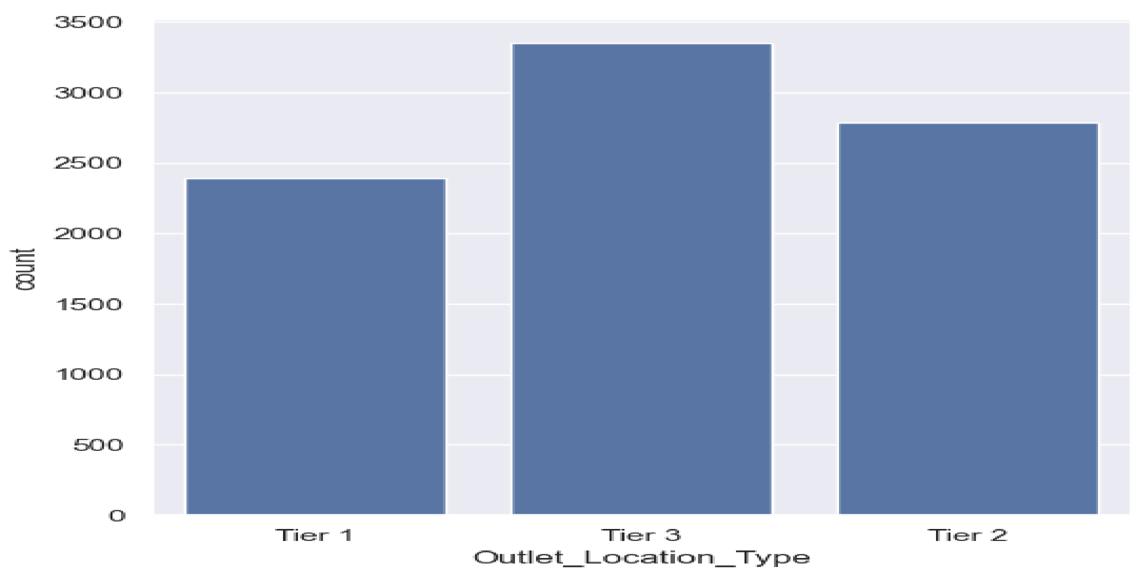


The resulting graph visually represents the distribution of different outlet location type in the dataset. Each bar represents the count of outlets belonging to a specific location type category. This plot helps in understanding the frequency of occurrence of different location types among the outlets in the dataset, providing insights into the distribution of outlet location.

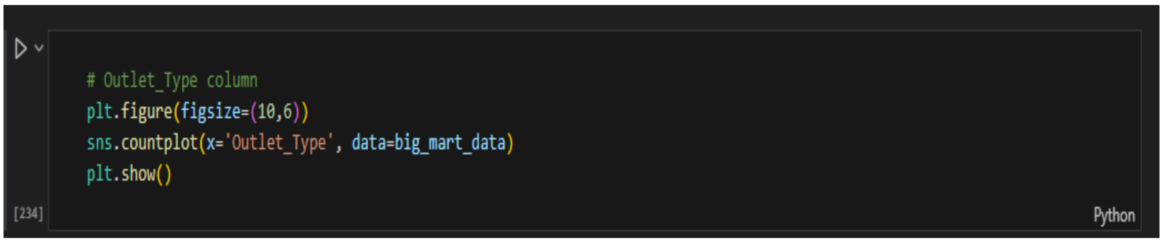


```
# Outlet_Location_Type column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Location_Type', data=big_mart_data)
plt.show()
```

[233] Python

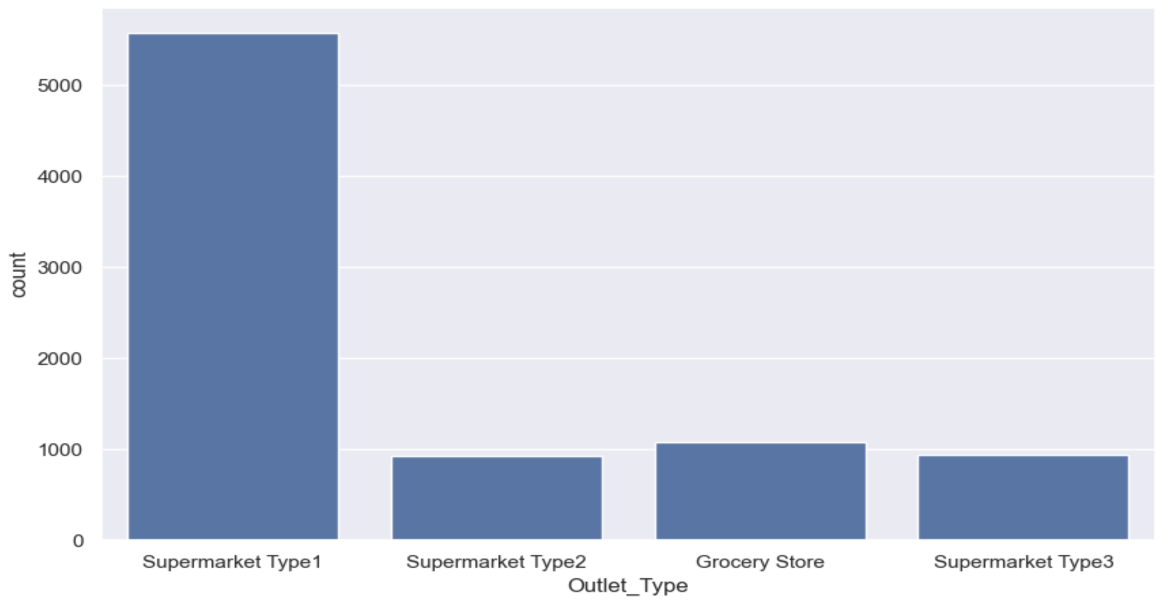


The resulting graph visually represents the distribution of different outlets in the dataset. Each bar represents the count of outlets belonging to a specific type category. This plot helps in understanding the frequency of occurrence of different outlet types among the outlets in the dataset, providing insights into the distribution of outlet types.

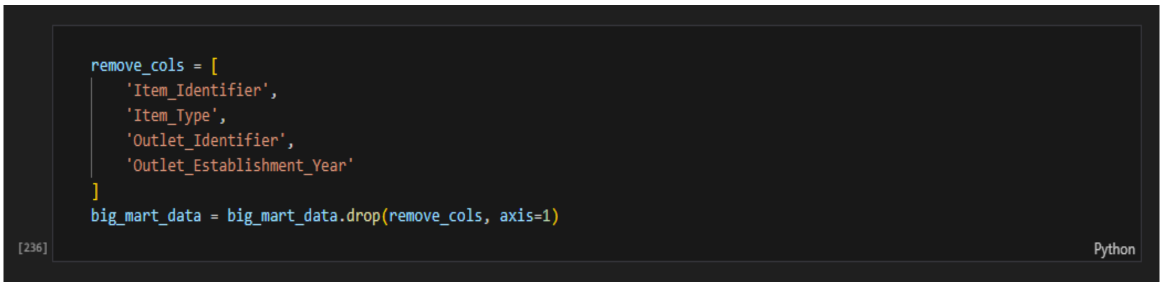


```
# Outlet_Type column
plt.figure(figsize=(10,6))
sns.countplot(x='Outlet_Type', data=big_mart_data)
plt.show()
```

[234] Python



This code removes specified columns from the data frame.

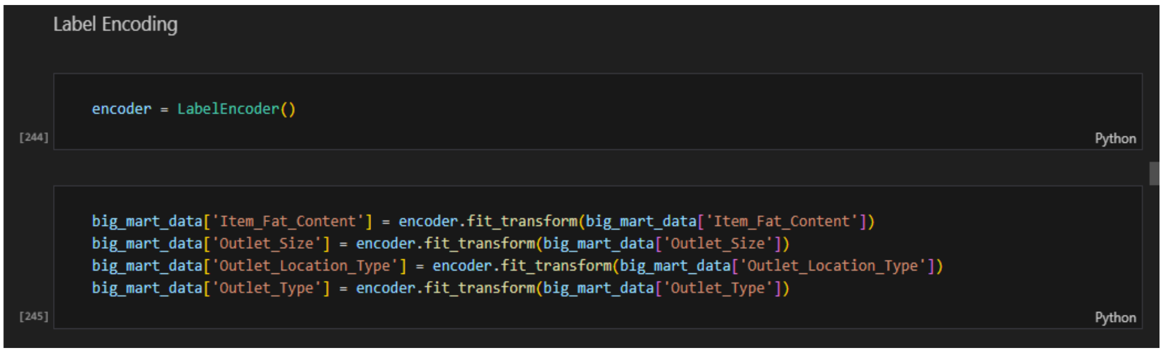


```
remove_cols = [
    'Item_Identifier',
    'Item_Type',
    'Outlet_Identifier',
    'Outlet_Establishment_Year'
]
big_mart_data = big_mart_data.drop(remove_cols, axis=1)
```

[236] Python

This code uses the LabelEncoder from the sci-kit-learn library to encode categorical variables in the data frame.

The fit_transform() method fits the encoder to the unique categories present in the specified column and transforms those categories into numerical labels. These numerical labels are then assigned to the respective columns in big_mart_data.



Label Encoding

```
encoder = LabelEncoder()
```

[244] Python

```
big_mart_data['Item_Fat_Content'] = encoder.fit_transform(big_mart_data['Item_Fat_Content'])
big_mart_data['Outlet_Size'] = encoder.fit_transform(big_mart_data['Outlet_Size'])
big_mart_data['Outlet_Location_Type'] = encoder.fit_transform(big_mart_data['Outlet_Location_Type'])
big_mart_data['Outlet_Type'] = encoder.fit_transform(big_mart_data['Outlet_Type'])
```

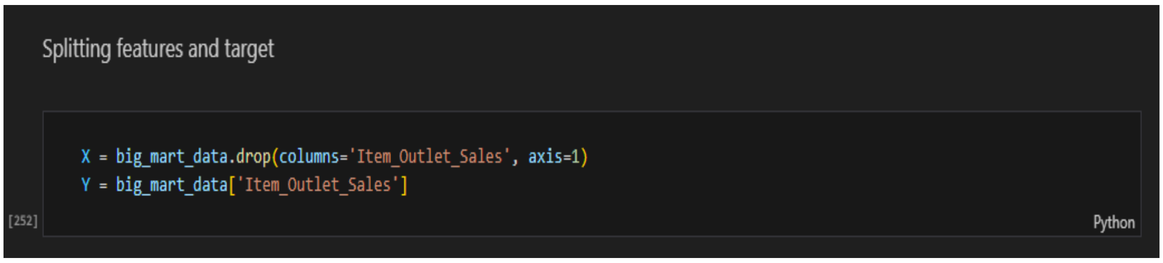
[245] Python

**2.1.3) Model Building:** It involve several steps, including researching and gathering reference materials, planning the design and dimensions of the model, selecting appropriate tools, and carefully following instructions or using one's creativity to construct the model.

We experimented with various machine learning algorithms to develop the prediction model. These algorithms included regression, decision tree, and random forests. we trained and assessed each model to determine which algorithm performed the best.

Splitting the dataset into features (X) and target (Y) is a common step in machine learning tasks. It allows us to separate the input variables (features) from the output variable (target), enabling us to apply various machine learning algorithms to predict or analyze the target variable based on the given features.

Here, we split into X and Y where X represents the features or independent variables of the dataset and Y represents the target or dependent variable of the dataset.

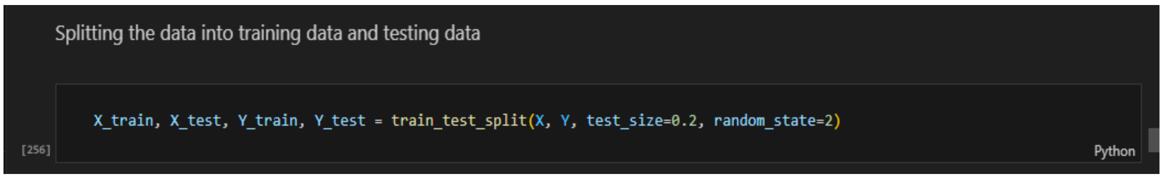

Splitting features and target

```python
X = big_mart_data.drop(columns='Item_Outlet_Sales', axis=1)
Y = big_mart_data['Item_Outlet_Sales']
```

The data is being split into training and testing tests using the train_test_split function from the sklearn.model_selection module.

i. X: It represents the independent variable.
ii. Y: It represents the dependent variable.
iii. test_size: It is the proportion of the data to be allocated to the testing dataset.
iv. random_state: The random seed used for splitting the data. By setting it to a specific value (in this case, 2, you can reproduce the same splits every time you run the code.

The following code splits the provided data into training and testing sets, with 80% of the data being used for training and 20% for testing.


Splitting the data into training data and testing data

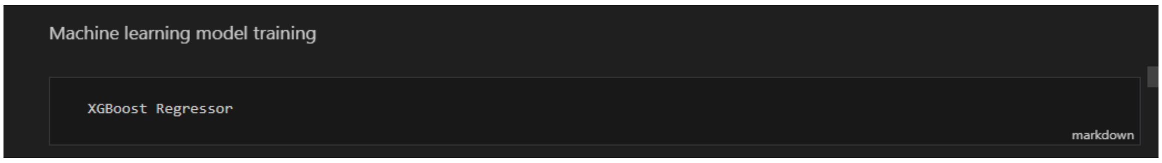
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

XGBoost Regressor is an implementation of the gradient boosting algorithm for regression problems.

To train a machine learning model using XGBoost regressor, you can follow these steps:

1. Import the necessary libraries
2. Create an instance of the XGBRegressor object
3. Fit the model to your training data

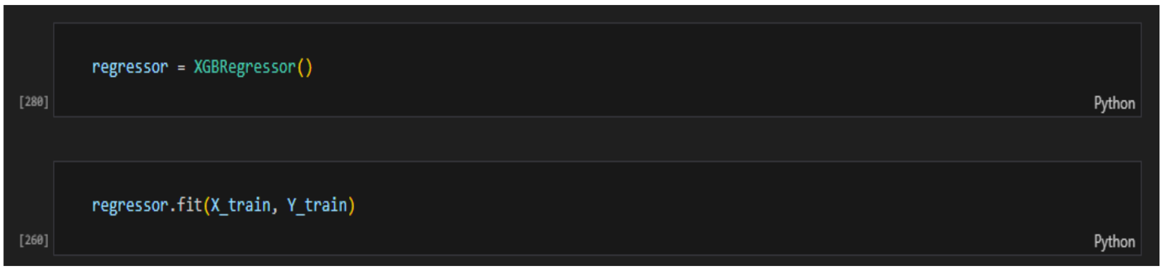
   Here, X_train represents the feature matrix and Y_train represents the target variable for training data.



Machine learning model training

```
XGBoost Regressor
```

markdown



```
regressor = XGBRegressor()
```

[280] Python

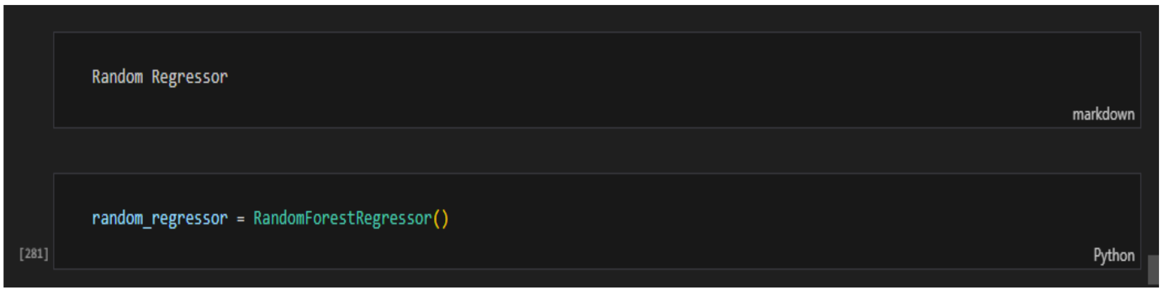
```
regressor.fit(X_train, Y_train)
```

[260] Python

The Random Regressor is a type of machine learning model used for regression tasks. It is mostly used for classification tasks. To train a machine learning model using Random Regressor, you can follow these steps:

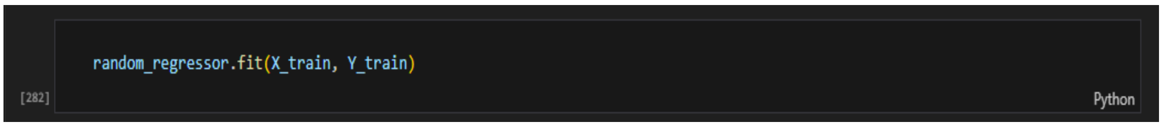
1. We first need to import it from the library.
2. Create an instance of the model.
3. We need to train the model using our training data

The fit() function is used to train the model. It takes in the features and target variables and learns the patterns in the data.



```
Random Regressor
```

```
[281]
random_regressor = RandomForestRegressor()
```



```
[282]
random_regressor.fit(X_train, Y_train)
```

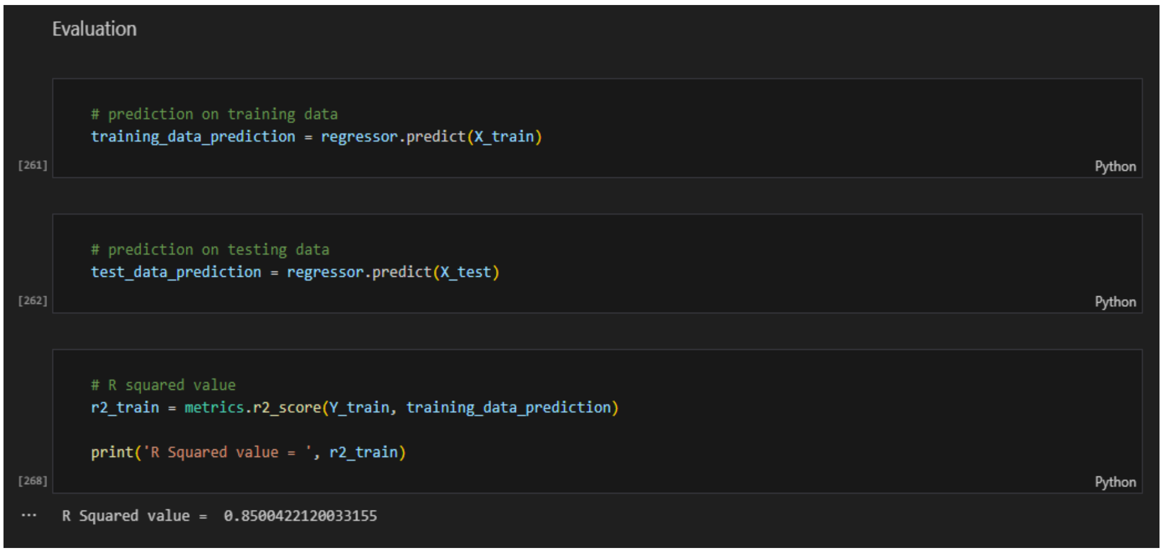
**2.1.4) Model Evaluation:** This refers to the process of assessing the performance and quality of a model based on various metrics.

Once the models were developed, we evaluated their performance using appropriate evaluation metrics, such as mean squared error (MSE), mean absolute error (MAE), or accuracy. We also considered the models' interpretability and ability to handle new data and changing trends.

The R-squared value is a statistical measure that represents the goodness of fit of a regression model. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.

In this case, the R-squared value on the training data is 0.8500422120033155. This means that approximately 85% of the variance in the dependent variable can be explained by the independent variables in the training dataset.

The higher the R-squared value, the better the model fits the data.



Evaluation

```
# prediction on training data
training_data_prediction = regressor.predict(X_train)
```

[261] Python

```
# prediction on testing data
test_data_prediction = regressor.predict(X_test)
```

[262] Python

```
# R squared value
r2_train = metrics.r2_score(Y_train, training_data_prediction)

print('R Squared value = ', r2_train)
```

[268] Python

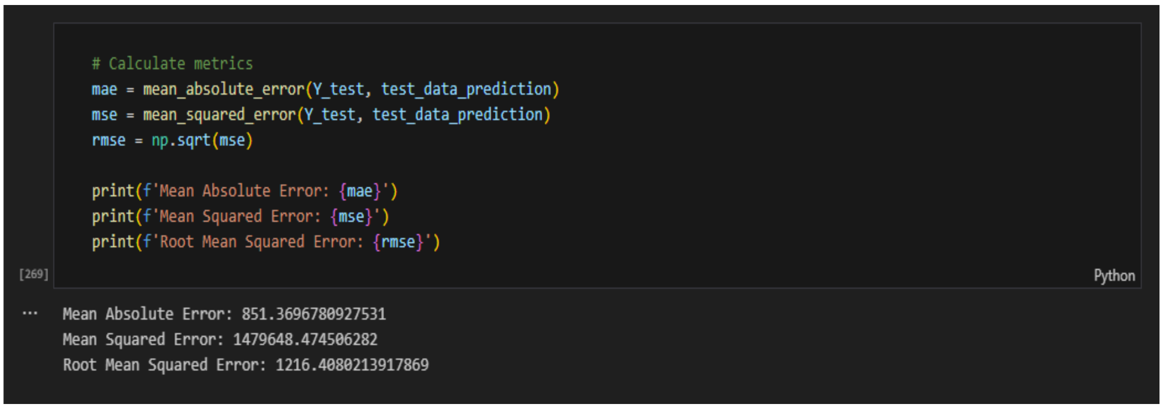
```
R Squared value =  0.8500422120033155
```

The Mean Absolute Error (MAE) is a metric that calculates the average absolute difference between the predicted values and the actual values.

The Mean Squared Error (MSE) is a metric that calculates the average squared difference between the predicted values and the actual values.

The Root Mean Squared Error (RMSE) is the square root of the MSE and provides a measure of the average magnitude of the error.

These metrics are useful to evaluate the performance of the model.

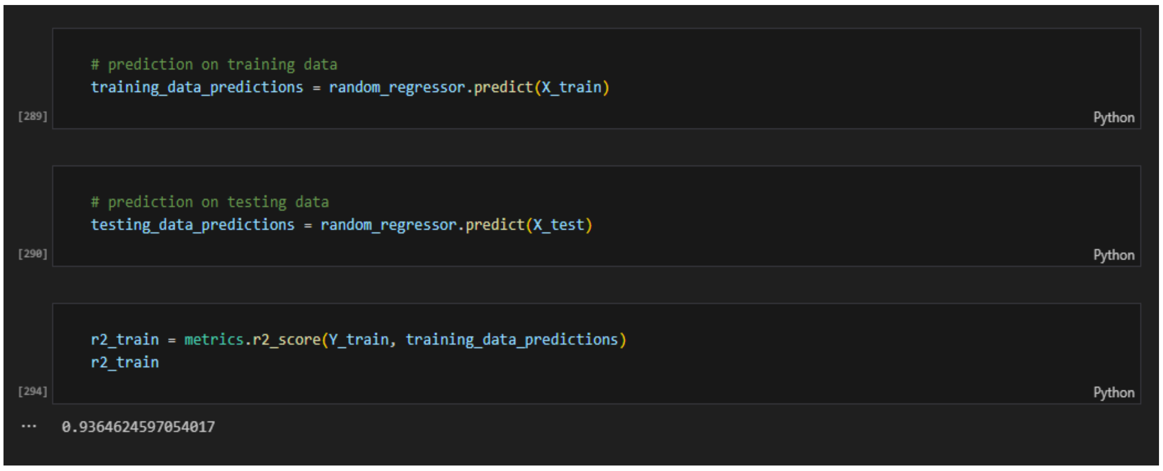


```
# Calculate metrics
mae = mean_absolute_error(Y_test, test_data_prediction)
mse = mean_squared_error(Y_test, test_data_prediction)
rmse = np.sqrt(mse)

print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
print(f'Root Mean Squared Error: {rmse}')
```

[269] Python

```
Mean Absolute Error: 851.3696780927531
Mean Squared Error: 1479648.474506282
Root Mean Squared Error: 1216.4080213917869
```

The following code is making predictions using a random regressor model on both the training data and testing data.



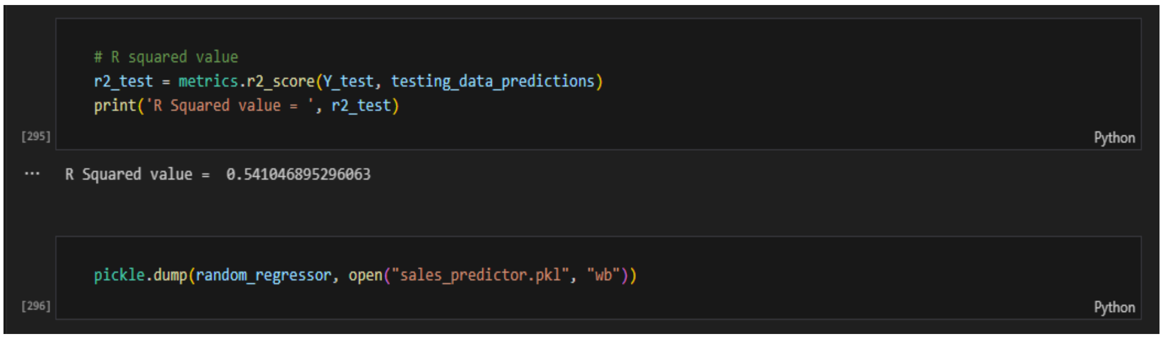
```
# prediction on training data
training_data_predictions = random_regressor.predict(X_train)
```
[289] Python

```
# prediction on testing data
testing_data_predictions = random_regressor.predict(X_test)
```
[290] Python

```
r2_train = metrics.r2_score(Y_train, training_data_predictions)
r2_train
```
[294] Python

```
... 0.9364624597054017
```

The R-squared value is a statistical measure that represents the proportion of the variance in the dependent variable (Y_test) that can be explained by the independent variable(s) (testing_data_predictions).



```
# R squared value
r2_test = metrics.r2_score(Y_test, testing_data_predictions)
print('R Squared value = ', r2_test)
```
[295] Python

```
... R Squared value =  0.541046895296063
```

```
pickle.dump(random_regressor, open("sales_predictor.pkl", "wb"))
```
[296] Python

**2.1.5) Model Deployment:** It refers to the process of making a trained machine learning model available for use in production environments. It involves the integration of the model into an application or system, allowing it to make predictions or carry out desired tasks.

After selecting the best-performing model, we deployed it in a production environment. This way the trained model can be loaded quickly and efficiently into the production system without the need to re-train it.

## 2.2) Models: For this project, two prediction models are considered: Random Regressor and XGBoost Regressor.

### 2.2.1) Random Regressor:

Random Regressor is a simple model that generates random predictions within the range of the observed target variable. It does not learn from the data and is often used as a baseline model for comparison.

It is a basic machine learning model used in regression tasks where the goal is to predict a continuous target variable. It is called "random" because it does not learn from the data and makes predictions based on random values.

#### *a) Advantages:*

Easy to implement and understand.

It is effective in handling large datasets that have many attributes.

Can be helpful in situations where no specific pattern or relationship is expected in the data.

#### *b) Disadvantages:*

Doesn't consider any data characteristics.

It may suffer from overfitting if the model is too complex as it is unable to capture complex patterns in the data.

Can provide inconsistent results due to randomness.

### 2.2.2) XGBoost Regressor:

XGBoost is a powerful machine-learning algorithm that is widely used for predictive modeling. It is an ensemble model based on a gradient boosting technique.

#### *a) Advantages:*

Can capture complex non-linear relationships between features and the target variable.

Handles missing data, outliers, and irrelevant features effectively.

Provides feature importance analysis, which helps in identifying the most significant features.

Can handle large datasets efficiently.

### ***b) Disadvantages:***

XGBoost requires a large amount of memory to store the decision trees and intermediate results during the training process.

Training time can be relatively longer than simpler models due to the ensemble nature.

It lacks transparency. It is considered as black-box algorithm because it can be difficult to interpret and understand how the model is making predictions.

# Chapter 3
# Flask

## 3.1) Flask:

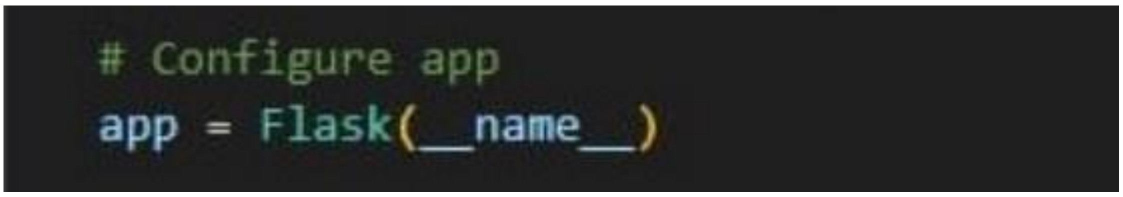
We connected the prediction model with Flask which is a popular Python web framework to enable its deployment as a web application. Flask integration enables us to create a user-friendly interface that enables smooth user interaction with the model.

**Flask Setup:** We must set up Flask and its dependencies before we can connect the model to Flask. This involves installing Flask via pip, creating a virtual environment that is it allows you to have separate Python environments for different projects and installing the required packages.

**Flask App structure:** It consists of

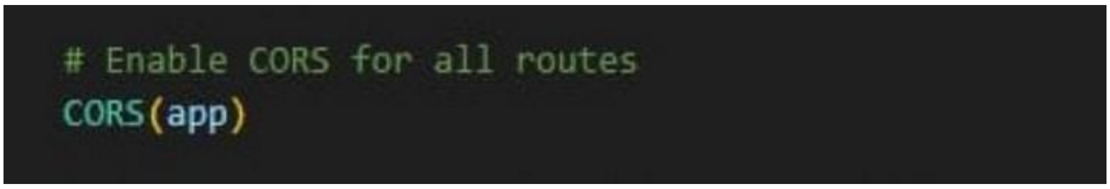
***app.py:*** This file serves as the main entry point of the Flask application, a primary script that defines routes, initiates the Flask application, and handles user requests. It is also where you would define any global settings or configuration variables. These are the standard elements that make up an app.py file.

1. Importing Flask and other necessary modules like request, render_template
   i. flask_cors for enabling CORS (Cross-Origin Resource Sharing) in the application
   ii. pickle for loading and saving Python objects (such as trained models).
   iii. SQLite3 for interacting with the SQLite database.
   iv. OS for operating system-related functionalities.
   v. warnings for filtering and ignoring warning messages.
2. Creating an instance of the Flask application



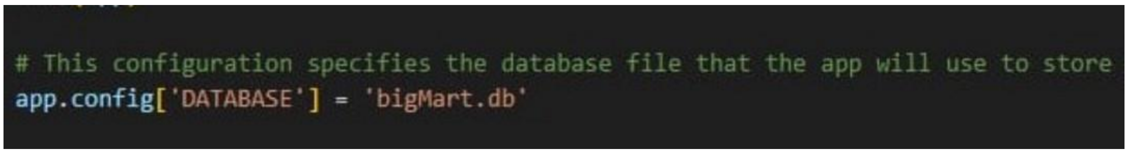
```
# Configure app
app = Flask(__name__)
```

3. The CORS middleware allows your app to handle requests from different origins (e.g., different domains) by adding appropriate headers to the responses so we enable CORS for all routes.



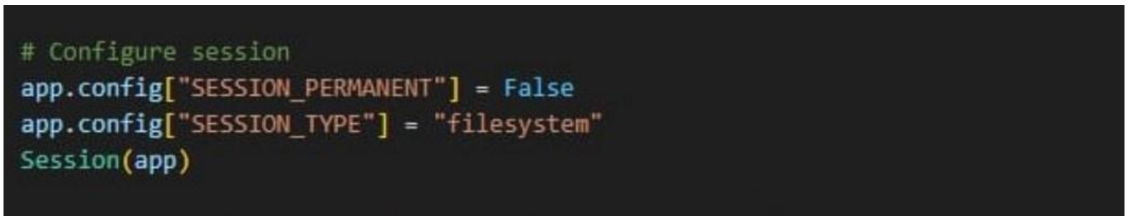
```
# Enable CORS for all routes
CORS(app)
```

4. Set the configuration for the SQLite database file (example bigMart.db) that the app will use to store and retrieve data.



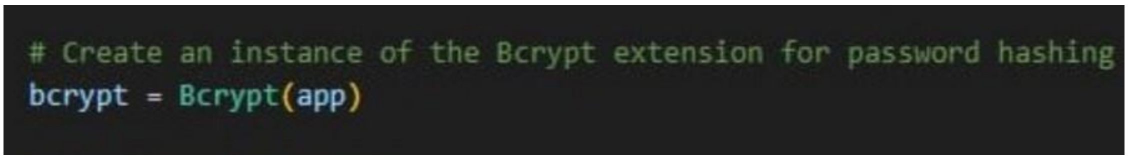
```
# This configuration specifies the database file that the app will use to store
app.config['DATABASE'] = 'bigMart.db'
```

5. SESSION PERMANENT is set to False to make the user session last only as long as the user is active where as SESSION_TYPE is set to "filesystem" to store the session data on the file system.



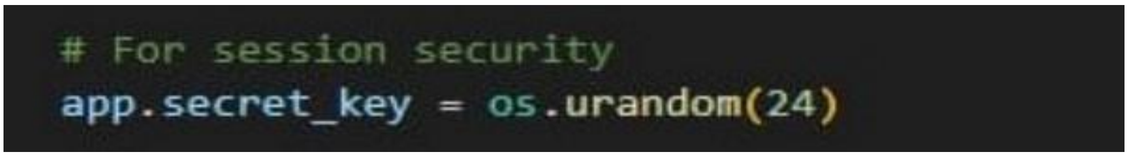
```
# Configure session
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)
```

6. Created an instance of the Bcrypt extension for password hashing. By using this you can securely hash and verify passwords in your Flask application.



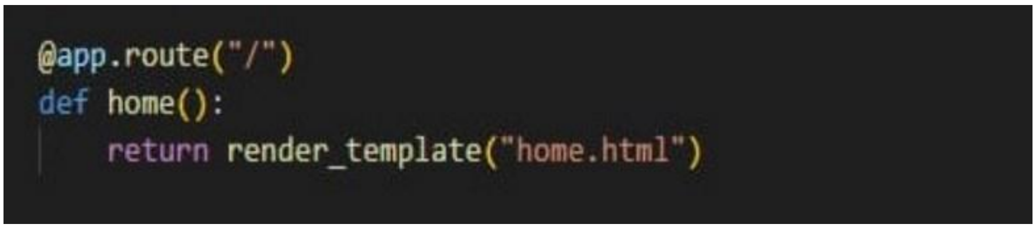
```
# Create an instance of the Bcrypt extension for password hashing
bcrypt = Bcrypt(app)
```

7. The security of the generated secret key is vital for maintaining the security of your session therefore generate a secret key for session security using os.urandom(24).



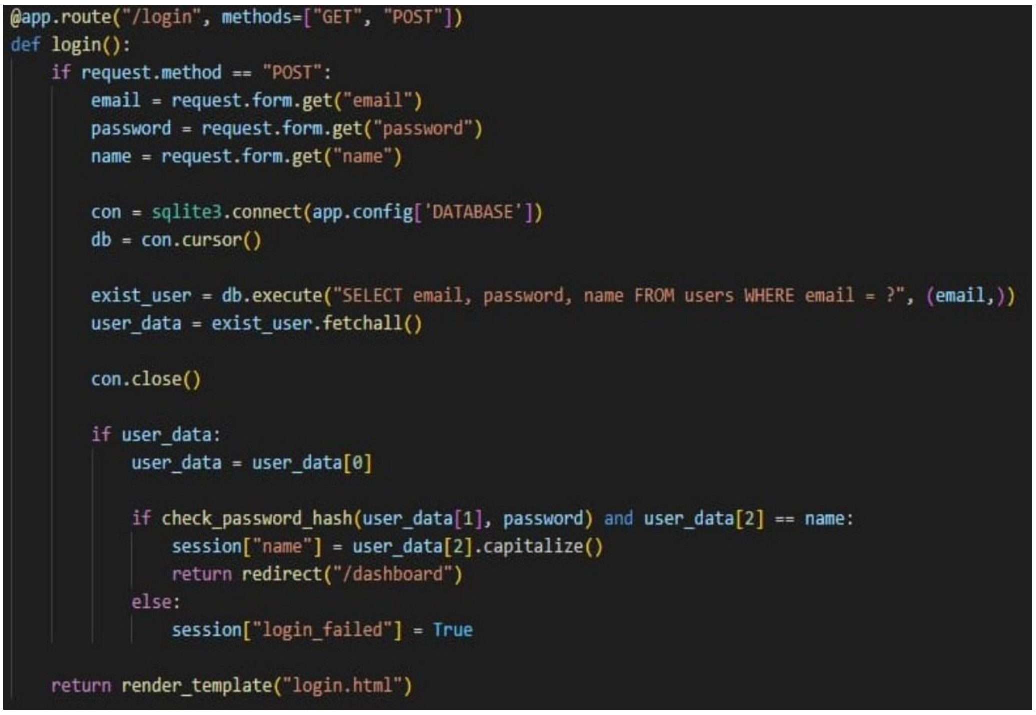
```
# For session security
app.secret_key = os.urandom(24)
```

8. Defining routes using decorators. This flask application has following routes:
   i. The / route renders the home.html template when the user visits the homepage.



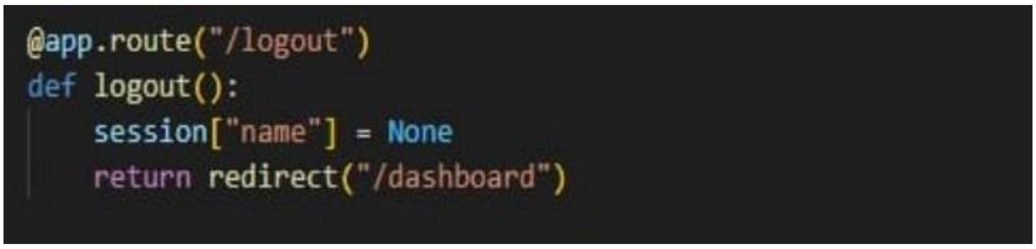
```python
@app.route("/")
def home():
    return render_template("home.html")
```

ii. The /login route handles both GET and POST requests. If the request method is POST, it retrieves the email, password, and name from the form fields. It then connects to a SQLite database and checks if a user with the provided email exists. If the user exists and the password matches, it sets the user's name as a session variable. If the user does not exist or the password doesn't match, it sets a session variable "login_failed" to True.



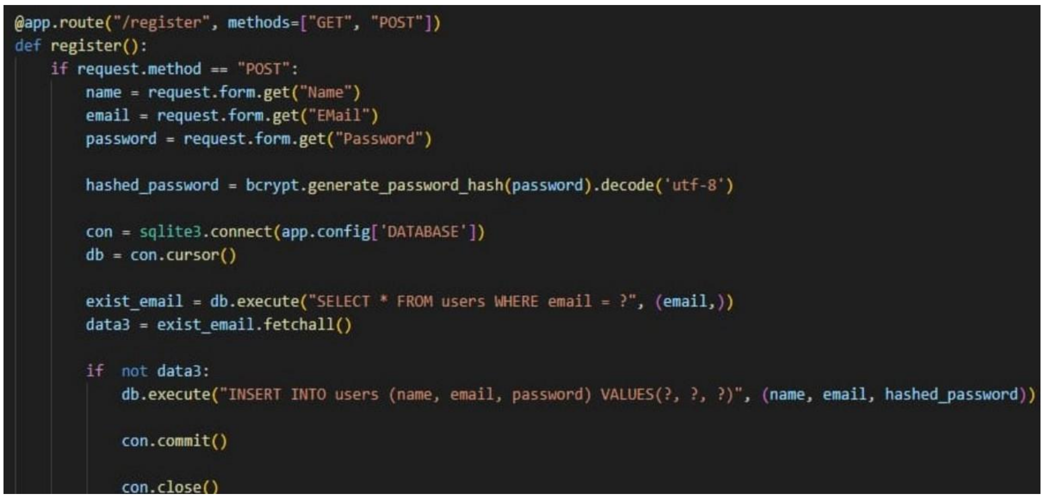
```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form.get("email")
        password = request.form.get("password")
        name = request.form.get("name")

        con = sqlite3.connect(app.config['DATABASE'])
        db = con.cursor()

        exist_user = db.execute("SELECT email, password, name FROM users WHERE email = ?", (email,))
        user_data = exist_user.fetchall()

        con.close()

        if user_data:
            user_data = user_data[0]

            if check_password_hash(user_data[1], password) and user_data[2] == name:
                session["name"] = user_data[2].capitalize()
                return redirect("/dashboard")
            else:
                session["login_failed"] = True

    return render_template("login.html")
```

iii. The /logout route is used to log out the user. It clears the name variable in the session and redirects the user to the /dashboard page.

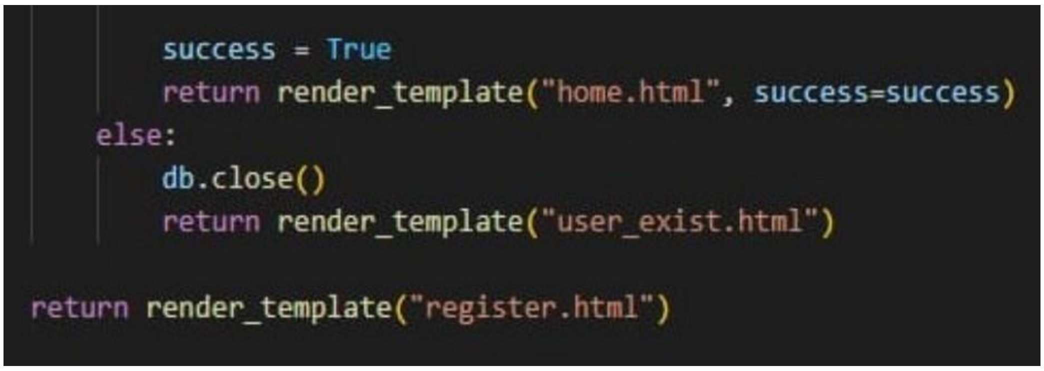


```python
@app.route("/logout")
def logout():
    session["name"] = None
    return redirect("/dashboard")
```

iv. The /register route is used for user registration. It handles both GET and POST requests. For a POST request, it retrieves the user's name, email, and password from the form data. It then generates a hashed password using bcrypt and inserts the user's information into the users table in the database.

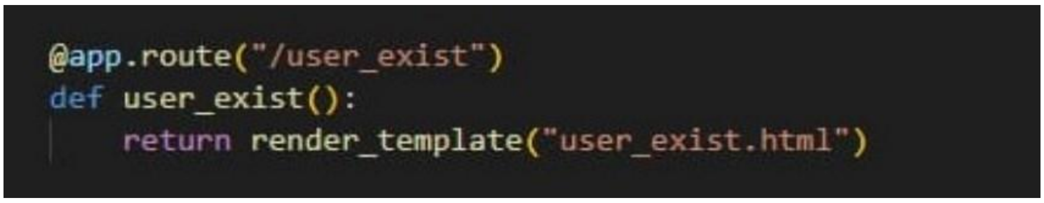


```
@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        name = request.form.get("Name")
        email = request.form.get("EMail")
        password = request.form.get("Password")

        hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

        con = sqlite3.connect(app.config['DATABASE'])
        db = con.cursor()

        exist_email = db.execute("SELECT * FROM users WHERE email = ?", (email,))
        data3 = exist_email.fetchall()

        if  not data3:
            db.execute("INSERT INTO users (name, email, password) VALUES(?, ?, ?)", (name, email, hashed_password))

            con.commit()

            con.close()
```



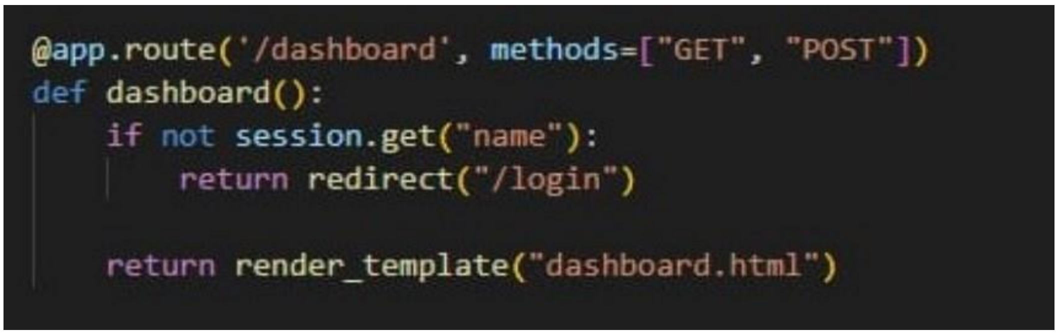
```
            success = True
            return render_template("home.html", success=success)
        else:
            db.close()
            return render_template("user_exist.html")

    return render_template("register.html")
```

v. The /user_exist route renders a tempelate for notifying the user that they already exist.



```
@app.route("/user_exist")
def user_exist():
    return render_template("user_exist.html")
```

vi. The /dashboard route checks if the user is logged  in and if not, it redirects them to the login page. If the user is logged in, it renders the dashboard tempelate.



```
@app.route('/dashboard', methods=["GET", "POST"])
def dashboard():
    if not session.get("name"):
        return redirect("/login")

    return render_template("dashboard.html")
```

9. Handling additional routes and HTTP methods
10. Running the Flask application

Overall in this file, the core functionality and behavior of the Flask application are defined.

***templates folder:*** It contains HTML templates to generate dynamic web pages. Using templates allows developers to separate the presentation layer from the logic layer of the web application, making it easier to maintain and update the design of the website independently from the underlying code.

***static folder:*** It is a directory within a web server's file system where files such as CSS, JavaScript, images, and other static assets are stored. These files do not change dynamically and are served as they are to the client, meaning that the server sends these files directly to the user's browser without modifying them.

***Big_Mart_Sales_Prediction.ipynb:*** It loads and applies the machine learning model that has been trained

Once the model is integrated, testing is done to ensure the functionality of the web application. Testing involves providing test data, confirming the model's predictions, and making sure the error-handling system performs as expected.

# Chapter 4

# Database

## 4.1) Database:

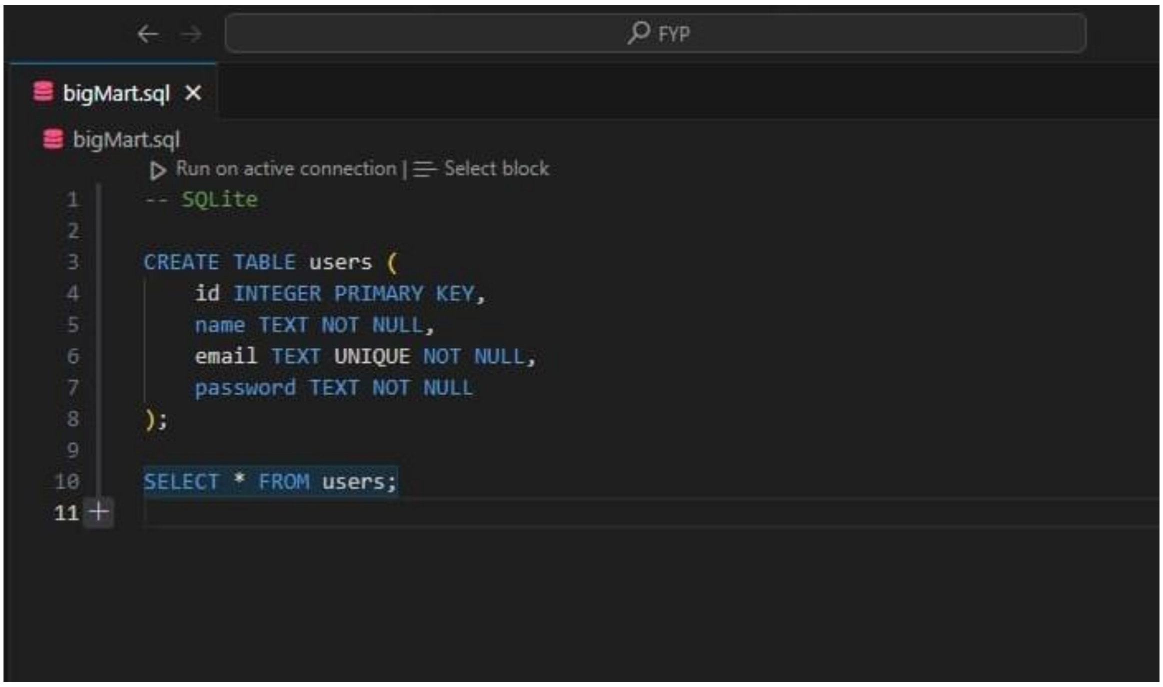
A database plays an essential role in storing and retrieving data. It enables effective management, organization, and analysis of massive amounts of data.

To connect our project with the database, we used SQLite.

The connected database allowed us to store and manage large volumes of data efficiently.

SQLite provides several advantages:

1. Minimal setup and configuration: SQLite3 does not require any additional setup or configuration. It is a simple library that can be added to an application with minimal effort.
2. Excellent performance for our needs: SQLite is known for its high performance.
3. Efficient management of large volumes of data: SQLite is capable of efficiently managing large datasets. It efficiently manages large volumes of data.
4. No significant performance degradation: Sqlite is designed to maintain high performance even as the database grows in size.



```sql
-- SQLite

CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL
);

SELECT * FROM users;
```

# Chapter 5
# Dashboard

## 5.1) Dashboard:

A sales dashboard provides a comprehensive overview of the sales performance of a business or organization. The purpose of the dashboard is to visualize and monitor the sales of a business or organization. It plays an essential role in providing insights and decision-making support for sales teams and managers.

Its key objectives are:

- The dashboard gives sales teams an efficient way to allocate their time and resources by giving them a centralized and real-time view of sales prediction. This will help in increasing productivity and optimizing efficiency.
- To increase sales performance by accurately predicting sales as a dashboard helps in identifying potential opportunities and areas for improvement.

Dashboard was designed by keeping user experience and usability in mind. It aimed to provide a clear visual representation of the prediction model's result.

To achieve this, the dashboard utilizes two main libraries: Streamlit and Plotly. Streamlit is a powerful open-source Python library that offers a user-friendly interface and allows users to interact with the application easily whereas Plotly is a graphing library that provides a variety of interactive and visually appealing plots.

Overall, the dashboard's design and functionality prioritize user experience and ease of use.

The user of this Streamlit app has the option to utilize the "Train.csv" file by default or upload a CSV file.

The app provides a sidebar with input parameters for the user to provide their own values for the features used in the sales prediction model.

The following input features are supported:

Item Weight: The weight of the item being sold.

Item Fat Content: The fat content of the item (options: "Low Fat" or "Regular").

Item Visibility: The visibility of the item in the store.

Item MRP: The maximum retail price of the item.

Outlet Size: The size of the outlet (options: "Small", "Medium", or "High").Outlet Location Type: The location type of the outlet (options: "Tier 1", "Tier 2", or "Tier 3").

Outlet Type: The type of the outlet (options: "Grocery Store", "Supermarket Type1", "Supermarket Type2", or "Supermarket Type3").

Once the user provides values for all the input features and clicks the "Predict Sales" button, the app loads the pre-trained sales prediction model from the "sales_predictor.pkl" file and makes a prediction using the provided input data. The predicted sales value is then displayed to the user.

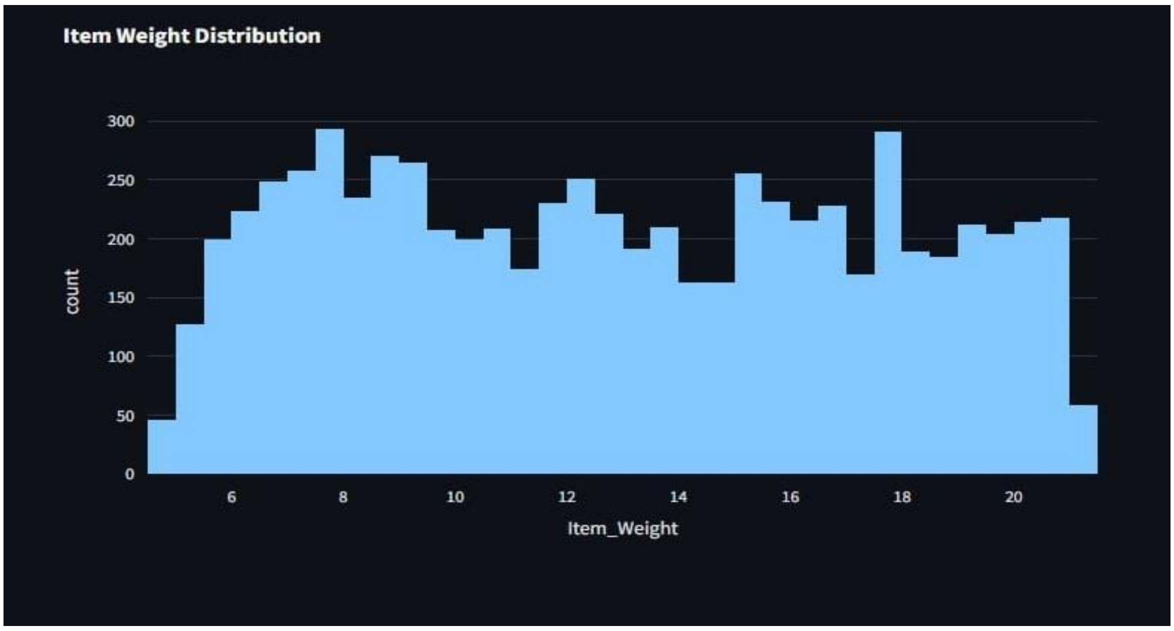
The user is presented with an error message in the event that any problems arise throughout the prediction process.

If any issues arise during the prediction process, such as missing values or incorrect data types, the app will display an error message indicating the specific problem. The user can then correct the input and try again.

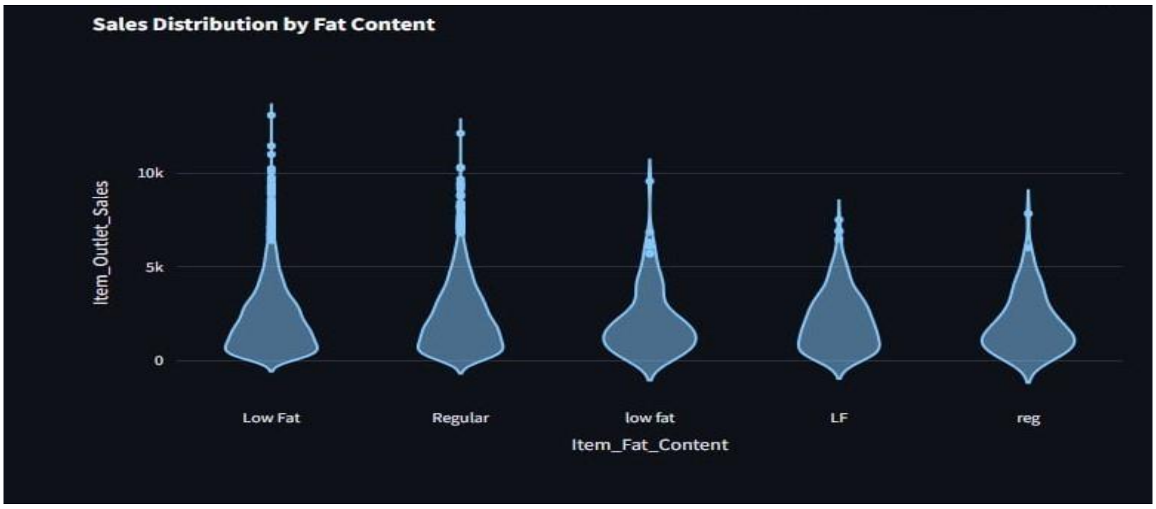
The app also provides a reset button that allows the user to clear all the input values and start fresh. This can be useful if the user wants to make multiple predictions with different input parameters.

Overall, the app provides a user-friendly interface for inputting data and predicting sales using a pre-trained model.

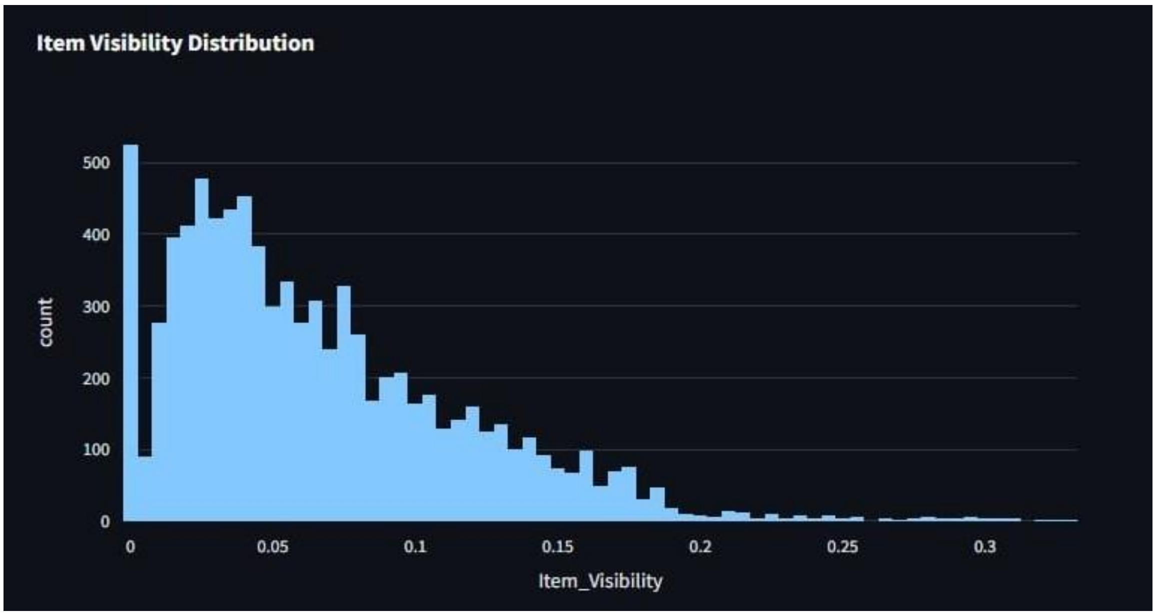
**5.1.1) Item Weight Distribution:** A histogram is a graphical representation that provides a visual display of the distribution of a dataset. In this case, a histogram can be used to show the distribution of item weights to gain insights useful for inventory management and marketing decisions. Through understanding this users identify the range of item weight which is useful in inventory management and marketing decisions.



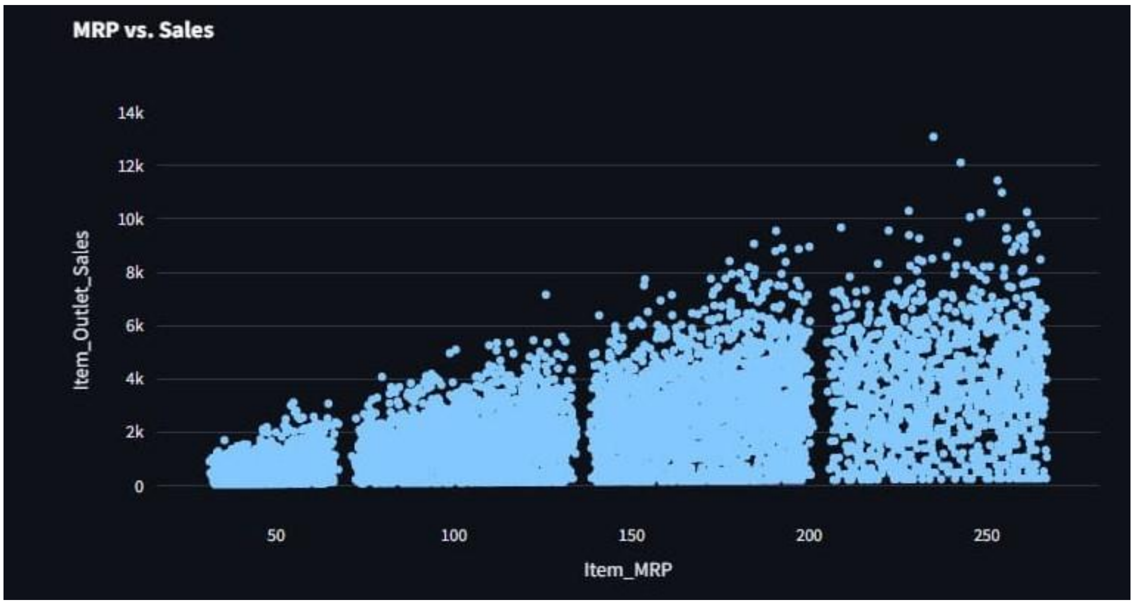
**5.1.2) Sales Distribution by Fat Content:** A violin plot is a type of visual representation that combines elements of a box plot and a kernel density plot. It can be used to illustrate the distribution of a quantitative variable. Here it shows the distribution of sales for different categories of item fat content. It helps the user to analyze the impact of fat content on sales and identify potential trends and patterns.



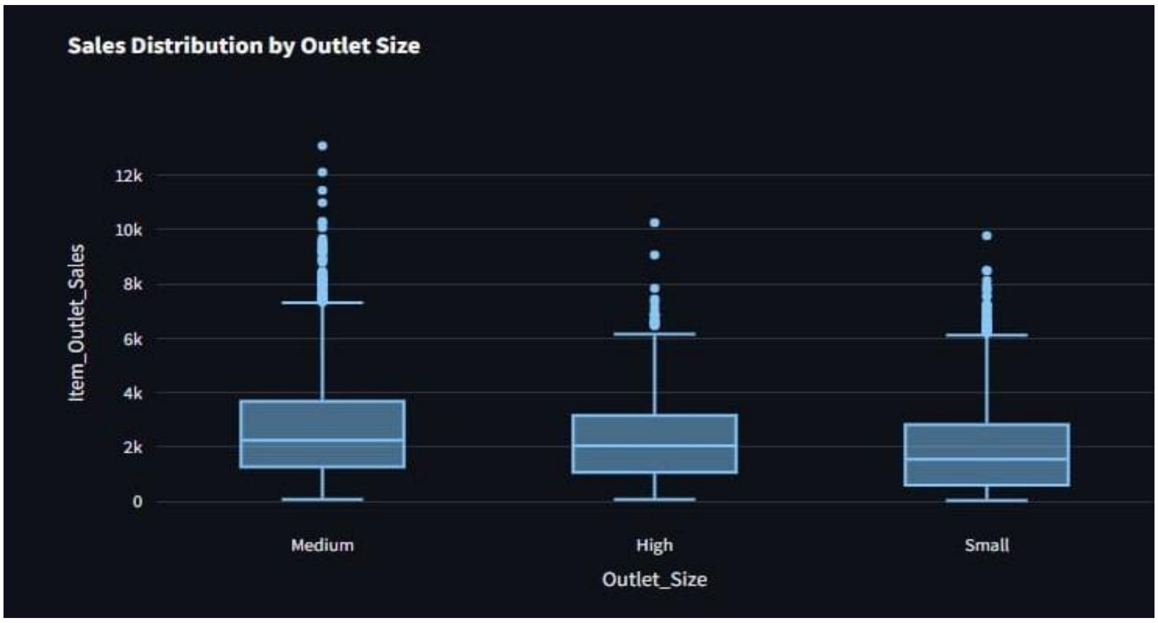
**5.1.3) Item Visibility Distribution:** Histogram, again is used to show the distribution of item visibility. Understanding this can be crucial for optimizing product planning placement and marketing strategies.



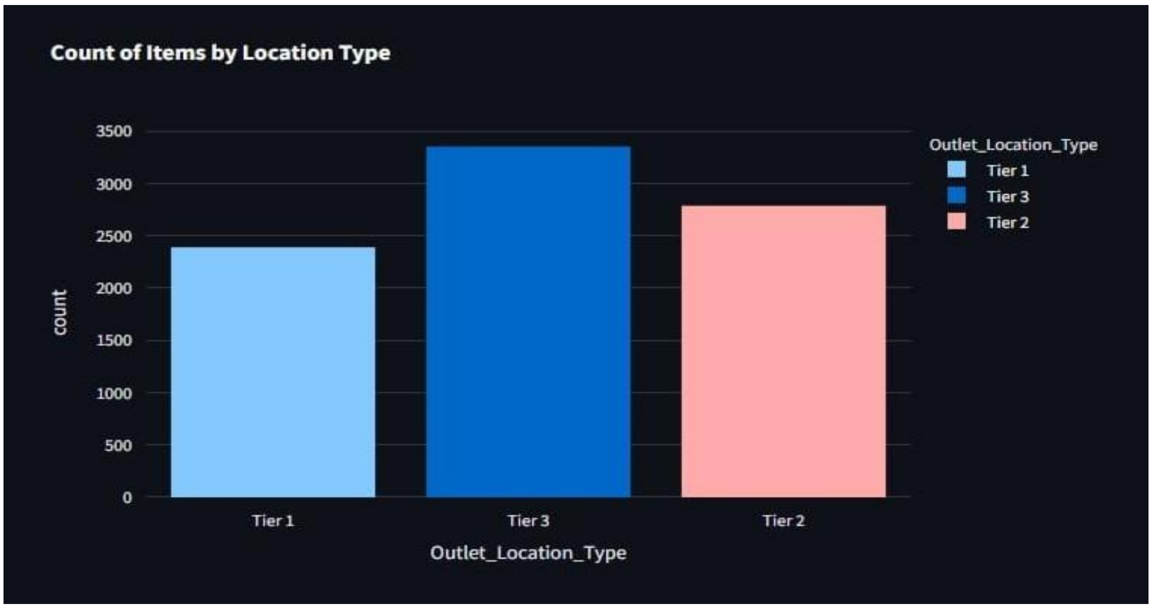
**5.1.4) MRP vs Sales:** A scatter plot is a type of graph that displays the relationship between two continuous variables. Here, it explores the relationship between maximum retail price and sales. It helps users identify potential correlations and patterns allowing for pricing strategy optimization.



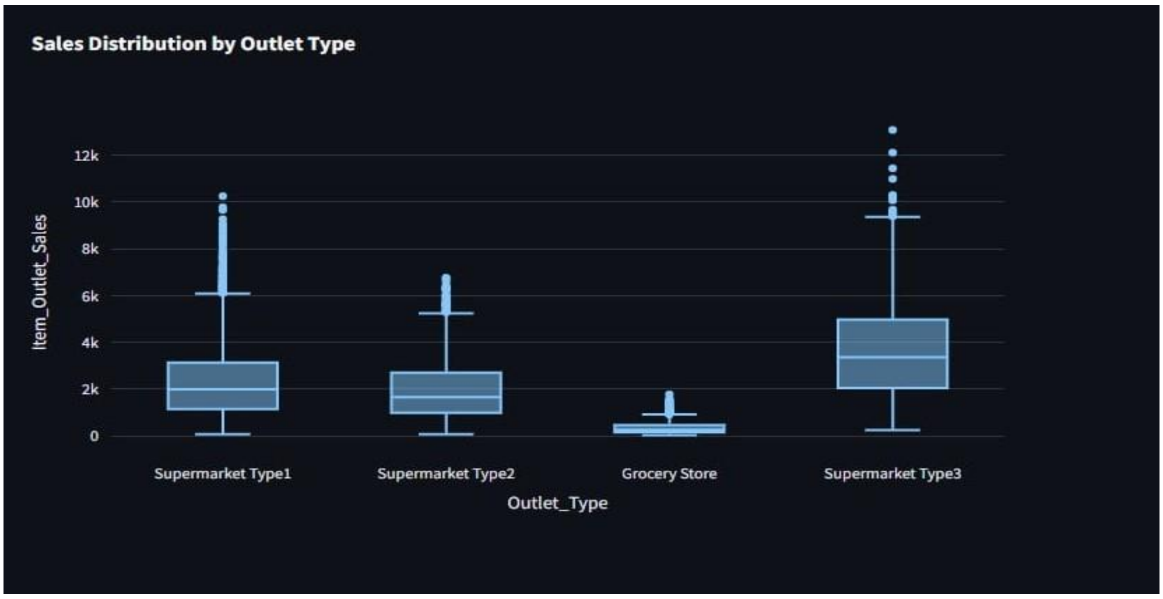
**5.1.5) Sales distribution by Outlet Size:** A box plot, sometimes referred to as a box and whisker plot, is a tool for statistical visualization that shows a dataset's distribution and summary statistics. Here, it displays the distribution of sales across different outlet size. It helps user in understanding how outlet size influences sales guiding decisions related to store expansion or downsizing.



**5.1.6) Counts of Items by Location Type:** The histogram shows the count of items based on the location type of outlets. It helps in understanding the distribution of items across different locations, supporting decisions related to market targeting and supply chain management.



**5.1.7) Sales Distribution by Outlet Type:** The box plot illustrates the distribution of sales across different types of outlets. It helps users understand how outlet type impacts sales, guiding decisions related to market strategies and resource allocation.



Overall, the sales prediction model dashboard serves as a powerful tool for organizations through data analysis and the application of various visualization techniques, decision-makers can obtain significant insights regarding sales trends, patterns, and possible problems. This makes it possible to allocate resources and make decisions based on data, as well as to recognize and seize growth opportunities.

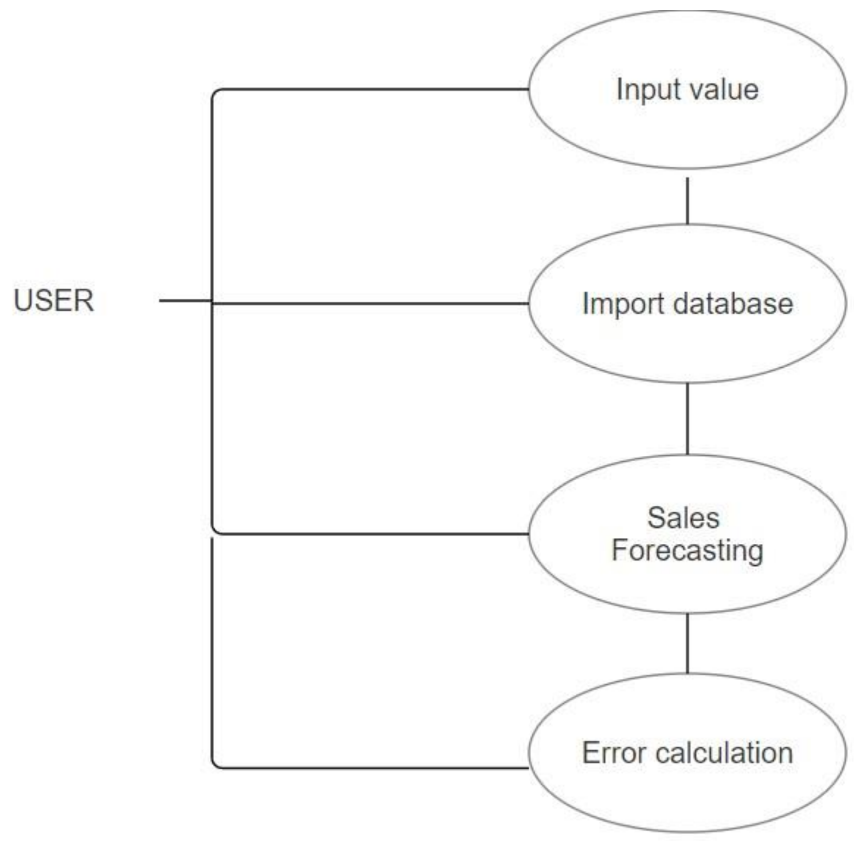
# Chapter 6
# Diagrams

## 6.1) UML Diagram:

The use case diagram is a technique used in the development of software or systems to capture the functional requirements of a system.

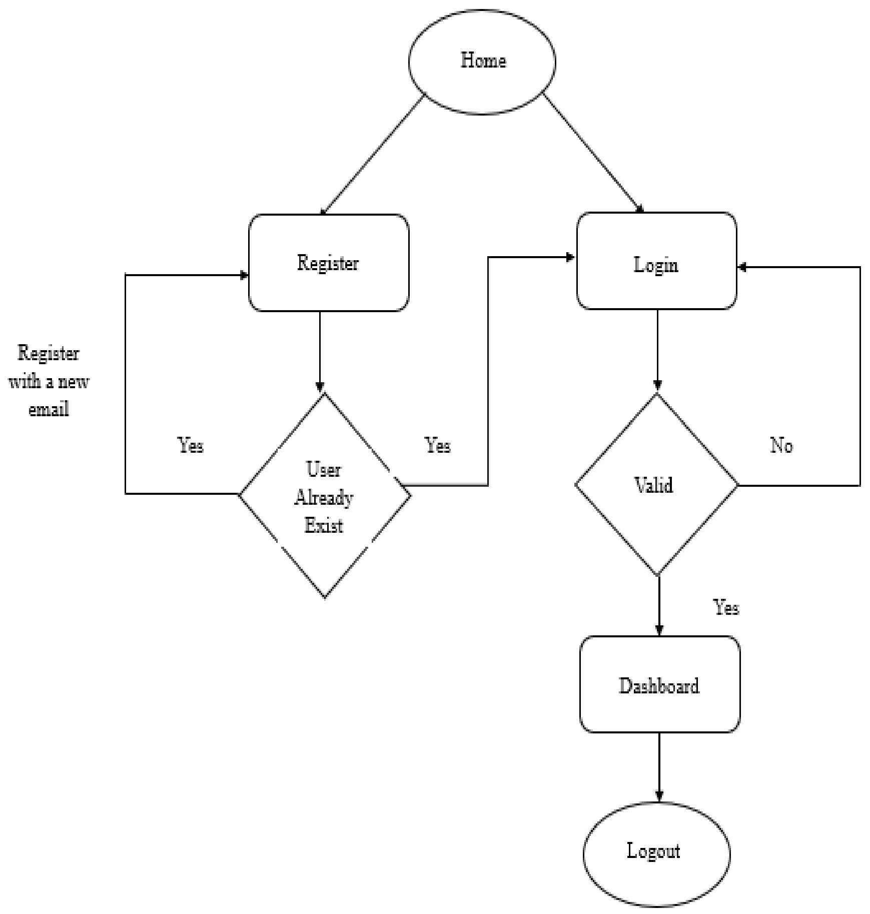
The user in this diagram represents someone who interacts with the system, such as a sales manager or analyst. This use case diagram provides an overview of the main functionalities and interactions of the sales forecasting application. It shows the flow of actions and inputs between the user and the system.

By using this diagram as the initial stage of development, the system developers can have a clear understanding of the required functionalities and how they should be implemented.



At the outset of the website, users are greeted with two prominent buttons: “Register” or “Login”. Upon selecting “Register”, users are prompted to provide their details. If the system detects an existing account with the provided information, users are redirected to a new page where they’re presented with two choices: to either proceed with logging in or to register with a different email address.

On the login page, users enter their credentials, and if all the information is accurate, they are granted access to the dashboard. The dashboard serves as a central hub where users can input data, visualize information, predict sales, and manage their account settings. Upon completion of their tasks, users can log out securely, ensuring the protection of their data and privacy.



# Chapter 7

# Theory

## 7.1) Limitations:

1. Prediction models for sales rely on various assumptions and previous data. They might not be reliable in forecasting sales in unexpected or unusual scenarios, like worldwide pandemics or economic downturns.
2. The accuracy of prediction models depends on the completeness and quality data of being used. If the data is insufficient or contains errors, it can affect the reliability of the predictions.
3. Sales prediction models may not take into account external factors that can significantly impact sales, such as changes in consumer preferences, competitor actions, or government regulations. These factors can influence sales outcomes but may not be captured in the model.

## 7.2) Targeted Users:

1. **Businesses:** The main target audience for sales prediction models are businesses and organizations that depend on accurate sales to decide on sales targets, resource allocation, and production planning. It will reduce the number of risks happening in business.
2. **Sales and marketing teams:** These models can help sales and marketing teams in determining the potential demand for products. discover target markets. They can have better plans and strategies to increase their sales if they make use of this model.
3. **Retailers**: Retail businesses can use sales prediction models to optimize pricing and provide them insights into future demand, allowing for more accurate inventory management, better product availability, and improved sales forecasting. It helps them forecast customer demand for specific products and plan their stock.
4. **Supplier:** With this model they will be able to foresee demand, leading to more efficient operations and better customer satisfaction.
5. **Investor:** Through data driven decisions investor trust and confidence can be strengthened.

## 7.3) Risks:

**1. Data Accuracy and Integrity:** The accuracy of sales predictions heavily relies on the quality and integrity of the data fed into the system. Any inaccuracies or inconsistencies in the data could lead to unreliable sales forecasts, impacting business decisions.

**2. Privacy and Data Security:** Sales data often contains sensitive information about customers, products, and financials. Any breach in the security measures of the dashboard app could result in unauthorized access to this confidential data, leading to legal and reputational consequences for the company.

**3. Overreliance on Predictive Models:** While predictive analytics can provide valuable insights into future sales trends, there is a risk of overreliance on these models. Unexpected market shifts, changes in consumer behavior, or external factors could render the predictions inaccurate, leading to misguided strategic decisions.

**4. User Error and Misinterpretation:** Users of the sales dashboard app may misinterpret the data or make errors in inputting information, resulting in flawed sales forecasts. Proper training and guidance are essential to ensure that users understand how to effectively use the dashboard and interpret the insights it provides.

**5. Regulatory Compliance:** Depending on the industry and geographical location, there may be regulatory requirements governing the collection, storage and user of sales data. Failure to comply with these regulations could result in penalties, fines, or legal actions against the company operating the sales dashboard app.

In conclusion, the project prediction model for sales has been effective in reaching its objectives of accurately forecasting sales and assisting in decision-making. The identification of underlying factors influencing sales performance was made possible by the model's ability to capture and analyze sales trends and patterns.