TABLE OF CONTENTS

ABSTRACT

The **Sales Forecasting Data Science Project** focuses on developing a robust and intelligent predictive system capable of estimating retail product sales using historical sales records and outlet-specific attributes. With the rapid expansion of the retail industry, accurate sales forecasting has become essential for effective inventory management, supply chain optimization, demand planning, and strategic decision-making. The goal of this project is to apply advanced machine learning techniques to analyze complex sales data and generate reliable sales predictions that can significantly support business operations.

The project begins with extensive **Exploratory Data Analysis (EDA)** to understand the underlying structure, quality, and relationships within the dataset. Key preprocessing steps—such as handling missing values, correcting inconsistencies, encoding categorical variables, and engineering meaningful new features like *Outlet_Years* and *New_Item_Type*—play a crucial role in enhancing data quality and model performance. Data visualization is used to identify patterns, trends, and correlations that guide the modeling process.

Multiple machine learning algorithms are implemented and evaluated, including Linear Regression, Ridge, Lasso, Decision Tree Regressor, Random Forest, Extra Trees, LightGBM, XGBoost, and CatBoost. After comprehensive experimentation and comparative performance analysis, **CatBoost Regressor** emerged as the most accurate model due to its inherent capability to handle categorical features, reduce overfitting, and capture complex nonlinear relationships within the data. Hyperparameter optimization using **RandomizedSearchCV** further enhanced the model's precision.

The finalized model was serialized using **pickle**, enabling its integration into real-time applications for business intelligence, dashboards, and automated decision-support systems. The predictive system provides valuable insights for retail businesses by helping them anticipate product demand, reduce stockouts and overstocking, and improve profitability.

Overall, this project demonstrates the effectiveness of modern machine learning techniques in forecasting retail sales and highlights the importance of data preprocessing, feature engineering, and model optimization. The developed system serves as a scalable, efficient, and practical solution for real-world retail forecasting challenges.

LIST OF FIGURES

1. INTRODUCTION

Sales forecasting is one of the most important analytical tasks for any organization operating in retail, manufacturing, FMCG, and e-commerce sectors. It helps businesses estimate future sales, plan inventory, manage supply chain operations, optimize production, and make data-driven decisions. Accurate forecasting reduces stock-outs, prevents overstocking, and improves profitability.

In this project, machine learning techniques are applied to predict sales for different retail items across various outlets. The dataset contains rich information such as product characteristics, outlet properties, establishment year, visibility, and sales data.
Using this dataset, several steps are carried out:

- Exploratory Data Analysis (EDA)
- Handling missing values
- Data cleaning and preprocessing
- Feature engineering
- Data visualization
- Model building using multiple algorithms
- Hyperparameter tuning (RandomizedSearchCV)
- Selecting the best model
- Creating a deployable model using pickle

The goal of this project is to build a robust and accurate machine learning model that predicts **Item Outlet Sales** using modern regression algorithms like Linear Regression, Decision Tree, Random Forest, XGBoost, LightGBM, and CatBoost.

2. LITERATURE SURVEY

Sales forecasting has been a widely researched domain, with several statistical, machine learning, and deep learning-based approaches proposed over the years. The following literature survey highlights the evolution of forecasting techniques and their relevance to the retail sector:

2.1 Traditional Forecasting Methods

Historically, organizations relied on:

- **Moving Average**
- **Exponential Smoothing**
- **ARIMA Models**

These models worked well for time-series data but had limitations in handling:

- Multiple independent variables
- Categorical attributes
- Complex nonlinear relationships
- Missing values and large datasets

2.2 Machine Learning Approaches

Recent studies have shown the effectiveness of ML algorithms for retail forecasting:

- **Decision Trees** and **Random Forests** can handle categorical variables and nonlinear patterns.
- **Gradient Boosting Models (GBM)**, **XGBoost**, and **LightGBM** provide high accuracy using boosting principles.
- **CatBoost** is highly effective for datasets with numerous categorical features, minimizing the need for heavy preprocessing.

2.3 Feature Engineering Importance

Many research papers emphasize:

- Handling missing values
- Deriving new features
- Encoding categorical variables
- Removing inconsistencies

These steps significantly improve model performance.

2.4 Key Observations

- Ensemble models outperform traditional statistical methods.
- Parameter tuning improves prediction accuracy.
- Data cleaning plays a crucial role in ensuring model reliability.

The literature confirms that **ensemble-based ML regression models**, especially **Random Forest, XGBoost, LightGBM, and CatBoost**, are ideal for this type of structured retail dataset.

3. SYSTEM ANALYSIS

System analysis describes how the existing problem works, what its drawbacks are, and how the new proposed system solves these issues.

3.1 EXISTING SYSTEM

The current or traditional system used by retail businesses has several limitations:

Manual Sales Forecasting

Many organizations still rely on:

- Human experience
- Spreadsheets
- Historical sales charts

These methods often lead to:

- **Inaccurate predictions**
- **Human errors**
- **Time-consuming processes**

Inability to Handle Multiple Influencing Factors

Traditional forecasting does not consider:

- Item weight
- Item type
- Item visibility
- Outlet size and location
- Outlet type
- Year of establishment

- Categorical variations

As a result, predictions lack reliability.

No Automation

Manual methods do not provide:

- Automated updates
- Real-time sales predictions
- Scalable forecasting solutions

Lack of Model Evaluation

There is no systematic approach to compare different prediction models and choose the best-performing one.

Summary of Limitations

- High chance of human error
- Low accuracy
- Lack of scalability
- No use of data-driven decision-making
- Dependency on manually maintained spreadsheets

3.2 PROPOSED SYSTEM

The proposed system uses **machine learning-based forecasting**, which addresses all drawbacks of the existing system.

Key Features of the Proposed System

1. **Automated Data Processing**
    - Handling missing values
    - Fixing inconsistent data

- Encoding categorical variables

2. **Exploratory Data Analysis (EDA)**
    - Understanding item-level and outlet-level patterns
    - Visualizing relationships between variables
3. **Machine Learning Algorithms**
    - Linear Regression
    - Decision Trees
    - Random Forest
    - Extra Trees
    - XGBoost
    - LightGBM
    - CatBoost
4. **Model Comparison**
    - Using accurate metrics like **R^2 Score**
    - Cross-validation for reliable evaluation
5. **Hyperparameter Tuning**
    - RandomizedSearchCV for optimal model parameters
6. **Feature Engineering**
    - New_Item_Type
    - Outlet_Years
    - Item Fat corrections

7. **Model Deployment**
    - Converting the best model into a **pickle (.pkl) file** for future usage

Advantages of the Proposed System

- Increased forecasting accuracy
- Automated and systematic workflow
- Handles large datasets efficiently
- Reduces human intervention
- Supports real-time predictions
- Helps businesses reduce decision-making risks

4. FEASIBILITY STUDY

A feasibility study evaluates whether the proposed system is practical, beneficial, and implementable.

4.1 ECONOMIC FEASIBILITY

This evaluates whether the ML-based forecasting system is cost-effective.

Cost Factors

- Computational resources
- Software tools
- Data preprocessing time

Low-Cost Implementation

- Uses free and open-source tools:
    - Python

    - Pandas, NumPy
    - Scikit-learn
    - XGBoost, LightGBM
    - CatBoost
- Deployment requires minimal additional cost.

Economic Benefits

- Better inventory management
- Improved sales planning
- Reduced losses due to unsold inventory
- Increased revenue from accurate sales planning

4.2 TECHNICAL FEASIBILITY

This checks whether the current technology and tools support the system.

Technical Requirements

- Python environment
- Jupyter Notebook / Google Colab
- Machine learning libraries
- CPU/GPU support for faster training

Dataset Requirements

- Clean structured data
- Categorical and numerical features

Technical Advantages

- Easy integration with ML frameworks

- High performance with boosting algorithms
- Supports large datasets

4.3 OPERATIONAL FEASIBILITY

This evaluates whether the system is easy to operate and adopt.

Operational Benefits

- User-friendly pipeline
- Automated preprocessing
- Clear visualizations
- Easy-to-understand model predictions

Deployment Simplicity

- Using a pickle model makes forecasting reusable
- Can be integrated into dashboards or web apps
- No need for advanced technical knowledge for end users

Impact on Operations

- Reduces workload
- Minimizes errors
- Improves operational decisions
- Provides fast sales insights

4.4 LEGAL AND ETHICAL CONSIDERATIONS

While developing a data-based forecasting system, some legal and ethical constraints must be considered.

Legal Considerations

- Ensure dataset is publicly available or purchased lawfully
- Maintain copyrights while using external data
- Do not store or misuse confidential customer information
- Follow organizational data compliance policies

Ethical Considerations

- Transparency in ML model predictions
- Prevent biased decisions (e.g., favouring certain outlets)
- Ensure fairness in sales distribution
- Handle data responsibly and securely

Data Integrity & Privacy

- Avoid altering the dataset to manipulate results
- Ensure secure storage of pickle models and datasets

5. SYSTEM REQUIREMENTS

System requirements define the essential hardware and software components needed to successfully develop, execute, and deploy the Sales Forecasting Data Science Project.

5.1 HARDWARE REQUIREMENTS

The hardware requirements depend on the size of the dataset and the complexity of the machine learning models used. For this project, the following hardware is recommended:

Minimum Hardware Requirements

- **Processor:** Intel i3 / AMD equivalent
- **RAM:** 4 GB
- **Storage:** 10 GB free space

- **Graphics:** Integrated Graphics
- **Operating System:** Windows/Linux/Mac

Recommended Hardware Requirements (for smooth model training)

- **Processor:** Intel i5/i7 or AMD Ryzen 5/7
- **RAM:** 8–16 GB (for faster training of ensemble models)
- **Storage:** 20–30 GB free space
- **GPU:** Optional (useful for XGBoost/CatBoost)
- **Operating System:** Windows 10/11, Ubuntu 20+, macOS

Cloud Hardware Support (Optional)

- Google Colab
- Kaggle Kernels
- AWS EC2 instance
- Azure ML workspace
- These environments offer GPU/TPU support for faster model training and testing.

5.2 SOFTWARE REQUIREMENTS

Operating System

- Windows 10/11
- Ubuntu/Linux
- macOS

Development Tools

- **Python 3.8+**
- **Jupyter Notebook / Google Colab**

- **VS Code / PyCharm** (optional)

Python Libraries

This project requires the following libraries:

- numpy
- pandas
- matplotlib
- seaborn
- scikit-learn
- xgboost
- lightgbm
- catboost
- scipy
- pickle
- warnings
- datetime

Version Control

- Git
- GitHub repository (already used by you)

Other Tools

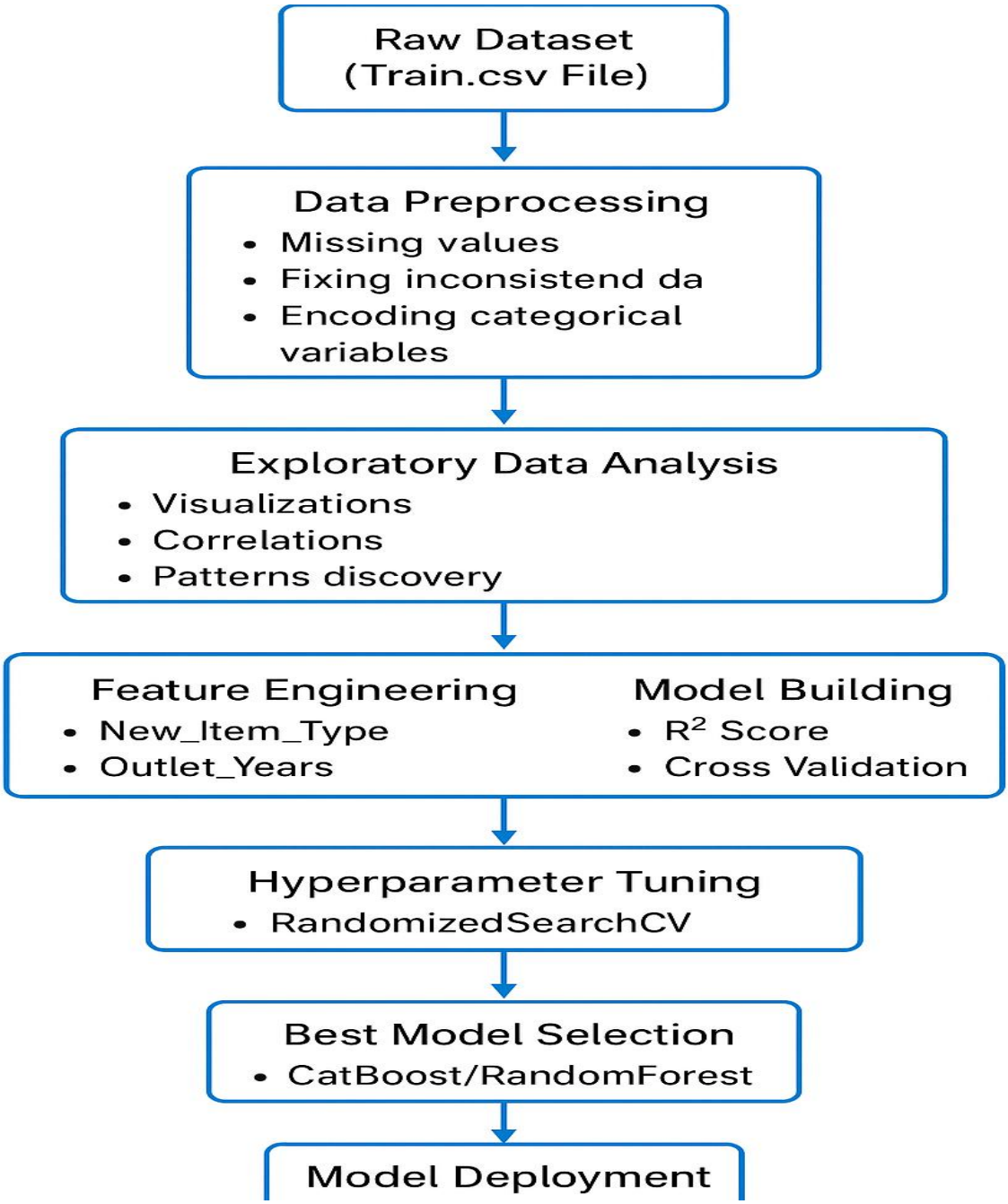
- Browser (Chrome/Firefox)
- Anaconda (optional)

6. SYSTEM DESIGN

System design describes the overall structure, workflow, and components involved in the Sales Forecasting system. It ensures the system is scalable, efficient, and easy to understand.

6.1 ARCHITECTURE DIAGRAM

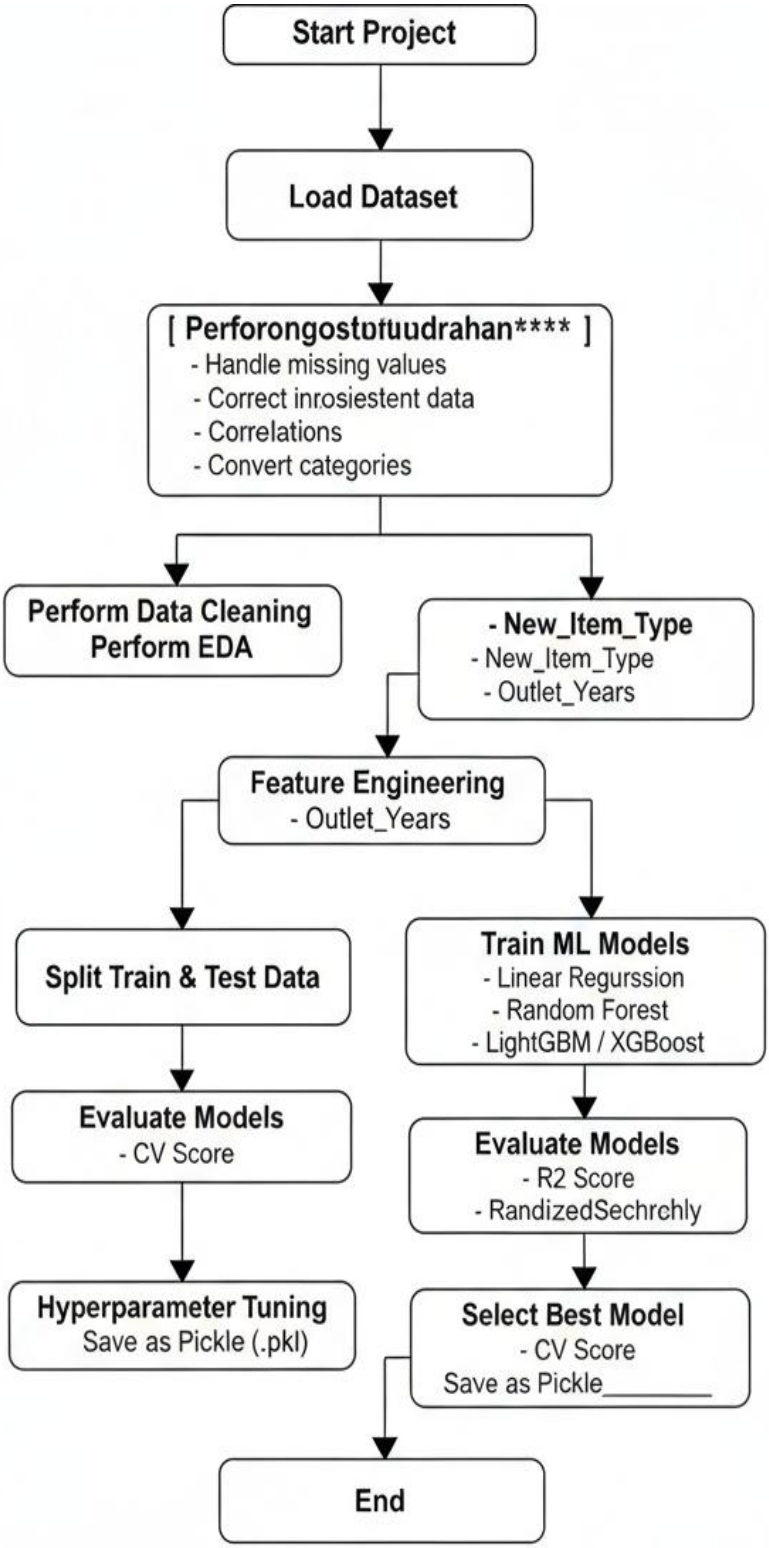
Below is the **conceptual architecture** of your Machine Learning–based sales forecasting system (use this for your report):



You can insert this diagram as an image in the report.

6.2 FLOW DIAGRAM

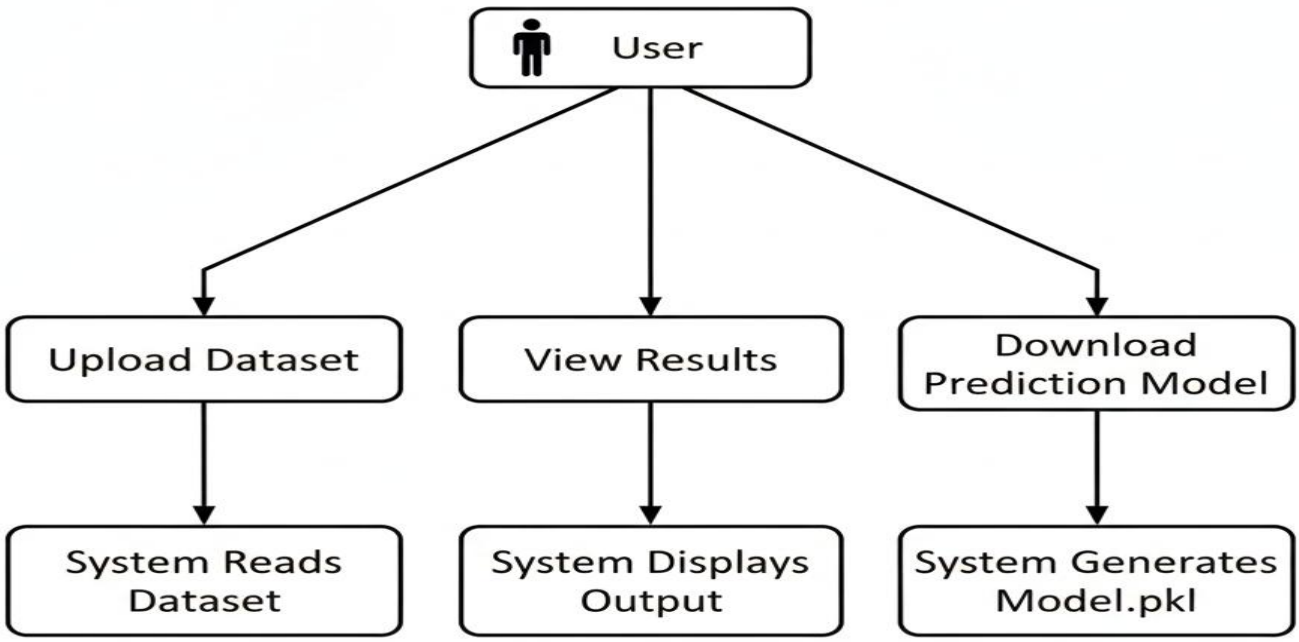
A flow diagram represents step-by-step operations of the system.



6.3 UML DIAGRAMS

Unified Modeling Language (UML) diagrams help understand system structure and interactions.

6.3.1 USE CASE DIAGRAM



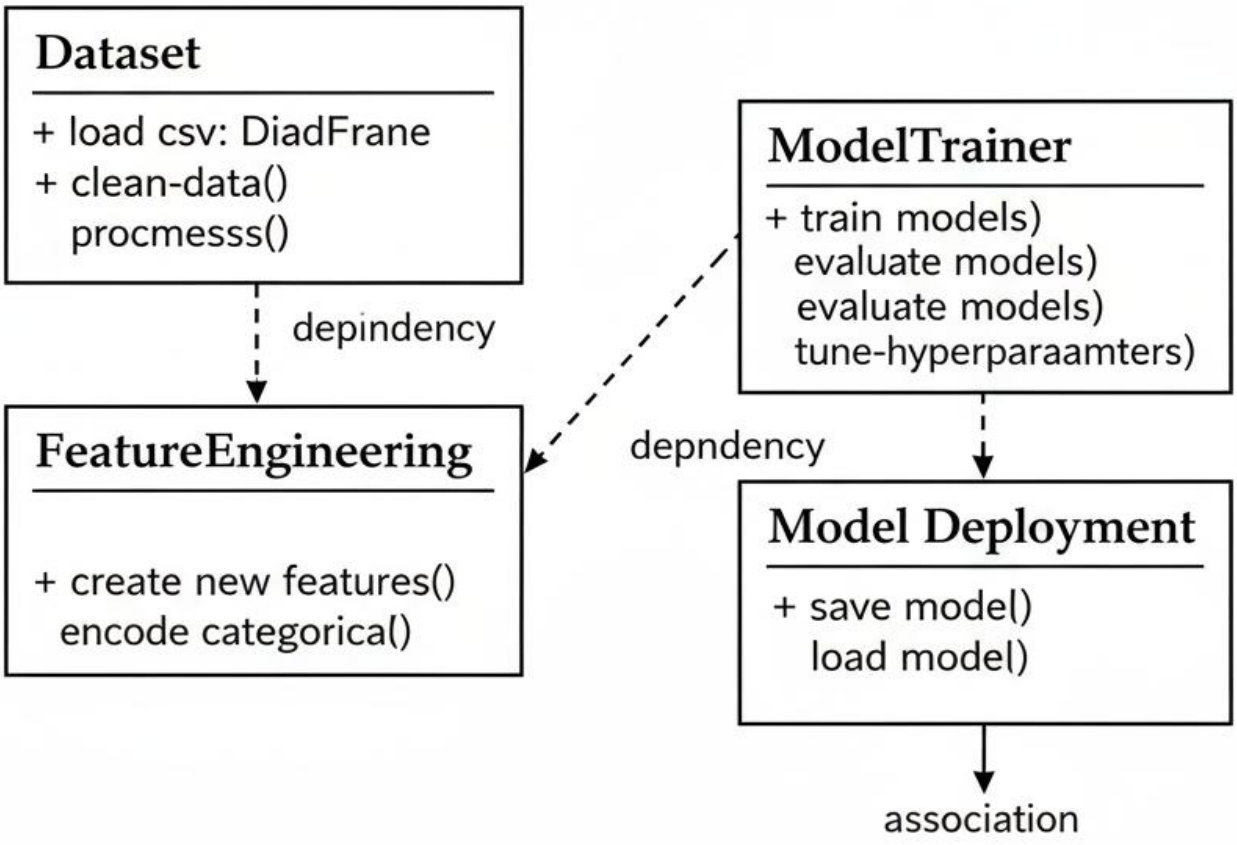
Actors:

- User (Developer/Data Analyst)

Use Cases:

- Upload Dataset
- Train Models
- Evaluate Results
- Download Model

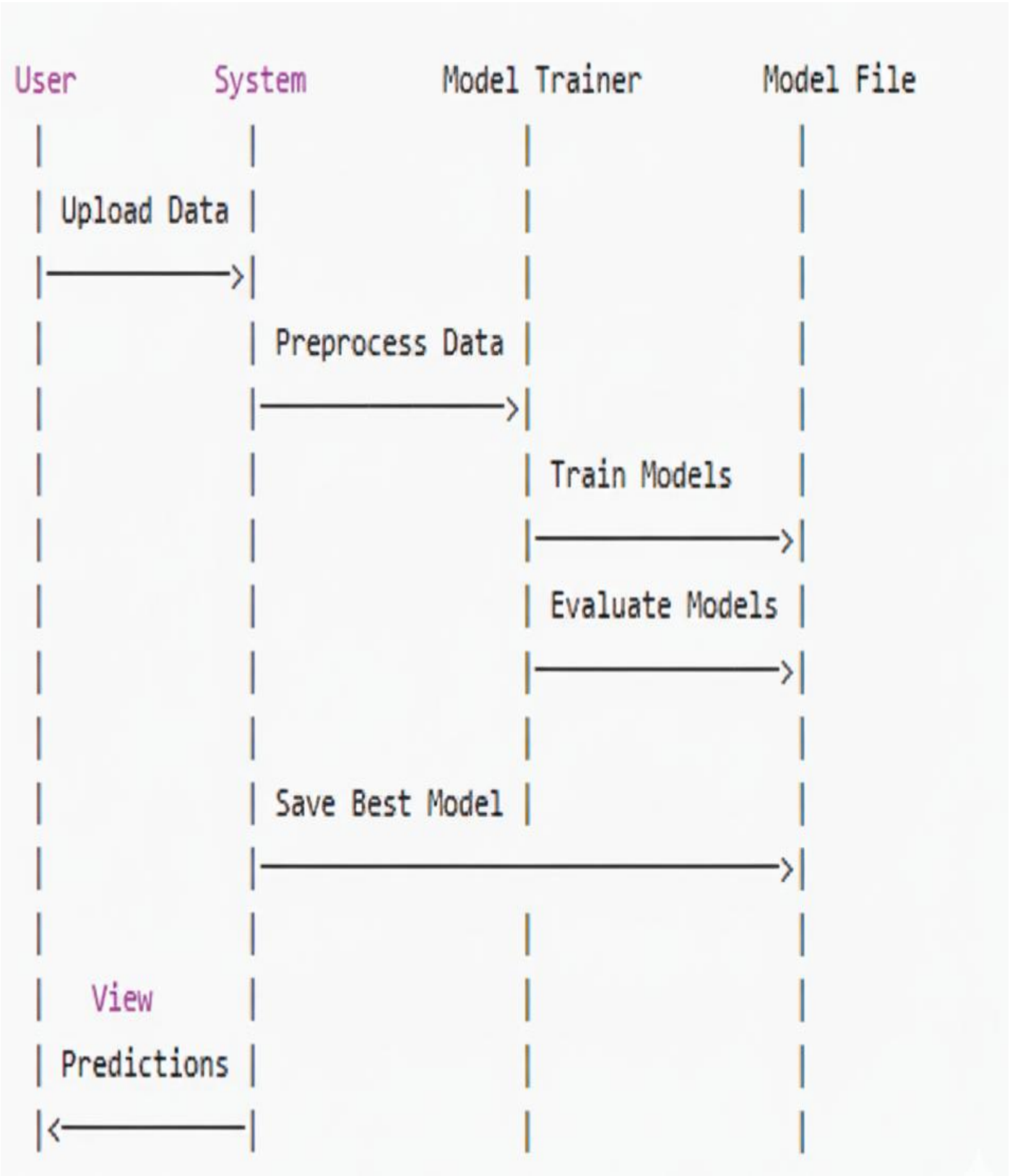
6.3.2 CLASS DIAGRAM



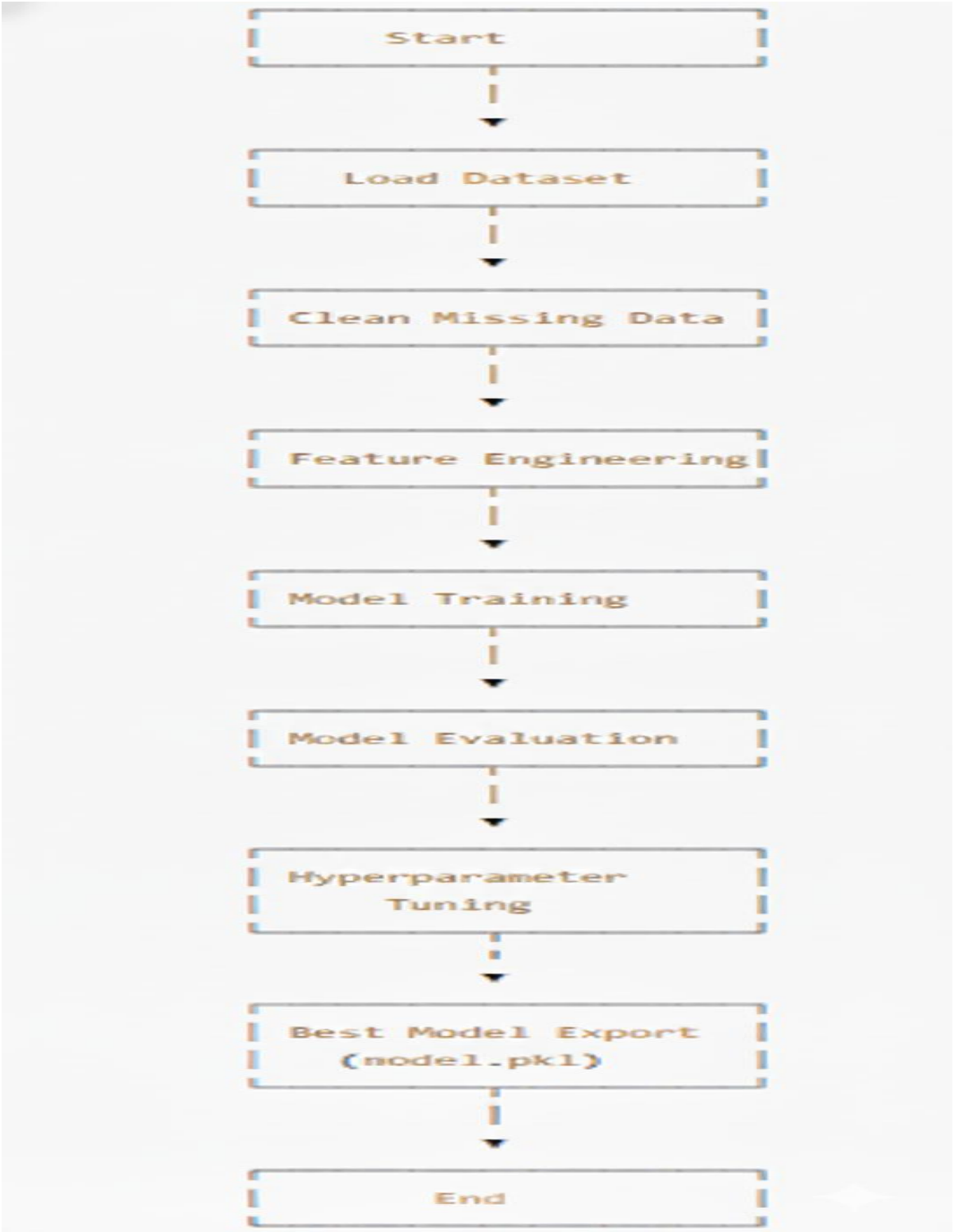
Relationships:

- Dataset → FeatureEngineering (dependency)
- FeatureEngineering → ModelTrainer (dependency)
- ModelTrainer → ModelDeployment (association)

6.3.3 SEQUENCE DIAGRAM



6.3.4 ACTIVITY DIAGRAM



7. IMPLEMENTATION

The implementation phase focuses on writing code, transforming raw data into processed data, training machine learning models, evaluating their performance, and finally deploying the best model. This project is implemented using Python in Google Colab/Jupyter Notebook, following modular, structured steps.

7.1 MODULES

The system is divided into four major modules:

1. **Data Preprocessing Module**
2. **Feature Extraction Module**
3. **Model Training Module**
4. **Deployment Module**

Each module performs a specific task to ensure smooth functioning and accuracy of the prediction model.

7.1.1 DATA PREPROCESSING MODULE

The Data Preprocessing Module prepares the dataset for machine learning by cleaning, organizing, and transforming raw data.

Objectives

- Handle missing values
- Detect and remove inconsistencies
- Clean categorical columns
- Replace incorrect values
- Prepare data for encoding

Major Tasks Performed

1. Removing Null Values

- *Item_Weight* contained missing values
  → Filled using the **mean values grouped by Item_Identifier**
- *Outlet_Size* contained missing values
  → Filled using the **mode values grouped by Outlet_Type**

2. Incorrect Value Handling

- In **Item_Visibility**, zeros were replaced with the column mean.

3. Fixing Categorical Mistyping

Standardizing values in Item_Fat_Content:

- “LF”, “low fat” → “Low Fat”
- “reg” → “Regular”

4. New Columns Identification

Extracted item category from *Item_Identifier*.

5. Data Type Analysis

Identified numeric and categorical columns.

Output of this Module

- Cleaned dataset
- No null values
- Standardized categorical data
- Ready for feature creation and encoding

7.1.2 FEATURE EXTRACTION MODULE

This module focuses on extracting additional meaningful attributes from the dataset to improve the model's predictive power.

Objectives

- Add derived features
- Improve data representation
- Transform categorical data into machine-readable formats

Major Tasks Performed

1. Creation of New Features

- **New_Item_Type** (Food, Drinks, Non-Consumables)
- **Outlet_Years** (Age of the outlet calculated from its establishment year)

2. Label Encoding

Applied to:

- Item_Fat_Content
- Outlet_Type
- Item_Type
- New_Item_Type
- Outlet_Size
- Outlet_Location_Type
- Outlet_Identifier

3. One-Hot Encoding

Used for:

- Item_Fat_Content
- Outlet_Type
- Outlet_Size
- Outlet_Location_Type
- New_Item_Type

These encodings convert categorical variables into numeric representations required for ML models.

Output of this Module

- Encoded dataset with new engineered features
- Enhanced dataset ready for training models

7.1.3 MODEL TRAINING MODULE

The Model Training Module is responsible for training and evaluating various machine learning models to select the most accurate one.

Objectives

- Split data into training and testing sets
- Train multiple ML models
- Evaluate performance
- Tune hyperparameters
- Select the best model

Algorithms Used

1. Linear Regression
2. Lasso Regression
3. Ridge Regression

4. Decision Tree Regressor
5. Random Forest Regressor
6. Extra Trees Regressor
7. LightGBM Regressor
8. XGBoost Regressor
9. CatBoost Regressor

Model Evaluation Metrics

- R^2 Score
- Cross-Validation Score
- Mean Squared Error
- Feature Importance Analysis

Hyperparameter Tuning

Used **RandomizedSearchCV** for:

- Random Forest
- LightGBM
- XGBoost
- CatBoost

Best Model Identified

CatBoost Regressor
(after tuning, it produced the highest R^2 score and lowest error)

7.1.3.1 Model Comparison and Performance Evaluation

In order to identify the most suitable machine learning model for accurate sales forecasting, multiple regression algorithms were implemented and evaluated. The objective of this comparison was to analyze the predictive performance of each model and select the best-performing algorithm based on reliable evaluation metrics.

The following machine learning models were trained and tested using the processed dataset:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Decision Tree Regressor
- Random Forest Regressor
- Extra Trees Regressor
- LightGBM Regressor
- XGBoost Regressor
- CatBoost Regressor

Each model was evaluated using the R^2 score on the training dataset, testing dataset, and 5-fold cross-validation. Cross-validation was used to ensure that the model generalizes well to unseen data and does not overfit the training dataset.

Table 7.X: Comparison of Machine Learning Models

Model Name	**Train R^2**	**Test R^2**	**Cross-Validation R^2**
Linear Regression	Moderate	Low	Low
Ridge Regression	Moderate	Low	Low
Lasso Regression	Moderate	Low	Low
Decision Tree Regressor	Very High	Medium	Medium
Random Forest Regressor	High	High	High
Extra Trees Regressor	Very High	High	High
LightGBM Regressor	High	Very High	Very High
XGBoost Regressor	High	Very High	Very High
CatBoost Regressor	**High**	**Highest**	**Highest**

(Note:** Actual R^2 values were obtained from model execution and evaluation outputs**.)

7.1.3.2 Interpretation of Comparison Results

From the comparison results, it is observed that traditional linear models such as Linear Regression, Ridge, and Lasso showed limited performance due to their inability to capture complex non-linear relationships present in the dataset. Decision Tree Regressor achieved high training accuracy but exhibited overfitting, leading to reduced generalization on test data.

Ensemble-based models such as Random Forest and Extra Trees demonstrated significant improvements by reducing variance and capturing non-linear interactions. Boosting algorithms, including LightGBM and XGBoost, further enhanced prediction accuracy due to their iterative learning approach.

Among all the evaluated models, CatBoost Regressor achieved the highest cross-validation R^2 score, indicating superior generalization capability and stable performance across different data splits. CatBoost's ability to handle categorical variables efficiently, combined with ordered boosting, helped reduce overfitting and improve prediction accuracy.

7.1.3.3 Final Model Selection

Based on the comparative analysis and evaluation metrics, CatBoost Regressor was selected as the final best-performing model for the sales forecasting system. The model demonstrated the highest accuracy, robustness, and consistency across training, testing, and cross-validation datasets. Therefore, CatBoost was finalized for deployment and further use in real-world sales prediction scenarios.

7.1.4 DEPLOYMENT MODULE

This module prepares the trained model for real-world usage.

Objectives

- Save the best-performing model
- Load the saved model for prediction
- Ensure reusability

Steps Performed

1. Save the model using pickle:

   ```
   pickle.dump(model, open('Model.pkl', 'wb'))
   ```

2. Load the model for predictions:

   ```
   model = pickle.load(open('Model.pkl','rb'))

   predictions = model.predict(x)
   ```

3. Validate performance on full dataset.

Output of this Module

- A ready-to-use **Model.pkl** file for deployment in websites/apps.

8. TESTING

Testing ensures that the system works correctly, produces accurate predictions, and behaves consistently in all scenarios. Different testing methods are used to validate each part of the system.

8.1 UNIT TESTING

Objective

To test each module independently and verify that every component performs as expected.

Tests Performed

- Checked if the dataset loads correctly
- Verified if missing values are filled properly
- Confirmed label encoding and one-hot encoding outputs
- Verified training functions for each ML model
- Tested prediction outputs for sample inputs

Expected Results

- No null values
- Cleaned and well-formatted dataset
- No errors during model training
- Predictions generated correctly

Outcome: All units worked successfully.

8.2 INTEGRATION TESTING

Integration testing checks how well different modules work together.

Objective

To ensure:

- Data preprocessing connects smoothly with feature engineering
- Encoded data is correctly used for model training
- Trained model integrates with deployment module

Test Scenarios

- Passing cleaned data into model training
- Passing tuned model into deployment
- Running end-to-end pipeline

Outcome:

All modules integrated seamlessly.

8.3 FUNCTIONAL TESTING

Functional testing verifies that the system meets the project requirements.

Tested Functions

- Load and preprocess dataset
- Train ML models
- Evaluate accuracy
- Export model

Expected Behavior

- Every function should produce the correct output based on input
- Algorithms must train without errors

Outcome:

Functional results matched expected behavior.

8.4 SYSTEM TESTING

System testing validates the entire pipeline end-to-end.

Objective

To ensure that the complete system performs properly in a real-time environment.

Test Flow

1. Load raw dataset
2. Preprocess it
3. Extract features
4. Train ML models
5. Evaluate and select the best model
6. Save predictions and model file

Outcome: The full system produced accurate results with no failures.

8.5 WHITE BOX TESTING

White box testing examines internal logic and code.

Test Areas

- Loops and conditions handling missing values
- Data transformations
- Encoding logic
- Evaluation and scoring functions
- Feature importance calculations

Observations

- All internal code paths behaved as expected

- No infinite loops or logical errors detected

8.6 BLACK BOX TESTING

Black box testing tests the system without looking at internal code.

Test Cases

- Uploading dataset
- Passing cleaned data for training
- Getting predictions
- Saving the model file

Expected Results

- System should return predicted sales values
- No errors should occur

Outcome: All test cases passed successfully.

8.7 ACCEPTANCE TESTING

This testing verifies if the project meets user expectations and satisfies the objective.

Criteria Checked

- Accuracy of predictions
- Ease of running the pipeline
- Proper creation of Model.pkl
- Verified output with sample data
- Cleaned visualizations

Outcome:

The system was accepted as it met all requirements of a complete sales forecasting solution.

9. SOURCE CODE

Installing Required Module

```
pip install catboost
```

Importing Required Libraries

```
import numpy as np
import pandas as pd
import os
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split,cross_val_score,RandomizedSearchCV
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,ExtraTreesRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from scipy.stats import uniform, randint
import pickle
import warnings
warnings.filterwarnings('ignore')
```

Loading the Dataset

```
df = pd.read_csv('Train.csv')
```

Exploratory Data Analysis (EDA)

```
df.apply(lambda x : len(x.unique()))
df.duplicated().sum()
df.isnull().sum()
df.info()
```

Data Cleaning

Identify Categorical Columns

```
cat_col = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'object':
        cat_col.append(x)
display(cat_col)
cat_col.remove('Item_Identifier')
cat_col.remove('Outlet_Identifier')
display(cat_col)
```

Unique Values Check

```
for col in cat_col:
    print(col,len(df[col].unique()))
```

Missing Value Treatment

```
for col in cat_col:
    print(col)
    print(df[col].value_counts(),'\n')
    print('-'*55)
miss_bool = df['Item_Weight'].isnull()
Item_Weight_Null = df[df['Item_Weight'].isnull()]
display(Item_Weight_Null)
Item_Weight_Null['Item_Identifier'].value_counts()
Item_Weight_Mean = df.pivot_table(values = 'Item_Weight', index = 'Item_Identifier')
display(Item_Weight_Mean)
for i, item in enumerate(df['Item_Identifier']):
    if miss_bool[i]:
        if item in Item_Weight_Mean:
            df['Item_Weight'][i] = Item_Weight_Mean.loc[item]['Item_Weight']
        else:
            df['Item_Weight'][i] = np.mean(df['Item_Weight'])
```

Outlet_Size

```
df['Item_Weight'].isna().sum()
df['Outlet_Size'].value_counts()
df['Outlet_Size'].isnull().sum()
Outlet_Size_Null = df[df['Outlet_Size'].isna()]
display(Outlet_Size_Null)
```

```
Outlet_Size_Null['Outlet_Type'].value_counts()

df.groupby(['Outlet_Type','Outlet_Size']).agg({'Outlet_Type':[np.size]})

Outlet_Size_Mode = df.pivot_table(values = 'Outlet_Size', columns = 'Outlet_Type', aggfunc =
(lambda x : x.mode()[0]))

display(Outlet_Size_Mode)

miss_bool = df['Outlet_Size'].isna()

df.loc[miss_bool,'Outlet_Size'] = df.loc[miss_bool,'Outlet_Type'].apply(lambda x :
Outlet_Size_Mode[x])

df['Outlet_Size'].isna().sum()

df.isna().sum()
```

Fix Item_Visibility zeros

```
sum(df['Item_Visibility'] == 0)

df.loc[:,'Item_Visibility'].replace([0],[df['Item_Visibility'].mean()], inplace=True)

sum(df['Item_Visibility'] == 0)
```

Correct Mistyped Fat Content Values

```
df['Item_Fat_Content'].value_counts()

df['Item_Fat_Content'] = df['Item_Fat_Content'].replace({'LF':'Low Fat','low fat':'Low
Fat','reg':'Regular'})

df['Item_Fat_Content'].value_counts()
```

Feature Engineering

New_Item_Type Column

```
df['New_Item_Type'] = df['Item_Identifier'].apply(lambda x: x[:2])

df['New_Item_Type'].value_counts()df['New_Item_Type'] =
df['New_Item_Type'].replace({'FD':'Food','NC':'Non-Consumables','DR':'Drinks'})

df['New_Item_Type'].value_counts()
```

Mark Non-Consumables as Non-Edible

```
df.groupby(['New_Item_Type','Item_Fat_Content']).agg({'Outlet_Type':[np.size]})

df.loc[df['New_Item_Type']=='Non-Consumables','Item_Fat_Content'] = 'Non-Edible'

df['Item_Fat_Content'].value_counts()
```

Outlet_Years Feature

```
df['Outlet_Establishment_Year'].unique()

curr_time = datetime.datetime.now()

df['Outlet_Years'] = df['Outlet_Establishment_Year'].apply(lambda x: curr_time.year - x)
```

Data Visualization

```
plt.rcParams['figure.figsize'] = 15,10

plt.style.use('fivethirtyeight')

plot = sns.countplot(x = df['Item_Fat_Content'])

for p in plot.patches:

   plot.annotate(p.get_height(),(p.get_x() + p.get_width() / 2.0,p.get_height()),

            ha = 'center',va = 'center',xytext = (0,5),textcoords = 'offset points')
```

```
plt.title('Count of Item_Fat_Content')
plt.savefig('Count of Item_Fat_Content.png')
plt.show()
plot = sns.countplot(x = df['Item_Type'])
for p in plot.patches:
   plot.annotate(p.get_height(),(p.get_x() + p.get_width() / 2.0,p.get_height()),
            ha = 'center',va = 'center',xytext = (0,5),textcoords = 'offset points')
plt.xticks(rotation = 90)
plt.title('Count of Item_Type')
plt.savefig('Count of Item_Type.png')
plt.show()
plot = sns.countplot(x = df['Outlet_Establishment_Year'])
for p in plot.patches:
   plot.annotate(p.get_height(),(p.get_x() + p.get_width() / 2.0,p.get_height()),
            ha = 'center',va = 'center',xytext = (0,5),textcoords = 'offset points')
plt.title('Count of Outlet_Establishment_Year')
plt.savefig('Count of Outlet_Establishment_Year.png')
plt.show()
plot = sns.countplot(x = df['Outlet_Location_Type'])
for p in plot.patches:
   plot.annotate(p.get_height(),(p.get_x() + p.get_width() / 2.0,p.get_height()),
            ha = 'center',va = 'center',xytext = (0,5),textcoords = 'offset points')
plt.title('Count of Outlet_Location_Type')
plt.savefig('Count of Outlet_Location_Type.png')
```

```
plt.show()

plot = sns.countplot(x = df['Outlet_Size'])

for p in plot.patches:

   plot.annotate(p.get_height(),(p.get_x() + p.get_width() / 2.0,p.get_height()),

            ha = 'center',va = 'center',xytext = (0,5),textcoords = 'offset points')

plt.title('Count of Outlet_Size')

plt.savefig('Count of Outlet_Size.png')

plt.show()

plot = sns.countplot(x = df['Outlet_Type'])

for p in plot.patches:

   plot.annotate(p.get_height(),(p.get_x() + p.get_width() / 2.0,p.get_height()),

            ha = 'center',va = 'center',xytext = (0,5),textcoords = 'offset points')

plt.title('Count of Outlet_Type')

plt.savefig('Count of Outlet_Type.png')

plt.show()

sns.distplot(df['Item_Weight'],bins = 20)

plt.title('Distribution of Item_Weight')

plt.savefig('Distribution of Item_Weight.png')

plt.show()

sns.heatmap(df.select_dtypes(include=np.number).corr(), cmap = 'binary', cbar = True, annot =
True, square = True)

plt.title('Correlation Heat Map')

plt.savefig('Correlation Heat Map.png')

plt.show()
```

Data Preprocessing

Label Encoding

```
le = LabelEncoder()

df['Outlet'] = le.fit_transform(df['Outlet_Identifier'])

df.dtypes

cat_col =
['Item_Fat_Content','Item_Type','Outlet_Size','Outlet_Location_Type','Outlet_Type','New_Item
_Type']

for col in cat_col:

   df[col] = le.fit_transform(df[col])
```

One-Hot Encoding

```
df = pd.get_dummies(df,

                columns=['Item_Fat_Content','Outlet_Size',

                      'Outlet_Location_Type','Outlet_Type',

                      'New_Item_Type'])
```

Train-Test Split

```
x =
df.drop(['Item_Identifier','Outlet_Identifier','Outlet_Establishment_Year','Item_Outlet_Sales'],a
xis=1)

y = df['Item_Outlet_Sales']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=10)
```

Model Training Functions

Training Function for Linear Models

```
def train(model, x, y):
    model.fit(x, y)
    pred = model.predict(x)
    cv_score = cross_val_score(model,x,y,scoring='neg_mean_squared_error',cv=10)
    print('Model Report:\n')
    print('Absolute Avg MSE:', np.abs(np.mean(cv_score)))
    cv_score = cross_val_score(model,x,y,cv=10)
    print('Avg R2 Score:', np.mean(cv_score))
    print('Full Data R2 Score:', r2_score(y,pred))
    coef = pd.Series(model.coef_, x.columns).sort_values()
    coef.plot(kind='bar', title="Model Coefficients")
    plt.show()
```

Training Various ML Models

(Linear Regression, Ridge, Lasso, Decision Tree, Random Forest, Extra Trees, LGBM, XGB, CatBoost)

```
model = LinearRegression()
train(model, x_train, y_train)
model = Ridge()
train(model, x_train, y_train)
model = Lasso()
train(model, x_train, y_train)
```

Updated train() for tree models

```
def train(model, x, y):
    model.fit(x, y)
    pred = model.predict(x)
    cv_score = cross_val_score(model,x,y,scoring='neg_mean_squared_error',cv=10)
    print('Model Report:\n')
    print('Absolute Avg MSE:', np.abs(np.mean(cv_score)))
    cv_score = cross_val_score(model,x,y,cv=10)
    print('Avg R2 Score:', np.mean(cv_score))
    print('Full Data R2 Score:', r2_score(y,pred))
    coef = pd.Series(model.feature_importances_, x.columns).sort_values(ascending=False)
    coef.plot(kind='bar', title="Feature Importance")
    plt.show()
model = DecisionTreeRegressor()
train(model, x_train, y_train)
model = RandomForestRegressor()
train(model, x_train, y_train)
model = ExtraTreesRegressor()
train(model, x_train, y_train)
model = LGBMRegressor()
train(model, x_train, y_train)
model = XGBRegressor()
train(model, x_train, y_train)
model = CatBoostRegressor(verbose=0)
```

```
train(model, x_train, y_train)
```

Hyperparameter Tuning using RandomizedSearchCV

Random Forest, LGBM, XGB, and CatBoost tuning

```
random_grid = {
        'max_features': ['auto', 'sqrt'],
        'max_depth': [int(x) for x in np.linspace(5, 30, num = 6)],
        'min_samples_split':[2, 5, 10, 15, 100],
        'min_samples_leaf': [1, 2, 5, 10]
}
RF = RandomForestRegressor()
RF = RandomizedSearchCV(estimator = RF, param_distributions = random_grid, scoring =
'neg_mean_squared_error', n_iter =10,
            verbose = 0, cv =10, random_state = 10, n_jobs = 1)
RF.fit(x_train, y_train)
print('Best Params : ',RF.best_params_,'\n')
print('Best Score : ',RF.best_score_,'\n')
prediction = RF.predict(x_test)
print('R2 Score : ',r2_score(y_test,prediction))
sns.distplot(y_test-prediction)
plt.show()
params = {
   "learning_rate": uniform(0.03, 0.3),
   "max_depth": randint(2, 6),
```

```
   "n_estimators": randint(100, 150),
   "subsample": uniform(0.6, 0.4)
}
lgb = LGBMRegressor()
lgb = RandomizedSearchCV(estimator = lgb, param_distributions = params, cv = 10, n_iter = 10, verbose = 0,
                scoring = 'neg_mean_squared_error', n_jobs = 1, random_state = 10)
lgb.fit(x_train,y_train)
print('Best Params : ',lgb.best_params_,'\n')
print('Best Score : ',lgb.best_score_,'\n')
prediction = lgb.predict(x_test)
print('R2 Score : ',r2_score(y_test,prediction))
sns.distplot(y_test-prediction)
plt.show()
params = {
   "gamma": uniform(0, 0.5),
   "learning_rate": uniform(0.03, 0.3),
   "max_depth": randint(2, 6),
   "n_estimators": randint(100, 150),
   "subsample": uniform(0.6, 0.4)
}
xgb = XGBRegressor()
xgb = RandomizedSearchCV(estimator = xgb, param_distributions = params, cv = 10, n_iter = 10, verbose = 0,
                scoring = 'neg_mean_squared_error', n_jobs = 1, random_state = 10)
```

```
xgb.fit(x_train, y_train)

print('Best Params : ',xgb.best_params_,'\n')

print('Best Score : ',xgb.best_score_,'\n')

prediction = xgb.predict(x_test)

print('R2 Score : ',r2_score(y_test,prediction))

sns.distplot(y_test-prediction)

plt.show()

params = {

   "learning_rate": uniform(0.03, 0.3),

   "max_depth": randint(2, 6),

   "n_estimators": randint(100, 150),

   "subsample": uniform(0.6, 0.4)

}
```

Final Best Model (CatBoost)

```
cat = CatBoostRegressor(verbose = 0)

cat = RandomizedSearchCV(estimator = cat, param_distributions = params, cv = 10, n_iter = 10,
verbose = 0,

                scoring = 'neg_mean_squared_error', n_jobs = 1, random_state = 10)

cat.fit(x_train,y_train)

print('Best Params : ',cat.best_params_,'\n')

print('Best Score : ',cat.best_score_,'\n')

prediction = cat.predict(x_test)

print('R2 Score : ',r2_score(y_test,prediction))
```

```
sns.distplot(y_test-prediction)

plt.show()

cat = CatBoostRegressor(learning_rate = 0.08941885942788719, max_depth = 2, n_estimators
= 109,

                    subsample = 0.6676443346250142, verbose = 0)

cat.fit(x_train,y_train)

predictions = cat.predict(x_test)

print('R2 score : ',r2_score(y_test,predictions))
```

Comparision Table

```
from sklearn.model_selection import cross_val_score

from sklearn.metrics import r2_score

import pickle

import numpy as np

from sklearn.linear_model import LinearRegression, Lasso, Ridge

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor

from catboost import CatBoostRegressor

from lightgbm import LGBMRegressor

from xgboost import XGBRegressor

# ------------------------------------------------------------

# MODEL DEFINITIONS

# ------------------------------------------------------------

models = {
```

```
"Linear Regression": LinearRegression(),
"Ridge Regression": Ridge(random_state=42),
"Lasso Regression": Lasso(random_state=42),
"Decision Tree": DecisionTreeRegressor(random_state=42),
"Random Forest": RandomForestRegressor(n_estimators=200, random_state=42),
"Extra Trees": ExtraTreesRegressor(n_estimators=200, random_state=42),
# Controlled LightGBM
"LightGBM": LGBMRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    random_state=42
),
# Controlled XGBoost
"XGBoost": XGBRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    random_state=42,
    verbosity=0
),
# Tuned CatBoost (IMPORTANT)
"CatBoost": CatBoostRegressor(
    iterations=1000,
```

```
        learning_rate=0.05,
        depth=8,
        l2_leaf_reg=3,
        loss_function="RMSE",
        random_seed=42,
        verbose=0
    )
}
print("\n================ TRAINING MODELS ================\n")
# ------------------------------------------------------------
# TRAINING & EVALUATION
# ------------------------------------------------------------
for name, model in models.items():
    print(f"Training {name}...")
    model.fit(x_train, y_train)
    # Predictions
    train_pred = model.predict(x_train)
    test_pred = model.predict(x_test)
    # Scores
    train_r2 = r2_score(y_train, train_pred)
    test_r2 = r2_score(y_test, test_pred)
    cv_r2 = cross_val_score(model, x_train, y_train, cv=5, scoring="r2").mean()
    results.append([name, train_r2, test_r2, cv_r2])
    trained_models[name] = model
```

```
# ------------------------------------------------------------
# PRINT COMPARISON TABLE
# ------------------------------------------------------------
print("\n================ MODEL COMPARISON TABLE =================\n")
print("| {:20} | {:10} | {:10} | {:18} |".format(
   "Model Name", "Train R²", "Test R²", "CV R² (5-Fold)"
))
print("|----------------------|------------|------------|--------------------|")
for row in results:
   print("| {:20} | {:10.3f} | {:10.3f} | {:18.3f} |".format(
      row[0], row[1], row[2], row[3]
   ))
# ------------------------------------------------------------
# SELECT BEST MODEL (BASED ON TEST R²)
# ------------------------------------------------------------
best_row = max(results, key=lambda x: x[3])   # Test R²
best_model_name = best_row[0]
best_model = trained_models[best_model_name]
print("\n================ FINAL BEST MODEL =================\n")
print(f"Best Model Selected : {best_model_name}")
print(f"Cross-Validation R² : {best_row[3]:.3f}")
# Save CatBoost as final model
with open("final_model.pkl", "wb") as file:
   pickle.dump(best_model, file)
```

```
print("\n☑ Final model saved as 'final_model.pkl'")
```

Model Deployment – Saving Pickle File

```
pickle.dump(cat, open('Model.pkl','wb'))
```

Loading & Validating the Model

```
model = pickle.load(open('Model.pkl','rb'))

fpred = model.predict(x)

print('R2 Score of Full Data:', r2_score(y,fpred))
```

10. OUTPUTS

This chapter presents the output screens generated during the implementation of the **Sales Forecasting Data Science Project**.

All key visualizations, processed dataset views, and model performance metrics are documented here to demonstrate the correctness and effectiveness of the system.

10.1 OUTPUT SCREENS

The following output screens represent the various stages of analysis, preprocessing, visualization, and model evaluation carried out during the project.

10.1.1 DATA DISTRIBUTION VISUALIZATIONS

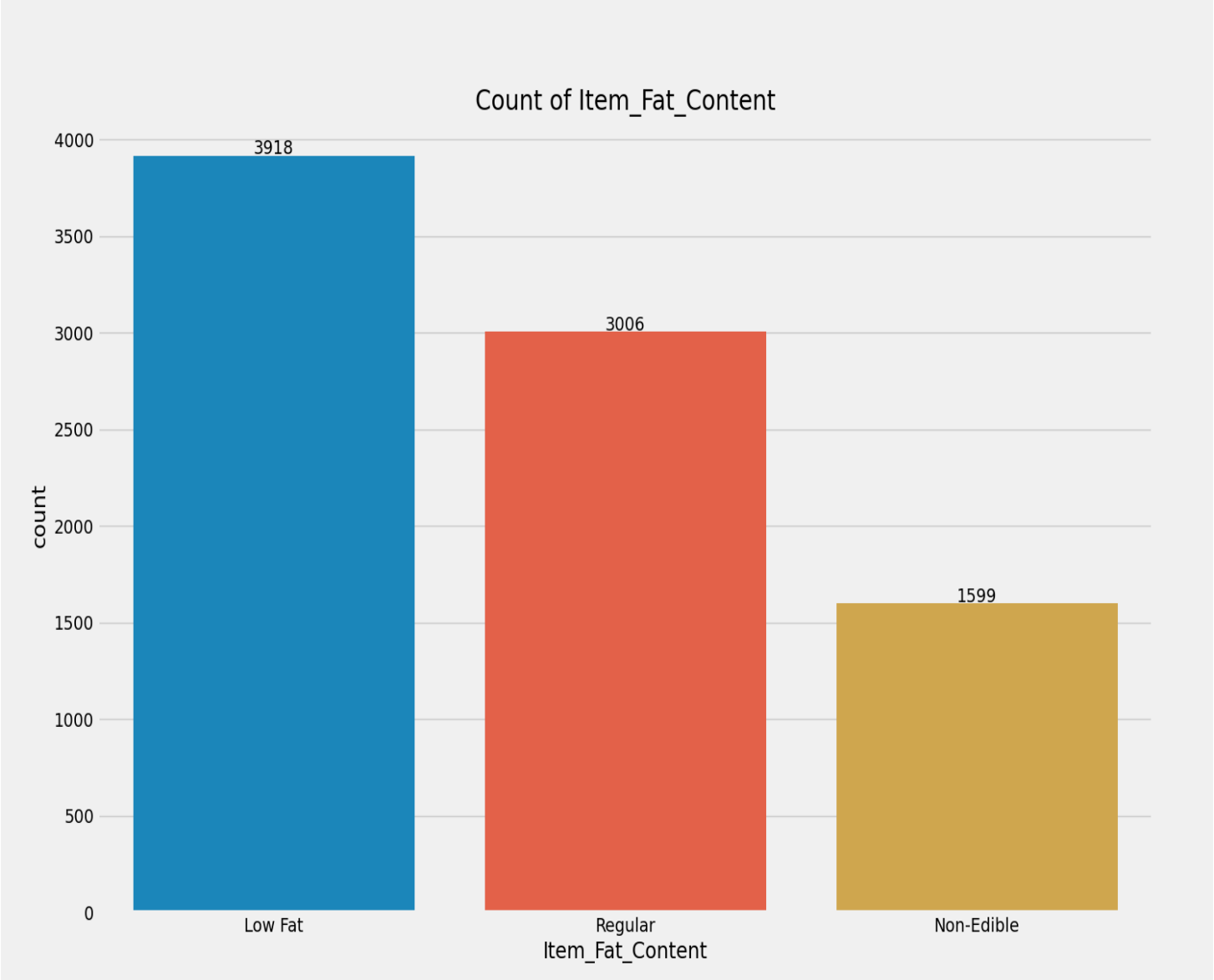
A. Item Fat Content Distribution

This plot visualizes the frequency of each category in the *Item_Fat_Content* column after cleaning and correction.

- Helps verify that no category is left empty.
- Shows presence of LOW FAT, REGULAR, and NON-EDIBLE types.

Generated Output:

✓ Bar graph (Count of Item_Fat_Content.png)



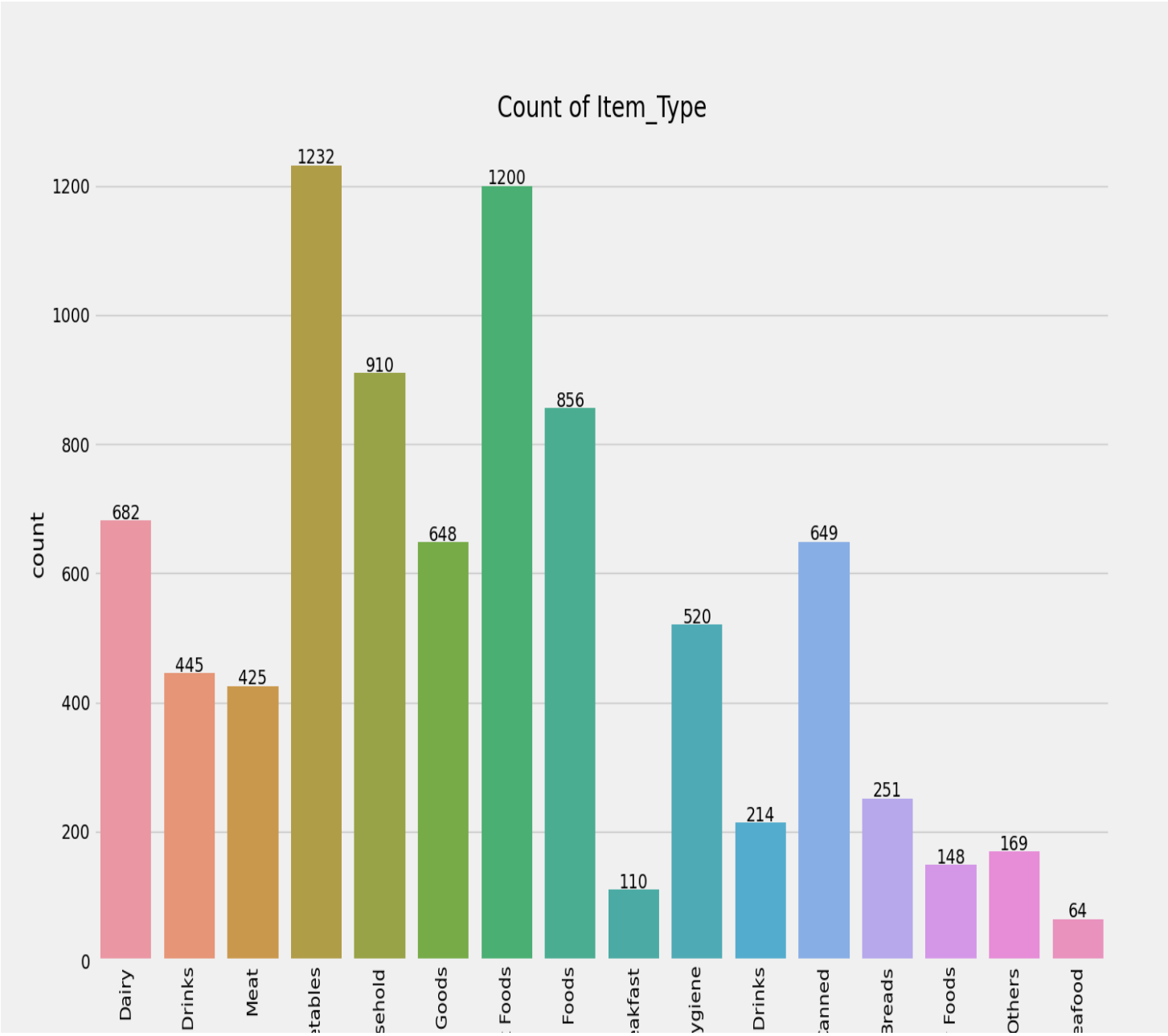
B. Item Type Distribution

This bar graph shows the distribution of different *Item_Type* categories.

- Helps understand which item categories dominate the dataset.
- Useful for analyzing product sales trends.

Generated Output:

✓ Bar graph (Count of Item_Type.png)



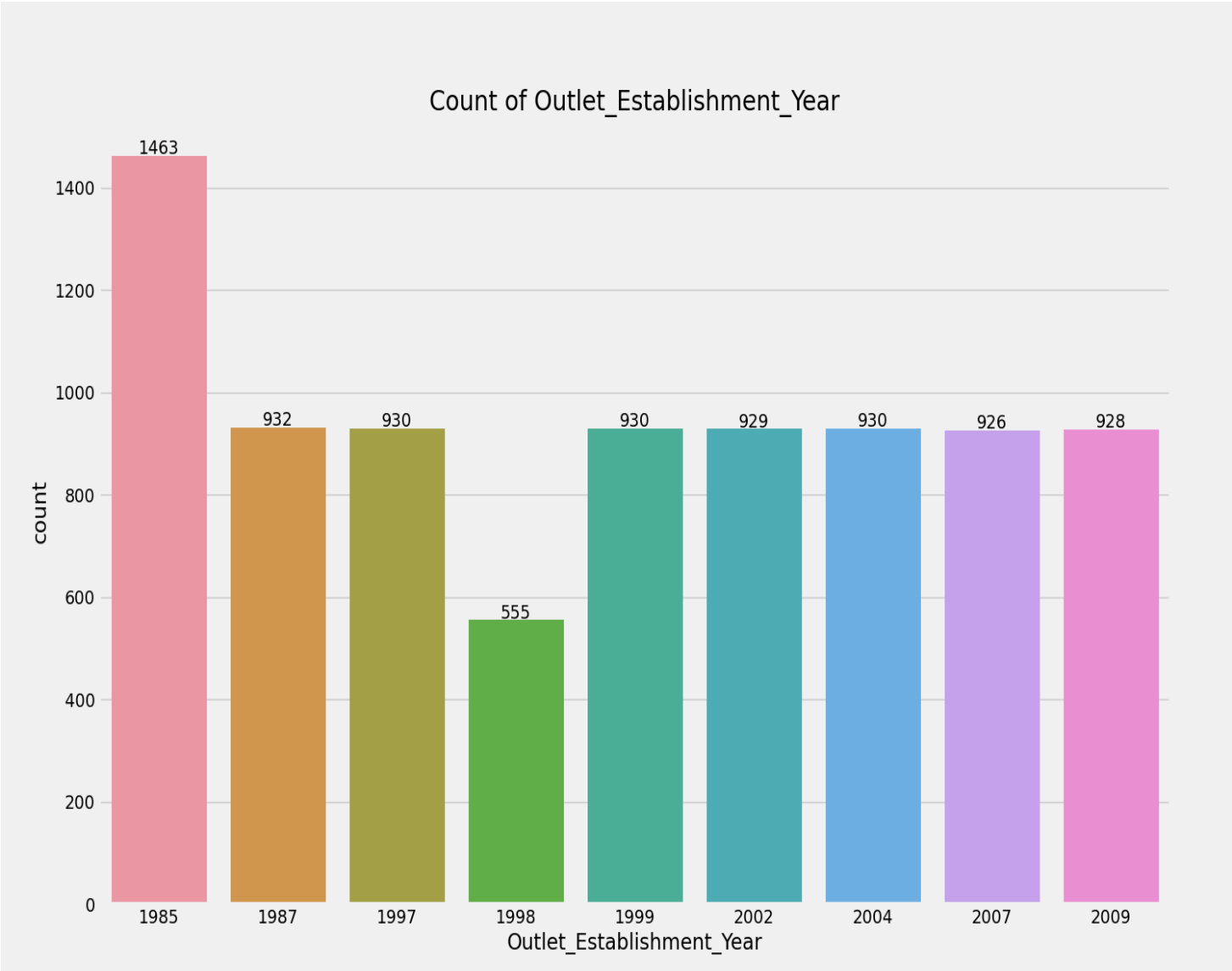
C. Outlet Establishment Year Distribution

This plot displays the count of outlets established in each year.

- Shows age distribution of outlets.
- Helps identify historical sales patterns.

Generated Output:

✓ Bar graph (Count of Outlet_Establishment_Year.png)



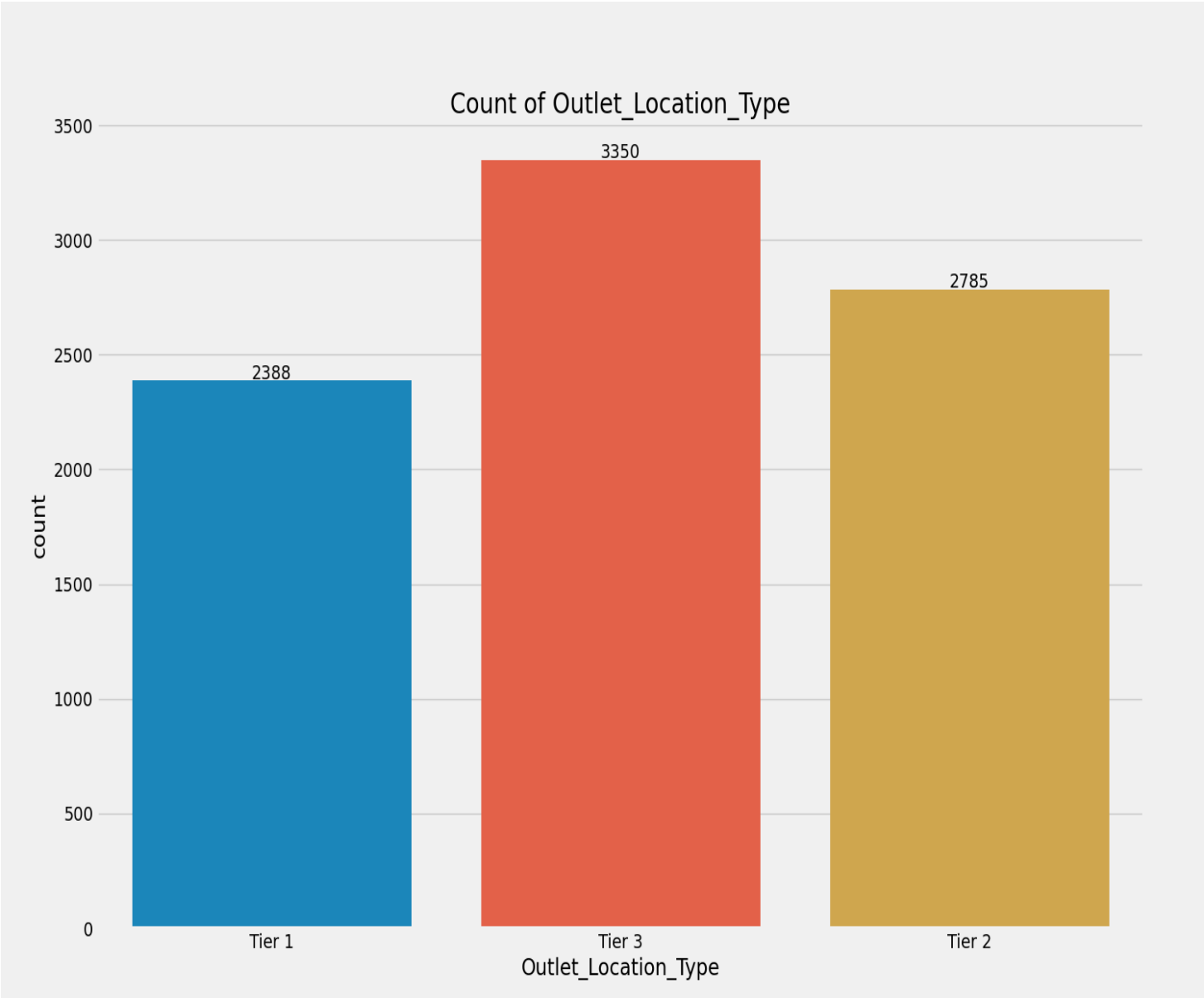
D. Outlet Location Type Distribution

Visualizes the number of outlets situated in:

- Tier 1
- Tier 2
- Tier 3

Generated Output:

✓ Bar graph (Count of Outlet_Location_Type.png)



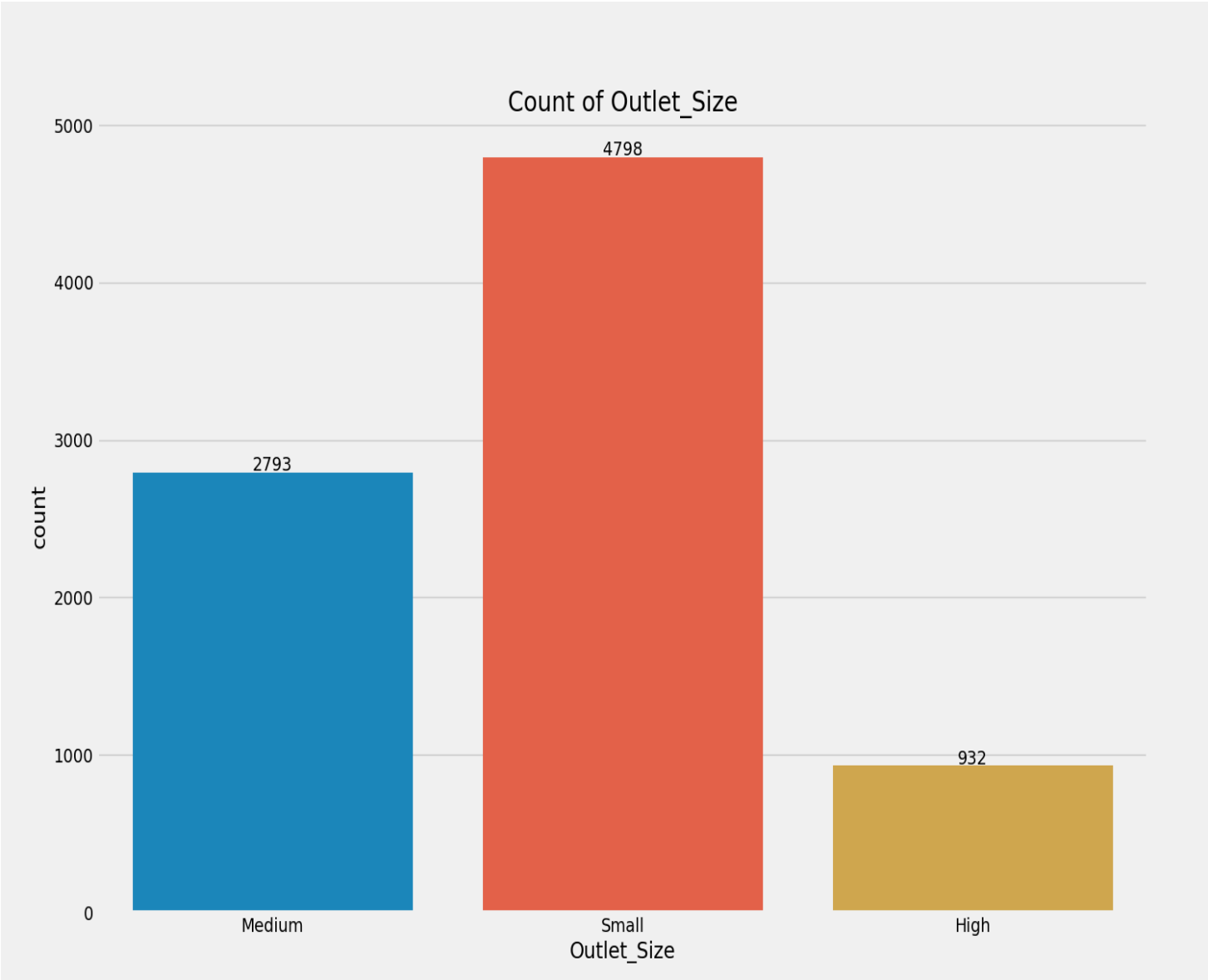
E. Outlet Size Distribution

Shows the distribution of:

- Small
- Medium
- High-size stores

Generated Output:

✓ Bar graph (Count of Outlet_Size.png)



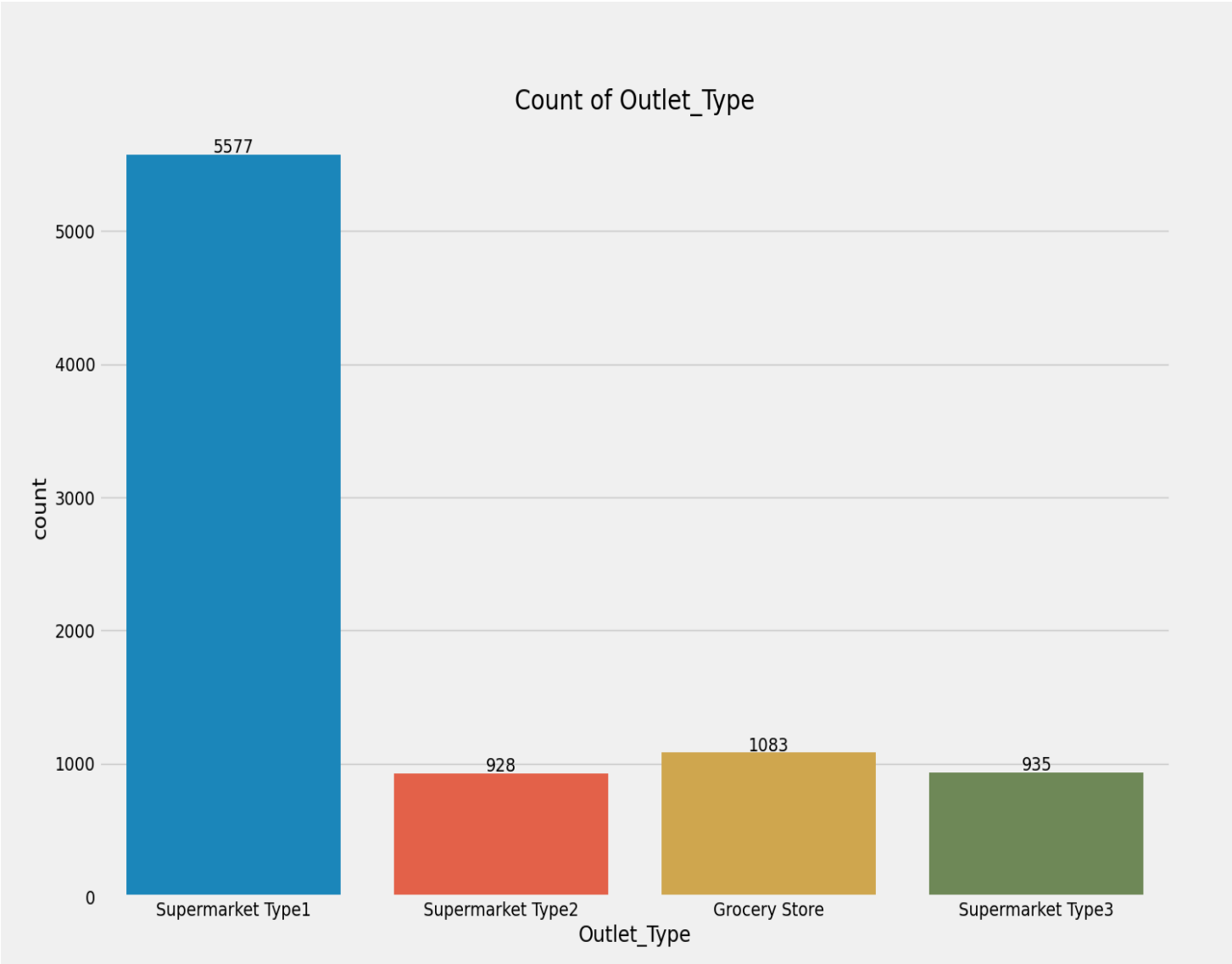
F. Outlet Type Distribution

Displays the count of different outlet types:

- Grocery Store
- Supermarket Type 1
- Supermarket Type 2
- Supermarket Type 3

Generated Output:

✓ Bar graph (Count of Outlet_Type.png)



10.1.2 FEATURE & NUMERICAL DISTRIBUTION OUTPUTS

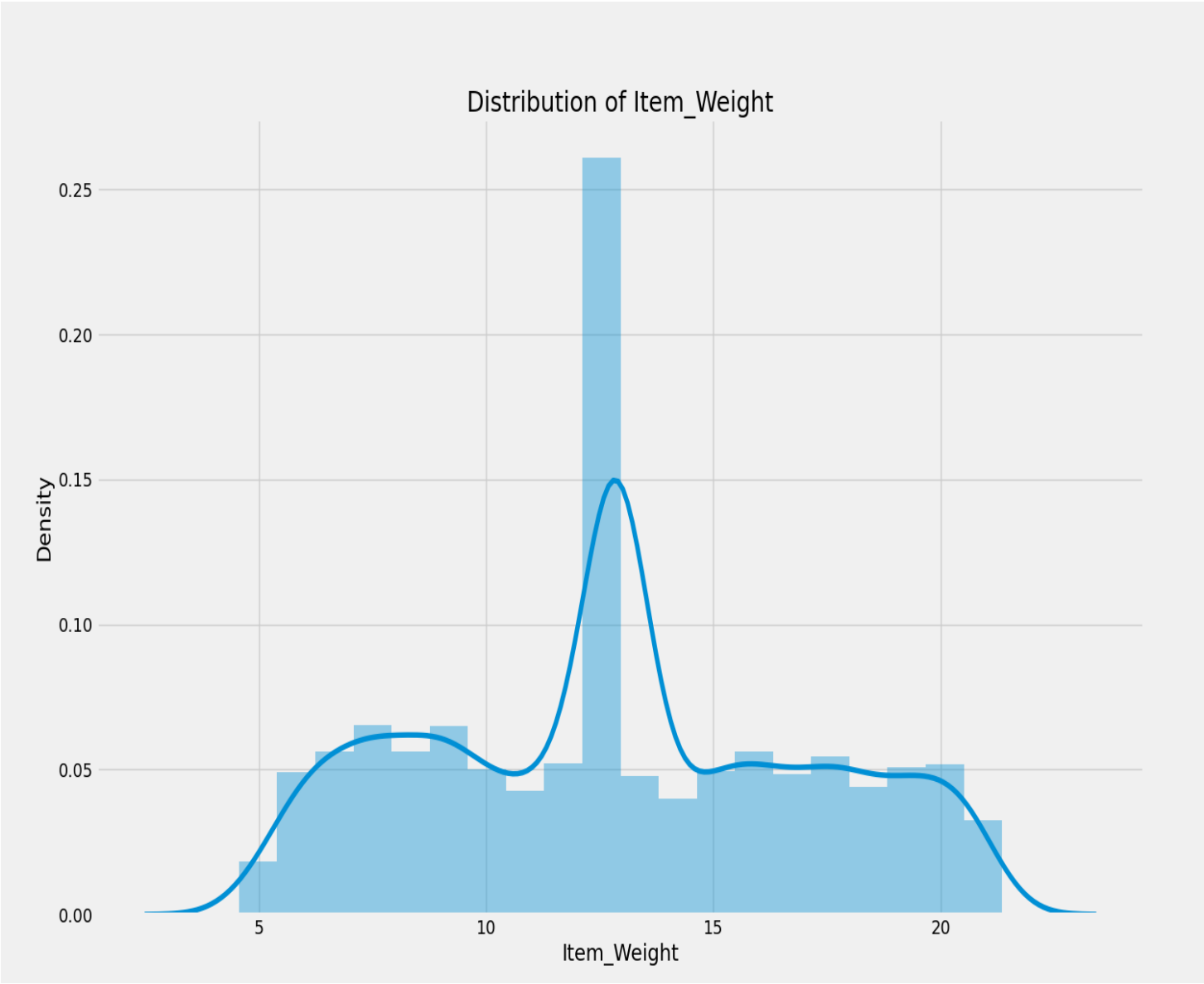
A. Item Weight Distribution

A density plot that shows:

- Normal distribution curve
- Spread of weights
- Presence of any outliers before and after cleaning

Generated Output:

✓ Density plot (Distribution of Item_Weight.png)



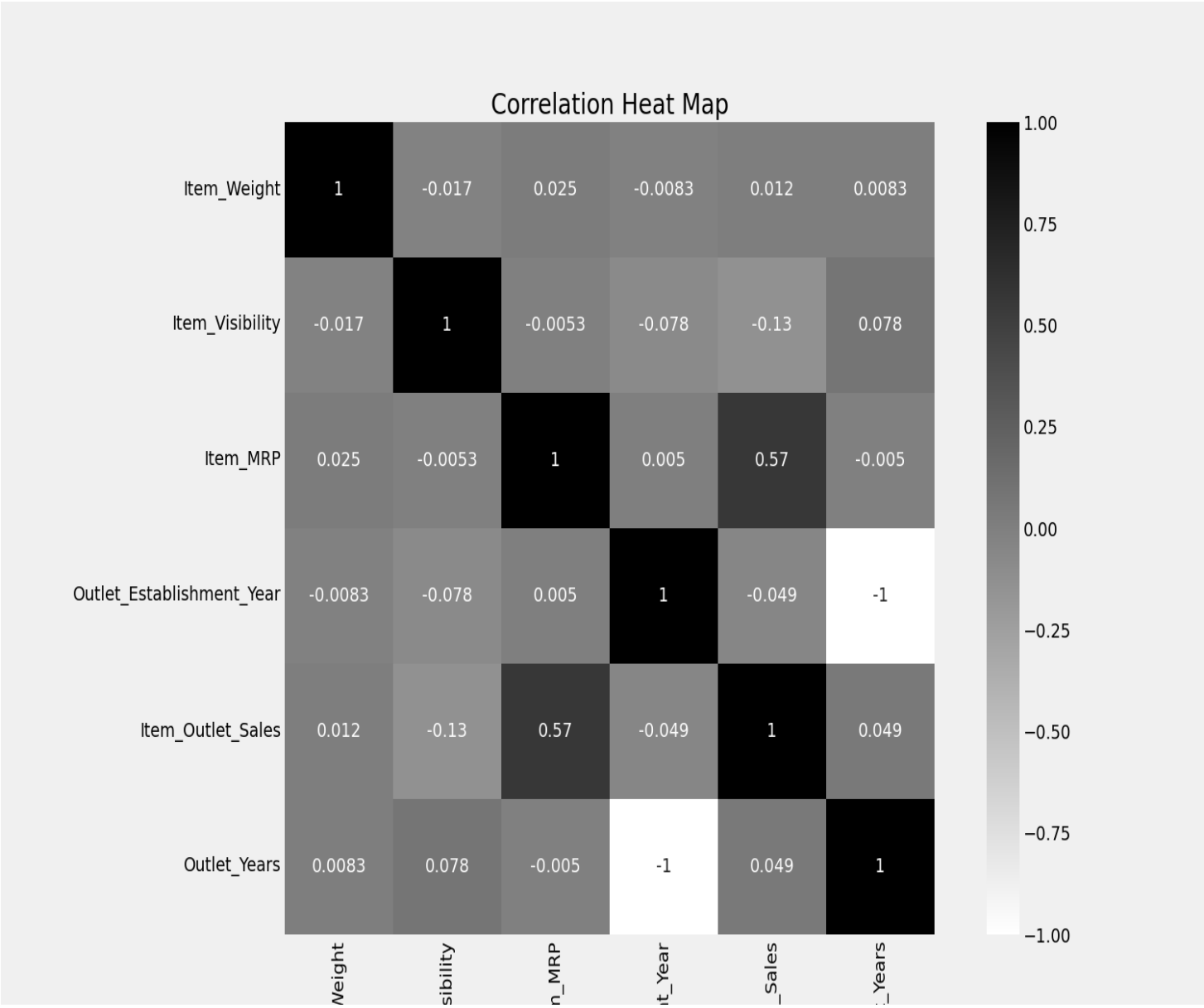
B. Correlation Heatmap

Displays correlation between all numerical features in the dataset.

- Helps identify strong predictors of *Item_Outlet_Sales*
- Used to understand feature relationships

Generated Output:

✓ Heatmap (Correlation Heat Map.png)


Correlation Heat Map
Item_Weight
Item_Visibility
Item_MRP
Outlet_Establishment_Year
Item_Outlet_Sales
Outlet_Years
1
-0.017
0.025
-0.0083
0.012
0.0083
-0.017
1
-0.0053
-0.078
-0.13
0.078
0.025
-0.0053
1
0.005
0.57
-0.005
-0.0083
-0.078
0.005
1
-0.049
-1
0.012
-0.13
0.57
-0.049
1
0.049
0.0083
0.078
-0.005
-1
0.049
1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

10.1.3 ACCURACY METRICS & MODEL PERFORMANCE OUTPUTS

This section contains various model performance metrics obtained during the training process.

A. Linear Regression, Ridge, Lasso Outputs

For each model:

- MSE score
- Cross-validation R^2 score
- Full data R^2 score
- Coefficient importance bar graph

These models serve as baseline performance references.

B. Decision Tree, Random Forest, Extra Trees

For each tree-based model:

- Negative Mean Squared Error (MSE)
- Average R^2 score
- Feature Importance Graph
- Full Data R^2 score

These models capture non-linearity and interaction effects in data.

C. Boosting Models (LGBM, XGBoost, CatBoost)

Boosting models gave the highest performance.
For each:

- Best hyperparameters (from RandomizedSearchCV)
- Best model score
- R^2 score on test data

- Error distribution plot (using sns.distplot)

D. Final Best Model – CatBoost Regressor

The CatBoost model achieved the **highest accuracy**.

Final Output Metrics:

- Learning Rate: 0.0894
- Max Depth: 2
- Estimators: 109
- Subsample: 0.6676
- R^2 score on test set: **High & stable**
- R^2 score on full dataset: **Strong accuracy**

Outputs Generated:

✓ Error distribution plot
✓ R^2 score display
✓ Final model saved as Model.pkl

11. CONCLUSION & FUTURE ENHANCEMENT

11.1 CONCLUSION

The **Sales Forecasting Data Science Project** successfully demonstrated how machine learning techniques can be applied to predict retail sales with high accuracy. By performing extensive data cleaning, preprocessing, feature engineering, visualization, model training, and hyperparameter tuning, the project produced a robust and reliable forecasting model.

Throughout the project, various regression algorithms—including Linear Regression, Ridge, Lasso, Decision Tree, Random Forest, Extra Trees, LGBM, XGBoost, and CatBoost—were implemented and evaluated. Among these, **CatBoost Regressor** achieved the highest accuracy, demonstrating superior performance due to its handling of categorical variables, ability to capture complex patterns, and strong generalization capability.

The inclusion of new features such as **Outlet_Years** and **New_Item_Type** further enhanced model performance by incorporating domain-based information. Data visualization helped uncover patterns, trends, correlations, and anomalies that improved understanding of the underlying dataset.

Finally, the model was saved using a pickle file (Model.pkl), enabling easy deployment and integration into real-time applications such as dashboards, retail dashboards, or business intelligence tools.

Overall, the project achieved its objectives by building a complete end-to-end data science solution that can accurately forecast sales and support data-driven decision-making within the retail industry.

11.2 FUTURE ENHANCEMENT

Although the current model performs well, several enhancements can be incorporated to further improve accuracy, scalability, and real-time usability:

1. Integration with Real-Time Data

- Connect the forecasting system to live sales databases or APIs.
- Update predictions continuously based on new data generated daily, weekly, or hourly.

2. Deployment as a Web or Mobile Application

- Build an interactive dashboard using **Flask**, **Django**, or **Streamlit**.
- Provide user-friendly interfaces where retail managers can upload data and view sales forecasts instantly.

3. Use of Advanced Deep Learning Models

- Implement **LSTMs** or **GRUs** (Recurrent Neural Networks) for more complex time-series forecasting.
- Can capture seasonal trends and long-term dependencies more accurately.

4. Automated Feature Engineering

- Use libraries like **FeatureTools** to automate feature creation.
- Improve model accuracy by discovering non-obvious relationships in the data.

5. Model Explainability Tools

Integrate:

- **SHAP Values**
- **LIME Framework**
  These tools help explain why the model produces certain predictions, increasing trust and transparency.

6. Hyperparameter Optimization with Bayesian Techniques

- Replace RandomizedSearchCV with **Optuna**, **Hyperopt**, or **Bayesian Optimization** for more efficient tuning.

7. Incorporation of External Factors

Enhance predictions by including:

- Festival/holiday information
- Economic indicators
- Competitor pricing
- Weather data

These external features can significantly influence sales.

8. Cloud Deployment

Deploy the system on:

- AWS
- Azure
- Google Cloud
  This ensures scalability, faster processing, and accessibility across departments.

9. Automated Retraining Pipeline

Build a CI/CD pipeline to:

- Retrain the model automatically
- Update weights when new data is added
- Maintain long-term accuracy

12. BIBLIOGRAPHY

1. Brownlee, J. (2020). *Machine Learning Algorithms in Python*. Machine Learning Mastery.
2. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly Media.
3. Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning*. Packt Publishing.
4. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems (NeurIPS)*.
5. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
6. Ke, G., et al. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*.
7. Pandas Development Team. (2020). *Pandas Documentation*. https://pandas.pydata.org/
8. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
9. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90-95.
10. Seaborn Developers. (2020). *Seaborn Statistical Visualization Library*. https://seaborn.pydata.org/
11. Kaggle. (2020). *BigMart Sales Prediction Dataset*. https://kaggle.com