Sales Forecasting Project

Technical Report

Ayman Mushtaq Ahmad, Amisha Tiwari, Ronhit Neema
Khoury College of Computer Science

December 9, 2024

Contents

1 Introduction

Sales forecasting is a critical component for businesses aiming to optimize inventory, manage supply chains efficiently, and enhance overall profitability. Accurate sales predictions enable organizations to make informed decisions regarding stock levels, marketing strategies, and resource allocation. This project focuses on developing a robust sales forecasting model using various machine learning techniques to predict future sales based on historical data.

The primary objectives of this project include:

- Conducting comprehensive data preprocessing and exploratory data analysis (EDA).
- Engineering relevant features to enhance model performance.
- Implementing and evaluating multiple machine learning models.
- Developing an interactive web application for data visualization and model interaction.

2 Methodology

The methodology section details the step-by-step approach undertaken to achieve the project's objectives. It encompasses data loading, exploration, feature engineering, model implementation, and evaluation.

2.1 Dataset Overview

The Project uses a publicly available dataset from Kaggle containing 8523 rows and 12 columns. Key Features include:

- **Target Variable**: Item_Outlet_Sales
- **Predictors**: Item_Type , Item_MRP etc.

2.2 Dataset Analysis and EDA

The dataset, in CSV format, was taken from kaggle and is loaded using Pandas, a powerful data manipulation library in Python. Initial exploration aimed to understand the dataset's structure, including the identification of missing values, duplicate records, and the distribution of various features.

- **Handling Missing and Duplicate Values:** Missing values were identified and addressed using appropriate imputation techniques. Duplicate records were detected and removed to ensure data integrity.
- **Summary Statistics:** Generated for numerical features to gain insights into their distribution, central tendency, and variability.
- **Categorical Feature Identification:** Determined categorical variables requiring encoding for model compatibility.

2.3 Feature Engineering

Feature engineering plays a pivotal role in enhancing model performance by creating meaningful variables from raw data.

- **Encoding Categorical Variables:** Utilized `LabelEncoder` to convert categorical features into numerical format.
- **Feature Scaling:** Applied `StandardScaler` to standardize features, ensuring that models sensitive to feature scaling, such as Linear Regression, perform optimally.
- **Additional Features:** Created new features such as *Product_Category* and *Outlet_Years* to capture more nuanced patterns in the data.

2.4 Model Implementation

Multiple machine learning models were implemented to capture different aspects of the data and improve forecasting accuracy.

- **Linear Regression:** Served as a baseline model to establish initial performance metrics.
- **Regularized Linear Models: Lasso** and **Ridge Regression** were employed to mitigate multicollinearity and enhance model generalizability.
- **Decision Tree Regressor:** Captures non-linear relationships within the data.
- **Ensemble Methods:**
  - **Random Forest Regressor:** An ensemble of decision trees to improve prediction accuracy and control overfitting.
  - **Extra Trees Regressor:** Similar to Random Forest but with more randomness in tree construction, focusing on feature importance.
- **Advanced Models:** Incorporating **CatBoost**, **LightGBM**, and **XGBoost** for enhanced performance.
- **Time-Series Model: ARIMA** model implemented for datasets with temporal dependencies.

2.5 Model Training and Hyperparameter Tuning

Models were trained using a train-test split of 70-30, ensuring a representative distribution of data in both sets. For models supporting hyperparameter tuning, `RandomizedSearchCV` was employed to identify optimal parameter configurations.

2.6 Evaluation

Model performance was assessed using several metrics to ensure a comprehensive evaluation:

- **R^2 Score:** Measures the proportion of variance explained by the model.
- **Root Mean Squared Error (RMSE):** Evaluates the average magnitude of prediction errors.
- **Mean Absolute Error (MAE):** Assesses the average absolute differences between predicted and actual values.
- **Mean Absolute Percentage Error (MAPE):** Provides error as a percentage, facilitating comparison across different scales.
- **Median Absolute Error (MedAE):** Offers robustness against outliers by considering the median of absolute errors.

3 Results

The performance of each implemented model was evaluated based on the aforementioned metrics. The results are summarized in Table 1.

Table 1: Model Performance Metrics

Model	R^2	RMSE	MAE	MAPE (%)
Linear Regression	0.573	1085	824	1.05
Lasso Regression	0.572	1086	823	1.05
Decision Tree	0.16	1522	1076	0.7
LightGBM	0.61	1045	756	0.61
CatBoost	0.60	1045	757	0.62

3.1 Model Comparison

LightGBM Regressor outperformed all other models, achieving the highest R^2 score of 0.6044. This indicates that the model explains 60% of the variance in sales, showcasing its effectiveness in capturing complex data relationships.

Best Performing Model: LGBM Regressor ⭐

Summary of Results

Model	R^2 (Test)	RMSE (Test)	MAE (Test)	MAPE (Test)	CV R^2 Mean	CV RMSE Mean	MedAE (Tes
Linear Regression	0.5730	1085.9426	824.0037	1.0550	0.5645	1096.2596	622.6179
Ridge Regression	0.5730	1085.9265	823.7305	1.0547	0.5653	1095.1819	624.4310
Lasso Regression	0.5724	1086.7343	823.9575	1.0536	0.5648	1095.7961	624.3222
Decision Tree Regressor	0.1605	1522.6045	1076.9090	0.7213	0.1913	1485.1418	719.7298
Random Forest Regressor	0.5954	1057.0015	759.2418	0.5702	0.5992	1051.1219	510.7392
Extra Trees Regressor	0.5359	1132.1498	806.7410	0.5920	0.5382	1128.9823	558.0403
LGBM Regressor	0.6044	1045.2286	756.1558	0.6109	0.6059	1042.9015	508.2648
XGB Regressor	0.6023	1048.0301	761.2413	0.6341	0.6040	1045.4111	505.1890
CatBoost Regressor	0.6044	1045.2390	757.6389	0.6221	0.6053	1043.7421	509.1777

Figure 1: Model Performance Comparison

4 Discussion

The project incorporates interactive Dash visualizations to compare models on key metrics, enabling an intuitive understanding of their performance. The model comparison bar chart clearly highlights the LightGBM Regressor as the best-performing model, with superior R^2, RMSE, and MAE scores compared to others. Additional visualizations, such as Predicted vs. Actual plots, demonstrate the alignment of predictions with actual sales values, showcasing the accuracy of the models. The residual distribution plot reveals the error spread, emphasizing minimal bias in the predictions for the LGBM Regressor. These visual tools not only validate the performance metrics but also enhance interpretability, aiding in model refinement and decision-making processes.

5 Conclusion

This project has successfully combined data mining techniques with an intuitive application to tackle the problem of retail sales forecasting. The LightGBM Regressor stood out as the most accurate model, with metrics and visualizations demonstrating its effectiveness. The implementation of the dashboard application marks a key achievement, as it simplifies the process of data analysis and forecasting for users. By integrating data upload, EDA, modeling, and visualization into a single interface, the dashboard makes advanced analytics accessible to both technical and non-technical users. The project has laid a strong foundation for future improvements, with a clear path to enhance functionality, scalability, and accuracy. It represents a practical and impactful solution for sales forecasting challenges.

6 Future Work

This study lays a solid foundation for sales forecasting using machine learning models. However, there are several avenues for future work:

- **Integration with Real-Time Time Data:** Incorporating real-time sales data with time features to enable dynamic time series based predictive analysis.

- **Exploration of Additional Features and dealing with complex datasets:** Including more granular features such as promotional activities, economic indicators, and seasonal trends to enhance model accuracy in more complex datasets.
- **Advanced Hyperparameter Tuning:** Utilizing more sophisticated hyperparameter optimization techniques like Bayesian Optimization to further improve model performance.
- **Scalability and Performance Optimization:** Ensuring the model scales efficiently with larger datasets and optimizing computational performance for faster predictions.

7 Application User Interface and Result visualizations

We have come up with a modular Web Application with modular design. The Application provides User friendly Upload,EDA, Modeling and visualization features for Users.

7.1 HomePage

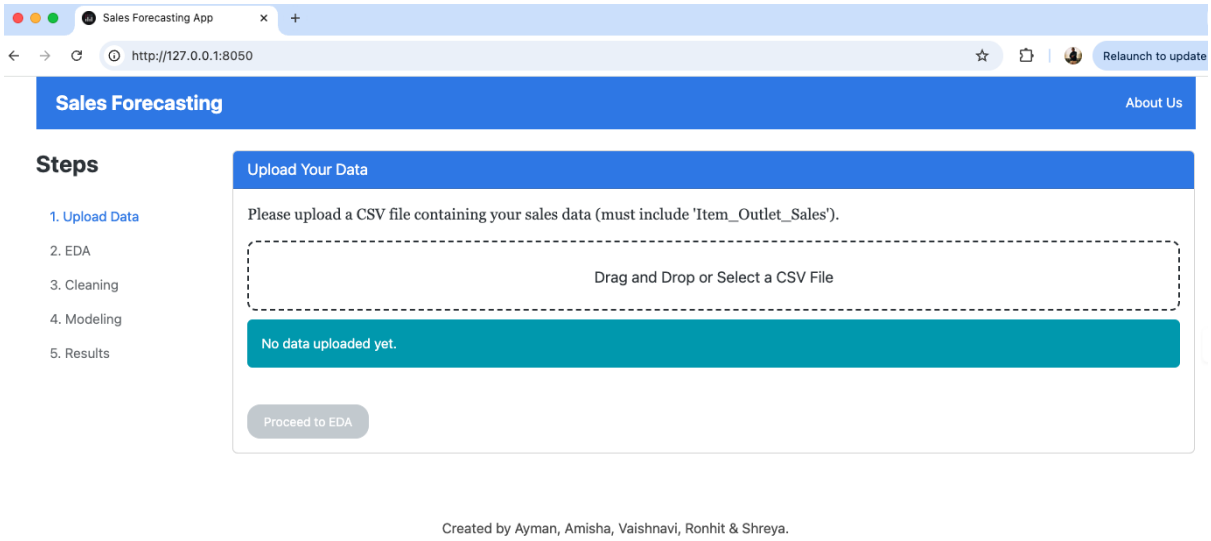

Figure 2: Application Homepage

7.2 EDA

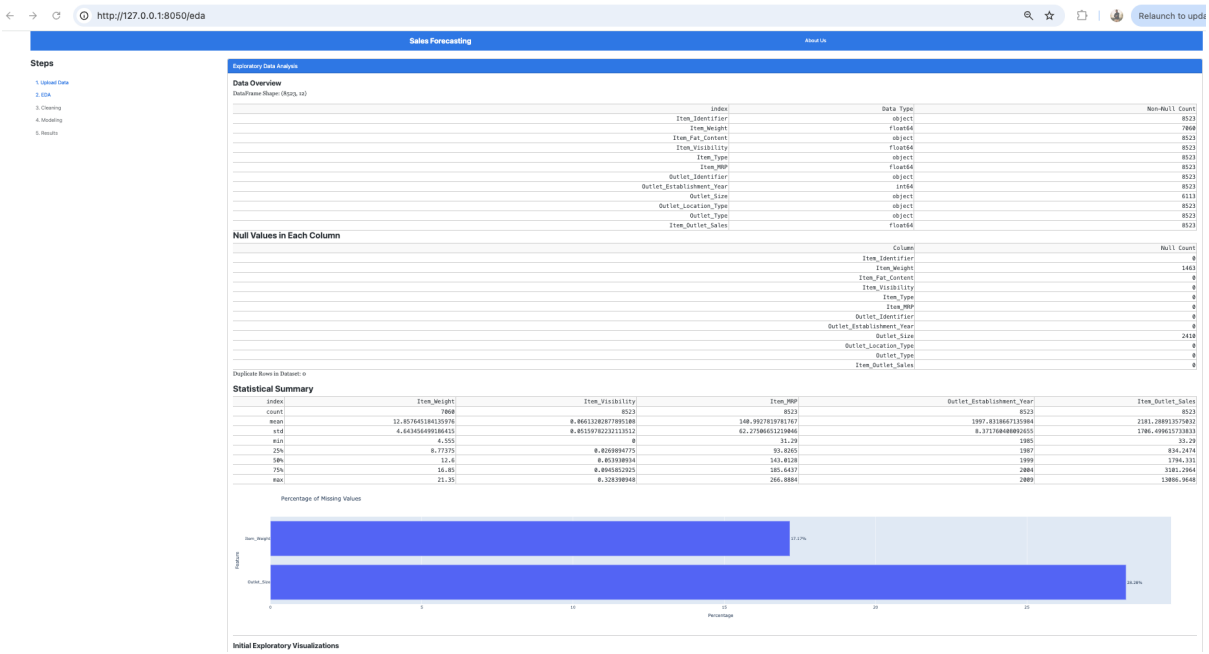

Figure 3: Application EDA layout

7.3 Clean Data Visualization

Figure 4: Application Data Clean Visualization

Figure 5: Application Advanced Data Clean Visualization

7.4 Model comparison visualizations

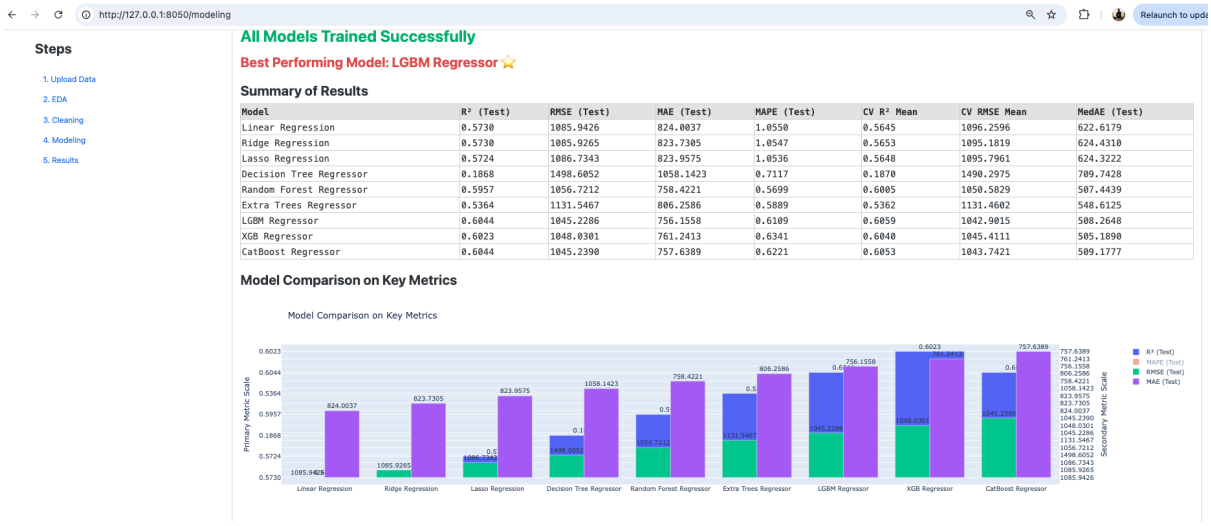

Figure 6: Application Model Visualization

Figure 7: Application Model score Visualization

8 References

References

[1] Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.

[2] Brownlee, J. (2020). *Machine Learning Mastery*. Machine Learning Mastery.

[3] Plotly Technologies Inc. (2019). *Dash: A Productive Python Framework for Building Web Analytics Applications*. `https://dash.plotly.com/`

[4] Grinberg, M. (2018). *Flask Web Development*. O'Reilly Media, Inc.

[5] Spot Intelligence. (2024). *Regression Metrics for Machine Learning*. `https://spotintelligence.com/2024/03/27/regression-metrics-for-machine-learning/`

[6] Airbyte. (2024). *Data Preprocessing*. `https://airbyte.com/data-engineering-resources/data-preprocessing`

[7] Comet. (2024). *Different Plots Used in Exploratory Data Analysis (EDA)*. `https://www.comet.com/site/blog/different-plots-used-in-exploratory-data-analysis-eda/`

[8] Aseaf Aldean. (2024). *Regression Model Performance: Key Metrics and Evaluation Techniques*. `https://medium.com/@aseafaldean/regression-model-performance-key-metrics-and-evaluation-techniques-db40e63f5fc1`

A Appendix A: Code

The project incorporates several Python scripts to manage data processing, model training, and the web application interface. Below are detailed explanations of each code file,

outlining their functions and providing illustrative code snippets to demonstrate their roles.

A.1 App.py

Description: This script initializes and runs the Dash web application, handling user interactions and integrating data upload, EDA, data cleaning, modeling, and result visualization.

Key Functions:

- **switch_page_display**: Manages the display of different pages based on the URL.
- **update_nav_links**: Enables or disables navigation links based on the completion of previous steps.
- **handle_file_upload**: Handles the uploading of CSV files and validates the data.
- **show_eda**: Displays the Exploratory Data Analysis content.
- **enable_proceed_to_cleaning**: Enables the button to proceed to data cleaning after EDA.
- **mark_eda_done**: Marks the completion of EDA.
- **perform_data_cleaning_and_show_visuals**: Performs data cleaning and displays post-cleaning visualizations.
- **store_cleaned_data**: Stores the cleaned data for further processing.
- **enable_modeling**: Enables the button to proceed to modeling after data cleaning.
- **mark_cleaning_done**: Marks the completion of data cleaning.
- **enable_train_button**: Enables the button to train all models after cleaning.
- **train_all_models**: Trains all machine learning models and identifies the best-performing model.
- **show_results**: Displays the results of the model training.
- **download_predictions**: Allows users to download predictions from the best model.
- **download_summary**: Allows users to download summary metrics of all models.
- **enable_proceed_to_results_button**: Enables the button to proceed to the results page after modeling.
- **navigate_steps**: Handles navigation between different steps of the application.
- **toggle_about_modal**: Toggles the visibility of the "About Us" modal.

Illustrative Code Snippets:

```python
@app.callback(
    [Output('original-data-store', 'data'),
     Output('upload-status', 'children'),
     Output('proceed-to-eda', 'disabled')],
    [Input('upload-data', 'contents')],
    [State('upload-data', 'filename')]
)
def handle_file_upload(contents, filename):
    if contents is not None:
        content_type, content_string = contents.split(',')
        decoded = base64.b64decode(content_string)
        try:
            if 'csv' in filename.lower():
                df = pd.read_csv(io.StringIO(decoded.decode('utf-8')))
                if 'Item_Outlet_Sales' not in df.columns:
                    return None, dbc.Alert("The CSV must contain '
    Item_Outlet_Sales' column.", color="danger"), True
                has_time = 'Time' in df.columns
                global models
                models = get_models(has_time=has_time)
                status = dbc.Alert(
                    "Data uploaded successfully! Detected " +
                    ("time-series data." if has_time else "standard
    data.") +
                    " Click 'Proceed to EDA' to continue.",
                    color="success"
                )
                return df.to_dict('records'), status, False
            else:
                return None, dbc.Alert("Unsupported file format. Please
     upload a CSV file.", color="danger"), True
        except Exception as e:
            return None, dbc.Alert(f"Error processing the file: {e}",
    color="danger"), True
    else:
        return None, dbc.Alert("No data uploaded yet.", color="info"),
    True
```

Listing 1: app.py: Handling File Upload

This function manages the uploading of CSV files, ensuring the presence of necessary columns and detecting if the data is time-series.

A.2 Modeling.py

Description: Contains functions to define, train, and evaluate various machine learning models used for sales forecasting.

Key Functions:

- **get_models**: Returns a dictionary of model names mapped to their instances, including an ARIMA model for time-series data if applicable.
- **parse_distribution**: Parses hyperparameter distributions from the configuration file.
- **get_param_distributions**: Retrieves hyperparameter distributions for a given model from the configuration file.

- **generate_model_report**: Generates cross-validation metrics and feature importance plots for a given model.
- **generate_additional_plots_and_metrics**: Generates residuals plots, partial dependence plots, and SHAP values for tree-based models.
- **train_selected_model**: Trains the selected model, performs hyperparameter tuning, and evaluates its performance. Handles both time-series and non-time-series models.

Illustrative Code Snippets:

```python
def get_models(has_time=False):
    """
    Returns a dictionary of model name to model instance.
    If has_time is True, an 'ARIMA' entry will also be included for time series.
    """
    models = {
        'Linear Regression': LinearRegression(),
        'Ridge Regression': Ridge(),
        'Lasso Regression': Lasso(),
        'Decision Tree Regressor': DecisionTreeRegressor(),
        'Random Forest Regressor': RandomForestRegressor(),
        'Extra Trees Regressor': ExtraTreesRegressor(),
        'LGBM Regressor': LGBMRegressor(),
        'XGB Regressor': XGBRegressor(),
        'CatBoost Regressor': CatBoostRegressor(verbose=0)
    }

    if has_time:
        # ARIMA entry - handled differently in train_selected_model
        models['ARIMA'] = 'ARIMA_PLACEHOLDER'

    return models
```

Listing 2: modeling.py: Defining Models

This function initializes various regression models, adding ARIMA if the dataset contains time-series data.

A.3 Data_processing.py

Description: Responsible for all data preprocessing tasks, including EDA, data cleaning, feature engineering, and preparing data for modeling.

Key Functions:

- **initial_eda**: Performs initial exploratory data analysis by generating data summaries and basic visualizations.
- **generate_pre_cleaning_plots**: Creates exploratory visualizations before data cleaning.
- **generate_advanced_pre_cleaning_plots**: Generates advanced exploratory plots, such as correlation heatmaps and scatter matrices.

- **perform_data_cleaning**: Cleans the dataset by handling missing values, standardizing categorical variables, creating additional features, and capping outliers.
- **generate_post_cleaning_plots**: Generates visualizations after data cleaning to assess the impact of the cleaning process.
- **generate_advanced_post_cleaning_plots**: Generates advanced plots to visualize differences between original and cleaned datasets.
- **preprocess_data**: Preprocesses the cleaned data by encoding categorical variables, one-hot encoding, and splitting the data into training and testing sets.

Illustrative Code Snippets:

```
def perform_data_cleaning(sales_data, log_transform_target=False, outlier_cap=True):
    """Cleans the dataset by handling missing values, normalizing 'Item_Fat_Content',
    adding new features, and capping outliers in 'Item_Outlet_Sales'.
    """

    # Fill missing values for 'Item_Weight' using item-specific averages
    Item_Weight_Mean = sales_data.pivot_table(values='Item_Weight', index='Item_Identifier', aggfunc='mean')
    item_weight_map = Item_Weight_Mean['Item_Weight'].to_dict()
    overall_mean_weight = sales_data["Item_Weight"].mean()
    sales_data["Item_Weight"] = sales_data["Item_Weight"].fillna(sales_data["Item_Identifier"].map(item_weight_map)).fillna(overall_mean_weight)

    # Fill missing 'Outlet_Size' using most frequent size per 'Outlet_Type'
    outlet_size_mode = sales_data.pivot_table(values='Outlet_Size', columns='Outlet_Type', aggfunc=(lambda x: x.mode()[0] if len(x.mode()) > 0 else np.nan))
    missing_bool = sales_data['Outlet_Size'].isna()
    sales_data.loc[missing_bool, 'Outlet_Size'] = sales_data.loc[missing_bool, 'Outlet_Type'].apply(lambda x: outlet_size_mode[x])

    # Replace zeros in 'Item_Visibility' with mean value
    item_visibility_overall_mean = sales_data["Item_Visibility"].replace(0, np.nan).mean()
    sales_data["Item_Visibility"] = sales_data["Item_Visibility"].replace(0, item_visibility_overall_mean)

    # Standardize 'Item_Fat_Content' values
    sales_data["Item_Fat_Content"] = sales_data["Item_Fat_Content"].replace({'LF': 'Low Fat', 'low fat': 'Low Fat', 'reg': 'Regular'})

    # Add 'Product_Category' feature from 'Item_Identifier'
    sales_data["Product_Category"] = sales_data["Item_Identifier"].str[:2]
    sales_data["Product_Category"] = sales_data["Product_Category"].replace({'FD': 'Food Item', 'NC': 'Non Consumable', 'DR': 'Drink'})

    # Update 'Item_Fat_Content' for non-consumable items
```

```
    sales_data.loc[sales_data['Product_Category'] == 'Non Consumable',
    'Item_Fat_Content'] = 'Non Edible'

    # Calculate 'Outlet_Years' as the number of years since outlet
    establishment
    current_year = datetime.datetime.now().year
    sales_data['Outlet_Years'] = current_year - sales_data['
    Outlet_Establishment_Year']

    # Additional Cleaning: Outlier capping for target (99th percentile)
    if 'Item_Outlet_Sales' in sales_data.columns and outlier_cap:
        cap_val = sales_data['Item_Outlet_Sales'].quantile(0.99)
        sales_data['Item_Outlet_Sales'] = np.where(sales_data['
    Item_Outlet_Sales'] > cap_val, cap_val, sales_data['
    Item_Outlet_Sales'])

    # Optional: Log transform target if desired
    if log_transform_target and 'Item_Outlet_Sales' in sales_data.
    columns:
        sales_data['Item_Outlet_Sales'] = np.log1p(sales_data['
    Item_Outlet_Sales'])

    return sales_data
```

Listing 3: data_processing.py: Performing Data Cleaning

This function handles missing values, standardizes categorical variables, engineers new features, and caps outliers to ensure data integrity and improve model performance.

B Appendix B: Configuration File

The configuration file defines hyperparameter tuning spaces for various machine learning models.

```
# Model Hyperparameter Configuration
# This YAML file defines the hyperparameter tuning space for machine
    learning models.
# Each model has its own set of hyperparameters with specific ranges or
    distributions.
# These parameters are used during hyperparameter tuning to find the
    best configuration for each model.
model_parameters:
  Random Forest Regressor:
    max_depth: [5, 10, 15, 20, 25, 30]
    max_features: [null, "sqrt"]
    min_samples_split: [2, 5, 10, 15, 100]
    min_samples_leaf: [1, 2, 5, 10]

  LGBM Regressor:
    n_estimators: { "dist": "randint", "low": 100, "high": 150 }
    learning_rate: { "dist": "uniform", "low": 0.03, "width": 0.3 }
    max_depth: { "dist": "randint", "low": 2, "high": 6 }
    subsample: { "dist": "uniform", "low": 0.6, "width": 0.4 }

  XGB Regressor:
    n_estimators: { "dist": "randint", "low": 100, "high": 150 }
    learning_rate: { "dist": "uniform", "low": 0.03, "width": 0.3 }
```

```
    max_depth: { "dist": "randint", "low": 2, "high": 6 }
    subsample: { "dist": "uniform", "low": 0.6, "width": 0.4 }
    gamma: { "dist": "uniform", "low": 0.0, "width": 0.5 }

  CatBoost Regressor:
    n_estimators: { "dist": "randint", "low": 100, "high": 150 }
    learning_rate: { "dist": "uniform", "low": 0.03, "width": 0.3 }
    depth: { "dist": "randint", "low": 2, "high": 6 }
    subsample: { "dist": "uniform", "low": 0.6, "width": 0.4 }
```

Listing 4: config.yaml

This YAML configuration file specifies the ranges and distributions for hyperparameters of different regression models, facilitating effective hyperparameter tuning during model training.

C Appendix C: Additional Figures