Predict the Item Price

El-Sharqawy Wael El-Sharqawy	23P0360
Ahmed Mostafa Gomaa Atia	23p0375
Abdelrhman Mohammed Mahmoud	23p0370
Nagy Ahmed Nagy	23p0365
Zeinab Tarek Abdelmoniem Mohamed	23P0420
Amira Khalaf Dabash	23P0371

Submitted to:

- Dr. Mariam Nabil
- Eng. Mohammed Essam

ENG – ASU – CAIE

Dec 26, 2024

Table of Contents

Table of Contents 2

Overview 3

Dataset Overview 3

Data Content 3

Data Head 3
Data Count and Data Types 3
Feature Types 4
Ploting Numerical Features 4
Outliers 4
Ploting Categorical Features 5
Numerical with Numerical Bivariate Analysis 6
Correlation Heatmap 6

Data Pre-Processing 7

Filling Missing Values 7
Scaling Numerical Features Using Robust Scaler 7
Encoding Categorical Features Using OneHotEncoder and Fill Missing 7
Handle Outliers With Z-Score Method 7

Feature Engineering 8

Feature Selection Using Lasso 8
Dimensionality Reduction Using PCA 8
Fix X3 Mapping 8
Drop Unuseful Columns 8

Models 9

1. Linear Regression 9
Filling Missing Values 9
Preprocessing Pipelines 9
Outlier Handling Function 10
Model Building 10

2. Random Forest Regression 11
Outlier Handling Function 11
Preprocessing Pipelines 11
Model Building 11

3. Support Vector Regression (SVR) 12
Handling Missing Values: 12
Scaling the Data: 12
Outlier Mitigation: 12
Feature Selection: 12
Dimensionality Reduction: 13
Model Building 13

4. CatBoost 13
Filling Missing Values 13
Handling Categorical Features 14
Scaling 14
Model Building 15

Conclusion 15

Accessing the Source Code 15

Overview

This report presents the development of a machine learning model to predict item prices using a dataset containing various features related to the items. The project focuses on evaluating the impact of different preprocessing techniques on the performance of several machine learning models. Preprocessing steps included handling missing values, feature scaling, outlier detection, and encoding categorical variables. Each model was paired with preprocessing methods tailored to its specific requirements.

We implemented and compared multiple algorithms, including **Linear Regression**, **Random Forests**, **SVR**, and **CatBoost** using metrics such as Mean Absolute Error (**MAE**), Mean Squared Error (**MSE**) and R^2 to evaluate performance.

Dataset Overview

Source: Kaggle Competitetion [CSE281 [24]: Predict the Item Price]
Features: From **'X1'** to **'X11'**
Target: predict the vlaue of column **'Y'**

Data Content

In our data there are 6000 different items and 12 features.

Data Head

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	8.23
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	6.09
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	7.65
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	6.60
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	6.90

Data Count and Data Types

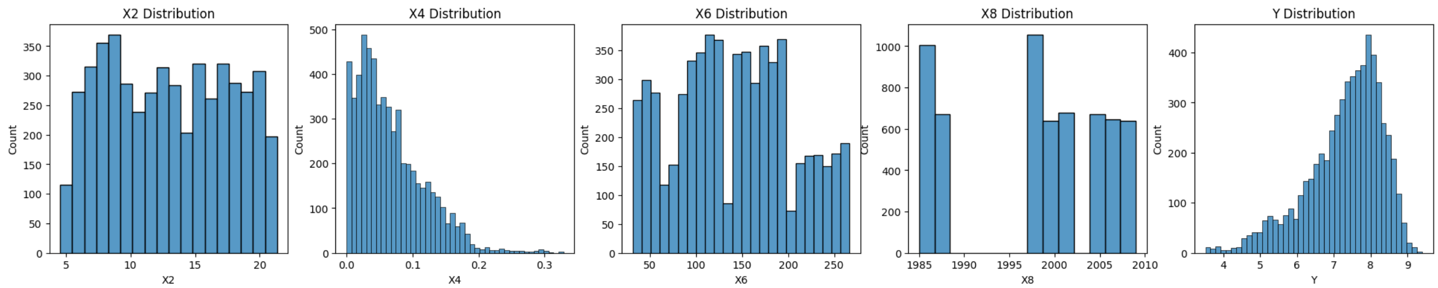
Column	Non-Null Count	Dtype
X1	6000	**object**
X2	**4994**	**float64**
X3	6000	**object**
X4	6000	**float64**
X5	6000	**object**
X6	6000	**float64**
X7	6000	**object**
X8	6000	**int64**
X9	**4289**	**object**
X10	6000	**object**
X11	6000	**object**
Y (Target variable)	6000	**float64**

Feature Types

1. 'X1', 'X3', 'X5', 'X7', 'X9', 'X10', 'X11' are categorical features
2. 'X2', 'X4', 'X6', 'X8', 'Y' are numerical features

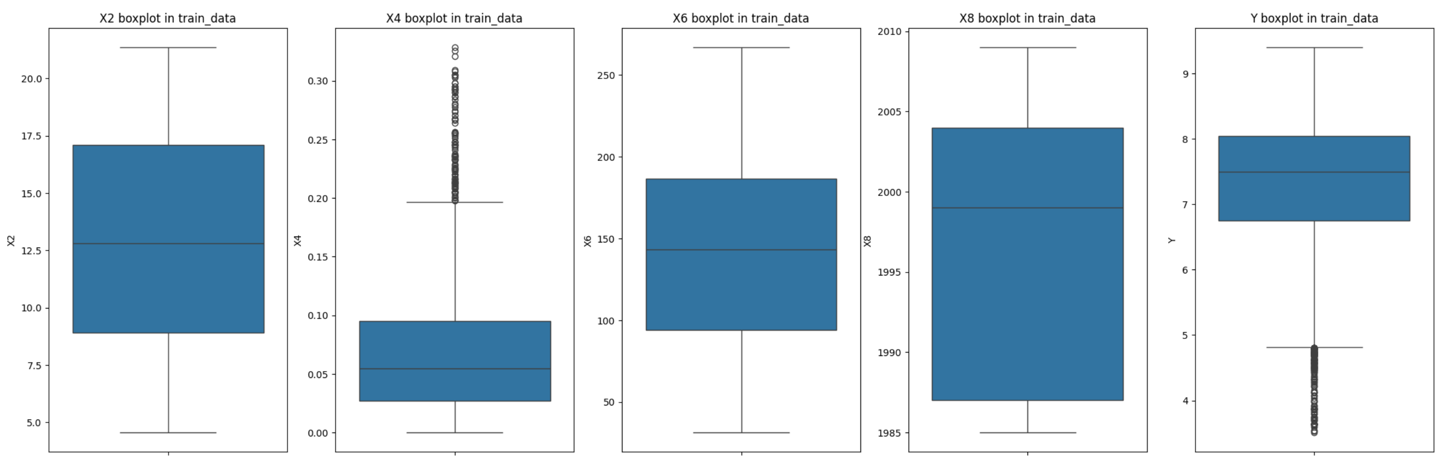
Exploratory Data Analysis

Ploting Numerical Features



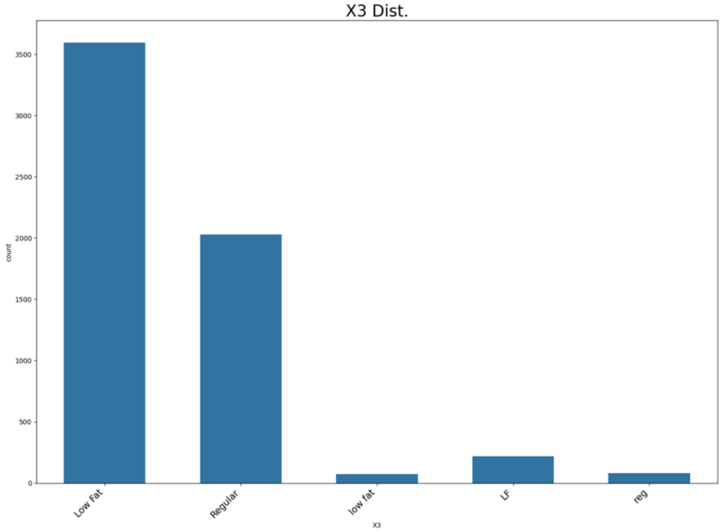
Observations:
- X2: range from 4.55 to 21.35
- X4: is right skewed
- X8: no values between 1990 and 1995
- Y: is smth like a normal distribution but left skewed

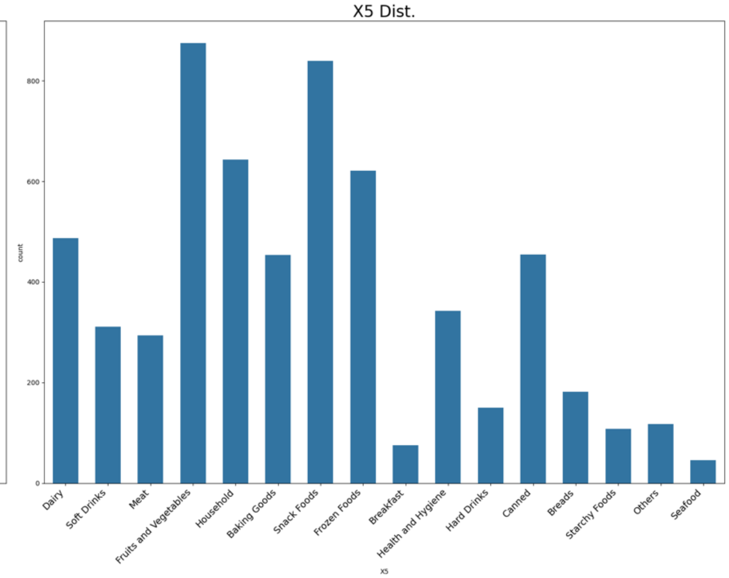
Outliers

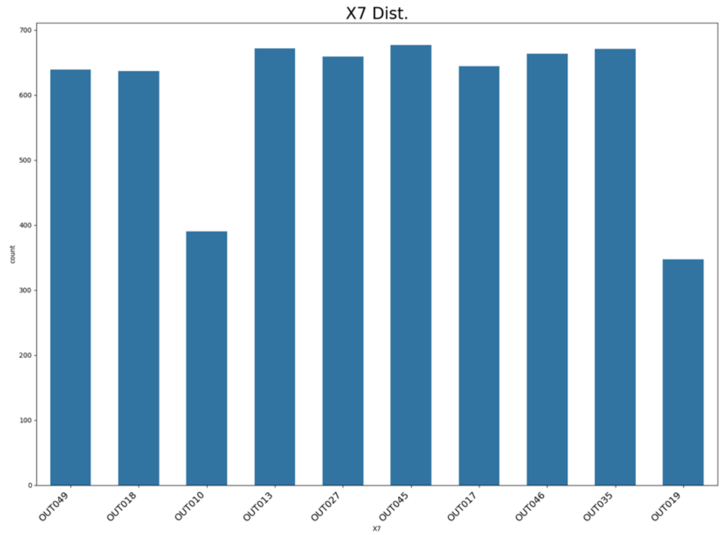


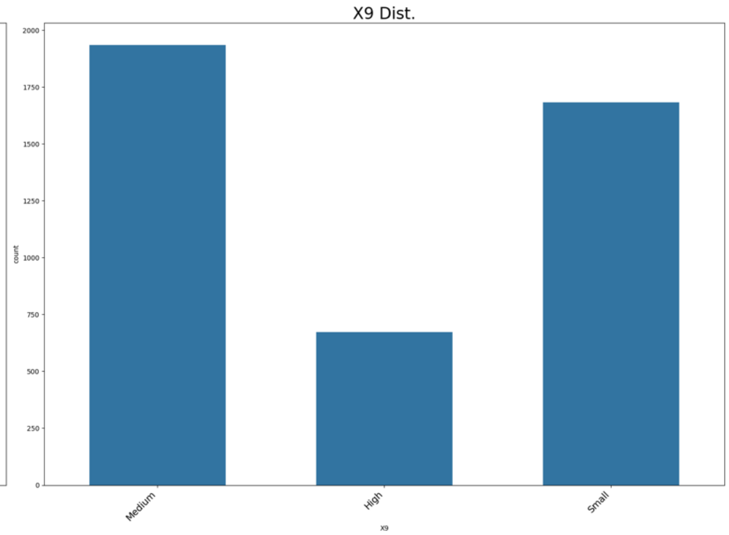
Observations:
- X4 and Y have outliers, the same thing with test data also

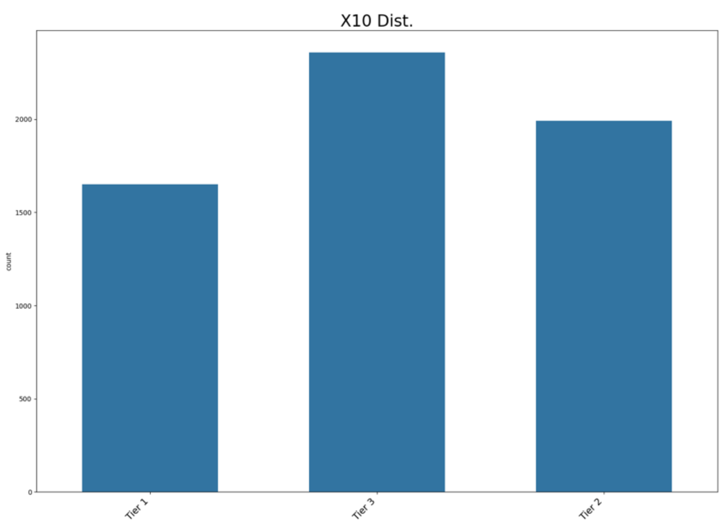
Ploting Categorical Features

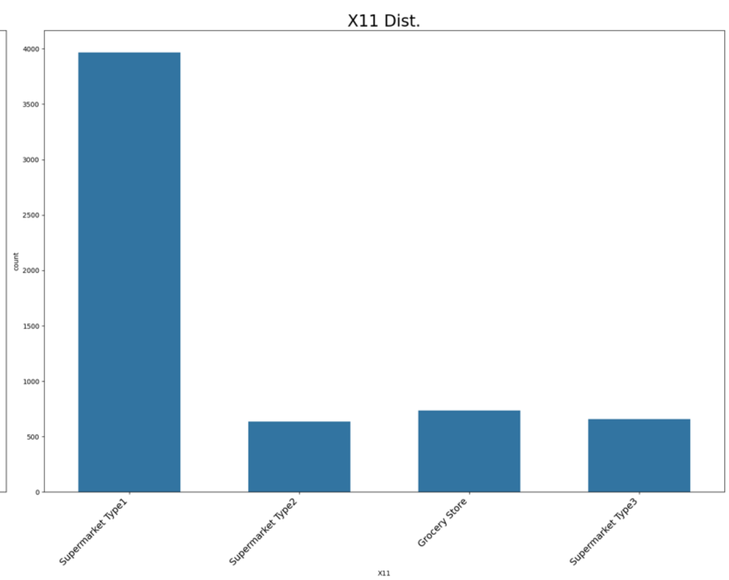








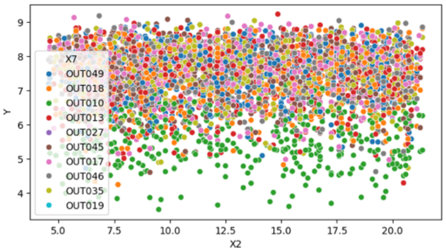


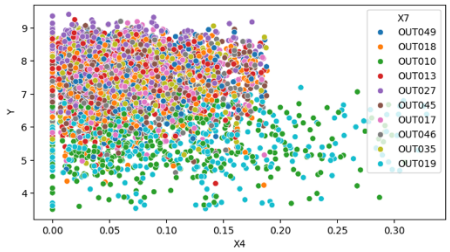


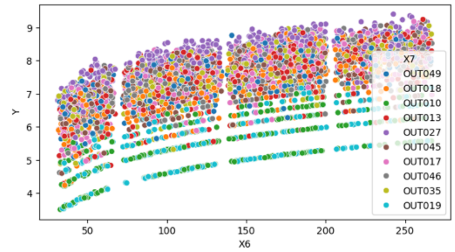
Observations:

- X3 has values need to be mapped, 'LF', 'low fat' and 'Low Fat' are the same thing, 'reg' and 'Regular' are the same thing, maybe we can make it binary feature.
- X5 has 16 features, we can reduce them.
- X10 can be mapped to 1, 2, 3.

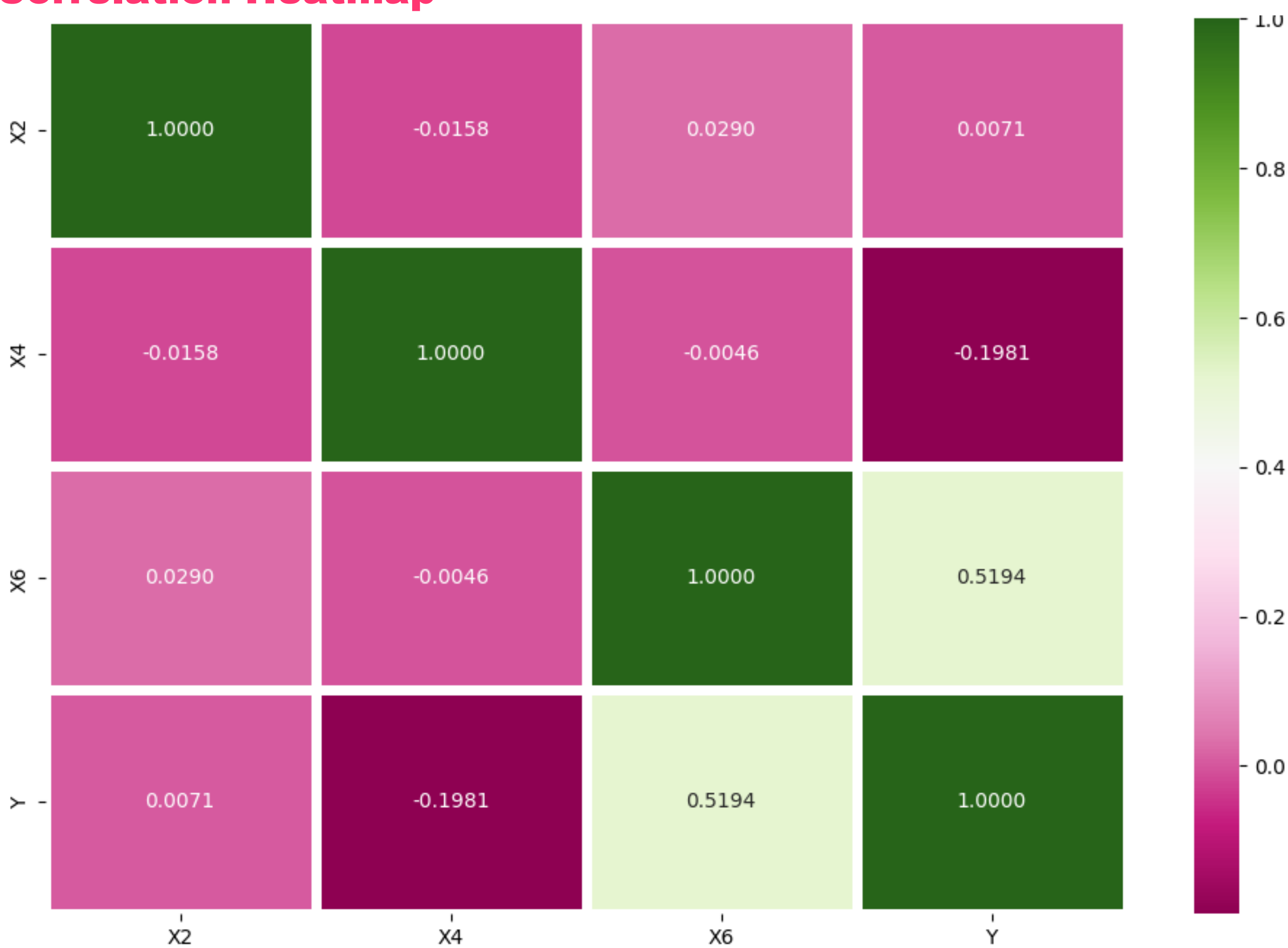
Numerical with Numerical Bivariate Analysis







Correlation Heatmap



Data Pre-Processing

Filling Missing Values

We noticed that for all X1 values, the corresponding X2 value is the same. So, fill the missing X2 using X2 value in another row with the same X1 value.

```
def group_imputation(data):
    for idx, row in data[data['X2'].isnull()].iterrows():
        # Find rows with the same X1 value but non-null X2
        matching_rows = data[(data['X1'] == row['X1'])
                             & (data['X2'].notnull())]

        if not matching_rows.empty:
            # Use the first matching row's X2 value to fill the missing X2
            data.at[idx, 'X2'] = matching_rows.iloc[0]['X2']
    return data
```

Scaling Numerical Features Using Robust Scaler

```
    num_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', RobustScaler())
    ])
```

Encoding Categorical Features Using OneHotEncoder and Fill Missing

```
    cat_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent', fill_value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
    ])
```

Handle Outliers With Z-Score Method

```
z_scores = np.abs(stats.zscore(X_train_scaled))
z_score_threshold = 3
X_train_scaled = np.where(z_scores > z_score_threshold, np.mean(
    X_train_scaled, axis=0), X_train_scaled)
```

Feature Engineering

Feature Selection Using Lasso

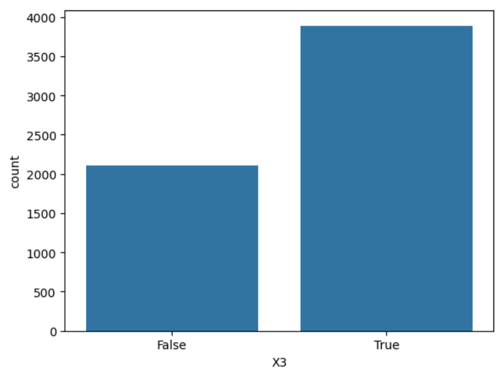
```
lasso = Lasso(alpha=0.01, random_state=42)
lasso.fit(X_train_scaled, y_train_full)
model = SelectFromModel(lasso, prefit=True)
X_train_selected = model.transform(X_train_scaled)
X_test_selected = model.transform(X_test_scaled)
```

Dimensionality Reduction Using PCA

```
pca = PCA(n_components=5, random_state=42)
X_train_pca = pca.fit_transform(X_train_selected)
X_test_pca = pca.transform(X_test_selected)
```

Fix X3 Mapping

```
def fix_X3_mapping(data):
    data['X3'] = data['X3'].map({'Low Fat': 1, 'low fat': 1, 'LF': 1, 'Regular': 0, 'reg': 0}).astype(bool)
    return data
```



Drop Unuseful Columns

After looking at the data, we can see that each value in X7 has a static set of values in X8, X9, X10, X11. So, we can use this information to drop these columns.

```
train_data.drop(['X8', 'X9', 'X10', 'X11'], axis=1, inplace=True)
test_data.drop(['X8', 'X9', 'X10', 'X11'], axis=1, inplace=True)
```

Models

1. Linear Regression

Linear Regression is a fundamental model that establishes a linear relationship between independent variables and the target variable.

Filling Missing Values

- For rows where X2 is missing, the corresponding value is filled using X2 from other rows with the same X1 value.

```
def group_imputation(data):
    for idx, row in data[data['X2'].isnull()].iterrows():
        # Find rows with the same X1 value but non-null X2
        matching_rows = data[(data['X1'] == row['X1'])
                             & (data['X2'].notnull())]

        if not matching_rows.empty:
            # Use the first matching row's X2 value to fill the missing X2
            data.at[idx, 'X2'] = matching_rows.iloc[0]['X2']
    return data
```

Preprocessing Pipelines

Numerical Data:

- Imputation: Missing numerical values are replaced with the median of the column using SimpleImputer.
- Scaling: Standardized using StandardScaler to center the data (mean=0, std=1), ensuring compatibility with the Random Forest model.

Categorical Data:

- Imputation: Missing categorical values are replaced with the most frequent value of the column.
- One-Hot Encoding: Converts categorical variables into a binary matrix using OneHotEncoder.

```
# Preprocessing
numerical_preprocessor = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_preprocessor = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Outlier Handling Function

- The function handle_outliers applies the Interquartile Range (IQR) method to cap or floor outliers in numerical columns

```
def handle_outliers(data, cols):
    for col in cols:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        data[col] = data[col].clip(lower_bound, upper_bound)
    return data
```

Model Building

```
# Linear Regression Model
linear_model = LinearRegression()

# Pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', linear_model)
])

# Training the model
pipeline.fit(X_train, y_train)
```

2. Random Forest Regression

Random Forest is an ensemble learning method that uses multiple decision trees to enhance predictive accuracy.

Outlier Handling Function

- The function handle_outliers applies the Interquartile Range (IQR) method to cap or floor outliers in numerical columns

```
def handle_outliers(data, cols):
    for col in cols:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        data[col] = data[col].clip(lower_bound, upper_bound)
    return data
```

Preprocessing Pipelines

Numerical Data:

- Imputation: Missing numerical values are replaced with the mean of the column using SimpleImputer.
- Scaling: Standardized using StandardScaler to center the data (mean=0, std=1), ensuring compatibility with the Random Forest model.

Categorical Data:

- Imputation: Missing categorical values are replaced with the most frequent value of the column.
- One-Hot Encoding: Converts categorical variables into a binary matrix using OneHotEncoder.

```
# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Model Building

```
model = RandomForestRegressor(n_estimators=300, max_depth=8, min_samples_leaf=7, min_samples_split=9,
bootstrap=True, random_state=42)

pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])
```

3. Support Vector Regression (SVR)

SVR is a versatile regression technique that aims to find a hyperplane with maximum margin for predicting continuous values.

Handling Missing Values:

- Used SimpleImputer with the strategy "mean" to fill missing values in the training data (X_train_full) with the mean of the respective feature.
- Ensured there are no missing (NaN) or infinite (Inf) values in the dataset after imputation.

```
imputer = SimpleImputer(strategy="mean")
X_train_scaled = imputer.fit_transform(X_train_full)
```

Scaling the Data:

- Applied RobustScaler to scale the features while reducing the impact of outliers. The scaler uses the median scaling.

```
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train_scaled)
X_test_scaled = scaler.transform(X_test_full)
```

Outlier Mitigation:

- Calculated z-scores for all scaled features to identify outliers (absolute z-score > 3).
- Replaced outlier values with the mean of the respective feature, mitigating extreme values that could affect model performance.

```
z_score_threshold = 3
for i in range(X_train_scaled.shape[1]):

    outliers = np.where(z_scores[:, i] > z_score_threshold)[0]

    for index in outliers:
        X_train_scaled[index, i] = np.mean(X_train_scaled[:, i])
```

Feature Selection:

- Used a Lasso regression model to perform feature selection by identifying and retaining only the most significant features.
- SelectFromModel was applied to transform the dataset, reducing it to a subset of important features.

```
lasso = Lasso(alpha=0.01, random_state=42)
lasso.fit(X_train_scaled, y_train_full)

model = SelectFromModel(lasso, prefit=True)
X_train_selected = model.transform(X_train_scaled)
X_test_selected = model.transform(X_test_scaled)
```

Dimensionality Reduction:

- Performed Principal Component Analysis (PCA) to reduce the feature space to 5 principal components. This step captures the majority of variance in the data while improving computational efficiency.

```
pca = PCA(n_components=5, random_state=42)
X_train_pca = pca.fit_transform(X_train_selected)
X_test_pca = pca.transform(X_test_selected)
```

Model Building

- Best Params using Grid Search: {'C': 1, 'epsilon': 0.01, 'kernel': 'rbf'}

```
param_grid = {
    'kernel': ['linear', 'rbf', 'poly'],
    'C': [0.1, 1, 10],
    'epsilon': [0.01, 0.1, 1]
}

grid = GridSearchCV(SVR(), param_grid, cv=5,
                    scoring='neg_mean_squared_error', n_jobs=-1)
grid.fit(X_train, y_train)
```

4. CatBoost

CatBoost is a gradient boosting algorithm designed to handle categorical data efficiently.

Filling Missing Values

- For rows where X2 is missing, the corresponding value is filled using X2 from other rows with the same X1 value.

```
def group_imputation(data):
    for idx, row in data[data['X2'].isnull()].iterrows():
        # Find rows with the same X1 value but non-null X2
        matching_rows = data[(data['X1'] == row['X1'])
                             & (data['X2'].notnull())]

        if not matching_rows.empty:
            # Use the first matching row's X2 value to fill the missing X2
            data.at[idx, 'X2'] = matching_rows.iloc[0]['X2']
    return data
```

Handling Categorical Features

- Columns X1, X5, and X7 are specified as categorical features for CatBoost because CatBoost handles categorical features natively, making it more efficient and accurate.

```
for col in ['X1', 'X5', 'X7']:
    test_data[col] = test_data[col].astype('category')
    train_data[col] = train_data[col].astype('category')
```

Drop Unuseful Columns

After looking at the data, we can see that each value in X7 has a static set of values in X8, X9, X10, X11. So, we can use this information to drop these columns.

```
train_data.drop(['X8', 'X9', 'X10', 'X11'], axis=1, inplace=True)
test_data.drop(['X8', 'X9', 'X10', 'X11'], axis=1, inplace=True)
```

Scaling

MaxAbs Scaling:

- Applied to columns X2 and X6:
- Scales values to be between -1 and 1 by dividing by the maximum absolute value.
- Preserves sparsity and is robust to outliers.

Log Transformation:

- Applied to column X4 using np.log1p:
- Reduces the impact of large values while avoiding errors from zeros or negative values.

```
# we can maxAbs with X2, X6
max_abs = MaxAbsScaler()
train_data[['X2', 'X6']] = max_abs.fit_transform(train_data[['X2', 'X6']])
test_data[['X2', 'X6']] = max_abs.transform(test_data[['X2', 'X6']])

# log with X4
train_data['X4'] = np.log1p(train_data['X4'])
test_data['X4'] = np.log1p(test_data['X4'])
```

Model Building

- Best params conclued after running Bayesian Search

```
# Define the CatBoost Regressor model
catboost_model = CatBoostRegressor(
    iterations=1500,
    learning_rate=0.025,
    depth=6,
    l2_leaf_reg=1e-05,
    bagging_temperature=0.762973005798845,
    colsample_bylevel=0.5693046782994058,
    max_bin=100,
    subsample=0.5615832599217679,
    random_state=21,
    cat_features=['X1', 'X5', 'X7', 'X3']
)

# Fit the model
catboost_model.fit(X_train, y_train)
```

Conclusion

Model	MAE (private)	MAE (public)
Linear Regression	0.391	0.416
Random Forest Regression	0.384	0.415
Support Vector Regression	0.372	0.402
CatBoost Regression	0.380	0.403

After evaluating various models, **CatBoost** Regressor and **Support Vector Regressor (SVR)** demonstrated the best performance. **CatBoost** excelled in handling categorical features and captured complex relationships effectively, thanks to its built-in support for categorical data and robust regularization. **SVR**, with carefully tuned hyperparameters, performed competitively, showcasing its strength in modeling non-linear patterns. Both models achieved low MAE, making them reliable for predicting the target variable. While CatBoost is recommended for its simplicity in preprocessing and overall accuracy, SVR remains a strong alternative for scenarios requiring lightweight implementation.

Accessing the Source Code

The complete code implementation for this project, including data preprocessing, feature engineering, model training, and evaluation, is available on GitHub. You can access it through the following link:
Predict-the-Item-Price-Kaggle-Competition

“Models are trained, tuned, and optimized with care. Keep innovating!”

“THIS PROJECT WAS BUILT WITH PASSION AND DEDICATION AT AIN SHAMS UNIVERSITY, EGYPT, WHERE EVERY ALGORITHM AND DATASET REFLECT THE COMMITMENT TO ADVANCING THE FRONTIERS OF MACHINE LEARNING AND KNOWLEDGE.”