BOSTON INSTITUTE OF ANALYTICS

CAPSTONE PROJECT REPORT

Sale Price Prediction for Big Basket Products

SUBMITTED BY

SHALINI DIWAKAR PATHAK

Sale Price Prediction for Big Basket Products

Abstract

The objective of this project is to predict the sale price of products in an Indian grocery supermarket using a machine learning approach. Accurate sale price prediction helps supermarkets like Big Basket to optimise pricing strategies, manage inventory, and enhance customer satisfaction. This project involves data exploration, preprocessing, feature engineering, model selection, training, tuning, and evaluation.

1. Business Problem

1.1 Objective

To predict the sale price of products sold in an Indian grocery supermarket using historical data. The dataset includes various features such as product details, category, sub_category, brand, type, rating, sale_price, and market_price.

1.2 Challenges

- Handling missing values and outliers in the data.
- Encoding categorical variables effectively.
- Selecting and tuning appropriate machine learning models.

1.3 Real World Impact

The accurate prediction of sale prices can significantly impact inventory management, pricing strategies, and overall customer satisfaction, leading to optimised operations and increased revenue for supermarkets.

2. Dataset

2.1 Data Description

The dataset contains the following features:

- Product name
- Category and sub_category information
- Brand information
- Type and rating of products
- Sale price and market price of products

2.2 Data Fields

- product: Unique name for each product.
- category: Category of the product.
- sub_category: Sub-category of the product.
- brand: Brand of the product.
- type: Type of the product.
- rating: Customer rating of the product.
- sale_price: Sale price of the product.
- market_price: Market price of the product.

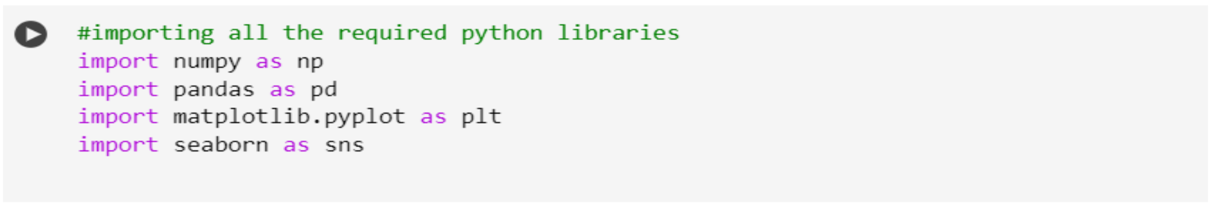
3. Data Exploration & Preprocessing

3.1 Data Exploration

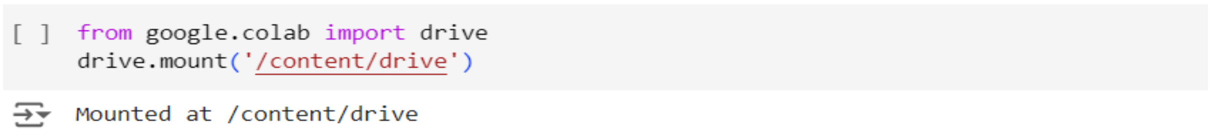
In this phase, we explore the dataset to understand its structure, features, and distributions. Key steps include:

- Loading the dataset.
- Checking for missing values and outliers.
- Descriptive statistics to understand numerical features.
- Visualizations to understand the distributions of features.

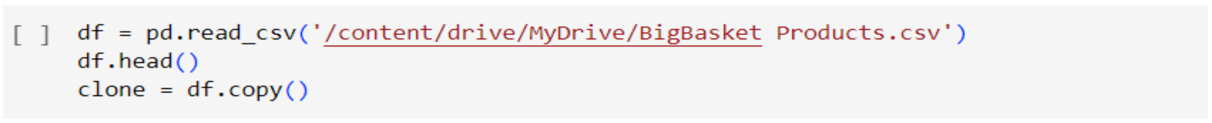
Importing all the required python libraries to perform further steps.



```
#importing all the required python libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```



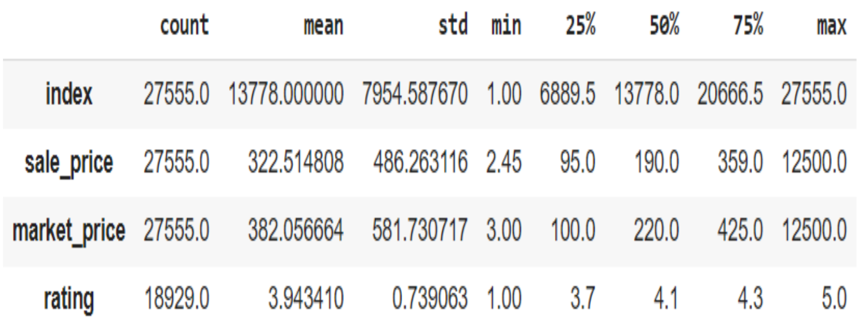
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```



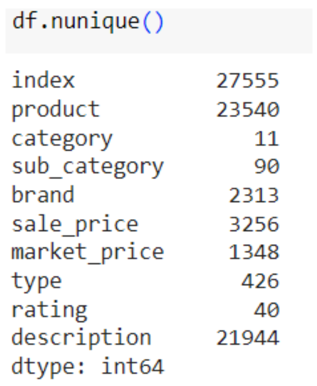
```
[ ] df = pd.read_csv('/content/drive/MyDrive/BigBasket Products.csv')
    df.head()
    clone = df.copy()
```

Performing Exploratory Data Analysis (EDA) by using the following steps.

- df.shape
- df.isnull.sum()
- df.describe()
- df.nunique()

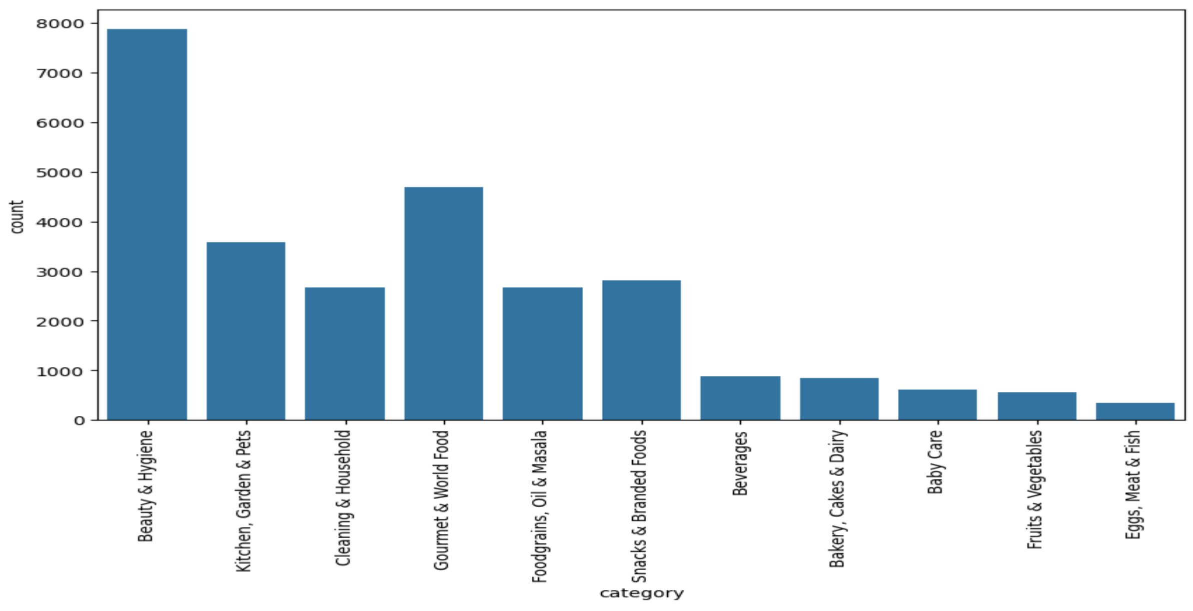


	count	mean	std	min	25%	50%	75%	max
index	27555.0	13778.000000	7954.587670	1.00	6889.5	13778.0	20666.5	27555.0
sale_price	27555.0	322.514808	486.263116	2.45	95.0	190.0	359.0	12500.0
market_price	27555.0	382.056664	581.730717	3.00	100.0	220.0	425.0	12500.0
rating	18929.0	3.943410	0.739063	1.00	3.7	4.1	4.3	5.0

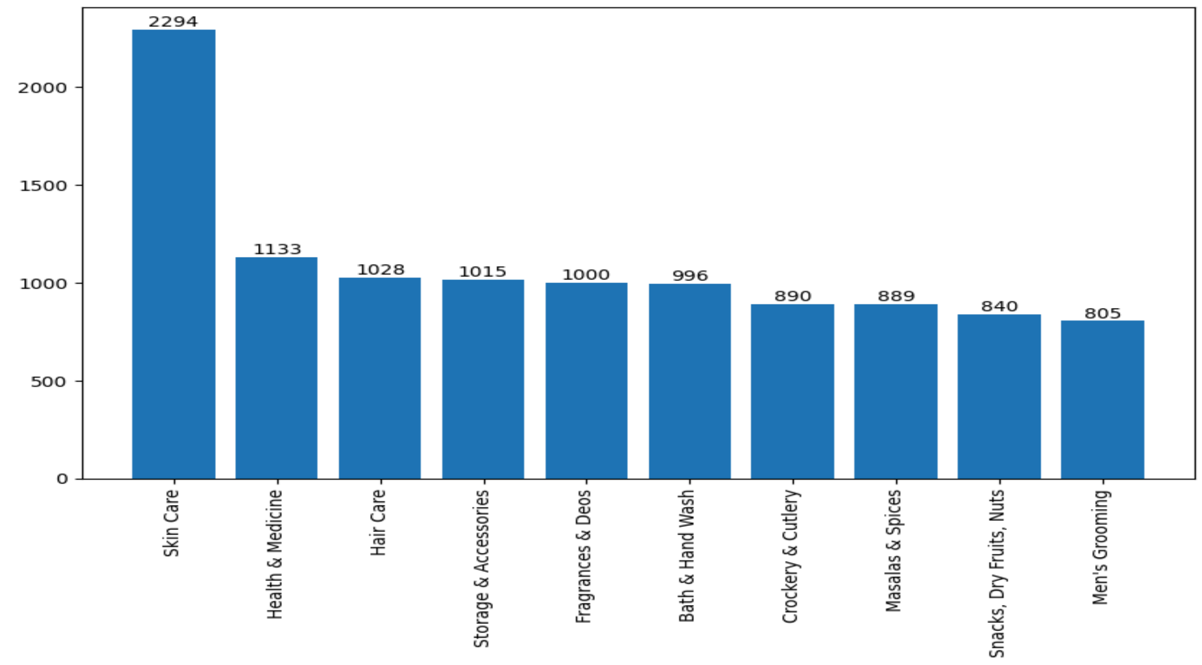


```
df.nunique()

index           27555
product         23540
category           11
sub_category       90
brand            2313
sale_price       3256
market_price     1348
type              426
rating             40
description     21944
dtype: int64
```

Here is the count plot for the category column showing all unique categories.

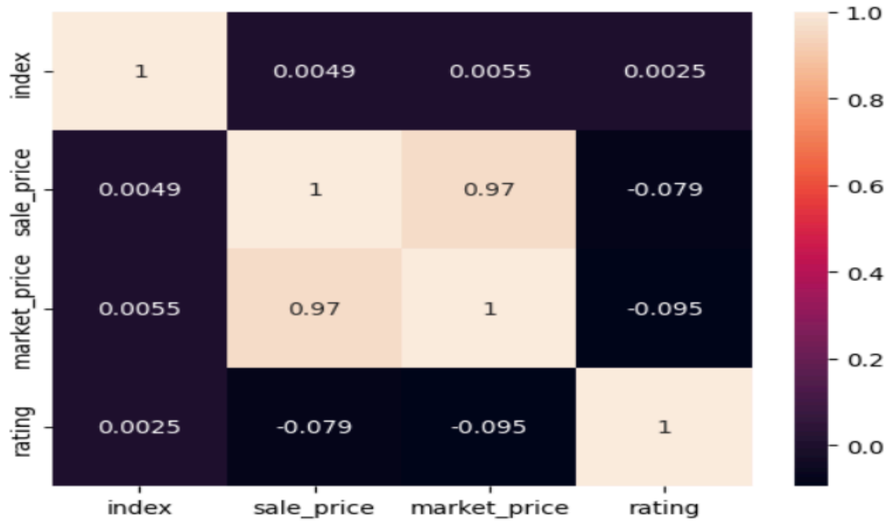


There are a total of 90 unique sub_categories, Here are the top 10 sub_categoires with maximum counts.



Here is a correlation matrix which shows that market_price and sale_price are highly correlated.

```
sns.heatmap(df.corr(numeric_only=True), annot=True)
plt.show()
#Here we can see that market_price and sale_price are highly correlated to each other
```



3.2 Data Preprocessing

In a dataset with columns like category, sub-category, type, and brand, which contain a large number of unique values, it can be challenging to handle these high cardinality features effectively. One common approach is to replace less frequent values with a more generic category, such as 'Others', and then apply frequency encoding. This process helps reduce the dimensionality of the data and can improve the performance of machine learning models.

3.2.1 Handling High Cardinality Features

Step 1: Identifying Rare Categories

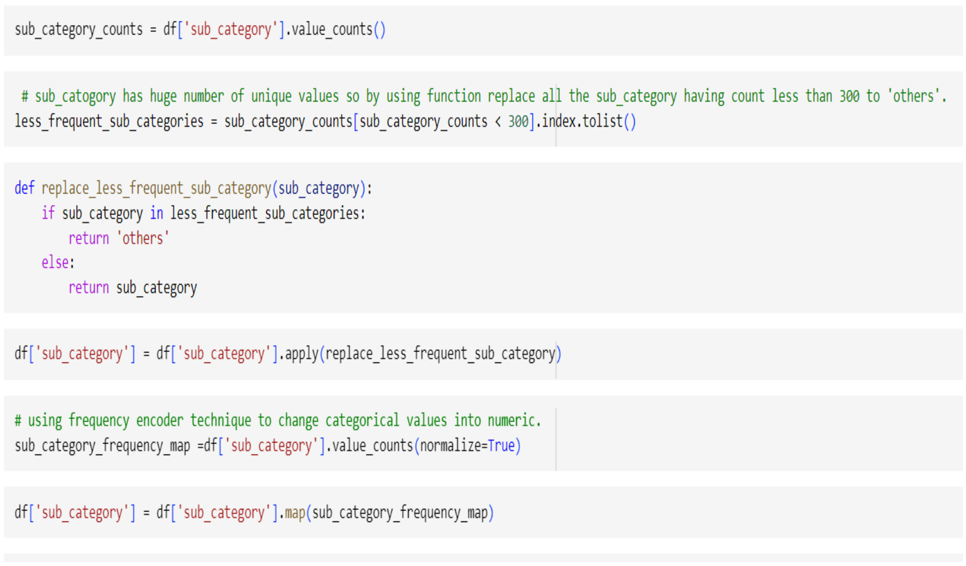
The first step is to identify the categories in each column that appear infrequently. This can be done by calculating the frequency of each category and then deciding on a threshold below which the categories will be considered rare. For example, if a category appears in less than 5% of the rows, it can be considered rare.

Step 2: Frequency Encoding

Frequency encoding is a technique where each unique category value in a column is replaced with its corresponding frequency from the original dataset. This method converts categorical data into numerical format, which can be directly used by machine learning algorithms. The advantage of frequency encoding is that it preserves the information about the distribution of the categories within the data.

Why Frequency Encoding?

1. Simplicity: It is easy to implement and interpret. The encoded values represent the prevalence of each category.
2. Information Preservation: It retains information about the distribution of categories, which can be useful for many machine learning algorithms.
3. Reduced Dimensionality: Unlike one-hot encoding, which increases the number of features significantly, frequency encoding results in only one additional feature per categorical variable.



```
sub_category_counts = df['sub_category'].value_counts()
```

```
 # sub_catogory has huge number of unique values so by using function replace all the sub_category having count less than 300 to 'others'.
less_frequent_sub_categories = sub_category_counts[sub_category_counts < 300].index.tolist()
```

```
def replace_less_frequent_sub_category(sub_category):
    if sub_category in less_frequent_sub_categories:
        return 'others'
    else:
        return sub_category
```

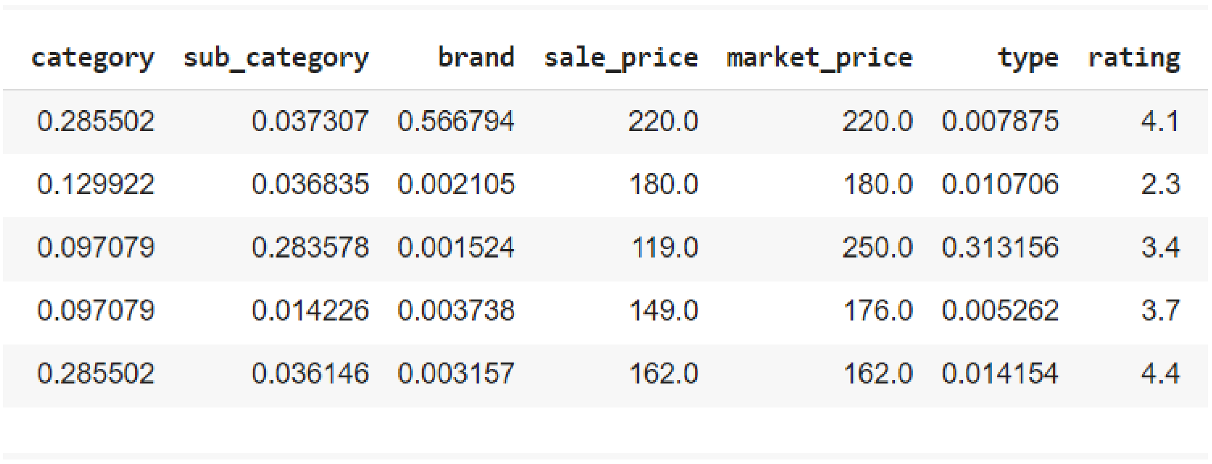
```
df['sub_category'] = df['sub_category'].apply(replace_less_frequent_sub_category)
```

```
# using frequency encoder technique to change categorical values into numeric.
sub_category_frequency_map =df['sub_category'].value_counts(normalize=True)
```

```
df['sub_category'] = df['sub_category'].map(sub_category_frequency_map)
```

By replacing rare categories with 'Others' and applying frequency encoding, we can effectively manage high cardinality features and convert them into a format suitable for machine learning models. This approach balances the need to reduce dimensionality with the preservation of important distributional information about the categories. Frequency encoding is particularly useful for categorical variables with many unique values, providing a numeric representation that models can easily process.

Now we have columns with numeric data.



category	sub_category	brand	sale_price	market_price	type	rating
0.285502	0.037307	0.566794	220.0	220.0	0.007875	4.1
0.129922	0.036835	0.002105	180.0	180.0	0.010706	2.3
0.097079	0.283578	0.001524	119.0	250.0	0.313156	3.4
0.097079	0.014226	0.003738	149.0	176.0	0.005262	3.7
0.285502	0.036146	0.003157	162.0	162.0	0.014154	4.4

4. Model Selection:

We'll select appropriate machine learning models for predicting sale prices. In this project, we'll consider:

- Linear Regression
- Decision Tree
- Random Forest
- Gradient boost

5. Model Training

In the next step of our project, we will import the `StandardScaler` and `train_test_split` functions from the Scikit-learn library. The `StandardScaler` will be used to standardize our numerical features by removing the mean and scaling to unit variance, which is particularly important for models like Linear Regression and Gradient Boosting that can be sensitive to feature scaling.

Standardization ensures that each feature contributes equally to the model's performance, preventing features with larger ranges from dominating the prediction process. The `train_test_split` function will be utilized to divide our dataset into training and testing sets. By splitting the data, we can train our models on one portion of the dataset and evaluate their performance on another, unseen portion. This helps to assess how well the models generalize to new data, preventing overfitting and ensuring robust and reliable predictions.

5.1 Importing Models and Making Predictions

In this phase of our project, we will import the necessary machine learning models: Linear Regression, Random Forest, Decision Tree, and Gradient Boosting. We will then proceed to train these models on our training data and evaluate their performance by calculating the Mean Squared Error (MSE) on the test set. MSE is a common metric used to measure the average squared difference between the observed and predicted values, providing insight into the accuracy of our models.

```python
#importing all the required models
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
```

```python
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor()
}
```

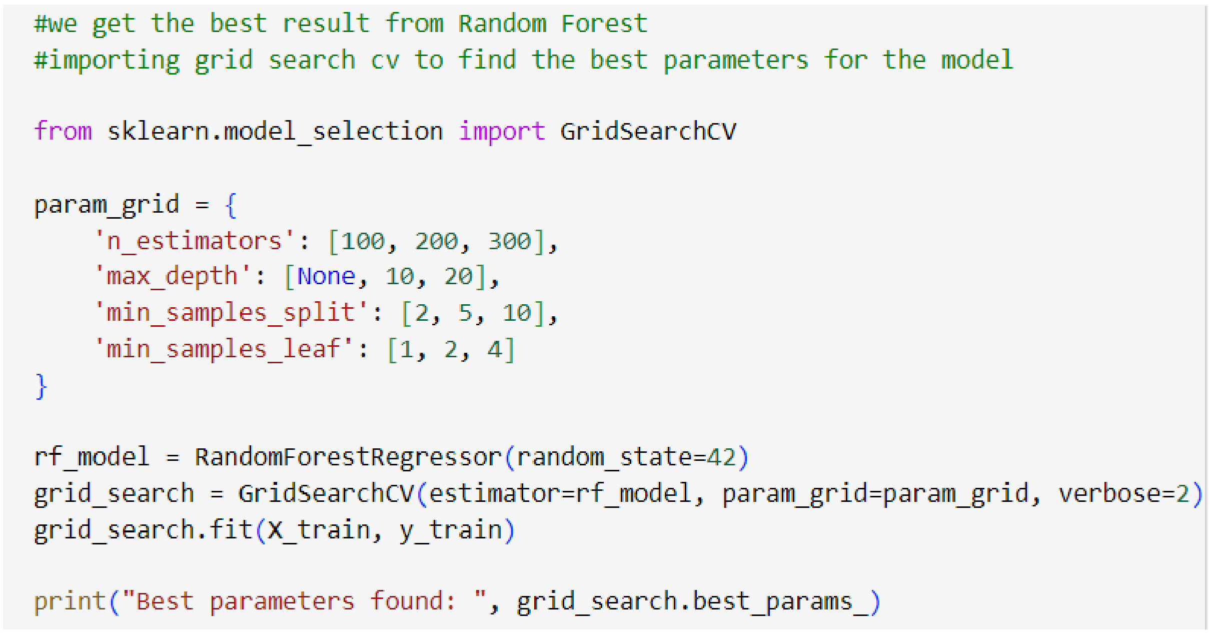
```python
# Calculating the MSE for each model.
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"{name} MSE: {mse}")
```

```
Linear Regression MSE: 0.07071377474439754
Decision Tree MSE: 0.0748626524054685
Random Forest MSE: 0.04383970163948539
Gradient Boosting MSE: 0.05156709492889826
```

5.2 Final Model Selection: Random Forest Regressor

Based on the Mean Squared Error (MSE) scores from our initial model evaluations, the Random Forest Regressor has demonstrated the lowest error, indicating it is the most accurate model for predicting sale prices in our dataset. Therefore, we will proceed with the Random Forest Regressor as our final model.

To further improve the model's performance, we can perform hyperparameter tuning using GridSearchCV or RandomizedSearchCV. Grid Search CV is a powerful and automated method for hyperparameter tuning in machine learning models. By systematically searching through a grid of parameter values and using cross-validation to assess their performance, it ensures that the best combination of parameters is found, leading to improved model accuracy and robustness.



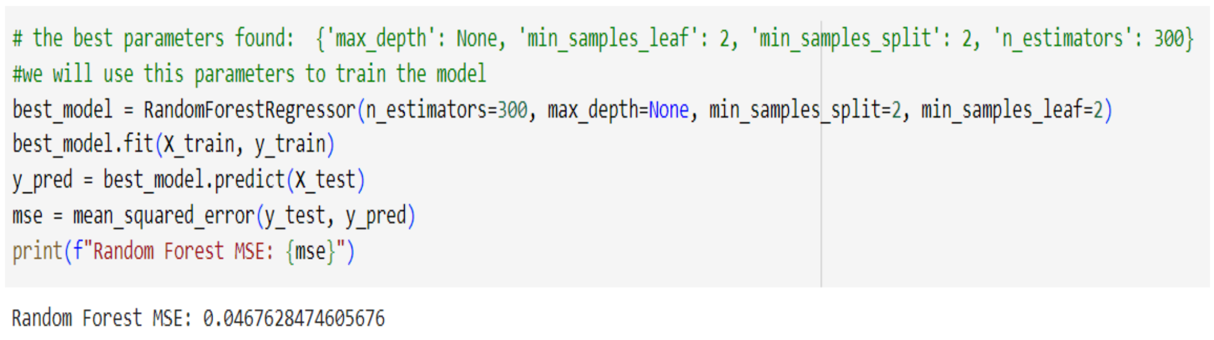
```
#we get the best result from Random Forest
#importing grid search cv to find the best parameters for the model

from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_model = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, verbose=2)
grid_search.fit(X_train, y_train)

print("Best parameters found: ", grid_search.best_params_)
```

5.3 Model Evaluation with Best Hyperparameters

After using Grid Search CV to determine the optimal hyperparameters for the Random Forest Regressor, we will now evaluate the model's performance using the following metrics:

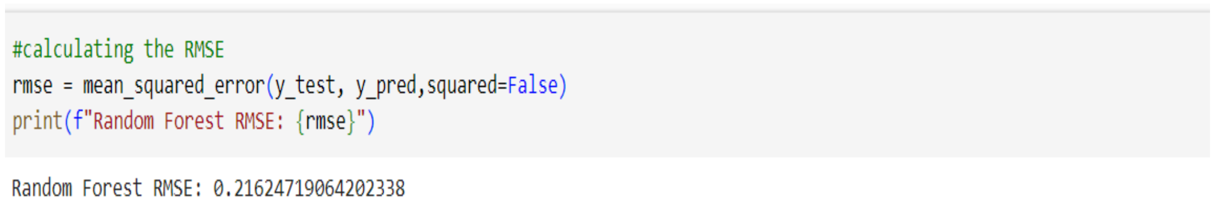
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared (R^2) Score
- Adjusted R-squared Score

These metrics provide a comprehensive understanding of the model's predictive accuracy and its goodness of fit.



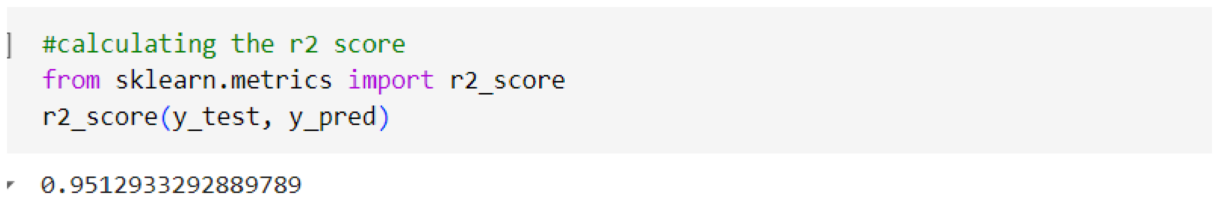
```
# the best parameters found:  {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 300}
#we will use this parameters to train the model
best_model = RandomForestRegressor(n_estimators=300, max_depth=None, min_samples_split=2, min_samples_leaf=2)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Random Forest MSE: {mse}")
```

```
Random Forest MSE: 0.0467628474605676
```



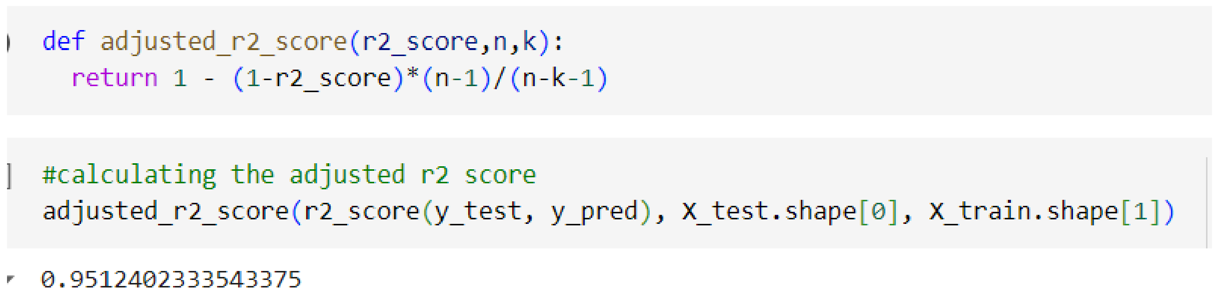
```
#calculating the RMSE
rmse = mean_squared_error(y_test, y_pred,squared=False)
print(f"Random Forest RMSE: {rmse}")
```

```
Random Forest RMSE: 0.21624719064202338
```



```
#calculating the r2 score
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.9512933292889789
```



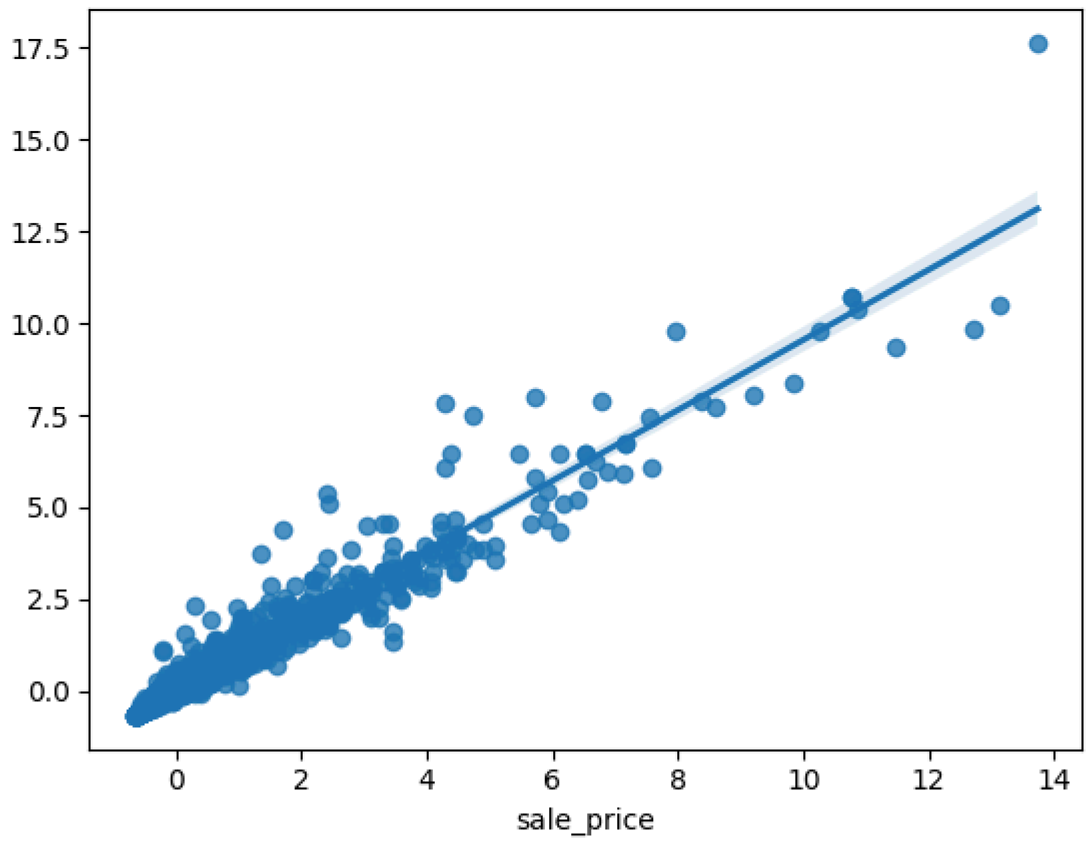
```
def adjusted_r2_score(r2_score,n,k):
  return 1 - (1-r2_score)*(n-1)/(n-k-1)
```

```
#calculating the adjusted r2 score
adjusted_r2_score(r2_score(y_test, y_pred), X_test.shape[0], X_train.shape[1])
```

```
0.9512402333543375
```

5.4 Visualisation of Actual Values vs Predicted Values

In our analysis, we visualised the relationship between the actual sale prices and the predicted sale prices generated by our Random Forest Regressor model. The graph reveals a close alignment between the two sets of values, indicating the model's effectiveness in capturing the underlying patterns within the data. The plot showcases a strong linear relationship, with the predicted values closely tracking the actual values across a range of sale prices. This visual confirmation underscores the accuracy and reliability of our predictive model in estimating sale prices for products in Big Basket. Such close agreement between actual and predicted values instils confidence in the model's ability to make precise predictions, offering valuable insights for decision-making and strategic planning within the retail industry.



Conclusion

In this project, we utilized data science techniques to develop a predictive model for sale price prediction for Big Basket. Through thorough data exploration, preprocessing, and feature engineering, we prepared the dataset for modeling. By employing machine learning algorithms such as Random Forest Regressor and rigorous evaluation, we identified a highly accurate model with an R^2 score of 0.9521, indicating strong predictive performance. This achievement holds significant implications for optimizing pricing strategies, inventory management, and customer satisfaction in the retail industry. While this model demonstrates success, ongoing refinement and validation against new data are essential for ensuring its reliability and relevance. Overall, this project showcases the transformative potential of data science in empowering businesses to make informed decisions and drive success in competitive markets.

References

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
3. https://www.databricks.com/glossary/machine-learning-models
4. https://letsdatascience.com/frequency-encoding/