E-COMMERCE WEBSITE ITEMS PRICE RECOMMENDATION SYSTEM

a project report submitted by

Ekta Gandhi
20MTS5710
20015587010

under the supervision of

Dr. Veena Jain
Associate Professor, Deen Dayal Upadhyaya College
University of Delhi, Delhi

and

Mr. Himanshu Kaundal
Insights Autotech Consultants

in partial fulfilment of the degree

Bachelor of Science in Mathematical Science

DEEN DAYAL UPADHYAYA COLLEGE
(University of Delhi)
Sector-3, Dwarka, New Delhi - 110078

MAY 2023

DECLARATION

The Project Report titled **E-commerce website price recommendation system** is being submitted to **Department of Operational Research, Deen Dayal Upadhyaya College, University of Delhi, New Delhi - 110078** for the award of **Bachelor of Science degree in Mathematical Science**. I have prepared this project report under the guidance and supervision of **Dr. Veena Jain, Associate Professor, Deen Dayal Upadhyaya College, University of Delhi, New Delhi-110078** and **Mr. Himanshu Kaundal, Insights Autotech Consultants**.

This project work has not been submitted in full or in part for the award of any other degree or diploma in any other university or institute to the best of my knowledge.

EKTA GANDHI
B.sc Mathematical Sciences
20MTS5710
20015587010

CERTIFICATE

I hereby certify that the Project titled **E-commerce website price recommendation system** which is submitted by **Ekta Gandhi (20015587010)**, in partial fulfilment of the requirement for the award of the degree of **Bachelor of Science in Mathematical Sciences**, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or in full for the ward of any other Degree or Diploma to this University or elsewhere.

Place: New Delhi

Date: 22/5/2023

Dr. Veena Jain

(Supervisor)

Associate Professor

Department of Operational Research

Deen Dayal Upadhyaya College

University of Delhi

ACKNOWLEDGEMENT

I would like to convey my heartfelt gratitude to Dr. Veena Jain for her time, tremendous direction, and assistance in the completion of my project. I would not have been able to complete this project without her help and cooperation.

I would also like to thank the company, Insights Autotech Consultants and Mr. Himanshu Kaundal for allowing me to conduct this project and providing guidance for the same.

EKTA GANDHI
B.sc Mathematical Sciences
20MTS5710
20015587010

ABSTRACT

The aim of this project was to recommend the prices of items listed on an e-commerce platform using machine learning techniques. The dataset consisted of item descriptions, brand names, and other relevant features.

The first step in the project was to pre-process the text data by removing stop words, converting to lowercase, and tokenizing the data. The data was then vectorized using various techniques like CountVectorizer and TfidfVectorizer.

The next step was to train various machine learning models like Ridge Regression, Random Forest Regression and Factorization Machines using the vectorized data. The models were tuned using cross-validation techniques and hyperparameter optimization.

Finally, the trained models were used to predict the prices of items on the test set, and the predictions were evaluated using the root mean squared error metric. The predictions were then transformed back to the original price scale using the exponential function.

The results showed that FM_FTRL performed the best. Overall, the project demonstrated the effectiveness of machine learning techniques in recommend prices for items on e-commerce platforms.

Index

List of Tables

List of Figures

Chapter -1

Operational Research: An Overview

1.1 Introduction

Operations research, abbreviated as OR, is a discipline concerned with the development and implementation of advanced analytical tools for better decision-making. It is sometimes considered a branch of the mathematical sciences.
Operations research finds optimal or near-optimal solutions to complicated decision-making problems by combining tools from other quantitative sciences including as modelling, statistics, and optimization. Operations research interacts with many other fields, including industrial engineering, due to its emphasis on practical applications. Operations research is frequently focused with determining the maximum (of profit, performance, or yield) or minimum values of some real-world objective (of loss, risk, or cost). Its methods have grown to address problems in a variety of industries, having originated in military activities before World War II.

1.2 History

McClosky and Trefthen coined the term "Operational Research" in 1940 in a small town in the United Kingdom called Bowdsey. This new science came as a result of research into military operations during WWII. There were strategic and tactical problems that were so complex that expecting adequate solutions from individuals or specialists was unrealistic. As a result, military leaders convened scientists from various disciplines and organised them into teams to help solve strategic and tactical problems, i.e., to discuss, evolve, and suggest ways and means to improve the execution of various military projects. They proposed certain approaches that demonstrated remarkable progress as a result of their collaborative efforts, experience, and deliberations. This new method of studying the system's operations in a systematic and scientific manner was known as Operations Research or Operational Research (abbreviated as O.R.).

The success of military teams drew the attention of industrial managers looking for solutions to their problems immediately after the war. Industrial operation research developed along different lines in the United Kingdom and the United States of America. The critical economic situation in the United Kingdom necessitated a drastic increase in production efficiency as well as the creation of new markets. The nationalisation of a few key industries expanded the potential field for OR. As a result, OR quickly spread from military to government, industrial, social, and economic planning.

India was one of the first countries to use O.R. The first operational research unit, known as the Regional Research Laboratory, was established in Hyderabad in 1949. Simultaneously, an additional unit was established in the Defence Science Laboratory to address the Stores, Purchase, and Planning issues.

An O.R. unit was established in the Indian Statistical Institute in Calcutta in 1953. The goal was to employ O.R. techniques in National Planning and Survey. The Operations Research Society

of India was founded in 1955 and was one of the first members of the International Federation of Operations Research Societies. Today, the use of O.R. techniques has spread from the army to a diverse range of departments at all levels.

1.3 Scientific Method in Operations Research

The scientific method is the most important feature of Operations Research. It is divided into three phases:

1.3.1 Judgement Phase: This phase consists of the following steps:

- identifying the real-life problem,
- selecting an appropriate goal and the values of various variables related to the goals,
- selecting an appropriate scale of measurement, and
- formulating an appropriate model of the problem, abstracting the essential information so that a solution at the decision-goal makers can be sought.

1.3.2 Research Phase: This is the largest and longest of the two phases. However, the remaining two are equally important because they serve as the foundation for a scientific method. This phase includes:

- observations and data collection to gain a better understanding of the problem,
- hypothesis and model formulation,
- observation and experimentation to test the hypothesis with additional data,
- analysis of the available information and hypothesis verification using pre-established measures of effectiveness,
- predictions of various results from the hypothesis, and
- generalization of the results and conjectures.

1.3.3 Action Phase: This phase consists of making recommendations for the decision-making process by those who first posed the problem for consideration, or by anyone in a position to make a decision affecting the operation in which the problem occurred.

1.4 Advantage of Operations Research

1.4.1 Better Systems: Often, an O.R. approach is initiated to analyse a specific decision-making problem, such as the best location for factories, whether to open a new warehouse, and so on. It also aids in the selection of cost-effective modes of transportation, job sequencing, production scheduling, and the replacement of old machinery, among other things

1.4.2 Better Control: The management of large organisations recognises that providing continuous executive supervision to every routine task is a difficult and costly endeavour. An O.R. approach may provide an analytical and quantitative foundation for the executive to identify the problem area. The most commonly used applications in this category are those for production scheduling and inventory replenishment.

1.4.3 Better Decisions: O.R. models aid in decision making and reduce the possibility of making incorrect decisions. The O.R. approach provides the executive with a better understanding of how he makes decisions.

1.4.4 Better Coordination: An operations-research-oriented planning model aids in the coordination of a company's various divisions.

1.5 Limitations of Operations Research

1.5.1 Reliance on an electronic computer: O.R. techniques attempt to find an optimal solution while taking all factors into account. These factors are enormous in modern society, and quantifying them and establishing relationships between them necessitate voluminous calculations that can only be handled by computers.

1.5.2 Non-Quantifiable Factors: Only when all of the elements related to a problem can be quantified can O.R. techniques provide a solution. Quantification is not possible for all relevant variables. Unquantifiable factors have no place in O.R. models.

1.5.3 Money and Time Expenses: When the basic data is subject to frequent changes, incorporating it into the O.R. models is an expensive endeavour. Furthermore, a reasonably good solution now may be preferable to a perfect O.R. solution later on.

1.5.4 Implementation: Decision implementation is a delicate task. It must account for the complexities of human relationships and behaviour.

1.6 Applications of Operations Research

Aside from scientific advancement, O.R. is primarily concerned with the techniques of applying scientific knowledge. It provides an understanding that provides the expert/manager with new insights and capabilities to determine better solutions to his decision-making problems quickly, competently, and confidently. O.R. has successfully entered many different areas of research in recent years, including Defence, Government, Service Organizations, and Industry. We briefly describe some O.R. applications in management functional areas:

1.6.1 Project Allocation and Distribution

- Optimal project resource allocation, including men, materials, machines, time, and money.
- Identifying and deploying the appropriate workforce.
- Project planning, management, and control.

1.6.2 Production and Facility Planning

- Choosing the size and location of the factory.
- Estimate the number of facilities needed.
- Forecasting for various inventory items, as well as calculating economic order quantities and reorder levels.
- Scheduling and sequencing of production runs through proper machine allocation.
- Transportation loading and unloading; and

- Warehouse location determination.
- Decisions on maintenance policy.

1.6.3 Program Decisions

- What, when, and how to buy to reduce procurement costs.
- Policies for bidding and replacement.

1.6.4 Marketing

- The timing of product introduction.
- Advertising media selection.
- Product mix selection
- Customer preferences for product size, colour, and packaging.

1.6.5 Organizational Behaviour

- Personnel Selection, Determination of Retirement Age and Skills
- Job assignment and recruitment policies.
- Employee recruitment.
- Training programme scheduling

1.6.6 Finance

- Capital needs, cash flow analysis
- Credit policies, credit risks, and so on.
- Investment choice.
- The company's profit plan.

1.6.7 Research and Development

- Product launch planning
- Oversight of R&D projects.
- Identification of research and development priorities.
- Project selection and budget preparation.
- Development project dependability and control as a result, it is possible to conclude that operation research can be widely used in management decisions and can also be used as a corrective measure.

1.7 Some Operation Research Techniques

1.7.1 Linear Programming: Linear Programming (LP) is a mathematical technique for allocating a fixed number of resources to meet a number of demands in such a way that some objective is optimised while other defined conditions are also met.

1.7.2 Queuing Theory: Queuing theory helps in estimating number of people waiting in queue, the expected waiting time in the queue, the server's expected idle time, and so on. Thus, this theory can be applied in situations where decisions must be made to minimise the length and duration of the queue while spending the least amount of money.

1.7.3 **Inventory Control Models:** It is concerned with the acquisition, storage, and handling of inventories in order to ensure inventory availability whenever needed while minimising waste and losses. It assists managers in determining the best reordering time, level, and quantity.

1.7.4 **Nonlinear Programming:** These methods can be used when the optimization problem or some of the constraints are not linear. Non-linearity can be created by factors such as a discount on the purchase price of large quantities.

1.7.5 **Network Scheduling-PERT and CPM:** A network scheduling technique is used to plan, schedule, and monitor large projects. Such large projects are common in the fields of construction, maintenance, computer system installation, R&D design, and so on. Projects undergoing network analysis are divided into individual tasks, which are then arranged in a logical sequence by determining which activities should be performed concurrently and which should be performed sequentially.

1.7.6 **Game Theory:** It is used to make decisions in conflicting situations with one or more opponents (i.e., players). In game theory, we consider two or more people with different goals, each of whose actions influences the game's outcomes. The game theory provides solutions to such games, assuming that each player wishes to maximise his profits while minimising his losses.

1.7.7 **Transportation Problem:** The transportation problem is a subset of linear programming problems in which the goal is to minimise the cost of distributing a product from multiple sources to multiple destinations.

1.7.8 **Simulation:** It is a technique that entails creating a model of a real-world situation and then conducting experiments on it. When it is too dangerous, difficult, or time consuming to conduct a real study or experiment to learn more about a situation, simulation is used.

Chapter -2

Exploratory Data Analysis

Data cleaning is a crucial process in Data Mining. It carries an important part in the building of a model. Data Cleaning can be regarded as the process needed, but everyone often neglects it. Data quality is the main issue in quality information management. Data quality problems occur anywhere in information systems. These problems are solved by data cleaning.

Data cleaning is fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabelled.

Generally, data cleaning reduces errors and improves data quality. Correcting errors in data and eliminating bad records can be a time-consuming and tedious process, but it cannot be ignored. Data mining is a key technique for data cleaning. Data mining is a technique for discovering interesting information in data. Data quality mining is a recent approach applying data mining techniques to identify and recover data quality problems in large databases. Data mining automatically extracts hidden and intrinsic information from the collections of data. Data mining has various techniques that are suitable for data cleaning.

Understanding and correcting the quality of your data is imperative in getting to an accurate final analysis. The data needs to be prepared to discover crucial patterns. Data mining is considered exploratory. Data cleaning in data mining allows the user to discover inaccurate or incomplete data before the business analysis and insights.

In most cases, data cleaning in data mining can be a laborious process and typically requires IT resources to help in the initial step of evaluating your data because data cleaning before data mining is so time-consuming. But without proper data quality, your final analysis will suffer inaccuracy, or you could potentially arrive at the wrong conclusion.

2.1 Handle missing data

There are different ways to handle missing values in a dataframe for ML models, including:

2.1.1 Removing missing values: One way to handle missing values is to remove the rows or columns containing missing values. This approach is appropriate when the number of missing values is small compared to the total number of observations or when the missing values do not affect the target variable significantly.

2.1.2 Imputing missing values: Imputation is the process of filling in missing values with estimated or predicted values. There are different techniques for imputing missing values, such as mean, median, mode imputation, k-Nearest Neighbors (k-NN) imputation, and regression imputation.

2.1.3 Creating a new category for missing values: If the missing values represent a meaningful category, such as "unknown" or "not applicable," they can be encoded as a separate category in the dataset.

2.1.4 Using algorithms that handle missing values: Some ML algorithms, such as decision trees and random forests, can handle missing values without the need for imputation or removal. These algorithms split the data based on available features and ignore missing values during the splitting process.

2.1.5 Using domain knowledge: In some cases, domain knowledge can help to impute missing values or determine their meaning. For example, in a medical dataset, missing values for a particular test result may indicate that the test was not performed or that the result was inconclusive.

2.2 Handle Categorical Data

Categorical data is a type of data that is used to group information with similar characteristics, while numerical data is a type of data that expresses information in the form of numbers.

Example of categorical data: gender

2.3 Why do we need encoding?

- Most machine learning algorithms cannot handle categorical variables unless we convert them to numerical values
- Many algorithm's performances even vary based upon how the categorical variables are encoded

2.4 Cleaning of Categorical Data

2.4.1 Convert all text to lowercase

Converting all text to lowercase is important to ensure that there are no issues with case sensitivity when the model is trained. For example, the words "Cat" and "cat" should be treated as the same word.

2.4.2 Remove digits

Digits may not be useful in text data, and can also be safely removed. This step can be achieved using regular expressions.

2.4.3 Remove punctuations

Punctuations such as commas, periods, and question marks do not add much meaning to text and can be safely removed. This step can be achieved using regular expressions.

2.4.4 Remove stop words

Stop words are common words such as "the," "a," and "an" that are unlikely to be useful for a machine learning model. Removing these words can help improve model accuracy by reducing the amount of noise in the data. There are many libraries available that provide pre-defined lists of stop words for various languages. For example, in Python, the NLTK library provides a list of English stop words.

2.4.5 Tokenize the text

Tokenization is the process of splitting the text into individual words or tokens. This step is important because it allows the model to work with individual words instead of entire sentences. There are several tokenization techniques available, such as whitespace tokenization, regular expression tokenization, and word-based tokenization.

2.4.6 Stemming or lemmatization

Stemming or lemmatization is the process of reducing words to their root form. This is useful because it can help reduce the number of unique words in the data, making it easier for the model to work with. For example, the words "running," "runs," and "run" would all be reduced to "run." There are several algorithms available for stemming and lemmatization, such as the Porter stemming algorithm and the WordNet lemmatizer.

2.5 Sentiment Analysis

Sentiment analysis is the process of determining the emotional tone of a piece of text. It involves analysing text to determine whether it expresses a positive, negative, or neutral sentiment. Sentiment analysis is useful in many applications, such as customer service, social media monitoring, and market research.

There are several techniques used for sentiment analysis, including rule-based approaches and machine learning approaches. Rule-based approaches involve creating a set of rules and patterns to identify sentiment in text. These rules may be based on specific keywords or phrases that are associated with positive or negative sentiment. Machine learning approaches involve training a model on a large dataset of labelled text, where each piece of text is associated with a positive, negative, or neutral sentiment. The model can then be used to classify new text based on its sentiment.

Sentiment analysis can be performed at different levels of granularity, such as at the document level, sentence level, or aspect level. At the document level, the overall sentiment of a piece of text is determined. At the sentence level, the sentiment of each individual sentence is determined. At the aspect level, the sentiment of specific aspects or entities in the text is determined. For example, in a product review, the sentiment of the product features may be analysed separately from the overall sentiment of the review.

Sentiment analysis can be challenging because language is often complex and nuanced, and the meaning of a piece of text can depend on its context. Additionally, sarcasm, irony, and other forms of figurative language can be difficult to detect. However, with the right techniques and tools, sentiment analysis can provide valuable insights into the emotions and opinions expressed in text data.

2.6 Data Visualization

Data visualization is the process of representing data visually, using charts, graphs, and other visual elements. The goal of data visualization is to make complex data more accessible and understandable to users, by presenting it in a format that is easy to interpret and analyse.

There are many different types of data visualizations, including:

2.6.1 Bar charts: These are used to show categorical data, and are useful for comparing different categories.

2.6.2 Line charts: These are used to show trends over time, and are useful for visualizing changes in data over a period of time.

2.6.3 Scatter plots: These are used to show the relationship between two variables, and are useful for identifying correlations and patterns in data.

2.6.4 Heat maps: These are used to show the distribution of data across a two-dimensional space, and are useful for identifying patterns in large datasets.

2.6.5 Tree maps: These are used to show hierarchical data, and are useful for visualizing the structure of complex datasets.

In addition to these types of visualizations, there are many other techniques and tools available for data visualization, such as dashboards, infographics, and interactive visualizations.

Data visualization is an important part of data analysis, because it allows users to quickly and easily identify patterns and trends in large datasets. By using visualizations to represent data, users can gain insights that may not be immediately apparent from the raw data. Additionally, visualizations can help users communicate their findings to others in a clear and compelling way, making it easier to share and collaborate on data analysis projects.

2.7 Label Encoding

Label Encoding is a technique used to convert categorical data into numerical data. In machine learning, algorithms typically require numerical data as input, which means that categorical data must be converted before it can be used to train a model.

Label Encoding involves assigning a numerical value to each unique category in the data. For example, if the data contains the categories "red," "green," and "blue," they might be encoded as 0, 1, and 2 respectively. The numerical values assigned to each category are arbitrary and have no inherent meaning; their purpose is simply to allow the data to be represented numerically.

Label Encoding is a simple and effective way to convert categorical data into numerical data, and is often used in machine learning pipelines. However, it has some limitations. For example, assigning numerical values to categories creates an implicit order between them, which may not always be appropriate. Additionally, if a categorical feature has many unique values, the resulting numerical feature may not provide much additional information to the model.

Overall, Label Encoding is a useful technique for converting categorical data into numerical data, but it should be used with caution and in conjunction with other techniques such as One-Hot Encoding or Target Encoding, depending on the specific requirements of the machine learning model being used.

2.8 Feature Engineering

Feature engineering is the process of transforming raw data into a set of features that can be used to train a machine learning model. The goal of feature engineering is to extract relevant information from the data and represent it in a way that is suitable for the model being used. Feature engineering involves several steps, including:

2.8.1 Data cleaning: This involves removing missing or erroneous data, and converting data into a consistent format.

2.8.2 Feature selection: This involves selecting the most important features from the data, based on their relevance to the problem being solved.

2.8.3 Feature extraction: This involves transforming the raw data into a set of features that can be used to train a model. This may involve techniques such as scaling, normalization, or one-hot encoding.

2.8.4 Feature transformation: This involves creating new features from existing features, using techniques such as polynomial expansion or principal component analysis (PCA).

2.8.5 Feature scaling: This involves scaling the features to ensure that they are all on a similar scale, to prevent some features from dominating others during the training process.

The process of feature engineering is highly dependent on the specific problem being solved and the data being used. It requires a deep understanding of the data and the problem domain, as well as knowledge of the machine learning algorithms being used.

Effective feature engineering can greatly improve the performance of a machine learning model, by providing it with relevant and informative features to train on. However, it is a time-consuming and iterative process that requires careful consideration and experimentation to get right.

2.9 New Features

Creating new features is an important aspect of feature engineering. It involves extracting relevant information from the existing features and combining them in a way that can improve the performance of the machine learning model. Here are some common techniques for creating new features:

2.9.1 Polynomial features: This technique involves creating new features by multiplying existing features together. For example, if a dataset has two features, x and y, we can create a new feature x^2 by squaring the feature x. Similarly, we can create a new feature xy by multiplying x and y together.

2.9.2 Interaction features: This technique involves creating new features by combining two or more existing features in an additive or subtractive way. For example, if a dataset has two features, x and y, we can create a new feature x+y by adding the two features together. Similarly, we can create a new feature x-y by subtracting x from y.

2.9.3 Time-based features: This technique involves creating new features based on the of day, week, or month. For example, if a dataset contains a date column, we can create a new feature that represents the day of the week, or the month of the year.

2.9.4 Text-based features: This technique involves creating new features based on the content of text data. For example, we can create a new feature that represents the length of a text string, or the frequency of a particular word or phrase.

2.9.5 Domain-specific features: This technique involves creating new features based on domain-specific knowledge or intuition. For example, in a medical dataset, we might create a new feature that represents the patient's age, or their body mass index (BMI).

These are just a few examples of techniques for creating new features. The key is to think creatively and experiment with different approaches to see which ones improve the performance of the machine learning model.

2.10 Label Binarizer

LabelBinarizer is a pre-processing technique used in machine learning to convert a categorical variable into a binary matrix. It is used to transform the categorical labels into numerical values that can be used as input for machine learning models.

Here is how the LabelBinarizer works:

- The LabelBinarizer takes a single categorical column as input, where each row represents a categorical label.
- The LabelBinarizer then converts the categorical labels into numerical values. Each unique categorical value is assigned a numerical value, starting from 0 and incrementing by 1 for each unique value.
- The LabelBinarizer then converts the numerical values into a binary matrix. For each row in the original categorical column, the corresponding row in the binary matrix has a 1 in the column corresponding to the numerical value of the original categorical label, and 0s in all other columns.

2.11 Tfidf Vectorizer

TF-IDF (Term Frequency - Inverse Document Frequency) vectorizer is a common technique used in natural language processing to convert text data into a numeric format that can be used as input for machine learning models.

Here's how the TF-IDF vectorizer works:

- The text data is first preprocessed, which may include tokenization, removing stopwords, and stemming.

- The TF-IDF vectorizer then calculates the term frequency for each word in the text data. Term frequency is simply the number of times a word appears in a document divided by the total number of words in the document.
- The TF-IDF vectorizer then calculates the inverse document frequency for each word in the text data. Inverse document frequency is a measure of how important a word is to the overall corpus of documents. It is calculated as the logarithm of the total number of documents divided by the number of documents containing the word.
- The TF-IDF vectorizer then multiplies the term frequency and inverse document frequency to get the TF-IDF score for each word in the text data.
- Finally, the TF-IDF vectorizer creates a matrix where each row represents a document and each column represents a word. The values in the matrix are the TF-IDF scores for each word in each document.

2.12 Sparse matrix

A sparse matrix is a matrix that contains a large number of zero elements. In other words, a sparse matrix is a matrix where most of the elements are zero. Sparse matrices are often encountered in scientific and engineering applications, where they are used to represent data that is naturally sparse, such as networks, graphs, and text documents.

To save memory and computation time, sparse matrices are often stored in compressed formats.

2.13 Compressed Sparse Row

Compressed Sparse Row (CSR) is a format for representing sparse matrices in a compressed format. In a sparse matrix, most of the elements are zero, so it is inefficient to store all the zeros. The CSR format stores only the non-zero elements of a sparse matrix, along with their row and column indices, in three separate arrays.

The CSR format is useful when working with large sparse matrices, as it reduces the amount of memory required to store the matrix.

Here's how the CSR format works:

- The non-zero elements of the matrix are stored in a 1D array, in row-major order.
- An array of column indices is created, which gives the column index for each non-zero element.
- An array of row pointers is created, which gives the index in the non-zero element array where each row starts.

Chapter -3

Machine Learning Techniques

3.1 Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable (also called the response variable or outcome variable) and one or more independent variables (also called explanatory or predictor variables). It is a technique that is commonly used in the field of statistics and machine learning to analyze and predict numerical data.

In linear regression, the goal is to find the best linear relationship between the dependent variable and the independent variables. This relationship is expressed as a mathematical equation in the form of a straight line, which can be used to make predictions about the value of the dependent variable given values of the independent variables.

The basic idea of linear regression is to find the line that best fits the data. The line is defined by an intercept (the point at which the line crosses the y-axis) and a slope (the rate at which the line rises or falls as the x-axis increases). The slope of the line represents the change in the dependent variable for a one-unit increase in the independent variable.

There are different types of linear regression, including simple linear regression, multiple linear regression, and polynomial regression. Simple linear regression involves only one independent variable, while multiple linear regression involves more than one independent variable. Polynomial regression allows for a more complex relationship between the dependent and independent variables by fitting a polynomial function to the data.

Linear regression is widely used in various fields, including economics, engineering, finance, and social sciences. It is often used for forecasting, modeling relationships between variables, and identifying patterns in data.

3.2 Ridge Regression

Ridge regression is a regularized linear regression technique that is used to deal with the problem of multicollinearity (correlation between independent variables) in the data. It is a modification of ordinary least squares regression that adds a penalty term to the regression equation to reduce the impact of multicollinearity.

In ridge regression, the goal is still to find the best linear relationship between the dependent variable and the independent variables, but with the addition of a regularization term. This term penalizes the magnitude of the coefficients (slopes) of the independent variables, forcing them to be small and thus reducing the impact of multicollinearity. The amount of regularization is controlled by a hyperparameter called lambda (λ), which determines the strength of the penalty term.

The ridge regression equation can be written as:

$Y = b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n + \lambda(b_1^2 + b_2^2 + ... + b_n^2)$

Where Y is the dependent variable, X1, X2, ..., Xn are the independent variables, b_0 is the intercept, and b_1, b_2, ..., b_n are the coefficients. The term $\lambda(b_1^2 + b_2^2 + ... + b_n^2)$ is the penalty term, which is added to the least squares objective function to minimize the sum of the squared errors (SSE) between the predicted and actual values.

Ridge regression can be used when the number of independent variables is greater than the number of observations or when the independent variables are highly correlated. It can improve the accuracy of the linear regression model and prevent overfitting by reducing the variance of the coefficients.

One of the disadvantages of ridge regression is that it does not perform feature selection, meaning that all independent variables are included in the model. However, it can still be useful in situations where multicollinearity is present and feature selection is not the main goal.

3.3 Lasso Regression

Lasso regression (short for Least Absolute Shrinkage and Selection Operator) is a regularized linear regression technique that is used for feature selection and to deal with the problem of multicollinearity (correlation between independent variables) in the data. It is a modification of ordinary least squares regression that adds a penalty term to the regression equation to reduce the impact of multicollinearity and shrink the coefficients of less important variables to zero.

In lasso regression, the goal is still to find the best linear relationship between the dependent variable and the independent variables, but with the addition of a regularization term. This term penalizes the magnitude of the coefficients (slopes) of the independent variables, forcing them to be small and shrinking less important variables to zero, thus performing feature selection. The amount of regularization is controlled by a hyperparameter called lambda (λ), which determines the strength of the penalty term.

The lasso regression equation can be written as:

$Y = b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n + \lambda(|b_1| + |b_2| + ... + |b_n|)$

Where Y is the dependent variable, X1, X2, ..., Xn are the independent variables, b0 is the intercept, and b_1, b_2, ..., b_n are the coefficients. The term $\lambda(|b_1| + |b_2| + ... + |b_n|)$is the penalty term, which is added to the least squares objective function to minimize the sum of the squared errors (SSE) between the predicted and actual values.

Lasso regression can be used for feature selection, where it identifies the most important independent variables and sets the coefficients of less important variables to zero. This is particularly useful when dealing with high-dimensional data with a large number of independent variables. Lasso regression also handles multicollinearity well by shrinking the coefficients of correlated independent variables towards zero.

One of the limitations of lasso regression is that it can only select features, not groups of correlated features. In addition, it can produce unstable and inconsistent solutions when the number of independent variables is larger than the number of observations. In such cases, it is recommended to use ridge regression or other regularized regression techniques.

3.4 Gradient Boosting Regression

Gradient boosting regression (GBR) is a machine learning technique that builds a predictive model by combining multiple weak models in a sequential manner. It is a type of boosting algorithm that uses the gradient descent method to optimize the loss function and improve the accuracy of the model.

In GBR, the goal is to minimize the loss function by iteratively adding weak models (usually decision trees) to the model and adjusting the weights of each model based on the errors made by the previous models. The final model is a weighted sum of all the weak models.

The GBR algorithm can be summarized as follows:

- Initialize the model with a constant value (e.g., the mean value of the target variable).
- Fit a weak model (e.g., a decision tree) to the training data and calculate the errors made by the model.
- Update the model by adding the weak model with a weight that minimizes the loss function.
- Repeat steps 2-3 until the desired level of accuracy is achieved or a maximum number of iterations is reached.

The main advantage of GBR is that it can handle both linear and non-linear relationships between the independent and dependent variables, making it suitable for a wide range of applications. It can also handle missing data and outliers well, and is less prone to overfitting than other machine learning techniques.

However, GBR has some disadvantages, such as being computationally intensive and sensitive to hyperparameters. It is important to tune the hyperparameters carefully to avoid overfitting and achieve the best possible performance.

Overall, GBR is a powerful machine learning technique that can be used for regression, classification, and other predictive modeling tasks.

3.5 Random Forest Regression

Random Forest Regression (RFR) is a popular machine learning algorithm that is used for regression tasks. It is a type of ensemble learning method that combines multiple decision trees to improve the accuracy and stability of the model.

In RFR, a set of decision trees is built on random subsets of the training data, and the final prediction is made by averaging the predictions of all the individual trees. This approach helps to reduce overfitting and improve the generalization ability of the model.

The RFR algorithm can be summarized as follows:

- Randomly select a subset of the training data (with replacement).
- Build a decision tree on the selected subset of data by recursively splitting the data based on the features that maximize the information gain.
- Repeat steps 1-2 for a fixed number of times (or until a stopping criterion is met) to create multiple decision trees.

- For a new data point, predict the output by averaging the predictions of all the individual trees.

The advantages of RFR include its ability to handle both linear and non-linear relationships between the independent and dependent variables, its robustness to outliers and missing data, and its ability to handle high-dimensional data with a large number of features. It is also relatively easy to implement and interpret.

However, RFR also has some limitations, such as its tendency to overfit noisy data and its potential for high computational costs when dealing with very large datasets.

Overall, RFR is a powerful and widely used machine learning algorithm that is well-suited for regression tasks in various fields, including finance, healthcare, and environmental sciences.

3.6 Stochastic Gradient Descent Regressor

Stochastic Gradient Descent (SGD) Regressor is a popular linear regression algorithm that is widely used in machine learning. It is an optimization algorithm that iteratively updates the weights of the regression coefficients to minimize the mean squared error (MSE) loss function.

In SGD, the regression coefficients are updated for each training example or mini-batch of training examples, rather than using the entire training set at once. This allows for faster convergence and lower memory requirements, making it suitable for large datasets.

The SGDRegressor is a specific implementation of SGD for linear regression tasks in Python. It has several hyperparameters that can be tuned to optimize the performance of the model, including the learning rate, the regularization strength, and the batch size.

Some advantages of SGDRegressor include its efficiency and scalability, its ability to handle large datasets and high-dimensional feature spaces, and its flexibility to work with different loss functions and regularizations. It is also relatively simple to implement and can provide good results with proper tuning of hyperparameters.

However, SGDRegressor also has some limitations, such as its sensitivity to the initial learning rate and the choice of regularization parameters. It may also require more iterations to converge to an optimal solution compared to other optimization algorithms.

Overall, SGDRegressor is a powerful and widely used linear regression algorithm that can provide accurate predictions on large datasets with high-dimensional features. It is particularly useful in situations where memory resources are limited and a fast convergence rate is desirable.

3.7 Follow-the-Regularized-Leader (FTRL)

Follow-the-Regularized-Leader (FTRL) is a machine learning algorithm that is used for online learning in classification and regression tasks. It is an extension of the Online Gradient Descent (OGD) algorithm that introduces regularization to the optimization process.

In FTRL, the weight vector is updated using a regularized update rule that balances the gradient of the loss function with a penalty term that encourages sparsity in the weight vector. The

algorithm dynamically adjusts the regularization strength based on the magnitude of the gradient and the previous weight vector, resulting in a more stable and robust optimization process.

FTRL is particularly useful in situations where the data arrives sequentially over time and needs to be processed in an online manner, such as in recommender systems or online advertising. It is also useful when dealing with high-dimensional and sparse feature spaces, as the regularization term helps to promote sparsity in the weight vector.

FTRL has several hyperparameters that can be tuned to optimize the performance of the model, including the learning rate, the regularization strength, and the penalty function. It also requires careful tuning of the hyperparameters to prevent overfitting and achieve optimal performance.

Overall, FTRL is a powerful machine learning algorithm for online learning in classification and regression tasks that can provide accurate and efficient predictions in high-dimensional and sparse feature spaces.

3.8 Factorization Machines with Follow-the-Regularized-Leader (FM_FTRL)

FM_FTRL (Factorization Machines with Follow-the-Regularized-Leader) is a machine learning algorithm that combines the concepts of Factorization Machines (FM) and Follow-the-Regularized-Leader (FTRL) to handle high-dimensional and sparse feature spaces in online learning settings.

FM_FTRL is based on the FM algorithm, which is a popular algorithm for modeling pairwise interactions between features. FM_FTRL extends FM by introducing regularization and the FTRL optimization algorithm to handle online learning in high-dimensional and sparse feature spaces.

In FM_FTRL, the weight vector is represented as a combination of feature embeddings and pairwise interaction factors, which are learned jointly using the FTRL algorithm with L1 and L2 regularization. The algorithm dynamically adjusts the regularization strength based on the magnitude of the gradient and the previous weight vector, resulting in a more stable and robust optimization process.

FM_FTRL is particularly useful in online advertising and recommender systems, where the data arrives sequentially over time and needs to be processed in an online manner. It can handle large-scale and high-dimensional feature spaces, and can learn from the interactions between features to make more accurate predictions.

FM_FTRL has several hyperparameters that can be tuned to optimize the performance of the model, including the learning rate, the regularization strength, and the embedding dimension. It also requires careful tuning of the hyperparameters to prevent overfitting and achieve optimal performance.

Overall, FM_FTRL is a powerful machine learning algorithm for online learning in recommendation systems and advertising, that can provide accurate and efficient predictions in high-dimensional and sparse feature spaces by modeling the pairwise interactions between features.

3.9 Performance Measures

R2 (R-squared), MSE (Mean Squared Error), RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and Adjusted R2 are all metrics used to evaluate the performance of regression models.

3.9.1 R2 (R-squared): It measures the proportion of the variance in the dependent variable (y) that is explained by the independent variables (X). R2 ranges between 0 and 1, with higher values indicating a better fit of the model to the data. R2 is calculated as 1 - (SSE/SST), where SSE is the sum of squared errors and SST is the total sum of squares.

3.9.2 MSE (Mean Squared Error): It measures the average squared difference between the predicted values and the actual values. MSE gives more weight to larger errors and is calculated as the average of the squared differences between the predicted and actual values.

3.9.3 RMSE (Root Mean Squared Error): It is the square root of the MSE and measures the average magnitude of the errors in the same units as the dependent variable. RMSE is a popular metric because it is interpretable and gives equal weight to all errors.

3.9.4 MAE (Mean Absolute Error): It measures the average absolute difference between the predicted values and the actual values. MAE gives equal weight to all errors and is less sensitive to outliers compared to MSE and RMSE.

3.9.5 Adjusted R2: It is a modified version of R2 that takes into account the number of independent variables in the model. Adjusted R2 penalizes the addition of unnecessary independent variables that do not improve the fit of the model. Adjusted R2 is calculated as 1 - [(1-R2)(n-1)/(n-p-1)], where n is the sample size and p is the number of independent variables in the model.

Overall, these metrics provide a way to measure the accuracy and goodness of fit of regression models, and can be used to compare different models and select the one that best fits the data.

Chapter - 4

Problem and Data Description

An E-commerce website wants us to build an algorithm that automatically suggests the right product prices from the user-inputted text descriptions of their products, including details like product category name, brand name, item condition, etc.

Business objectives and constraints

- The goal is to solve the problem of suggesting the appropriate price of products to online sellers.
- No latency constraints, because we would like to suggest a highly accurate price to the seller, even if it takes a reasonable amount of time.

Data Description

- The dataset has been taken from Kaggle.
- The training dataset for our machine learning model has a shape of (1482535, 8), with 8 columns including an additional column for the price model that needs to be trained. In contrast, the test dataset has a shape of (693359,7) without the price column as it is the target variable we need to predict.
- After dropping irrelevant rows with negative pricing, the training dataset's shape was reduced to (1481661, 8). These rows were dropped because negative pricing is not possible, and such data would affect the model's accuracy negatively.
- To ensure uniformity in the data pre-processing stage, we merged both datasets, resulting in a shape of (2175020, 8). This merging of datasets is necessary to avoid discrepancies between the training and test datasets, which can lead to inaccurate predictions by the model. Therefore, the merged dataset was used as the input for the pre-processing stage.

Data fields

The files used were tab-delimited and it contained the following columns:

- **train_id or test_id** - the id of the listing
- **name** - the title of the listing.
- **item_condition_id** - the condition of the items provided by the seller
- **category_name** - category of the listing
- **brand_name**
- **price** - the price that the item was sold for. This is the target variable that you will predict. The unit is USD. This column doesn't exist in test.tsv since that is what you will predict.
- **shipping** - 0 if shipping fee is paid by seller and 1 by buyer
- **item_description** - the full description of the item.

Chapter -5

Solution Methodology

In this report, we present the steps taken to analyse and model the data for our problem statement. The following steps were performed:

5.1 **Data Collection:** We collected all the relevant data required for our problem statement. The data was in a structured format and contained all the required fields. The data was taken from Kaggle website.

5.2 **Data Cleaning:** We cleaned the data by removing any missing values, duplicate data, and irrelevant data points. We also normalized and standardized the data to make it consistent and easier to work with.

5.3 **Data Visualization:** We visualized the data using various graphs and charts to identify patterns and trends. This helped us understand the data better and choose appropriate ML models.

5.4 **Apply ML Models:** We applied appropriate ML models such as Different types of regression. We evaluated the performance of the models using suitable metrics such as R square, MSE, MAE, RMSE.

5.5 **Analyzing The Result:** As the initial results were not satisfactory, we tried the following steps:

5.5.1 **Feature Engineering:** We performed feature engineering to extract meaningful features from the data. This helped improve the performance of the ML models.

5.5.2 **Apply same models:** Result was still not satsfactory. We modified the data by adding new data points or removing irrelevant data points. This helped improve the performance of the ML models.

5.6 **Apply Different ML Models:** As the performance was not satisfactory after adding features also, we tried applying different ML models. We chose appropriate models that were best suited for our problem statement.

5.7 **Final Result:** Once we achieved satisfactory results, we used the ML model to predict the final price.

In conclusion, the above steps helped us analyse and model the data effectively, and we were able to achieve satisfactory results. We believe that the process of data cleaning, visualization, and model building are iterative and requires continuous refinement, and these steps are crucial in achieving the desired results.

Chapter-6

Data Preprocessing and Visualization

6.1 Data Cleaning

6.1.1 Firstly, will import some necessary libraries and load data for the project.

- The pandas library is used for data manipulation and analysis.
- The numpy library is used for numerical operations.
- The matplotlib and seaborn libraries are used for data visualization.
- The nltk library is used for natural language processing.
- The vaderSentiment library is used for sentiment analysis.
- The textblob library is used for text processing.
- The sklearn library is used for machine learning algorithms and data preprocessing.
- The wordbatch library is used for text processing.

6.1.2 We loaded the training and testing datasets to do analysis.

train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

Table 6.1 : Training Dataset

test_id	name	item_condition_id	category_name	brand_name	shipping	item_description
0	Breast cancer "I fight like a girl" ring	1	Women/Jewelry/Rings	NaN	1	Size 7
1	25 pcs NEW 7.5"x12" Kraft Bubble Mailers	1	Other/Office supplies/Shipping Supplies	NaN	1	25 pcs NEW 7.5"x12" Kraft Bubble Mailers Lined...
2	Coach bag	1	Vintage & Collectibles/Bags and Purses/Handbag	Coach	1	Brand new coach bag. Bought for [rm] at a Coac...
3	Floral Kimono	2	Women/Sweaters/Cardigan	NaN	0	-floral kimono -never worn -lightweight and pe...
4	Life after Death	3	Other/Books/Religion & Spirituality	NaN	1	Rediscovering life after the loss of a loved o...

Table 6.2 : Test Dataset

6.1.3 To clean the training dataset, we removed any rows where the price column had a value of 0 or less. This resulted in a new dataframe where the price column only contained positive values and also created a new column of log price.

6.1.4 Next we have added a column to each dataframe to indicate whether the data was from the training set or the test set. This was done using the is_train column, which was set to 1 for the training set and 0 for the test set.

6.1.5 Than we concatenated the train and test dataframes into a single dataframe called merge, which we will use for further analysis.

	name	item_condition_id	category_name	brand_name	price	shipping	item_description	log_price	is_train
0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet	2.397895	1
1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...	3.970292	1
2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...	2.397895	1
3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...	3.583519	1
4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity	3.806662	1

Table 6.3 : Head of merged dataset

	name	item_condition_id	category_name	brand_name	price	shipping	item_description	log_price	is_train
693354	Quartz crystal on Flint stone	1	Home/Home Décor/Home Décor Accents	NaN	NaN	0	Flint/Quartz cluster. Self mined ✨measures 3x2...	NaN	0
693355	It Cosmetics - Travel Bundle	1	Beauty/Makeup/Makeup Sets	IT Cosmetics	NaN	1	It Cosmetics travel bundle. Includes: Brow pow...	NaN	0
693356	Galaxy S8 hard shell case	1	Electronics/Cell Phones & Accessories/Cases, C...	NaN	NaN	1	New. Free shipping Basstop case	NaN	0
693357	Hi low floral kimono	2	Women/Swimwear/Cover-Ups	NaN	NaN	0	Floral kimono. Tropical print. Open front. Hi ...	NaN	0
693358	FREESHIP 2 Floral Scrub Tops, medium.	2	Women/Tops & Blouses/T-Shirts	NaN	NaN	1	2 Floral scrub tops. Worn less than 5 times ea...	NaN	0

Table 6.4 : Tail of merged dataset

6.1.6 We split the category_name column of merge datasets into three subcategories - sub_category_1, sub_category_2, and sub_category_3 and dropped the original category_name column. This allowed us to analyze the data at a more granular level and gain insights into the specific categories of products being sold.

	name	item_condition_id	brand_name	price	shipping	item_description	log_price	is_train	sub_category_1	sub_category_2	sub_category_3
0	MLB Cincinnati Reds T Shirt Size XL	3	NaN	10.0	1	No description yet	2.397895	1	Men	Tops	T-shirts
1	Razer BlackWidow Chroma Keyboard	3	Razer	52.0	0	This keyboard is in great condition and works ...	3.970292	1	Electronics	Computers & Tablets	Components & Parts
2	AVA-VIV Blouse	1	Target	10.0	1	Adorable top with a hint of lace and a key hol...	2.397895	1	Women	Tops & Blouses	Blouse
3	Leather Horse Statues	1	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...	3.583519	1	Home	Home Décor	Home Décor Accents
4	24K GOLD plated rose	1	NaN	44.0	0	Complete with certificate of authenticity	3.806662	1	Women	Jewelry	Necklaces

Table 6.5 : merged dataset after category split

6.1.7 We first checked for missing values in our datasets and it was found that there were missing values in various columns. To handle these missing values, we replaced them with the string 'missing'. After the replacement, we again checked for missing values in the merge datasets and it was found that there were no more missing values in any of the columns except for price column, because we have merged train and test data and we have to predict price for test dataset, that's why this is missing and created a copy of merge dataset.

6.1.8 In the data cleaning phase, we started by cleaning the text data in various columns to prepare it for further analysis. Specifically, we performed several operations on the columns sub_category_1, sub_category_2, sub_category_3, brand_name and name. First, we converted all the text to lowercase using the str.lower() function. Next, we removed all special characters, punctuation, and symbols using the str.replace()

function with regular expressions. Finally, we removed any numbers from the text using the same function with the '\d+' regular expression pattern.

These steps helped to standardize the text data across the different columns and remove any unwanted characters that might affect further analysis.

6.1.9 We performed text preprocessing on the item_description also to prepare the data for analysis. We have done the same steps that are mentioned above and other than that we removed all stop words from the column. After removing stop words, we tokenized the remaining words and at last we have lemmatize the remaining words (reducing them to their base form) using a custom-defined function.

Overall, these preprocessing steps help to clean and normalize the text data, making it easier for machine learning models to analyze and classify the data accurately.

6.2 Data Visualization

6.2.1 We generated word clouds to visualize the most common words in the item_description column of both the train and test datasets. Stop words were removed during the process to filter out common words that do not provide meaningful information.

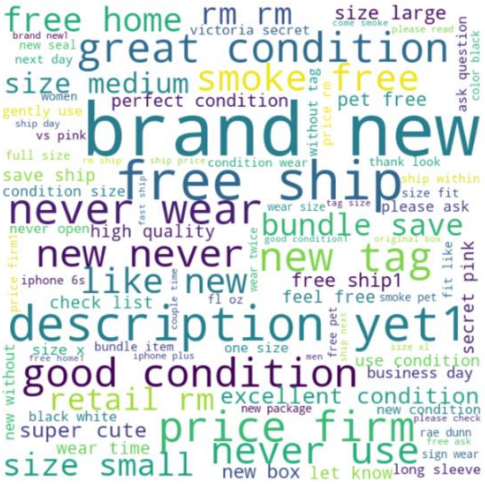

Figure 6.1: Word Cloud of item description

This figure displays the word cloud generated from the train dataset, where the most frequent words are represented by larger font sizes. The words “new”, “brand” and “free ship” appear to be the most commonly used in the descriptions.

6.2.2 This figure displays the distribution of the original price column, which appears to be heavily skewed to the right, indicating a high concentration of lower-priced items.

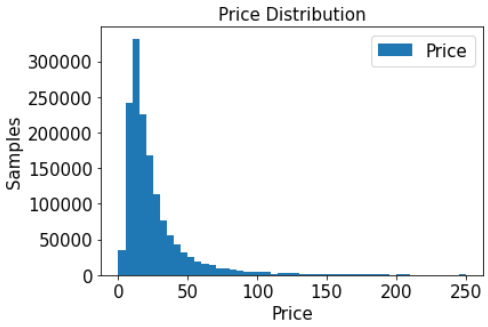

Figure 6.2 : Price Distribution

6.2.3 Whereas this figure shows the distribution of the log_price column, which appears to be more symmetric and centered around a mean value.

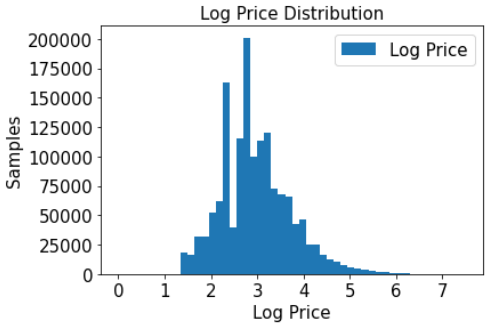

Figure 6.3 : Log Price Distribution

This transformation can help us better understand the distribution of prices in the train dataset, and can be used as a feature in our predictive models to improve their accuracy.

6.2.4 We created a bar chart to visualize the distribution of the top 20 brands in the train dataset. The brands were ranked based on the number of items associated with each brand, and the top brand was found to be pink and nike, with over 500,000 items in the dataset.

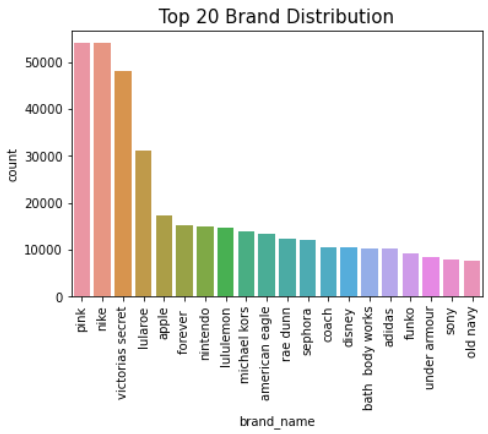

Figure 6.4 : Top 20 Brand Distribution

The chart shows that the distribution of brands is highly skewed, with a few brands having a large number of items while most of the brands have a small number of items.

6.2.5 We created a bar chart to visualize the top 20 most expensive brands in the train dataset. The brands were ranked based on the mean price of their items, and the top brand was found to be demdaco, with an average price of over $450. The second and third most expensive brands were proenza schoulder and auto meter, respectively.

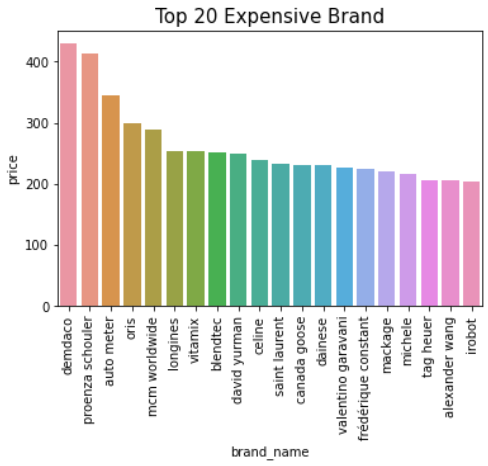

Figure 6.5 : Top 20 Expensive Brand Distribution

The chart shows that the most expensive brands have an average price of over $350, while most of the brands have an average price of under $250.

6.2.6 We created a bar chart to visualize the distribution of items across the sub-category 1 level in the train dataset.

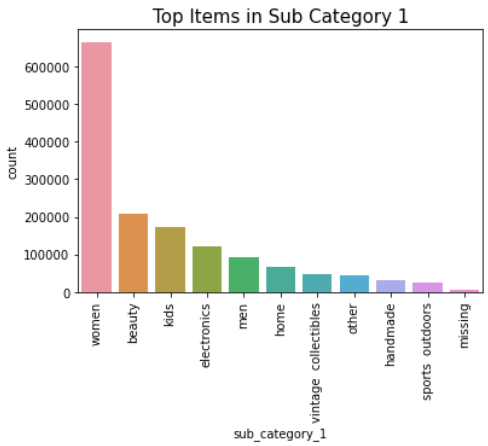

Figure 6.6 : Top Items in Sub Category 1

This figure shows that the women sub-category has the highest count of items, with over 630,000 items, followed by the beauty sub-category with over 200,000 items.

6.2.7 Similarly, we created a bar chart to visualize the distribution of items across the sub-category 2 and sub-category 3 level in the train dataset.

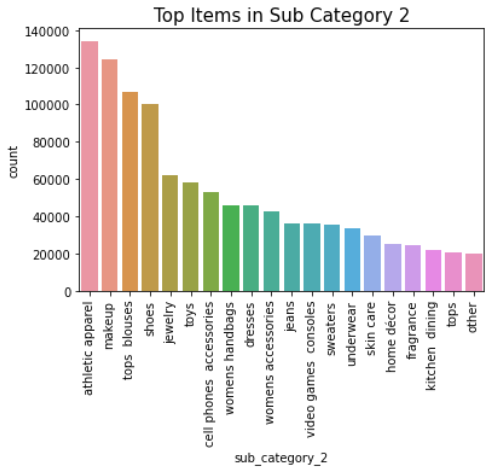

Figure 6.7 : Top Items in Sub Category 2

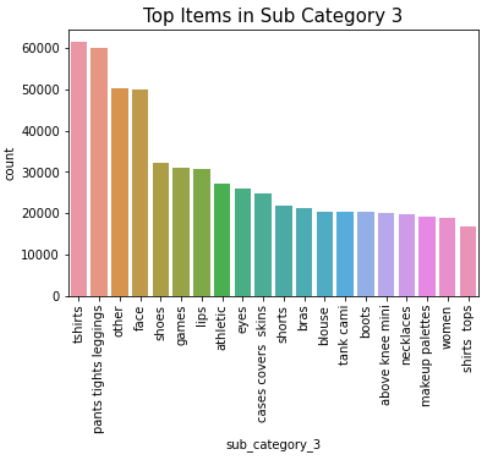

Figure 6.8 : Top Items in Sub Category 3

6.2.8 We analyzed the item_condition_id variable in the train dataset and created a bar chart to visualize the frequency count of each value. The bar chart revealed that the majority of items in the dataset have a relatively good condition, with item_condition_id values of 1,2 or 3 being the most frequent. In contrast, items in poor condition (with item_condition_id values of 4 or 5) are relatively rare in the dataset.

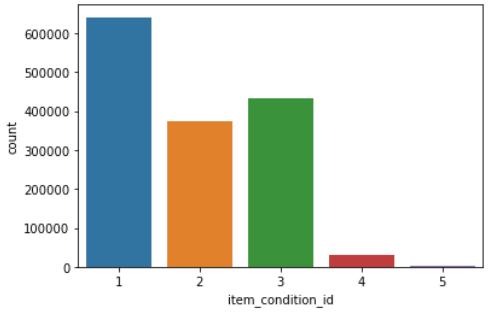

Figure 6.9 : Item Condition Id

6.2.9 We analyzed the name variable in the train dataset and created a horizontal bar chart to visualize the top 20 most common product names. The chart revealed that the most frequent product name in the dataset was bundle with over 3,100 occurrences, followed by lularoe tc leggings with over 1200 occurrences each. Other popular product names included lularoe os leggings, reserved, coach purse, on hold, and miss me jeans.

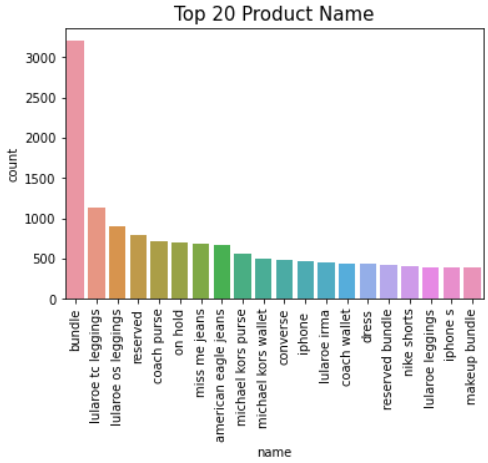

Figure 6.10 : Top 20 Product Name

6.2.10 We analyzed the shipping variable in the train dataset and created a bar chart to visualize the frequency count of each value. The bar chart revealed that the majority of items in the dataset have the shipping cost paid by the seller, while a smaller number of items have the shipping cost paid by the buyer.

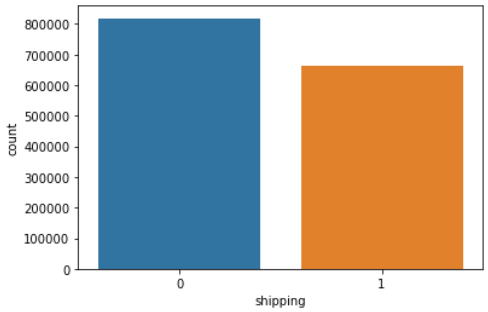

Figure 6.11 : Shipping

6.2.11 We investigated the relationship between the log price of products and their shipping type in the train dataset. The plot shows two bar graph for product prices - one for products with shipping cost paid by the seller, and another for products with shipping cost paid by the buyer.

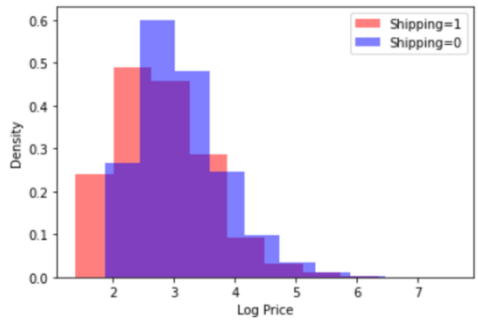

Figure 6.12 : Shipping Product Price vs Non Shipping Product Price

The overlaid graph revealed that, on average, products with the shipping cost paid by the seller tend to have slightly higher prices than products with the shipping cost paid by the buyer. However, the distribution of prices for products with the shipping cost

paid by the seller is more spread out and has a longer right tail, indicating that there are more expensive products in this category.

6.2.12 We investigated the relationship between the log price of products and their brand type (branded vs unbranded) in the train dataset. The plot shows two bar graph for product prices - one for products with known brands, and another for products with unknown brands.

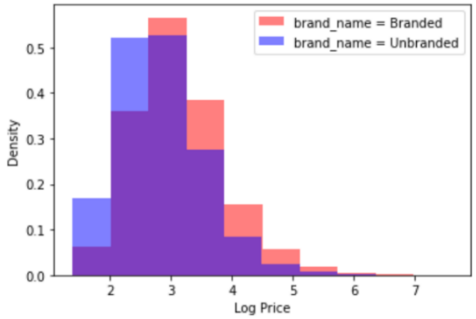

Figure 6.13 : Branded Product Price vs Unbranded Product Price

The overlaid graphs revealed that products with known brands tend to have slightly higher prices on average than products with unknown brands. The distribution of prices for products with unknown brands is more spread out and has a longer right tail, indicating that there are more expensive products in this category.

Chapter – 7

Data analysis and results

7.1 Sentiment Analysis and Label Encoding

7.1.1 In order to prepare our data for machine learning models, we used label encoding to transform categorical variables into numerical ones. Specifically, we used the LabelEncoder function from the preprocessing module of the scikit-learn library to transform the following columns of the merge datasets:

- brand_name
- sub_category_1
- sub_category_2
- sub_category_3
- name

7.1.2 Than we performed sentiment analysis on the item descriptions using the TextBlob library and the SentimentIntensityAnalyzer from the NLTK library. The polarity and subjectivity scores were calculated for each item description. Additionally, we classified the descriptions as positive, negative, or neutral based on the presence of positive or negative sentiment using the sentiment scores. This analysis will be used to understand the relationship between the sentiment of the product description and the product price and than we dropped the item_description column.

7.1.3 We are splitting the combined dataset merge into train and test datasets again like we did earlier and the predictors (independent variables) were stored in a variable called x, and the target variable (dependent variable) was stored in a variable called y, i.e. log_price. The data was then randomly split into training and testing sets using the train_test_split function from the sklearn library.

7.1.4 Once the label encoding and sentiment analysis is done, our training dataset looks like this:

	name	item_condition_id	brand_name	price	shipping	log_price	sub_category_1	sub_category_2	sub_category_3	polarity	subjectivity	description
0	947456	3	4940	10.0	1	2.397895	5	102	775	0.000000	0.000000	0
1	1226767	3	3943	52.0	0	3.970292	1	30	215	0.600000	0.725000	1
2	190573	1	4646	10.0	1	2.397895	10	103	97	0.136250	0.422500	1
3	768605	1	4940	35.0	1	3.583519	3	55	410	0.232121	0.598182	1
4	63283	1	4940	44.0	0	3.806662	10	58	543	0.100000	0.400000	0

Table 7.1: Training dataset

7.2 Models

7.2.1 The models that we applied to predict the price were:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Gradient Boosting Regressor
- Random Forest Regressor
- Stochastic Gradient Descent (SGD) Regressor

7.2.2 We trained each model on a training dataset and made predictions on a test dataset. We compared the performance of several regression models on a test dataset using the following evaluation metrics: R-squared, MSE, RMSE, MAE, and Adjusted R-squared.

	R-squared	MSE	RMSE	MAE	Adjusted R-squared
Random Forest Regressor	0.49	0.28	0.53	0.39	0.49
Gradient Boosting Regressor model	0.31	0.38	0.62	0.48	0.31
Linear Regression	0.08	0.51	0.72	0.56	0.08
Ridge Regression	0.08	0.51	0.72	0.56	0.08
SGD Model	0.08	0.51	0.72	0.56	0.08
Lasso Regression	0.03	0.54	0.74	0.57	0.03

Table 7.2: Model Evaluation 1

7.2.3 Further analysis is necessary to identify additional relevant features that could improve the accuracy of the models. So we will use the copy of merge dataset that we created earlier:

	name	item_condition_id	brand_name	price	shipping	item_description	log_price	is_train	sub_category_1	sub_category_2	sub_category_3
0	MLB Cincinnati Reds T Shirt Size XL	3	NaN	10.0	1	No description yet	2.397895	1	Men	Tops	T-shirts
1	Razer BlackWidow Chroma Keyboard	3	Razer	52.0	0	This keyboard is in great condition and works ...	3.970292	1	Electronics	Computers & Tablets	Components & Parts
2	AVA-VIV Blouse	1	Target	10.0	1	Adorable top with a hint of lace and a key hol...	2.397895	1	Women	Tops & Blouses	Blouse
3	Leather Horse Statues	1	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...	3.583519	1	Home	Home Décor	Home Décor Accents
4	24K GOLD plated rose	1	NaN	44.0	0	Complete with certificate of authenticity	3.806662	1	Women	Jewelry	Necklaces

Table 7.3: Merged Dataset

7.3 Feature Engineering

7.3.1 We performed feature engineering to enhance the predictive power of our model. This involved creating new variables based on the existing ones in our dataset. Here is a brief description of the features created:

- **name_first:** first word in the name column
- **name_first_count:** count of how many times a name_first appears in the dataset
- **sub_category_1_count:** count of how many times a sub_category_1 appears in the dataset
- **sub_category_2_count:** count of how many times a sub_category_2 appears in the dataset
- **sub_category_3_count:** count of how many times a sub_category_3 appears in the dataset
- **brand_name_count:** count of how many times a brand_name appears in the dataset
- **name_lower_count:** count of lowercase letters in the name column
- **name_upper_count:** count of uppercase letters in the name column
- **description_lower_count:** count of lowercase letters in the item_description column
- **description_upper_count:** count of uppercase letters in the item_description column
- **name_len:** length of the name column
- **des_len:** length of the item_description column
- **name_desc_len_ratio:** ratio of name length to item_description length
- **desc_word_count**: count of words in the item_description column
- **mean_des:** mean length of words in the item_description column
- **name_word_count:** count of words in the name column
- **mean_name:** mean length of words in the name column
- **desc_letters_per_word:** average number of letters per word in the item_description column
- **name_letters_per_word:** average number of letters per word in the name column
- **NameLowerRatio:** ratio of lowercase letters in the name column to the length of the name column
- **NameUpperRatio:** ratio of uppercase letters in the name column to the length of the name column
- **DescriptionLowerRatio:** ratio of lowercase letters in the item_description column to the length of the item_description column
- **DescriptionUpperRatio:** ratio of uppercase letters in the item_description column to the length of the item_description column
- **NamePunctCount:** count of punctuation marks in the name column
- **DescriptionPunctCount:** count of punctuation marks in the item_description column
- **NamePunctCountRatio:** ratio of punctuation marks in the name column to the count of words in the name column

- **DescriptionPunctCountRatio:** ratio of punctuation marks in the item_description column to the count of words in the item_description column
- **NameDigitCount:** count of digits in the name column
- **DescriptionDigitCount:** count of digits in the item_description column
- **NameDigitCountRatio:** ratio of digits in the name column to the count of words in the name column
- **DescriptionDigitCountRatio:** ratio of digits in the item_description column to the count of words in the item_description column
- **weird_characters_desc:** count of non-alphanumeric characters in the item_description column
- **weird_characters_name:** count of non-alphanumeric characters in the name column

These features were likely created to capture more information about the products in the dataset that may be predictive of their price.

7.3.2 Cleaned and transformed the dataset again in the same way like we did earlier as we have used the unprocessed dataset to extract new feature we updated the copy of merge dataset with this dataset and applied the same model again on dataset with extra feature.

7.3.3 The results of the evaluation are presented in the table below, sorted by R-squared in descending order.

	R-squared	MSE	RMSE	MAE	Adjusted R-squared
Random Forest Regressor	0.53	0.27	0.52	0.38	0.53
Gradient Boosting Regressor model	0.36	0.36	0.60	0.46	0.36
Linear Regression	0.13	0.49	0.70	0.54	0.13
Ridge Regression	0.13	0.49	0.70	0.54	0.13
SGD Model	0.13	0.49	0.70	0.54	0.13
Lasso Regression	0.07	0.52	0.72	0.56	0.07

Table 7.4 : Model Evaluation 2

Based on the results, we can see that the Random Forest Regressor performed the best among all, with the highest R-squared and lowest MSE and RMSE. The Linear Regression, Ridge Regression, Lasso Regression, and SGD Regressor models had lower R-squared scores and higher MSE and RMSE values. But the highest R-squared score obtained was only 0.53, which indicates that the models were unable to accurately predict the target variable. This is likely due to the fact that we did not have enough features to capture the complexity of the target variable.

7.3.4 Some changes are necessary that could improve the accuracy of the models. So we will use the copy of merge dataset that we created earlier, and will try some new models:

	name	item_condition_id	brand_name	price	shipping	item_description	log_price	is_train	sub_category_1	sub_category_2	...	DescriptionPunctCountRatio	NameDigitCount	DescriptionDigitCount	NameDigitCountRatio	DescriptionDigitCountRatio
0	mlb cincinnati reds t shirt size xl	3	unbranded	10.0	1	description yet	2.397895	1	men	tops	...	0.000000	0	0	0.0	0.000000
1	razer blackwidow chroma keyboard	3	razer	52.0	0	keyboard great condition work like come box ...	3.970292	1	electronics	computers & tablets	...	0.083333	0	0	0.0	0.000000
2	ava-viv blouse	1	target	10.0	1	adorable top hint lace key hole back ! pale pi...	2.397895	1	women	tops & blouses	...	0.103448	0	2	0.0	0.068966
3	leather horse statues	1	unbranded	35.0	1	new tag . leather horse . retail [rm] each	3.583519	1	home	home décor	...	0.281250	0	0	0.0	0.000000
4	24k gold plated rose	1	unbranded	44.0	0	complete certificate authenticity	3.806662	1	women	jewelry	...	0.000000	2	0	0.5	0.000000

5 rows × 47 columns

Table 7.5: Merged Dataset with extra Features

7.4 Tfidf Vectorizer and Label Binarizer

7.4.1 In order to prepare our data for machine learning models, we used label encoding to transform categorical variables into numerical ones in our earlier steps but that's not working that good so now we will now use label binarizer for some columns, that are:

- brand_name
- sub_category_1
- sub_category_2
- sub_category_3

7.4.2 We are using the TfidfVectorizer to transform the item_description column of the dataset. Unlike earlier, where we only performed sentiment analysis on this column, we are now using the entire column as a feature for the model. The TfidfVectorizer is used to convert text data into numerical feature vectors, which are then used as input for the machine learning model. We are using the max_features parameter to limit the number of features to 300,000.

7.4.3 Now we will drop some columns that we have already transformed and that are no longer needed in their original form: brand_name, sub_category_1, sub_category_2, sub_category_3, item_description, price, log_price.

7.4.4 Next we will convert our data in standard form using MinMaxScaler and we converted the train and test dataframes into sparse matrices using csr_matrix(). Then, we concatenated the label-binarized columns, the tf-idf transformed item_description column, and the sparse matrices horizontally using hstack() from the same module to create the final sparse training and testing matrices.

7.4.5 Finally we will split the data as we did earlier to apply machine learning models.

7.5 Different Models

7.5.1 The models that we applied to predict the price were:

- Ridge Regression
- FTRL
- FM_FTRL

We trained each model on a training dataset and made predictions on a test dataset. We compared the performance of several regression models on a test dataset using the following evaluation metrics: R-squared, MSE, RMSE, MAE, and Adjusted R-squared.

7.5.2 In order to improve the model performance, we have tried several new models. We have used Ridge Regression, FTRL and FM_FTRL. After evaluating the performance of each model, we have accepted the FM_FTRL model as it has the highest R-squared value of 0.60. The list of models used and their corresponding R-squared values are shown in the table below:

	R-squared	MSE	RMSE	MAE	Adjusted R-squared
FM_FTRL	0.60	0.22	0.47	0.35	0.60
Ridge	0.53	0.26	0.51	0.39	0.53
FTRL	0.51	0.27	0.52	0.40	0.51

Table 7.6 : Model Evaluation 3

Note that this is an improvement over our earlier approach, where we only used a sentiment analysis approach on the item_description column and were doing labelencoding on all categorical columns.

Chapter-8

Final Prediction and Conclusion

8.1 We used FM_FTRL model to make predictions on the unseen test data, and obtained the predicted logarithm of prices and then we converted it to normal price.

test_id	name	item_condition_id	category_name	brand_name	shipping	item_description	price
0	Breast cancer "I fight like a girl" ring	1	Women/Jewelry/Rings	NaN	1	Size 7	11.685978
1	25 pcs NEW 7.5"x12" Kraft Bubble Mailers	1	Other/Office supplies/Shipping Supplies	NaN	1	25 pcs NEW 7.5"x12" Kraft Bubble Mailers Lined...	14.353110
2	Coach bag	1	Vintage & Collectibles/Bags and Purses/Handbag	Coach	1	Brand new coach bag. Bought for [rm] at a Coac...	42.691685
3	Floral Kimono	2	Women/Sweaters/Cardigan	NaN	0	-floral kimono -never worn -lightweight and pe...	15.049997
4	Life after Death	3	Other/Books/Religion & Spirituality	NaN	1	Rediscovering life after the loss of a loved o...	11.945440

Table 8 : Final Test Dataset

8.2 After predicting the price, when we visualize price distribution of traing and test dataset, we found that,

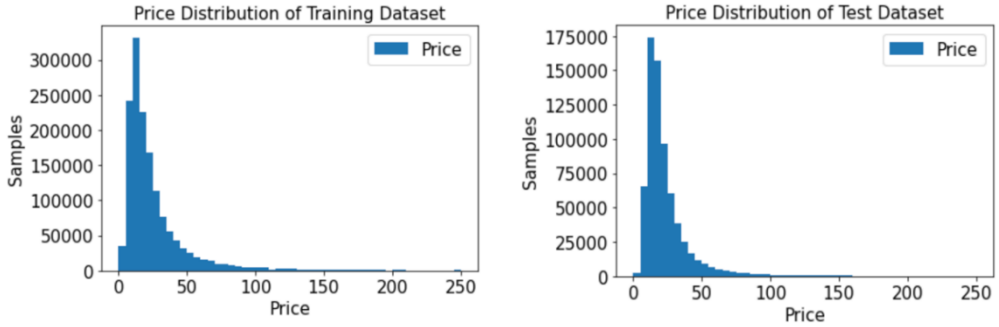

Figure 8.1 : Price Distribution (Training, Final Test)

the distributions of actual prices and predicted prices are similar, although there are some differences. In both cases, there is a peak around 10-20, indicating that many properties are priced in this range. There are also a smaller number of properties that are priced much higher, in the range of 100-200. However, the distribution of actual prices has a longer tail than the predicted prices, indicating that there are more properties with very high prices in the actual data. Overall, the predicted prices seem to capture the general shape of the price distribution, but may not be as accurate in capturing the extreme values.

8.3 Based on the analysis performed, we have developed a model to predict the prices of items based on their descriptions, category names, and other relevant features. We have used a machine learning model, namely FM_FTRL to achieve the best possible results with R2 of 0.60.

We have also compared the distribution of the price of both the training and test sets and observed that they are quite similar. This indicates that our model has not overfit to the training data and should perform well on unseen test data and saved them to a final test file for submission. Overall, the project has been successful in achieving the objective of predicting item prices based on their descriptions and other features.

Annexure

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import wordcloud
from wordcloud import WordCloud, STOPWORDS
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.preprocessing import LabelEncoder
from textblob import TextBlob
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack, csr_matrix
from sklearn.preprocessing import LabelBinarizer
import re
import wordbatch
from wordbatch.models import FM_FTRL
from wordbatch.models import FTRL

train = pd.read_csv("train.tsv", sep = '\t')
train = train.set_index(train.columns[0])
train.shape

train.head()

test = pd.read_csv("test.tsv", sep = '\t')
test = test.set_index(test.columns[0])
test.shape

test.head()

train = train[train.price > 0].reset_index(drop=True)
train.shape

train['log_price'] = np.log1p(train['price'])
```

```
train['is_train'] = 1
test['is_train'] = 0

merge = pd.concat([train, test])
merge.shape

merge.head()

merge.tail()

new = merge["category_name"].str.split("/", n = 2, expand = True)
merge["sub_category_1"] = new[0]
merge["sub_category_2"] = new[1]
merge["sub_category_3"] = new[2]
merge = merge.drop(['category_name'], axis =1)

merge.head()

merge.isnull().sum()

merge['item_description'].fillna(value='missing',inplace=True)
merge['sub_category_1'].fillna(value='missing',inplace=True)
merge['sub_category_2'].fillna(value='missing',inplace=True)
merge['sub_category_3'].fillna(value='missing',inplace=True)
merge['brand_name'].fillna(value='unbranded',inplace=True)

merge.isnull().sum()

merge_copy = merge.copy()

merge["sub_category_1"] = merge["sub_category_1"].str.lower()
merge["sub_category_1"] = merge["sub_category_1"].str.replace('[!"#$%&\'()*+,-
./:;<=>?@[\\]^_`{|}~]','')
merge["sub_category_1"] = merge["sub_category_1"].str.replace('\d+','')

merge["sub_category_2"] = merge["sub_category_2"].str.lower()
merge["sub_category_2"] = merge["sub_category_2"].str.replace('[!"#$%&\'()*+,-
./:;<=>?@[\\]^_`{|}~]','')
merge["sub_category_2"] = merge["sub_category_2"].str.replace('\d+','')

merge["sub_category_3"] = merge["sub_category_3"].str.lower()
merge["sub_category_3"] = merge["sub_category_3"].str.replace('[!"#$%&\'()*+,-
./:;<=>?@[\\]^_`{|}~]','')
merge["sub_category_3"] = merge["sub_category_3"].str.replace('\d+','')

merge["brand_name"] = merge["brand_name"].str.lower()
merge["brand_name"] = merge["brand_name"].str.replace('[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]','')
merge["brand_name"] = merge["brand_name"].str.replace('\d+','')

merge["name"] = merge["name"].str.lower()
merge["name"] = merge["name"].str.replace('[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]','')
merge["name"] = merge["name"].str.replace('\d+','')

merge["item_description"] = merge["item_description"].str.lower()
merge["item_description"] =
```

```
merge['item_description'].str.replace('[!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~]','')
merge["item_description"] = merge['item_description'].str.replace('\d+','')

stop_words = stopwords.words('english')
merge["item_description"] = merge["item_description"].apply(lambda x: ' '.join([word for word in
x.split() if word not in (stop_words)]))

df = pd.DataFrame(merge["item_description"])
df['tokenized_sents'] = df.apply(lambda row: nltk.word_tokenize(row['item_description']), axis=1)
merge["item_description"] = df['tokenized_sents']

lemmatizer = WordNetLemmatizer()
def lemmatize_words(item_description):
   words = item_description
   words = [lemmatizer.lemmatize(word,pos='v') for word in words]
   return ' '.join(words)
merge['item_description'] = merge['item_description'].apply(lemmatize_words)

test = merge.loc[merge['is_train'] == 0]
train = merge.loc[merge['is_train'] == 1]

stopwords=set(stop_words)
word_cloud = WordCloud(width = 600, height = 600,background_color ='white',
stopwords=stopwords,min_font_size = 10).generate("1 ".join(train['item_description']))
plt.figure(figsize = (15, 10))
plt.imshow(word_cloud)
plt.axis('off')
plt.show()

plt.hist(train['price'], bins=50, range=[0,250], label='Price')
plt.title('Price Distribution', fontsize=15)
plt.xlabel('Price', fontsize=15)
plt.ylabel('Samples', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(fontsize=15)
plt.show()

plt.hist(train['log_price'], bins=50, range=[0,7.5], label='Log Price')
plt.title('Log Price Distribution', fontsize=15)
plt.xlabel('Log Price', fontsize=15)
plt.ylabel('Samples', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(fontsize=15)
plt.show()

b20 = train['brand_name'].value_counts()[1:21].reset_index()
ax = sns.barplot(x="brand_name", y="count", data=b20)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
ax.set_title('Top 20 Brand Distribution', fontsize=15)
plt.show()
```

```
top20_brand = train.groupby('brand_name', axis=0)['price'].mean()
df_expPrice = pd.DataFrame(top20_brand.sort_values(ascending=False)[0:20].reset_index())
ax = sns.barplot(x="brand_name", y="price", data=df_expPrice)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
ax.set_title('Top 20 Expensive Brand', fontsize=15)
plt.show()

s1 = train['sub_category_1'].value_counts().reset_index()
ax = sns.barplot(x="sub_category_1", y="count", data=s1)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
ax.set_title('Top Items in Sub Category 1', fontsize=15)
plt.show()

s1 = train['sub_category_2'].value_counts()[0:20].reset_index()
ax = sns.barplot(x="sub_category_2", y="count", data=s1)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
ax.set_title('Top Items in Sub Category 2', fontsize=15)
plt.show()

s1 = train['sub_category_3'].value_counts()[0:20].reset_index()
ax = sns.barplot(x="sub_category_3", y="count", data=s1)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
ax.set_title('Top Items in Sub Category 3', fontsize=15)
plt.show()

sns.countplot(x="item_condition_id", data=train)

n20 = train['name'].value_counts()[0:20].reset_index()
ax = sns.barplot(x="name", y="count", data=n20)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
ax.set_title('Top 20 Product Name', fontsize=15)
plt.show()

sns.countplot(x="shipping", data=train)

prices_ship1 = [p for p, s in zip(train['log_price'], train['shipping']) if s == 1]
prices_ship0 = [p for p, s in zip(train['log_price'], train['shipping']) if s == 0]
plt.hist(prices_ship1, color='r', alpha=0.5, density=True, label='Shipping=1')
plt.hist(prices_ship0, color='b', alpha=0.5, density=True, label='Shipping=0')
plt.xlabel('Log Price')
plt.ylabel('Density')
plt.legend()
plt.show()

prices_branded = [p for p, s in zip(train['log_price'], train['brand_name']) if s != 'unbranded']
prices_unbranded = [p for p, s in zip(train['log_price'], train['brand_name']) if s == 'unbranded']
plt.hist(prices_branded, color='r', alpha=0.5, density=True, label='brand_name = Branded')
plt.hist(prices_unbranded, color='b', alpha=0.5, density=True, label='brand_name = Unbranded')
plt.xlabel('Log Price')
plt.ylabel('Density')
plt.legend()
plt.show()
```

```
plt.figure(figsize=(15,5))
sns.boxplot(x=train['item_condition_id'],y=np.log(train['price']+1))
plt.title('Box Plot of item condition id VS Product price')
plt.ylabel('log(price+1)')
plt.show()

plt.figure(figsize=(7,10))
sns.boxplot(y=train['sub_category_1'],x=np.log(train['price']+1))
plt.title('Box Plot of Sub_Category_1 VS Product Price')
plt.show()

merge[['polarity', 'subjectivity']] = merge['item_description'].apply(lambda x:
pd.Series(TextBlob(x).sentiment))
analyzer = SentimentIntensityAnalyzer()
sentiment_scores = merge['item_description'].apply(analyzer.polarity_scores)
merge['description'] = sentiment_scores.apply(lambda x: -1 if x['neg'] > x['pos'] else (1 if x['pos'] >
x['neg'] else 0))
merge = merge.drop(['item_description'], axis=1)

label_encoder = preprocessing.LabelEncoder()
merge['brand_name'] = label_encoder.fit_transform(merge['brand_name'])
merge['sub_category_1'] = label_encoder.fit_transform(merge['sub_category_1'])
merge['sub_category_2'] = label_encoder.fit_transform(merge['sub_category_2'])
merge['sub_category_3'] = label_encoder.fit_transform(merge['sub_category_3'])
merge['name'] = label_encoder.fit_transform(merge['name'])

test = merge.loc[merge['is_train'] == 0]
train = merge.loc[merge['is_train'] == 1]

test = test.drop(['is_train'], axis=1)
train = train.drop(['is_train'], axis=1)

train.head()

x = train.drop(['log_price', 'price'], axis=1)
y = train.loc[:, train.columns == 'log_price']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

lr = LinearRegression()
lr.fit(x_train, y_train)
lr_pred = lr.predict(x_test)

alpha = 1.0
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(x_train, y_train)
ridge_pred = ridge_model.predict(x_test)

lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)
lasso_pred = lasso.predict(x_test)

gbr_regressor = GradientBoostingRegressor(n_estimators=100, max_depth=4, random_state=42)
gbr_regressor.fit(x_train, y_train)
gbr_pred = gbr_regressor.predict(x_test)
```

```
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(x_train, y_train)
rfr_pred = rfr.predict(x_test)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
sgd_model = SGDRegressor(max_iter=100000, tol=1e-3, random_state=42)
sgd_model.fit(x_train_scaled, y_train)
sgd_pred = sgd_model.predict(x_test_scaled)

models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression', 'Gradient Boosting Regressor
model', 'Random Forest Regressor','SGD Model']
y_preds_list = [lr_pred, ridge_pred, lasso_pred, gbr_pred, rfr_pred, sgd_pred]

def evaluate_regression_models(y_test, y_preds_list):
    num_models = len(y_preds_list)
    n = len(y_test)
    results = {}
    for i in range(num_models):
        model_name = models[i]
        y_pred = y_preds_list[i]
        r2 = r2_score(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)
        mae = mean_absolute_error(y_test, y_pred)
        adj_r2 = 1 - ((1 - r2) * (n - 1)) / (n - 2 - 1)
        results[model_name] = {'R-squared': '{:.2f}'.format(r2),
                               'MSE': '{:.2f}'.format(mse),
                               'RMSE': '{:.2f}'.format(rmse),
                               'MAE': '{:.2f}'.format(mae),
                               'Adjusted R-squared': '{:.2f}'.format(adj_r2)}
    return results

results = pd.DataFrame(evaluate_regression_models(y_test, y_preds_list)).transpose()

sorted_results_1 = results.sort_values(by='R-squared', ascending=False)
sorted_results_1

merge = merge_copy()

merge['name_first'] = merge["name"].str.split(" ", expand = True)[0]
merge['name_first_count'] = merge.groupby('name_first')['name_first'].transform('count')
merge['sub_category_1_count'] =
merge.groupby('sub_category_1')['sub_category_1'].transform('count')
merge['sub_category_2_count'] =
merge.groupby('sub_category_2')['sub_category_2'].transform('count')
merge['sub_category_3_count'] =
merge.groupby('sub_category_3')['sub_category_3'].transform('count')
merge['brand_name_count'] = merge.groupby('brand_name')['brand_name'].transform('count')
merge['name_lower_count'] = merge.name.str.count('[a-z]')
merge['name_upper_count'] = merge.name.str.count('[A-Z]')
merge['description_lower_count'] = merge.item_description.str.count('[a-z]')
```

```
merge['description_upper_count'] = merge.item_description.str.count('[A-Z]')
merge['name_len'] = merge['name'].apply(lambda x: len(x))
merge['des_len'] = merge['item_description'].apply(lambda x: len(x))
merge['name_desc_len_ratio'] = merge['name_len']/merge['des_len']
merge['desc_word_count'] = merge['item_description'].apply(lambda x: len(x.split()))
merge['mean_des'] = merge['item_description'].apply(lambda x: float(len(x.split())) / len(x)) * 10
merge['name_word_count'] = merge['name'].apply(lambda x: len(x.split()))
merge['mean_name'] = merge['name'].apply(lambda x: float(len(x.split())) / len(x))  * 10
merge['desc_letters_per_word'] = merge['des_len'] / merge['desc_word_count']
merge['name_letters_per_word'] = merge['name_len'] / merge['name_word_count']
merge['NameLowerRatio'] = merge['name_lower_count'] / merge['name_len']
merge['NameUpperRatio'] = merge['name_upper_count'] / merge['name_len']
merge['DescriptionLowerRatio'] = merge['description_lower_count'] / merge['des_len']
merge['DescriptionUpperRatio'] = merge['description_upper_count'] / merge['des_len']
merge['NamePunctCount'] = merge.name.str.count('[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]')
merge['DescriptionPunctCount'] =
merge.item_description.str.count('[!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~]')
merge['NamePunctCountRatio'] = merge['NamePunctCount'] / merge['name_word_count']
merge['DescriptionPunctCountRatio'] = merge['DescriptionPunctCount'] /
merge['desc_word_count']
merge['NameDigitCount'] = merge.name.str.count('[0-9]')
merge['DescriptionDigitCount'] = merge.item_description.str.count('[0-9]')
merge['NameDigitCountRatio'] = merge['NameDigitCount'] / merge['name_word_count']
merge['DescriptionDigitCountRatio'] = merge['DescriptionDigitCount']/merge['desc_word_count']
non_alphanumpunct = re.compile(u'[^A-Za-z0-9\.?!,; \(\)\[\]\'\"\$]+')
merge['weird_characters_desc'] = merge['item_description'].str.count(non_alphanumpunct)
merge['weird_characters_name'] = merge['name'].str.count(non_alphanumpunct)

merge["sub_category_1"] = merge["sub_category_1"].str.lower()
merge["sub_category_1"] =
merge["sub_category_1"].str.replace('[!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~]','')
merge["sub_category_1"] = merge["sub_category_1"].str.replace('\d+','')

merge["sub_category_2"] = merge["sub_category_2"].str.lower()
merge["sub_category_2"] =
merge["sub_category_2"].str.replace('[!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~]','')
merge["sub_category_2"] = merge["sub_category_2"].str.replace('\d+','')

merge["sub_category_3"] = merge["sub_category_3"].str.lower()
merge["sub_category_3"] =
merge["sub_category_3"].str.replace('[!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~]','')
merge["sub_category_3"] = merge["sub_category_3"].str.replace('\d+','')

merge["brand_name"] = merge["brand_name"].str.lower()
merge["brand_name"] = merge["brand_name"].str.replace('[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]','')
merge["brand_name"] = merge["brand_name"].str.replace('\d+','')

merge["name"] = merge["name"].str.lower()
merge["name"] = merge["name"].str.replace('[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]','')
merge["name"] = merge["name"].str.replace('\d+','')
```

```
merge["name_first"] = merge["name_first"].str.lower()
merge["name_first"] = merge["name_first"].str.replace('[!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]','')
merge["name_first"] = merge["name_first"].str.replace('\d+','')

merge["item_description"] = merge["item_description"].str.lower()
merge["item_description"] =
merge['item_description'].str.replace('[!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~]','')
merge["item_description"] = merge['item_description'].str.replace('\d+','')

stop_words = stopwords.words('english')
merge["item_description"] = merge["item_description"].apply(lambda x: ' '.join([word for word in
x.split() if word not in (stop_words)]))

df = pd.DataFrame(merge["item_description"])
df['tokenized_sents'] = df.apply(lambda row: nltk.word_tokenize(row['item_description']), axis=1)
merge["item_description"] = df['tokenized_sents']

lemmatizer = WordNetLemmatizer()
def lemmatize_words(item_description):
   words = item_description
   words = [lemmatizer.lemmatize(word,pos='v') for word in words]
   return ' '.join(words)
merge['item_description'] = merge['item_description'].apply(lemmatize_words)

merge[['polarity', 'subjectivity']] = merge['item_description'].apply(lambda x:
pd.Series(TextBlob(x).sentiment))
analyzer = SentimentIntensityAnalyzer()
sentiment_scores = merge['item_description'].apply(analyzer.polarity_scores)
merge['description'] = sentiment_scores.apply(lambda x: -1 if x['neg'] > x['pos'] else (1 if x['pos'] >
x['neg'] else 0))

merge_copy = merge.copy()

merge = merge.drop(['item_description'], axis=1)

label_encoder = preprocessing.LabelEncoder()
merge['brand_name'] = label_encoder.fit_transform(merge['brand_name'])
merge['sub_category_1'] = label_encoder.fit_transform(merge['sub_category_1'])
merge['sub_category_2'] = label_encoder.fit_transform(merge['sub_category_2'])
merge['sub_category_3'] = label_encoder.fit_transform(merge['sub_category_3'])
merge['name'] = label_encoder.fit_transform(merge['name'])
merge['name_first'] = label_encoder.fit_transform(merge['name_first'])

test = merge.loc[merge['is_train'] == 0]
train = merge.loc[merge['is_train'] == 1]

test = test.drop(['is_train'], axis=1)
train = train.drop(['is_train'], axis=1)

train.head()

x = train.drop(['log_price', 'price'], axis=1)
y = train.loc[:, train.columns == 'log_price']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_pred = lr.predict(x_test)

alpha = 1.0
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(x_train, y_train)
ridge_pred = ridge_model.predict(x_test)

lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)
lasso_pred = lasso.predict(x_test)

gbr_regressor = GradientBoostingRegressor(n_estimators=100, max_depth=4, random_state=42)
gbr_regressor.fit(x_train, y_train)
gbr_pred = gbr_regressor.predict(x_test)

rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(x_train, y_train)
rfr_pred = rfr.predict(x_test)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
sgd_model = SGDRegressor(max_iter=100000, tol=1e-3, random_state=42)
sgd_model.fit(x_train_scaled, y_train)
sgd_pred = sgd_model.predict(x_test_scaled)

models = ['Linear Regression', 'Ridge Regression', 'Lasso Regression', 'Gradient Boosting Regressor
model', 'Random Forest Regressor','SGD Model']
y_preds_list = [lr_pred, ridge_pred, lasso_pred, gbr_pred, rfr_pred, sgd_pred]

results = pd.DataFrame(evaluate_regression_models(y_test, y_preds_list)).transpose()

sorted_results_2 = results.sort_values(by='R-squared', ascending=False)
sorted_results_2

merge = merge_copy
merge.head()

label_binarizer = LabelBinarizer(sparse_output=True)

brand_name = label_binarizer.fit_transform(merge['brand_name'])
train_brand_name = brand_name[:len(train)]
test_brand_name = brand_name[len(train):]

sub_category_1 = label_binarizer.fit_transform(merge['sub_category_1'])
train_sub_category_1 = sub_category_1[:len(train)]
test_sub_category_1 = sub_category_1[len(train):]

sub_category_2 = label_binarizer.fit_transform(merge['sub_category_2'])
train_sub_category_2 = sub_category_2[:len(train)]
test_sub_category_2 = sub_category_2[len(train):]
```

```
sub_category_3 = label_binarizer.fit_transform(merge['sub_category_3'])
train_sub_category_3 = sub_category_3[:len(train)]
test_sub_category_3 = sub_category_3[len(train):]

label_encoder = preprocessing.LabelEncoder()

merge['name'] = label_encoder.fit_transform(merge['name'])
merge['name_first'] = label_encoder.fit_transform(merge['name_first'])

tfidf = TfidfVectorizer(use_idf=False, max_features=300000)
tfidf.fit(merge['item_description'])
item_description = tfidf.transform(merge['item_description'])
train_item_description = item_description[:len(train)]
test_item_description = item_description[len(train):]

merge = merge.drop(['is_train', 'brand_name', 'sub_category_1', 'sub_category_2','sub_category_3',
            'item_description', 'price', 'log_price'], axis=1)

scaler = MinMaxScaler()
merge_normalized = scaler.fit_transform(merge)
train_normalized = merge_normalized[:len(train)]
test_normalized = merge_normalized[len(train):]

sparse_train = csr_matrix(train_normalized)
sparse_test = csr_matrix(test_normalized)

sparse_merge_train = hstack((train_brand_name, train_sub_category_1, train_sub_category_2,
train_sub_category_3,
                    train_item_description, sparse_train)).tocsr()
sparse_merge_test = hstack((test_brand_name, test_sub_category_1, test_sub_category_2,
test_sub_category_3,
                   test_item_description, sparse_test)).tocsr()

y = pd.Series(y['log_price'])

x_train, x_test, y_train, y_test = train_test_split(sparse_merge_train, y, test_size=0.2,
random_state=42)

alpha = 1.0
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(x_train, y_train)
ridge_pred = ridge_model.predict(x_test)

ftrl = FTRL(alpha = 0.01, beta = 0.1, L1 = 0.00001, L2 = 1.0, D = sparse_merge_train.shape[1], iters =
25, inv_link = "identity", threads = 1)
ftrl.fit(x_train, y_train)
ftrl_pred = ftrl.predict(x_test)

fm_ftrl = FM_FTRL(alpha=0.035, beta=0.001, L1=0.00001, L2=0.15, D=sparse_merge_train.shape[1],
            alpha_fm=0.05, L2_fm=0.0, init_fm=0.01, iters=30,
            D_fm=int(np.ceil(2 * np.sqrt(sparse_merge_train.shape[1]))), e_noise=0,
inv_link="identity", threads=4)
fm_ftrl.fit(x_train, y_train)
fm_ftrl_pred = fm_ftrl.predict(x_test)
```

```
models = ['Ridge', 'FTRL', 'FM_FTRL']
y_preds_list = [ridge_pred, ftrl_pred, fm_ftrl_pred]

results = pd.DataFrame(evaluate_regression_models(y_test, y_preds_list)).transpose()

sorted_results_3 = results.sort_values(by='R-squared', ascending=False)
sorted_results_3

fm_ftrl_pred_test = fm_ftrl.predict(sparse_merge_test)

fm_ftrl_pred_test_price = np.exp(fm_ftrl_pred_test)
test['price'] = fm_ftrl_pred_test_price

test = test.drop(['is_train'], axis=1)

test.head(10)

plt.hist(train['price'], bins=50, range=[0,250], label='Price')
plt.title('Price Distribution of Training Dataset', fontsize=15)
plt.xlabel('Price', fontsize=15)
plt.ylabel('Samples', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(fontsize=15)
plt.show()

plt.hist(test['price'], bins=50, range=[0,250], label='Price')
plt.title('Price Distribution of Test Dataset', fontsize=15)
plt.xlabel('Price', fontsize=15)
plt.ylabel('Samples', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(fontsize=15)
plt.show()

test.to_csv("final_test.tsv", sep = '\t')
```

Bibliography

- Machine Learning Using Python by Manaranjan Pradhan (Consultant, Indian Institute of Management, Bangalore) and U Dinesh Kumar (Professor, Indian Institute of Management, Bangalore)
- www.kaggle.com
- www.w3school.com
- www.geeksforgeeks.org
- www.stackoverflow.com