**PART 1)**

**First** install the necessary packages and the libraries.

**The second** part of the code is the scape of the data from the website.

This code scrapes data from a retail store website for a range of dates and stores the data in a pandas Data Frame. Here's a breakdown of the code:

This code scrapes data from a retail store website for a range of dates and stores the data in a pandas Data Frame. The code starts by defining a start date and an end date, and then creating a date range using the pandas date range function. The code then checks if it can access the first webpage to scrape data. It uses the requests library to make a GET request to the webpage URL, with a timeout of 15 seconds. It stores the response in a variable called response. The code then checks the status code of the response to see if the request was successful. If the status code is 200, it means the request was successful. If the request was successful, the code creates an empty Data Frame with columns named 'date', 'likes', 'dislikes', 'followers', and 'price'. The code then loops through each date in the date range, and for each date, it generates a webpage URL and uses the requests library to make a GET request to that URL. It then parses the HTML content of the response using the BeautifulSoup library, and extracts the values of the 'likes', 'dislikes', 'followers', and 'price' fields from the HTML using the find () method. The code then writes the date and the extracted values of 'likes', 'dislikes', 'followers', and 'price' fields into the Data Frame. Finally, the code converts the 'price', 'likes', 'dislikes', and 'followers' columns in the Data Frame to numeric types using the pandas to numeric () function. The code then saves the Data Frame to a CSV file named 'My Retail Store Website Data\_new.csv'.

**The third** part of the code is exploring the data. The code first prints the first three rows of the Data Frame. It then prints the summary statistics of the numerical features in the Data Frame. Next, it plots the distribution of each numerical feature in the Data Frame on a logarithmic scale. It then plots a bar chart of the correlation between each numerical feature and the target feature (price). Finally, it plots a heatmap of the correlation between all numerical columns in the Data Frame. The code is used to analyze data and identify patterns. It can be used to understand the distribution of data, the correlation between different features, and the relationship between features and the target variable.

**The fourth** part of code is the data preparation and the feature engineering. This sector has subsectors.

1. The code first creates a DataFrame of the date column in the original DataFrame. It then instantiates a Feature Engineering Object and fits and transforms the date DataFrame. The code then adds the extracted features to the original DataFrame and drops the date column. It then checks the datatypes of the DataFrame and visualizes and sorts all features based on correlation to the target. Finally, it visualizes a heatmap of the correlations between all columns. The code is used to extract features from the date column and add them to the original DataFrame. The code can be used to improve the accuracy of machine learning models by providing more features to the model.
2. The code splits the DataFrame into two sets, a training set and a testing set. The training set is used to train the machine learning model, and the testing set is used to evaluate the performance of the model. This is done to ensure that the model is not overfitting the training data.
3. The code applies a StandardScaler to the likes, dislikes, and follower’s columns, and leaves the remaining columns unchanged. This is done to prepare the data for machine learning. The code first fits the StandardScaler to the training data. This means that the StandardScaler learns the mean and standard deviation of the training data. It then transforms both the training and testing data using the StandardScaler. This ensures that the training and testing data are on the same scale.

**The fifth** part of the code is used to test a variety of regression models on a dataset. The first step is to create a dictionary of regressors, where each key is the name of a regressor, and the value is the regressor object itself. The regressors in the dictionary are created with a random state of 17, and the max\_iter parameter is increased for some regressors to improve fit.

Once the dictionary of regressors is created, the next step is to create a blank data frame for performance metrics. The data frame will have the following columns:

* model: The name of the regressor
* run\_time: The time it took to train the regressor.
* rmse: The root means squared error of the regressor on the test set.
* cross\_val\_rmse\_average: The average root means squared error of the regressor on 10 folds of the cross validation set.
* cross\_val\_rmse\_scores: A list of the root mean squared errors of the regressor on the 10 folds of the cross validation set.

The next step is to loop through the dictionary of regressors. For each regressor, the following steps are taken:

* The progress of the loop is printed.
* The start time is recorded.
* The regressor is instantiated.
* The regressor is trained on the training set.
* The regressor is used to predict the labels of the test set.
* The cross-validation scores of the regressor are calculated.
* The performance information of the regressor is stored in a dictionary.
* The performance information dictionary is added to the data frame.

Finally, the data frame is sorted by the RMSE column in ascending order. This shows the models with the lowest RMSE at the top of the data frame.

This code can be used to test a variety of regression models on a dataset and to compare their performance.

**The sixth** part of code is used to train a Gaussian process regressor on a dataset and to predict the labels of some new data. The first step is to import the GaussianProcessRegressor and RBF classes from scikit-learn.

Next, some data is generated. The data consists of three points, each with two features and one label.

The kernel is defined. The kernel is a function that defines the relationship between the features and the labels. In this case, the RBF kernel is used.

The Gaussian process regressor is created. The regressor is created with the following parameters:

* kernel: The kernel function
* alpha: The regularization parameter
* n\_restarts\_optimizer: The number of times to restart the optimizer.

The regressor is fit to the data. The regressor is fit using the fit () method.

The regressor is used to predict the labels of some new data. The predict () method is used to predict the labels of the new data. The return\_std parameter is set to True, so the standard deviation of the predictions is also returned.

The output of the code is the predicted labels and the standard deviation of the predictions. In this case, the predicted labels are [0.9, 1.9, 2.9] and the standard deviation of the predictions is [0.1, 0.1, 0.1].

This code can be used to train a Gaussian process regressor on a dataset and to predict the labels of some new data.

PART 2)

In the original data have pattern. The generating data has no pattern. Τhis changed in all of them figure.

The train with the original has not same RMSE and cross\_val\_rmse\_average, but the generator has the same values.

In the generate have more the error in relation to with the origin.

In the original data have very good Actual Price and Fitted Price, but in the generate have not fit good.