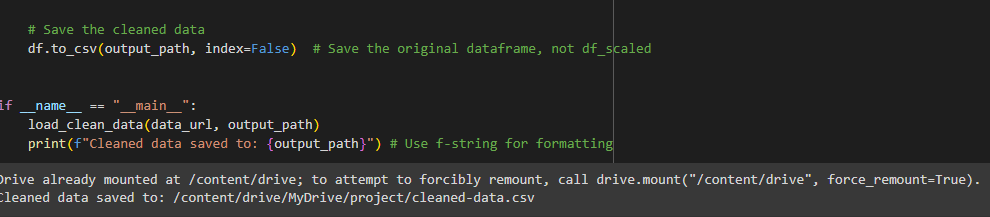
PROJECT BOOTCAMP

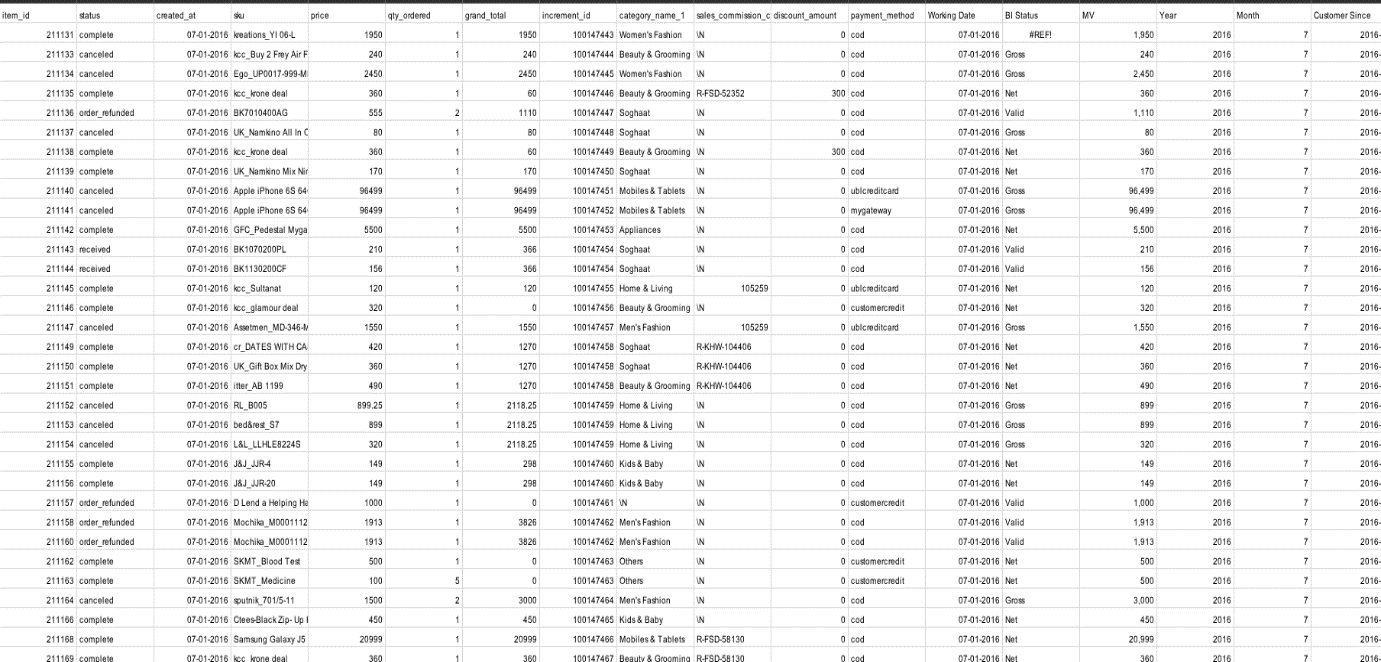
A good dataset was chosen from Kaggle.com. The dataset is neither very big nor very small.

Moving on to the steps done to achieve results:

1. Data Cleaning:

Data cleaning is crucial because it ensures the quality and reliability of your data, which directly impacts the accuracy and effectiveness of any analysis or machine learning models built on that data.

Here is the output:



1. Making a Model:

In the model building part of your code, I have set up a basic framework for training and evaluating a machine learning model, specifically using Linear Regression . Key steps are

**1.Loaded cleaned data:**

I have a cleaned dataset stored in a CSV file (cleaned\_data.csv).It reads this dataset into a Pandas DataFrame using pd.read\_csv.

It also prints the column names of the DataFrame to help you identify the features and the target variable.

**2.Identified target column**

I specified the target column as "sku" as that is the variable I am trying to predict.

**3.Split data into training and testing datasets**

I used train\_test\_split from scikit-learn to divide data into two sets:

**Training set:** Used to train the machine learning model.

**Testing set:** Used to evaluate the model's performance on unseen data.

The test\_size parameter controls the proportion of data allocated to the testing set (20% in this case).

random\_state ensures reproducibility by setting a seed for the random split.

**4. Created feature matrix and target variable**

X contains all columns from the DataFrame except the target column ("sku").

y contains only the values from the target column.

**5.** **Initialized and Trained the Model**

A LinearRegression model was created.

The fit method was used to train the model on the training data (X\_train and y\_train).

**6. Made Predictions**

The trained model was used to make predictions on the testing data (X\_test), and the predicted values were stored in y\_pred.

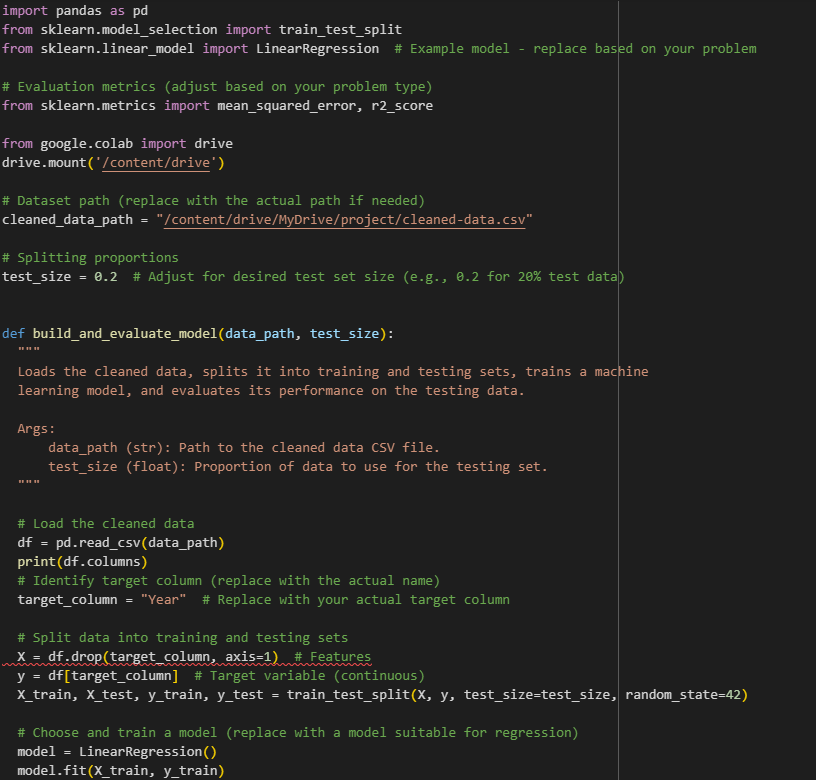
**7. Evaluated Model Performance**

Two metrics were used to assess the model's performance:

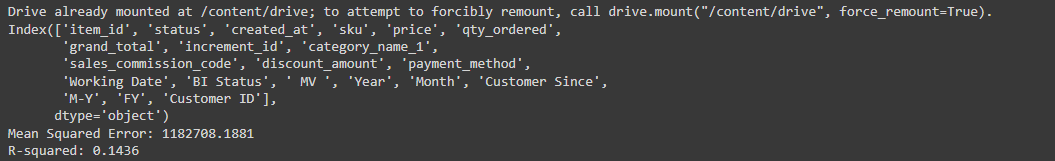
**Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual values.

**R-squared (R²):** Indicates how well the model explains the variance in the target variable.

These metrics were calculated using mean\_squared\_error and r2\_score from scikit-learn.

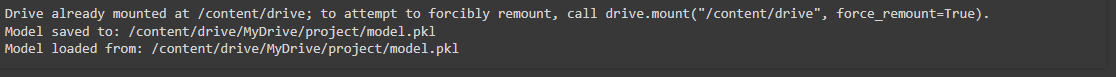


The output is:



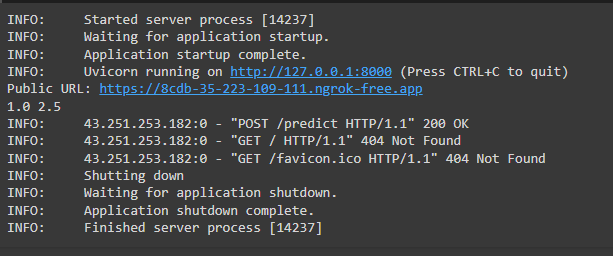
3)model load and save:

This was the output:



1. Creating a fast api

This was the result:



1. **Deployment at hugging face**

**1. Importing Libraries and Handling Dependencies:**

* **Import and Install gradio**:
  + First import the gradio library.
* **Import requests and gradio as gr**:
  + Imports the requests library for making HTTP requests to my API.
  + Imports gradio and assigns it the alias gr for easier use.

**2. Defining the predict Function:**

* **api\_url**:
  + Sets the URL of your FastAPI endpoint, which is currently hosted on ngrok.
* **data**:
  + Creates a dictionary to hold the input features (feature1 and feature2) that will be sent to the API.
* **response = requests.post(api\_url, json=data)**:
  + Sends a POST request to my API endpoint.
  + The json=data part ensures that the input data is sent in JSON format, which is how my FastAPI endpoint is likely expecting it.
* **if response.status\_code == 200**:
  + Checks if the API request was successful. A status code of 200 typically indicates a successful request.
  + If successful, it extracts the prediction value from the API's JSON response and returns it.
* **else**:
  + If the request was not successful, it returns an error message including the HTTP status code received from the API.

**3. Creating and Launching the Gradio Interface:**

* **iface = gr.Interface(...)**: Creates a Gradio interface.
  + fn=predict: Specifies that the predict function should be called when the user interacts with the interface.
  + inputs=["number", "number"]: Creates two input components of type "number" for the user to enter the values of feature1 and feature2.
  + outputs="text": Creates an output component of type "text" to display the prediction or error message returned by the predict function.
* **iface.launch()**: Launches the Gradio interface, making it accessible in your browser or within your Colab environment.
* **This is the output**:
* 