****

**Department of Artificial Intelligence**

**Amrita School of Engineering**

**Coimbatore 641 112, Tamil Nadu, India**

**Price Prediction of Used Cars**

***A project submitted***

***in partial fulfilment of the requirements for the degree of***

***Masters of Technology in Artificial Intelligence***

**By**

**Hemanth Raj P – (CB.SC.P2CSE23006)**

**Sanjay M – (CB.SC.P2CSE23012)**

**Patel Heet – (CB.SC.P2CSE23022)**

**Supervised by:**

**Dr. SenthilKumar T**

**Nov, 2023**

**MACHINE LEARNING**

|  |  |  |  |
| --- | --- | --- | --- |
| **Roll No** | **Name** | **Official Email id** | **Contribution** |
| CB.SC.P2CSE23006 | Hemanth Raj P | cb.sc.p2cse23006@cb.students.amrita.edu | Linear regression  Ridge Regression  Lasso Regression  K means with SVM  Bayesian Classifier  GRU |
| CB.SC.P2CSE23012 | Sanjay M | cb.sc.p2cse23012@cb.students.amrita.edu | Data preprocessing-Noise  One Hot Encoding  Naïve Bayes Classifier  Decision Tree  K-Means with CNN  LSTM |
| CB.SC.P2CSE23022 | Heet Patel | cb.sc.p2cse23022@cb.students.amrita.edu | K Means with KNN  Fuzzy C-Means with KNN  K Means with CNN  Logistic Regression |

**GitHub URL of the project page:**

[**https://github.com/HRJ10/Price-Prediction-of-Used-Cars/new/main**](https://github.com/HRJ10/Price-Prediction-of-Used-Cars/new/main)

**Kaggle URL of the dataset page:**

[**https://www.kaggle.com/datasets/sujay1844/used-car-prices/data**](https://www.kaggle.com/datasets/sujay1844/used-car-prices/data)

**SECTION 1**

* 1. **Application Name:** Price Prediction of Used Cars
  2. **Provide a set of analytical questions**
     1. **Why:**
  + Why is accurate price prediction important for used cars?
  + Why are certain features considered more influential in determining a used car's price?
  + Why do historical sales data and market trends play a crucial role in prediction models?
  + Why should external factors such as economic conditions and fuel prices be taken into account?
  + Why is it important to consider regional variations in pricing when predicting used car prices?
    1. **What:**
  + What features of a used car have the most significant impact on its price?
  + What types of data sources are essential for training an effective price prediction model?
  + What statistical or machine learning techniques are suitable for modelling used car prices?
  + What role do outlier detection and data cleaning play in improving the accuracy of price predictions?
  + What are the key challenges in obtaining and processing historical sales data for training the model?
    1. **How:**
  + How can machine learning algorithms be employed to analyze and predict used car prices?
  + How do you validate the accuracy and reliability of a used car price prediction model?
  + How should the model handle categorical variables, such as make and model, in the prediction process?
  + How can feature engineering enhance the predictive power of the model?
  + How do you account for the temporal aspect of the data, such as the impact of model years on pricing, in the prediction model?
  1. **Provide a set of questions for prediction**

**1.3.1 Data Exploration and Understanding:**

* What are the key features or variables available in the dataset for predicting used car prices?
* Which features are likely to have a direct impact on the pricing of used cars?
* Are there any missing or incomplete values in the dataset, and how will they be handled during the modelling process?
* How is the distribution of the target variable (used car prices) in the dataset? Are there any outliers?
* What categorical variables, such as make, model, or fuel type, are present, and how will they be encoded for the prediction model?

**1.3.2 Feature Engineering:**

* Which features can be transformed or engineered to better capture patterns in used car prices?
* How can you handle categorical variables to ensure they contribute effectively to the prediction model?
* Are there interactions between features that should be explicitly considered in the modeling process?

**1.3.3 Model Selection and Training:**

* Which machine learning algorithms are suitable for the task of predicting used car prices?
* How will you split the dataset into training and testing sets to evaluate the model's performance?
* What hyperparameters will you tune to optimize the model's predictive ability?
* Are there ensemble methods that may enhance the predictive performance of the model?

**1.3.4 Evaluation and Validation:**

* What metrics will be used to evaluate the performance of the price prediction model?
* How will you handle overfitting or underfitting issues in the model?
* What validation strategies will be employed to ensure the model generalizes well to unseen data?

**1.3.5 Interpretability and Explainability:**

* How will you interpret the results of the model and explain its predictions, especially to non-technical stakeholders?
* Are there specific features that have a more significant impact on the predicted prices, and how can this information be communicated effectively?
* Deployment and Monitoring:
* What steps are involved in deploying the model for real-world predictions?
* How will you monitor the model's performance over time and update it as needed?
  1. **Technologies Used:**

|  |  |
| --- | --- |
| **Editor** | Jupyter notebook |
| **Language** | Python |

* 1. **Why is this Application required?**

The Price Prediction of Used Cars through machine learning serves as a transformative force in the used car market, fundamentally altering how buyers and sellers navigate transactions. The necessity for such predictive models arises from the intricacies involved in determining the value of a used car, influenced by a myriad of factors. Machine learning, with its ability to process vast datasets and discern complex relationships, offers a sophisticated solution to unravel these influences. As online platforms provide an increasing volume of data, machine learning becomes an indispensable tool to distill meaningful insights from this wealth of information.

One of the primary advantages lies in the adaptability and personalization offered by machine learning models. Tailoring predictions to specific regions, markets, or car types ensures a level of accuracy and relevance that traditional methods may struggle to achieve. The automation of the pricing process not only streamlines decision-making but also introduces an element of competitiveness to the market. Sellers armed with accurate predictions can strategically price their cars, attracting more buyers, while consumers benefit from the efficiency and convenience of automated pricing.

Risk mitigation is another crucial aspect addressed by machine learning models. The inherent uncertainties associated with pricing used cars, such as the risk of overpricing or under pricing, are significantly reduced. This contributes to a more transparent and trustworthy market environment, where buyers and sellers can make decisions with greater confidence.

Furthermore, the adaptability of machine learning models to changing market conditions ensures that predictions remain relevant and up-to-date. Economic fluctuations, fuel price variations, and other external factors impacting the used car market can be dynamically incorporated into the models, providing real-time insights. This adaptability not only facilitates informed decision-making but also contributes to overall customer satisfaction by aligning pricing strategies with current market dynamics.

In summary, the application of machine learning in the Price Prediction of Used Cars is a multifaceted solution that goes beyond mere automation. It revolutionizes the market by offering personalized, data-driven insights, fostering transparency, and empowering both buyers and sellers to engage in transactions with a heightened level of confidence and efficiency.

* 1. **List of similar applications:**

|  |  |
| --- | --- |
| **Application Name** | **URL** |
| Kelley Blue Book (KBB) | [**https://www.kbb.com/**](https://www.kbb.com/) |
| Edmunds | <https://www.edmunds.com/> |
| Carfax | <https://www.carfax.com/cars-for-sale> |
| TrueCar | <https://www.truecar.com/> |

* 1. **What is unique in your project:**

**Hybrid Model Approach:**

The program employs a hybrid approach by using two different types of models for different aspects of the prediction. The LSTM model is utilized for predicting the category of the used car, while a logistic regression model is used for predicting the exact price within that category.

**Category Prediction Using LSTM:**

The LSTM model is used to predict the category of the used car. This approach of categorizing cars before predicting the exact price allows for a more structured and potentially accurate prediction, as different categories may have distinct pricing patterns.

**Separation of Concerns:**

The program separates the tasks of category prediction and exact price prediction, utilizing different models for each. This modular approach enhances the interpretability of the prediction process and allows for more straightforward model management and updates.

**User-Friendly Input:**

The program prompts the user to input various details about the used car, such as ODO reading, year, age, mileage, engine, power, seats, fuel type, transmission type, and owner type. This user-friendly input process makes the program accessible for users without a technical background.

**Scalability and Reusability:**

The use of joblib for loading models and a separate preprocessing function (preprocess\_input\_lstm) contributes to the scalability and reusability of the code. It allows for easy integration with other applications and the reuse of components in different contexts.

**Adaptability to Categorical Features:**

The program handles categorical features such as fuel type, transmission type, and owner type by encoding them appropriately, making the model adaptable to different categorical values and choices made by users.

**Use of Deep Learning (LSTM) and Traditional ML (Logistic Regression):**

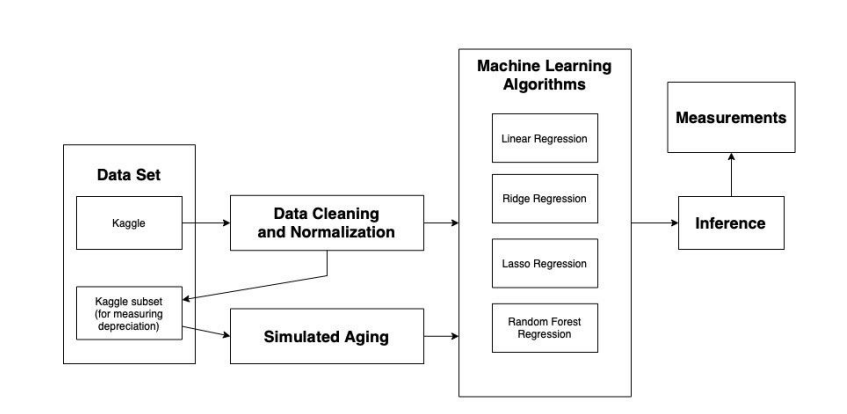
The combination of an LSTM neural network, a type of deep learning model suitable for sequential data, and a logistic regression model, a traditional machine learning algorithm, showcases the flexibility of using different approaches based on the nature of the problem.

**SECTION 2**

* 1. **Journal Details**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Paper Name** | **Journal Name** | **Citation** |
| 1 | Object Detection and Used Car Price Predicting Analysis System (UCPAS) Using Machine Learning Technique | Linguistics and Culture Review | Yadav, A., Kumar, E., & Yadav, P. K. (2021). Object detection and used car price predicting analysis system (UCPAS) using machine learning technique. Linguistics and Culture Review, 5(S2), 1131-1147. https://doi.org/10.21744/lingcure.v5nS2.1660. |
| 2 | Machine Learning-Powered Mobile App for Predicting Used Car Prices | Big-data Service and Intelligent Computation | BDSIC 2021, November 19–21, 2021, Xiamen, China © 2021 Association for Computing Machinery. ACM ISBN 978-1-4503-9055-2/21/11. . . $15.00 https://doi.org/10.1145/3502300.3502307 |
| 3 | Car Price Prediction | International Conference on Information Management & Machine Intelligence | ICIMMI 2022, December 23, 24, 2022, Jaipur, India © 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-9993-7/22/12. . . $15.00 https://doi.org/10.1145/3590837.3590864 |
| 4 | Pre-owned car price prediction by employing machine learning techniques | Journal of Decision Analytics and Intelligent Computing | Original Scientific Article Journal of Decision Analytics and Intelligent Computing Vol. 3 issue 1, (2023) 167-184https://doi.org/10.31181/jdaic10008102023n |
| 5 | An Automated Car Price Prediction System Using Effective Machine Learning Techniques | International Conference on Computational Intelligence and Sustainable Engineering Solutions | 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)978-1-6654-8004-8/22/$31.©2022 IEEEDOI:10.1109/CISES54857.2022.9844350 |
| 6 | CarPricePrediction:An Application of Machine Learning | International Conference on Inventive Computation Technologies | 2023 International Conference on Inventive Computation Technologies (ICICT)|979-8-3503-9849-6/23/$31.00 ©2023IEEEDOI:10.1109/ICICT57646.2023.10134142 |
| 7 | Machine Learning Techniques To Predict The Price Of Used Cars | Second International Conference on Electronics and Sustainable Communication Systems | 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC) | 978-1-6654-2867-5/21/$31.00 ©2021 IEEEDOI:10.1109/ICESC51422.2021.9532845 |
| 8 | Predicting the Prices of the Used Cars using Machine Learning for Resale | International Students' Conference on Electrical, Electronics and Computer Science | 2023 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS) | 979-8-3503-9874-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/SCEECS57921.2023.10063133 |
| 9 | Prediction of Used Car Price Based on Supervised Learning Algorithm | International Conference on Networking, Communications and Information Technology | 2021 International Conference on Networking, Communications and Information Technology (NetCIT) | 978-1-6654-0070-1/21/$31.00 ©2021 IEEEDOI:10.1109/NetCIT54147.2021.00036 |
| 10 | Used Car Price Prediction using Machine Learning: A Case Study | International Symposium on Signal, Image, Video and Communications | 2022 11th International Symposium on Signal, Image, Video and Communications (ISIVC) | 978-1-6654-8724-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISIVC54825.2022.9800719 |

* 1. **Model Diagram:**



The Model diagram is a flowchart that describes a process involving data handling and machine learning. Here is a description of the flow:

**Data Set:**

There are two boxes inside the "Data Set" box. The first one is labelled "Kaggle," which indicates that the dataset comes from Kaggle, a popular platform for data science competitions and datasets. The second one is labelled "Kaggle subset (for measuring depreciation)," suggesting a specific subset of the Kaggle data is used for measuring depreciation.

**Data Cleaning and Normalization:**

The data from Kaggle and the Kaggle subset is then subjected to "Data Cleaning and Normalization." This is a crucial step in data preprocessing that involves cleaning the data by handling missing values, outliers, or incorrect data and normalizing it to ensure that the data is in a format or scale that is suitable for analysis.

**Simulated Aging:**

After cleaning and normalization, there is a process labelled "Simulated Aging." This might refer to a technique where data is manipulated to simulate the effect of aging or time on the dataset, possibly to understand depreciation.

**Machine Learning Algorithms:**

The cleaned and aged data is then fed into various "Machine Learning Algorithms," which are listed as "Linear Regression," "Ridge Regression," "Lasso Regression," and "Random Forest Regression." These are common algorithms used for regression tasks in machine learning, each with its own approach to finding patterns in the data.

**Measurements:**

The output from the machine learning algorithms seems to go into a box labelled "Measurements," which implies that the results of the machine learning algorithms are quantified or measured in some way.

Inference: Finally, the "Measurements" lead to "Inference," which suggests that the measurements are used to make inferences or draw conclusions from the data analysis.

**SECTION 3**

* 1. **Dataset Description**

The dataset comprises comprehensive information and attributes related to used cars, with a primary focus on the target variable— the price of the used cars, measured in lakhs. The data has been obtained from "Cars24."

**Data Type:**

The dataset is tabular, organized in rows and columns. Each row corresponds to a specific listing for a used car, while each column represents a distinct attribute or feature associated with these cars.

**Features:**

* Make and Model: Specifies the brand and model of the car.
* Location or City of Sale: Indicates the city or location where the car is being sold.
* Year of Manufacture: Represents the manufacturing year of the car.
* Mileage: Reflects the distance the car has traveled in terms of kilometers.
* Odometer (Kilometers Driven): Provides the reading on the odometer, indicating the total distance the car has been driven.
* Fuel Type: Specifies the type of fuel the car utilizes, such as petrol or diesel.
* Transmission Type: Indicates the transmission system of the car, distinguishing between manual and automatic.
* Number of Owners: Represents the count of previous owners the car has had.
* Engine Displacement: Specifies the total volume of all cylinders in the engine.
* Engine Horsepower: Represents the power output of the car's engine.
* Number of Seats: Indicates the total seating capacity of the car.
* Price When the Car Was New: Reflects the original price of the car when it was first sold as new.

This dataset provides a comprehensive overview of various factors that may influence the pricing of used cars, allowing for analysis and insights into the dynamics of the used car market.

* 1. **Preprocessing**

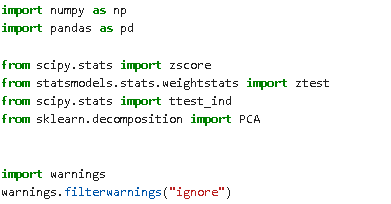
In the realm of machine learning, preprocessing plays a pivotal role as it involves refining and transforming raw data before feeding it into a model. This essential step encompasses a variety of techniques, including data cleaning by managing missing values and eliminating duplicates, feature transformation through processes like standardization or normalization, encoding categorical variables, and implementing dimensionality reduction. Moreover, preprocessing addresses tasks such as handling imbalanced data, splitting the dataset into training and testing sets, managing outliers, and, in the context of natural language processing, undertaking text-specific operations like tokenization and stemming. The effectiveness of preprocessing is evident in its ability to enhance model performance by addressing challenges such as overfitting and biases, ultimately contributing to the development of accurate and resilient machine learning models.

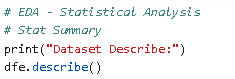
* + 1. **Descriptive and Statistical Feature Analysis**

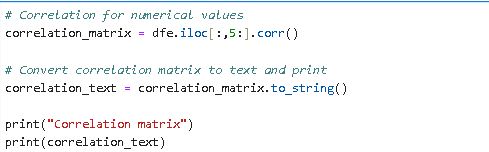
**Descriptive Statistics:**

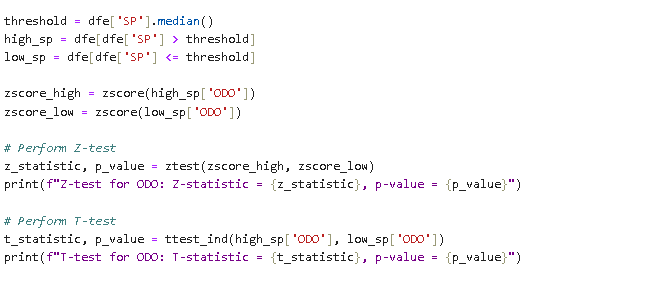
Descriptive statistics constitute a fundamental branch of statistical analysis focused on summarizing and elucidating the essential features of a dataset. This approach aims to provide a concise overview, offering insights into the central tendencies, variability, and distribution of data. Measures of central tendency, such as the mean, median, and mode, convey information about where the data clusters. Meanwhile, measures of dispersion, including range, variance, and standard deviation, illuminate the spread or variability of values within the dataset. Assessing the shape of the distribution involves skewness and kurtosis, indicating whether the data is symmetrically distributed or exhibits tails that deviate from a normal distribution. Percentiles help to discern the position of values within the dataset relative to others, and frequency distributions and graphical representations like histograms and box plots offer visual insights into the data's structure. These descriptive statistical tools collectively serve as a foundational step in data analysis, facilitating the identification of patterns, trends, and outliers. In the context of machine learning and data science, descriptive statistics serve as a crucial starting point for exploratory data analysis, guiding subsequent steps in data preprocessing and modelling.

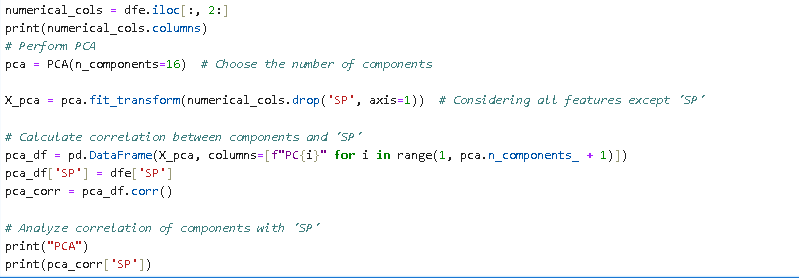
**Statistical Analysis.ipynb:**

 ****

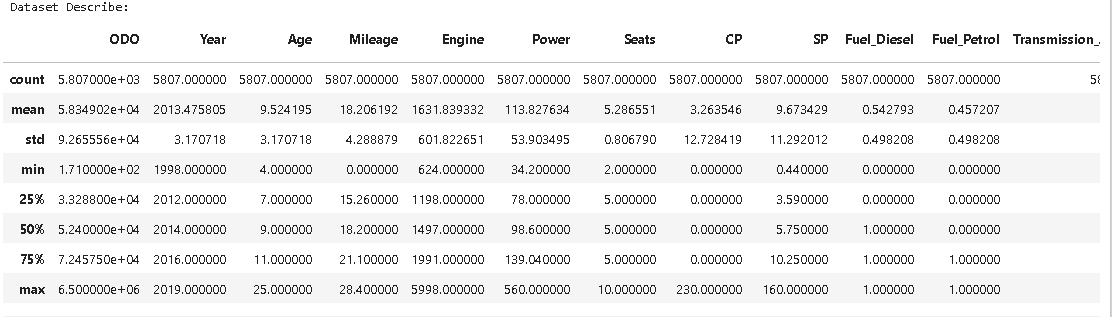


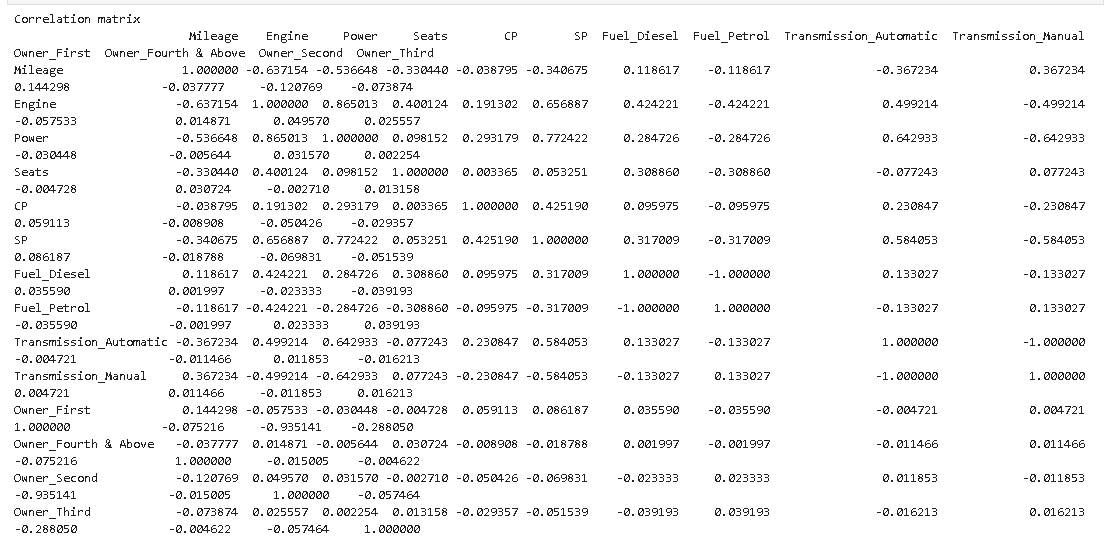
****

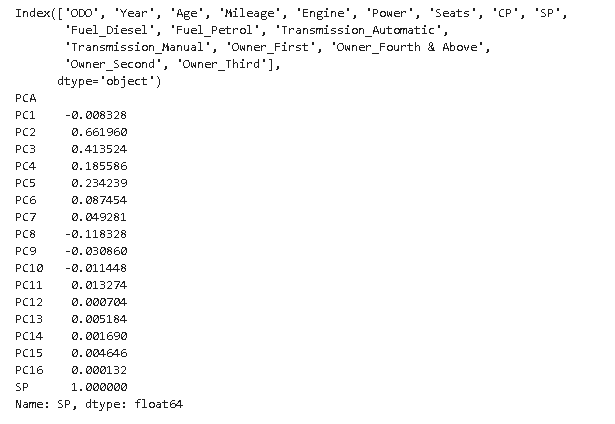
****

****

**Output:**

****

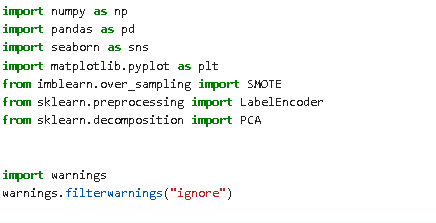


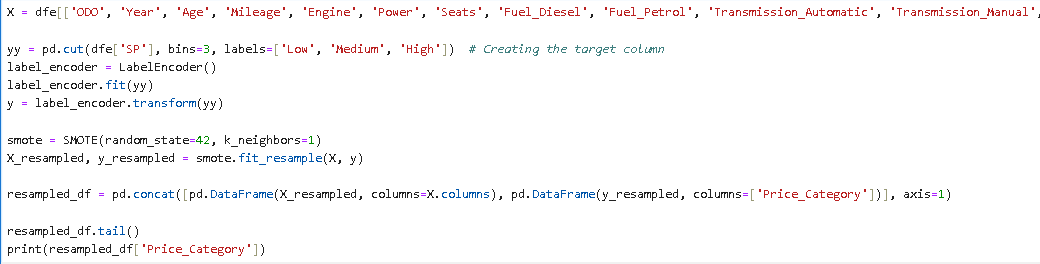
****

**Inference:**

* In this script, several Python libraries are utilized to facilitate various aspects of data analysis and machine learning. The inclusion of **`numpy`** and `pandas` is foundational, as these libraries are widely employed for numerical operations and data manipulation in Python. The **`scipy.stats`** library comes into play for statistical functions, and the **`statsmodels.stats.weightstats`** module is specifically employed for weighted statistical tests. Additionally, the `**sklearn.decomposition**` module from scikit-learn is harnessed for its functionality in implementing Principal Component Analysis (PCA), a technique used to reduce the dimensionality of the dataset.
* The initial step involves loading a dataset named "encoded-train.csv" using the **`pd.read\_csv`** function from the pandas library. This dataset likely contains information relevant to the prediction of used car prices.
* The script delves into descriptive statistics using the **`describe()`** function from pandas, providing a comprehensive summary of the dataset's statistical measures such as mean, standard deviation, minimum, maximum, and quartiles for each numerical column. This summary serves as an essential exploration tool, aiding in the understanding of the data distribution.
* Moving forward, a correlation matrix is computed for the numerical columns, offering insights into the relationships between variables. Particularly, the script focuses on the 'SP' variable, presumably representing the price of used cars, highlighting its correlation with other features.
* The analysis progresses with a segmentation strategy based on the median of the 'SP' column, resulting in two subsets, namely **`high\_sp**` and `**low\_sp`**. Z-scores are then calculated for the 'ODO' (odometer readings) column within these subsets, followed by z-tests and t-tests to assess the significance of differences in mean z-scores and means between the subsets. This statistical approach aids in identifying patterns related to the 'ODO' variable across different price levels.
* The script concludes with the application of Principal Component Analysis (PCA) to reduce the dataset's dimensionality. The resultant principal components are examined for their correlation with the target variable 'SP', offering insights into which components are most influential in predicting used car prices. This comprehensive analysis, ranging from statistical summaries to advanced techniques like PCA, lays a robust foundation for subsequent steps in the development of a machine learning model for predicting used car prices.
  + 1. **Noise Removal (SMOTE Algorithm):**

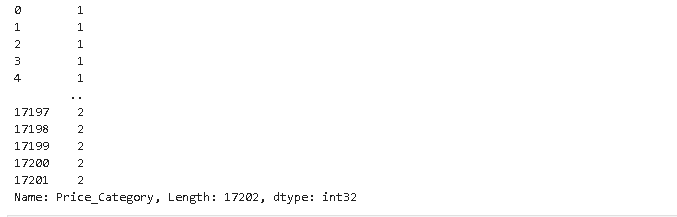
**SMOTE.ipynb:**

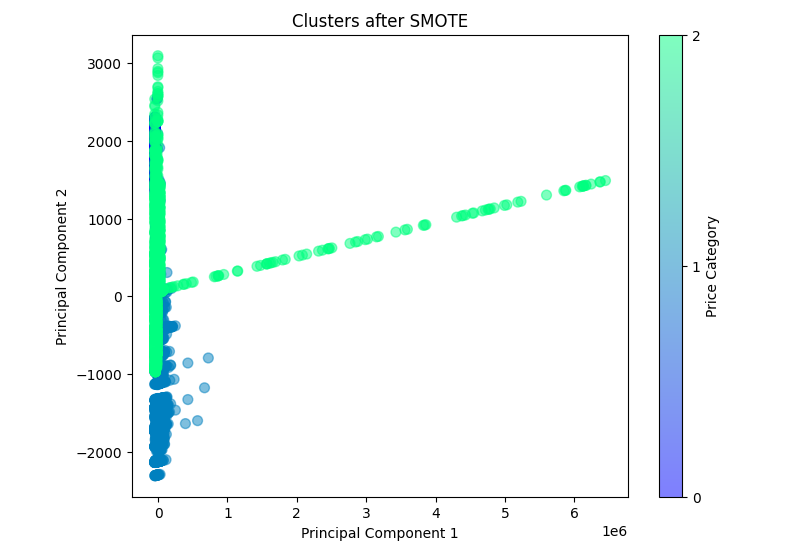
****

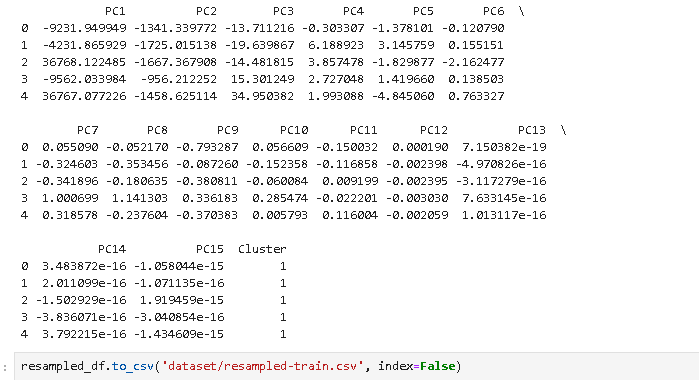
****

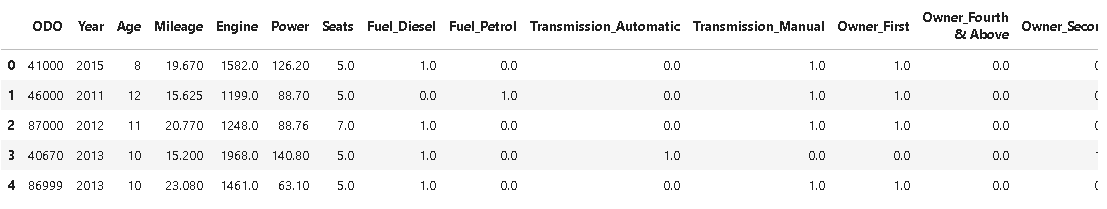


**Output:**

****

****

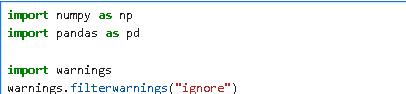
****

****

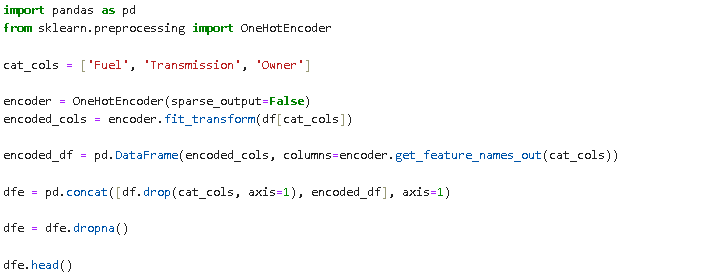
**Inference:**

* The provided Python script focuses on the preprocessing and analysis of a dataset related to used car prices, employing various techniques to address imbalances and enhance the dataset for subsequent machine learning tasks.
* Beginning with the loading of the dataset from a CSV file, key features such as odometer readings, year, age, mileage, engine specifications, power, seats, and categorical variables like fuel type, transmission, and ownership history are selected for analysis.
* The script then introduces a target variable by categorizing the original price ('SP') into three distinct classes: Low, Medium, and High, providing a categorical representation of the price range.
* To handle class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) from the imbalanced-learn library is applied. SMOTE works by generating synthetic samples for the minority classes, thereby mitigating imbalances in the dataset and ensuring more robust model training.
* Following the resampling process, Principal Component Analysis (PCA) is employed for dimensionality reduction. This technique transforms the dataset into a set of uncorrelated principal components, allowing for a more concise representation of the data while retaining its essential information.
* The resulting principal components are visualized in a scatter plot, illustrating distinct clusters based on the categorized price ranges. This step provides insights into how the resampled data is distributed in reduced-dimensional space.
* Finally, the script exports the transformed and resampled data into a new CSV file named 'resampled-train.csv.' This file serves as a preprocessed and balanced dataset, ready for further analysis or machine learning model training. The comprehensive approach adopted in this script ensures that potential issues related to imbalanced data are effectively addressed, setting the stage for more accurate and reliable predictions in the domain of used car price categorization.
  + 1. **One Hot Encoding:**

**One Hot Encoding.ipynb:**

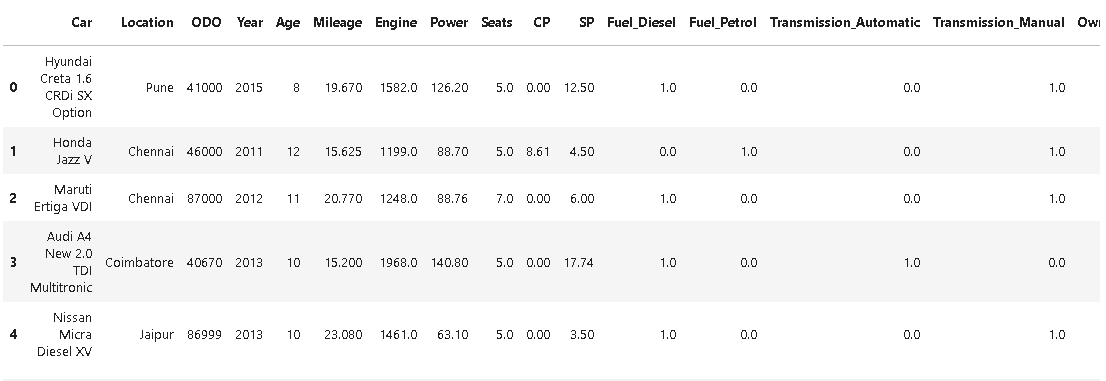
****







**Output:**



**Inference:**

* This Python script performs categorical feature encoding and data preprocessing on a dataset related to used cars. After loading the dataset from a CSV file named "clean-train.csv" using pandas, the script focuses on categorical feature transformation.
* Three categorical columns—'Fuel,' 'Transmission,' and 'Owner'—are identified for one-hot encoding. The script utilizes the OneHotEncoder from scikit-learn to convert these categorical variables into a binary matrix representation.
* The encoded columns are then combined with the original dataframe, resulting in a new dataframe named 'dfe.' To handle any potential missing values, the script removes rows containing NaN (not a number) values. The preprocessed dataframe is displayed using the 'head()' function to provide a glimpse of the transformed data.
* Finally, the script exports the processed dataset to a new CSV file named "encoded-train.csv," ensuring that the data is now ready for use in machine learning models.
* This series of steps ensures that categorical variables are appropriately encoded, facilitating their integration into predictive models for tasks such as used car price prediction.
* The encoding process is crucial as it transforms qualitative data into a format suitable for mathematical modeling, enhancing the effectiveness of subsequent analyses and machine learning applications.

**SECTION 4**

* 1. **Regression**

Regression is a statistical technique used in machine learning and statistics to model the relationship between a dependent variable (also called the target or response variable) and one or more independent variables (also called predictors or features). The goal of regression analysis is to understand how the independent variables impact the dependent variable and to make predictions based on this understanding.

* + 1. **Linear Regression**

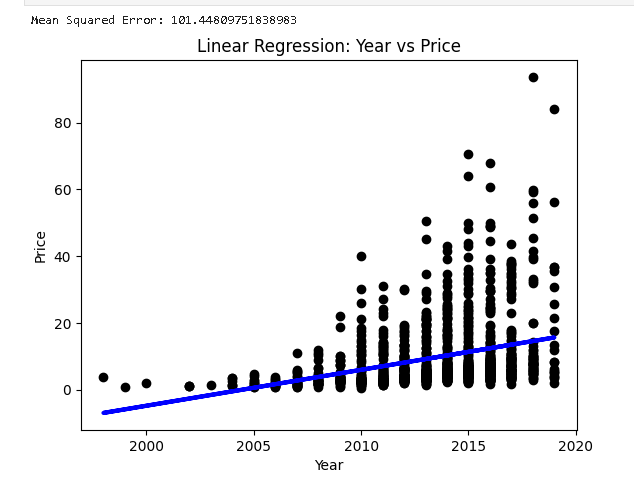
Linear regression is a statistical method used for modelling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. The goal of linear regression is to find the bestfitting line that describes the relationship between the variables. This line is called the regression line or the line of best fit.

**linear.ipynb:**





**Output:**

****

**Inference:**

* pandas is used for data manipulation and analysis.
* train\_test\_split from sklearn.model\_selection is used to split the dataset into training and testing sets.
* LinearRegression from sklearn.linear\_model is the linear regression model used for fitting the data.
* mean\_squared\_error from sklearn.metrics is used to evaluate the model's performance.
* matplotlib.pyplot is used for data visualization.
* Load dataset from a CSV file ('cleantrain.csv') into a DataFrame (df).
* Selects only the 'Year' and 'SP' columns from the DataFrame.
* Drops rows with missing values.
* Splits the data into input features (X) and the target variable (y).
* Uses train\_test\_split to split the data into training and testing sets (80% training, 20% testing).
* Creates a linear regression model and fits it to the training data
* Uses the trained model to make predictions on the test set.
* Computes the mean squared error between the true values (y\_test) and the predicted values (y\_pred).
* Prints the mean squared error as a measure of model performance.
* Plots a scatter plot of the test data points.
* Plots the regression line based on the predictions (y\_pred).
* Sets labels and title for better visualization. Displays the plot.

**Parameter set in the Model :**

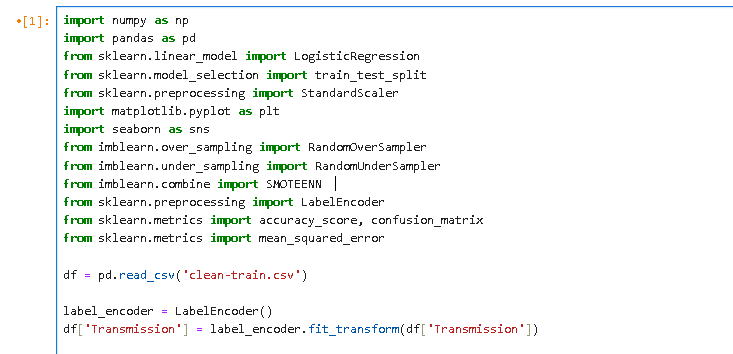
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |

**Parameters evaluated in the model:**

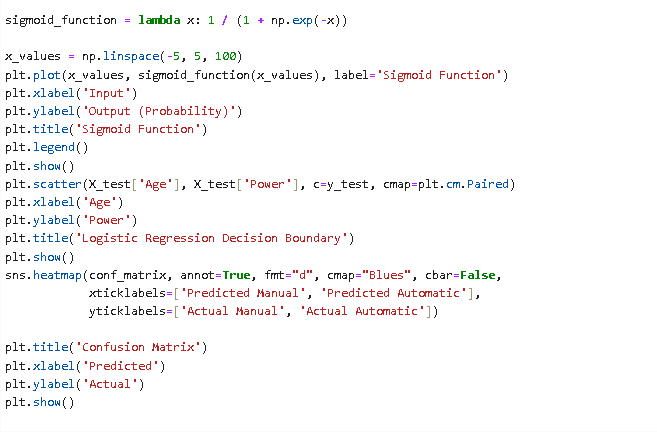
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Mean Square error | Mean Square Error (MSE) is a metric commonly used to measure the average squared difference between the actual and predicted values in a set of data. | 101.448 |

* + 1. **Logistic Regression**

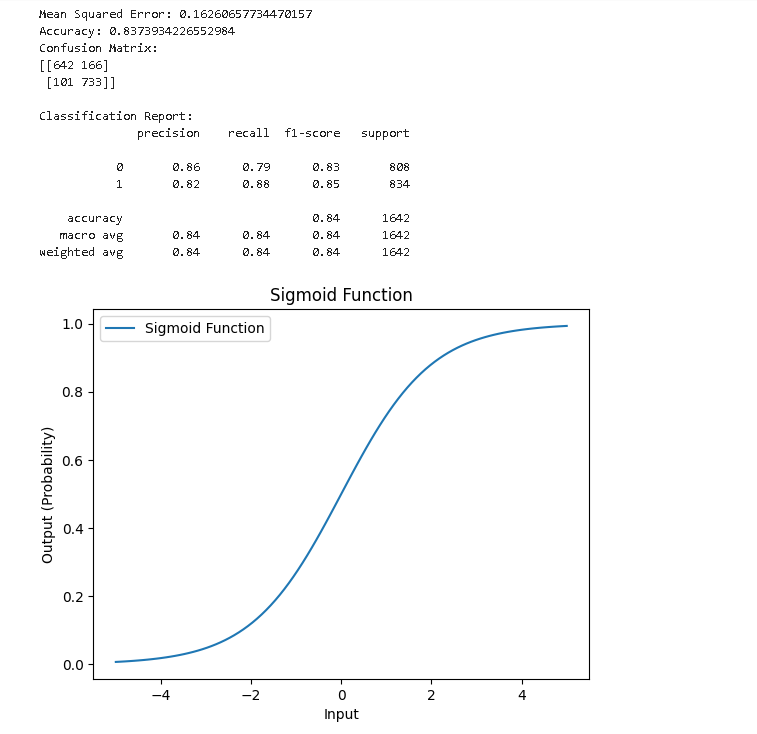
Logistic Regression is a statistical method used for binary classification, which means predicting the probability of an observation belonging to one of two classes. Despite its name, logistic regression is actually a classification algorithm, not a regression algorithm. It's widely used in various fields, including medicine, finance, and machine learning. logistic regression is a powerful and widely used algorithm for binary classification problems.

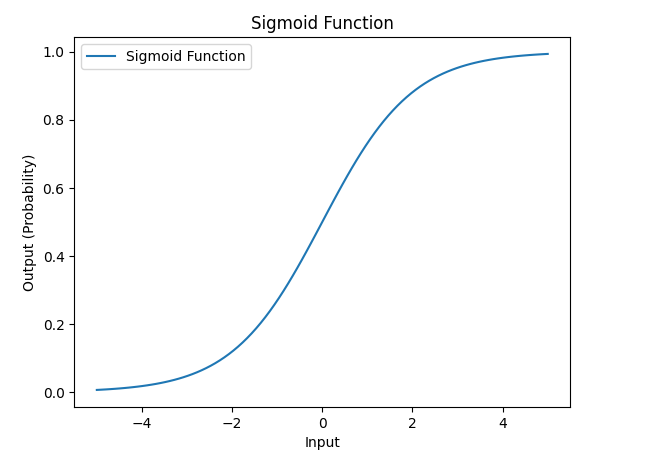
**logesticregression.ipynb: **  

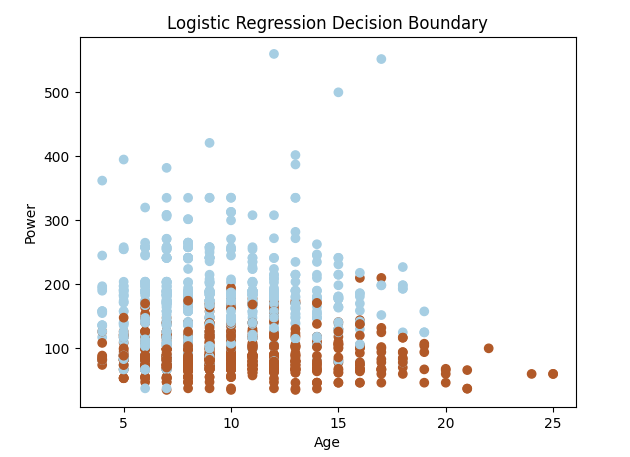


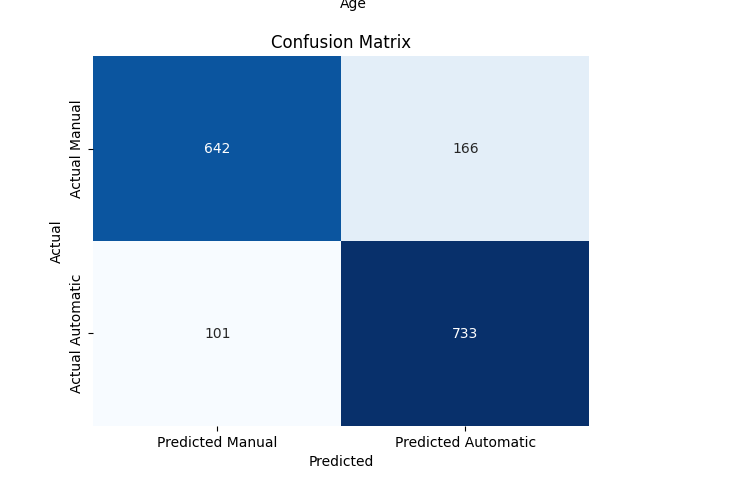


**Output:**









**Inference:**

This Python script appears to be implementing a binary classification task using logistic regression to predict the 'Transmission' type (manual or automatic) based on features such as 'Age', 'Mileage', 'Engine', 'Power', and 'Seats'. Let me break down the code for you:

1. Importing Libraries:

* `numpy` and `pandas` for data manipulation.
* `LogisticRegression` from `sklearn.linear\_model` for logistic regression.
* train\_test\_split` from `sklearn.model\_selection` for splitting the dataset into training and testing sets.
* `StandardScaler` from `sklearn.preprocessing` for standardizing the input features.
* `matplotlib.pyplot` and `seaborn` for data visualization.
* `RandomOverSampler`, `RandomUnderSampler`, and `SMOTEENN` from `imblearn` for handling class imbalance.
* `LabelEncoder` from `sklearn.preprocessing` for encoding categorical labels.
* Various metrics like `accuracy\_score`, `confusion\_matrix`, and `mean\_squared\_error` from `sklearn.metrics`.

2. Reading Data:

* Reads a CSV file named 'cleantrain.csv' into a pandas DataFrame (`df`).

3. Label Encoding:

* `. Encodes the 'Transmission' column (assuming it contains categorical values like 'manual' and 'automatic') using `LabelEncoder

4. Feature and Target Variable Selection:

* Defines the feature matrix `X` and the target variable `y` based on selected columns in the DataFrame.

5. Handling Class Imbalance:

* Uses `RandomOverSampler` to balance the class distribution in the dataset.

6. TrainTest Split:

* Splits the dataset into training and testing sets using `train\_test\_split`.

7. Standardization:

* Standardizes the feature matrices of the training and testing sets using `StandardScaler`.

8. Logistic Regression Model:

* Creates a logistic regression model and fits it to the standardized training data.

9. Prediction and Thresholding:

* Predicts probabilities for the test set and converts them to binary predictions using a threshold of 0.5.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Threshold | The term "threshold" usually refers to a decision boundary used to classify instances into different classes | 0.5 |

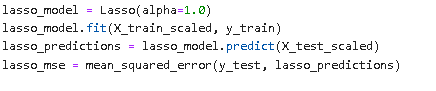
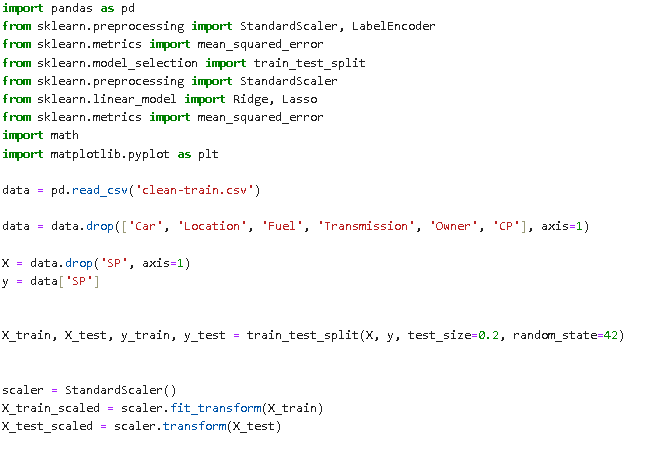
**Parameters evaluated in the model:**

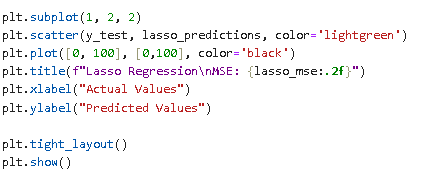
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Mean Square error | Mean Square Error (MSE) is a metric commonly used to measure the average squared difference between the actual and predicted values in a set of data. | 101.448 |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.837 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[642 , 166],  [101, 733]] |

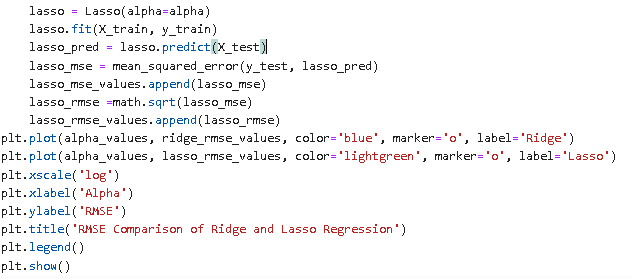
* + 1. **Lasso Regression:**

Lasso regression, also known as Least Absolute Shrinkage and Selection Operator, stands out as a highly effective regularization technique in the field of machine learning. What distinguishes Lasso is its unique capability not only to create a predictive model but also to conduct feature selection by driving specific coefficients precisely to zero. This regularization approach introduces a penalty term into the linear regression cost function, directly proportional to the absolute values of the coefficients. By adjusting the regularization strength parameter, often referred to as 'alpha,' Lasso promotes sparsity in the model, effectively removing less influential features. This feature selection attribute proves advantageous, especially when working with high-dimensional datasets, as it allows for the identification and utilization of only the most impactful features. This, in turn, enhances model interpretability and improves its ability to generalize to new data. I value Lasso regression for its dual role in prediction and model simplification, prioritizing the most relevant features and rendering it a valuable tool in my machine learning .

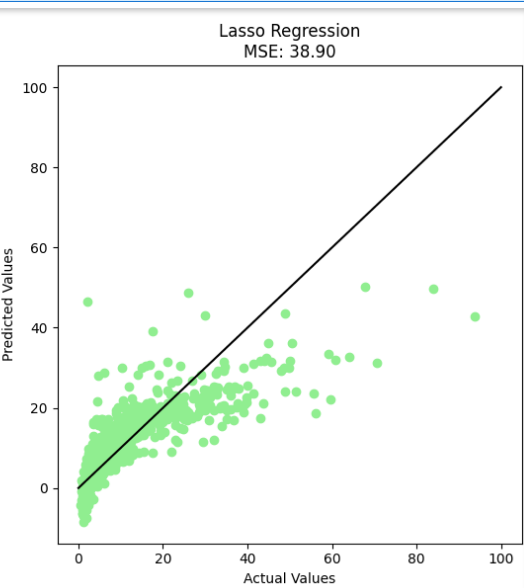
**RidgeandLasso.ipynb:**

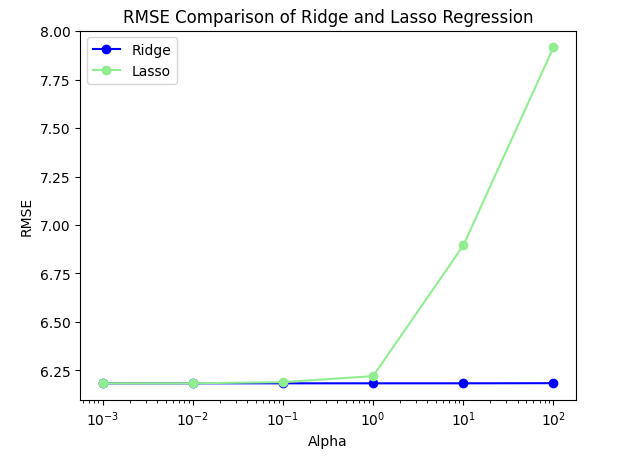






**Output:**

****

****

**Inference:**

* The presented Python script is centered around the exploration and application of Lasso regression, a powerful regularization technique in machine learning.
* To begin, the script imports necessary libraries and reads a dataset from a CSV file ('clean-train.csv') into a pandas DataFrame.
* A crucial step in data preprocessing follows, involving the removal of specified columns deemed irrelevant for the analysis.
* Subsequently, the dataset is divided into features (X) and the target variable (y), setting the stage for supervised learning.
* The data is further partitioned into training and testing sets using the `train\_test\_split` function, a common practice to assess model performance on unseen data.
* Additionally, feature scaling is employed via the `StandardScaler` to standardize the features, a prerequisite for many machine learning algorithms.
* The script then proceeds to train a Lasso regression model with a fixed regularization strength (`alpha = 1.0`) and evaluates its predictive performance on the testing set.
* The mean squared error (MSE) is calculated as a metric for model assessment. Notably, the script incorporates a visualization aspect, generating a scatter plot that juxtaposes the predicted values against the actual values, providing a tangible representation of the model's predictive accuracy.
* Expanding on the analysis, the script delves into a more nuanced investigation of Lasso regression by varying the regularization parameter (`alpha`).
* Through an iterative process, the script trains multiple Lasso regression models with different alpha values and records both MSE and root mean squared error (RMSE) for each configuration.
* This meticulous exploration enables a nuanced understanding of how different regularization strengths influence the model's performance, shedding light on the interplay between regularization and predictive accuracy.
* The script culminates in a comparative visualization, plotting the RMSE values of both Ridge and Lasso regression models across a spectrum of alpha values.
* The logarithmic scale on the x-axis enhances the visibility of differences in RMSE. This final step provides a comprehensive view of how varying alpha values impact the trade-off between bias and variance, contributing valuable insights into the feature selection capabilities and generalization performance of Lasso regression.
* In essence, the script serves as a robust exploration of Lasso regression, combining quantitative metrics and visual representations to unravel the nuances of this regularization technique in the context of predictive modeling.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| alpha | The regularization parameter controls the amount of shrinkage applied to the coefficients of the model | 1.0 |

**Parameters evaluated in the model:**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Mean Square error | Mean Square Error (MSE) is a metric commonly used to measure the average squared difference between the actual and predicted values in a set of data. | 38.90 |

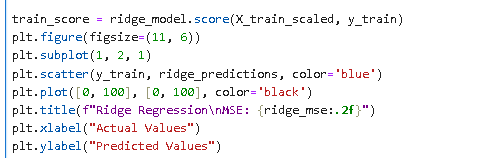
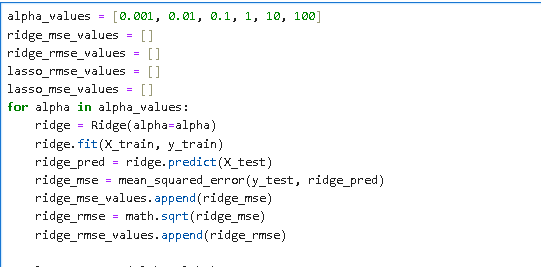
* + 1. **Ridge Regression:**

Ridge Regression, a valuable technique in machine learning, extends linear regression by incorporating regularization to address issues such as multicollinearity and potential overfitting. What distinguishes Ridge Regression is its addition of a regularization term in the cost function, which is proportional to the square of the magnitude of the coefficients. This inclusion encourages the model not only to fit the data well but also to maintain small coefficients. This feature becomes particularly beneficial when dealing with high-dimensional datasets or when certain features exhibit high correlation. The regularization strength is determined by a hyperparameter, often referred to as 'alpha,' and adjusting this parameter enables a balance between fitting the data and preventing overfitting. As a result, Ridge Regression stands out as a robust tool in my machine learning toolkit, offering a

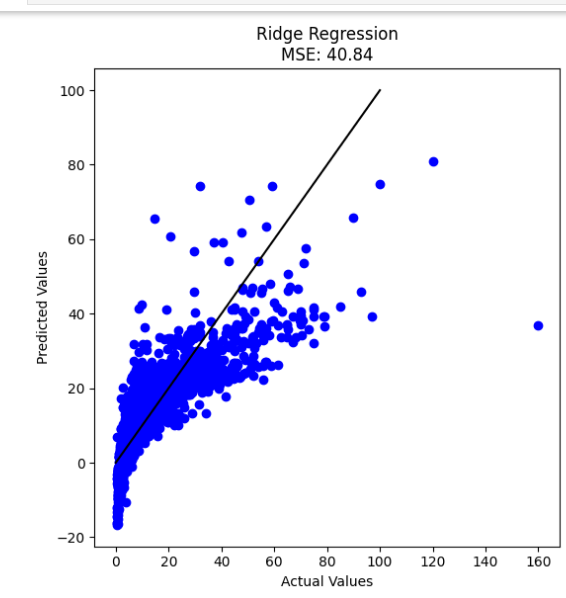
means to handle complex datasets while promoting stability in model coefficients.

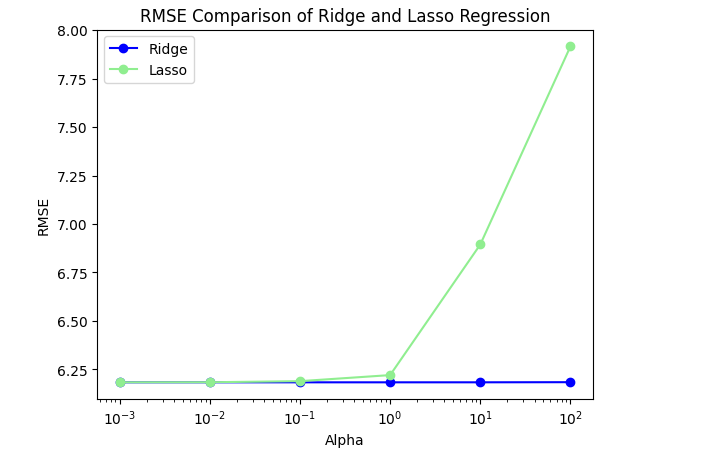
**RidgeandLasso.ipynb:**

****

** **

**Output:**

****

****

**Inference:**

* The script initiates by loading a dataset from a CSV file ('clean-train.csv') into a pandas DataFrame named `data`.
* Following this, it conducts data preprocessing by removing specific columns, namely 'Car', 'Location', 'Fuel', 'Transmission', 'Owner', and 'CP', which are deemed unnecessary for the analysis.
* Subsequently, the dataset is divided into two parts: features (denoted as `X`) and the target variable (denoted as `y`).
* The script then further splits the data into training and testing sets using the `train\_test\_split` function from scikit-learn.
* To ensure consistency in the randomness of the split, the parameter `random\_state` is set to 42.
* The feature values are standardized using the `StandardScaler` from scikit-learn, a crucial preprocessing step for certain machine learning algorithms.
* The Ridge regression model is then instantiated and trained on the standardized training data with a predefined regularization strength (alpha).
* Performance metrics such as mean squared error (MSE) and the R-squared score are calculated on the training set.
* The script also includes a visualization component, showcasing a scatter plot that compares actual target values with predicted values generated by the Ridge regression model on the training set. The black line in the plot represents a perfect prediction scenario.
* Furthermore, the script delves into a more comprehensive analysis by exploring the impact of different alpha values on Ridge regression performance.
* It iterates over a predefined set of alpha values (0.001, 0.01, 0.1, 1, 10, 100) and, for each alpha, trains a Ridge regression model on the training set.
* The model's predictions are then evaluated on the testing set, and both mean squared error (MSE) and root mean squared error (RMSE) are computed and stored.
* This iterative process allows for a systematic examination of how varying regularization strengths influence the model's predictive capabilities.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| alpha | The regularization parameter controls the amount of shrinkage applied to the coefficients of the model | 1.0 |

**Parameters evaluated in the model:**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Mean Square error | Mean Square Error (MSE) is a metric commonly used to measure the average squared difference between the actual and predicted values in a set of data. | 38.90 |

**Comparison of Regression Models :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm Name** | **Parameter-1 (Test Size)** | **Parameter -2**  **(Random State)** | **Parameter -3**  **(Alpha)** |
| **Linear Regression** | 0.2 | 42 | - |
| **Logistic Regression** | 0.2 | 42 | - |
| **Ridge Regression** | 0.2 | 42 | 0.01 |
| **Lasso Regression** | 0.2 | 42 | 0.01 |

**SECTION 5**

**5.1 Classifiers**:

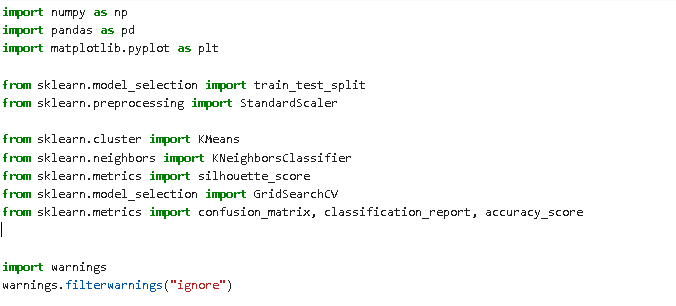
In the realm of machine learning, a classifier is an algorithm designed to learn and recognize patterns within labeled datasets, enabling it to make predictions or decisions about new, unseen data. Essential to the supervised learning paradigm, classifiers are trained on datasets where input instances are associated with known output labels. These algorithms aim to discern relationships between input features, which describe the attributes of instances, and the corresponding output labels or classes. During the training phase, the classifier fine-tunes its internal parameters to minimize the disparity between its predictions and the true labels in the training data. Once trained, the classifier utilizes its learned model to predict the labels of new instances it has not encountered before, facilitating the generalization of learned patterns to unseen data. Various classifier types exist, such as logistic regression, decision trees, random forests, support vector machines, and k-nearest neighbors, each tailored to specific applications. Evaluation metrics, including accuracy, precision, recall, and F1 score, assess the classifier's performance. Striking a balance between fitting the training data well and generalizing to new instances is crucial to avoid underfitting and overfitting. Classifiers find applications in diverse fields, including image recognition, spam filtering, medical diagnosis, and sentiment analysis. The choice of a classifier depends on the characteristics of the data and the objectives of the classification task.

**5.1.1 K Means with KNN:**

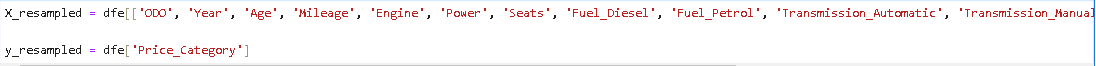
In a combined approach utilizing K-Means clustering and K-Nearest Neighbors (KNN), K-Means serves as an unsupervised learning technique to group data points into clusters based on their features. This clustering step is advantageous for dimensionality reduction, effectively grouping similar data points together. Subsequently, a new feature is engineered, representing the cluster assignments obtained from K-Means. This new feature, along with the original features, is then employed as input for training a KNN classifier. The KNN algorithm, being a supervised learning method, predicts the class labels of new data points based on the majority class among their 'K' nearest neighbors. By incorporating cluster assignments as additional features, this combined approach aims to capture underlying patterns and enhance the separation of different classes, potentially improving the overall performance of the final model. It's crucial to carefully select the number of clusters in K-Means and thoroughly evaluate the impact on model performance through techniques like cross-validation.

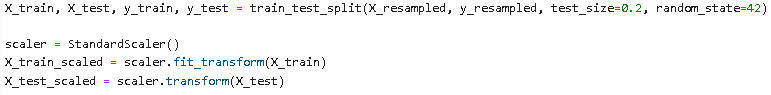
**Hyperparameter Tuning approach used:** Grid Search CV

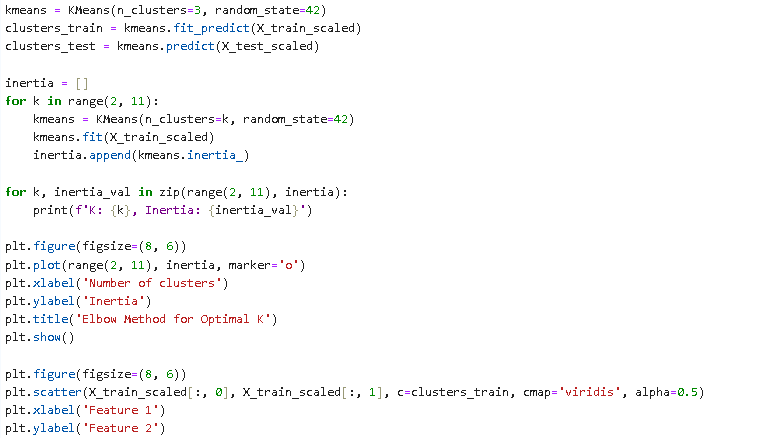
**Kmeans with KNN.ipynb:**

****

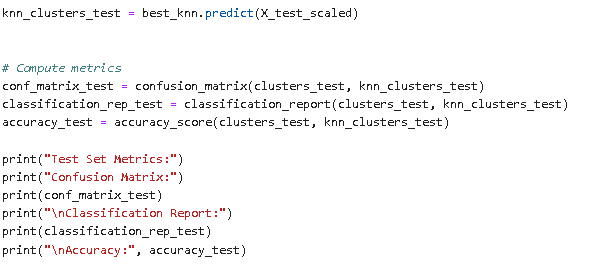
****

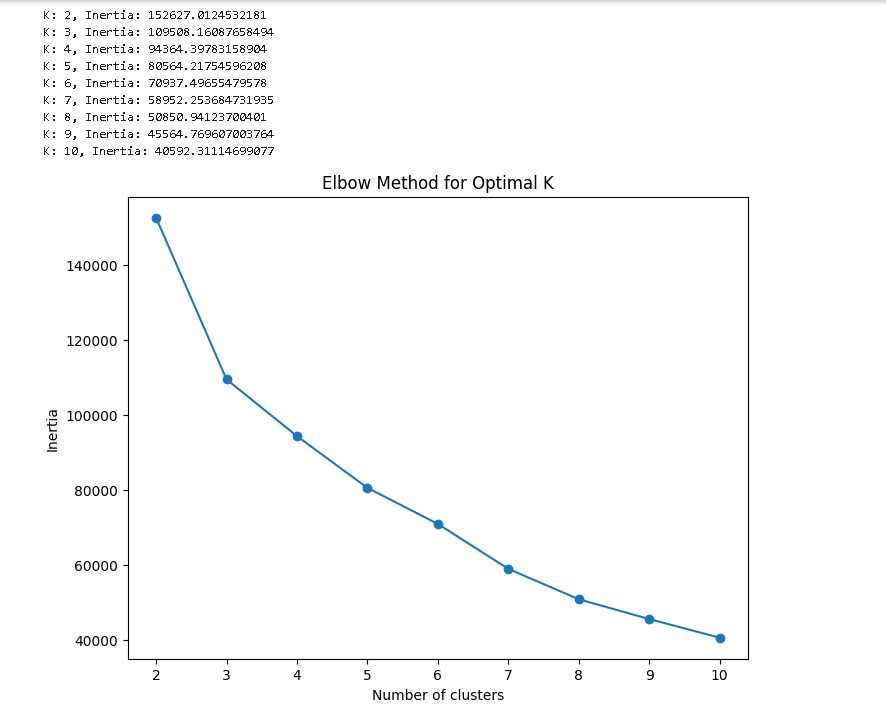
****

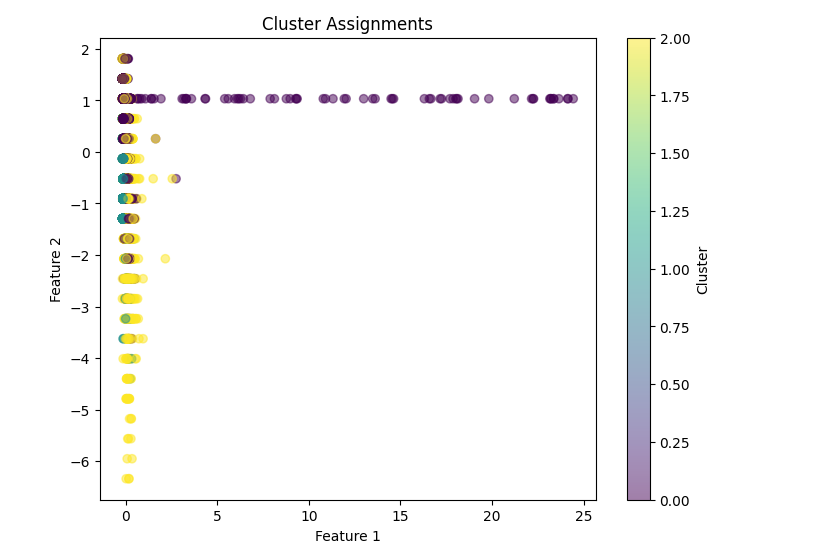
****

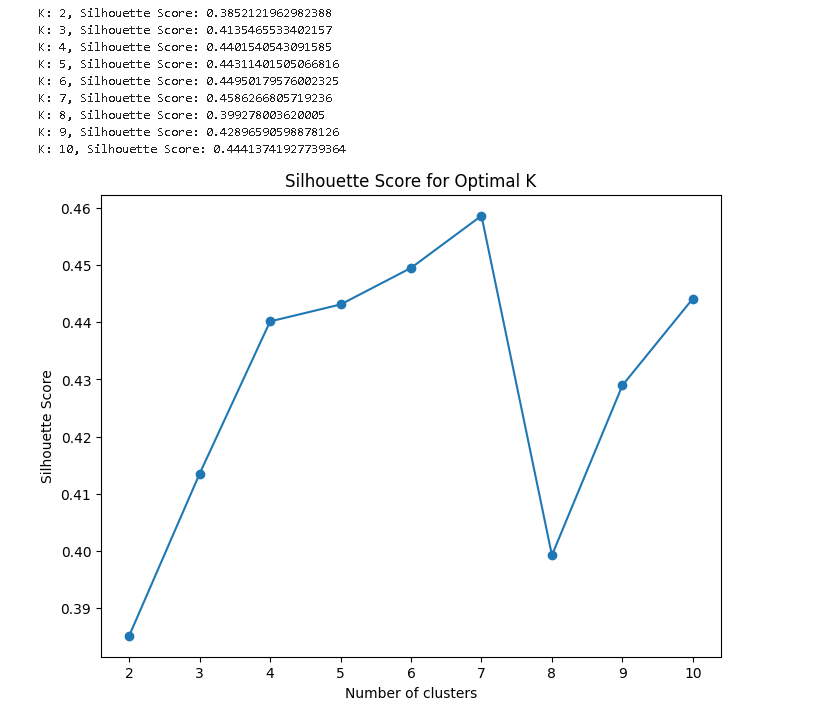
****

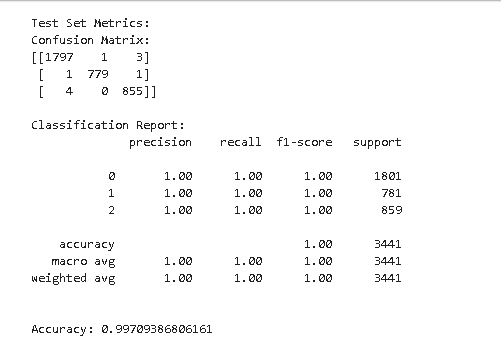
****

****

****

****

****

****

**Inference:**

* This Python program is a machine learning pipeline that involves clustering and classification tasks. It uses the scikit-learn library for various machine learning operations.
* The dataset is loaded from a CSV file ("dataset/resampled-train.csv") using the pandas library. The features (X\_resampled) and target variable (y\_resampled) are then extracted from the dataset. The dataset is split into training and testing sets using the train\_test\_split function.
* Standard scaling is applied to the features using StandardScaler to normalize the data. K-means clustering is then performed on the training set with a specified number of clusters (3 in this case). The Elbow Method is employed to determine the optimal number of clusters by iterating through different values of k and plotting the inertia (within-cluster sum of squares) for each k.
* The program also visualizes the cluster assignments on a scatter plot using the first two scaled features. Silhouette scores are computed to assess the quality of the clustering for different values of k.
* Next, a K-nearest neighbors (KNN) classifier is trained on the clusters obtained from K-means. Grid search is used to find the best hyperparameters for the KNN model. The optimal KNN model is then trained on the scaled training set, and predictions are made on the scaled test set.
* Finally, the program evaluates the performance of the KNN model on the test set by computing and printing metrics such as confusion matrix, classification report, and accuracy.
* The code is designed to analyze and model a dataset, specifically focusing on clustering with K-means and classification with KNN, providing insights into the structure of the data and assessing the predictive performance of the model.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| No. of Clusters | Number of the centroid which will be formed | 2 |
| Grid Search CV | The grid search explores various alpha values using 5-fold cross-validation. | n\_neighbors: 3, 5, 7, 9 |

**Parameters evaluated in the model:**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Inertia | Inertia, in the context of K-means clustering, refers to the sum of squared distances between each data point and its assigned cluster centroid. | 40592.311 |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.997 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1797 ,1 , 3] ,[ 1 ,779, 1] [ 4, 0, 855]] |
| Silhouette Score | The Silhouette Score is a metric used to calculate the goodness of a clustering technique, such as K-means clustering. It measures how well-defined the clusters are within the data. | 0.4441 |

**5.1.2 Fuzzy C-Means with KNN :**

Fuzzy C-Means (FCM) and K-Nearest Neighbors (KNN) are two distinct machine learning techniques, and they are typically used for different purposes. However, it's possible to combine these methods in certain scenarios.

Fuzzy C-Means is a clustering algorithm that is an extension of the classic K-Means clustering algorithm. While K-Means assigns each data point to exactly one cluster, Fuzzy C-Means assigns a degree of membership to each data point for every cluster. The degree of membership represents the probability or likelihood of a data point belonging to a particular cluster. FCM allows for soft clustering, where data points can belong to multiple clusters simultaneously with different degrees of membership.

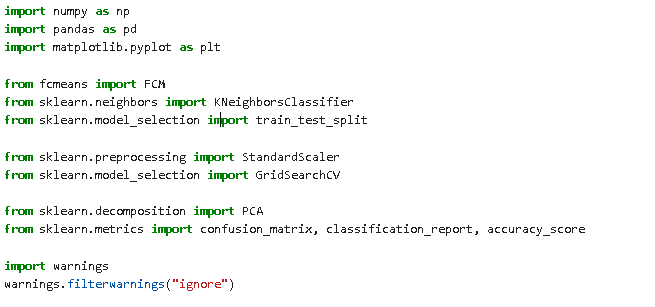
K-Nearest Neighbors (KNN):\*\* K-Nearest Neighbors is a supervised learning algorithm used for classification and regression tasks. It classifies or predicts the target variable of a data point by considering the majority class or average value of its k-nearest neighbors in the feature space.

While FCM is mainly used for clustering and KNN for classification, there may be scenarios where you want to incorporate the results of clustering into a classification task. For example, you might use FCM to perform soft clustering on your data and then use the cluster memberships as additional features for a KNN classifier. This way, the KNN classifier can take into account not only the original features but also the information about the soft cluster memberships assigned by FCM.

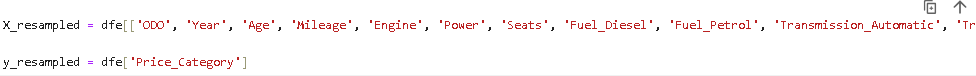
Combining FCM with KNN would involve using FCM to generate cluster memberships and then training a KNN classifier on the original features along with the cluster memberships. The resulting model would be able to classify data points based on both the original feature space and the information from the fuzzy clustering.

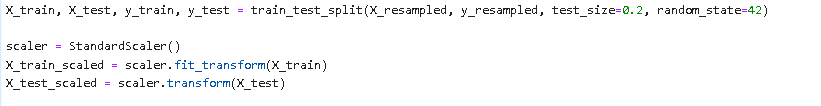
**Hyperparameter Tuning approach used:** Grid Search CV

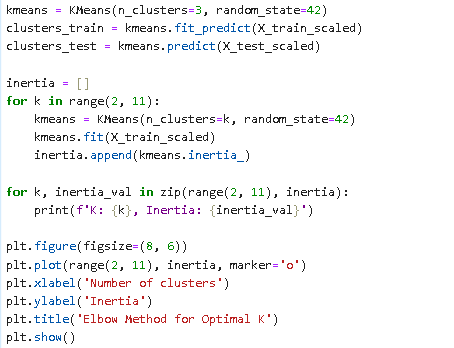
**Fuzzy C-Means with KNN.ipynb:**

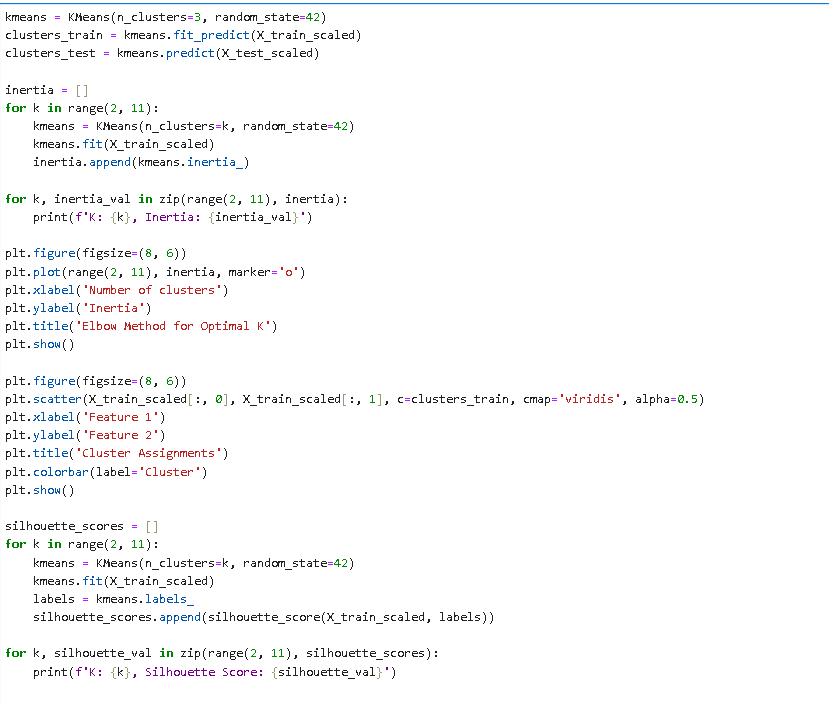
****

****

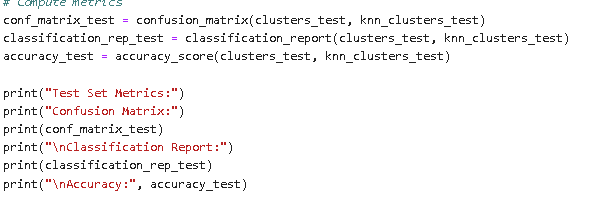
****

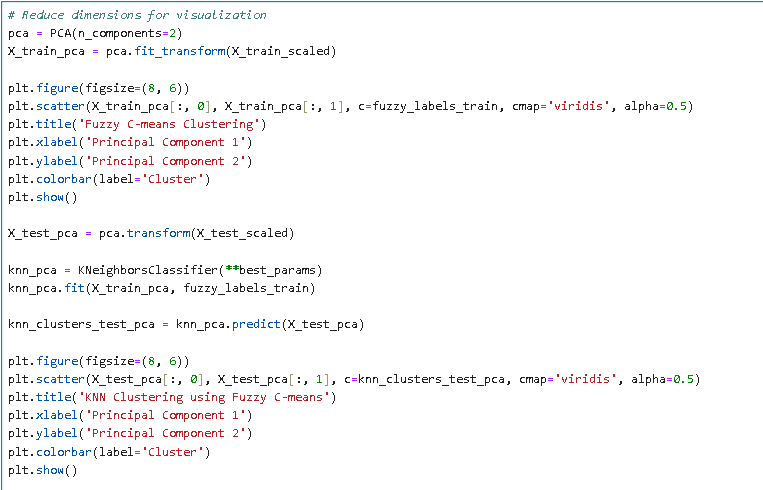
****

****

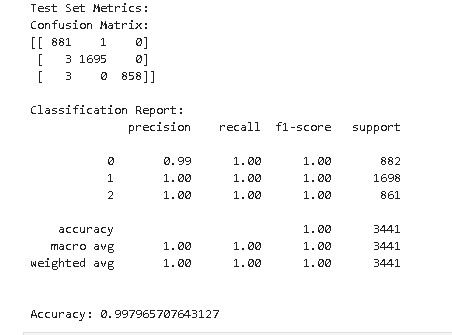
****

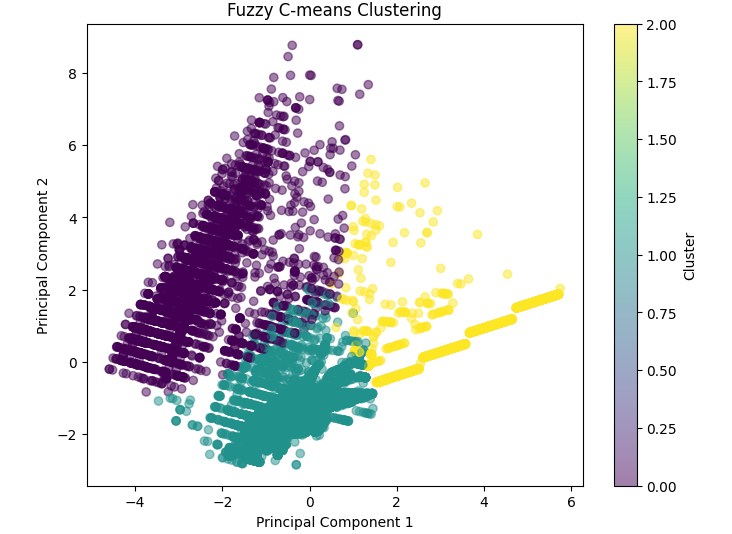
****

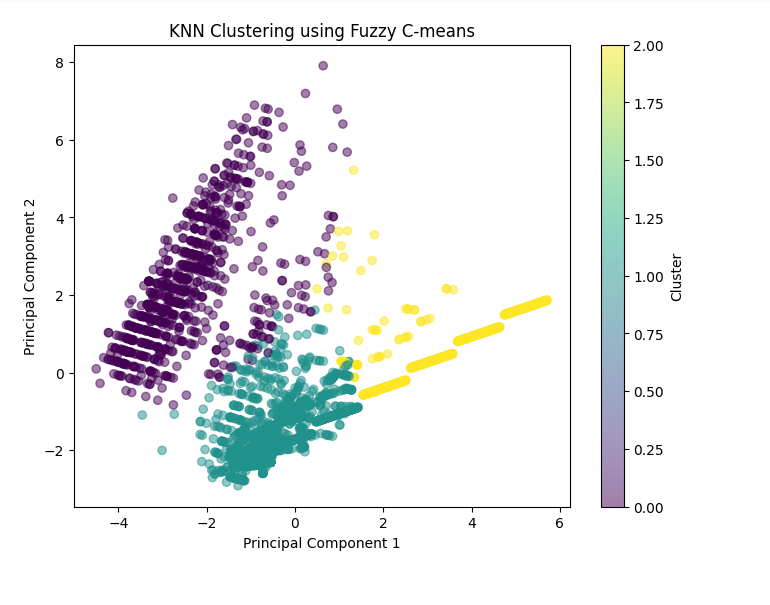
****

****

**Output:**

****

****

****

**Inference:**

* This Python program combines Fuzzy C-means (FCM) clustering with K-Nearest Neighbors (KNN) classification for a dataset loaded from "dataset/resampled-train.csv".
* It begins by extracting features and the target variable from the dataset. The data is then split into training and testing sets using the train\_test\_split function. Standard scaling is applied to normalize the features.
* The FCM clustering is performed with the fcmeans library, creating three fuzzy clusters. Fuzzy labels are assigned to both the training and testing sets.
* These fuzzy labels are then used as cluster assignments for subsequent KNN classification. A KNN classifier is trained using GridSearchCV to find the optimal hyperparameters, and predictions are made on the test set.
* The program evaluates the performance of the KNN model by computing metrics such as confusion matrix, classification report, and accuracy. Additionally, it uses Principal Component Analysis (PCA) to reduce the dimensionality of the data for visualization purposes.
* The fuzzy clusters obtained from FCM are visualized in a scatter plot using the first two principal components.
* Following the same approach, a KNN classifier is trained on the reduced-dimensional data and used to predict clusters for the test set.
* The results are visualized in another scatter plot, illustrating the clusters obtained by KNN using Fuzzy C-means. These visualizations provide insights into how well the clustering and classification algorithms perform on the data and can aid in understanding the structure of the dataset.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| No. of Clusters | Number of the centroid which will be formed | 3 |
| Grid Search CV | The grid search explores various alpha values using 5-fold cross-validation. | 'priors': [None, [0.1, 0.9], [0.3, 0.7], [0.5, 0.5], [0.7, 0.3], [0.9, 0.1]] |

**Parameters evaluated in the model:**

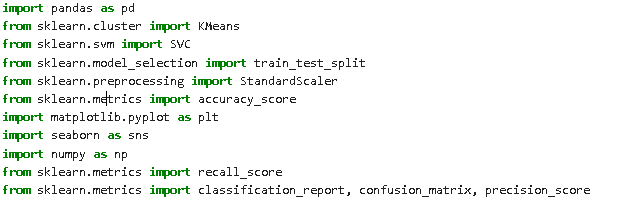
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.997 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[ 881, 1, 0],  [ 3, 1695, 0]  [ 3, 0, 858]] |

**5.1.3 K-Means with SVM :**

Combining K-Means clustering with Support Vector Machines (SVM) can be a powerful strategy for certain machine learning tasks. K-Means, an unsupervised clustering algorithm, is employed initially to group data points into clusters based on their similarity. Each data point is assigned to a cluster, providing a form of dimensionality reduction by capturing underlying patterns in the data. Following K-Means clustering, a new feature is engineered, representing the cluster assignments obtained from K-Means. This new feature, combined with the original features, serves as input for training an SVM classifier. SVM, a supervised learning algorithm, constructs an optimal hyperplane to maximize the margin between different classes. By incorporating the cluster assignments as additional features, the combined approach aims to enhance the separation of classes, potentially improving the overall performance of the SVM model. It is crucial to carefully choose the number of clusters in K-Means and thoroughly evaluate the impact on model performance through techniques such as cross-validation and validation metrics. The effectiveness of this approach depends on the nature of the data and the specific problem being addressed.

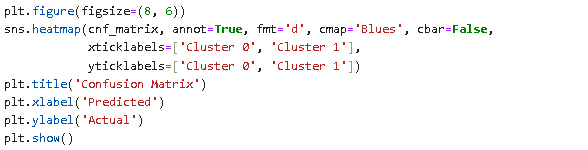
**Hyperparameter Tuning approach used:** Regularization parameter

**KmeanswithSVM.ipynb:**

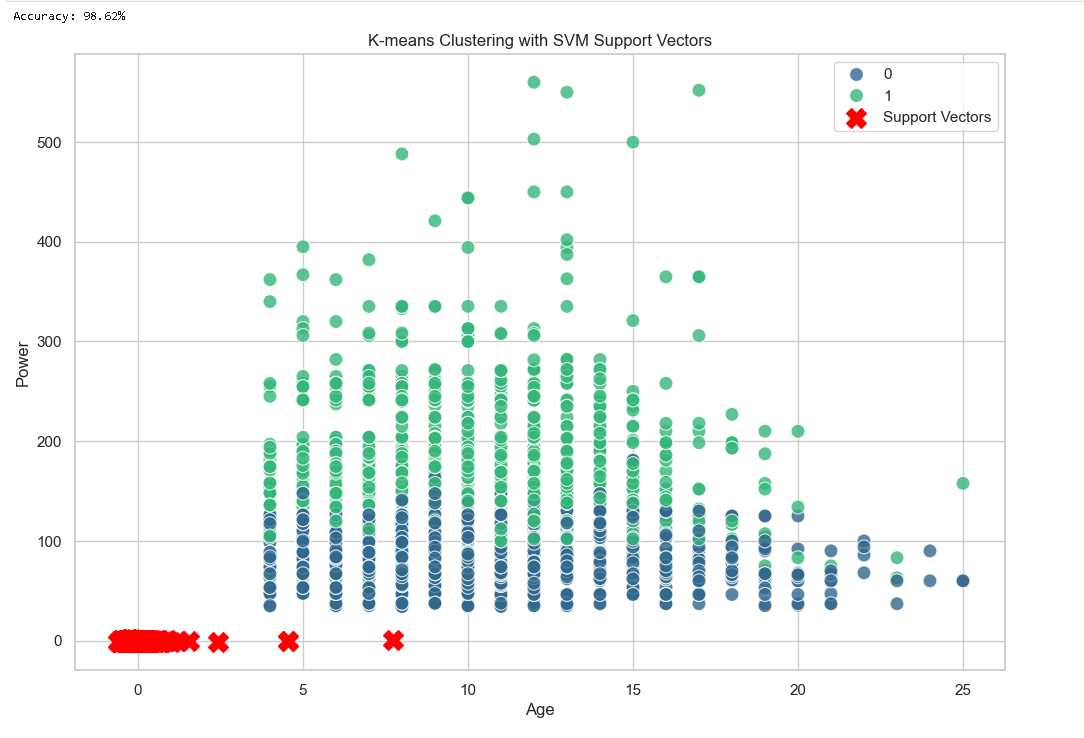
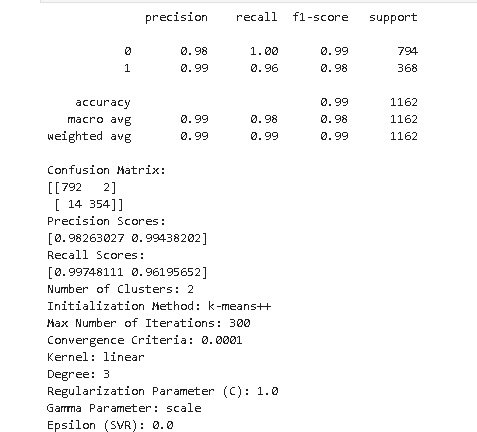
****

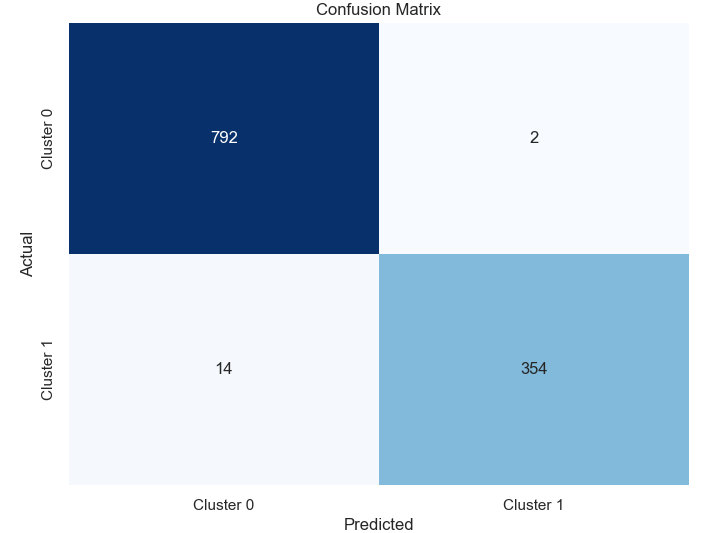
****

****

****

**Output:**

** **

****

**Inference:**

* The provided Python program is designed for a comprehensive analysis of a dataset using two main techniques: K-Means clustering and Support Vector Machine (SVM) classification.
* The program starts by loading a dataset from a CSV file named 'clean-train.csv' using the Pandas library. The dataset is then stored in a DataFrame (`df`). To ensure consistent scales across features, feature scaling is performed using `StandardScaler` specifically on a subset of features, including 'ODO', 'Year', 'Age', 'Mileage', 'Engine', 'Power', 'Seats', 'CP', and 'SP'.
* The scaled features are subjected to K-Means clustering with two clusters (`n\_clusters=2`). The resulting cluster assignments are incorporated into the DataFrame as a new column named 'Cluster'. This step groups similar data points together based on their features.
* The data is then divided into two separate DataFrames (`cluster\_0` and `cluster\_1`) based on the cluster assignments obtained from K-Means. This allows for further analysis and examination of each cluster independently.
* The dataset is split into training and testing sets using `train\_test\_split`. SVM is employed for classification with a polynomial kernel of degree 3 (`kernel='poly'`). The SVM model is trained on the training set (`X\_train, y\_train`) and subsequently used to make predictions on the test set (`X\_test`), stored in the `predictions` variable.
* The accuracy of the SVM classifier is computed using the `accuracy\_score` function from scikit-learn, comparing the predicted values (`predictions`) with the true labels (`y\_test`).
* A scatter plot is generated to visualize the K-Means clustering results using the 'Age' and 'Power' features. Support vectors from the SVM model are highlighted on the same plot, providing insight into the decision boundary.
* A series of performance metrics are printed, including the classification report, confusion matrix, precision scores, and recall scores. These metrics offer a detailed understanding of the SVM classifier's performance on the test set.
* Information about the K-Means clustering, such as the number of clusters, initialization method, maximum number of iterations, and convergence criteria, is printed. Similarly, details about the SVM model, including the kernel type, degree, regularization parameter (C), gamma parameter, and epsilon for Support Vector Regression (SVR), are also printed.
* The confusion matrix is visualized using a heatmap to provide a clear representation of the true positive, true negative, false positive, and false negative predictions made by the SVM model.
* The program leverages various Python libraries, including scikit-learn for machine learning tasks, Pandas for data manipulation, Matplotlib for plotting, Seaborn for enhanced visualization, and NumPy for numerical operations.
* In summary, the program integrates preprocessing, clustering, classification, visualization, and model evaluation, offering a comprehensive approach to exploring and understanding the dataset.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Kernel | kernel is a function that computes the dot product of two data points in a transformed feature space. | Poly |
| No. of Clusters | Number of the centroid which will be formed | 2 |
| Regularization Parameter | The regularization parameter, often denoted as C, is a crucial hyperparameter in machine learning algorithms, particularly in models like Support Vector Machines (SVM), logistic regression, and linear regression. | 1.0 |
| kernel | In Support Vector Machines (SVM), the kernel is a crucial component that determines the form of the decision boundary that the algorithm creates to separate different classes. | linear |

**Parameters evaluated in the model:**

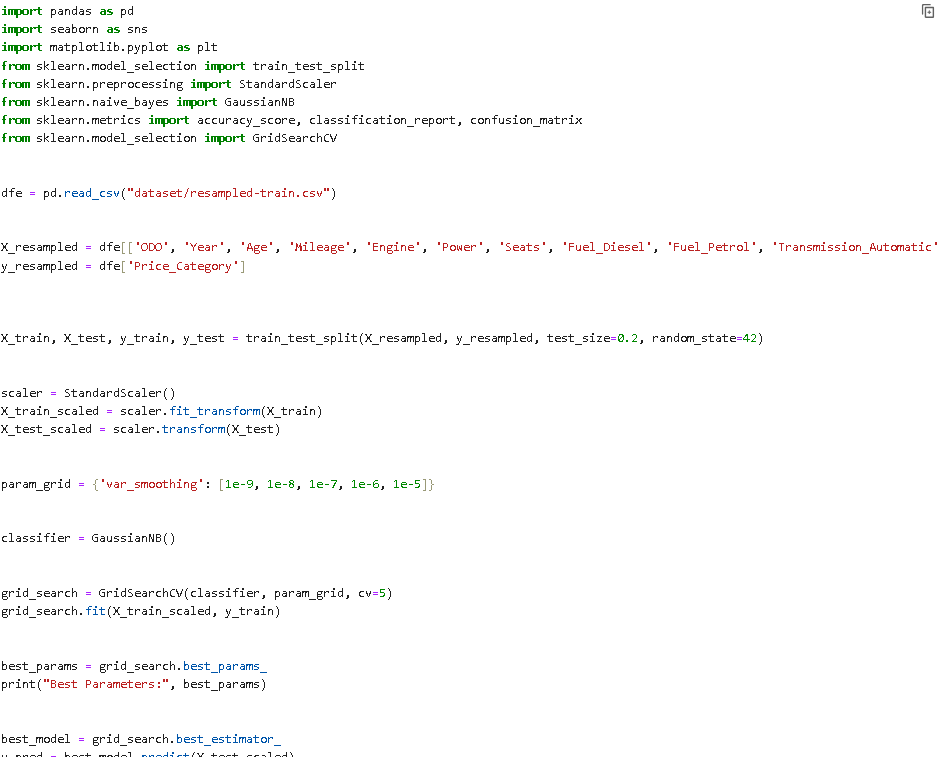
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter Name | Purpose | | | Value | |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | | | 0.98 | |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | | | [[792, 2]  [ 14, 354]] | |
| Precision Scores | | The precision scores in the context of a classification problem represent the accuracy of positive predictions made by a model for each class. | [0.98263027, 0.99438202] | |
| Recall Scores | | The recall scores, also known as sensitivity or true positive rate, measure the ability of a classification model to capture and correctly identify instances of a particular class. | [0.99748111, 0.96195652] | |
| Degree | | In the context of Support Vector Machines (SVM), specifically when using a polynomial kernel, the degree parameter represents the degree of the polynomial used by the kernel function | 3 | |

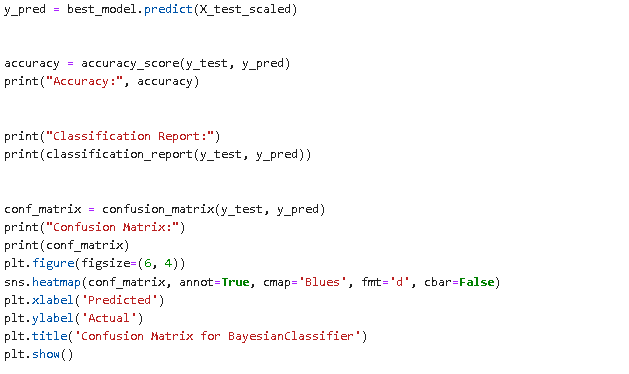
**5.1.4 Bayesian Classifier:**

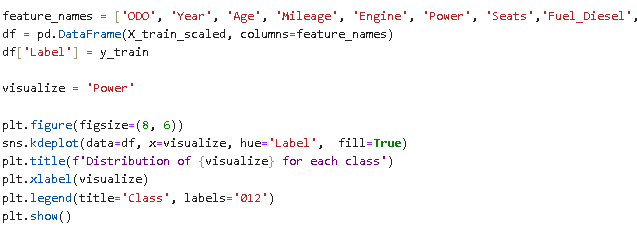
A Bayesian classifier is a statistical model that utilizes Bayes' theorem to make probabilistic predictions or classifications based on observed evidence. It is particularly useful in the field of machine learning and data mining. The classifier calculates the probability of a hypothesis or class given the available data by combining prior knowledge with new evidence. In essence, it updates its beliefs about the likelihood of different outcomes as new information becomes available. The classifier begins with a prior probability, representing the initial belief in the likelihood of different classes. As new data is observed, the prior probability is adjusted using Bayes' theorem, which takes into account the likelihood of observing the data given each class and the marginal likelihood of the data. This iterative process refines the classifier's predictions, making it a powerful tool for handling uncertainty and making informed decisions in various applications, such as spam filtering, medical diagnosis, and sentiment analysis.

**Hyperparameter Tuning approach used:** Smoothing parameter

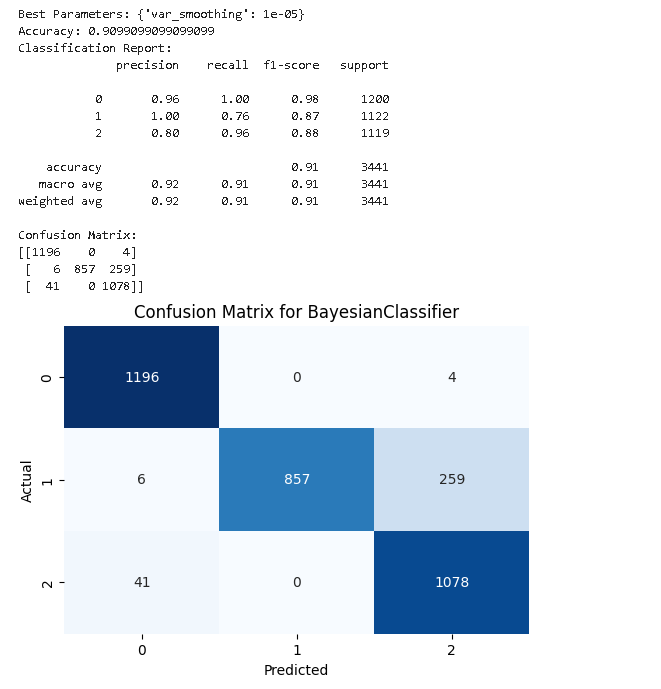
**BayesianClassifier.ipynb:**

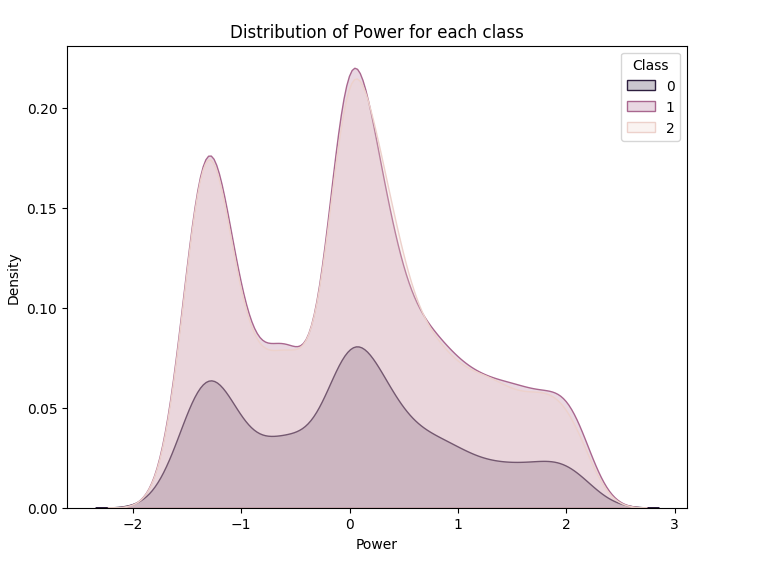
****

****

****

**Output:**

****

****

**Inference:**

* The provided Python program is a comprehensive implementation of a Gaussian Naive Bayes classifier using the scikit-learn library, tailored for a dataset related to automotive pricing.
* The initial steps involve importing essential libraries, including pandas for efficient data manipulation, seaborn and matplotlib for data visualization, and scikit-learn for machine learning functionalities.
* The dataset, residing in the file "resampled-train.csv," is loaded into a pandas DataFrame (`dfe`).
* This dataset presumably contains various automotive features such as odometer reading (ODO), year, age, mileage, engine specifications, power, seats, and categorical variables like fuel type, transmission, and owner history.
* After loading the data, the features (`X\_resampled`) and the target variable (`y\_resampled`) are separated.
* The dataset is then split into training and testing sets using the `train\_test\_split` function, with 80% of the data used for training and the remaining 20% for testing.
* To ensure consistent feature scaling, the program employs `StandardScaler` to standardize the feature sets (`X\_train\_scaled` and `X\_test\_scaled`).
* The subsequent steps focus on model training and evaluation. A grid search is conducted to identify the optimal hyperparameter for the Gaussian Naive Bayes classifier, specifically the `var\_smoothing` parameter.
* This parameter controls the smoothing applied to the covariance matrix, and the grid search explores different values to find the most suitable one. The best model is selected based on the grid search results.
* The program then evaluates the performance of the trained model on the testing set. It calculates accuracy using the `accuracy\_score` function and provides additional insights through a printed report, including the best hyperparameters, accuracy score, a classification report detailing precision, recall, and F1-score for each class, and a confusion matrix.
* The confusion matrix is further visualized using a heatmap, offering an intuitive representation of the model's ability to correctly classify instances and identify potential areas of improvement.
* Finally, the program explores the distribution of a specific feature, 'Power,' for each class in the training set. It achieves this by creating a DataFrame (`df`) with the scaled training features and corresponding labels, followed by a kernel density plot using `sns.kdeplot`. This visualization helps in understanding how the feature 'Power' varies across different classes.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |

|  |  |  |
| --- | --- | --- |
| Grid Search CV | The grid search explores various alpha values using 5-fold cross-validation. | n\_neighbors: 9,8,7,6,5 |
| smoothing parameter | This parameter is used to add a specified amount of smoothing to the variances of all features, especially when dealing with features that have a variance close to zero. The purpose of smoothing is to avoid numerical instability when calculating probabilities. | var\_smoothing |

**Parameters evaluated in the model:**

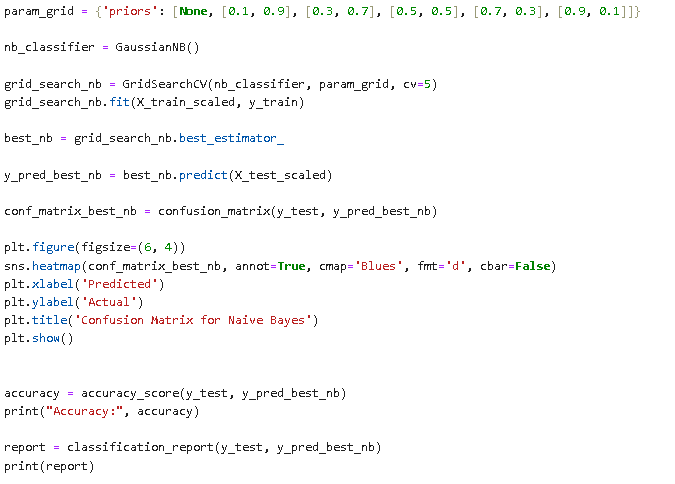
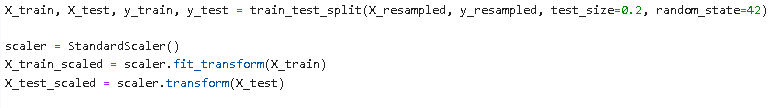
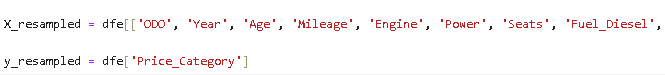
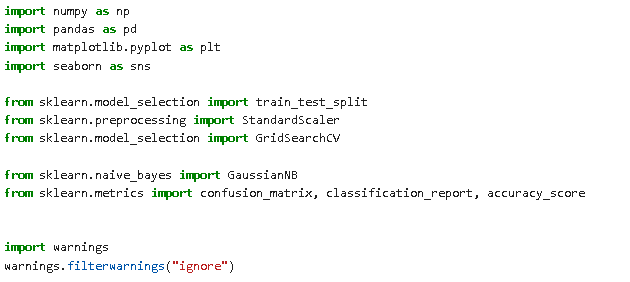
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.9099 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1196 , 0 , 4]  [ 6 , 857, 259]  [ 41 , 0, 1078]] |

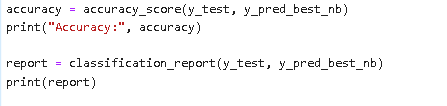
**5.1.5 Naive Bayes Classifier:**

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem, which is used for classification tasks. It assumes that the features used to describe instances are conditionally independent given the class label. This is a simplifying assumption that can greatly reduce the computational complexity of the model. Despite its simplicity, Naive Bayes classifiers often perform surprisingly well in various real-world applications. The classifier calculates the probability of a particular instance belonging to a certain class by combining the prior probability of that class with the product of the conditional probabilities of each feature given the class. In other words, it estimates the likelihood of a class given the observed features. Naive Bayes is widely used in text classification, spam filtering, sentiment analysis, and other tasks where the independence assumption holds reasonably well. Its efficiency and effectiveness, particularly with high-dimensional data, make it a popular choice for certain applications in machine learning.

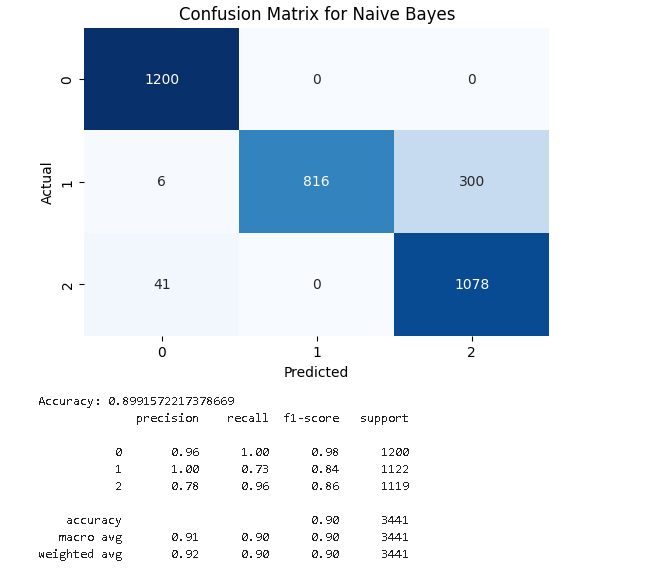
**Hyperparameter Tuning approach used:** Grid Search CV

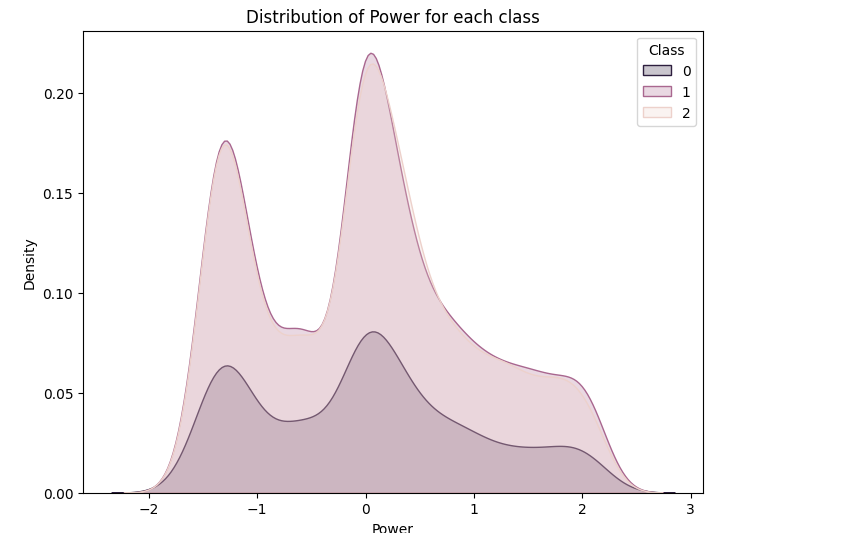
**Naive Bayes.ipynb:**

****



**Output:**





**Inference:**

* This Python program is designed for a classification task, specifically to predict the 'Price\_Category' in a dataset.
* It starts by importing essential libraries, including NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn modules for machine learning tasks.
* The dataset is loaded from a CSV file into a Pandas DataFrame, and relevant features ('ODO', 'Year', 'Age', etc.) are selected for the feature set (`X\_resampled`), while the target variable ('Price\_Category') is assigned to `y\_resampled`.
* The dataset is then split into training and testing sets using the `train\_test\_split` function. To ensure consistent scaling, the features are standardized using `StandardScaler`. Subsequently, a parameter grid is defined for hyperparameter tuning of the Gaussian Naive Bayes classifier.
* The `GridSearchCV` performs a cross-validated search, trying different combinations of prior probabilities for the Naive Bayes model on the training data.
* A Gaussian Naive Bayes classifier is instantiated, and the grid search is executed, identifying the best combination of hyperparameters.
* The program extracts the best estimator from the grid search results. Using this best model, it predicts the target variable on the test set and generates a confusion matrix.
* The confusion matrix is then visualized as a heatmap using Seaborn and Matplotlib. Additionally, the program calculates and prints the accuracy score and provides a detailed classification report, offering insights into the precision, recall, and F1-score for each class.
* In summary, this program encompasses data preprocessing, model training with hyperparameter tuning, model evaluation with a confusion matrix and classification report, and visualization of the confusion matrix—all crucial steps in developing and assessing a machine learning classifier.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Grid Search CV | The grid search explores various alpha values using 5-fold cross-validation. | n\_neighbors: 3, 5, 7, 9 |

**Parameters evaluated in the model:**

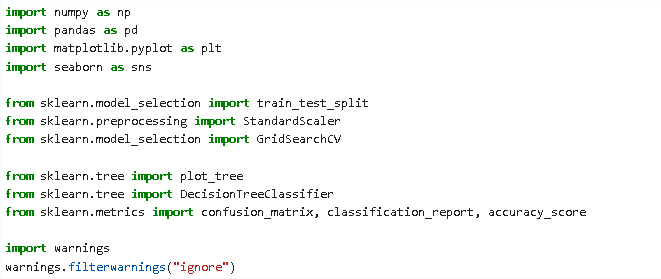
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.8991 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1200, 0 , 0]  [ 6 , 816, 300]  [ 41 , 0, 1078]] |

**5.1.6 Decision Tree:**

A Decision Tree is a versatile and interpretable machine learning algorithm used for both classification and regression tasks. The fundamental concept behind a Decision Tree is to recursively split the dataset into subsets based on the most significant feature at each node. These splits create a hierarchical tree structure where each internal node represents a decision based on a specific feature, and each leaf node corresponds to the final predicted outcome. The algorithm selects the best feature to split on by maximizing information gain or minimizing impurity measures, such as Gini impurity or entropy. Decision Trees are known for their simplicity, ease of interpretation, and ability to handle both numerical and categorical data. However, they are prone to overfitting, and techniques like pruning or ensemble methods, such as Random Forests, are often employed to enhance their performance and generalization to new, unseen data. Overall, Decision Trees are valuable tools in machine learning due to their transparency and applicability in various domains.

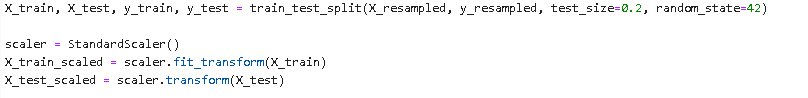
**Hyperparameter Tuning approach used:** Grid Search CV

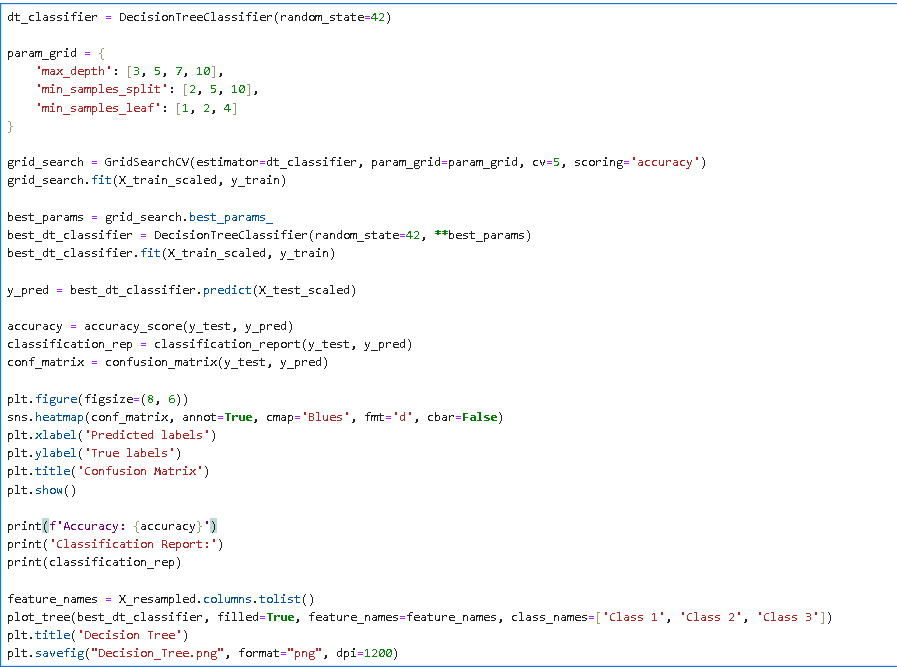
**Decision Tree Classifier.ipynb:**



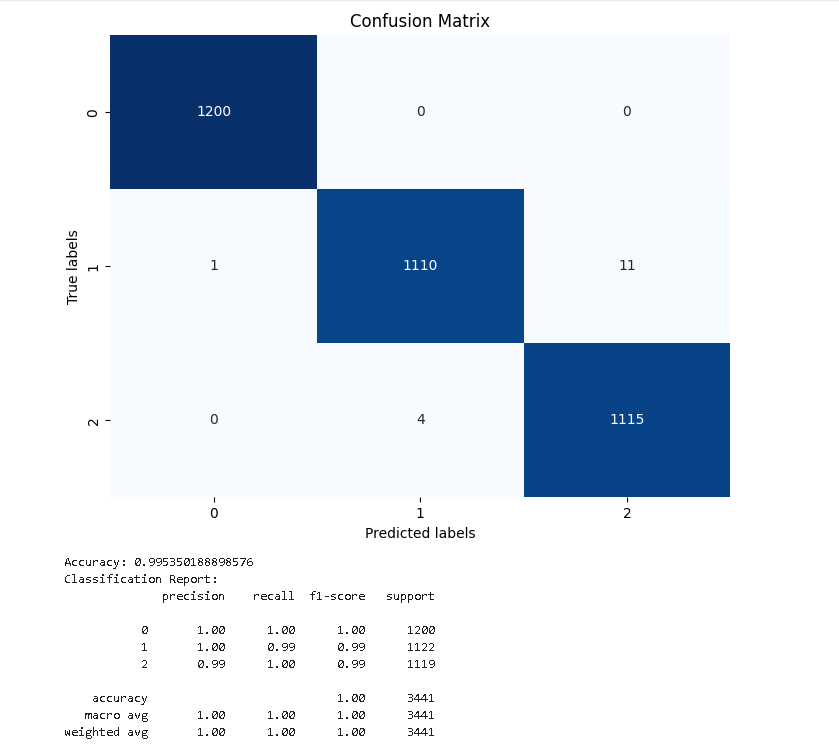


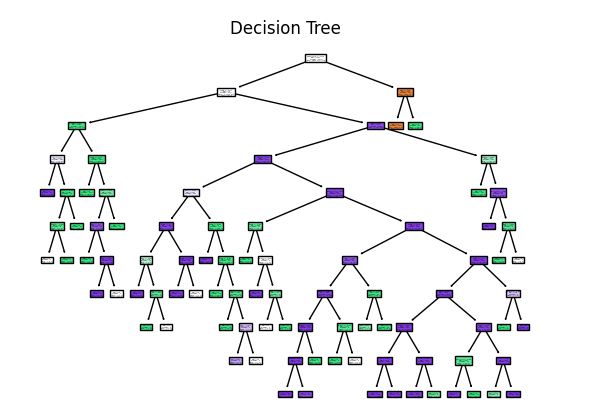






**Output:**



****

**Inference:**

* This Python program employs a Decision Tree classifier for a classification task, aiming to predict the 'Price\_Category' of a dataset.
* The code begins by importing necessary libraries such as NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn modules.
* The dataset is loaded from a CSV file, and relevant features and the target variable are extracted. The data is then split into training and testing sets using the `train\_test\_split` function, and the features are standardized using `StandardScaler`.
* A Decision Tree classifier is instantiated, and hyperparameter tuning is performed using grid search with cross-validation (`GridSearchCV`).
* The hyperparameters considered include the maximum depth of the tree, minimum samples required to split an internal node (`min\_samples\_split`), and minimum samples required in a leaf node (`min\_samples\_leaf`). The best hyperparameters are determined, and a new Decision Tree classifier is trained with these optimal settings.
* The program then predicts the target variable on the test set, calculates the accuracy, generates a classification report, and constructs a confusion matrix. The confusion matrix is visualized as a heatmap using Seaborn and Matplotlib. Additionally, the Decision Tree is plotted and saved as an image file ('Decision\_Tree.png') for better interpretability.
* In summary, this program demonstrates a comprehensive pipeline for training, optimizing, and evaluating a Decision Tree classifier for a specific dataset. It showcases hyperparameter tuning, model evaluation metrics, and visualization techniques to enhance understanding and interpretation of the model's performance.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Grid Search CV | The grid search explores various alpha values using 5-fold cross-validation. | 'max\_depth': [3, 5, 7, 10], 'min\_samples\_split': [2, 5, 10], 'min\_samples\_leaf': [1, 2, 4] |

**Parameters evaluated in the model:**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.99535 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1200, 0 , 0]  [ 1, 1110, 4]  [ 0 , 4, 1115]] |

**SECTION 6**

**Deep learning:**

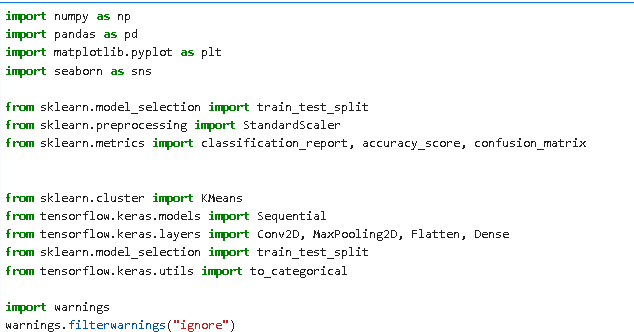
Deep learning is a subset of artificial intelligence that focuses on training machines to learn and make decisions in a manner akin to the human brain. At its core are neural networks, computational models inspired by biological neural networks, comprising layers of interconnected nodes or artificial neurons. These networks can be deep, consisting of multiple hidden layers, enabling them to automatically learn hierarchical representations of data. To train deep learning models, large amounts of labelled data are required, allowing the model to adjust its parameters by minimizing the difference between predicted and actual outputs during the training process. Activation functions introduce non-linearities, crucial for learning complex relationships in the data. Backpropagation, a key algorithm, is employed to update the model's weights and biases based on the training error. Convolutional Neural Networks (CNNs) are specialized for grid-like data, particularly images, using convolutional layers. Recurrent Neural Networks (RNNs) handle sequential data through loops in their architecture. Transfer learning is a common strategy, leveraging pre-trained models on large datasets for tasks with smaller datasets. The applications of deep learning span various domains, including image and speech recognition, natural language processing, autonomous vehicles, and medical diagnosis, showcasing its effectiveness in automating complex tasks and extracting meaningful patterns from data.

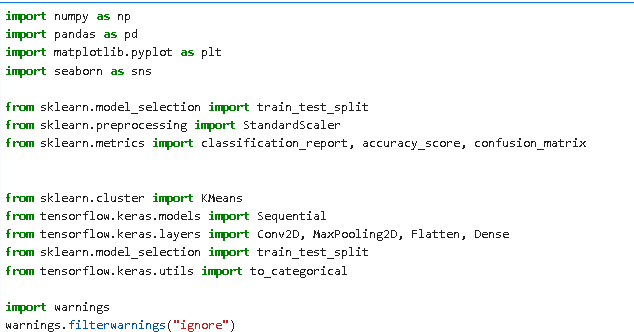
**6.1 K-Means with Convolutional Neural Network:**

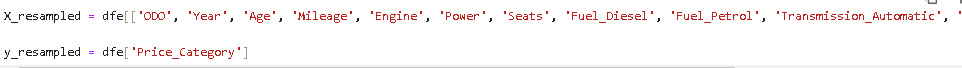
In a synergistic approach to machine learning, K-Means clustering and Convolutional Neural Networks (CNNs) can be integrated to address specific challenges, particularly in scenarios involving vast image datasets. K-Means, an unsupervised clustering algorithm, takes on the initial task of grouping images based on pixel values or extracted features. This clustering effectively categorizes images into distinct visual patterns or similarities, allowing for a structured organization of the data. Subsequently, a Convolutional Neural Network, tailored for tasks like image recognition, benefits from this pre-processing step. Instead of training on the entire dataset, the CNN is trained on the clusters identified by K-Means, treating each cluster as a pseudo-class. This approach serves as a form of semi-supervised learning, where the CNN can leverage the structured information provided by K-Means to enhance its performance. Through this combined strategy, the challenges posed by large, unlabelled datasets are mitigated, and the CNN can be fine-tuned on the labelled data to excel in specific tasks such as image classification or object detection.

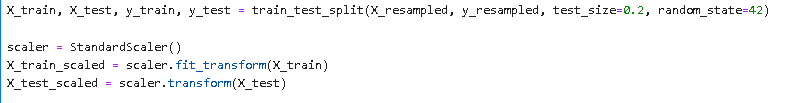
**Hyperparameter Tuning approach used:** Grid Search CV,n\_clusters

**Kmeans with CNN.ipynb:**





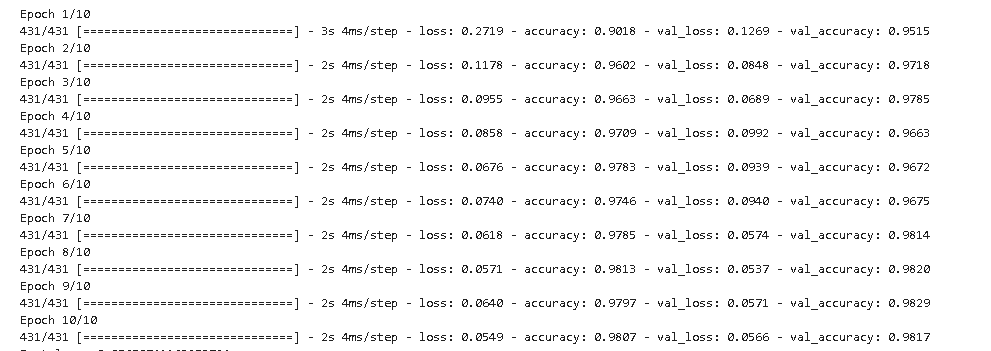


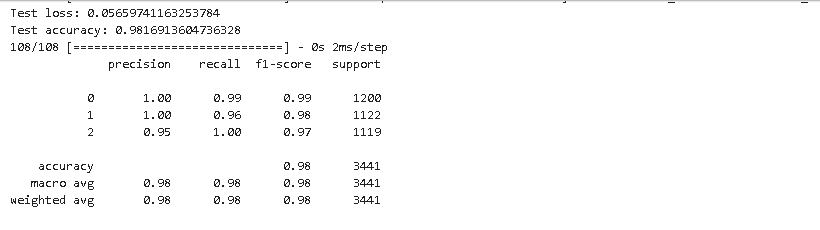


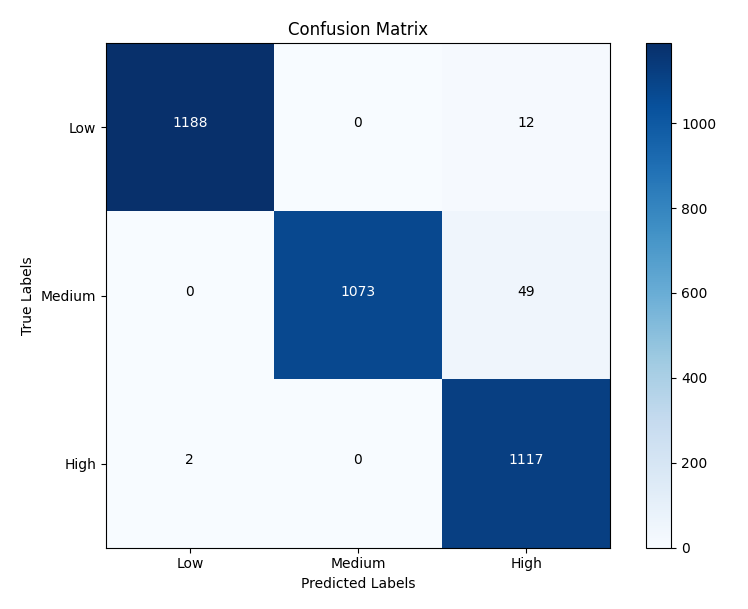




**Output:**

****





**Inference:**

* This Python program leverages machine learning techniques, particularly K-Means clustering and Convolutional Neural Networks (CNNs), to classify a dataset of resampled car data into price categories.
* The initial steps involve importing necessary libraries, such as NumPy, Pandas, Matplotlib, Seaborn, and various modules from scikit-learn and TensorFlow.
* The dataset, loaded from a CSV file, includes features like ODO, Year, Age, Mileage, Engine, Power, Seats, and categorical variables related to Fuel, Transmission, and Owner details.
* The data is split into training and testing sets, and standard scaling is applied to normalize the features. K-Means clustering is then employed with 64 clusters, and the transformed data is reshaped into a grid structure.
* This clustered data is fed into a Convolutional Neural Network (CNN), which consists of convolutional, max-pooling, flattening, and dense layers. The model is compiled with categorical cross entropy loss and the Adam optimizer.
* The CNN is trained on the clustered data, and its performance is evaluated on the testing set. The program outputs the test loss and accuracy.
* Additionally, it generates a classification report and a confusion matrix to assess the model's performance in predicting price categories (Low, Medium, High).
* The confusion matrix is visualized using Matplotlib, providing insights into the true and predicted labels, helping to identify the model's strengths and weaknesses in classifying car prices.
* The program integrates K-Means clustering as a preprocessing step to capture underlying patterns in the data, which the CNN then utilizes for improved classification. Finally, warnings are filtered to maintain code readability.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Epoch | In the context of machine learning, especially neural network training, an epoch refers to one complete pass through the entire training dataset during the training phase. In other words, it represents a single iteration over the entire set of training samples. During each epoch, the model's parameters (weights and biases) are adjusted based on the computed gradients of the loss function, aiming to minimize the difference between the predicted outputs and the actual targets. | 10 |
| Batch size | In machine learning, particularly in training neural networks, the batch size is a hyperparameter that defines the number of training samples utilized in one iteration. During each iteration, the model's parameters are updated based on the gradients computed from the selected batch of training data. | 32 |
| Activation function | An activation function is a mathematical operation applied to the output of each neuron in a neural network. It introduces non-linearity to the network, allowing it to learn complex patterns and relationships in the data. | relu |

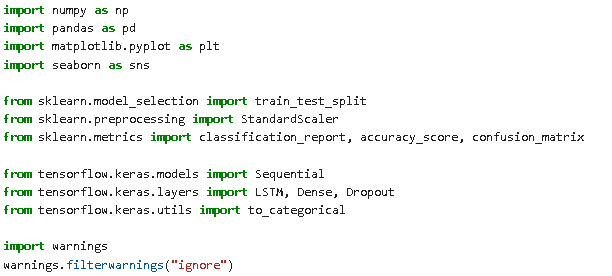
**Parameters evaluated in the model:**

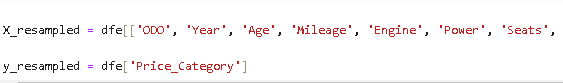
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.98169 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1188, 0 , 12]  [ 1, 1073, 49]  [ 2, 0, 1117]] |
| Test loss | The term "test loss" refers to the value of the loss function on a machine learning model when it is evaluated on a separate dataset that was not used during the training process. This separate dataset is typically called the "test set" or "holdout set," and it serves as a measure of the model's performance on unseen data. | 0.0565 |

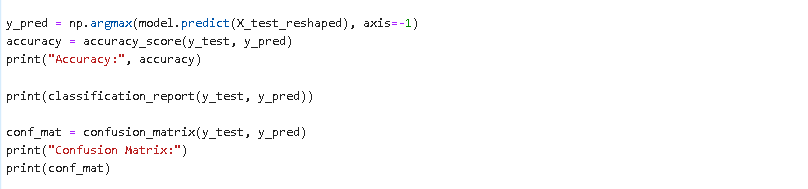
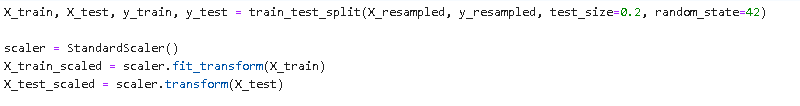
**6.2 LSTM** **:**

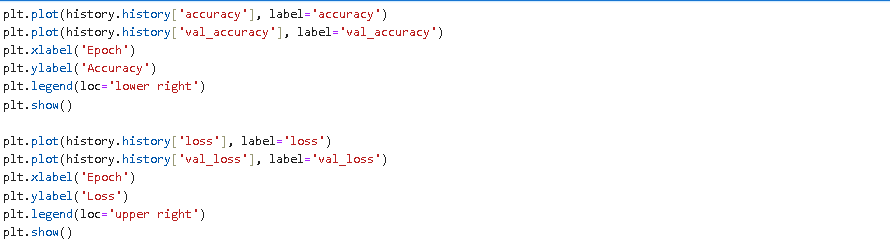
Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the challenges of capturing and learning long-term dependencies in sequential data. Unlike traditional RNNs, LSTM networks have a more sophisticated memory cell structure that enables them to retain and update information over extended sequences. The key innovation of LSTMs lies in their ability to selectively remember or forget information through specialized gates, namely the input gate, forget gate, and output gate. These gates regulate the flow of information, facilitating the learning of temporal patterns and dependencies in sequential data. The input gate controls the introduction of new information, the forget gate manages the removal of irrelevant information from the memory cell, and the output gate determines the information to be passed to the next time step. This mechanism mitigates the vanishing gradient problem, allowing LSTMs to capture dependencies over longer time horizons. LSTMs find widespread applications in tasks involving sequential data, such as natural language processing, speech recognition, and time-series prediction, where the ability to model and remember complex dependencies is crucial for accurate predictions.

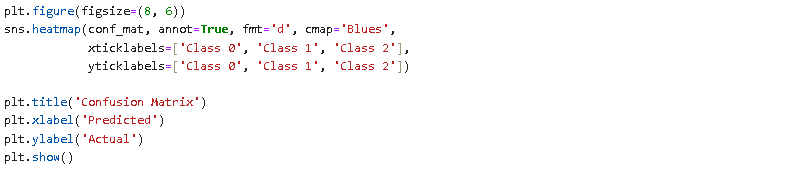
**LSTM.ipynb:**

****

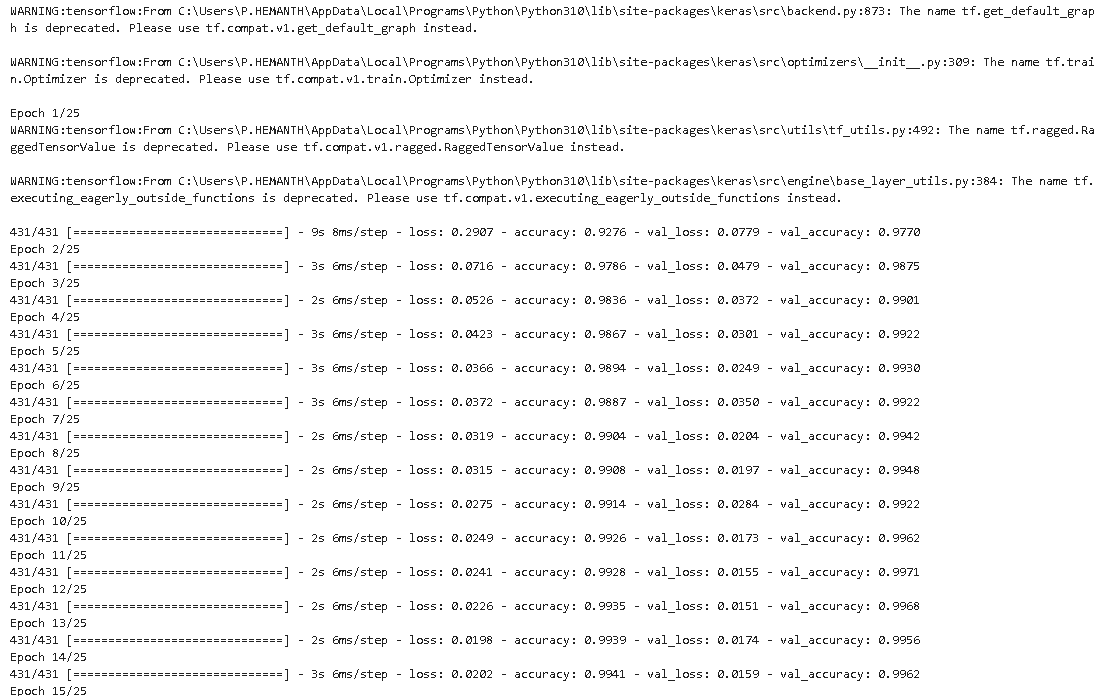


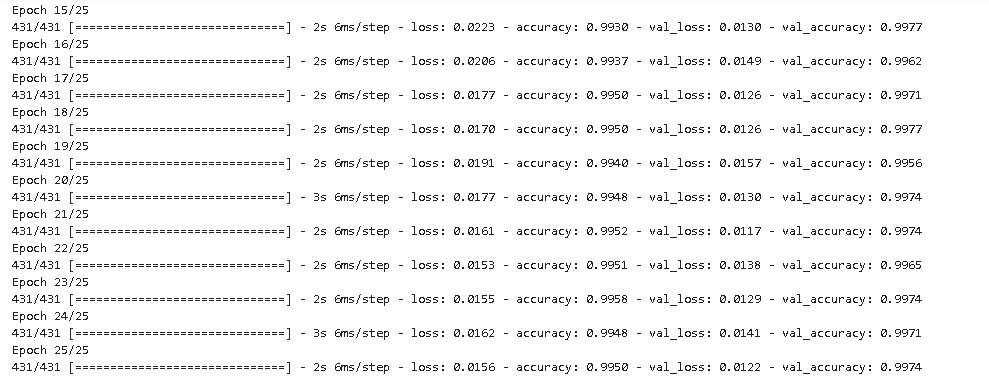


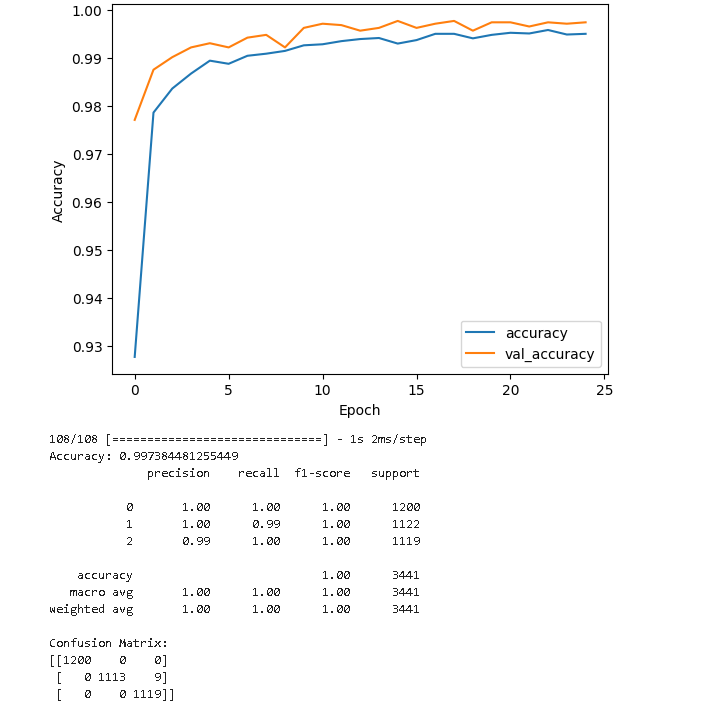


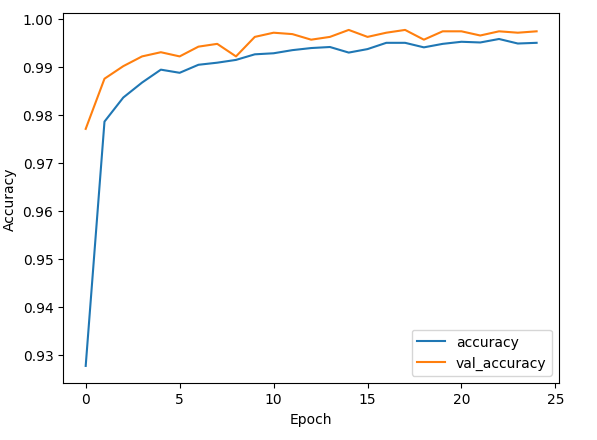


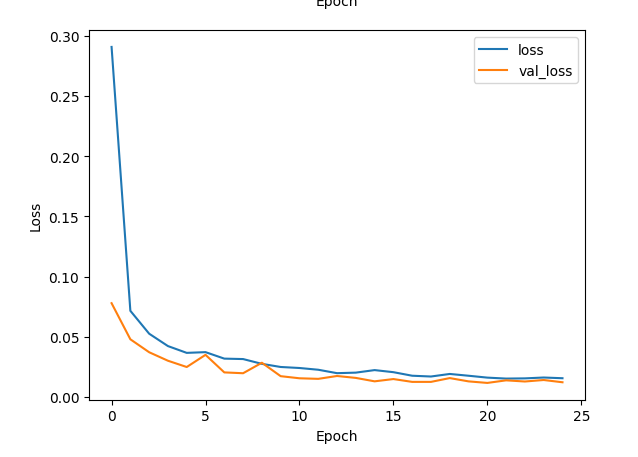
**Output:**

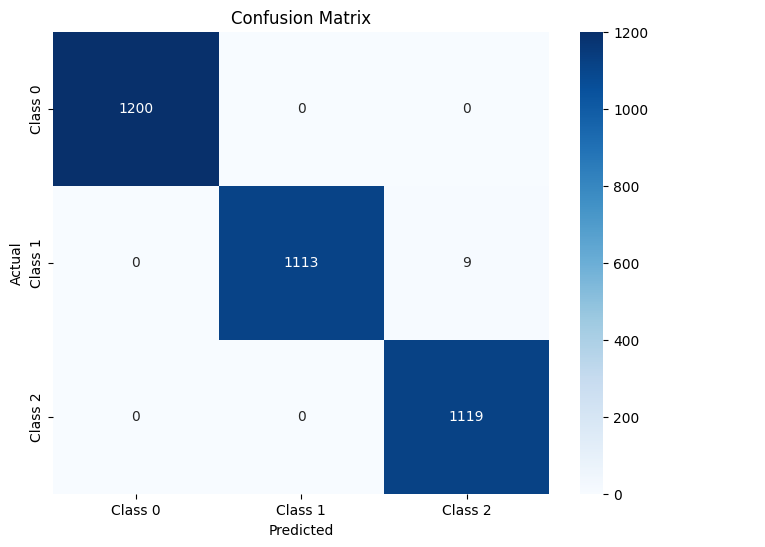
****

****

****







**Inference:**

* In this Python program, the implementation of a Long Short-Term Memory (LSTM) neural network is carried out using the TensorFlow and Keras libraries, specifically for a multi-class classification task.
* The initial steps involve loading a dataset in CSV format using the Pandas library. The dataset is then split into two subsets, one for training and the other for testing, using the `train\_test\_split` function from scikit-learn.
* The features (`X\_resampled`) and the target variable (`y\_resampled`) are extracted from the dataset to form the input for the subsequent modeling steps.
* To ensure uniformity in feature scaling, the program employs the `StandardScaler` from scikit-learn to standardize the features, resulting in a mean of 0 and a standard deviation of 1.
* The reshaping of the input data is performed to meet the requirements of the LSTM layer, transforming it into a three-dimensional array with dimensions `(batch\_size, time\_steps, features)`.
* The target variable, representing the classes for the classification task, undergoes one-hot encoding using the `to\_categorical` function. This process converts categorical labels into a binary matrix representation, facilitating the model's interpretation of multiple classes.
* The LSTM neural network model is constructed using the Keras Sequential API. The model architecture comprises three LSTM layers, each followed by a dropout layer.
* Dropouts are introduced to mitigate overfitting by randomly disabling a fraction of neurons during training. The final layer is a dense layer with a softmax activation function, suitable for multi-class classification tasks.
* The model is compiled with the Adam optimizer and categorical crossentropy as the loss function, setting the stage for training.
* The training phase involves fitting the model to the training data over 25 epochs with a batch size of 32. Throughout training, the model's performance is assessed using a validation set (`X\_test\_reshaped` and `y\_test\_cat`), allowing the evaluation of its generalization capabilities to unseen data.
* For insights into the model's learning process, the program utilizes Matplotlib to generate visualizations of the training history, depicting changes in accuracy and loss over the course of the epochs.
* Additionally, the accuracy, classification report, and confusion matrix are computed and printed for a comprehensive evaluation of the model's performance on the test set.
* In summary, this program encompasses various crucial steps in building and evaluating an LSTM model for multi-class classification, demonstrating the significance of data preprocessing, model architecture, training, and performance assessment through visualization and metrics.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Epoch | In the context of machine learning, especially neural network training, an epoch refers to one complete pass through the entire training dataset during the training phase. In other words, it represents a single iteration over the entire set of training samples. During each epoch, the model's parameters (weights and biases) are adjusted based on the computed gradients of the loss function, aiming to minimize the difference between the predicted outputs and the actual targets. | 25 |
| Batch size | In machine learning, particularly in training neural networks, the batch size is a hyperparameter that defines the number of training samples utilized in one iteration. During each iteration, the model's parameters are updated based on the gradients computed from the selected batch of training data. | 32 |
| Optimizer | An optimizer is a key component in training machine learning models, responsible for adjusting the model's parameters (weights and biases) during the training process to minimize the error or loss function. The primary goal of an optimizer is to find the optimal set of parameters that allows the model to make accurate predictions on new, unseen data. | adam |
| Activation Function | An activation function is a mathematical operation applied to the output of each neuron in a neural network. It introduces non-linearity to the network, allowing it to learn complex patterns and relationships in the data. | Sigmoid |
| LSTM Unit | In the context of Long Short-Term Memory (LSTM) networks, the term "LSTM unit" refers to the fundamental building block or cell structure of an LSTM layer. LSTMs are a type of recurrent neural network (RNN) designed to capture and learn long-term dependencies in sequential data. The LSTM unit is a specialized cell that allows the network to selectively store, read, and write information over extended sequences, addressing the vanishing gradient problem associated with traditional RNNs. | 100 |

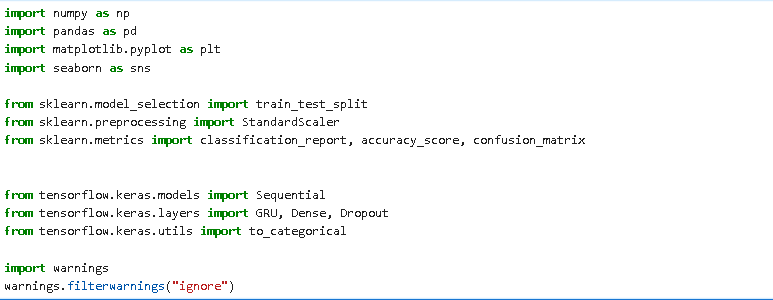
**Parameters evaluated in the model:**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.99738 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1200, 0, 0]  [ 0, 1113, 9]  [ 0, 0, 1119]] |
| Training Loss | In the context of machine learning, the term "training loss" refers to the value of the loss function computed during the training phase of a model. The loss function quantifies the difference between the predicted output and the actual target values for the training dataset. The goal of training a machine learning model is to minimize this loss, as it represents the error or discrepancy between the model's predictions and the ground truth. | 0.0209 |
| Testing Accuracy | Testing accuracy, also known as test accuracy, is a metric used to evaluate the performance of a machine learning model on a separate dataset that was not used during the training phase. This dataset, often referred to as the test set or holdout set, consists of examples that the model has not seen before. Testing accuracy provides insights into how well the trained model generalizes to new, unseen data. | 77.77% |
| Validation Loss | In machine learning, particularly during the training of a model, the term "validation loss" refers to the value of the loss function computed on a separate validation dataset. The validation dataset is distinct from the training dataset and is not used during the training process. Instead, it serves as an independent set of examples that the model has not seen before. The validation loss is a crucial metric for assessing how well the model generalizes to new, unseen data. | 0.0122 |

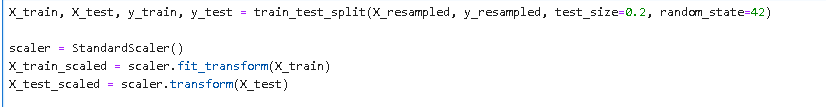
**6.3 GRU:**

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture designed to address challenges in modeling sequential data by introducing a sophisticated gating mechanism. In GRU, each unit contains a reset gate and an update gate, enhancing its ability to selectively incorporate or discard information from previous time steps. The reset gate determines the extent to which past information should be forgotten, while the update gate regulates the balance between new input and existing hidden state. GRU's simplicity lies in combining the memory cell and hidden state into a single state vector, making it computationally efficient compared to traditional RNNs and even Long Short-Term Memory (LSTM) networks. The mathematical formulation involves computing a candidate hidden state based on the reset gate and input, and then updating the hidden state using the update gate. GRU has found success in various applications, such as natural language processing and time-series analysis, where capturing long-term dependencies is crucial. Its effectiveness and efficiency make it a valuable choice for tasks involving sequential data.

**GRU.ipynb:**

****

****

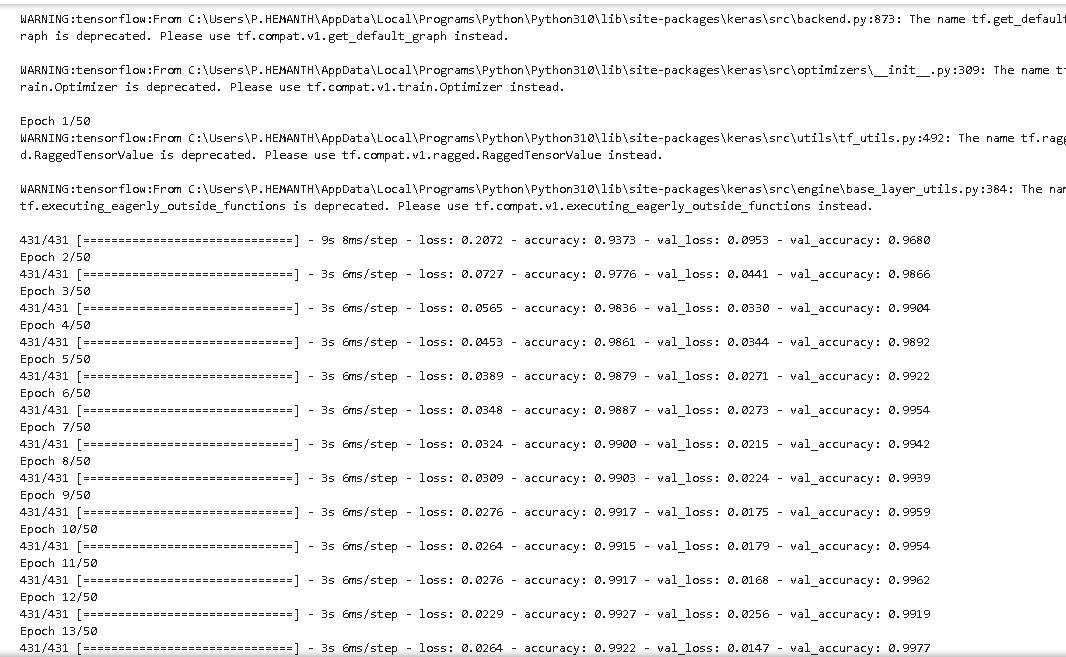
****

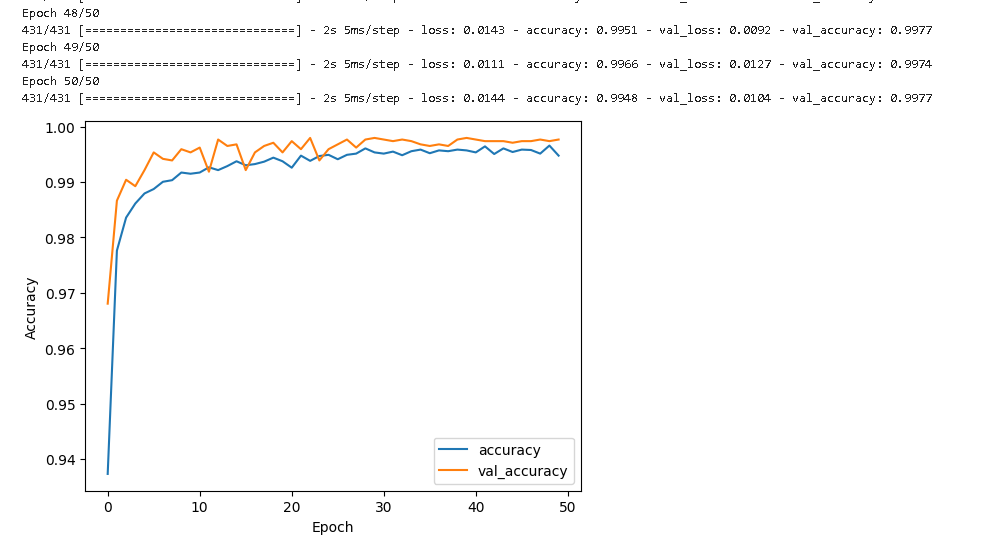
****

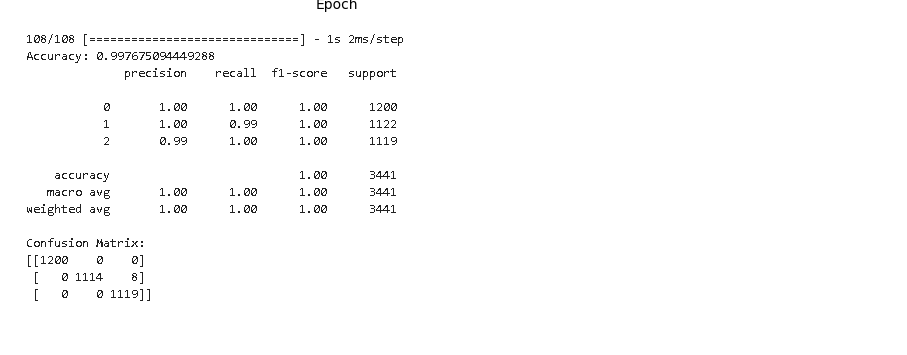
****

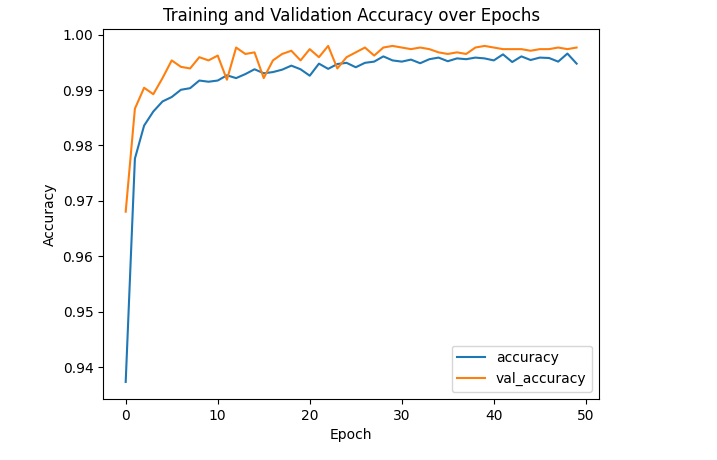
****

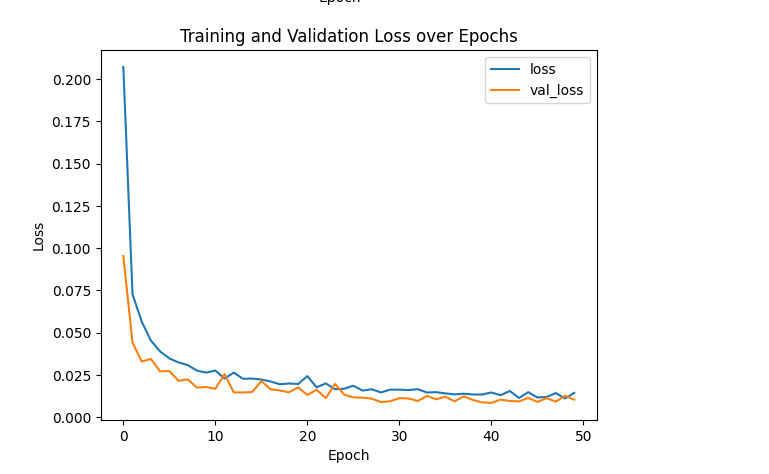
**Output:**

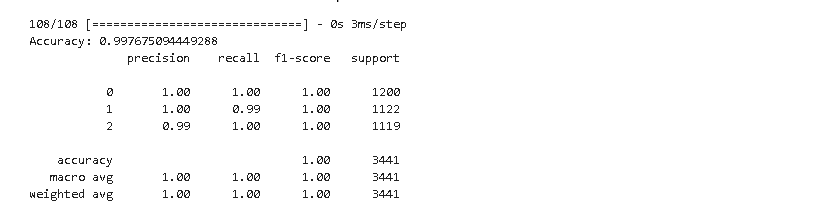
****

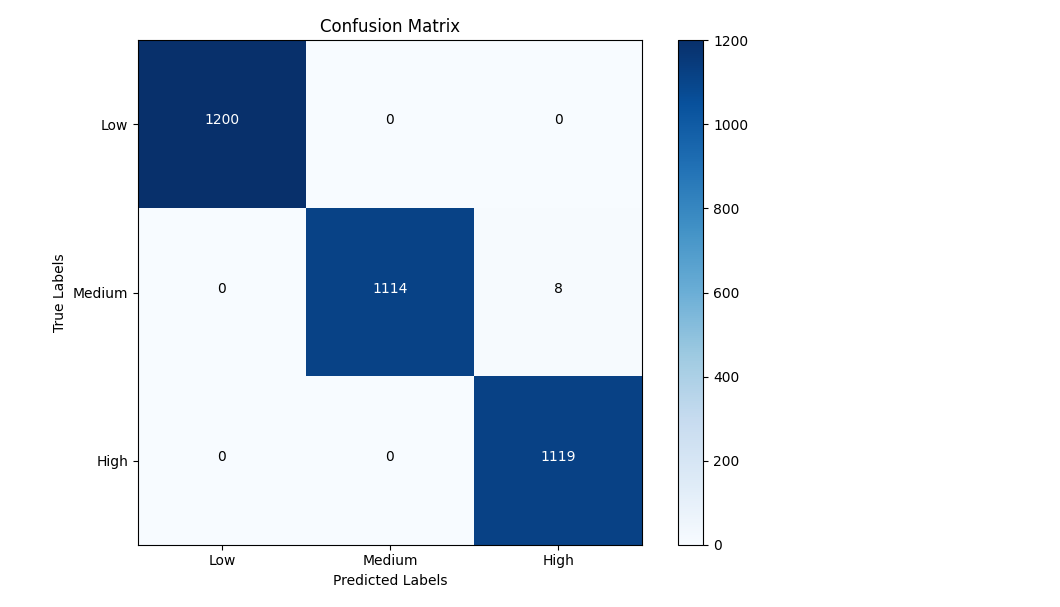
****

****

****

****

****

****

**Inference:**

* The Python script presented here is a comprehensive implementation of a machine learning model geared towards predicting price categories based on a set of input features.
* The initial stage involves importing crucial libraries, such as NumPy for numerical operations, pandas for data manipulation, and TensorFlow's Keras module for building neural networks. The dataset, sourced from a CSV file through pandas, is then subjected to a standard machine learning workflow.
* The data is split into training and testing sets utilizing the `train\_test\_split` function from scikit-learn. Following this, the input features (`X\_resampled`) and the corresponding target variable (`y\_resampled`) are isolated. Ensuring uniform scaling, the input features undergo standardization using scikit-learn's `StandardScaler`.
* However, what distinguishes this script is the subsequent reshaping of the input data to introduce a third dimension. This transformation is essential to align with the requirements of GRU layers, which expect a 3D input shape.
* The target variable (`y\_resampled`) undergoes one-hot encoding using Keras's `to\_categorical` function, a standard preprocessing step for classification tasks with categorical outcomes.
* Subsequently, a sequential model is constructed using Keras. This model comprises three GRU layers, each configured with 50 units, interleaved with dropout layers designed to mitigate overfitting—a common concern in neural network training.
* The architecture culminates in a dense output layer equipped with a softmax activation function, indicative of a multi-class classification scenario with three classes: Low, Medium, and High. The model is then compiled, specifying the Adam optimizer, categorical cross entropy as the loss function, and accuracy as the evaluation metric.
* The training phase involves feeding the model with the prepared data through the `fit` method. The specified parameters include the training data, the number of epochs (set to 50), batch size (32), and validation data.
* The training process is monitored, and key metrics, such as accuracy and loss, are recorded over epochs. These metrics are subsequently visualized using Matplotlib to gain insights into the model's learning trajectory.
* Once trained, the model is evaluated on the test set, and its performance is rigorously assessed. Metrics such as accuracy, a classification report, and a confusion matrix are printed, offering a granular understanding of the model's predictive capabilities.
* The script goes further by visualizing the confusion matrix, providing a graphical representation of the model's success and challenges in predicting different classes.
* In summary, this script encapsulates the end-to-end process of building, training, and evaluating a GRU-based neural network for a multiclass classification task. The detailed explanations at each stage illuminate the rationale behind each step, making it a valuable resource for understanding the intricacies of implementing a sophisticated machine learning model. The visualizations serve as informative tools for model assessment and potential refinement.

**Parameter set in the Model :**

|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |
| Epoch | In the context of machine learning, especially neural network training, an epoch refers to one complete pass through the entire training dataset during the training phase. In other words, it represents a single iteration over the entire set of training samples. During each epoch, the model's parameters (weights and biases) are adjusted based on the computed gradients of the loss function, aiming to minimize the difference between the predicted outputs and the actual targets. | 50 |
| Batch size | In machine learning, particularly in training neural networks, the batch size is a hyperparameter that defines the number of training samples utilized in one iteration. During each iteration, the model's parameters are updated based on the gradients computed from the selected batch of training data. | 32 |
| Optimizer | An optimizer is a key component in training machine learning models, responsible for adjusting the model's parameters (weights and biases) during the training process to minimize the error or loss function. The primary goal of an optimizer is to find the optimal set of parameters that allows the model to make accurate predictions on new, unseen data. | adam |
| Activation Function | An activation function is a mathematical operation applied to the output of each neuron in a neural network. It introduces non-linearity to the network, allowing it to learn complex patterns and relationships in the data. | softmax |
| GRU Unit | The Gated Recurrent Unit (GRU) unit is the fundamental building block of a Gated Recurrent Unit (GRU) neural network, specifically designed for processing sequential data. It is a type of recurrent neural network (RNN) cell that aims to address some limitations of traditional RNNs, such as difficulties in learning long-term dependencies due to the vanishing gradient problem. | 100 |

**Parameters evaluated in the model:**

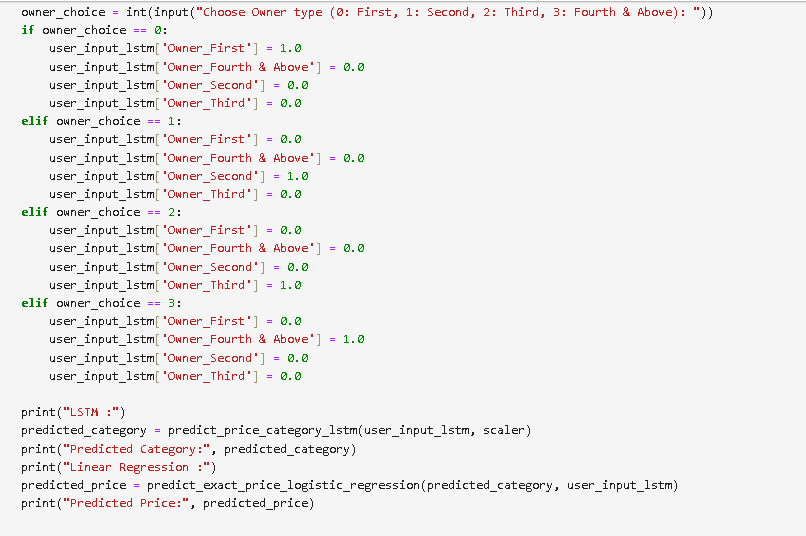
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Accuracy | Accuracy is a common metric used to evaluate the performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances. | 0.99767 |
| Confusion matrix | A confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of test data for which the true values are known | [[1200 0 0]  [ 0 1114 8]  [ 0 0 1119]] |
| Training Loss | In the context of machine learning, the term "training loss" refers to the value of the loss function computed during the training phase of a model. The loss function quantifies the difference between the predicted output and the actual target values for the training dataset. The goal of training a machine learning model is to minimize this loss, as it represents the error or discrepancy between the model's predictions and the ground truth. | 0.0152 |
| Testing Accuracy | Testing accuracy, also known as test accuracy, is a metric used to evaluate the performance of a machine learning model on a separate dataset that was not used during the training phase. This dataset, often referred to as the test set or holdout set, consists of examples that the model has not seen before. Testing accuracy provides insights into how well the trained model generalizes to new, unseen data. | 77.77% |
| Validation Loss | In machine learning, particularly during the training of a model, the term "validation loss" refers to the value of the loss function computed on a separate validation dataset. The validation dataset is distinct from the training dataset and is not used during the training process. Instead, it serves as an independent set of examples that the model has not seen before. The validation loss is a crucial metric for assessing how well the model generalizes to new, unseen data. | 0.0167 |

**Section –7**

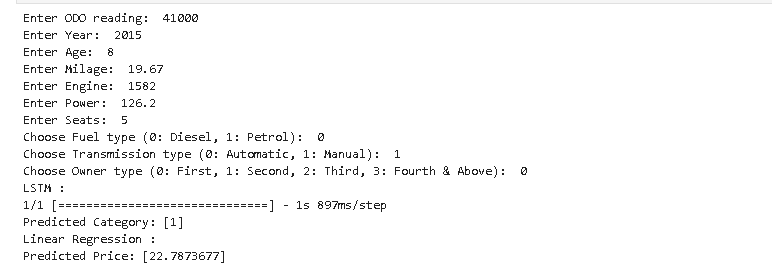
**Console based Application**

**Application.ipynb:**





**Output:**



**Inference:**

* The provided Python program is a comprehensive machine learning model designed to predict both the category and price of a car based on various input features.
* The initial section of the code includes the necessary imports, such as NumPy for numerical operations, Pandas for data manipulation, joblib for model storage, and scikit-learn modules for preprocessing.
* These libraries collectively provide a robust foundation for the machine learning tasks performed later in the program.
* Following the imports, the program proceeds to load pre-trained models. These models include a Long Short-Term Memory (LSTM) model (`lstm.h5`), known for its effectiveness in sequence prediction tasks, and a linear regression model (`linear.pkl`), a widely-used algorithm for regression problems.
* Additionally, a scaler (`scaler.pkl`) is loaded, suggesting that the input features undergo a scaling process, a common practice in machine learning to ensure consistent and meaningful comparisons among different features.
* Two essential preprocessing functions are introduced: `preprocess\_input\_lstm` and `predict\_price\_category\_lstm`. The former function takes care of scaling and reshaping the input data to match the requirements of the LSTM model, which expects a specific format. The latter function utilizes the loaded LSTM model to predict the category of the car based on the processed input.
* The program then introduces another function, `predict\_exact\_price\_logistic\_regression`, which uses the pre-trained linear regression model to predict the exact price of the car. This function takes into account the predicted category from the LSTM model and additional user input, providing a more detailed and precise price estimation.
* User interaction is a key aspect of the program. It prompts the user to input various car features such as ODO reading, Year, Age, Mileage, Engine, Power, Seats, Fuel type, Transmission type, and Owner type. The user inputs related to Fuel type, Transmission type, and Owner type are converted into a one-hot encoded format, a common technique for representing categorical variables in machine learning models.
* After obtaining user input, the program leverages the LSTM model to predict the car category. Subsequently, the linear regression model predicts the exact price based on both the predicted category and the additional user input. Finally, the program displays the predicted category and price to the console.
* It's crucial to highlight that this program seems to be a demonstration or educational tool, and when applying it in real-world scenarios, careful consideration of model accuracy, validation, and user input validation is essential for reliable predictions and robust application.