

Car Price Prediction (Linear Regression)

ABSTRACT

A car price prediction has been a high interest research area, as it requires noticeable effort and knowledge of the field expert. Considerable number of distinct attributes are examined for the reliable and accurate prediction. To build a model for predicting the price of used cars in Bosnia and Herzegovina, we applied Linear Regression algorithm. However, the mentioned technique is applied for predicting the car prices. The dataset used for the prediction was taken from UCI Machine Learning Repository. Furthermore, the model was evaluated using test data and the accuracy of 87.38% was obtained.

TABLE OF CONTENTS

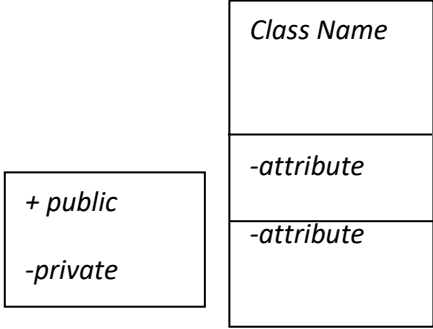
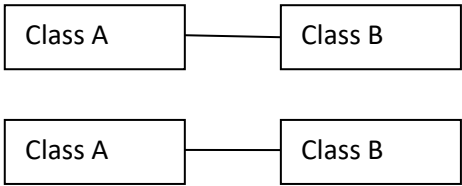
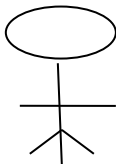
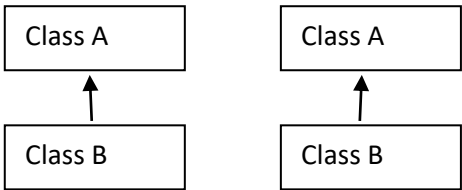
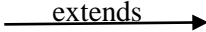

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	2
	LIST OF FIGURES	5
	LIST OF SYMBOLS	6
1.	CHAPTER 1: INTRODUCTION	9
	➤ GENERAL	9
	➤ OBJECTIVE	10
	➤ EXISTING SYSTEM	11
	1.3.1 EXISTING SYSTEM DISADVANTAGES	11
	1.3.2 LITERATURE SURVEY	12
	1.4 PROPOSED SYSTEM	15
	1.4.1 PROPOSED SYSTEM ADVANTAGES	15
2.	CHAPTER 2: PROJECT DESCRIPTION	16
	2.1 GENERAL	16
	2.2 METHODOLOGIES	16
	2.2.1 MODULES NAME	17
	2.2.2 MODULES EXPLANATION	17
	2.3 TECHNIQUE USED OR ALGORITHM USED	18
3.	CHAPTER 3: REQUIREMENTS	19
	3.1 GENERAL	19
	3.2 HARDWARE REQUIREMENTS	19
	3.3 SOFTWARE REQUIREMENTS	20
	3.4 FUNCTIONAL REQUIREMENTS	20
	3.5 NON-FUNCTIONAL REQUIREMENTS	21
4.	CHAPTER 4: SYSTEM DESIGN	22
	4.1 GENERAL	22
	4.2 UML DIAGRAMS	23
	4.2.2 USE CASE DIAGRAM	23
	4.2.3 CLASS DIAGRAM	24
	4.2.4 OBJECT DIAGRAM	25
	4.2.5 STATE DIAGRAM	26
	4.2.6 ACTIVITY DIAGRAM	27
	4.2.7 SEQUENCE DIAGRAM	28

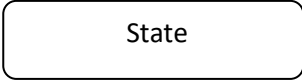
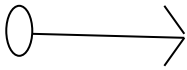
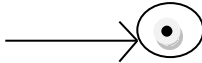
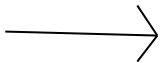
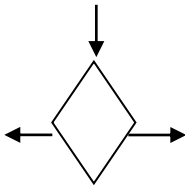
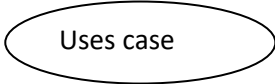
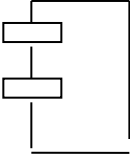
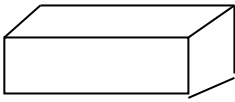
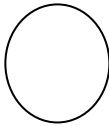
	4.2.8 COLLABORATION DIAGRAM	29
	4.2.9 COMPONENT DIAGRAM	30
	4.2.10 DATA FLOW DIAGRAM	31
	4.2.11 DEPLOYMENT DIAGRAM	33
	4.2.12 SYSTEM ARCHITECTURE	34
5.	CHAPTER 5: DEVELOPMENT TOOLS	35
	5.1 PYTHON	35
	5.2 HISTORY OF PYTHON	35
	5.3 IMPORTANCE OF PYTHON	35
	5.4 FEATURES OF PYTHON	36
	5.5 LIBRARIES USED IN PYTHON	37
6.	CHAPTER 6: IMPLEMENTATION	38
	6.1 IMPLEMENTATION	38
7.	CHAPTER 7: SNAPSHOTS	45
	7.1 SNAPSHOTS	45
8.	CHAPTER 8: SOFTWARE TESTING	52
	8.1 GENERAL	52
	8.2 DEVELOPING METHODOLOGIES	52
	8.3 TYPES OF TESTING	52
9.	CHAPTER 9: FUTURE ENHANCEMENT	55
	9.1 GENERAL	55
10	CHAPTER 10:	56
	10.1 CONCLUSION	56
	10.2 REFERENCES	57

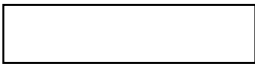

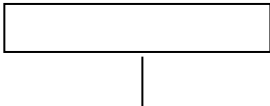
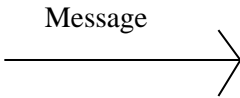
LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO.
2.3.2	Use Case Diagram	26
4.2	Class Diagram	27
4.3	Object Diagram	28
4.4	State Diagram	29
4.5	Activity Diagram	30
4.6	Sequence Diagram	31
4.7	Collaboration Diagram	32
4.8	Component Diagram	33
4.9	Data Flow Diagram	34
4.1	Deployment Diagram	36
4.12	System Architecture	37

LIST OF SYMSBOLS

S.NO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single class.
4.	Aggregation		Interaction between the system and external environment
5.	Relation (uses)	Uses	Used for additional process communication.
6.	Relation (extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.

8.	State		State of the processes.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Use case		Interact ion between the system and external environment.
14.	Component		Represents physical modules which are a collection of components.
15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to

			some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message		Represents the message exchanged.

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Car Price Prediction has been a high interest research area. Accurate car price prediction involves expert knowledge, because price usually depends on many distinctive features and factors. Typically, most significant ones are brand and model, age, horsepower and mileage. The fuel type used in the car as well as fuel consumption per mile highly affect price of a car due to a frequent change in the price of a fuel. Different features like exterior color, type of transmission, dimensions, safety, air condition, interior, navigation will also influence the car price. In this project, we applied different methods and techniques in order to achieve higher precision of the used car price prediction.

1.2 OBJECTIVE

To build a model for predicting the price of used cars in Bosnia and Herzegovina, we applied Linear Regression Algorithm. The mentioned technique is applied to labelled data. The data used for the prediction was collected from UCI Machine Learning Repository. Furthermore, the model was evaluated using test data and the accuracy of 87.38% was obtained.

1.3 Existing System

In existing system, they are implementing car price prediction system using K-Nearest Neighbor algorithm.

1.3.1 Existing System Disadvantages:

- With large data, the prediction stage might be slow.
- Sensitive to the scale of the data and irrelevant features.
- Requires high memory

1.3.2 LITERATURE SURVEY:

Title: Predicting the price of used cars using machine learning techniques.

Author: Pudaruth,

Year:2017

Description: According to a World Economic Forum's report, AI-enabled automation will generate 133 million new jobs globally by 2022. And in India itself, the demand for AI talent pool is expected to skyrocket with the government's steps towards digitization, and multiple organizations accelerating their digital transformation initiatives. Are you ready to ride the wave? The BITS Pilani 11-month online PG Programme in AI & ML is designed to help working professionals like you to develop an understanding of AI & ML and its various building blocks.

Title: An expert system of price forecasting for used cars using adaptive neuro fuzzy inference.

Year: 2009

Author: Wu, J. D., Hsu, C. C., & Chen, H. C.

Description: An expert system for used cars price forecasting using adaptive neuro-fuzzy inference system (ANFIS) is presented in this paper. The proposed system consists of three parts: data acquisition system, price forecasting algorithm and performance analysis. The effective factors in the present system for price forecasting are simply assumed as the mark of the car, manufacturing year and engine style. Further, the equipment of the car is considered to raise the performance of price forecasting. In price forecasting, to verify the effect of the proposed ANFIS, a conventional artificial neural network (ANN) with back-propagation (BP) network is compared with proposed ANFIS for price forecast because of its adaptive learning capability. The ANFIS includes both fuzzy logic qualitative approximation and the adaptive neural network capability. The experimental result pointed out that the proposed expert system using ANFIS has more possibilities in used car price forecasting.

Title: New Model for Residual Value Prediction of the Used Car Based on BP Neural Network and Nonlinear Curve Fit. In Measuring Technology and Mechatronics Automation

Year: 2011

Author: Gongqi, Yansong & Qiang

Description: A new model for predicting the residual value of the private used car with various conditions, such as manufacturer, mileage, time of life, etc., was developed in this paper. A comprehensive method combined by the BP neural network and nonlinear curve fit was introduced for optimizing the model due to its flexible nonlinearity. Firstly, some distribution curves of residual value of the used cars were analyzed in time domain. Then, the BP neural network (NN) was established and used to extract the feature of the distribution curves in various conditions. A set of schemed data was used to train the NN and reached the training goal. Finally, the schemed data as inputs and the NN outputs were organized for nonlinear curve fit. Conclusion was drawn that the newly proposed model is feasible and accurate for residual value prediction of the used cars with various conditions.

1.4 Proposed System

In this system we are implementing effective car price prediction system using linear regression. We can give the input as in CSV file or manual entry to the system. After taking input the algorithms applied. After accessing data set the operation is performed and effective results are obtained.

1.4.1 Proposed System Advantages:

- Easy to predict
- High accuracy rate

CHAPTER 2

PROJECT DESCRIPTION

2.1 GENERAL:

The fuel type used in the car as well as fuel consumption per mile highly affect price of a car due to a frequent change in the price of a fuel. Different features like exterior color, door number, type of transmission, dimensions, safety, air condition, interior, whether it has navigation or not will also influence the car price. In this paper, we applied different methods and techniques in order to achieve higher precision of the used car price prediction.

2.2 METHODOLOGIES

MODULE:

- **Data Collection**
- **Data Pre-processing**
- **Model Building**
- **Evaluation**

MODULE DESCRIPTION

Data Collection

To build and develop Machine Learning models, you must first acquire the relevant dataset. This dataset will be comprised of data gathered from multiple and disparate sources which are then combined in a proper format to form a dataset. Dataset formats differ according to use cases. For instance, a business dataset will be entirely different from a medical dataset. While a business dataset will contain relevant industry and business data, a medical dataset will include healthcare-related data. There are many sources for collecting datasets such as Kaggle, UCI Machine Learning Repository.

Data Pre-processing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. Data Pre-processing includes:

- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling
- Handling Outliers

Model Building

A machine learning model is built by learning and generalizing from training data, then applying that acquired knowledge to new data it has never seen before to make predictions and fulfill its purpose. Linear Regression model is used for predicting the values and the values predicted are evaluated by R2 score.

Evaluation

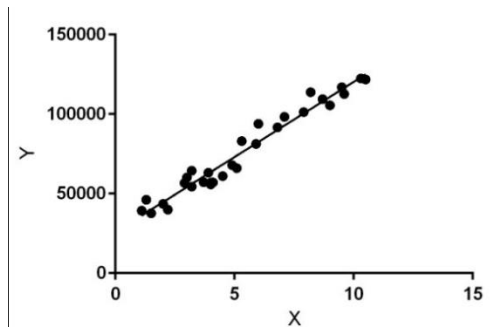
The algorithm was applied on the train dataset and the model is evaluated by the test dataset using R2 score. The R2 score is a very important metric that is used to evaluate the performance of a regression-based machine learning model. It is pronounced as R squared and is also known as the coefficient of determination. It works by measuring the amount of variance in the predictions explained by the dataset. R-square is a comparison of residual sum of squares (SS_{res}) with total sum of squares (SS_{tot}).
$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

2.3 TECHNIQUE USED OR ALGORITHM USED

Single machine learning classifier approach that has been used in all previous researches was also tested in this research. The whole data set collected in this research has been split into training (70%) and testing (30%) subsets and Linear Regression model was built.

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.



While training the model we are given:
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)

$$y = \theta_1 + \theta_2 \cdot x$$

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ_1 and θ_2 values.

θ_1 : intercept

θ_2 : coefficient of x

CHAPTER 3

REQUIREMENTS ENGINEERING

3.1 GENERAL

These are the requirements for doing the project. Without using these tools and software's we can't do the project. So, we have two requirements to do the project. They are

1. Hardware Requirements.
2. Software Requirements.

3.2 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system do and not how it should be implemented.

- PROCESSOR : DUAL CORE 2 DUOS.
- RAM : 4GB DD RAM
- HARD DISK : 250 GB

3.3 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- OPERATING SYSTEM : WINDOWS 7/8/10/11
- PLATFORM : JUPYTER NOTEBOOK
- PROGRAMMING LANGUAGE : PYTHON

3.4 FUNCTIONAL REQUIREMENTS

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behavior, in this paper ensemble method for car prices prediction was proposed. To apply ensemble of machine learning classifiers a new attribute “price rank” with values: cheap, moderate and expensive has been added to the data set. This attribute divides cars into three price categories: cheap (price < 12 000 BAM), moderate (12 000 BAM <= price < 24 000 BAM) and expensive (24 000 BAM <= price).

3.5 NON-FUNCTIONAL REQUIREMENTS

The major non-functional Requirements of the system are as follows

➤ **Usability**

The system is designed with completely automated process hence there is no or less user intervention.

➤ **Reliability**

The system is more reliable because of the qualities that are inherited from the chosen platform java. The code built by using java is more reliable.

➤ **Performance**

This system is developing is done in the high-level languages and using the advanced front-end and back-end technologies it will give response to the end user on client system with in very less time.

➤ **Supportability**

The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform, which is having JVM, built into the system.

CHAPTER 4

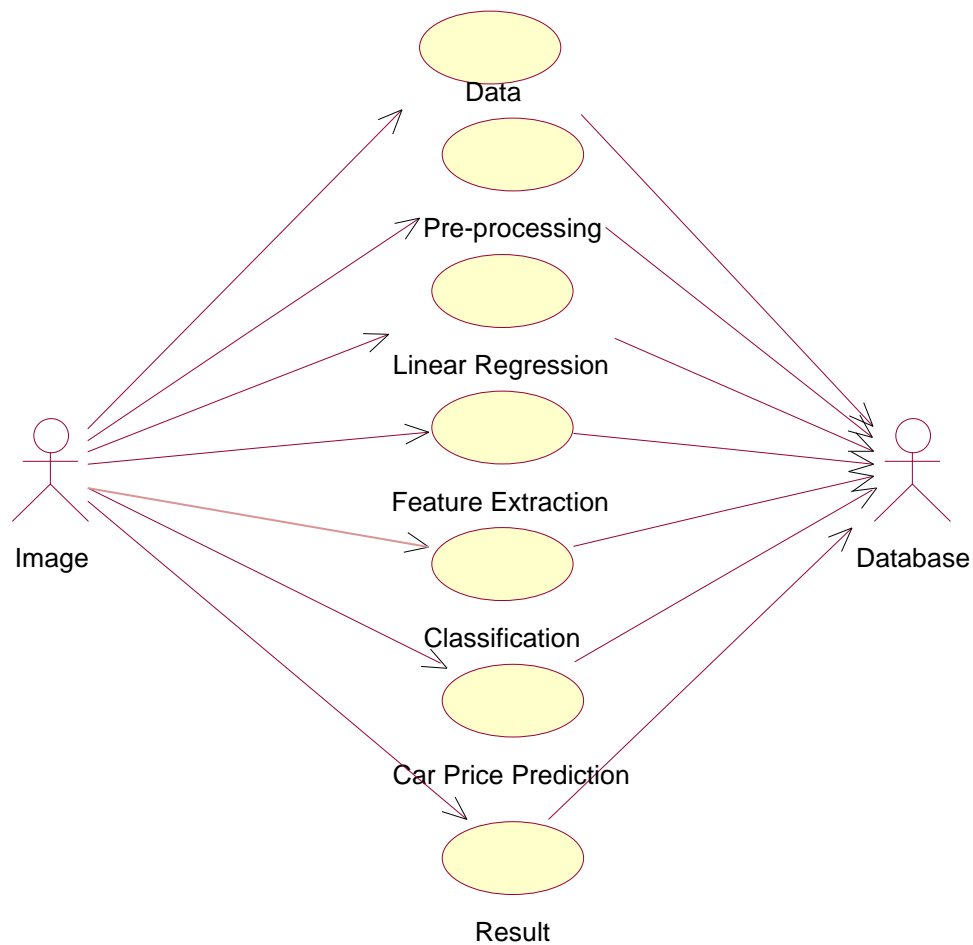
SYSTEM DESIGN

4.1 GENERAL

Design Engineering deals with the various UML [Unified Modelling language] diagrams for the implementation of project. Design is a meaningful engineering representation of a thing that is to be built. Software design is a process through which the requirements are translated into representation of the software. Design is the place where quality is rendered in software engineering. Design is the means to accurately translate customer requirements into finished product.

UML Diagrams

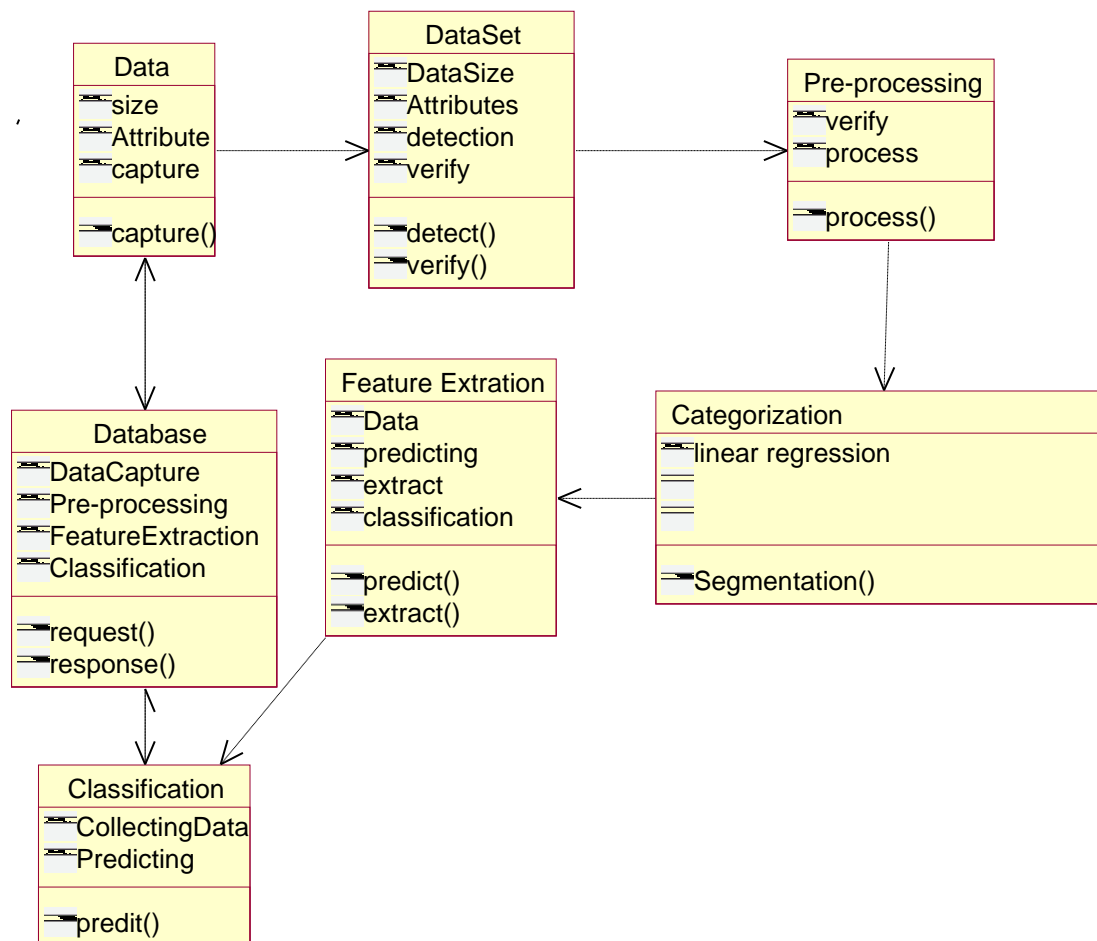
USE CASE DIAGRAM:



EXPLANATION:

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. The above diagram consists of user as actor. Each will play a certain role to achieve the concept.

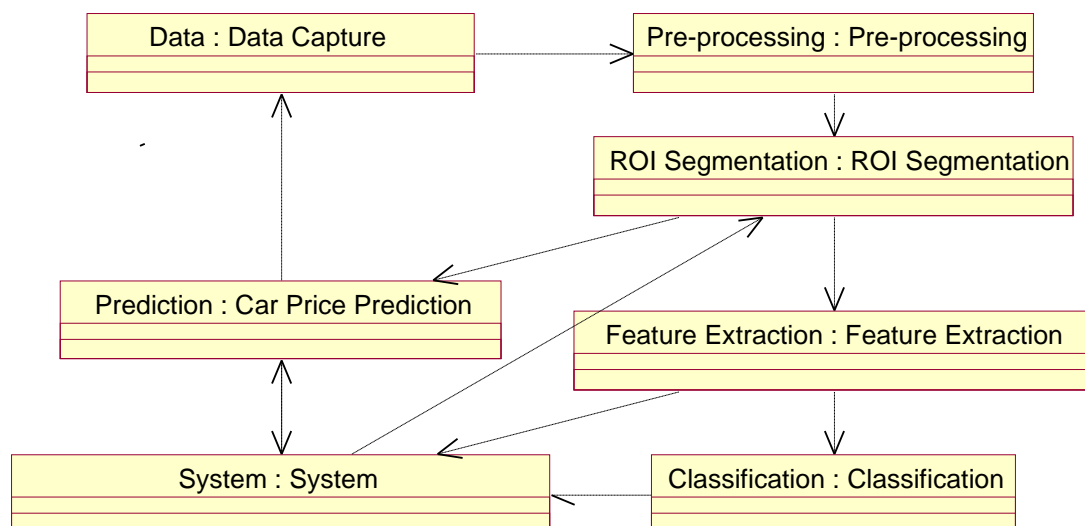
CLASS DIAGRAM:



EXPLANATION:

In this class diagram represents how the classes with attributes and methods are linked together to perform the verification with security. From the above diagram shown the various classes involved in our project.

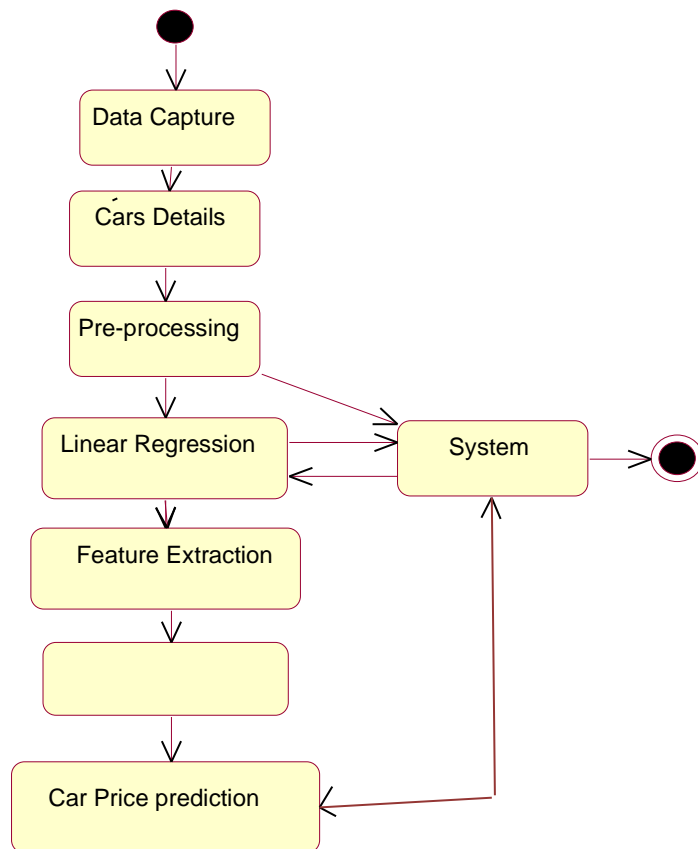
OBJECT DIAGRAM:



EXPLANATION:

In the above diagram tells about the flow of objects between the classes. It is a diagram that shows a complete or partial view of the structure of a modeled system. In this object diagram represents how the classes with attributes and methods are linked together to perform the verification with security.

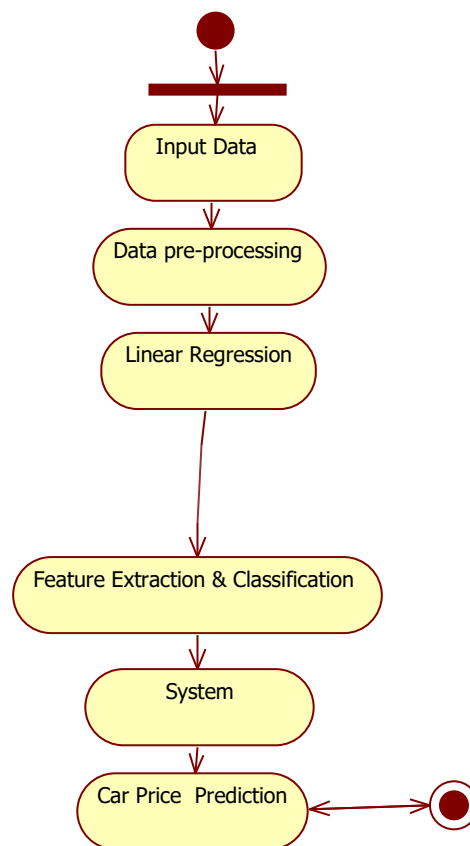
STATE DIAGRAM:



EXPLANATION:

State diagram are a loosely defined diagram to show workflows of stepwise activities and actions, with support for choice, iteration and concurrency. UML, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. UML activity diagrams could potentially model the internal logic of a complex operation. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structural development.

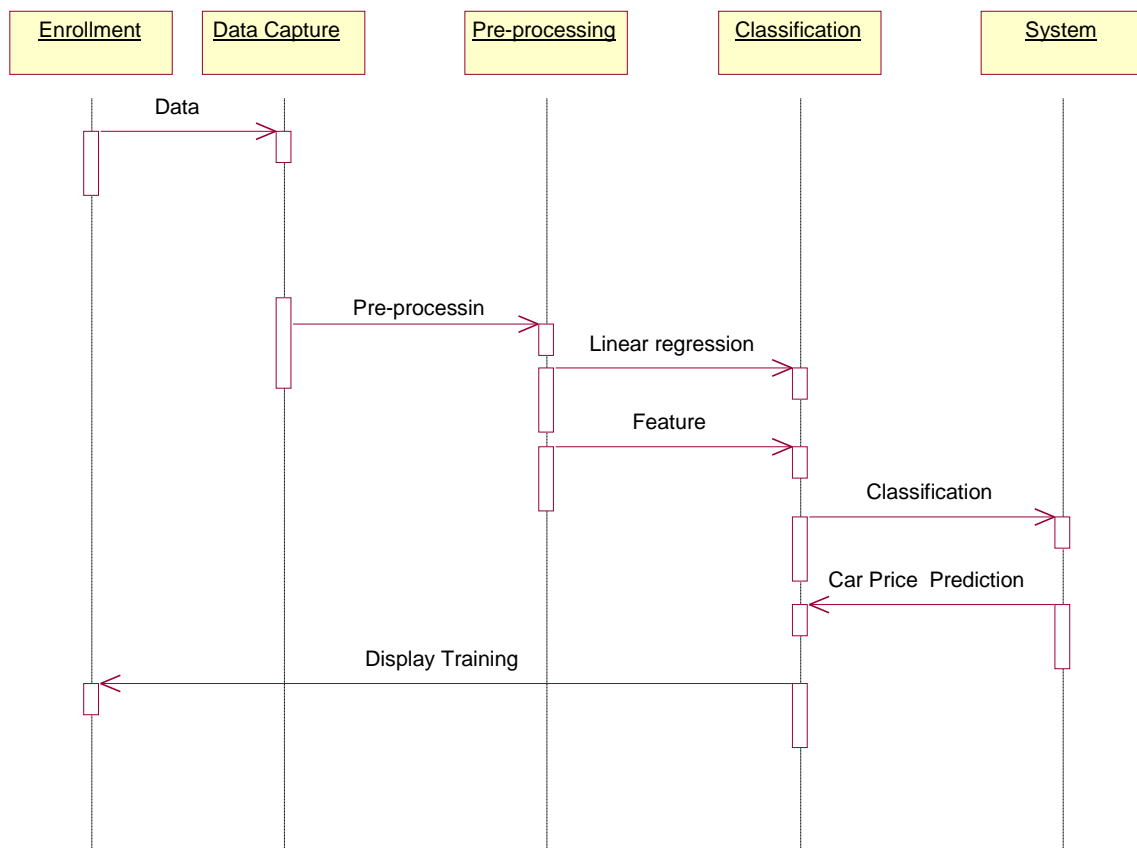
ACTIVITY DIAGRAM:



EXPLANATION:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

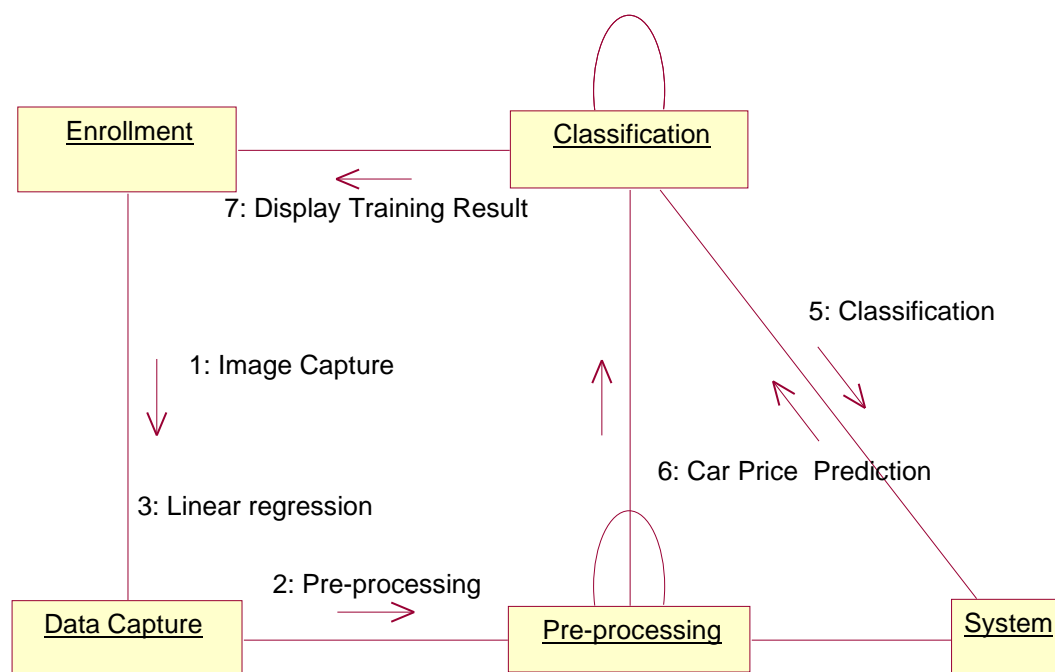
SEQUENCE DIAGRAM:



EXPLANATION:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

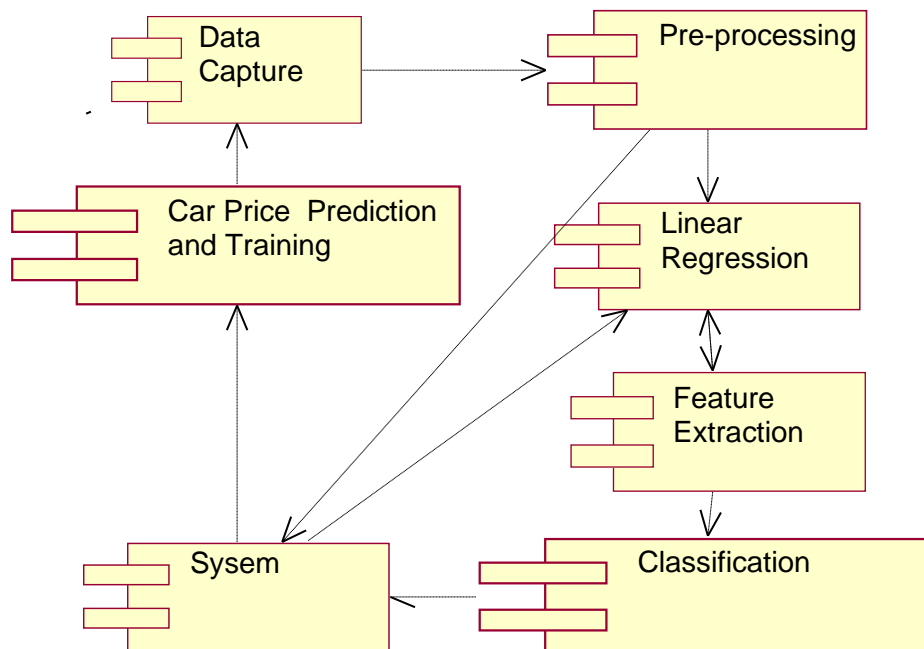
COLLABORATION DIAGRAM:



EXPLANATION:

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.

COMPONENT DIAGRAM:

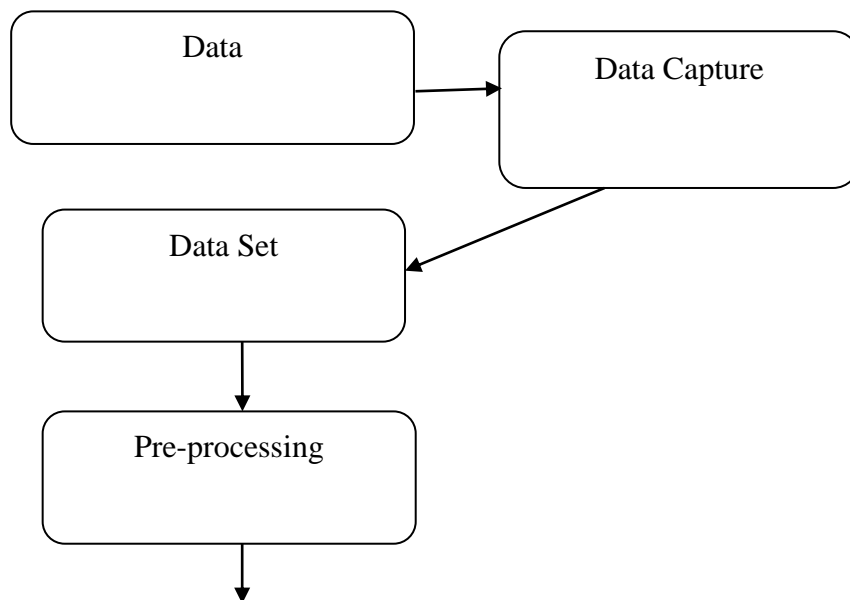


EXPLANATION:

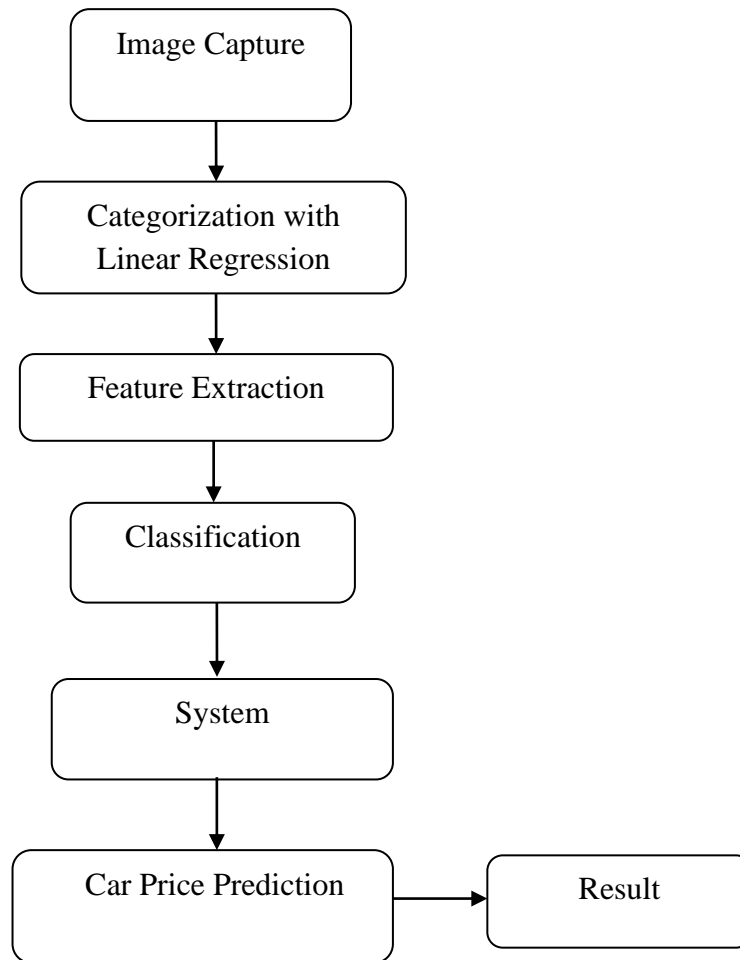
In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

DATA FLOW DIAGRAM:

Level-0:



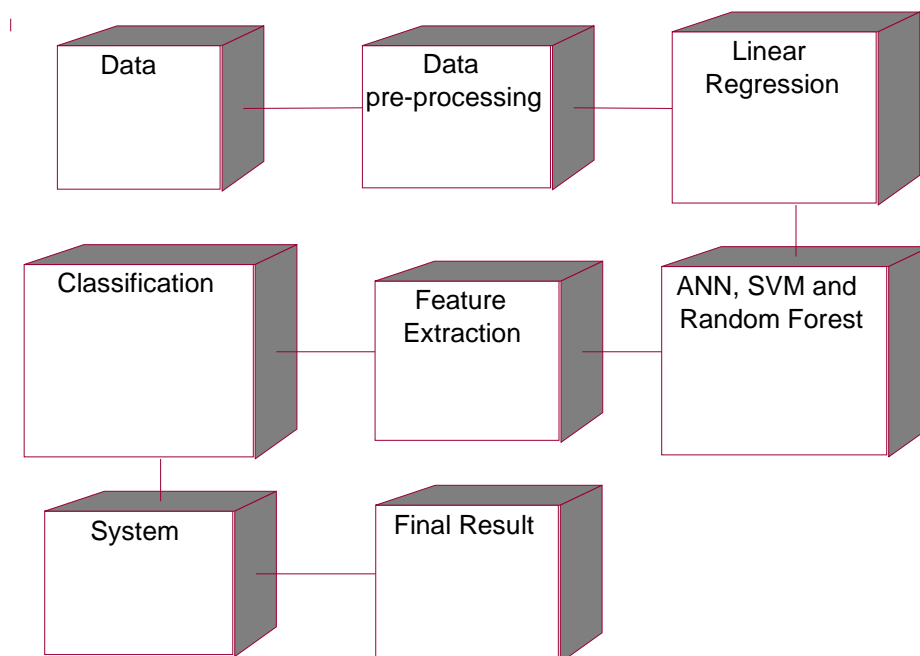
Level-1



EXPLANATION:

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

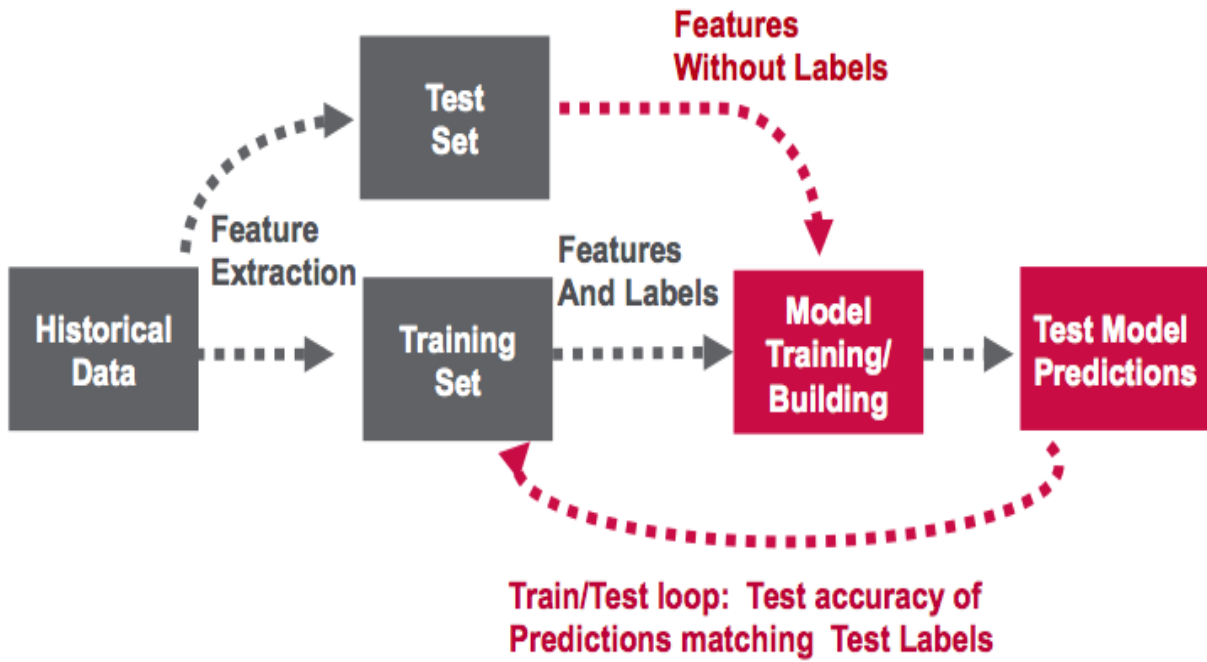
DEPLOYMENT DIAGRAM:



EXPLANATION:

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. User gives main query and it converted into sub queries and sends through data dissemination to data aggregators. Results are to be showed to user by data aggregators. All boxes are components and arrow indicates dependencies.

SYSTEM ARCHITECTURE:



System Architecture

CHAPTER 5

DEVELOPMENT TOOLS

Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Importance of Python

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Features of Python

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** – Python provides interfaces to all major commercial databases.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Libraries used in python:

- NumPy - mainly useful for its N-dimensional array objects.
- pandas - Python data analysis library, including structures such as data frames.
- matplotlib - 2D plotting library producing publication quality figures.
- scikit-learn - the machine learning algorithms used for data analysis and data mining tasks.



Figure: NumPy, Pandas, Matplotlib, Scikit-learn

CHAPTER 6

IMPLEMENTATION

6.1 CODE

```
1. import warnings
2. warnings.filterwarnings('ignore')
3. import numpy as np
4. import pandas as pd
5. import matplotlib.pyplot as plt
6. import seaborn as sns
7. %matplotlib inline
8. from matplotlib.pyplot import xticks
9.
10. df = pd.DataFrame(pd.read_csv("CarPrice_Assignment.csv"))
11. df.head()
12. df.shape
13. df.describe()
14. df.info()
15. df.columns
16.
17. # Data Cleaning
18.
19. sum(df.duplicated(subset = 'car_ID')) == 0
20. df.isnull().sum()*100/df.shape[0]
21. df.price.describe()
22. sns.distplot(df['price'])
23.
24. plt1 = sns.countplot(df['symboling'])
25. plt1.set(xlabel = 'Symbol', ylabel= 'Count of Cars')
26. plt.show()
27. plt.tight_layout()
28.
29. df_sym = pd.DataFrame(df['symboling'].value_counts())
30. df_sym.plot.pie(subplots=True, labels = df_sym.index.values, autopct='%1.1f%%', figsize = (15,7.5))
31. # Unsquish the pie.
32. plt.gca().set_aspect('equal')
33. plt.show()
34. plt.tight_layout()
35.
36. plt1 = df[['symboling', 'price']].groupby("symboling").mean().plot(kind='bar', legend = False,)
37. plt1.set_xlabel("Symbol")
38. plt1.set_ylabel("Avg Price (Dollars)")
39. xticks(rotation = 0)
40. plt.show()
41.
42. df.CarName.values[0:10]
43. df['brand'] = df.CarName.str.split(' ').str.get(0).str.upper()
44. len(set(df.brand.values))
45.
46. fig, ax = plt.subplots(figsize = (15,5))
47. plt1 = sns.countplot(df['brand'], order=pd.value_counts(df['brand']).index,)
48. plt1.set(xlabel = 'Brand', ylabel= 'Count of Cars')
49. xticks(rotation = 90)
50. plt.show()
51. plt.tight_layout()
52.
53. df['brand'] = df['brand'].replace(['VW', 'VOKSWAGEN'], 'VOLKSWAGEN')
54. df['brand'] = df['brand'].replace(['MAXDA'], 'MAZDA')
55. df['brand'] = df['brand'].replace(['PORCSHCE'], 'PORSCHE')
56. df['brand'] = df['brand'].replace(['TOYOUTA'], 'TOYOTA')
57.
```

```

58. fig, ax = plt.subplots(figsize = (15,5))
59. plt1 = sns.countplot(df['brand'], order=pd.value_counts(df['brand']).index,)
60. plt1.set(xlabel = 'Brand', ylabel= 'Count of Cars')
61. xticks(rotation = 90)
62. plt.show()
63. plt.tight_layout()
64.
65. df.brand.describe()
66.
67. df_comp_avg_price = df[['brand','price']].groupby("brand", as_index =
    False).mean().rename(columns={'price':'brand_avg_price'})
68. plt1 = df_comp_avg_price.plot(x = 'brand', kind='bar',legend = False, sort_columns = True, figsize =
    (15,3))
69. plt1.set_xlabel("Brand")
70. plt1.set_ylabel("Avg Price (Dollars)")
71. xticks(rotation = 90)
72. plt.show()
73.
74. df = df.merge(df_comp_avg_price, on = 'brand')
75. df['brand_category'] = df['brand_avg_price'].apply(lambda x : "Budget" if x < 10000
76. else ("Mid_Range" if 10000 <= x < 20000
77. else "Luxury"))
78.
79. ## Fuel Type
80. df_fuel_avg_price = df[['fueltype','price']].groupby("fueltype", as_index =
    False).mean().rename(columns={'price':'fuel_avg_price'})
81. plt1 = df_fuel_avg_price.plot(x = 'fueltype', kind='bar',legend = False, sort_columns = True)
82. plt1.set_xlabel("Fuel Type")
83. plt1.set_ylabel("Avg Price (Dollars)")
84. xticks(rotation = 0)
85. plt.show()
86.
87. ## Aspiration
88. df_aspir_avg_price = df[['aspiration','price']].groupby("aspiration", as_index =
    False).mean().rename(columns={'price':'aspir_avg_price'})
89. plt1 = df_aspir_avg_price.plot(x = 'aspiration', kind='bar',legend = False, sort_columns = True)
90. plt1.set_xlabel("Aspiration")
91. plt1.set_ylabel("Avg Price (Dollars)")
92. xticks(rotation = 0)
93. plt.show()
94.
95. ## Door Numbers
96. df_door_avg_price = df[['doornumber','price']].groupby("doornumber", as_index =
    False).mean().rename(columns={'price':'door_avg_price'})
97. plt1 = df_door_avg_price.plot(x = 'doornumber', kind='bar',legend = False, sort_columns = True)
98. plt1.set_xlabel("No of Doors")
99. plt1.set_ylabel("Avg Price (Dollars)")
100. xticks(rotation = 0)
101. plt.show()
102.
103. ## Car Body
104. df_body_avg_price = df[['carbody','price']].groupby("carbody", as_index =
    False).mean().rename(columns={'price':'carbody_avg_price'})
105. plt1 = df_body_avg_price.plot(x = 'carbody', kind='bar',legend = False, sort_columns = True)
106. plt1.set_xlabel("Car Body")
107. plt1.set_ylabel("Avg Price (Dollars)")
108. xticks(rotation = 0)
109. plt.show()
110.
111. ## Drivewheel
112. df_drivewheel_avg_price = df[['drivewheel','price']].groupby("drivewheel", as_index =
    False).mean().rename(columns={'price':'drivewheel_avg_price'})
113. plt1 = df_drivewheel_avg_price.plot(x = 'drivewheel', kind='bar', sort_columns = True,legend = False,)
114. plt1.set_xlabel("Drive Wheel Type")
115. plt1.set_ylabel("Avg Price (Dollars)")
116. xticks(rotation = 0)
117. plt.show()
118.
119. ## Wheel base
120. plt1 = sns.scatterplot(x = 'wheelbase', y = 'price', data = df)
121. plt1.set_xlabel('Wheelbase (Inches)')
122. plt1.set_ylabel('Price of Car (Dollars)')
123. plt.show()

```

```

124.
125. ## Car Dimensions
126. fig, axs = plt.subplots(2,2,figsize=(15,10))
127. plt1 = sns.scatterplot(x = 'carlength', y = 'price', data = df, ax = axs[0,0])
128. plt1.set_xlabel('Length of Car (Inches)')
129. plt1.set_ylabel('Price of Car (Dollars)')
130. plt2 = sns.scatterplot(x = 'carwidth', y = 'price', data = df, ax = axs[0,1])
131. plt2.set_xlabel('Width of Car (Inches)')
132. plt2.set_ylabel('Price of Car (Dollars)')
133. plt3 = sns.scatterplot(x = 'carheight', y = 'price', data = df, ax = axs[1,0])
134. plt3.set_xlabel('Height of Car (Inches)')
135. plt3.set_ylabel('Price of Car (Dollars)')
136. plt3 = sns.scatterplot(x = 'curbweight', y = 'price', data = df, ax = axs[1,1])
137. plt3.set_xlabel('Weight of Car (Pounds)')
138. plt3.set_ylabel('Price of Car (Dollars)')
139. plt.tight_layout()
140.
141. ## Engine Specifications
142. ##### Engine Type, Cylinder, Fuel System
143. fig, axs = plt.subplots(1,3,figsize=(20,5))
144. #
145. df_engine_avg_price = df[['enginetype','price']].groupby("enginetype", as_index =
    False).mean().rename(columns={'price':'engine_avg_price'})
146. plt1 = df_engine_avg_price.plot(x = 'enginetype', kind='bar', sort_columns = True, legend = False, ax
    = axs[0])
147. plt1.set_xlabel("Engine Type")
148. plt1.set_ylabel("Avg Price (Dollars)")
149. xticks(rotation = 0)
150. #
151. df_cylindernumber_avg_price = df[['cylindernumber','price']].groupby("cylindernumber", as_index =
    False).mean().rename(columns={'price':'cylindernumber_avg_price'})
152. plt1 = df_cylindernumber_avg_price.plot(x = 'cylindernumber', kind='bar', sort_columns = True, legend =
    False, ax = axs[1])
153. plt1.set_xlabel("Cylinder Number")
154. plt1.set_ylabel("Avg Price (Dollars)")
155. xticks(rotation = 0)
156. #
157. df_fuelsystem_avg_price = df[['fuelsystem','price']].groupby("fuelsystem", as_index =
    False).mean().rename(columns={'price':'fuelsystem_avg_price'})
158. plt1 = df_fuelsystem_avg_price.plot(x = 'fuelsystem', kind='bar', sort_columns = True, legend = False,
    ax = axs[2])
159. plt1.set_xlabel("Fuel System")
160. plt1.set_ylabel("Avg Price (Dollars)")
161. xticks(rotation = 0)
162. plt.show()
163.
164. ### Engine Size, Bore Ratio, Stroke, Horsepower & Compression Ratio
165. fig, axs = plt.subplots(3,2,figsize=(20,20))
166. #
167. plt1 = sns.scatterplot(x = 'enginesize', y = 'price', data = df, ax = axs[0,0])
168. plt1.set_xlabel('Size of Engine (Cubic Inches)')
169. plt1.set_ylabel('Price of Car (Dollars)')
170. #
171. plt2 = sns.scatterplot(x = 'boreratio', y = 'price', data = df, ax = axs[0,1])
172. plt2.set_xlabel('Bore Ratio')
173. plt2.set_ylabel('Price of Car (Dollars)')
174. #
175. plt3 = sns.scatterplot(x = 'stroke', y = 'price', data = df, ax = axs[1,0])
176. plt3.set_xlabel('Stroke')
177. plt3.set_ylabel('Price of Car (Dollars)')
178. #
179. plt4 = sns.scatterplot(x = 'compressionratio', y = 'price', data = df, ax = axs[1,1])
180. plt4.set_xlabel('Compression Ratio')
181. plt4.set_ylabel('Price of Car (Dollars)')
182. #
183. plt5 = sns.scatterplot(x = 'horsepower', y = 'price', data = df, ax = axs[2,0])
184. plt5.set_xlabel('Horsepower')
185. plt5.set_ylabel('Price of Car (Dollars)')
186. #
187. plt5 = sns.scatterplot(x = 'peakrpm', y = 'price', data = df, ax = axs[2,1])
188. plt5.set_xlabel('Peak RPM')
189. plt5.set_ylabel('Price of Car (Dollars)')
190. plt.tight_layout()

```



```

191. plt.show()
192.
193. ## City Mileage & Highway Mileage
194. df['mileage'] = df['citympg']*0.55 + df['highwaympg']*0.45
195. plt1 = sns.scatterplot(x = 'mileage', y = 'price', data = df)
196. plt1.set_xlabel('Mileage')
197. plt1.set_ylabel('Price of Car (Dollars)')
198. plt.show()
199.
200. ## Bivariate Analysis
201. ##### Brand Category - Mileage
202. plt1 = sns.scatterplot(x = 'mileage', y = 'price', hue = 'brand_category', data = df)
203. plt1.set_xlabel('Mileage')
204. plt1.set_ylabel('Price of Car (Dollars)')
205. plt.show()
206.
207. ##### Brand Category - Horsepower
208. plt1 = sns.scatterplot(x = 'horsepower', y = 'price', hue = 'brand_category', data = df)
209. plt1.set_xlabel('Horsepower')
210. plt1.set_ylabel('Price of Car (Dollars)')
211. plt.show()
212.
213. ##### Mileage - Fuel Type
214. plt1 = sns.scatterplot(x = 'mileage', y = 'price', hue = 'fueltype', data = df)
215. plt1.set_xlabel('Mileage')
216. plt1.set_ylabel('Price of Car (Dollars)')
217. plt.show()
218.
219. ##### Horsepower - Fuel Type
220. plt1 = sns.scatterplot(x = 'horsepower', y = 'price', hue = 'fueltype', data = df)
221. plt1.set_xlabel('Horsepower')
222. plt1.set_ylabel('Price of Car (Dollars)')
223. plt.show()
224.
225. # Linear Regression Model
226. auto = df[['fueltype', 'aspiration', 'carbody', 'drivewheel', 'wheelbase', 'carlength', 'carwidth',
'curbweight', 'enginetype',
227. 'cylindernumber', 'enginesize', 'boreratio', 'horsepower', 'price', 'brand_category', 'mileage']]
228. auto.head()
229.
230. ## Visualising the Data
231. plt.figure(figsize=(15, 15))
232. sns.pairplot(auto)
233. plt.show()
234.
235. plt.figure(figsize=(10, 20))
236. plt.subplot(4,2,1)
237. sns.boxplot(x = 'fueltype', y = 'price', data = auto)
238. plt.subplot(4,2,2)
239. sns.boxplot(x = 'aspiration', y = 'price', data = auto)
240. plt.subplot(4,2,3)
241. sns.boxplot(x = 'carbody', y = 'price', data = auto)
242. plt.subplot(4,2,4)
243. sns.boxplot(x = 'drivewheel', y = 'price', data = auto)
244. plt.subplot(4,2,5)
245. sns.boxplot(x = 'enginetype', y = 'price', data = auto)
246. plt.subplot(4,2,6)
247. sns.boxplot(x = 'brand_category', y = 'price', data = auto)
248. plt.subplot(4,2,7)
249. sns.boxplot(x = 'cylindernumber', y = 'price', data = auto)
250. plt.tight_layout()
251. plt.show()
252.
253. ### Data Preparation
254. cyl_no = pd.get_dummies(auto['cylindernumber'], drop_first = True)
255.
256. auto = pd.concat([auto, cyl_no], axis = 1)
257.
258. brand_cat = pd.get_dummies(auto['brand_category'], drop_first = True)
259.
260. auto = pd.concat([auto, brand_cat], axis = 1)
261.
262. eng_typ = pd.get_dummies(auto['enginetype'], drop_first = True)

```

```

263.
264. auto = pd.concat([auto, eng_typ], axis = 1)
265.
266. drwh = pd.get_dummies(auto['drivewheel'], drop_first = True)
267.
268. auto = pd.concat([auto, drwh], axis = 1)
269.
270. carb = pd.get_dummies(auto['carbody'], drop_first = True)
271.
272. auto = pd.concat([auto, carb], axis = 1)
273. asp = pd.get_dummies(auto['aspiration'], drop_first = True)
274. auto = pd.concat([auto, asp], axis = 1)
275. fuelt = pd.get_dummies(auto['fueltype'], drop_first = True)
276. auto = pd.concat([auto, fuelt], axis = 1)
277. auto.drop(['fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginetype',
'cylindernumber', 'brand_category'], axis = 1, inplace = True)
278.
279. ### Model Building
280.
281. ##### Splitting the Data into Training and Testing sets
282.
283. from sklearn.model_selection import train_test_split
284. np.random.seed(0)
285. df_train, df_test = train_test_split(auto, train_size = 0.7, test_size = 0.3, random_state = 100)
286.
287. from sklearn.preprocessing import MinMaxScaler
288. scaler = MinMaxScaler()
289. num_vars = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize', 'bore_ratio',
'horsepower', 'price', 'mileage']
290.
291. df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
292.
293. df_train.head()
294.
295. df_train.describe()
296.
297. plt.figure(figsize = (16, 10))
298. sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
299. plt.show()
300.
301. y_train = df_train.pop('price')
302. X_train = df_train
303.
304. from sklearn.feature_selection import RFE
305. from sklearn.linear_model import LinearRegression
306.
307. lm = LinearRegression()
308. lm.fit(X_train, y_train)
309.
310. rfe = RFE(lm, 10) # running RFE
311. rfe = rfe.fit(X_train, y_train)
312.
313. list(zip(X_train.columns, rfe.support_, rfe.ranking_))
314.
315. col = X_train.columns[rfe.support_]
316. col
317.
318. X_train_rfe = X_train[col]
319.
320. import statsmodels.api as sm
321. X_train_rfe = sm.add_constant(X_train_rfe)
322.
323. lm = sm.OLS(y_train, X_train_rfe).fit() # Running the linear model
324.
325. print(lm.summary())
326.
327. from statsmodels.stats.outliers_influence import variance_inflation_factor
328.
329. vif = pd.DataFrame()
330. X = X_train_rfe
331. vif['Features'] = X.columns
332. vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
333. vif['VIF'] = round(vif['VIF'], 2)

```

```

334. vif = vif.sort_values(by = "VIF", ascending = False)
335. vif
336.
337. X_train_new1 = X_train_rfe.drop(["twelve"], axis = 1)
338.
339.
340. # Adding a constant variable
341. import statsmodels.api as sm
342. X_train_lm = sm.add_constant(X_train_new1)
343.
344. lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model
345.
346. #Let's see the summary of our linear model
347. print(lm.summary())
348.
349. X_train_new2 = X_train_new1.drop(["mileage"], axis = 1)
350.
351.
352. # Adding a constant variable
353. import statsmodels.api as sm
354. X_train_lm = sm.add_constant(X_train_new2)
355.
356. lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model
357.
358. #Let's see the summary of our linear model
359. print(lm.summary())
360.
361. from statsmodels.stats.outliers_influence import variance_inflation_factor
362.
363. vif = pd.DataFrame()
364. X = X_train_new2
365. vif['Features'] = X.columns
366. vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
367. vif['VIF'] = round(vif['VIF'], 2)
368. vif = vif.sort_values(by = "VIF", ascending = False)
369. vif
370.
371. X_train_new3 = X_train_new2.drop(["curbweight"], axis = 1)
372.
373.
374. # Adding a constant variable
375. import statsmodels.api as sm
376. X_train_lm = sm.add_constant(X_train_new3)
377.
378. lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model
379.
380. #Let's see the summary of our linear model
381. print(lm.summary())
382.
383. from statsmodels.stats.outliers_influence import variance_inflation_factor
384.
385. vif = pd.DataFrame()
386. X = X_train_new3
387. vif['Features'] = X.columns
388. vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
389. vif['VIF'] = round(vif['VIF'], 2)
390. vif = vif.sort_values(by = "VIF", ascending = False)
391. vif
392.
393. X_train_new4 = X_train_new3.drop(["sedan"], axis = 1)
394.
395.
396. # Adding a constant variable
397. import statsmodels.api as sm
398. X_train_lm = sm.add_constant(X_train_new4)
399.
400. lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model
401.
402. #Let's see the summary of our linear model
403. print(lm.summary())
404.
405. X_train_new5 = X_train_new4.drop(["wagon"], axis = 1)
406.

```

```

407.
408. # Adding a constant variable
409. import statsmodels.api as sm
410. X_train_lm = sm.add_constant(X_train_new5)
411.
412. lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model
413.
414. #Let's see the summary of our linear model
415. print(lm.summary())
416.
417. from statsmodels.stats.outliers_influence import variance_inflation_factor
418.
419. vif = pd.DataFrame()
420. X = X_train_new5
421. vif['Features'] = X.columns
422. vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
423. vif['VIF'] = round(vif['VIF'], 2)
424. vif = vif.sort_values(by = "VIF", ascending = False)
425. vif
426.
427. X_train_new6 = X_train_new5.drop(["dohcv"], axis = 1)
428.
429.
430. # Adding a constant variable
431. import statsmodels.api as sm
432. X_train_lm = sm.add_constant(X_train_new6)
433.
434. lm = sm.OLS(y_train,X_train_lm).fit() # Running the linear model
435.
436. #Let's see the summary of our linear model
437. print(lm.summary())
438.
439. y_train_price = lm.predict(X_train_lm)
440.
441. fig = plt.figure()
442. sns.distplot((y_train - y_train_price), bins = 20)
443. fig.suptitle('Error Terms', fontsize = 20) # Plot heading
444. plt.xlabel('Errors', fontsize = 18) # X-label
445.
446. ### Making Predictions
447.
448. num_vars = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize','boreatio',
'horsepower', 'price','mileage']
449.
450. df_test[num_vars] = scaler.transform(df_test[num_vars])
451.
452. y_test = df_test.pop('price')
453. X_test = df_test
454.
455. X_test_new = X_test[['carwidth', 'horsepower', 'Luxury', 'hatchback']]
456.
457. # Adding a constant variable
458. X_test_new = sm.add_constant(X_test_new)
459.
460. y_pred = lm.predict(X_test_new)
461.
462. from sklearn.metrics import r2_score
463. r2_score(y_test, y_pred)
464.
465. fig = plt.figure()
466. plt.scatter(y_test,y_pred)
467. fig.suptitle('y_test vs y_pred', fontsize=20) # Plot heading
468. plt.xlabel('y_test', fontsize=18) # X-label
469. plt.ylabel('y_pred', fontsize=16) # Y-label
470.
471.
472.

```

CHAPTER 7

SNAPSHOTS

7.1 OUTPUT

In [4]: 1 df.head()

Out[4]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	engineLocation	wheelbase	...	enginesize	fuelsystem	boreRatio	stroke
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	104.11707
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	104.11707
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	104.11707
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	104.11707
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	104.11707

5 rows × 26 columns

In [6]: 1 df.describe()

Out[6]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight	enginesize	boreRatio	stroke	compressionratio	horsepower
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	103.000000	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329756	3.255415	10.142537	104.11707
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.270844	0.313597	3.972040	39.544167
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000	70.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.580000	3.410000	9.400000	116.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	288.000000

In [7]: 1 df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
#   Column              Non-Null Count  Dtype
---  -
0   car_ID              205 non-null    int64
1   symboling           205 non-null    int64
2   CarName             205 non-null    object
3   fueltype            205 non-null    object
4   aspiration           205 non-null    object
5   doornumber          205 non-null    object
6   carbody             205 non-null    object
7   drivewheel          205 non-null    object
8   engineLocation      205 non-null    object
9   wheelbase           205 non-null    float64
10  carlength            205 non-null    float64
11  carwidth             205 non-null    float64
12  carheight            205 non-null    float64
13  curbweight           205 non-null    int64
14  enginetype           205 non-null    object
15  cylindernumber       205 non-null    object
16  enginesize           205 non-null    int64
17  fuelsystem           205 non-null    object
18  boreRatio            205 non-null    float64
19  stroke              205 non-null    float64
20  compressionratio     205 non-null    float64
21  horsepower           205 non-null    int64
22  peakrpm             205 non-null    int64
23  citympg              205 non-null    int64
24  highwaympg          205 non-null    int64
25  price               205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

```
In [10]: 1 df.isnull().sum()*100/df.shape[0]
```

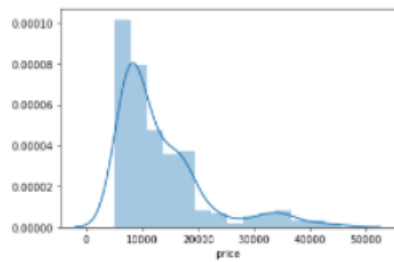
```
Out[10]: car_ID      0.0
symboling    0.0
CarName      0.0
fueltype     0.0
aspiration   0.0
doornumber   0.0
carbody      0.0
drivewheel   0.0
engine        0.0
wheelbase    0.0
carlength    0.0
carwidth     0.0
carheight    0.0
curbweight   0.0
enginetype   0.0
cylindernumber 0.0
engine_size  0.0
fuelsystem   0.0
bore_ratio   0.0
stroke       0.0
compressionratio 0.0
horsepower   0.0
peakrpm      0.0
citympg      0.0
highwaympg   0.0
price        0.0
dtype: float64
```

```
In [11]: 1 df.price.describe()
```

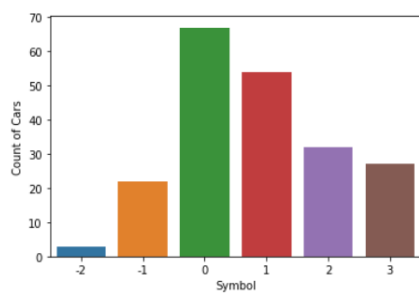
```
Out[11]: count      205.000000
mean      13276.710571
std       7988.852332
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      16503.000000
max      45400.000000
Name: price, dtype: float64
```

```
In [12]: 1 sns.distplot(df['price'])
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1d638400ee0>
```

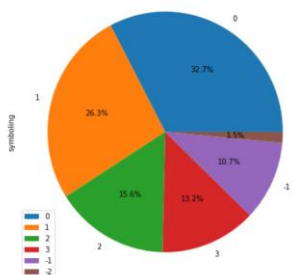


```
1 plt1 = sns.countplot(df['symboling'])
2 plt1.set(xlabel = 'Symbol', ylabel= 'Count of Cars')
3 plt1.show()
4 plt.tight_layout()
```



<Figure size 432x288 with 0 Axes>

```
1 df_sym = pd.DataFrame(df['symboling'].value_counts())
2 df_sym.plot.pie(subplots=True, labels = df_sym.index.values, autopct='%1.1F%%', figsize = (15,7.5))
3 # Unquish the pie.
4 plt.gca().set_aspect('equal')
5 plt.show()
6 plt.tight_layout()
```

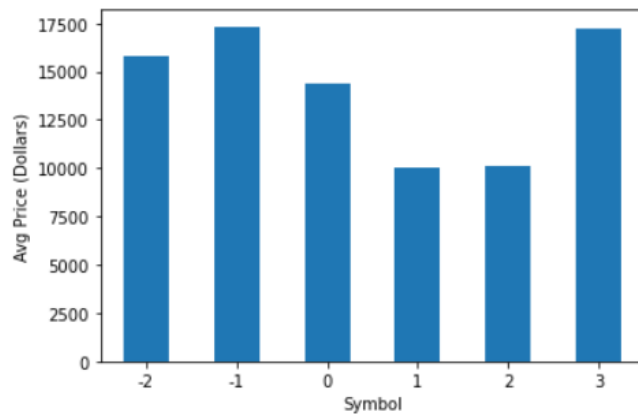


<Figure size 432x288 with 0 Axes>

```

1 plt1 = df[['symboling', 'price']].groupby("symboling").mean().plot(kind='bar', legend = False,)
2 plt1.set_xlabel("Symbol")
3 plt1.set_ylabel("Avg Price (Dollars)")
4 xticks(rotation = 0)
5 plt.show()

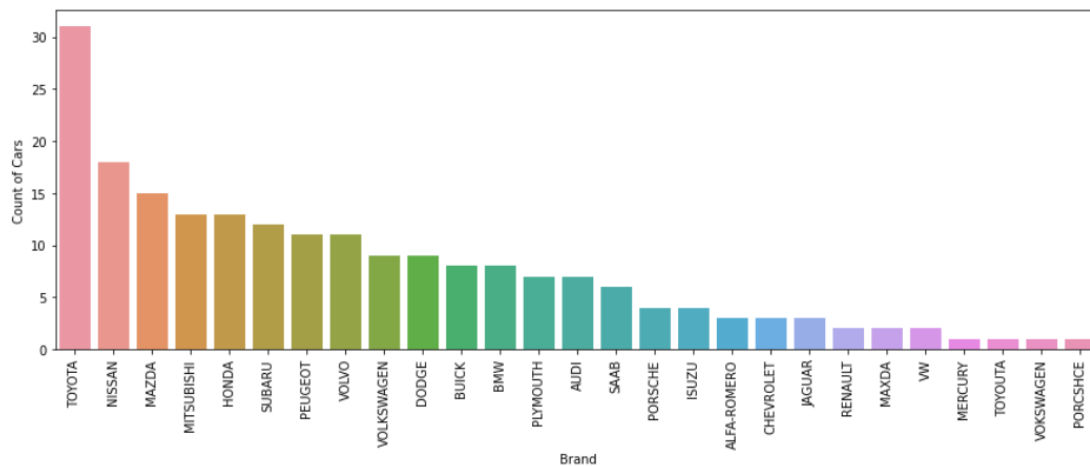
```



```

1 fig, ax = plt.subplots(figsize = (15,5))
2 plt1 = sns.countplot(df['brand'], order=pd.value_counts(df['brand']).index,)
3 plt1.set(xlabel = 'Brand', ylabel= 'Count of Cars')
4 xticks(rotation = 90)
5 plt.show()
6 plt.tight_layout()

```

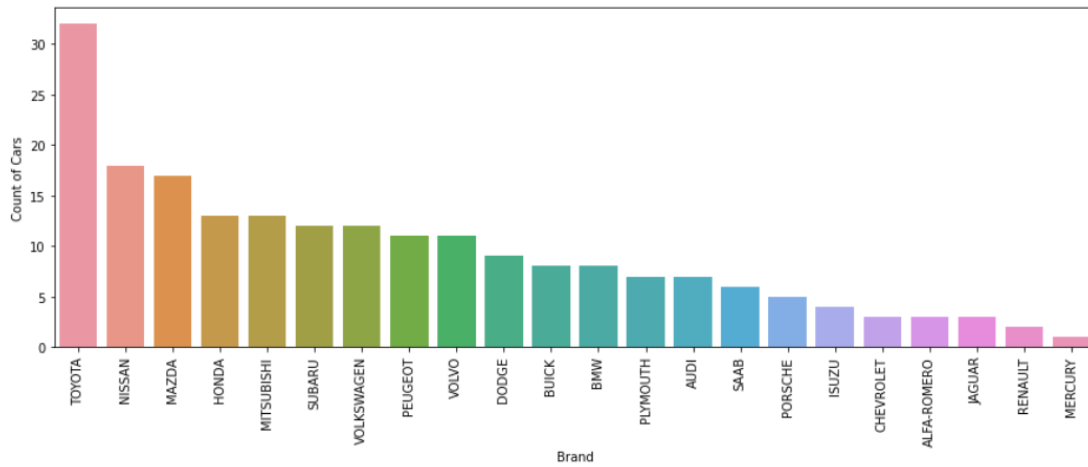


<Figure size 432x288 with 0 Axes>

```

1 fig, ax = plt.subplots(figsize = (15,5))
2 plt1 = sns.countplot(df['brand'], order=pd.value_counts(df['brand']).index,)
3 plt1.set(xlabel = 'Brand', ylabel = 'Count of Cars')
4 xticks(rotation = 90)
5 plt.show()
6 plt.tight_layout()

```

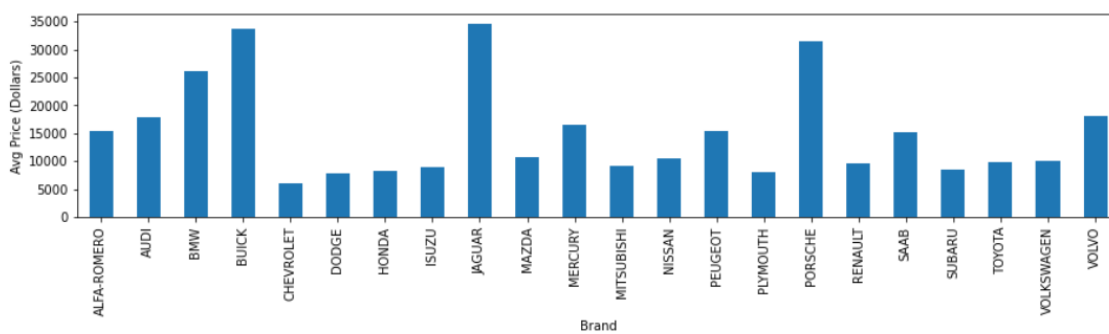


<Figure size 432x288 with 0 Axes>

```

1 df_comp_avg_price = df[['brand','price']].groupby("brand", as_index = False).mean().rename(columns={'price':'brand_avg_price'})
2 plt1 = df_comp_avg_price.plot(x = 'brand', kind='bar', legend = False, sort_columns = True, figsize = (15,3))
3 plt1.set_xlabel("Brand")
4 plt1.set_ylabel("Avg Price (Dollars)")
5 xticks(rotation = 90)
6 plt.show()

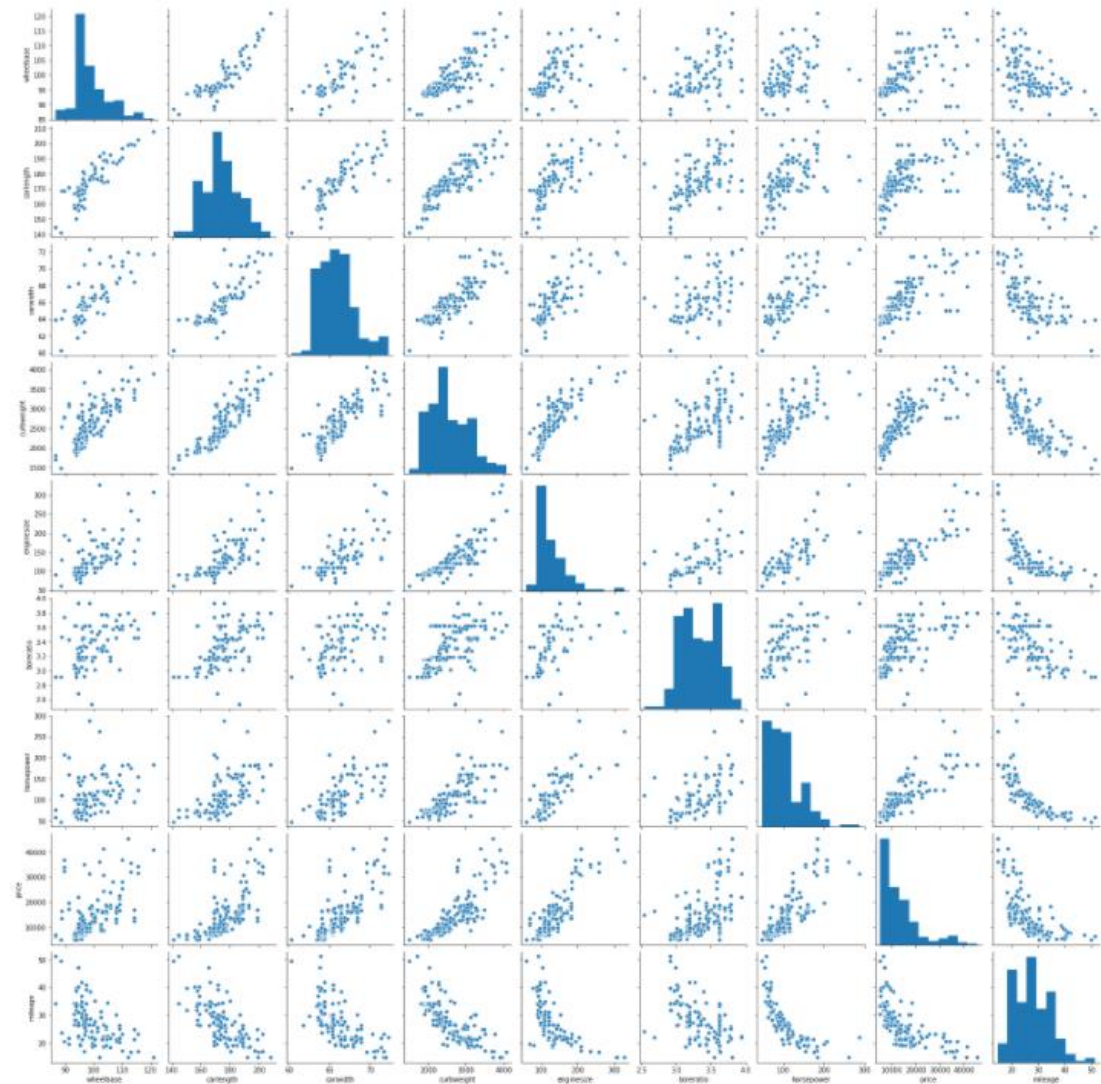
```



VISUALIZING THE DATA

```
In [43]: 1 plt.figure(figsize=(15, 15))
2         sns.pairplot(auto)
3         plt.show()
```

<Figure size 1080x1080 with 0 Axes>



TRAINING DATA

In [57]: 1 df_train.head()

Out[57]:

	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	horsepower	price	mileage	five	...	ohcv	rotor	fwd	rwd	hardtop	hatchback
122	0.244828	0.426016	0.291667	0.272692	0.139623	0.230159	0.083333	0.068818	0.530864	0	...	0	0	1	0	0	0
125	0.272414	0.452033	0.666667	0.500388	0.339623	1.000000	0.395833	0.466890	0.213992	0	...	0	0	0	1	0	1
166	0.272414	0.448780	0.308333	0.314973	0.139623	0.444444	0.266667	0.122110	0.344307	0	...	0	0	0	1	0	1
1	0.068966	0.450407	0.316667	0.411171	0.260377	0.626984	0.262500	0.314446	0.244170	0	...	0	0	0	1	0	0
199	0.610345	0.775610	0.575000	0.647401	0.260377	0.746032	0.475000	0.382131	0.122085	0	...	0	0	0	1	0	0

5 rows × 31 columns

In [58]: 1 df_train.describe()

Out[58]:

	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	horsepower	price	mileage	five	...	ohcv	rotor
count	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	143.000000	...	143.000000	143.000000
mean	0.411141	0.525476	0.461655	0.407878	0.241351	0.497946	0.227302	0.219310	0.358265	0.062937	...	0.062937	0.027972
std	0.205581	0.204848	0.184517	0.211269	0.154619	0.207140	0.165511	0.215682	0.185980	0.243703	...	0.243703	0.165472
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
25%	0.272414	0.399187	0.304167	0.245539	0.135849	0.305556	0.091667	0.067298	0.198903	0.000000	...	0.000000	0.000000
50%	0.341379	0.502439	0.425000	0.355702	0.184906	0.500000	0.191667	0.140343	0.344307	0.000000	...	0.000000	0.000000
75%	0.503448	0.669919	0.550000	0.559542	0.301887	0.682540	0.283333	0.313479	0.512346	0.000000	...	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000

8 rows × 31 columns

Making Predictions

```
In [81]: 1 num_vars = ['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesize', 'boreratio', 'horsepower', 'price', 'mileage']
2
3 df_test[num_vars] = scaler.transform(df_test[num_vars])
```

```
In [82]: 1 y_test = df_test.pop('price')
2 X_test = df_test
```

```
In [83]: 1 X_test_new = X_test[['carwidth', 'horsepower', 'Luxury', 'hatchback']]
2
3 # Adding a constant variable
4 X_test_new = sm.add_constant(X_test_new)
```

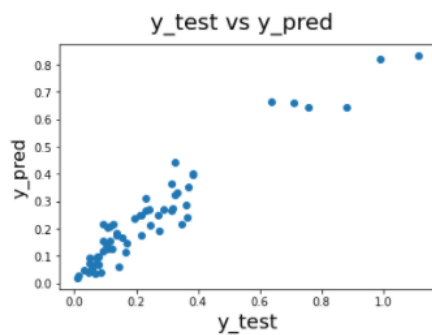
```
In [84]: 1 y_pred = lm.predict(X_test_new)
```

```
In [85]: 1 from sklearn.metrics import r2_score
2 r2_score(y_test, y_pred)
```

Out[85]: 0.8986678382302791

```
In [86]: 1 fig = plt.figure()
2 plt.scatter(y_test, y_pred)
3 fig.suptitle('y_test vs y_pred', fontsize=20)           # Plot heading
4 plt.xlabel('y_test', fontsize=18)                     # X-label
5 plt.ylabel('y_pred', fontsize=16)                     # Y-label
```

Out[86]: Text(0, 0.5, 'y_pred')



CHAPTER 8

SOFTWARE TESTING

8.1 GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

8.2 DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

8.3 Types of Tests

8.3.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.3.2 Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures: interfacing systems or procedures must be invoked.

8.3.3 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

8.3.4 Performance Test

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

8.3.5 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

8.3.6 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing for Data Synchronization:

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

8.2.7 Build the test plan

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identify the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

CHAPTER 9

FUTURE ENHANCEMENT

9.1 FUTURE ENHANCEMENT

Although, this system has achieved astonishing performance in car price prediction problem our aim for the future research is to test this system to work successfully with various data sets. We will extend our test data with eBay and OLX used cars data sets and validate the proposed approach.

CHAPTER 10

CONCLUSION & REFERENCE

10.1 CONCLUSION

Car price prediction can be a challenging task due to the high number of attributes that should be considered for the accurate prediction. The major step in the prediction process is collection and preprocessing of the data. In this research, PHP scripts were built to normalize, standardize and clean data to avoid unnecessary noise for machine learning algorithms.

Data cleaning is one of the processes that increases prediction performance, yet insufficient for the cases of complex data sets as the one in this research. Applying single machine algorithm on the data set accuracy was less than 50%. Therefore, the ensemble of multiple machine learning algorithms has been proposed and this combination of ML methods gains accuracy of 92.38%. This is significant improvement compared to single machine learning method approach. However, the drawback of the proposed system is that it consumes much more computational resources than single machine learning algorithm.

REFERENCES

- [1] Agencija za statistiku BiH. (n.d.), retrieved from: <http://www.bhas.ba>. [accessed July 18, 2018.]
- [2] Listiani, M. (2009). Support vector regression analysis for price prediction in a car leasing application (Doctoral dissertation, Master thesis, TU Hamburg-Harburg).
- [3] Richardson, M. S. (2009). Determinants of used car resale value. Retrieved from: <https://digitalcc.coloradocollege.edu/islandora/object/coccc%3A1346> [accessed: August 1, 2018.]
- [4] Wu, J. D., Hsu, C. C., & Chen, H. C. (2009). An expert system of price forecasting for used cars using adaptive neuro-fuzzy inference. *Expert Systems with Applications*, 36(4), 7809-7817.
- [5] Du, J., Xie, L., & Schroeder, S. (2009). Practice Prize Paper—PIN Optimal Distribution of Auction Vehicles System: Applying Price Forecasting, Elasticity Estimation, and Genetic Algorithms to Used-Vehicle Distribution. *Marketing Science*, 28(4), 637-644.
- [6] Gongqi, S., Yansong, W., & Qiang, Z. (2011, January). New Model for Residual Value Prediction of the Used Car Based on BP Neural Network and Nonlinear Curve Fit. In *Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on* (Vol. 2, pp. 682-685). IEEE.
- [7] Pudaruth, S. (2014). Predicting the price of used cars using machine learning techniques. *Int. J. Inf. Comput. Technol*, 4(7), 753-764. Noor, K., & Jan, S. (2017).
- [8] Vehicle Price Prediction System using Machine Learning Techniques. *International Journal of Computer Applications*, 167(9), 27-31.
- [9] Auto pijaca BiH. (n.d.), Retrieved from: <https://www.autopijaca.ba>. [accessed August 10, 2018].
- [10] Weka 3 - Data Mining with Open-Source Machine Learning Software in Java. (n.d.), Retrieved from: <https://www.cs.waikato.ac.nz/ml/weka/>. [August 04, 2018].
- [11] Ho, T. K. (1995, August). Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on* (Vol. 1, pp. 278-282). IEEE.
- [12] Russell, S. (2015). *Artificial Intelligence: A Modern Approach* (3rd edition). PE.
- [13] Ben-Hur, A., Horn, D., Siegelman, H. T., & Vapnik, V. (2001). Support vector clustering. *Journal of machine learning research*, 2(Dec), 125-137.
- [14] Aizerman, M. A. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25, 821-837.