Introduction To Mathematical Data Analytics

Dr. Mai Dao

FINAL PROJECT REPORT

Team - 6

Sai Teja Bheema

Jonathan Paul Gallegos

Oso Akindele Odunayo

Shanthanam Sangar

1.Background

In today's world, cars are an essential part of our day-to-day life for most people. No matter where you live in the world, a car is needed to transport you to work, the store or anywhere in today's large metropolitan areas. Therefore, it is so important to study the cost of used cars because this has a significant effect on people's livelihood. Most people, especially ones living in rural and farming areas, are looking for their own car. A side effect of an increase of diverse job needs and recreational priority is the greater need for individual transportation, versus the previous group transportation practices of our past. It was typical that a family would only have one car, this was because of how expensive cars were previously, but also due to only one subset of the family needing to leave at a time. The parents must now utilize multiple vehicles to transport children to school or other activities while the parent or parents go to their respective jobs. The car market both used and new is increasing year to year, and it is important to understand where the market is going and what is affecting it most.

Our goal is to build a regression model that can predict the price of used car, given certain attributes of any given car that we will use as attributes. For the classification part, we would like to classify whether a given car transmission is manual or automatic.

The dataset that we are using are the **CAR DETAILS FROM CAR DEKHO.CSV** dataset for all three algorithms. For Hypothesis testing, ridge regression and Naïve bayes.

The dataset is from from Kaggle.com

It has 4340 observations and 8 columns which are 'name', 'selling_price', 'km_driven', 'fuel', 'seller_type', 'transmission', 'Past_Owners', 'Age'.

Out of these 8 columns, our dependent variable is **selling_price** which we need to predict, and all other columns are our independent or explanatory variables.

2.Literature Review

2.1 Hypothesis Testing

Hypothesis testing can be used to find evidence of which predictors have a significant effect on a regression model. As we will discuss in a moment, one of our topics and statistical models we are using to analyze our data set is regularized regression. One outcome of regularized regression is the shrinkage of coefficients towards zero which makes them biased. Therefore, when using hypothesis testing to find significant coefficients to use a linear regression model and apply our testing before modeling our regression using any other methods. Hypothesis testing is the process of determining an assumed truth, the null hypothesis, and weighing the results of a sample of the population we have made our assumption about against that truth. It typically comes in this form:

The NULL H_0: The average of a population is $$$

Versus the Alternative H_1: The average of a population is NOT $$$

This setup can be, essentially, applied to any inference about a population. After we have a clear hypothesis, we want to collect our data, and statistically test is. To do this you need to find out what distribution the population follows i.e., F distribution, T distribution, chi-squared distribution, etc. This is because, taking our example above, when we find the average of our sample of a population, we need to know how it differs from Null Hypothesis. We do this by looking at the distribution the population comes from and finding what the probability is that we would have the sample average that we do. If that probability is significantly small, typically based off a set value of significance (α) then we can reject our null hypothesis with evidence supporting the alternative hypothesis. Otherwise, we fail to reject.

2.2 Regularized Regression

When we make our regression model, we want to minimize the error sum of squares (SSE).

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$

We can do this by shrinking the coefficients B_j, known as penalizing, like this:

Ridge Regularized Regression:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Lasso Regularized Regression:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

λ is known as the tuning parameter and is responsible for controlling the strength of the penalty, if λ = 0 then Regularized Regression simply becomes Least Squares Regression. When λ is sufficiently large, $\hat{B}_\lambda$ will approach zero or, in Lasso cases, equally zero. λ can be found by using Cross Validation (CV). CV is the average of all the Mean Square Errors (MSEs) found by creating models through training data and finding the error between the predicted values from your training model, and your actual values of your test/validation set. λ is chosen by running a range of λ in your training sets and finding the smallest CV. You can then refit your model with your selected value of λ.

2.3 Naïve Bayes

When trying to find the probability of a class, we can use Bayes Classifiers. This allows us to find the probability of a response class occurring, given a class of predicter has occurred. The equation for this closely follows Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{\sum_i P(B|A_i)P(A_i)}$$

$$p_k(x) = P(Y = k \mid X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}.$$

Where π_k or P(Y = k) is the probability that a randomly chosen observation comes from the kth class of the response. Another way to say this would be the proportion of responses in the kth class. $f_k(x)$ or P(X = x|Y = k) is the density function of X given Y = k. Having both those probabilities we can find the conditional probability $p_k(x)$ which, as we stated above, is the probability of a response clasas occurring, given a class of the kth predicter has occurred. This is all what makes up Bayes Classifiers, and how we get LDA, QDA and Naïve Bayes. But what distinguishes those three mentioned?

Each have to do with assumptions made about $f_k(x)$. For Naïve Bayes, we assume independence between $f_{ki}(x)$ so we get this equation for $f_k(x)$:

$$f_k(x) = f_{k1}(x_1) \times \cdots \times f_{kp}(x_p), \quad k = 1, \ldots, K.$$

This means that in order to use Naïve Bayes, we need to assume that the density functions for each X_i given Y_k are independent of each other.

3.Methodology

3.1 DATA PREPROCESSING

This initial analysis was done in order to be familiar with data set and to do data preprocessing.

```
                         name  year  selling_price  km_driven    fuel  \
0               Maruti 800 AC  2007          60000      70000  Petrol
1    Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
2        Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3      Datsun RediGO T Option  2017         250000      46000  Petrol
4       Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner
0  Individual       Manual   First Owner
1  Individual       Manual   First Owner
2  Individual       Manual   First Owner
3  Individual       Manual   First Owner
4  Individual       Manual  Second Owner
```

Figure 1

Figure 1. gives brief summary on the data set. The data set contains 8 variables,4340 entries. Numeric and string type data are they're in the data set. Below table summarizes about data types and missing values. There is no missing value in the data set.

```
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
```

Data	Missing Values
name	0
year	0
selling_price	0
km_driven	0
fuel	0
seller_type	0
transmission	0
owner	0

Figure 2

From the Figure 2 it is clear that there are no missing values in our dataset.

Manufacturing year is converted as Age by subtracting manufacture year by this year 2022 in order to make the analysis reliable.

	name	selling_price	km_driven	fuel	seller_type	\
0	Maruti 800 AC	60000	70000	Petrol	Individual	
1	Maruti Wagon R LXI Minor	135000	50000	Petrol	Individual	
2	Hyundai Verna 1.6 SX	600000	100000	Diesel	Individual	
3	Datsun RediGO T Option	250000	46000	Petrol	Individual	
4	Honda Amaze VX i-DTEC	450000	141000	Diesel	Individual	

	transmission	owner	Age
0	Manual	First Owner	15
1	Manual	First Owner	15
2	Manual	First Owner	10
3	Manual	First Owner	5
4	Manual	Second Owner	8

Figure 3

From figure 3, we can see that owner field has ordinal type so, converting this to numerical by changing individual levels like 'New Car', 'First Owner', 'Second Owner', 'third Owner' and so on to 0,1,2,3,4.

3.2 DESCRIPTIVE STATISTICS

	selling_price	km_driven	Past_Owners	Age
count	4.340000e+03	4340.000000	4340.000000	4340.000000
mean	5.041273e+05	66215.777419	1.447005	8.909217
std	5.785487e+05	46644.102194	0.712191	4.215344
min	2.000000e+04	1.000000	0.000000	2.000000
25%	2.087498e+05	35000.000000	1.000000	6.000000
50%	3.500000e+05	60000.000000	1.000000	8.000000
75%	6.000000e+05	90000.000000	2.000000	11.000000
max	8.900000e+06	806599.000000	4.000000	30.000000

Figure 4

Figure 4, shows the descriptive statistics of our dataset. From that we can see the distribution of fields like selling price, kilometers driven, Past owners, Age of the car.

3.2 Exploratory Data Analysis (EDA)

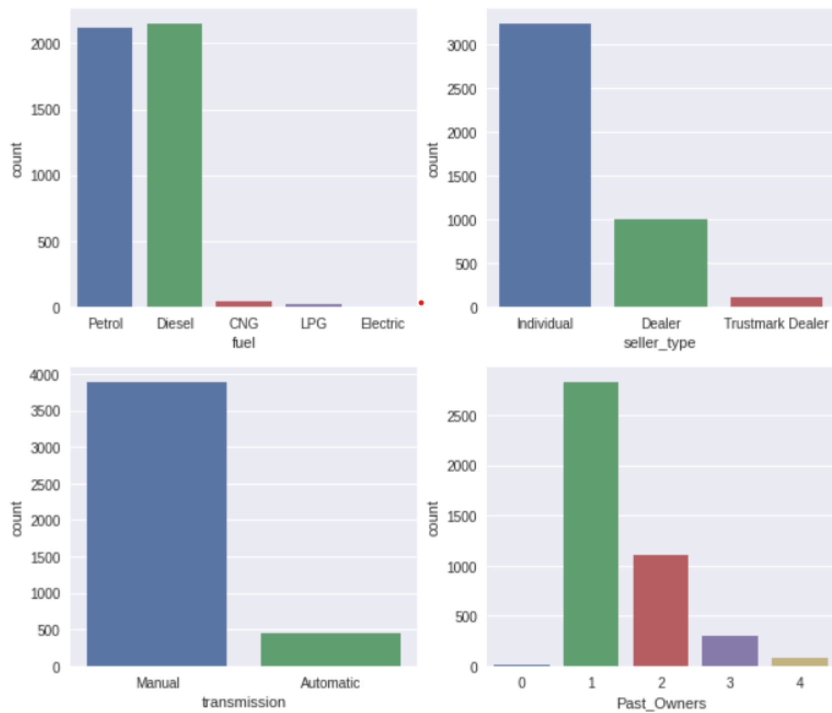


Figure 5

Next, Univariate analysis was done in order to understand the data. From Figure 5, we can see the different class distribution in the categorical variables. Figure 6, shows distribution in field Age.

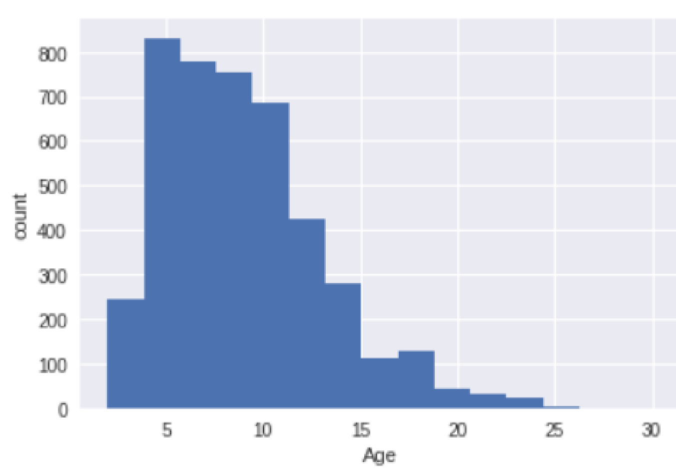


Figure 6

3.3 OUTLIER ANALYSIS

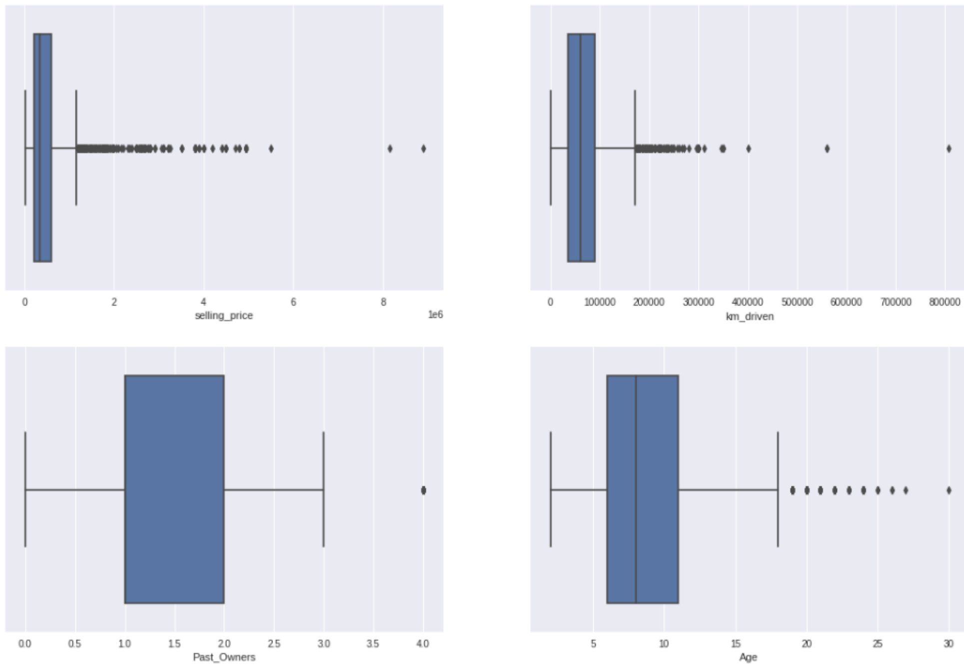


Figure 7

We plot box plot on the numerical data to check outliers in the data. From Figure 7, we can see that most of the outliers show in the plots are not by any error or data entry. Some cars have extreme driven miles on them, there can be expensive cars, and age of a used car can be in 20s to 30s.

3.3 CORRELATION ANALYSIS

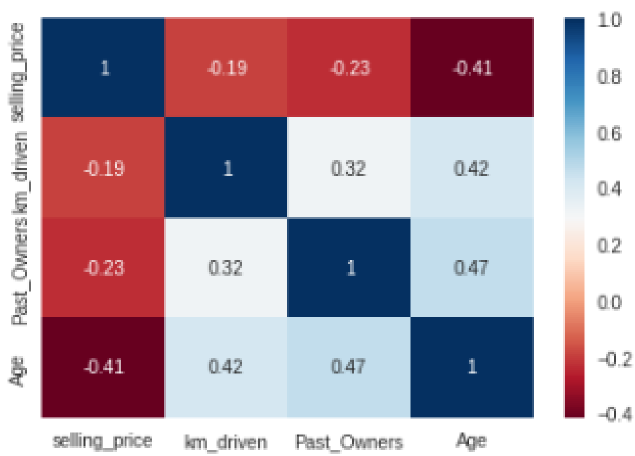


Figure 8

Below multivariate analysis/heat map was created to analyze the interaction/correlation between variables. Based on correlation analysis selling price has only negative correlation age, kms driven and past owners.

3.4 DATA PREPARATION

In this step raw data is processed and converted into a comprehensible format. Collected data may have different formats. Directly analyzing poor quality data will make poor quality models. So, preprocessing the data will improve the accuracy of the created model as well as machine can process the data.

3.4.1 Creating Dummies:

Numerical variable is used for categorical features. In this step dummy variables were created for the categories such as fuel type, seller type, transmission. Below table shows the data after creating dummy variables.

	selling_price	km_driven	Past_Owners	Age	fuel_CNG	fuel_Diesel	fuel_Electric	fuel_LPG	fuel_Petrol	seller_type_Dealer	seller_type_Individual	seller_type_Trustmark Dealer	transmission_Automatic	transmission_Manual
0	60000	0.086783	0.25	0.464286	0	0	0	0	1	0	1	0	0	1
1	135000	0.061988	0.25	0.464286	0	0	0	0	1	0	1	0	0	1
2	600000	0.123976	0.25	0.285714	0	1	0	0	0	0	1	0	0	1
3	250000	0.057028	0.25	0.107143	0	0	0	0	1	0	1	0	0	1
4	450000	0.174807	0.50	0.214286	0	1	0	0	0	0	1	0	0	1

Figure 9

3.4.2 Normalization:

The normalization was done in order to transform the data in the columns to same scale. It increases the quality of the data. Also, normalization can help on identifying duplicates. Below figure shows the data after normalization. We are normalizing the data before(beginning) we are performing the regression and classification. We have normalized all the numerical variables except the target variable (selling price) as we have to predict the selling price.

	name	selling_price	km_driven	fuel	seller_type	transmission	Past_Owners	Age
0	Maruti 800 AC	60000	0.086783	Petrol	Individual	Manual	0.25	0.464286
1	Maruti Wagon R LXI Minor	135000	0.061988	Petrol	Individual	Manual	0.25	0.464286
2	Hyundai Verna 1.6 SX	600000	0.123976	Diesel	Individual	Manual	0.25	0.285714
3	Datsun RediGO T Option	250000	0.057028	Petrol	Individual	Manual	0.25	0.107143
4	Honda Amaze VX i-DTEC	450000	0.174807	Diesel	Individual	Manual	0.50	0.214286
...	...	...	...	...	...	...	...	...
4335	Hyundai i20 Magna 1.4 CRDi (Diesel)	409999	0.099181	Diesel	Individual	Manual	0.50	0.214286
4336	Hyundai i20 Magna 1.4 CRDi	409999	0.099181	Diesel	Individual	Manual	0.50	0.214286
4337	Maruti 800 AC BSIII	110000	0.102900	Petrol	Individual	Manual	0.50	0.392857
4338	Hyundai Creta 1.6 CRDi SX Option	865000	0.111579	Diesel	Individual	Manual	0.25	0.142857
4339	Renault KWID RXT	225000	0.049590	Petrol	Individual	Manual	0.25	0.142857

Figure 10

We are normalizing the data before(beginning) we are performing the regression and classification. We have normalized all the numerical variables except the target variable (selling price) as we have to predict the selling price.

Rest of the steps were as follows. The results and model parameters are discussed in coming chapter.

Then the data set was dividend into two groups, 1) Training Data 2) Test Data. Training Data was used to develop the models. Test data is used to test the model accuracy. Having less training data will create greater variance in estimated parameter. Having too much data in training set will cause the test data to be shrinker so that model evaluation become inaccurate.

Then, hypothesis testing is done in order to identify significant parameters in the model development. As a part of hypothesis testing to find p-values we perform OLS regression. After that removing insignificant variables from the dataset and then performing regularized regression to predict selling price of used cars. Regularization gives some penalty to certain complex models. There are two types of regularized regression.

1. Ridge Regression: this method is used when the number of predictor variables exceeds the number of observations. Performs L2 regularization, i.e., adds penalty equivalent to square of the magnitude of coefficients. Minimization objective = LS Obj + α * (sum of square of coefficients)

2. Lasso Regression: This is liner regression type model uses shrinkage, where data values are shrunk towards central point. This model is useful when we have high level of multicollinearity. Performs L2 regularization, i.e., adds penalty equivalent to square of the magnitude of coefficients. Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

In this work both the ridge regression and lasso regression models were developed and analyzed.

Naïve Bayes

Naïve bayes is supervised learning method based on Bayes theorem with the assumption of conditional independence between every pair of variables. The different naive Bayes classifiers differ based on their assumptions on the distribution of the independent variable given dependent variable.

Some of the Naïve Bayes' method categories are:

1. Gaussian Naive Bayes: in this model likelihood of the features are assumed to be Gaussian.
2. Multinomial Naive Bayes: This is used for multinomially distributed Data.
3. Complement naive Bayes: Adapted from standard multinomial Bayes algorithm and it is suitable for imbalanced data set.
4. Bernoulli Naive Bayes: this is suitable for data that is distributed in multivariate Bernoulli distributions.

We performed correlation analysis just to find if there is a correlation between predictors. From the heatmap, none of the predictors are highly correlated to the target variable (selling price). The Naive Bayes model is called "Naive" precisely because it assumes features are conditionally independent of one another despite the fact that this assumption very rarely holds. However, the Naive Bayes model can still achieve strong classification accuracy even in cases where its conditional independence assumption is significantly inaccurate. See the famous paper On the Optimality of the Simple Bayesian Classifier under 0-1 loss for a mathematically rigorous treatment of this topic.

3.5 SOFTWARE IMPLEMENTATION:

The entire project analysis is done using python programming language on Google Colab platform. Various python packages and libraries are used to perform mathematical and statistical analysis. Packages like ‘pandas’ for data manipulation and analysis. ‘Numpy’ for adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. ‘Sci-kit learn’ as it features various classification, regression and clustering algorithms. ‘statsmodels’ is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. ‘Matplotlib’ makes easy things easy and hard things possible. ‘Seaborn’ is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics

4.Results

4.1 MODEL DEVELOPMENT

Following Data Analysis techniques were used in our works to make inferences and predict prices.

1. Hypothesis testing
2. Regularized regression
3. Naïve Bayes

HYPOTHESIS TESTING:

Hypothesis testing was done to identify the significant parameters. Ordinary Least squared model was developed to find p-values for the hypothesis testing. This primarily to avoid overfitting and find the significant parameters.

Below are the developed OLS model parameters.

OLS Regression Results

Dep. Variable:	selling_price	**R-squared:**	0.459
Model:	OLS	**Adj. R-squared:**	0.457
Method:	Least Squares	**F-statistic:**	366.6
Date:	Fri, 13 May 2022	**Prob (F-statistic):**	0.00
Time:	00:01:42	**Log-Likelihood:**	-62411.
No. Observations:	4340	**AIC:**	1.248e+05
Df Residuals:	4329	**BIC:**	1.249e+05
Df Model:	10		
Covariance Type:	nonrobust		

Figure 11

Selling Price is the dependent variable and model has 10number of degrees of freedom. Hypothesis test results are summarized as below.

	coef	std err	t	P>|t|	[0.025	0.975]
km_driven	-7.87e+05	1.36e+05	-5.804	0.000	-1.05e+06	-5.21e+05
Past_Owners	-7.984e+04	4.25e+04	-1.877	0.061	-1.63e+05	3571.546
Age	-9.975e+05	5.35e+04	-18.644	0.000	-1.1e+06	-8.93e+05
fuel_CNG	2.664e+05	9.05e+04	2.943	0.003	8.9e+04	4.44e+05
fuel_Diesel	5.519e+05	7.18e+04	7.691	0.000	4.11e+05	6.93e+05
fuel_Electric	-3.568e+05	3.59e+05	-0.995	0.320	-1.06e+06	3.46e+05
fuel_LPG	3.122e+05	1.02e+05	3.050	0.002	1.12e+05	5.13e+05
fuel_Petrol	2.62e+05	7.17e+04	3.655	0.000	1.21e+05	4.03e+05
seller_type_Dealer	3.136e+05	3.18e+04	9.852	0.000	2.51e+05	3.76e+05
seller_type_Individual	2.428e+05	3.2e+04	7.598	0.000	1.8e+05	3.06e+05
seller_type_Trustmark Dealer	4.793e+05	4.37e+04	10.975	0.000	3.94e+05	5.65e+05
transmission_Automatic	9.524e+05	4.57e+04	20.855	0.000	8.63e+05	1.04e+06
transmission_Manual	8.344e+04	4.48e+04	1.862	0.063	-4436.426	1.71e+05

Omnibus:	4368.888	**Durbin-Watson:**	1.938
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	502954.806
Skew:	4.659	**Prob(JB):**	0.00

Figure 12

From the hypothesis test it can be seen that fuel Electric has a high p value therefore it can be excluded in our regression models. Transmission manual and past owners also have higher p value but they are not removed since the p-value is near the border.

REGULARIZED REGRESSION:

Before performing regression, we have split our data set into train and test, here we are using 70% of our data for training and remaining 30% for testing.

For the regularized regression we are using ridge and lasso regression,

RIDGE REGRESSION RESULTS:

So here are the results for our ridge regression, using alpha value 0.1:

```
coeffients: [-710537.98902067 -131713.24392353 -882820.63943373 -116191.77746643
  135273.23589435  -73029.50117084 -129607.50238706   23695.55922562
  -48061.51733131  205238.95037296  405678.95764597 -405678.95764597]
-----------------------------------------------------
intercept: 1169299.0970165194
-----------------------------------------------------
R2-Score: 0.43563739266011536
```

Figure 13

From the R2 score, we can say that this ridge model can only explain 43% variability of selling price using predictors like miles driven, number of past owners and so on.

LASSO REGRESSION RESULTS:

Then Lasso regression was developed with following model parameters, using alpha value as 0.1:

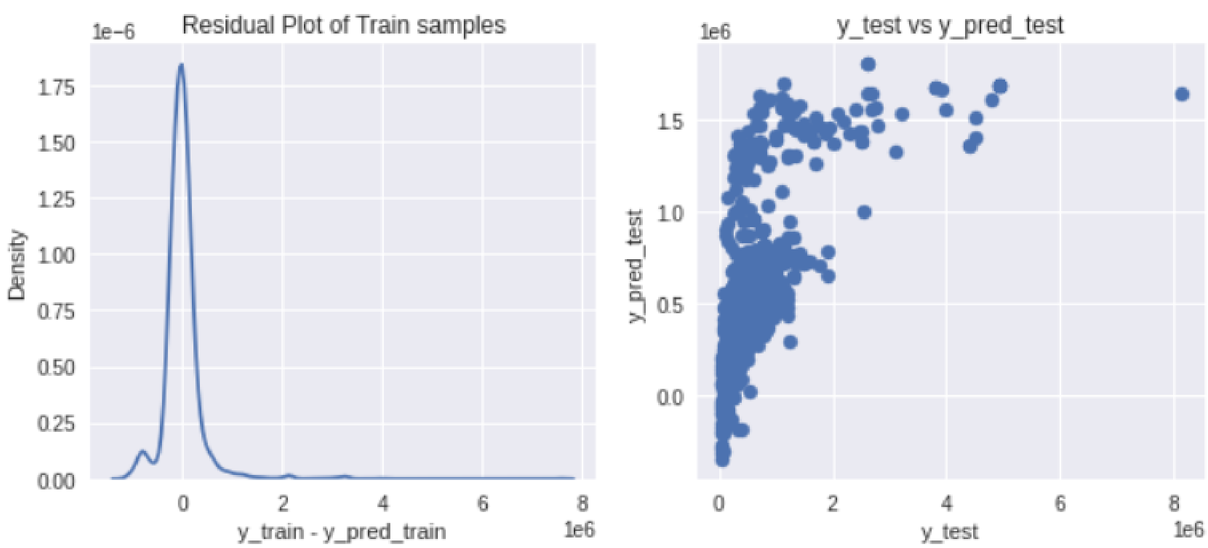
```
coeffients: [-7.21675432e+05 -1.04437370e+05 -9.71948116e+05 -0.00000000e+00
  2.69427124e+05  6.21981799e+04 -5.49140097e+03  3.03118428e+04
 -3.61965728e+04  2.26638432e+05  8.47064352e+05 -3.45795373e-08]
-----------------------------------------------------
intercept: 633163.8669866546
-----------------------------------------------------
R2-Score: 0.43991662887097205
```

From the R2 score, even lasso model can only explain 43% variability of selling price using predictors like miles driven, number of past owners and so on. As this is not a great score for a prediction we have tried using cross validation to perform regression.

RIDGE REGRESSION USING CROSS VALIDATION RESULTS:

Here ridge regression attempts to show the relationship between independent and dependent variable. In this model R^2 for test results is only 44% while MSE score remains a very high value. So, this is not good prediction model. Even the cross validation for 5 folds could not explain variability in selling price more than 50%.

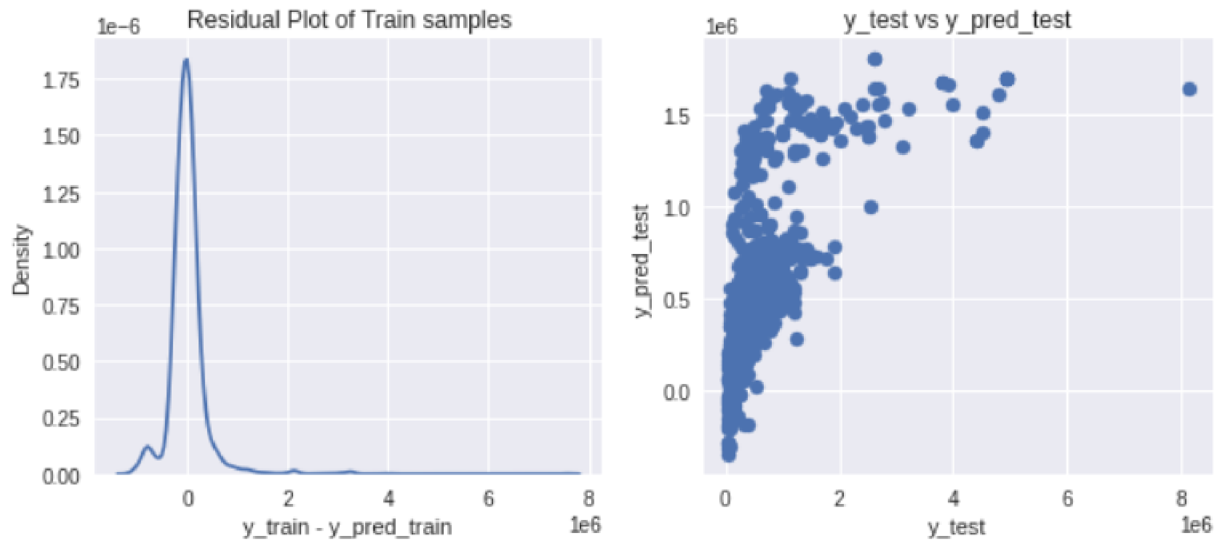
```
Train R2-score : 0.47
Test R2-score : 0.44
Train mse-score : 164308725911.3
Test mse-score : 221678113743.63
Train CV scores : [0.49188358 0.36413493 0.50123305 0.48470748 0.47001539]
Train CV mean : 0.46
```



LASSO REGRESSION USING CROSS VALIDATION RESULTS:

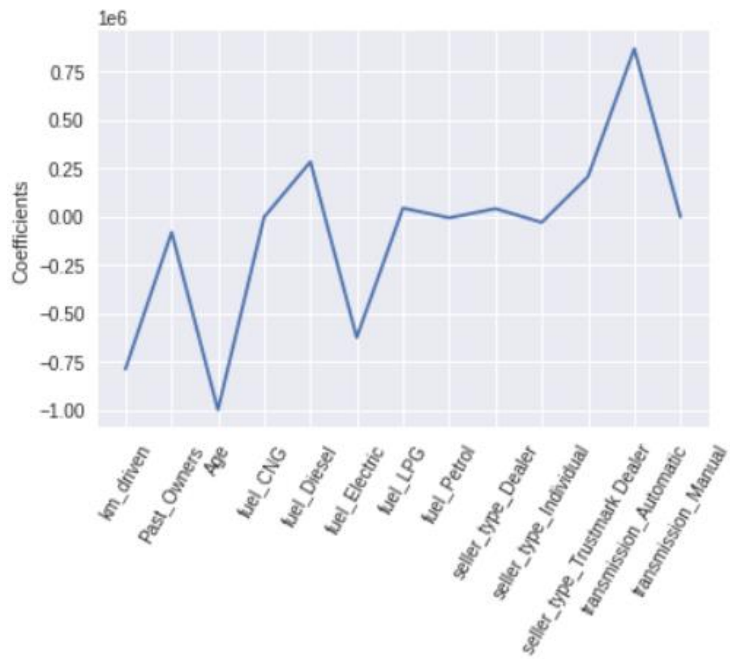
Now lasso regression attempts to show the relationship between independent and dependent variable. Even this model R^2 is similar to ridge regression, test results is only 44% while MSE score remains a very high value. So, even this is not a good prediction model. Even the cross validation for 5 folds could not explain variability in selling price more than 50%.

```
Train R2-score : 0.47
Test R2-score : 0.44
Train mse-score : 164302719481.07
Test mse-score : 221503973444.16
Train CV scores : [0.49216978 0.36499275 0.5008215  0.4847804  0.46988328]
Train CV mean : 0.46
```



At first, we got a high R2 value for test set because of the train-test split, We, did change the train-test split and initiate a random state then we got our MSE and R2 better for the train compared to test data.

Our prediction model was not doing a great job in predicting the selling price of a used car. As regularization is used for feature selection as a part of dimentionality reduction. So, we used lasso to perform feature extraction.



From the figure it is shows the impact of predictors on the variability of selling price of a used car. We can see that trusted dealer has strong positive effect and age has strong negative effect. Fields like fuel has low significance in the model.

NAÏVE BAYES:

Gaussian Naïve Bayes and Bernoulli Naïve Bayes are used classify the whether a vehicle is equipped with transmission manual or not. Figure 12 a show the confusion matrix when gaussian is used and 12 b shows the confusion matrix when Bernoulli Naïve Bayes is used.

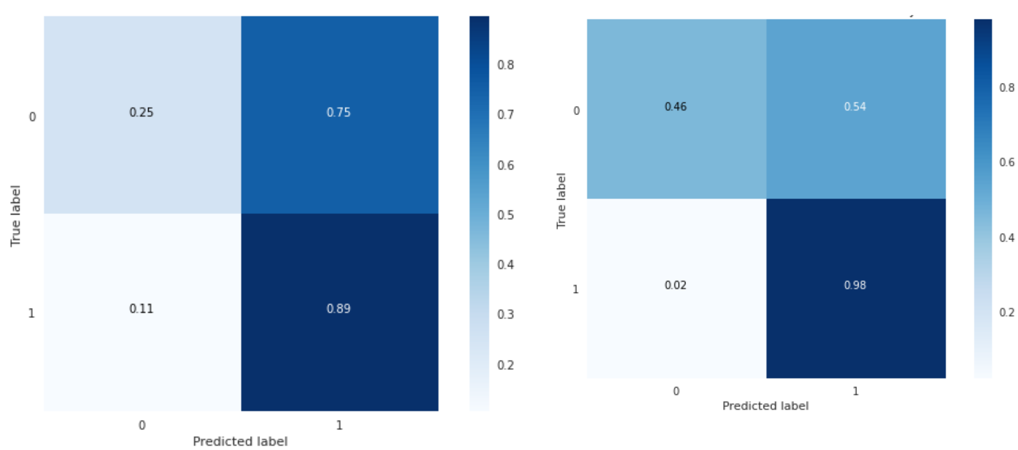


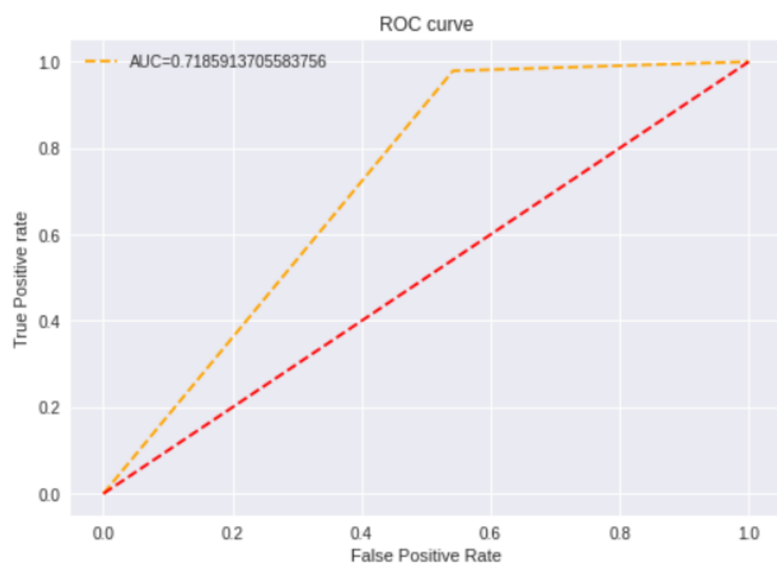
Figure 4 (a,b)

As it can be seen from above analysis, Gaussian Naïve Bayes provide high prediction accuracy this could be because the features might follow gaussian distribution. Bernoulli Naive Bayes is good at handling

Boolean/binary attributes, while Multinomial Naive Bayes is good at handling discrete values and Gaussian naive Bayes is good at handling continuous values. The predictors we have are mixed—binary attributes from dummy variables and continuous values as variables like miles driven, age, and previous owners we tried using Gaussian Naive Bayes. This model is classified with 93% accuracy. While using Bernoulli Naive Bayes model it is only able to classify with 83% accuracy. So, we went on using Gaussian Naïve Bayes as it is able to classify better.

	precision	recall	f1-score	support
0	0.69	0.46	0.55	120
1	0.95	0.98	0.96	1182
accuracy			0.93	1302
macro avg	0.82	0.72	0.76	1302
weighted avg	0.92	0.93	0.92	1302

This is the classification report of gaussian naïve bayes model classifying car transmission type.



Here is the AUC-ROC curve for the gaussian naïve bayes classifier model, showing an AUC score of 0.71.

5.Conclusion

Number of cars purchased and sold is continuously increasing therefore used car price prediction is topic of interest for the people. Data analysis is used to identify significant features which influences the price the used cars. Also, prediction model was developed to used car price prediction. The data set used for this study contains 8 variables and 4240 entries. The data was preprocessed to and checked for outliers and redundancy and converted into comprehensive format. Then the data set was splitted into training and test data set. Ordinary Least squared model was developed for the hypothesis testing. The training data to test data ratio is 70:30. Based on hypothesis test significant variables were identified and then ridge regression, and lasso regression models were created and prediction results were compared. Then used cross validation as well. But the results for the regression models were not that satisfying as the models were not even able to explain the variability of the selling price by 50%. This may be due to the data we are using as it is a mix of continuous and discrete variables and from the analysis there is not much strong correlation between the variables. So, for the future we can remove outliers from our data and then try predicting the selling price. For the classification part, Gaussian Naïve bayes was used to predict the transmission manual variable and the obtained result was 94.68 %. The results are satisfying as the classifier is able to classify transmission type with 94.4%.

6.References

Dataset-link : https://www.kaggle.com/code/mohaiminul101/car-price-prediction/data?select=CAR+DETAILS+FROM+CAR+DEKHO.csv

AlShared , A. A. S. (2021, December). Adobe Acrobat: PDF Edit, convert, sign tools.

Google. Retrieved May 12, 2022, from https://chrome.google.com/webstore/detail/adobe-acrobat-pdf-edit- co/efaidnbmnnnibpcajpcglclefindmkaj?hl=en-GB

Dao, M. (2022). *Math646 Introduction to Math Data Analysis: Chapter 2* [PowerPoint slides].

Blackboard. https://blackboard.wichita.edu/ultra/courses/_147615_1/cl/outline

Dao, M. (2022). *Math646 Introduction to Math Data Analysis: Chapter 4* [PowerPoint slides].

Blackboard. https://blackboard.wichita.edu/ultra/courses/_147615_1/cl/outline

Dao, M. (2022). *Math646 Introduction to Math Data Analysis: Chapter 5* [PowerPoint slides].

Blackboard. https://blackboard.wichita.edu/ultra/courses/_147615_1/cl/outline

MATH646-Project

May 13, 2022

0.0.1 Reading and Understanding the Dataset

[ ]:
```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings

%matplotlib inline
warnings.simplefilter(action='ignore')
plt.style.use('seaborn')
```

[ ]:
```
#load dataset
df_main = pd.read_csv('/content/CAR DETAILS FROM CAR DEKHO.csv')
```

[ ]:
```
#viewing top 5 rows of dataset
df_main.head()
```

[ ]:
```
                       name  year  selling_price  km_driven    fuel  \
0             Maruti 800 AC  2007          60000      70000  Petrol
1  Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
2      Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3    Datsun RediGO T Option  2017         250000      46000  Petrol
4     Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner
0  Individual       Manual   First Owner
1  Individual       Manual   First Owner
2  Individual       Manual   First Owner
3  Individual       Manual   First Owner
4  Individual       Manual  Second Owner
```

[ ]:
```
#checking dimensions
df_main.shape
```

[ ]: (4340, 8)

[ ]:
```
df_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

Categorical data distribution

[ ]:
```
df_main['seller_type'].value_counts()
```

[ ]:
```
Individual          3244
Dealer               994
Trustmark Dealer     102
Name: seller_type, dtype: int64
```

[ ]:
```
df_main['fuel'].value_counts()
```

[ ]:
```
Diesel      2153
Petrol      2123
CNG           40
LPG           23
Electric       1
Name: fuel, dtype: int64
```

[ ]:
```
df_main['transmission'].value_counts()
```

[ ]:
```
Manual       3892
Automatic     448
Name: transmission, dtype: int64
```

Checking for missing values

[ ]:
```
#missing values
df_main.isna().sum()
```

[ ]:
```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
dtype: int64
```

0.0.2 Data Preprocessing

[ ]:
```
df_main['Age'] = 2022 - df_main['year']     #rewriting Manufacture year as Age
df_main.drop('year',axis=1,inplace = True)
```

[ ]:
```
df_main.head()
```

[ ]:
```
                        name  selling_price  km_driven    fuel seller_type  \
0              Maruti 800 AC          60000      70000  Petrol  Individual
1   Maruti Wagon R LXI Minor         135000      50000  Petrol  Individual
2       Hyundai Verna 1.6 SX         600000     100000  Diesel  Individual
3     Datsun RediGO T Option         250000      46000  Petrol  Individual
4      Honda Amaze VX i-DTEC         450000     141000  Diesel  Individual

  transmission         owner  Age
0       Manual   First Owner   15
1       Manual   First Owner   15
2       Manual   First Owner   10
3       Manual   First Owner    5
4       Manual  Second Owner    8
```

[ ]:
```
#renaming Owners as Past_Owners
df_main.rename(columns = {'owner':'Past_Owners'},inplace = True)
```

[ ]:
```
df_main['Past_Owners'].value_counts()
```

[ ]:
```
First Owner             2832
Second Owner            1106
Third Owner              304
Fourth & Above Owner      81
Test Drive Car            17
Name: Past_Owners, dtype: int64
```

```
[ ]: df_main['Past_Owners'].replace(['First Owner'],1, inplace=True)
     df_main['Past_Owners'].replace(['Second Owner'],2, inplace=True)
     df_main['Past_Owners'].replace(['Third Owner'],3, inplace=True)
     df_main['Past_Owners'].replace(['Fourth & Above Owner'],4, inplace=True)
     df_main['Past_Owners'].replace(['Test Drive Car'],0, inplace=True)
```

```
[ ]: df_main['Past_Owners'].value_counts()
```

```
[ ]: 1    2832
     2    1106
     3     304
     4      81
     0      17
     Name: Past_Owners, dtype: int64
```

```
[ ]: #numerical stats
     df_main.describe()
```

```
[ ]:       selling_price      km_driven  Past_Owners          Age
     count   4.340000e+03    4340.000000  4340.000000  4340.000000
     mean    5.041273e+05   66215.777419     1.447005     8.909217
     std     5.785487e+05   46644.102194     0.712191     4.215344
     min     2.000000e+04       1.000000     0.000000     2.000000
     25%     2.087498e+05   35000.000000     1.000000     6.000000
     50%     3.500000e+05   60000.000000     1.000000     8.000000
     75%     6.000000e+05   90000.000000     2.000000    11.000000
     max     8.900000e+06  806599.000000     4.000000    30.000000
```

```
[ ]: df_main.head()
```

```
[ ]:                        name  selling_price  km_driven    fuel seller_type  \
     0             Maruti 800 AC          60000      70000  Petrol  Individual
     1  Maruti Wagon R LXI Minor         135000      50000  Petrol  Individual
     2      Hyundai Verna 1.6 SX         600000     100000  Diesel  Individual
     3    Datsun RediGO T Option         250000      46000  Petrol  Individual
     4     Honda Amaze VX i-DTEC         450000     141000  Diesel  Individual

       transmission  Past_Owners  Age
     0       Manual            1   15
     1       Manual            1   15
     2       Manual            1   10
     3       Manual            1    5
     4       Manual            2    8
```

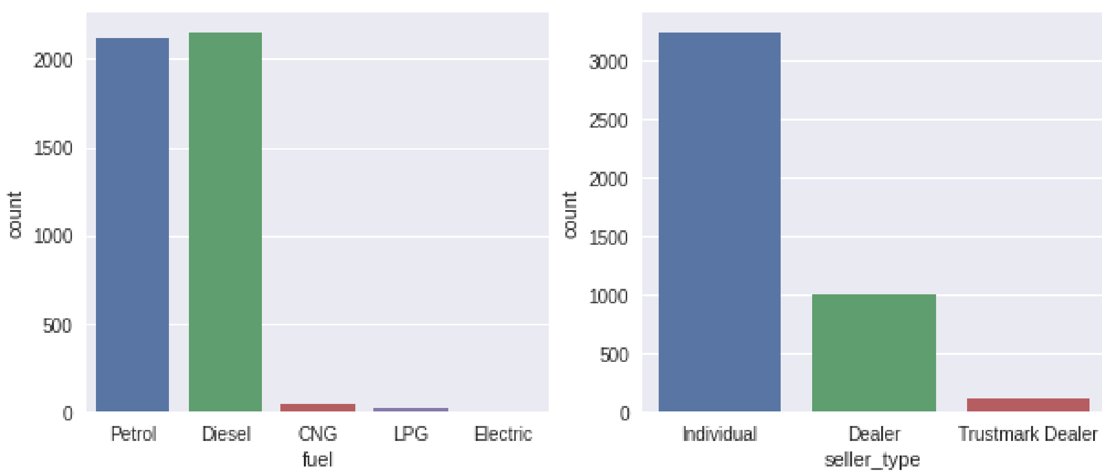
0.0.3 Exploratory Data Analysis (EDA)

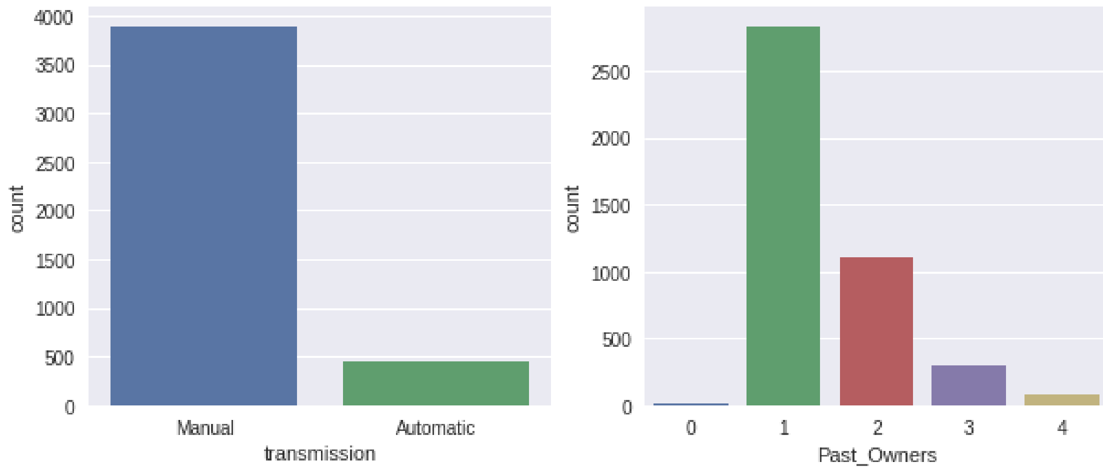
Univariate Analysis

[ ]:
```
df_main.columns
```

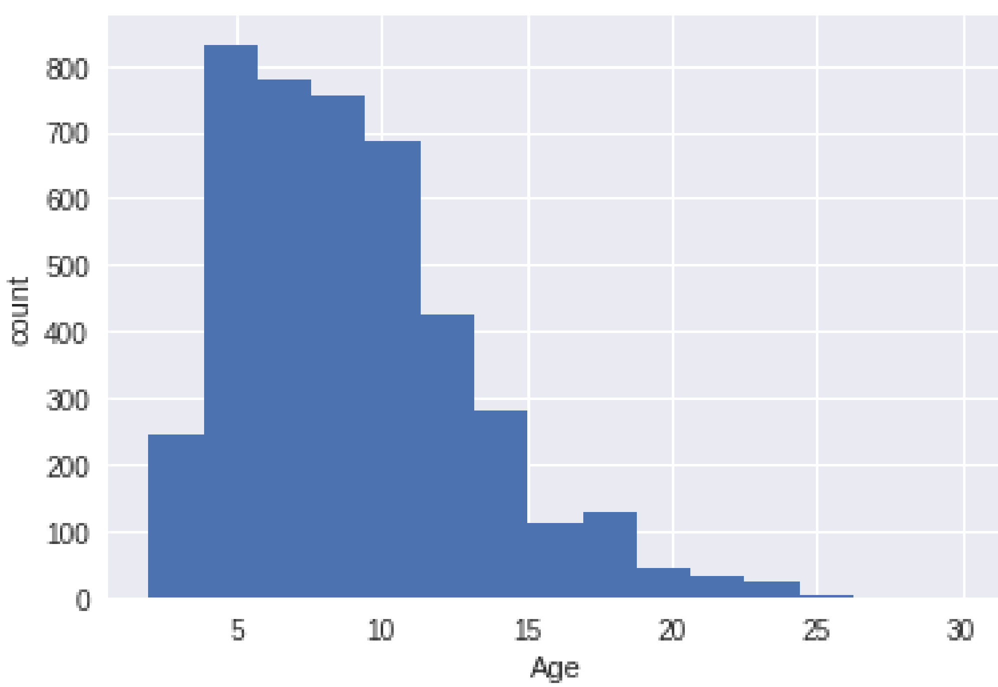
[ ]:
```
Index(['name', 'selling_price', 'km_driven', 'fuel', 'seller_type',
       'transmission', 'Past_Owners', 'Age'],
      dtype='object')
```

[ ]:
```
cat_cols = ['fuel','seller_type','transmission','Past_Owners']
i=0
while i < 4:
    fig = plt.figure(figsize=[10,4])
    #ax1 = fig.add_subplot(121)
    #ax2 = fig.add_subplot(122)

    #ax1.title.set_text(cat_cols[i])
    plt.subplot(1,2,1)
    sns.countplot(x=cat_cols[i], data=df_main)
    i += 1

    #ax2.title.set_text(cat_cols[i])
    plt.subplot(1,2,2)
    sns.countplot(x=cat_cols[i], data=df_main)
    i += 1

    plt.show()
```



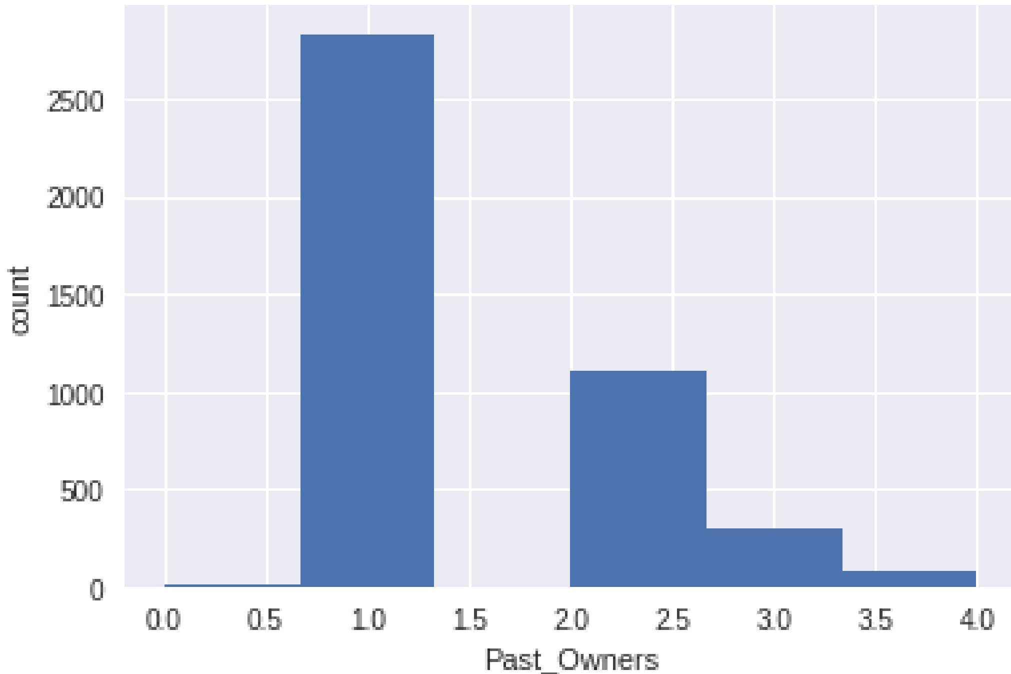


[ ]:
```
plt.hist(df_main['Age'],bins=15)
plt.xlabel('Age')
plt.ylabel('count')
plt.show()
```



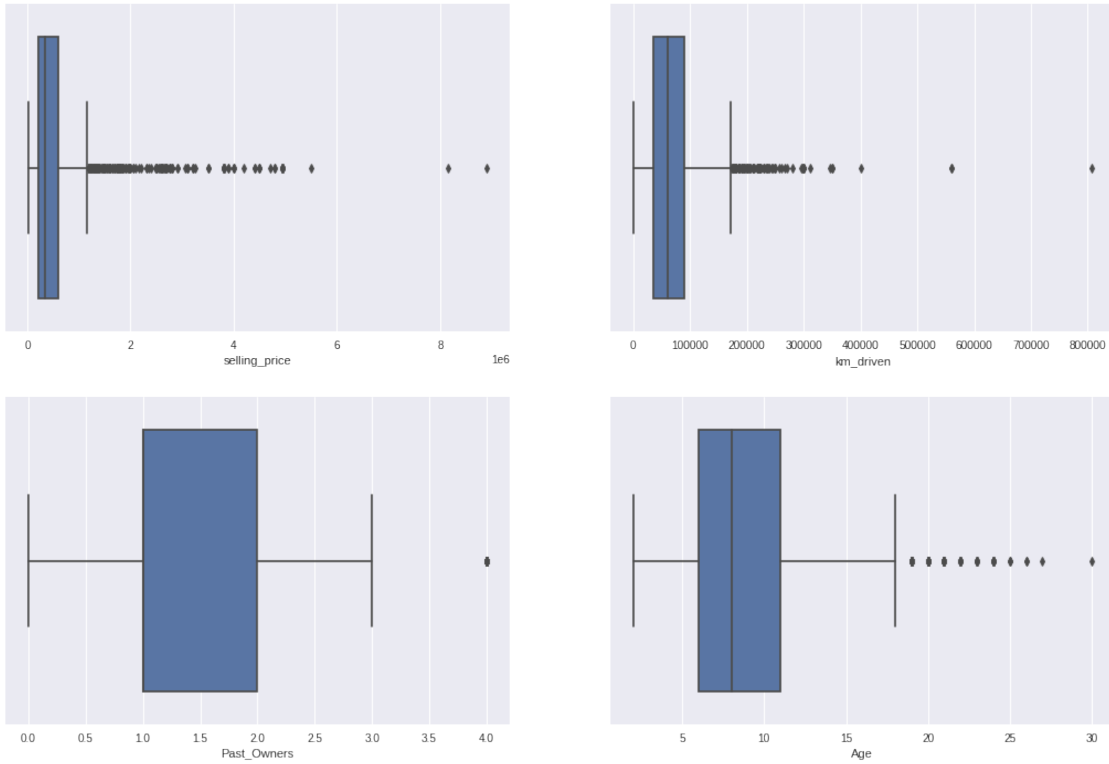
[ ]:
```
plt.hist(df_main['Past_Owners'],bins=6)
plt.xlabel('Past_Owners')
plt.ylabel('count')
```

```
plt.show()
```



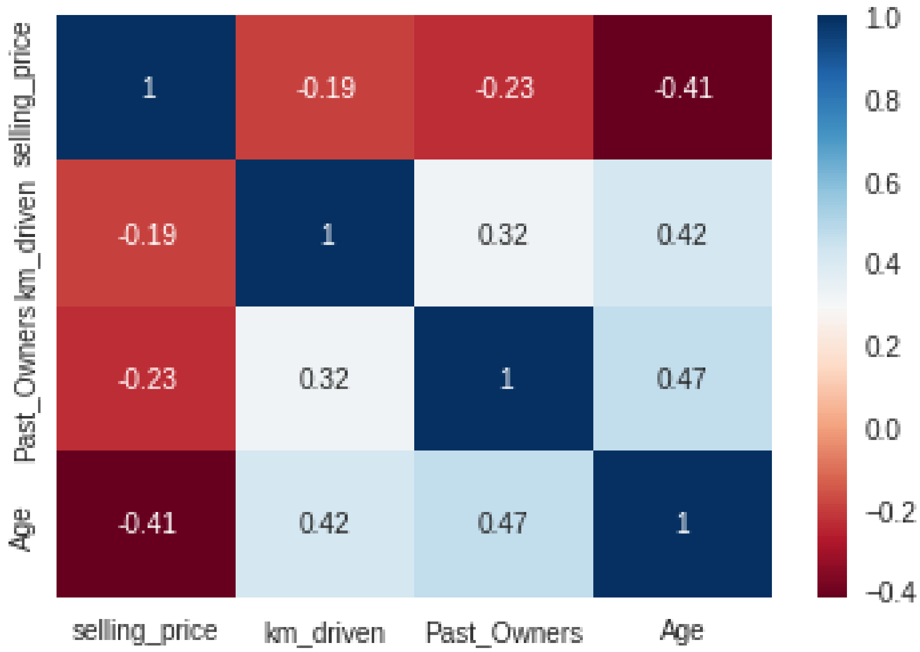
[ ]:
```
fig, ax= plt.subplots(nrows= 2, ncols = 2, figsize= (18,12))

sns.boxplot(df_main['selling_price'],data=df_main, ax=ax[0][0])
sns.boxplot(df_main['km_driven'], ax=ax[0][1])
sns.boxplot(df_main['Past_Owners'], ax=ax[1][0])
sns.boxplot(df_main['Age'], ax=ax[1][1])
```

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0f6f4a1210>



Bivariate/Multi-Variate Analysis

[ ]:
```
sns.heatmap(df_main.corr(), annot=True, cmap="RdBu")
plt.show()
```



[ ]:
```
df_main.corr()['selling_price']
```

[ ]:
```
selling_price    1.000000
km_driven       -0.192289
Past_Owners     -0.228091
Age             -0.413922
Name: selling_price, dtype: float64
```

0.0.4 Data Preparation

Normalization

[ ]:
```
# copy the data
nor =['km_driven', 'Past_Owners', 'Age']

# apply normalization techniques
for i in nor:
    df_main[i]= (df_main[i] - np.min(df_main[i])) / (np.max(df_main[i]) - np.
 ↪min(df_main[i]))
# view normalized data
df_main
```

[ ]:
```
                              name  selling_price  km_driven    fuel  \
0                     Maruti 800 AC          60000   0.086783  Petrol
1           Maruti Wagon R LXI Minor         135000   0.061988  Petrol
```

```
2                      Hyundai Verna 1.6 SX         600000    0.123976  Diesel
3                    Datsun RediGO T Option         250000    0.057028  Petrol
4                     Honda Amaze VX i-DTEC         450000    0.174807  Diesel
…                                         …              …         …    …
4335  Hyundai i20 Magna 1.4 CRDi (Diesel)         409999    0.099181  Diesel
4336           Hyundai i20 Magna 1.4 CRDi         409999    0.099181  Diesel
4337                  Maruti 800 AC BSIII         110000    0.102900  Petrol
4338     Hyundai Creta 1.6 CRDi SX Option         865000    0.111579  Diesel
4339                     Renault KWID RXT         225000    0.049590  Petrol

     seller_type transmission  Past_Owners       Age
0     Individual       Manual         0.25  0.464286
1     Individual       Manual         0.25  0.464286
2     Individual       Manual         0.25  0.285714
3     Individual       Manual         0.25  0.107143
4     Individual       Manual         0.50  0.214286
…            …            …          …         …
4335  Individual       Manual         0.50  0.214286
4336  Individual       Manual         0.50  0.214286
4337  Individual       Manual         0.50  0.392857
4338  Individual       Manual         0.25  0.142857
4339  Individual       Manual         0.25  0.142857

[4340 rows x 8 columns]
```

Creating Dummies for Categorical Features

[ ]:
```
df_main.drop(labels='name',axis= 1, inplace = True)
```

[ ]:
```
df_main.head()
```

[ ]:
```
   selling_price  km_driven    fuel seller_type transmission  Past_Owners  \
0          60000   0.086783  Petrol  Individual       Manual         0.25
1         135000   0.061988  Petrol  Individual       Manual         0.25
2         600000   0.123976  Diesel  Individual       Manual         0.25
3         250000   0.057028  Petrol  Individual       Manual         0.25
4         450000   0.174807  Diesel  Individual       Manual         0.50

        Age
0  0.464286
1  0.464286
2  0.285714
3  0.107143
4  0.214286
```

[ ]:
```
df_main['transmission'].value_counts()
```

[ ]:
```
Manual       3892
Automatic     448
Name: transmission, dtype: int64
```

[ ]:
```
df_copy = df_main.copy()
```

[ ]:
```
df_copy.head()
```

[ ]:
```
   selling_price  km_driven    fuel seller_type transmission  Past_Owners  \
0          60000   0.086783  Petrol  Individual       Manual         0.25
1         135000   0.061988  Petrol  Individual       Manual         0.25
2         600000   0.123976  Diesel  Individual       Manual         0.25
3         250000   0.057028  Petrol  Individual       Manual         0.25
4         450000   0.174807  Diesel  Individual       Manual         0.50

        Age
0  0.464286
1  0.464286
2  0.285714
3  0.107143
4  0.214286
```

[ ]:
```
df_main = pd.get_dummies(data = df_main)
```

[ ]:
```
df_main.head()
```

[ ]:
```
   selling_price  km_driven  Past_Owners       Age  fuel_CNG  fuel_Diesel  \
0          60000   0.086783         0.25  0.464286         0            0
1         135000   0.061988         0.25  0.464286         0            0
2         600000   0.123976         0.25  0.285714         0            1
3         250000   0.057028         0.25  0.107143         0            0
4         450000   0.174807         0.50  0.214286         0            1

   fuel_Electric  fuel_LPG  fuel_Petrol  seller_type_Dealer  \
0              0         0            1                   0
1              0         0            1                   0
2              0         0            0                   0
3              0         0            1                   0
4              0         0            0                   0

   seller_type_Individual  seller_type_Trustmark Dealer  \
0                       1                             0
1                       1                             0
2                       1                             0
3                       1                             0
4                       1                             0
```

```
   transmission_Automatic  transmission_Manual
0                       0                    1
1                       0                    1
2                       0                    1
3                       0                    1
4                       0                    1
```

[ ]:
```
df_main.columns
```

[ ]:
```
Index(['selling_price', 'km_driven', 'Past_Owners', 'Age', 'fuel_CNG',
       'fuel_Diesel', 'fuel_Electric', 'fuel_LPG', 'fuel_Petrol',
       'seller_type_Dealer', 'seller_type_Individual',
       'seller_type_Trustmark Dealer', 'transmission_Automatic',
       'transmission_Manual'],
      dtype='object')
```

0.0.5 Train-Test Split

[ ]:
```
# Separating target variable and its features
y = df_main['selling_price']
X = df_main.drop('selling_price',axis=1).astype(float)
```

0.0.6 Hypothesis Testing

[ ]:
```
from scipy import stats
from sklearn.linear_model import LinearRegression
from statsmodels.compat import lzip
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
```

[ ]:
```
model = sm.OLS(y, X)
model = model.fit()
model.params
```

[ ]:
```
km_driven                  -787004.500260
Past_Owners                 -79837.645940
Age                        -997497.174563
fuel_CNG                    266446.011766
fuel_Diesel                 551941.962516
fuel_Electric              -356821.919625
fuel_LPG                    312229.647591
fuel_Petrol                 262006.478737
```

```
seller_type_Dealer              313621.497096
seller_type_Individual          242848.642235
seller_type_Trustmark Dealer    479332.041655
transmission_Automatic          952360.782195
transmission_Manual              83441.398791
dtype: float64
```

[ ]:
```
model.summary()
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.459
Model:                            OLS   Adj. R-squared:                  0.457
Method:                 Least Squares   F-statistic:                     366.6
Date:                Fri, 13 May 2022   Prob (F-statistic):               0.00
Time:                        00:01:42   Log-Likelihood:                -62411.
No. Observations:                4340   AIC:                         1.248e+05
Df Residuals:                    4329   BIC:                         1.249e+05
Df Model:                          10
Covariance Type:            nonrobust
================================================================================
================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
----------------
km_driven                    -7.87e+05   1.36e+05     -5.804      0.000
-1.05e+06   -5.21e+05
Past_Owners                 -7.984e+04   4.25e+04     -1.877      0.061
-1.63e+05    3571.546
Age                         -9.975e+05   5.35e+04    -18.644      0.000
-1.1e+06   -8.93e+05
fuel_CNG                     2.664e+05   9.05e+04      2.943      0.003
8.9e+04    4.44e+05
fuel_Diesel                  5.519e+05   7.18e+04      7.691      0.000
4.11e+05    6.93e+05
fuel_Electric               -3.568e+05   3.59e+05     -0.995      0.320
-1.06e+06    3.46e+05
fuel_LPG                     3.122e+05   1.02e+05      3.050      0.002
1.12e+05    5.13e+05
fuel_Petrol                   2.62e+05   7.17e+04      3.655      0.000
1.21e+05    4.03e+05
seller_type_Dealer           3.136e+05   3.18e+04      9.852      0.000
2.51e+05    3.76e+05
seller_type_Individual       2.428e+05     3.2e+04      7.598      0.000
```

```
1.8e+05     3.06e+05
seller_type_Trustmark Dealer  4.793e+05   4.37e+04     10.975      0.000
3.94e+05     5.65e+05
transmission_Automatic        9.524e+05   4.57e+04     20.855      0.000
8.63e+05     1.04e+06
transmission_Manual           8.344e+04   4.48e+04      1.862      0.063
-4436.426    1.71e+05
==============================================================================
Omnibus:                     4368.888   Durbin-Watson:                   1.938
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           502954.806
Skew:                           4.659   Prob(JB):                         0.00
Kurtosis:                      54.908   Cond. No.                     1.02e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 9.12e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

Removing fuel_Electric as it is having high p-value

[ ]:
```
X_filtered= X.drop(['fuel_Electric'],axis=1)
```

[ ]:
```
X_filtered.columns
```

[ ]:
```
Index(['km_driven', 'Past_Owners', 'Age', 'fuel_CNG', 'fuel_Diesel',
       'fuel_LPG', 'fuel_Petrol', 'seller_type_Dealer',
       'seller_type_Individual', 'seller_type_Trustmark Dealer',
       'transmission_Automatic', 'transmission_Manual'],
      dtype='object')
```

0.0.7 Spliting the dataset into train-test split

[ ]:
```
from sklearn.model_selection import train_test_split
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_filtered, y, test_size=0.
 ↪30, random_state=20)
print("x train: ",X_train.shape)
print("x test: ",X_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (3038, 12)
x test:  (1302, 12)
```

```
y train:  (3038,)
y test:  (1302,)
```

0.0.8 Model Creation/Evaluation

1. Ridge Regression:

- Performs L2 regularization, i.e. adds penalty equivalent to square of the magnitude of coefficients
- Minimization objective = LS Obj + * (sum of square of coefficients)

2. Lasso Regression:

- Performs L1 regularization, i.e. adds penalty equivalent to absolute value of the magnitude of coefficients
- Minimization objective = LS Obj + * (sum of absolute value of coefficients)

  Note that here 'LS Obj' refers to 'least squares objective', i.e. the linear regression objective without regularization.

0.0.9 Ridge Regression

[ ]:
```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=0.1, normalize=True)
ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)
print('coeffients:',ridge.coef_)
print('-----------------------------------------------------')
print('intercept:',ridge.intercept_)
print('-----------------------------------------------------')
print('R2-Score:',ridge.score(X_test, y_test))
```

```
coeffients: [-710537.98902067 -131713.24392353 -882820.63943373 -116191.77746643
  135273.23589435  -73029.50117084 -129607.50238706   23695.55922562
  -48061.51733131  205238.95037296  405678.95764597 -405678.95764597]
-----------------------------------------------------
intercept: 1169299.0970165194
-----------------------------------------------------
R2-Score: 0.43563739266011536
```

0.0.10 Lasso Regression

[ ]:
```
from locale import normalize
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1, normalize=True)
```

```
lasso.fit(X_train, y_train)
lasso_pred = lasso.predict(X_test)
print('coeffients:',lasso.coef_)
print('------------------------------------------------------')
print('intercept:',lasso.intercept_)
print('------------------------------------------------------')
print('R2-Score:',lasso.score(X_test,y_test))
```

```
coeffients: [-7.21675432e+05 -1.04437370e+05 -9.71948116e+05 -0.00000000e+00
  2.69427124e+05  6.21981799e+04 -5.49140097e+03  3.03118428e+04
 -3.61965728e+04  2.26638432e+05  8.47064352e+05 -3.45795373e-08]
------------------------------------------------------
intercept: 633163.8669866546
------------------------------------------------------
R2-Score: 0.43991662887097205
```

0.1 Regularized Regression with Cross Validation

[ ]:
```
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
```

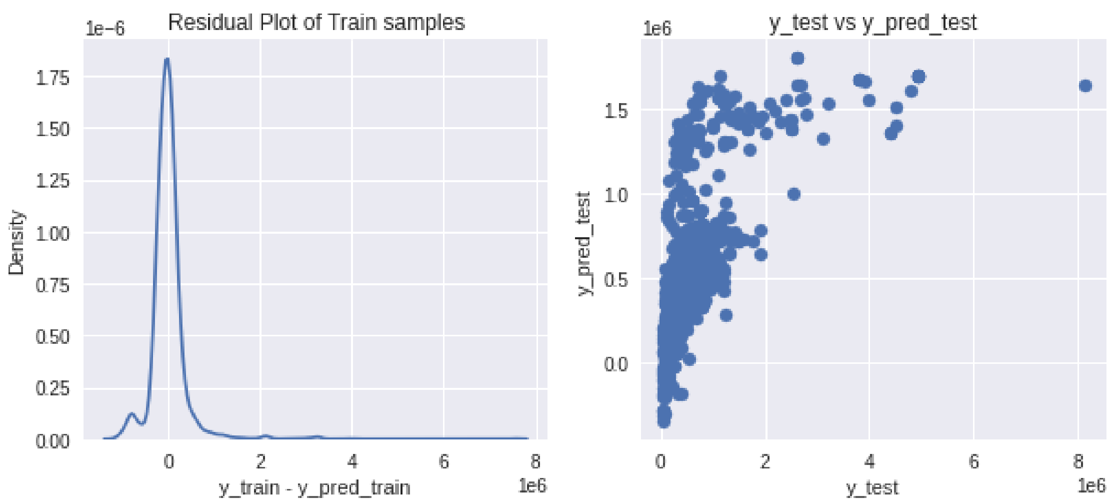
[ ]:
```
CV = []
R2_train = []
R2_test = []

def car_pred_model(model,model_name):
    # Training model
    model.fit(X_train,y_train)

    # R2 score of train set
    y_pred_train = model.predict(X_train)
    R2_train_model = r2_score(y_train,y_pred_train)
    R2_train.append(round(R2_train_model,2))
    mse_train = mean_squared_error(y_train,y_pred_train)

    # R2 score of test set
    y_pred_test = model.predict(X_test)
    R2_test_model = r2_score(y_test,y_pred_test)
    R2_test.append(round(R2_test_model,2))
    mse_test=mean_squared_error(y_test,y_pred_test)

    # R2 mean of train set using Cross validation
    cross_val = cross_val_score(model ,X_train ,y_train ,cv=5)
    cv_mean = cross_val.mean()
    CV.append(round(cv_mean,2))
```

```
    # Printing results
    print("Train R2-score :",round(R2_train_model,2))
    print("Test R2-score :",round(R2_test_model,2))
    print("Train mse-score :",round(mse_train,2))
    print("Test mse-score :",round(mse_test,2))
    print("Train CV scores :",cross_val)
    print("Train CV mean :",round(cv_mean,2))

    # Plotting Graphs
    # Residual Plot of train data
    fig, ax = plt.subplots(1,2,figsize = (10,4))
    ax[0].set_title('Residual Plot of Train samples')
    sns.distplot((y_train-y_pred_train),hist = False,ax = ax[0])
    ax[0].set_xlabel('y_train - y_pred_train')

    # Y_test vs Y_train scatter plot
    ax[1].set_title('y_test vs y_pred_test')
    ax[1].scatter(x = y_test, y = y_pred_test)
    ax[1].set_xlabel('y_test')
    ax[1].set_ylabel('y_pred_test')

    plt.show()
```

lINEAR REGRESSION TO COMPARE RESULTS

[ ]:
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
car_pred_model(lr,"Linear_regressor.pkl")
print('coeffients:',lr.coef_)
print('intercept:',lr.intercept_)
```

```
Train R2-score : 0.47
Test R2-score : 0.44
Train mse-score : 164302572650.96
Test mse-score : 221578773391.43
Train CV scores : [0.49321295 0.36342675 0.50107349 0.48438796 0.47018241]
Train CV mean : 0.46
```

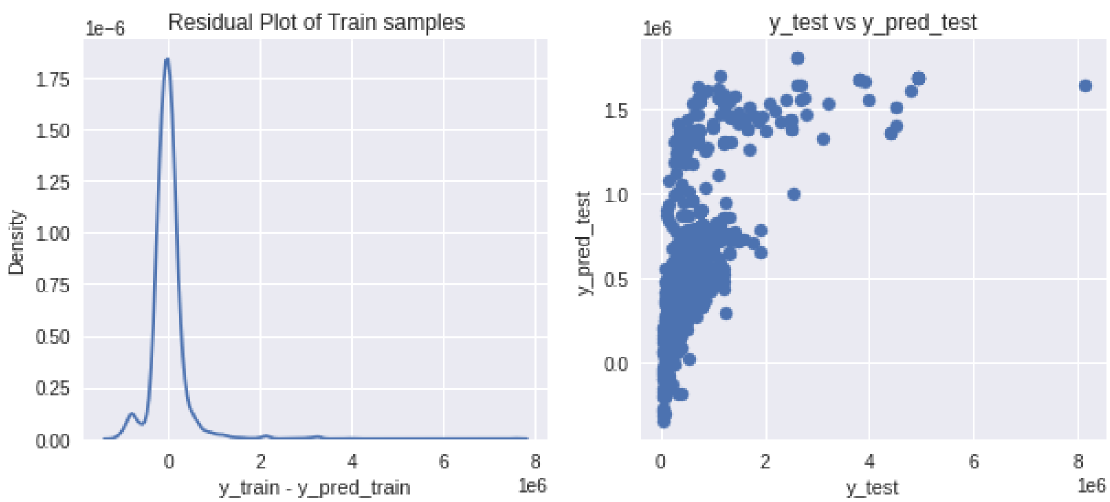



```
coeffients: [-721770.98122492 -104451.32343538 -971950.63630291  -81581.6457903
  187886.01299802  -19257.70075604  -87046.66645169  -43280.26266189
 -109792.90869415  153073.17135605  423538.29121923 -423538.29121923]
intercept: 1211857.0473111868
```

0.1.1 Ridge Regression

[ ]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV

# Creating Ridge model object
rg = Ridge()
# range of alpha
alpha = np.logspace(-3,3,num=14)

# Creating RandomizedSearchCV to find the best estimator of hyperparameter
rg_rs = RandomizedSearchCV(estimator = rg, param_distributions =␣
 ↪dict(alpha=alpha))
car_pred_model(rg_rs,"ridge.pkl")
```

```
Train R2-score : 0.47
Test R2-score : 0.44
Train mse-score : 164308725911.3
Test mse-score : 221678113743.63
Train CV scores : [0.49188358 0.36413493 0.50123305 0.48470748 0.47001539]
Train CV mean : 0.46
```

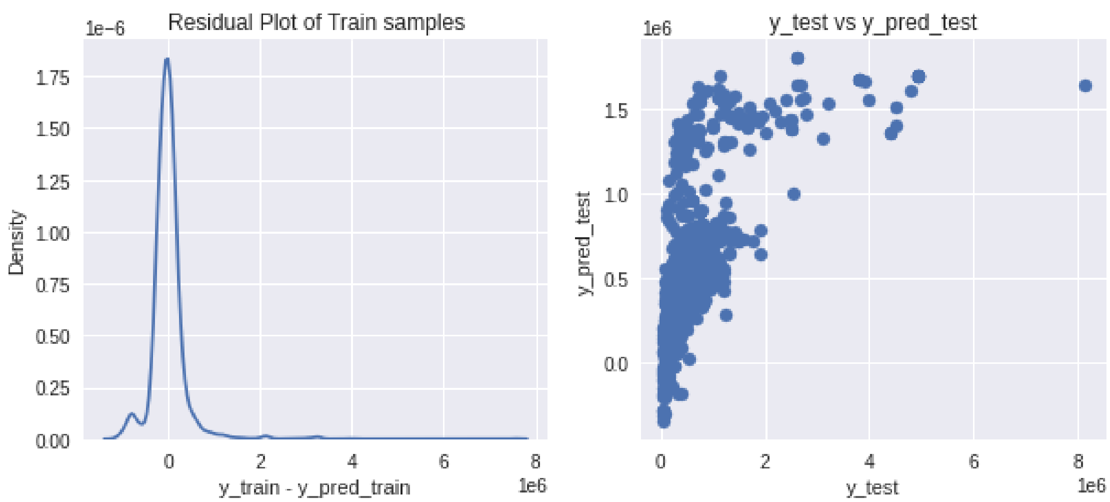


0.1.2 Lasso Regression

[ ]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import RandomizedSearchCV

ls = Lasso()
alpha = np.logspace(-3,3,num=14) # range for alpha
ls_rs = RandomizedSearchCV(estimator = ls, param_distributions =␣
 ↪dict(alpha=alpha))
```

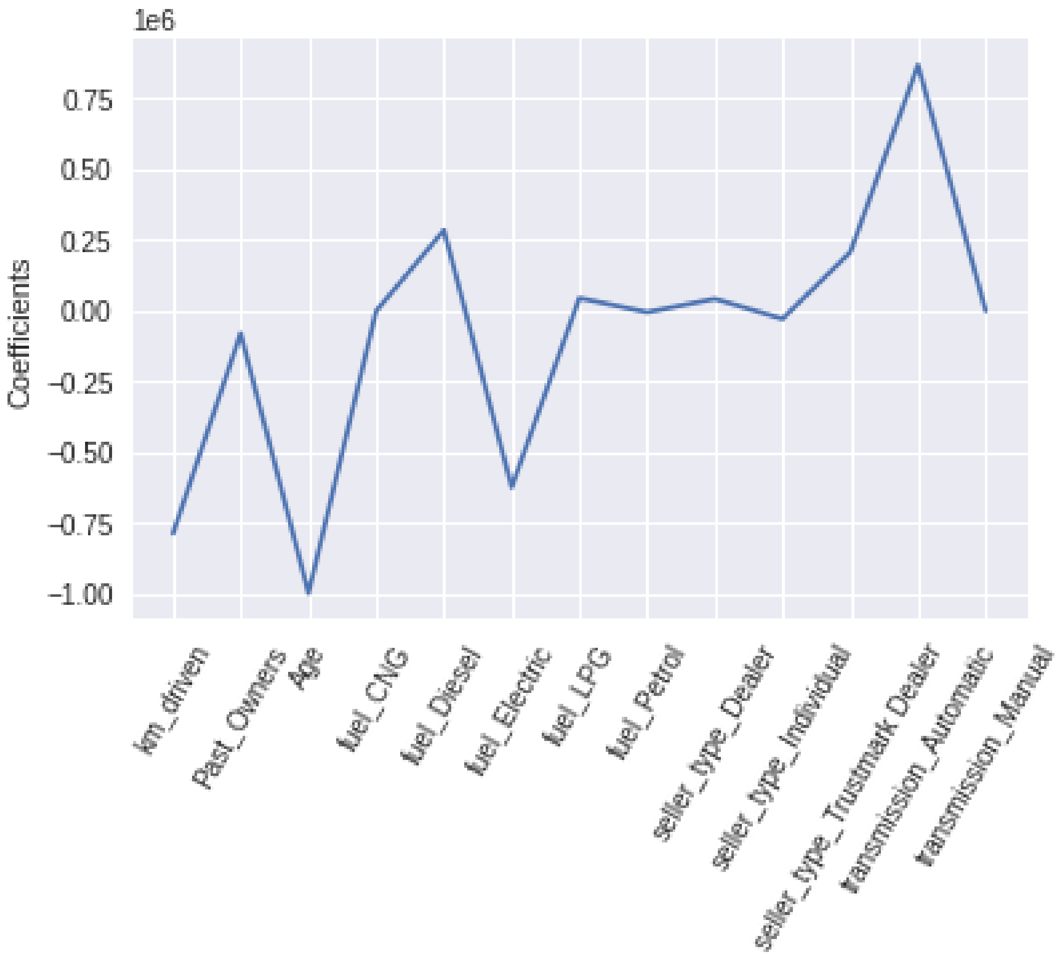
[ ]:
```
car_pred_model(ls_rs,"lasso.pkl")
```

```
Train R2-score : 0.47
Test R2-score : 0.44
Train mse-score : 164302719481.07
Test mse-score : 221503973444.16
Train CV scores : [0.49216978 0.36499275 0.5008215  0.4847804  0.46988328]
Train CV mean : 0.46
```



0.1.3 Lasso for feature selection

[ ]:
```
from matplotlib import axis
cols = df_main.drop('selling_price', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
_ = plt.plot(range(len(cols)), lasso_coef)
_ = plt.xticks(range(len(cols)), cols, rotation = 60)
_ = plt.ylabel('Coefficients')
plt.show()
```



0.1.4 Naive Bayes

[ ]:
```
#initiating predictors and target variable
y = df_copy['transmission'].replace(['Automatic','Manual'],[0,1])
X = df_main.drop(['transmission_Automatic','transmission_Manual'],axis=1)
```

Train - Test split

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1)
print("x train: ",X_train.shape)
print("x test: ",X_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (3038, 12)
```

```
x test:  (1302, 12)
y train:  (3038,)
y test:  (1302,)
```

0.1.5 Function for Confusion Matrix plot

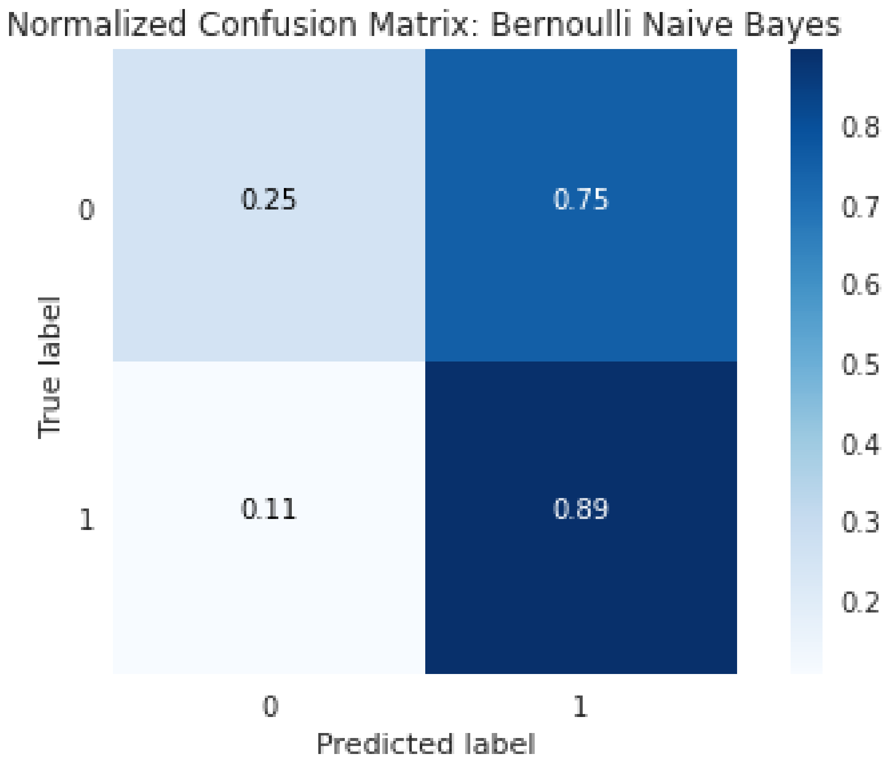
[ ]:
```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    import itertools
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

[ ]:
```python
#importing libraries
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix

#Navie bayes using Gaussian Naive Bayes
NB_model = GaussianNB().fit(X_train,y_train)

NB_pred = NB_model.predict(X_test)
```

0.1.6 Classifing using Bernoulli Naive Bayes

[ ]:
```
#Navie bayes using Bernoulli Naive Bayes
NBB_model = BernoulliNB().fit(X_train,y_train)
NBB_pred = NBB_model.predict(X_test)


from sklearn.metrics import accuracy_score


print('Accuracy Score is {:.5}'.format(accuracy_score(y_test,NBB_pred)*100))
print(pd.DataFrame(confusion_matrix(y_test,NBB_pred)))

sns.set_style('white')
class_names = ['0','1']
plot_confusion_matrix(confusion_matrix(y_test,NBB_pred),
                      classes= class_names, normalize = True,
                      title='Normalized Confusion Matrix: Bernoulli Naive␣
 ↪Bayes')
```

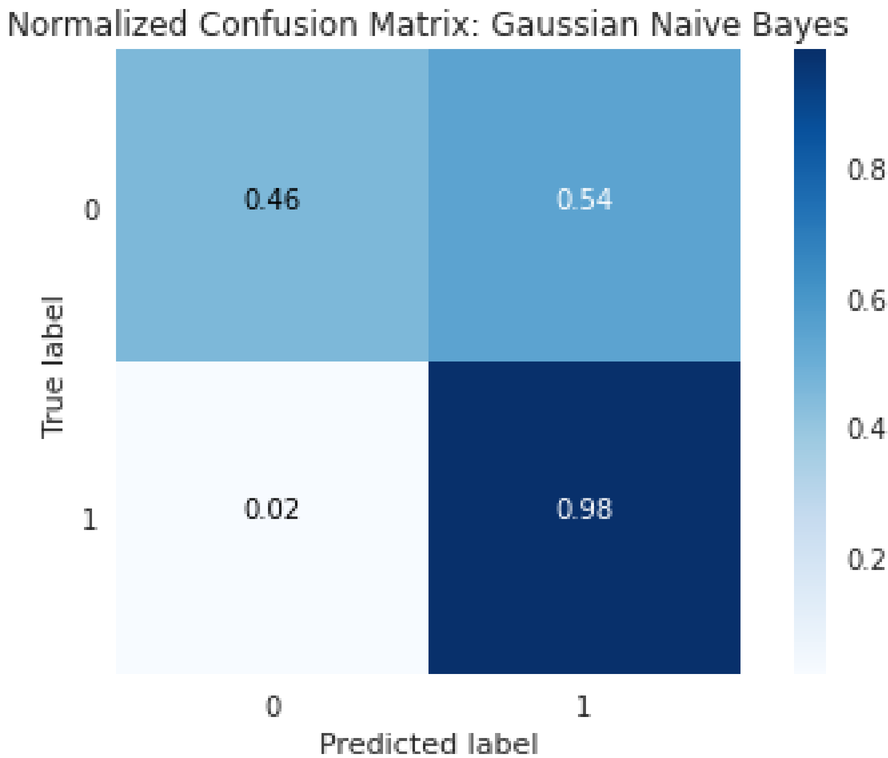
```
Accuracy Score is 83.487
     0     1
0   30    90
1  125  1057
[[0.25       0.75      ]
 [0.10575296 0.89424704]]
```



0.1.7 Classifying using Gaussian Naive Bayes

[ ]:
```
from sklearn.metrics import accuracy_score


print('Accuracy Score is {:.5}'.format(accuracy_score(y_test,NB_pred)*100))
print(pd.DataFrame(confusion_matrix(y_test,NB_pred)))

sns.set_style('white')
class_names = ['0','1']
plot_confusion_matrix(confusion_matrix(y_test,NB_pred),
                      classes= class_names, normalize = True,
                      title='Normalized Confusion Matrix: Gaussian Naive Bayes')
```

```
Accuracy Score is 93.088
    0     1
0  55    65
1  25  1157
[[0.45833333 0.54166667]
 [0.02115059 0.97884941]]
```



[ ]:
```
#importing metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
```

[ ]:
```
print(classification_report(y_test,NB_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.46      0.55       120
           1       0.95      0.98      0.96      1182

    accuracy                           0.93      1302
   macro avg       0.82      0.72      0.76      1302
weighted avg       0.92      0.93      0.92      1302
```

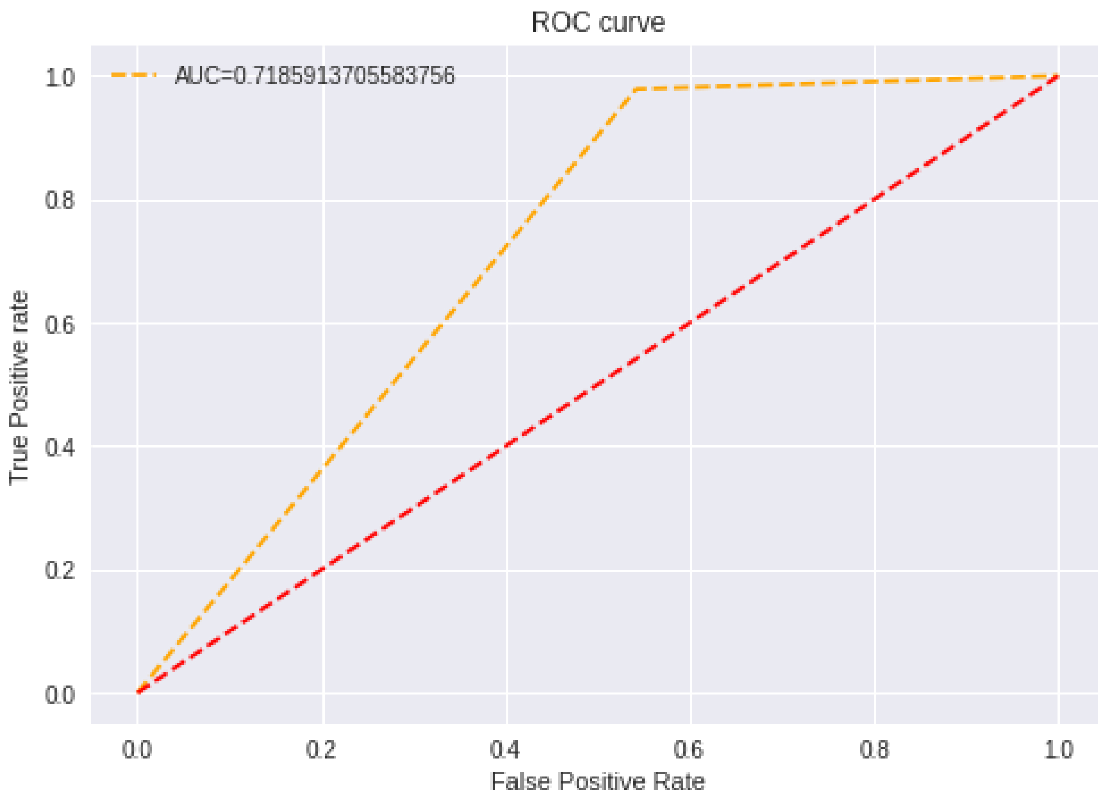
[ ]:
```
print("Precision score : {} %".format(precision_score(y_test,NB_pred)*100))
print("Recall score : {} %".format(recall_score(y_test,NB_pred)*100))
print("AUC_score is {}%".format(roc_auc_score(y_test,NB_pred)*100))
```

```
print("Accuracy is {}%".format(accuracy_score(y_test,NB_pred)*100))
```

```
Precision score : 94.68085106382979 %
Recall score : 97.88494077834179 %
AUC_score is 71.85913705583756%
Accuracy is 93.08755760368663%
```

[ ]:

```
# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')
from sklearn import metrics




# model = LogisticRegression(C=0.8, random_state=0,solver='lbfgs')
# model.fit(X_train, y_train)
# Y_predict = model.predict(X_test)

#define metrics
#y_pred_proba = log_regression.predict_proba(X_test)[::,1]

fpr, tpr, _ = metrics.roc_curve(y_test,  NB_pred)

auc =  metrics.roc_auc_score(y_test, NB_pred)

# plot roc curves
plt.plot(fpr, tpr, linestyle='--',color='orange', label="AUC="+str(auc))
plt.plot([0, 1], [0, 1],'r--')

# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show()
```



The End

1 Saving it to a PDF file

[ ]:
```
!wget -nc https://raw.githubusercontent.com/suewsu875/colab-pdf/master/
 ↪colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('MATH646-Project.ipynb',notebookpath="/content/drive/MyDrive/Colab␣
 ↪Notebooks/")
```

```
--2022-05-13 04:14:15--  https://raw.githubusercontent.com/suewsu875/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
```

```
HTTP request sent, awaiting response… 200 OK
Length: 1848 (1.8K) [text/plain]
Saving to: ‘colab_pdf.py’

colab_pdf.py        100%[===================>]   1.80K  --.-KB/s    in 0s

2022-05-13 04:14:15 (31.8 MB/s) - ‘colab_pdf.py’ saved [1848/1848]

Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```

[ ]:

```
```