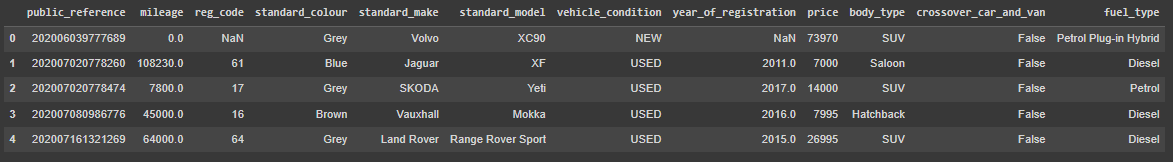
Title page

# Data Understanding and exploration

## Meaning and Type of Features; Analysis of Univariate

## Distributions

import pandas as pd  
import warnings  
warnings.filterwarnings('ignore')  
# Load data  
data = pd.read\_csv("AutoTrader Dataset.csv")



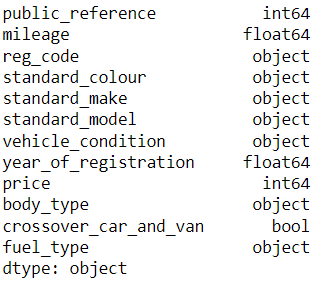
data.shape



The above given code snippet loads the dataset csv as a Pandas dataframe and assign it to a variable data. Through shape function, it is learnt that the dataset contains 12 features and 402,005 rows/records.

The below snippet shows datatypes of the features of the dataset. Public reference and price are int type features while mileage and year of registration are a float type. Whereas; rest of the features are categorical in nature and are object type except crossover car and van which is a Boolean type.

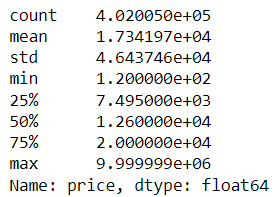
# Check data types of columns  
print(data.dtypes)



Univariate distribution of price,Mileage and year of registration. is explored by the describe() function in the below code.

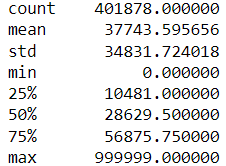
# Analyze the distribution of selling price

print(data["price"].describe())



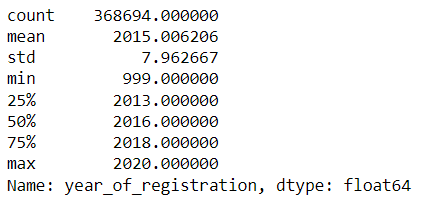
# Analyze the distribution of mileage

print(data["mileage"].describe())



# Analyze the distribution of year\_of\_registration

print(data["year\_of\_registration"].describe())



## Analysis of Predictive Power of Features

The Predictive power is calculated by statistical Correlation dependency between features. The correlation is found by using corr and a heatmap is also plotted for better visualization.

Predictive power of mileage with respect to label ‘price’ has been calculated in the below snippet. Mileage has negative correlation with price of a car. In other words, if mileage goes up, price goes down but it is a weak correlation.

corr\_matrix = data.corr()

print(corr\_matrix["mileage"]["price"])



Similarly, year of registration has a positive correlation with price of a car. Price increases with the earlier registration year.

corr\_matrix = data.corr()

print(corr\_matrix["year\_of\_registration"]["price"])



Furthermore, predictive power of four features has been calculated and shown in a heat map plot below.

import seaborn as sns

import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (10,5)

corr = data.corr()

sns.heatmap(corr,

xticklabels=corr.columns.values,

yticklabels=corr.columns.values,

annot=True)

plt.show()



## Data Processing for Data Exploration and Visualisation

Data exploration involves plotting different features in frequency distribution and a combination of two features in comparison.

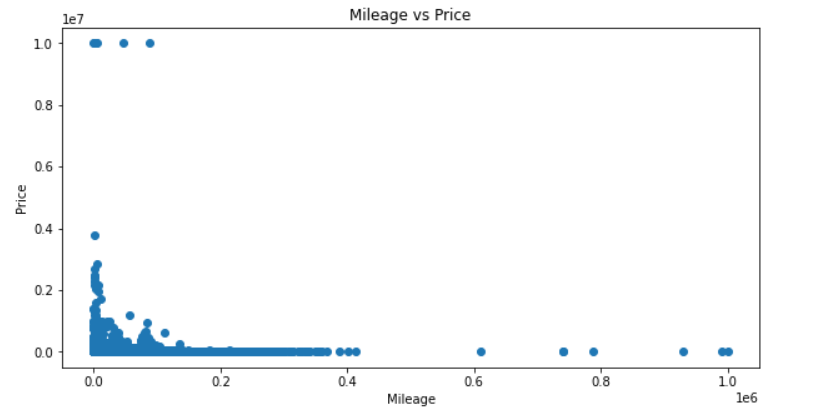
plt.scatter(data["mileage"], data["price"])

plt.xlabel("Mileage")

plt.ylabel("Price")

plt.title("Mileage vs Price")

plt.show()



The above illustration of mileage vs price, distribution of mileage and price shows how their values change in the dataset.

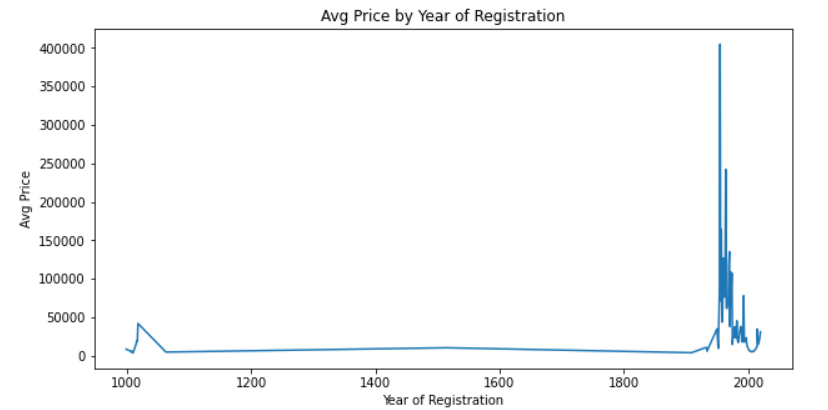
data.groupby("year\_of\_registration")["price"].mean().plot()

plt.xlabel("Year of Registration")

plt.ylabel("Avg Price")

plt.title("Avg Price by Year of Registration")

plt.show()



Below is the frequency distribution of body type feature of car dataset.

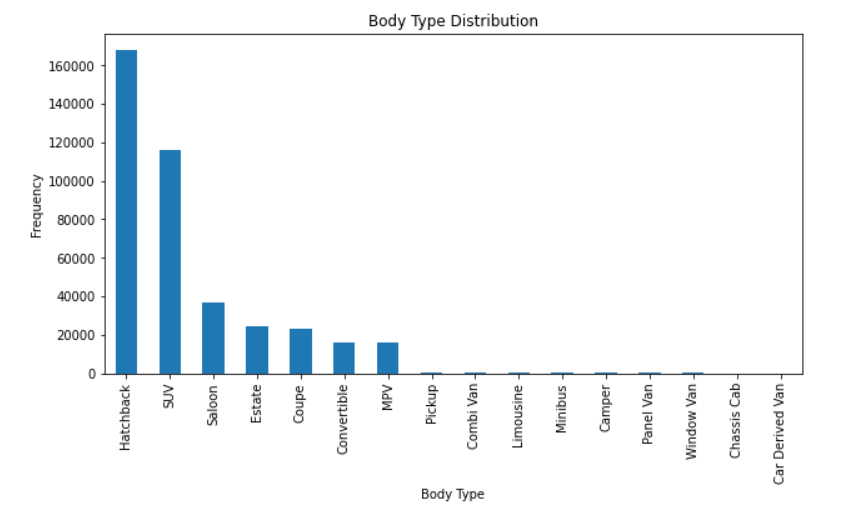
data["body\_type"].value\_counts().plot(kind="bar")

plt.xlabel("Body Type")

plt.ylabel("Frequency")

plt.title("Body Type Distribution")

plt.show()



From the output, it can be inferred that most of the recorded cars belong to Hatchback category, followed by SUV, Saloon and Coupe. Moreover, Convertibale and MPV have same frequency and the rest of categories have very less count relatively.

import seaborn as sns

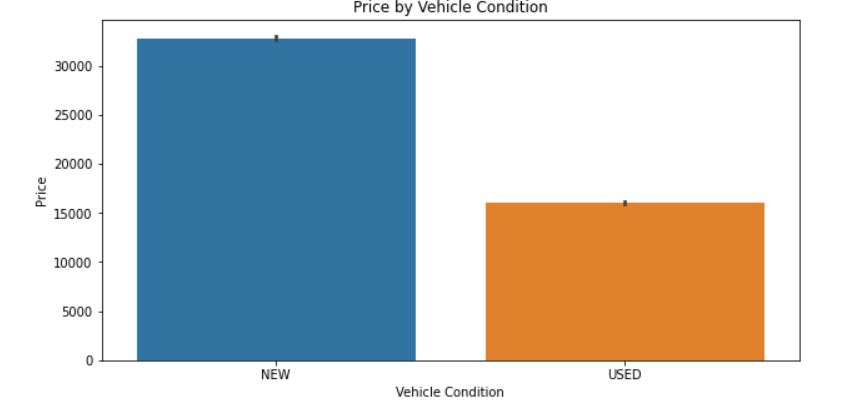
sns.barplot(x="vehicle\_condition", y="price", data=data)

plt.xlabel("Vehicle Condition")

plt.ylabel("Price")

plt.title("Price by Vehicle Condition")

plt.show()



Afore given code and plot demonstrates a group bar plot of price and condition of vehicle; either it is new or used. It can be seen that New cars in majority and their price is definitely higher than used ones comparatively.

# 2. Data Processing for Data Exploration and Visualisation

## 2.1. Dealing with Missing Values, Outliers, and Noise

Missing values have been treated by replacing them with the mean of all available values of that feature. Missing values of Mileage and Registration year were imputed by their mean values. Whereas features including reg\_color, standard color, standard make and model, vehicle condition, body type and fuel type are categorical values therefore, they have been treated by replacing their missing values by the mode or most frequent value.

It follows label encoding of categorical features because string values can be used in model training because all the features get converted to float while training and a string cannot be casted to float.

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

# create an imputer object with strategy as 'mean'

imputer = SimpleImputer(strategy='mean')

# fit and transform the imputer on numerical variables

data[['mileage', 'year\_of\_registration']] = imputer.fit\_transform(data[['mileage', 'year\_of\_registration']])

# create an imputer object with strategy as 'most\_frequent'

imputer = SimpleImputer(strategy='most\_frequent')

# fit and transform the imputer on categorical variables

data[['reg\_code', 'standard\_colour', 'standard\_make', 'standard\_model','vehicle\_condition','body\_type','fuel\_type']] = imputer.fit\_transform(data[['reg\_code', 'standard\_colour', 'standard\_make', 'standard\_model','vehicle\_condition','body\_type','fuel\_type']])

# Create an object of LabelEncoder

le = LabelEncoder()

# fit and transform the label encoder on categorical variables

data[['reg\_code', 'standard\_colour', 'standard\_make', 'standard\_model','vehicle\_condition','body\_type','fuel\_type']] = data[['reg\_code', 'standard\_colour', 'standard\_make', 'standard\_model','vehicle\_condition','body\_type','fuel\_type']]

Treating outliers by setting upper and lower limits to values of a feature and the values exceeding these limits have been clipped.

The final output shows that there have been no missing values found in the dataset after data processing.

# set the lower and upper limits for the outliers

lower\_limit = data["price"].quantile(0.05)

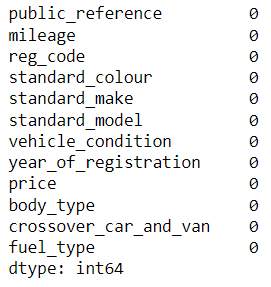
upper\_limit = data["price"].quantile(0.95)

# clip the values outside the range

data["price"] = data["price"].clip(lower\_limit, upper\_limit)

# Identify missing values

print(data.isnull().sum())



## 2.2. Feature Engineering, Data Transformations, Feature Selection

In the below code snippet, dataset has been labelencoded while the features of mileage and price were tried by standard scaler. The following output shows the dataframe after the engineering and transformation.

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

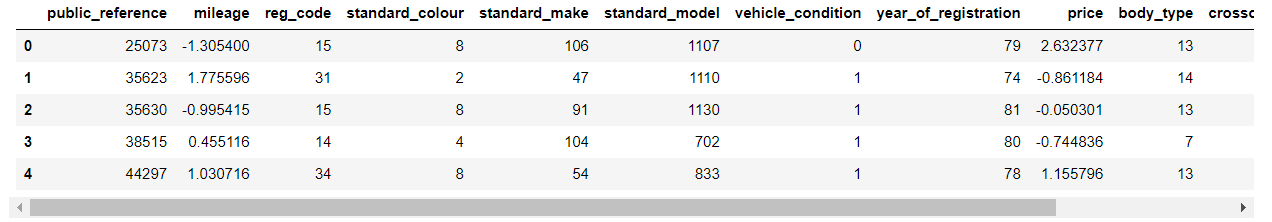
data = data.apply(LabelEncoder().fit\_transform)

data.head()

scaler = StandardScaler()

data[["mileage", "price"]] = scaler.fit\_transform(data[["mileage", "price"]])

data.head()



All the features except public reference were selected for the training. In the below code, X variable is assigned the same dataframe dropping public reference and price.

Another variable Y contains only the price feature of the dataset.

These variable were given as parameters to train test split function as input and output/label.

# Split data into training and testing sets

X = data.drop(["price",'public\_reference'] , axis=1)

y = data["price"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 3. Model Building

## 3.1. Algorithm Selection, Model Instantiation and Configuration

Lasso Model has been selected as a first algorithm for training and regression prediction of price of car.

# Choose a suitable algorithm

from sklearn.linear\_model import Lasso

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.model\_selection import GridSearchCV

# Fit and tune the model

lasso = Lasso(random\_state=42)

lasso.fit(X\_train, y\_train)

# predicting on test data

y\_pred = lasso.predict(X\_test)

# calculating mean absolute error and mean squared error

mae = mean\_absolute\_error(y\_test, y\_pred)

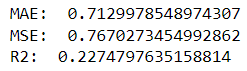
mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("MAE: ", mae)

print("MSE: ", mse)

print("R2: ", r2)



According to above output, Lasso predicted car price with Mean Absolute Average of 0.7129 and with Mean Squared Error and R2 of 0.7670 and 0.2274 respectively.

from sklearn.linear\_model import LinearRegression

# Select algorithm

model = LinearRegression()

# Train model

model.fit(X\_train, y\_train)

# predicting on test data

y\_pred = model.predict(X\_test)

# calculating mean absolute error and mean squared error

mae = mean\_absolute\_error(y\_test, y\_pred)

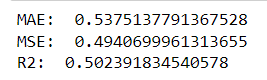
mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("MAE: ", mae)

print("MSE: ", mse)

print("R2: ", r2)



The second model is linear regression model. It predicted price with a good Mean Average Error of 0.5375, Mean Squared Error of 0.4940 and R2 score of 0.5023. Linear regression is the comparatively better model than Lasso algorithm.

## 3.2. Grid Search, and Model Ranking and Selection

GridSearch CV has been used to find optimal hyper parameter values and the linear regression model was tuned again as best estimator on the optimal parameters given by GridSearchCV. It has been found that linear regression performs with its maximum performance on default parameters.

param\_grid = {'normalize':[True,False], 'fit\_intercept':[True,False]}

grid\_search = GridSearchCV(model, param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Select the best model

best\_model = grid\_search.best\_estimator\_

# predicting on test data

y\_pred = best\_model.predict(X\_test)

# calculating mean absolute error and mean squared error

mae = mean\_absolute\_error(y\_test, y\_pred)

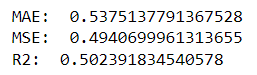
mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("MAE: ", mae)

print("MSE: ", mse)

print("R2: ", r2)



# 4. Model Evaluation and Analysis

## 4.1. Evaluation/Analysis

Predictions of tuned and optimized Linear Regression model was evaluated with respect to true values (actual price labels).

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(best\_model, X\_test, y\_test, cv=5)

# Analyze true vs predicted plot

y\_pred = best\_model.predict(X\_test)

plt.scatter(y\_test, y\_pred, c='b', label='True values')

plt.scatter(y\_pred,y\_test, c='r', label='Predicted values')

plt.xlabel("True Prices")

plt.ylabel("Predicted prices")

plt.title("True Prices vs Predicted Prices")

plt.legend()

plt.show()

The plot below shows true prices and predicted prices.



## 4.2. Feature Importance

Feature importance shows to what degree each feature contributes to the predictions made by the model.

# Gain and discuss insights based on feature importance

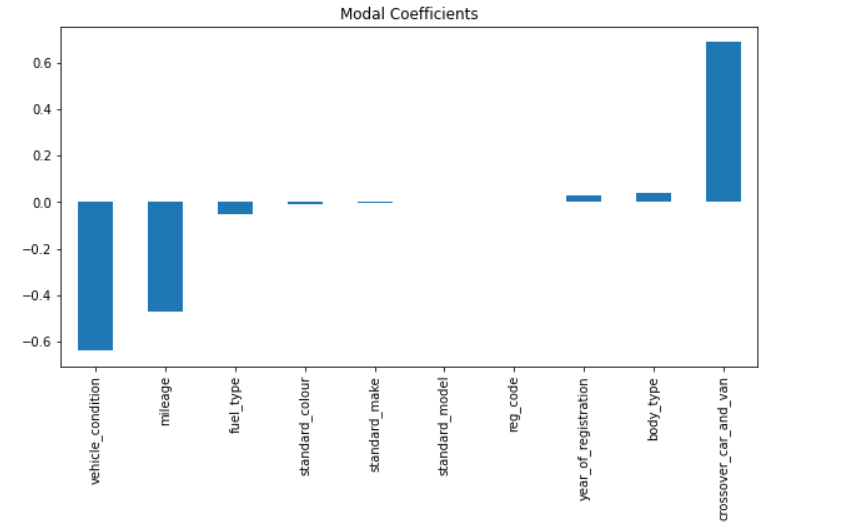
coef = pd.Series(best\_model.coef\_, X\_train.columns).sort\_values()

plt.figure(figsize=(10,5))

coef.plot(kind='bar', title='Modal Coefficients')

plt.show()

vehicle condition is negative important to price prediction, followed by mileage and fuel type while Crossover car and van as the maximum positive importance to price prediction, followed by body type and registration year.



## 

## 4.3. Fine-Grained Evaluation

The predictions of linear regression have bene evaluated by the score of mean absolute error, mean squared error and R2 score. MAE is the error between pair of true and predicted price values. There is an error or 0.53 in the predicted car price. MSE is the average squared error between the predicted and actual price of a car. R2 tells how data fits the model where 1 means all the variations have been explained by the model while 0 means the poorest understanding of the data. 0.5023 value represents that model learnt only 50% of the variations in the data.

# Analyze individual predictions and distribution of scores/losses

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("MAE: ", mae)

print("MSE: ", mse)

print("R2: ", r2)

