**Car Price Prediction With Machine Learning in Python — Portfolio Project**

Using Pandas, Numpy, Scikit-Learn, Streamlit and Streamlit Cloud

[[Victor Umunna](https://victorumunna.medium.com/?source=post_page-----11f57812a1b--------------------------------)](https://victorumunna.medium.com/?source=post_page-----11f57812a1b--------------------------------)

[[Python in Plain English](https://python.plainenglish.io/?source=post_page-----11f57812a1b--------------------------------)](https://python.plainenglish.io/?source=post_page-----11f57812a1b--------------------------------)

[Victor Umunna](https://victorumunna.medium.com/?source=post_page-----11f57812a1b--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fdc0de295f01c&operation=register&redirect=https%3A%2F%2Fpython.plainenglish.io%2Fcar-price-prediction-with-machine-learning-in-python-portfolio-project-11f57812a1b&user=Victor+Umunna&userId=dc0de295f01c&source=post_page-dc0de295f01c----11f57812a1b---------------------post_header-----------)

Published in

[Python in Plain English](https://python.plainenglish.io/?source=post_page-----11f57812a1b--------------------------------)

·

9 min read

·

Jul 12, 2023

56

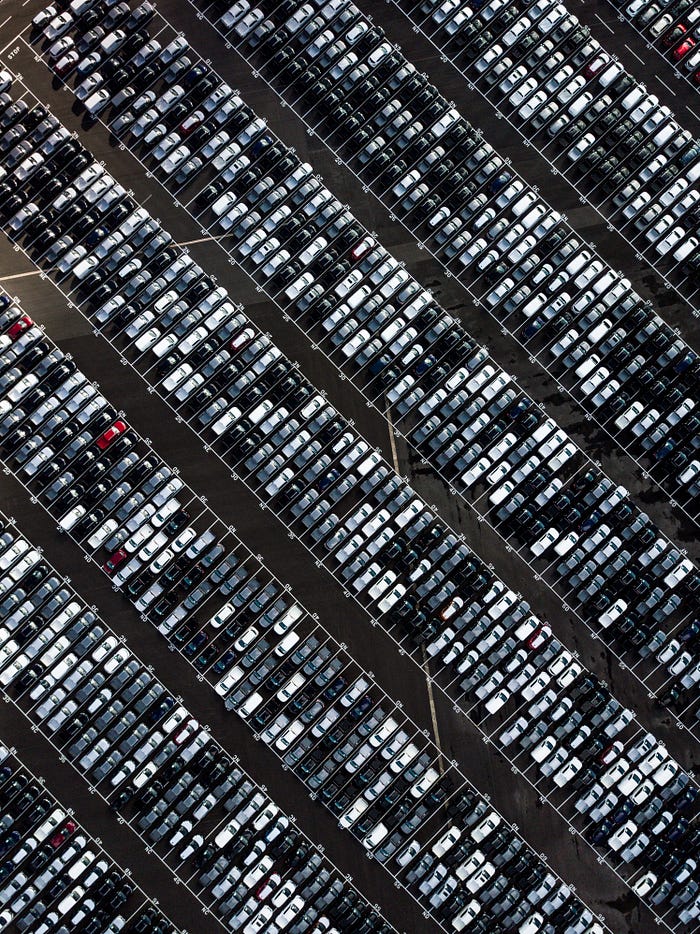


Photo by [Ryan Searle](https://unsplash.com/@ryansearle) on [Unsplash](https://unsplash.com/" \t "_blank)

**Introduction**

Estimating the sale prices of cars on auction is one of the basic data science projects that can help you build a good portfolio. By finishing this article, you will be able to get different car data from a car auction site and predict the price of those cars using various types of linear regression algorithms.

**Steps Involved**

1. Web Scraping a car auction site using the beautiful soup package
2. Cleaning the raw data and creating new features
3. Data visualisation on the car price data
4. Modelling the data using the algorithms
5. Evaluating the built model using the evaluation metrics
6. Web App Development and Deployment

Let’s get started

**Web Scraping**

Web scraping is an important skill for any data scientist to have. It is the process of extracting specified information and data from a website and transforming it into structured data for analysis.

This is the first and most important step because that is how we get the car data we will analyse and build a model.

Firstly, I searched Google for car auction sites that have good descriptions of all cars, and then I finally picked [Auction Export](https://www.auctionexport.com/en/Inventory).

I web-scraped this [site](https://www.auctionexport.com/en/Inventory/Search_Results?OrderBy=None&Offset=0&PerPage=50&Keywords=&IsManual=True&vehicleType=2&MakeModel%5B0%5D.autoMake=&MakeModel%5B0%5D.autoModel=&MakeModel%5B0%5D.autoTrim=&price_from=0&price_to=1000&autoYear_from=1901&autoYear_to=2023&mileage_from=0&mileage_to=0&withPicsOnly=false&ExcludeNonDrivable=false&SearchName=&NotifyDays=7&NotifyFrequency=24) for cars using the Beautiful Soup Python package. The desired features are the name, price, mileage, colour, and transmission of each car. I successfully saved 2055 car details in a CSV file, which is good enough to work with.

Here is the code:

**Data Cleaning**

To produce high-quality models, it is necessary to format and correct our cleaned data thoroughly. This particular task involves ensuring that the data is immaculate and polished.

Let's examine the data's shape after removing 10 rows that contain null values.

data.shape  
>>> (2986, 6)

I'll review each column and format them accordingly.

**Name**

In this column, you'll find a list of car brands available for purchase. I encountered a challenge as the values included both brand and model names. However, I only required the brand name for my task. Also, there was inconsistency in the way words are stored, resulting in different values for the same brand, such as Mercedes, Mercedes, and Mercedes, being recognised as distinct names.

I cleaned this column using the following methods:

* I extracted just the brand name from the values. Luckily, the brand names are the first words in the values. For example, I removed ‘Toyota’ from ‘Toyota Yaris’ by indexing just the first word in the values

data['name'].apply(lambda x: x.split(' ')[0])

* Then I made the brand names uniform using the ‘replace’ function

data.replace({'manufacturer' :   
 {'MAZDA' : 'Mazda', 'JAGUAR':'Jaguar', 'AUDI' : 'Audi', 'NISSAN': 'Nissan', 'MINI': 'Mini',   
 'VOLKSWAGEN':'Volkswagen', 'VAUXHALL':'Vauxhall', 'TOYOTA':'Toyota', 'SKODA':'Skoda', 'FORD':'Ford',  
 'Bmw':'BMW','SUZUKI' : 'Suzuki', 'RENAULT':'Renault', 'PEUGEOT':'Peugeot', 'CITROEN':'Citroen',  
 'VOLVO':'Volvo', 'FIAT':'Fiat', 'Ds':'DS', 'DACIA':'Dacia', 'ABARTH':'Abarth', 'SMART':'Smart',   
 'smart':'Smart','SEAT':'Seat', 'MITSUBISHI':'Mitsubishi', 'KIA':'Kia', 'HYUNDAI':'Hyundai',  
 'HONDA':'Honda','MASERATI':'Maserati', 'PORSCHE':'Porsche', 'INFINITI':'Infiniti', 'Alfa':'Alfa-Romero',  
 'Mercedes': 'Mercedes-Benz', 'MERCEDES-BENZ': 'Mercedes-Benz', 'Mercedes-benz': 'Mercedes-Benz',  
 'Land':'Land-Rover', 'LAND': 'Land-Rover'}  
 })

This reduced the unique count of cars from 74 to 40. Brilliant, right?

**Price**

For this column, I removed the pound (£) sign and the comma and then converted the value from *object* to *integer.*

data['price'].str.replace(',', ''). str.replace(£', '').astype(np.int64)

**Year**

Instead of using the manufacturing year as is, I opted to convert it to the age of the cars for this column.

* First, I converted the values in the year column to integers

data['year'].astype(str).astype(np.int64)

* Then I created a new column called age by subtracting the years from the current year

Current\_Year = 2023  
data['age'] = Current\_Year - data['year']  
data['age'] = data['age'].astype(np.int64)

**Mileage**

This accounts for the number of miles travelled or covered.

I cleaned the values by removing the commas, spaces, and ‘miles’ attached to the values.

data['mileage'].str.replace(',', '').str.replace(',', '').str.replace('miles', '').astype(np.int64)

**Engine**

This accounts for the engine type of each car.

I cleaned the value to reflect just four types — Petrol, Diesel, Hybrid, Electric and plug-in hybrid.

data = data.replace({'engine' :   
 {'Petrol hybrid': 'Hybrid', 'Petrol hybrid': 'Hybrid',   
 'Petrol / electric hy' : 'Hybrid', 'Petrol plug-in hybri': 'Plug\_in\_hybrid',  
 'Petrol/electric' : 'Hybrid'}  
 })

**Transmission**

This accounts for the transmission for each car.

I ensured that the naming convention remained consistent to distinguish between Manual, Semi-automatic, and Automatic types.

data.replace({'transmission' :   
 {'Semi auto': 'Semiautomatic', 'Semiauto': 'Semiautomatic',  
 'Semi automatic': 'Semiautomatic', 'Manual ': 'Manual',  
 'Semi-automatic': 'Semiautomatic', 'G-tronic+': 'Automatic',  
 'Cvt': 'Automatic'}  
   
 })

This ends the data cleaning part.

**Data Exploration**

This procedure enables us to analyse and gain insights from our data. It will also help us determine whether the data we are working with is statistically correct.

There are numerous details to examine, and the code below will assist you in doing so.

#the most expensive cars in the collection  
data.nlargest(5, 'price')  
  
#cars with the most miles travelled in the collection  
data.nlargest(5, 'mileage')  
  
#oldest cars in the collection  
data.nlargest(5, 'age')  
  
#numbers of cars under the various car manufacturers  
data.manufacturer.value\_counts()  
  
#count of the manufacture years   
data.year.value\_counts()

We can check the correlation between our features

# correlation  
corr = data.corr()  
corr



Correlation matrix

This shows that there is a positive correlation between the age of the car and the number of miles driven. This means that the lesser the age of the car, the lesser the miles driven.

Also, there is a negative correlation between the age of the car and the price. This means that the older the car, the lower the price.

Finally, there is a negative correlation between the miles driven and the price of the cars.

We can also visualise these using a heatmap.



Heatmap of the data

Graphs can be plotted to show the relationship between the age of the car, the price and the mileage

**Model Building**

This is the part where we build our predictive model using regression algorithms.

First, I imported the necessary libraries.

import pandas as pd  
from sklearn.preprocessing import LabelEncoder, MinMaxScaler  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.linear\_model import LinearRegression, Ridge  
from sklearn.ensemble import ExtraTreesRegressor, RandomForestRegressor  
from lightgbm import LGBMRegressor  
from xgboost import XGBRegressor  
from catboost import CatBoostRegressor  
from sklearn.metrics import mean\_absolute\_error, accuracy\_score, mean\_squared\_error  
from sklearn import metrics  
import pickle

Then, I dropped the features that would not be used for this problem. I dropped the **name** and **year**

#drop the name and year columns   
data = data.drop(['name', 'year'], axis=1)

I performed the following preprocessing steps:

* Label the categorical columns with the LabelEncoder module. Each column's values are encoded and given a unique number.

# label encode the categorical values  
le\_manufacturer = LabelEncoder()  
le\_engine = LabelEncoder()  
le\_transmission = LabelEncoder()  
  
data['manufacturer'] = le\_manufacturer.fit\_transform(data['manufacturer'])  
data['engine'] = le\_engine.fit\_transform(data['engine'])  
data['transmission'] = le\_transmission.fit\_transform(data['transmission'])

* I created my X and Y variables. Manufacturer, age, mileage, engine, and transmission are the X variables (independent), while Price is the Y variable (target or dependent).

# creating X and y variables  
X = data.drop('price', axis=1)  
  
# log transform the price column  
y = np.log(data['price'])

The use of a logarithm in this comes from its ability to ‘linearize’ the relationship between the independent (X) and dependent (Y) variables. This means that the logarithm of Y corresponds to a linear function of the logarithm of X.

* Split the X and Y variables into training and testing data

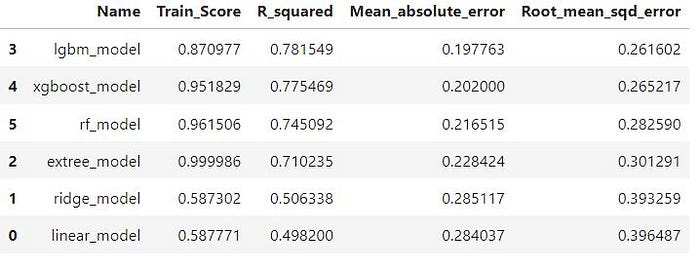
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,   
 test\_size=0.3, random\_state=42)

* Normalisation was used to feature-scale the X\_train and X\_test variables. Normalisation is a scaling technique used in Machine Learning to change the values of numeric columns in a dataset to use a standard scale during data preparation. In machine learning, normalisation is the process of converting data into the range [0, 1] (or any other range) or simply transforming data onto the unit sphere.

# feature scale the X\_train and X\_test values  
  
norm = MinMaxScaler().fit(X\_train)  
  
#transform training data  
X\_train = norm.transform(X\_train)  
  
#transform testing data  
X\_test = norm.transform(X\_test)

Finally, we develop various regression models and evaluate the metrics. Regression models that I create include Linear Regression, Ridge, ExtraTrees, LGBM, XGBoost, and RandonForest regressors.

models = {  
 'linear\_model': LinearRegression(),  
 'ridge\_model': Ridge(random\_state=123),  
 'extree\_model':ExtraTreesRegressor(random\_state = 123),  
 'lgbm\_model':LGBMRegressor(random\_state = 123),  
 'xgboost\_model':XGBRegressor(random\_state = 123),  
 'rf\_model' : RandomForestRegressor(random\_state = 123)  
 }  
  
  
def train\_model(models: dict) -> pd.DataFrame:  
 """  
It takes a dictionary containing a key pair of model names and estimators.  
 It returns a data frame containing the metrics of the trained model.  
 """  
 my\_dict = {}  
 name\_list, train\_score\_list, r\_sqd\_list, mae\_list, rmse\_list = [], [], [], [], []  
 for name, estimator in models.items():  
 # fit  
 estimator.fit(X\_train, y\_train)  
  
 # make predictions  
 y\_pred = estimator.predict(X\_test)  
  
 # metrics  
 train\_score = estimator.score(X\_train, y\_train)  
 r\_sqd = metrics.r2\_score(y\_test, y\_pred)  
 mae = metrics.mean\_absolute\_error(y\_test, y\_pred)  
 mse = metrics.mean\_squared\_error(y\_test, y\_pred)  
 rmse = np.sqrt(mse)  
  
 # add the metrics to the empty list  
 name\_list.append(name)  
 train\_score\_list.append(train\_score)  
 r\_sqd\_list.append(r\_sqd)  
 mae\_list.append(mae)  
 rmse\_list.append(rmse)  
  
 my\_dict["Name"] = name\_list  
 my\_dict["Train\_Score"] = train\_score\_list  
 my\_dict["R\_squared"] = r\_sqd\_list  
 my\_dict["Mean\_absolute\_error"] = mae\_list  
 my\_dict["Root\_mean\_sqd\_error"] = rmse\_list  
  
 my\_dataframe = pd.DataFrame(my\_dict)  
 my\_dataframe = my\_dataframe.sort\_values("Root\_mean\_sqd\_error")  
 return my\_dataframe  
  
  
  
train\_model(models)



result of the different models

The metrics for evaluating our models are:

* R-squared is a statistical measure that represents the proportion of the variance explained by an independent variable or variables in a regression model for a dependent variable. It is also referred to as the coefficient of determination.
* The mean absolute error (MAE) is a measure of the difference between two continuous variables. It is calculated as the average of the absolute difference between the predicted and actual values.
* Root Mean Square Error (RMSE) is a quadratic scoring rule that also measures the average magnitude of the error. It is calculated as the square root of the average squared differences between predicted and actual values.

The RMSE is thought to be the best measure of our error because it gives more weight to points farther away from the mean than the MAE, which seeks to minimise the impact of outliers.

The LGBM model has the lowest RMSE and will be considered the best-performing model.

We will attempt to improve the model by using the GridSearchCV module for hyperparameter tuning to obtain the best parameters for our models.

#create the grid  
grid = {'max\_depth': [3,4,5],'n\_estimators':[100, 200, 300]}  
  
#Instantiate GridSearchCV  
model = GridSearchCV (estimator = LGBMRegressor(random\_state = 123), param\_grid = grid, scoring ='neg\_root\_mean\_squared\_error'))  
  
#fit the   
model.fit(X\_train,y\_train, verbose = False)

#predict with the model  
y\_pred = model.predict(X\_test)

#get the metrics  
grid\_model = pd.DataFrame()  
 'model': ['LGBM'],  
 'r\_squared': [metrics.r2\_score(y\_test, y\_pred)],  
 'mae': [mean\_absolute\_error(y\_test, y\_pred)],  
 'rmse': [np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))]  
 })  
grid\_model



LGBM model with improved metrics

The RMSE was reduced, but not by much.

We can run the model on new data to see how it does.

This brings us to the end of the model-building process. Let us save this model as a pickle file.

data = {"model": model, "normalisation": norm}  
with open('../models/regressor.pkl', 'wb') as file:  
 pickle.dump(data, file)

**Web App Development and Deployment**

The goal of developing and deploying a web app for your project is for people to be able to use it without understanding what is going on in the backend.

In this section, we will build a simple web app using Streamlit and deploy the app using Streamlit Cloud

[Streamlit](https://www.streamlit.io/) is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps.

[Streamlit Cloud](https://streamlit.io/cloud) is an open and free platform for the community to deploy, discover, and share Streamlit apps and code.

Here is the code that I used to create the app.

Check out these resources to learn how to build and deploy Streamlit apps:

* Streamlit docs — <https://docs.streamlit.io/>
* How to Build Your First Data Science Web App in Python by Data Professor — <https://www.youtube.com/watch?v=ZZ4B0QUHuNc>
* Python Streamlit Full Course by Nileng Production — <https://www.youtube.com/watch?v=RjiqbTLW9_E&list=PLa6CNrvKM5QU7AjAS90zCMIwi9RTFNIIW>
* How to Deploy Your App to Streamlit Community Cloud by Data Professor — <https://www.youtube.com/watch?v=HKoOBiAaHGg>
* Deploy Streamlit Apps For Free on Community Cloud by Fanilo Andrianasolo — <https://www.youtube.com/watch?v=kULH9m_Os50>

This comes to the end of our portfolio project. 👏🏽

In this project we have done:

* Web scraping using BeautifulSoup
* Data Cleaning using pandas and numpy
* Exploratory Analysis using pandas
* Performed data preprocessing techniques such as label encoding, train-test split, normalization
* Built 5 Regression models using Scikit-learn and evaluate the metrics
* Built a Streamlit Web App and Deploy it using Streamlit Cloud

Link to my Github Page for this project: <https://github.com/VictorUmunna/Car-Price-Prediction>

**If you found this article useful, please give it a thumbs up and share it with other data enthusiasts. Follow me on [Linkedin](http://linkedin.com/in/umunna-victor-obinna" \t "_blank),**[**Twitter**](https://twitter.com/Mazi_Obinna)**, and**[**GitHub**](https://github.com/VictorUmunna)**.**

*More content at*[***PlainEnglish.io***](https://plainenglish.io/)*.*

*Sign up for our*[***free weekly newsletter***](http://newsletter.plainenglish.io/)*. Follow us on*[***Twitter***](https://twitter.com/inPlainEngHQ), [***LinkedIn***](https://www.linkedin.com/company/inplainenglish/)*,*[***YouTube***](https://www.youtube.com/channel/UCtipWUghju290NWcn8jhyAw)*, and*[***Discord***](https://discord.gg/GtDtUAvyhW)***.***