Car Deal Spotter – Technical Guide

Student: Glen Devlin

Student Number: 12524303

Email: glen.devlin4@mail.dcu.ie

Project Supervisor: Darragh O'Brien

Date Finished: 20/5/18

Motivation

The motivation behind this project is to allow users to find cars being sold at a good value price, on the website CarsIreland.ie. CarsIreland.ie is a website that lets people sell second hand cars. Users of the site upload pictures of their car accompanied by details such as the price, year, odometer etc. along with their contact details. To find cars that were being sold at a good price I scraped thousands of ads like this and built a model for each type of car I scraped. The model was to predict a car's value and then using this I was able to find cars that were being sold at a good price i.e. their selling price was less than their estimated value. These cars were what I considered to be good deals.

This is useful as it helps consumers avoid getting ripped off. Users can browse through a particular type of car and see a list of cars for sale; accompanied by their selling price, their estimated vale and the difference between these two numbers. It also helps cars sellers get an idea of how much they should be selling their cars for. Also to assist both buyers and sellers, the application part of this project comes with a feature that allows users to value a car. User's enter in details about a car and using the same model and training data used to predict the estimated value of the cars on sale, the application can tell user's how much their car is worth.

The over-all process I followed to accomplish this was firstly to build a web-scraper. Using this I scraped thousands of ads from CarsIreland.ie, choosing cars that had plenty of models for sale. The models I chose were: Audi A4, BMW 3 Series and 5 Series, Ford focus, Nissan Qashqai, Volkswagen Gold and Passat, Opel Astra and Skoda Octavia. The number of models I had for each of these fluctuated with Ford Focus being the car I had the most ads of. As the data was scraped it was automatically stored in an SQLite database. From here I could clean that data and then fill in missing values so I could use the data as training data. After I had enough training data I could then construct a model that was able to predict a cars price. The model performed the prediction using Multiple Linear Regression. Every features relation to the price was plotted using linear regression and then theses were added together to get the estimated value.

After the cars that were on sale were given a predicted price they were ready to be seen by the user. The user interface portion of this project is an Android application. The application allowed users to enter search parameters and browse cars that were for sale using these search parameters. They could scroll through a list and click on these cars to get more details. They could also press a button that would connect them directly to the advert on CarsIreland.ie. To allow the application to see the data, I set up and online MySQL database on the hosting site pythonanywhere.com. This allowed the phone to speak to a python program on the server that could retrieve information from the MySQL database. As stated earlier the application also had access to the algorithm that made the models. This site also allowed me to easily implement the python programs I had made earlier. Allowing me to host the algorithm I used to create my models online, this is what allows users to value their cars.

Research

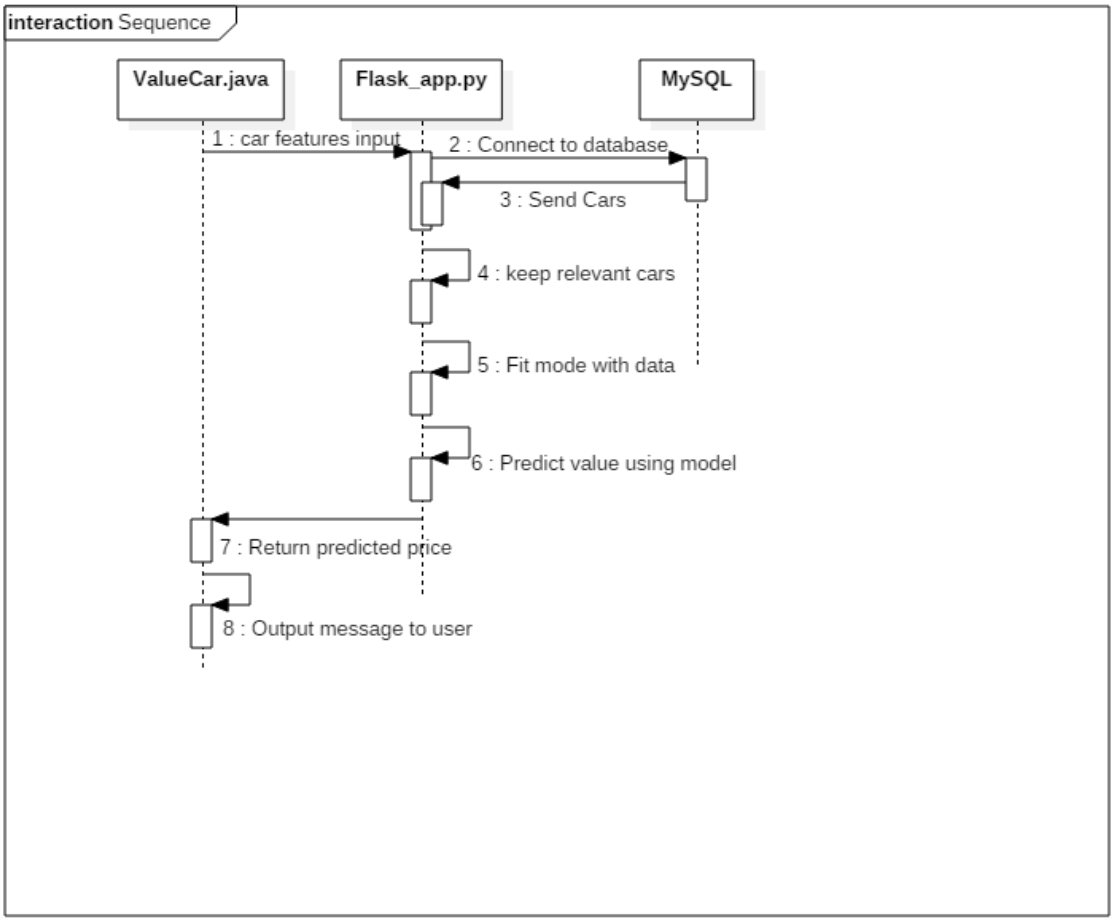
This project involved me learning many knew things. Although the Android application was coded in Java, the vast majority of this application was done in python. Python is a language that is still relatively new to me but I chose it as I learned that its strengths lay in data management and machine learning. I found the language very easy to use and its functionality perfectly suited to what I was doing. I also chose python because of the wide range of libraries it had related to what I was doing. Web scraping is also something new to me but I found the python library Beautiful soup and this made parsing HTML very easy. This was used in conjunction with the requests library, which retrieved the content from the ads page.

This project was my first time implementing any kind of machine learning algorithms. When looking for a library to use, the one that seemed most often recommended was sci-kit learn. This library seemed to be easy to use and easy to understand and in the end, it proved to be both of these things. I also learned how to use the Pandas library and the Numpy library as sci-kit needs to make use of the Data Frames and arrays these libraries offer respectively. To accompany the implementation of the machine learning I also had to choose a predictive classifier. The classifier I chose was multiple linear regression as I found it both easier to understand and it seemed like a classifier that would offer good results. I learned to understand both how multiple linear regression worked and what steps I could take to improve it once I had it implemented. I learned all of the information relating to making the predictions, such as using sci-kit, from the website Dataquest.io.

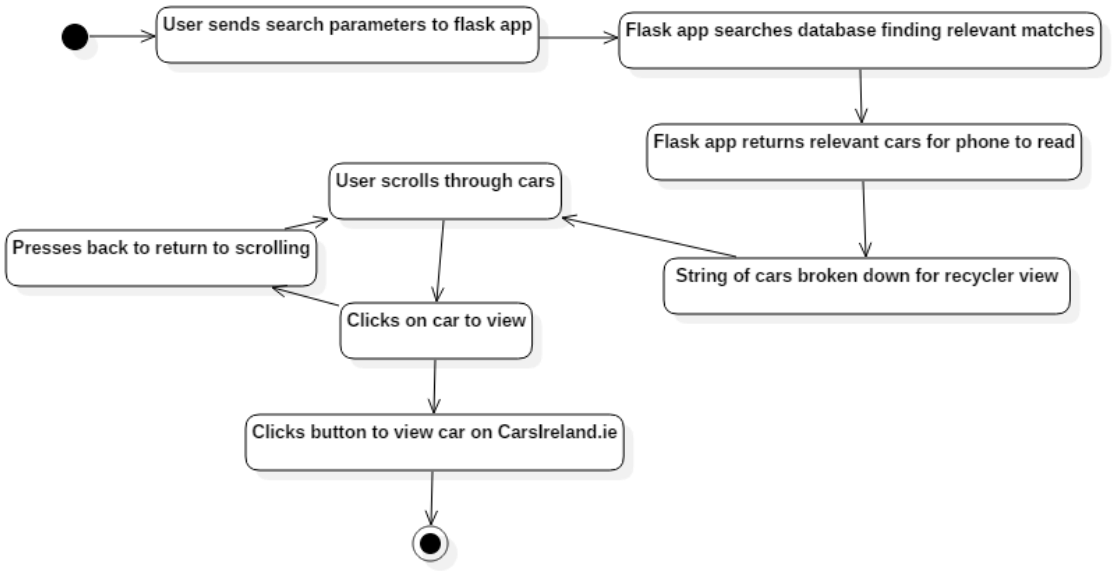
When it came to choose a place for hosting my programs and database I looked at a few options. In the end I chose python anywhere because it seemed at the time, to be geared directly towards what I needed. It offered simple tools for accessing a MySQL database and creating a web app which allowed me to quickly use the I had already made online. It also allowed me to install sci-kit on the version of python I was using in the web app. It offered me frameworks that would allow me to communicate between my home machine and the web app and an android device and the web app. The framework I chose was Flask. Flask is a micro-framework for python. I was torn between Django and Flask but I chose flask because it offered the functionality I needed and it was easier to implement.

Design

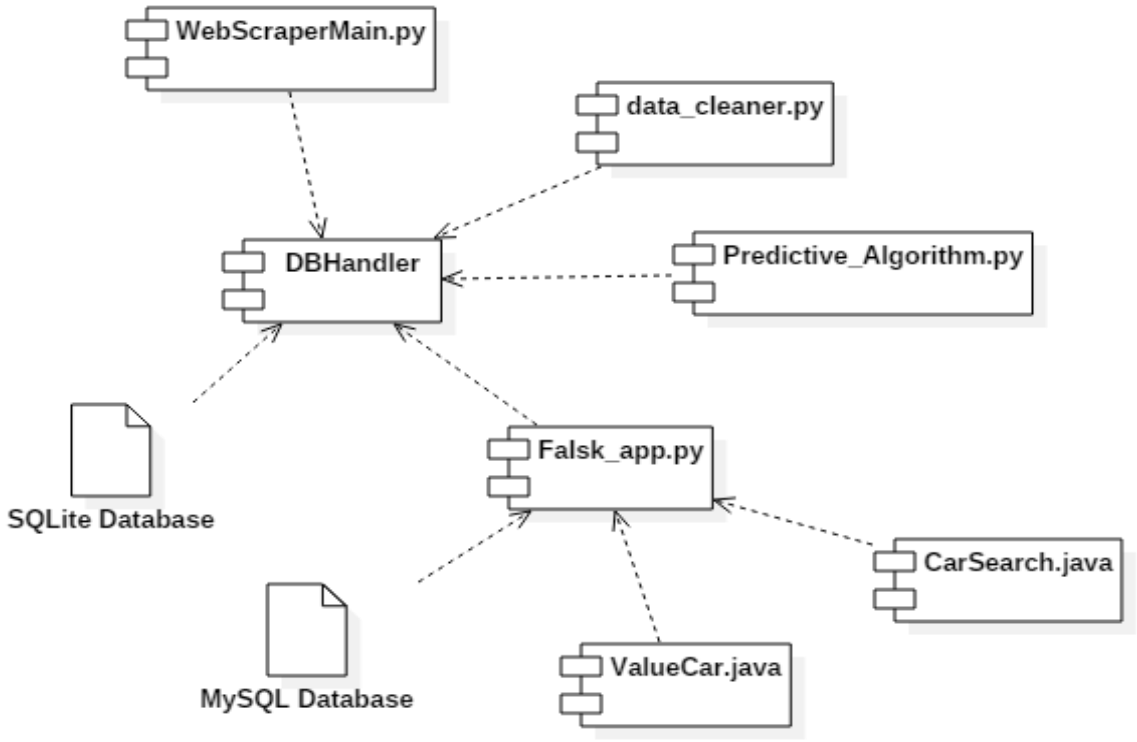
A sequence Diagram of the Value Car feature of the front-end app.



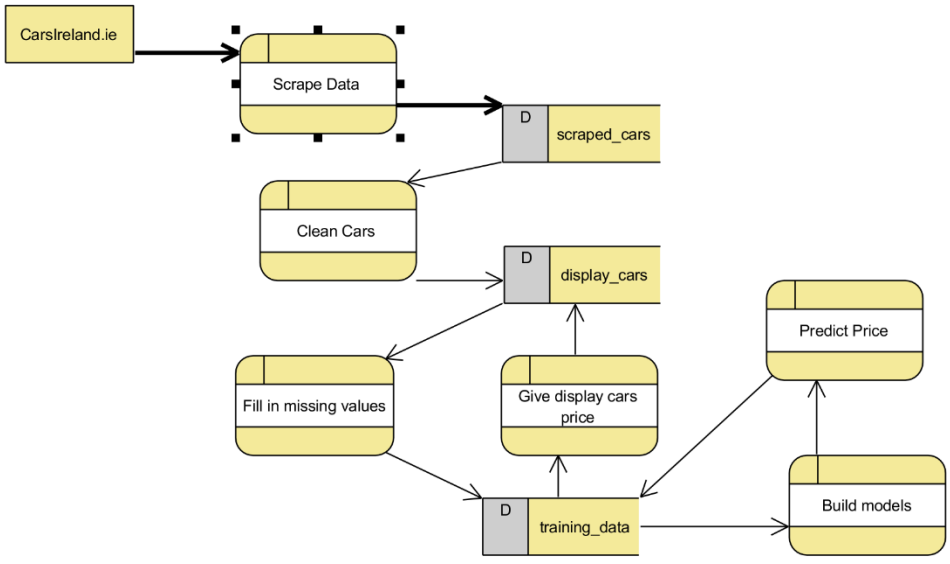
An activity diagram showing how the Browsing feature of the android app works



A component diagram showing most relevant components of the system and how they communicate



A data flow diagram of the early stages of the scraping and prediction process



Implementation

Data

The Data I am using consists of thousands of car ads. The number of ads per model fluctuates with the type of car I have the most of being the Ford Focus. The features I take from each ad are the URL, an ID, Make, Model, Year, Price, Odometer, Fuel Type, Engine Size, Colour, Body, Owners, Transmission, County, Road Tax, Fuel Economy, along with some other features which are not used, such as the date the car was scraped. The cars are given two more attributes later in the process. They are given a Predicted Price (this is the estimated value of the car) and a Price Difference (The difference between the Predicted Price and the actual Price).

Data is split across two tables. Display cars, which are the latest cars scraped. They have their features cleaned and trimmed of useless information but they do not have missing attributes filled. These cars are then added to the training data and then they have their attributes filled. This is because the model cannot accept Nan values. The training data operates on itself when it is increased. Giving every car a predicted price. Cars are then matched with cars in the display set by their ids, so that the cars in the display set each have a predicted price.

Python

Web scraper

The web scraper is a program I made to scrape the car ads from CarsIreland.ie. Each make on CarsIreland.ie has a corresponding number, as does each model. By getting the each of these numbers the scraper is able to construct the URLs for pages each made up of 20 car ads for that particular make and model. The scraper then goes through each of these pages and retrieves the URL for each car ad. Once the scraper has the URLs it begins visiting every car page retrieving the content from that page. It takes the car's main features along with the cars county and ID which is embedded in the URL of the car ad. Once it has all the features it requires for a car, it begins writing the car to a SQLite database.

The Web Scraper makes use of two python libraries:

- BeautifulSoup: A Python library used for extracting data out of HTML and XML files. Car features are embedded between <td> tags on a car page so I use BeautifulSoup to find all these tags and only save the first few as these contain the information I want. To find the address I look for the HTML attribute labelled address and then search for a county in the returned string.

- Requests: A Python library used for HTTP requests. It is what the scraper uses to check that URLs are open and to retrieve the content form those pages. Requests returns the HTML content so BeautifulSoup can parse through it.

The only component the web scraper interacts with is the DBHandler. It used the DBHandler to write data to the SQLite databases.

https://www.carsireland.ie/search-results.php?make_id=26&model_id=253

You can notice the make ID and the model ID in this URL. The scraper constructs the first of these links then uses the that to scroll through the pages of car ads getting all the cars URLs for each car ad.

Data Cleaner

The data cleaner was a program I originally used to clean data, however as time went on I started to rely less on python for cleaning and more on SQL. The Data cleaner program is made up of a number of functions for finding features in a column that do not belong. It also contains a function called fix_row(), which can take car with its features in the wrong place and then sort them into the right place. The most useful functions this program contains are functions that fill in the data of missing columns. This is crucial to build a training data set that can be used by the model. The Data Cleaner uses the DBHandler to read and write to the SQLite database files.

DBHandler

The DBHandler (short or Database Handler) is a program to facilitate the communication between all my other programs and the database tables. It is made up of many functions for reading and writing to the databases. It is how the predictive algorithm, data cleaner and web scraper all speak to the database. It is also how the databases are uploaded to the MySQL databases on pythonanywhere.com.

Predictive Algorithm

The predictive algorithm is the core component of this project. It builds each of the predictive models that are used to give cars a predicted price. It reads in the training data as a list, starts at the first car model and takes cars from the list with the same car model, then it converts this data into a data frame. After the data is converted into a Data Frame it can then be fitted to a model. This model can then be used to predict the price of each of the cars in the list and write the price back into the Predicted Price column in the SQLite database. After it has done the first car model which is Audi A4s, it moves onto the next car model and repeats the process.

It contains other functions that can produce metrics based on a given model, it can show feature correlation, feature variance, mean squared error between the predicted price and feature variance. It can also display the coefficients and intercept of a car model.

- Sci-Kit Learn: Sci-kit Learn is a machine learning library for Python. The linear regression models are implemented using Sci-kit. The models can fit a Data Frame and then individual cars can be passed to it to make predictions. Sci-kit is also used to encode categorical features as numbers so that they can be used in the models. The mean squared error is also an implementation from the sci-kit library.

- Pandas: Pandas is a Python library use to transfer lists into Data Frames. Data Frames are structures that look similar to how Data is stored in a Database. Data is stored as rows and columns with each column having a column heading. Sci-kit models can only fit the data in the form of Data Frames.

- Numpy: A Python package for scientific computation. Does some calculations in the predictive algorithm, such as getting the squared root of the mean squared error. Sci-kit depends on Numpy being installed

The predictive algorithm reads in data using the DBHandler and writes the predictive algorithm back to the databases using the DBHandler.

Flask App

Flask is a micro-framework for constructing web apps using Python. It allowed me to construct a Python program in which every function had a URL address. That made the functions reachable form my laptop and the android application. Whatever was connecting to the flask app could then read the return result from the screen. For example:

http://gdevlin.pythonanywhere.com/hello_world

By visiting this address, the user would be greeted with the string “Hello World”. They could then use either the requests library in python or the URL class in java to read the content that was returned. Variables could also be passed to the functions.:

http://gdevlin.pythonanywhere.com/hello/Glen

Visiting an address like this could display the String “Hello Glen” to the user, as the values behind the second slash could be accepted as a string variable. Using this function, I was able to pass information to the flask app or retrieve information from the flask app by reading the content of a page.

The flask app spoke directly to the MySQL database stored on pyhtonanywhere.com. It is through the flask app that I upload data to the database. It is also through the flask app the Android application speaks to the database. Cars are pulled from the database through the flask app so users can browse through cars that are for sale. The flask app also holds a

similar implementation of the predictive algorithm. The user passes the car features from the Value Car feature on the android app to the flask through the URL. The app takes those features, boots up the corresponding predictive model, then returns the predicted price of the car to the screen. The Android app can the read the value that is returned and display this as an alert to the user.

Java

Android Application

Browse cars

The browse cars feature of the android application is made up of 3 activities. The search activity, the browse activity and the view activity. The search activity allows the users to enter in search parameters through dropdown menus. These search parameters are then sent to the flask app inside of a URL. The flask app then pulls cars that are for sale from the database and prints them to the screen. Data is parsed as a string which is then broken up when it reaches the Android application. In the string, individual car features are split by a pipe operator ('|') and cars are split from other cars using a left curly brace ('{'). The android app reads all of this information from the screen and then passes the arraylist containing the cars onto the browse activity.

The browse activity is a list containing all the cars. The list is an android widget known as a recycler view. The recycler view is a custom scrollable list that uses a custom layout. This layout shows some information about the car. Clicking on one of these layouts will pass the user onto the view car activity. Here they can see all the car information that was stored in the database. There is also a button at the bottom of this activity that will link the user to the original ad on CarsIreland.ie.

Value Car

The value car feature works by allowing users to enter car information and then pass these features onto the flask app which then boots up the predictive algorithm, which creates a predictive model that can evaluate a car and give it a predicted price. All features are entered through drop down menus apart from odometer which must be entered manually. These features are then inserted into a URL and using the java URL class it creates a connection to that URL. On connection to the URL the Flask app reads in the parameters and sends these to the model as features to get a predicted price. Once the page loads the car's predicted price will be there and the application will be able to read it.

Sample code

Predictive model

Here you can see the model being fit using sci-kit and returned, cars are only compared to cars of the same model so 'Make' is an irrelevant feature here

```
#Create multiple linear regression model and return it
def multiple_regression_model(car_data_frame):
mult_lin_reg = LinearRegression()

dependent_variables = list(car_data_frame.columns.values)
dependent_variables.remove('ID')
dependent_variables.remove('Make')
dependent_variables.remove('Price')

mult_lin_reg = mult_lin_reg.fit(car_data_frame[dependent_variables],
car_data_frame['Price'])

return mult_lin_reg
```

Code for returning the mean squared error and the root mean squared error

```
#Get mean squared error for a model
def get_mean_square_error(list_of_cars, price_series):
    car_predicted_prices = []
    #car_predicted_prices = list_of_cars

    for car in list_of_cars:
        car_predicted_prices.append(car[20])

    mse = mean_squared_error(car_predicted_prices, price_series)
    return mse

#Get the root mean squared error
def get_RMSE(mse):
    return np.sqrt(mse)
```

A function for printing the coefficients from each model, they can be found using model.coef_

```
def model_coeffients_and_intercept():
    table_choice = "training"
    list_of_cars = read_cars(table_choice)#Read Training Cars

    car_selection = 1
    while car_selection < 9:
        make = str(car_selection)
```

```
        selected_cars = make_and_model(list_of_cars, make)

        car_data_frame = convert_list_to_dataframe(selected_cars)
        mult_regression_model = multiple_regression_model(car_data_frame)

        for coeffiecient in mult_regression_model.coef_:
            print(coeffiecient)

        print(mult_regression_model.coef_)
        print()

        car_selection += 1
```

Data must be normalised to get accurate measures of some of the metrics

```
#Normalise data frame to get correlations
def normalise_data_frame(car_data_frame):
    features_to_normalise = list(car_data_frame.columns.values)
    features_to_normalise.remove('ID')
    features_to_normalise.remove('Make')
    features_to_normalise.remove('Price')

    for feat in features_to_normalise:
        car_data_frame[feat] = (car_data_frame[feat] -
car_data_frame[feat].min()) / (car_data_frame[feat].max() -
car_data_frame[feat].min())

    return car_data_frame
```

Flask app

This is some code from the flask app, notice the start of the address starting with add_training_car followed by the cars features as variables being sent after each /. Car URLs must be encoded and decoded on this end to keep their original form

```
#Add car to training_cars table
@app.route('/add_training_car/<URL>/<ID>/<Make>/<Model>/<Year>/<Price>/<Od
ometer>/<Fuel_Type>/<Engine_Size>/<Colour>/<Body>/<Owners>/<Transmission>/
<NCT>/<Road_Tax>/<Fuel_Economy>/<SIMI_Dealer>/<Last_Update>/<Date_Scraped>
/<excess>/<Predicted_Price>/<price_diff>')
def add_car_training(URL, ID, Make, Model, Year, Price, Odometer,
Fuel_Type, Engine_Size, Colour, Body, Owners, Transmission, NCT, Road_Tax,
Fuel_Economy, SIMI_Dealer, Last_Update, Date_Scraped, excess,
Predicted_Price, price_diff):

    car_url = URL.replace('FORWARDSLASH', '/')
    car_url = car_url.replace('QUESTIONMARK', '?')
    car_url = car_url.replace('PERCENTAGESIGN', '%')
    car_price_diff = int(price_diff)
```

```
    car = [car_url, ID, Make, Model, Year, Price, Odometer, Fuel_Type,
Engine_Size, Colour, Body, Owners, Transmission, NCT, Road_Tax,
Fuel_Economy,
                SIMI_Dealer, Last_Update, Date_Scraped, excess,
Predicted_Price, car_price_diff]

    #return str(car)
    #return str(car)

    conn = MySQLdb.connect("GDevlin.mysql.pythonanywhere-services.com",
"GDevlin", "cardealspotter", "GDevlin$Car_Deal_Spotter")
    c = conn.cursor()

    add_car = ("INSERT INTO training_cars "
               "(URL, ID, Make, Model, Year, Price, Odometer, Fuel_Type,
Engine_Size, Colour, Body, Owners, Transmission, NCT_Due, Road_Tax, "
               "Fuel_Economy, SIMI_Dealer, Last_Update, Date_Scraped,
excess, Predicted_Price, price_diff) "
               "Values(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s, %s, %s, %s, %s, %s, %s, %s, %s);")

    c.execute(add_car, car)
    conn.commit()
    return "Entry Added!"
```

How cars are converted to strings separated by special characters. This function doesn't have and address as it's only accessed by other internal functions

```
#Converts a list to string, so app can read
#cars seperted by {, features seperated by |
def convert_list_to_string(list_of_cars):
    whole_cars_string = ''
    for car in list_of_cars:
        for feat in car:
            feat = str(feat)
            whole_cars_string += feat + '|'
        whole_cars_string += "{"

    print(whole_cars_string)
    return whole_cars_string
```

Android app

Code in an Android app to break down the strings back into separate cars

```
private void breakDownCarsStrings(String string_of_cars){
        Log.d("check string", string_of_cars);
        String[] each_car_as_string = string_of_cars.split("\\{");
        ArrayList<String> cars_seperated = new
ArrayList<String>(Arrays.asList(each_car_as_string));
        Log.d("Array", String.valueOf(cars_seperated));
```

```
    try {
        for (int i = 0; i < cars_seperated.size(); i++) {
            ArrayList<String> temp_car_list = new
ArrayList<String>(Arrays.asList(cars_seperated.get(i).split("\\|")));
            String estimatedValue = temp_car_list.get(temp_car_list.size() - 2);
            String priceDiffValue = temp_car_list.get(temp_car_list.size() - 1);
            String county = temp_car_list.get(temp_car_list.size() - 1);

            ArrayList<String> carToAdd = new
ArrayList<String>(temp_car_list.subList(0, 16));
            carToAdd.add(estimatedValue);
            carToAdd.add(priceDiffValue);

            cars.add(carToAdd);
        }
    }catch (ArrayIndexOutOfBoundsException e){
        e.printStackTrace();
    }

    for(int i = 0; i < cars.size(); i++){
        Log.d("Each Array", String.valueOf(cars.get(i)));
    }
}
```

Code to show Android app reading returned results

```
protected String doInBackground(String... params) {
    String flaskResult = " ";
    try {
        URL connectToFlask = new URL(params[0]);
        HttpURLConnection connection;
        connection = (HttpURLConnection) connectToFlask.openConnection();
        connection.connect();

        Scanner flaskScanner = new Scanner(connectToFlask.openStream());
        while(flaskScanner.hasNext())
            flaskResult += flaskScanner.next();
        connection.disconnect();
        flaskScanner.close();

        return flaskResult;
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Results

Testing Algorithm performance

To help bolster algorithm performance the algorithm contains a number of helper functions that can gauge the performance of the algorithm on a set of results. These include functions for gathering the mean squared error(MSE) and root mean squared error(RMSE). Functions for finding the variance of features and the correlation of dependent features with the price. Following results from earlier data.

Mean Squared Error(MSE)

Measures the average of the squares of the error.

```
Mean Squared Error:  3411820.75058
```

Root Mean Squared Error(RMSE)

Measures the difference between values predicted by the regression model

```
Root Mean Squared Error:  1847.11146133
```

Correlation

The correlation function displays how the a given feature correlate with the price. The closer the correlation was to 1 the higher the correlation. An example:

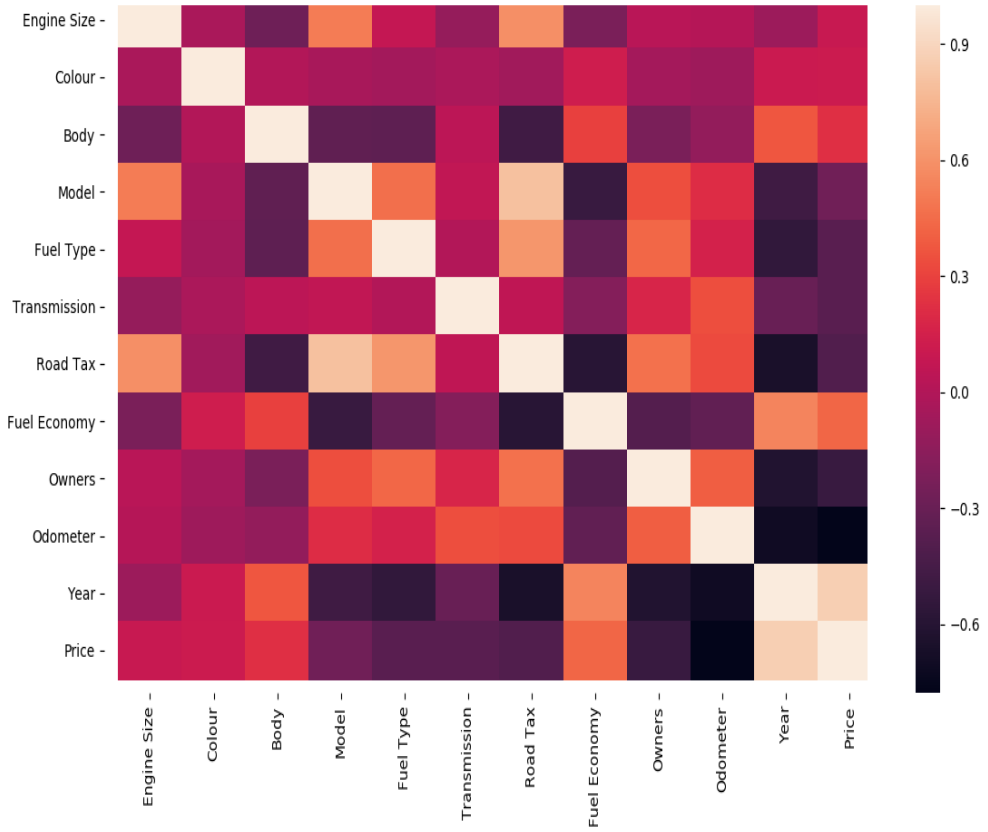
```
Show correlation
Engine Size     0.093688
Colour          0.113947
Body            0.226028
Model           0.265895
Fuel Type       0.363959
Transmission    0.365598
Road Tax        0.401269
Fuel Economy    0.427895
Owners          0.511646
Odometer        0.778768
Year            0.862673
Price           1.000000
```

From this we can gather that the odometer and year had the highest correlation with the price and engine size had the lowest.

Heatmap

To look for collinearity the algorithm script can also produce a correlation matrix heatmap of a dataset. It accomplishes this using the seaborn library.

Sample Heatmap



From this heatmap we can see that price and year have a very high positive correlation and that price and odometer have a high negative correlation. If two feature other than price shared a high correlation, then these could be removed to avoid duplicating information.

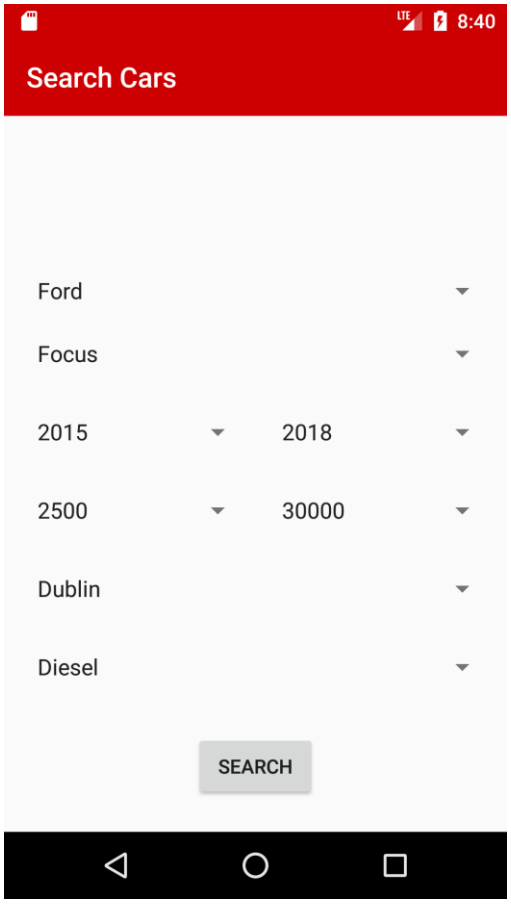
Variance

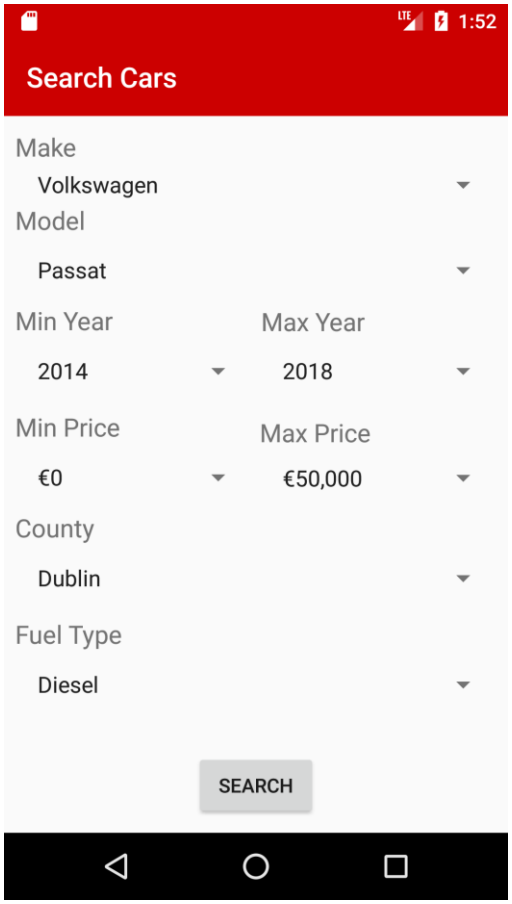
The script also contains a variance function to display the variance among features. If feature has a variance of 0 it will have negligible impact on the prediction. This is the case most of the time for the 'model' feature. However sometimes there will be multiple models for there are a few types of 3 series so for this reason model is included as a predictive feature.

User Testing

User testing on my front-end helped me notice problems with the application, some minor and some bigger and some that must have been in there for quite a while, flying under the radar. All users felt that the drop-down menus that let you pick car features could have been labelled more clearly so I addressed this problem first.

Below you can see some of the changes, the left is what the search previously looked like and the right is the same functionality with more direction.





I also added some details to give more instruction such as the list items now containing the text ‘more details’. I changed the odometer input from miles to km. It is still converted to miles to be used by the model. User testing let me find very minor inconsistencies such as not all car details being capitalised. Through user testing I also noticed a bug that would make my app crash that I was surprised I hadn’t noticed before. If users didn’t select a legitimate choice when valuing a car, the app would crash.

Problems solved

Datasets

I had originally planned on using the website carszone.ie for constructing my dataset, however their terms of use forbade scraping. I contacted them but got a response saying that I still could not scrape their data. With this I went looking for a site that would allow me access to data so I could build a dataset. I found CarsIreland.ie which has a very liberal terms of use policy. It does not forbid scraping and explicitly states that users have the right to copy the data.

Too much information

I was having trouble transferring massive amounts of car data from my MySQL database to the android phone. To solve this issue, I receive cars on a year by year. So, if a user chooses a range of cars between 2016 and 2018, I will first get all the cars from 2018, then 2017, then 2016. The knock-on effect from this is that cars are now sorted in descending order by year when a user goes to browse. This could be sorted easily by sorting the arraylist by some other value but I feel year is a good way of sorting the cars anyway.

Messing up Data

After I initially scraped I made a mistake which I did not notice immediately. Features of the cars in my dataset were being put in the wrong column. This is because sometimes when a value is missing it shows up as ‘Information not Available’ on the page, while other times it doesn’t show up at all. Rather than completely discard that data I decided to fix it. I wrote code that would find items in a row that were wrong and try and find the correct information in the row.

Future Work

Automate

I would like in the future to build a more automated system. A system that would periodically run each of the components without any input from me. I think this would be possible using python anywhere as it allows users to set up timed tasks. The system could scrape every few days, and then clean data automatically while still putting rows it can't clean to the side (maybe due to a new colour it can't generalise). It could constantly re-predict prices as it got more data allowing it to grow more accurate over time.

Cross Site

I would eventually like the data I can use to come from many different car sites. This would make a more beneficial product for the user. The spotter could find the best value cars from different sites and then direct users towards that site. This would be similar to how sites compare hotel prices or insurance. However, there may be a challenge getting datasets, as stated above.

Alerts

I had originally planned to have an alert feature but ended up not having the time to implement it. It would work by allowing users to enter alert criteria, which would have been similar to search criteria. The device would then periodically check the display database looking for matching cars, making note of the matching cars it finds so it doesn't alert the user about the same car twice.

Cross Platform

I would have like to build a front-end application that would run across multiple OS. This would mean re-developing the front-end using something like React Native as opposed the either Java for Android or Swift for iOS.