Data Exploration and Analysis of AutoTrader Car Listings Dataset

Christian Jordan - 14061768

13 January 2023

1 Introduction

The importance of data understanding and pre-processing is often understated, however in the reality of Data Science projects it can be the most time-consuming yet valuable investment that can be made. "Auto Trader is the UK and Ireland's largest digital automotive marketplace, bringing together the largest and most engaged consumer audience with the largest pool of vehicle sellers"

2 Pre-Requisites

Python is the primary programming language used in this project utilising the Jupyter Notebook package. Other packages used include NumPy, SciPy, Matplotlib, Seaborn and SkLearn. Not all code has been included in this document. You can find all the information and the full notebook needed to re-create this project on my GitHub using the link below. The data used cannot be redistributed.

https://github.com/christianmcb

3 Data Exploration and Understanding

The data provided for this project is from AutoTrader and consists of various features that may or may not relate to the listing price. The purpose of this project is to process and visualise the data to find the best predictors for the **price** of a vehicle.

3.1 Meaning and Type of Features

Figure 1 shows that the dataset is large and contains over 390,000 unique observations, with a total of 11 features. They may be self-explanatory, but for completeness I have described them below, some of the feature have been renamed for ease of use. Knowledge of the year suggests that this does not need to be a floating point number, and may function better as an integer so has been amended.

```
# Print the number of rows and columns in the dataset
print('The shape of the dataset is: {}'.format(cars.shape))
print('')

# Get feature information and data types
cars.info();

# Change the data type of 'year' to integer
cars['year'] = cars['year'].astype('Int64')
```

```
The shape of the dataset is: (393378, 11)

<class 'pandas.core.frame.DataFrame'>
Int64Index: 393378 entries, 202006039777689 to 201512149444029
Data columns (total 11 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   mileage                393254 non-null  float64
 1   reg_code               368286 non-null  object
 2   colour                 388242 non-null  object
 3   make                   393378 non-null  object
 4   model                  393378 non-null  object
 5   condition              393378 non-null  object
 6   year                   366851 non-null  float64
 7   price                  393378 non-null  int64
 8   body                   392560 non-null  object
 9   car_van                393378 non-null  bool
 10  fuel                   392820 non-null  object
dtypes: bool(1), float64(2), int64(1), object(7)
memory usage: 33.4+ MB
```

Figure 1: Obtain data types and size

1. mileage: The mileage on the listing date, continuous feature, floating point number.
2. reg_code: The registration plate number, categorical feature, text that relates closely to the year.
3. colour: The colour, categorical feature.
4. make: The manufacturer, categorical feature.
5. model: The model name, categorical feature.
6. condition: The vehicle condition, binary feature consisting of "NEW" and "USED".
7. year: The registration year, discrete feature, floating point number.
8. price: The listing price, continuous feature, integer value.
9. body: The body type, categorical feature.
10. : car_van: Boolean field, TRUE if van, FALSE if car.
11. : fuel: The fuel type (petrol, diesel, etc), categorical Feature.

public_reference	mileage	reg	colour	make	model	condition	year	price	type	car_van	fuel
202006300703207	7066.00000	67	Purple	Volkswagen	Caddy Life	USED	2017	17975	MPV	True	Petrol
202010275504469	2879.00000	20	Black	Nissan	Navara	USED	2020	29990	Pickup	True	Diesel
202010285515479	281500.00000	06	Silver	Volkswagen	Caravelle	USED	2006	7650	MPV	True	Diesel
202011015668810	151000.00000	59	Black	Volkswagen	Transporter Sportline	USED	2009	13000	Combi Van	True	Diesel
202005209437331	0.00000	NaN	Blue	SsangYong	Musso	NEW	¡NA¿	25631	Pickup	True	Diesel

Table 1: First five rows of dataset.

A first glance at the dataset in Table 1 shows data that is now well formatted and easy to understand. Things to note; the mixture of capitalised and non-capitalised observations within features. Also, missing values that are present in the "reg" and "year" columns.

	mileage	year	price
count	393254.00000	366851.00000	393378.00000
mean	38514.83862	2014.98779	17177.78633
std	34758.48800	7.97477	46827.96995
min	0.00000	999.00000	120.00000
25%	11510.00000	2013.00000	7450.00000
50%	29435.50000	2016.00000	12495.00000
75%	57630.00000	2018.00000	19990.00000
max	999999.00000	2020.00000	9999999.00000

Table 2: Decriptive statistics of numeric features.

3.2 Analysis of Distributions

3.2.1 Price Analysis

Moving onto analysing the univariate distributions of the features, start with the distribution of the "price" feature. From Table 2 price has a large range between **120.00** and **999,999.00** with mean **17177.79**, median **12495.00** and standard deviation of **34758.00**. Figure 2 consists of four plots, labelled A, B, C and D for reference, showing detailed distribution of the price feature. From the descriptive statistics and visualisations we can make the following observations:

- A and C show the large range confirmed by the descriptive statistics.
- Price feature is **positively** (or right), skewed; that is, most of the observations of price tend towards a lower price.
- Positive skew means that the median of **12495.00** is a better representation of central tendency than the mean.
- The outliers labelled by the box-plot in A, are any values outside of the scale of B (approximately 0-40,000).
- Large standard deviation implies a lot of variability in the price of vehicle listings.

3.2.2 Mileage Analysis

Again from Table 2, the range of mileage observations is between **0.00** and **999,999.00**, with mean **38,514.84** and median **29,435.00** and standard deviation **34758.49**. Similar to price distribution, it is represented in the four plots in Figure 3.

- The range is large shown in Plot A, with many outliers as labelled by the box-plot (1.5 x IQR).

- Positively skewed shown by C and D, as mileage increases the price decreases, therefore the median will be a better representation of central tendency.
- Again, large standard deviation compared with it's values implies a lot of variation in the mileage feature.
- Plot D includes the split between the binary feature "condition". This shows us that NEW cars have extremely low mileage which make up the jump in the histogram.

3.2.3 Year Analysis

The final numeric feature is year, of which there are some interesting observations to be made surrounding its statistics and distribution. From Table 2, the range is between **999** and **2020**, with mean **2014.99**, median **2016** and standard deviation **7.97**. Figure 4 shows the visual distribution of the year feature, and the following observations can be made:

- The range of year is from 999 to 2020. It is obvious there are some clear outliers or erroneous values in these observations since motor vehicles have not been around this long. The more likely range of values is probably closer to 1900 - 2020.
- Excluding outliers for the box-plot shows most observations lie between 2006 and 2020.
- Notice that vehicles with "condition" == "NEW" do not exist in the histogram, plot D. Therefore likely to be missing values.

3.2.4 Categorical Data Distributions

Viewing the distribution of categorical (or qualitative) data is as simple as the number of occurrences for each unique value, or the frequency distribution. Figure 5 shows frequency plots for each of the categorical features with the top 20 (if 20 exist) number of occurrences in descending order.

- **Colour** shows that most of our observations have one of the first 6 colours in the graph, probably making up over 90% of the dataset.
- **Make** however is much more uniform, with large frequencies across a variety of makes. The mode is BMW which makes up nearly 10% of the dataset.
- **Model** again is even more uniformly distributed, which can be expected with 1168 unique models in the dataset. With a completely uniform distribution, each model would be expected to have approximately 330 occurrences, which tells us the models in the visualisation are much more frequent.
- **Condition** is very in-balanced, with many more USED vehicles in the data than NEW ones. Nearly 94% of the vehicles are USED.
- **Type** is another feature with most of our observations made up of only a few vehicle types. Approximately 70% of the data is made up of either Hatchback or SUV.
- **Car_van** as may be expected, many more vehicle listings are cars than vans, therefore the in-balance is over 99.5% cars.
- **Fuel** again is made up mostly of Petrol or Diesel vehicles, and it might be interesting to look at these seperately in future analysis.

```
# Create figure for subplots
fig, ax = plt.subplots(4,1, figsize=(10,20), constrained_layout=True)

# View boxplot distribution for price
sns.boxplot(x='price', data=cars, ax=ax[0]);
ax[0].set_title('Box Plot Distribution of Price');

sns.boxplot(x='price', data=cars, ax=ax[1], showfliers=False);
ax[1].set_title('Box Plot Distribution without Outliers')

# View histogram distribution for price
sns.histplot(x='price', data=cars, ax=ax[2]);
ax[2].set_title('Histogram Distribution of Price');

# View histogram distribution for price under 150k split by condition
sns.histplot(x='price', data=cars.loc[cars['price'] <= 40000], bins=21, kde=True, ax=
    ax[3]);
ax[3].set_title('Histogram Distribution of price Without Outliers');
```

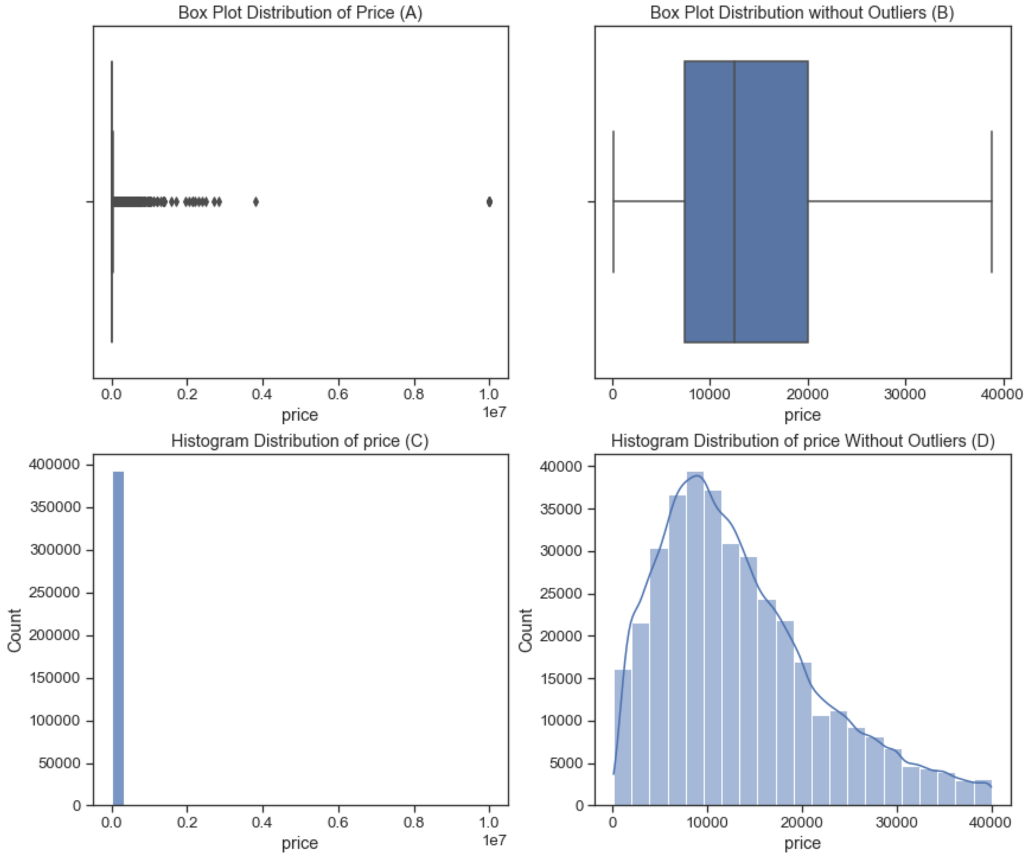

Figure 2: Visualisation of price distribution.

```python
# Create figure for subplots
fig, ax = plt.subplots(2,2, figsize=(12,12), constrained_layout=True)

# View boxplot distribution for mileage
sns.boxplot(x='mileage', data=cars, ax=ax[0,0]);
ax[0,0].set_title('Box Plot Distribution of Mileage (A)');

# View histplot distribution for mileage
sns.boxplot(x='mileage', data=cars, ax=ax[0,1], showfliers=False);
ax[0,1].set_title('Box Plot Distribution of Mileage without Outliers (B)');


# View histogram distribution for mileage
sns.histplot(x='mileage', data=cars, ax=ax[1,0], bins=30);
ax[1,0].set_title('Histogram Distribution of mileage (C)');

# View histplot distribution for subset of cars with mileage less than 200k,
# split by condition
sns.histplot(x='mileage', data=cars.loc[cars['mileage'] <= 125000], bins=40, hue='
                                        condition', multiple='stack', ax=ax[1,1]);
ax[1,1].set_title('Histogram Distribution Without Outliers (D)');
```

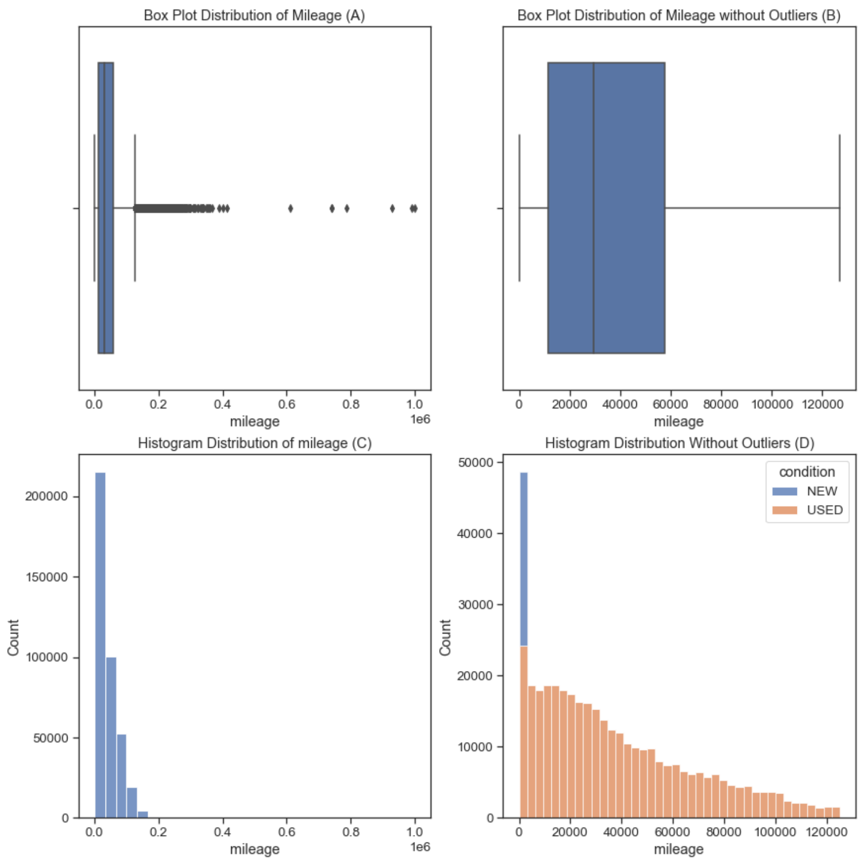

Figure 3: Visualisation of mileage distribution.

```python
# Create figure for subplots
fig, ax = plt.subplots(2,2, figsize=(12,12), constrained_layout=True)

# View boxplot distribution for year
sns.boxplot(x='year', data=cars, ax=ax[0,0]);
ax[0,0].set_title('Box Plot Distribution of Year (A)');

# View boxplot distribution without outliers
sns.boxplot(x='year', data=cars, showfliers=False, ax=ax[0,1]);
ax[0,1].set_title('Box Plot Distribution of Year Without Outliers (B)');

# View histogram distribution for year
sns.histplot(x='year', data=cars, ax=ax[1,0], bins=30);
ax[1,0].set_title('Histogram Distribution of year (C)');

# View histogram distribution for years above 2000 split by condition
sns.histplot(x='year', data=cars.loc[cars['year'] >= 2006], bins=15, hue='condition',
                                            multiple='stack', ax=ax[1,1]);
ax[1,1].set_title('Histogram Distribution Without Visual Outliers (D)');
```

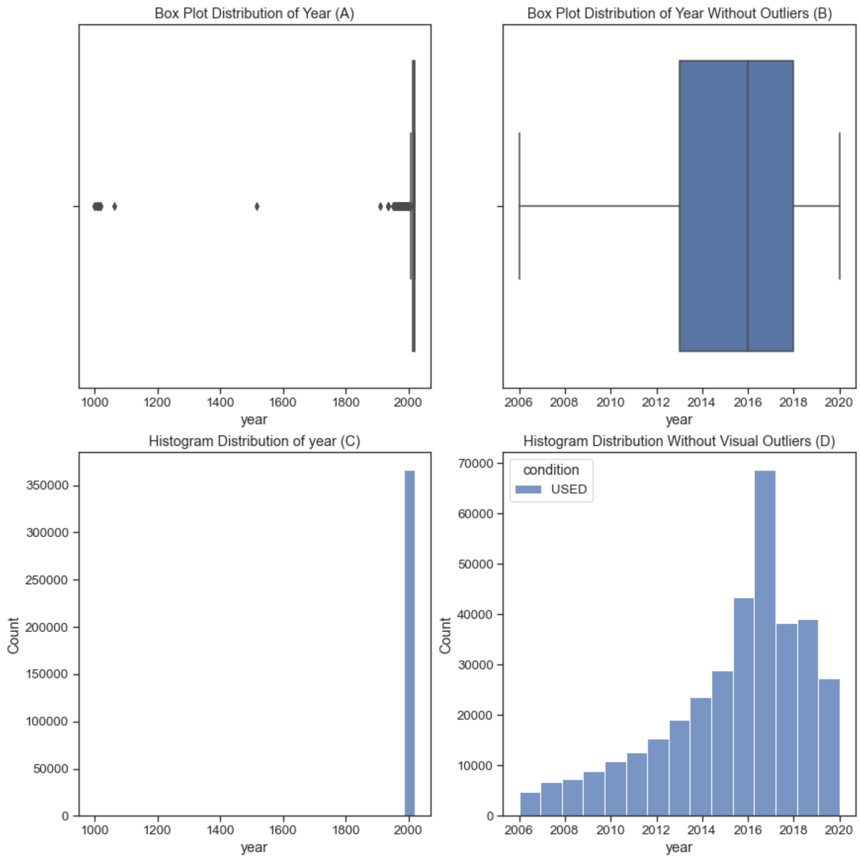

Figure 4: Visualisation of year distribution.

```python
# Create figure for subplots
fig, axs = plt.subplots(4, 2, figsize=(20,25), constrained_layout=True)

# Loop through categorical features and plot frequency distribution of values
for feature, ax in zip(features, axs.ravel()):
    cars[feature].value_counts().head(20).plot(kind='bar', ax=ax);

# Delete final plot as only 7 necessary.
fig.delaxes(axs[3,1])
```

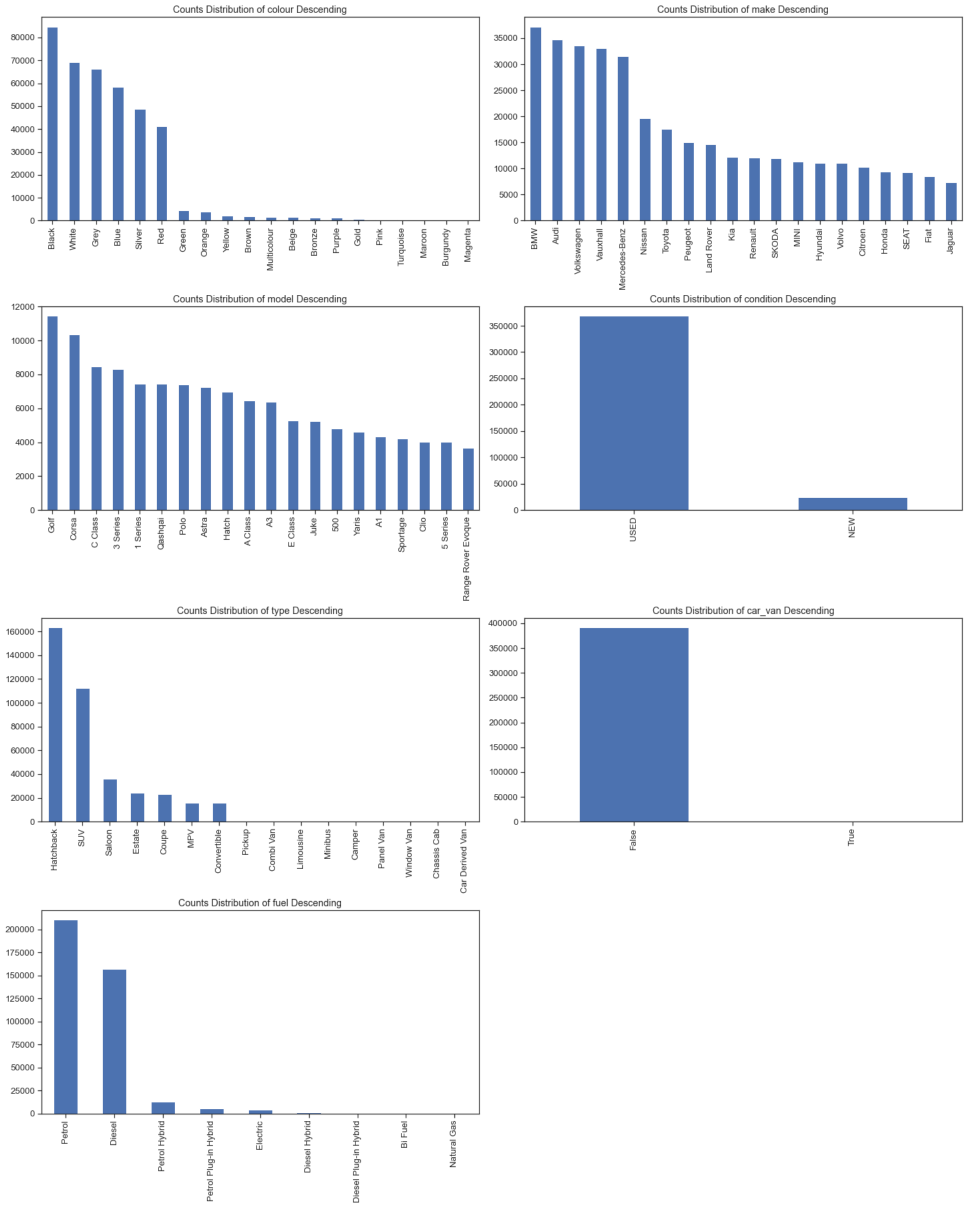

Figure 5: Distribution of Categorical Features

4 Data Pre-Processing

4.1 Noise, Outliers and Missing Values

Noisy data can render analysis useless if not dealt with in the correct manor, therefore it is important to explore and acknowledge this early in our analysis.

4.1.1 Missing Values

Now the distribution of data is known, it's key to deal with features that have missing data.

```
# Get number of missing values for each column
cars.isnull().sum()
```

	0
mileage	124
reg	25092
colour	5136
make	0
model	0
condition	0
year	26545
price	0
type	818
car_van	0
fuel	558

Table 3: Number of missing values per feature.

4.1.2 Year and Reg Missing Values

From Table 3 notice that year and reg features have a large number of missing values that is over 5% of the total observations. It is known from Figure 4 that only vehicles with condition = USED appear in the year distribution. Therefore, set vehicles with condition = NEW to the year 2021 and similarly reg to 21.

```
# Set 'year' and 'reg' to 2021 for 'NEW' cars
cars.loc[cars['condition'] == 'NEW', 'year'] = 2021
cars.loc[cars['condition'] == 'NEW', 'reg'] = 21

# Check missing values
cars[['year','reg']].isnull().sum()
```

	0
year	2058
reg	605

Table 4: Number of missing values in year and reg.

Following this correction, the remaining missing values can be input using our measure of central tendency so that it doesn't alter the overall distribution of the data. With heavily skewed data, the best measure of central tendency is the median, therefore input the remaining missing year values with the median value.

```
print('The median value of year is: {:2f}'.format(cars['year'].median()))

# Fill the missing values with the median of the feature
cars['year'] = cars['year'].fillna(cars['year'].median())
```

```
Output:
The median value of year is: 2017.000000
```

4.1.3 Colour Missing Values

Dealing with the missing values in the colour feature is a difficult task, and grouping the data by colour can provide some meaningful information. The below code creates a new feature to compare listings with colour and listings without colour specified.

```
# Take subset of cars with colour and price
cars_col = cars[['colour', 'price']].copy()

# Map values to new has_colour feature
cars_col['has_colour'] = cars['colour'].isnull().map({True:'No Colour',False:'Colour'
                                                     })

# Group values to see difference in mean and median values.
cars_col.groupby('has_colour').agg(['size','mean','median'])
```

ANOVA Difference in Means

Use ANOVA (analysis of variance) to check if there is a statistical difference in the means of the two groups, cars that have a colour and cars that do not have colour specified. Since the ANOVA test assumes normally distributed data, the data has been transformed using log 10 to achieve this.

```
# Transform data using the logarithm base 10
cars_col['price_'] = np.log10(cars_col['price'])

# Create subplots and plot data
fig, axs = plt.subplots(1,2, figsize=(14,6), constrained_layout=True)

sns.histplot(x='price_', data=cars_col, bins=20, ax=axs[0]);
sns.boxplot(x='price_', y='has_colour', data=cars_col, ax=axs[1]);
```

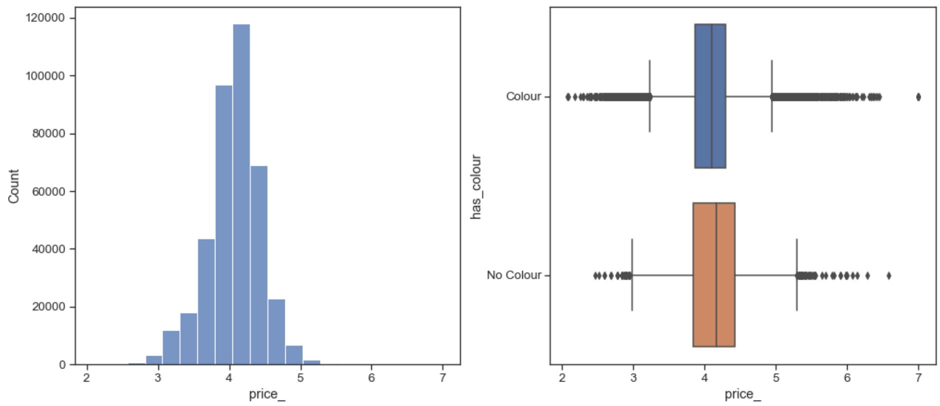

Figure 6: Transformed price data to logarithm base 10 split by colour.

Null Hypothesis: There is no difference in the means of the groups.
Alternate Hypothesis: There is a difference in the means of the groups.

```
# Run oneway ANOVA test on colour data
stats.f_oneway(cars_col[cars_col['has_colour'] == 'Colour']['price_'], cars_col[
                                            cars_col['has_colour'] != 'Colour']['
                                            price_'])
```

```
Output:
F_onewayResult(statistic=188.24114591303345, pvalue=7.873008428802948e-43)
```

Since the P-Value calculated by the one way ANOVA test is smaller than 0.05 (the statistical significance level of 95%), reject the null hypothesis. This means there is evidence to suggest that having no colour defined could have an impact on the price of a can. Therefore, keep missing values as seperate colour for the time being.

```
# Keep missing values in colour as 'Undefined'
cars['colour'] = cars['colour'].fillna('Undefined')
```

4.1.4 Outliers

Seeing the outliers labelled by the box-plots when visualising distributions shows that observations in the dataset lie very far from the central points of the features. However, when dealing with outliers it is important to recognise this variation in the data, and consider whether outliers are natural variation or incorrect values.

4.1.5 Year Outliers

One example of outliers in the dataset can be seen in Figure 4, Plot A, which shows a few data points with year prior to 1600. Table 5 shows the first five rows returned from the subset of data that has year prior to 1935. It is clear that a lot of these points are incorrect, however the Austin Seven on the first row can be considered a legitimate observation.

After some trial and error, find that vehicles listed prior to 1930 are all incorrect and can therefore have the year removed. This allows them to be dealt with as missing values by setting to np.nan.

```
# Create subset of observations with year less than 1935
inc_year = cars.loc[cars['year'] < 1935].copy()

print('Showing {} entries in the incorrect year dataset.'.format(len(inc_year)))
inc_year.head()
```

```
Output:
Showing 21 entries in the incorrect year dataset.
```

mileage	reg	colour	make	model	condition	year	price	type	car_van	fuel
26000.00000	NaN	Black	Austin	Seven	USED	1933.00000	9995	Saloon	False	Petrol
14000.00000	07	Blue	Toyota	Prius	USED	1007.00000	7000	Hatchback	False	Petrol Hybrid
96659.00000	65	Black	Audi	A4 Avant	USED	1515.00000	10385	Estate	False	Diesel
37771.00000	63	Black	Smart	fortwo	USED	1063.00000	4785	Coupe	False	Petrol
30000.00000	59	Red	Toyota	AYGO	USED	1009.00000	4695	Hatchback	False	Petrol

Table 5: First five rows with year prior to 1935

```
# Set observations with year less than 1930 to null so we can deal with them
                                    alongside null values.
cars.loc[cars['year'] < 1930, 'year'] = np.nan
```

4.1.6 Outlier Detection

There is a variety of different algorithms to detect outliers in data, many of which rely on the data being normally distributed. In the numeric features in this dataset, the data is skewed and therefore many of the assumptions cannot be applied without transforming the data.

One common method for removing outliers that can be robust to skewed data is using the lower quartile, upper quartile and the inter-quartile range to find upper and lower bounds. Performing the analysis on this dataset gave unsatisfactory results, removing almost 16% of observations that could be key to the analysis.

```
def remove_outliers(df,columns, IQR_mult):
    # Loop through specified columns
    for col in columns:
        print('Working on column: {}'.format(col))

        # Get lower quartile, upper quartile and inter-quartile range
        q1, q3 = np.percentile(df[col], [25, 75])
        IQR = q3 - q1

        # Set upper and lower bounds
        upper = q3 + (IQR * IQR_mult)
        lower = q1 - (IQR * IQR_mult)
        print('Upper bound: {}, Lower Bound: {}'.format(upper, lower))

        #
        df = df[(df[col] < upper) & (df[col] > lower)]
    return df

data = remove_outliers(cars, ['year','mileage','price'], 1.5)

# Get proportion of removed data
print('Proportion of removed data: ', len(data)/len(cars))
```

Code a re-work of the function supplied by Peter Grant (2012).

```
Output:
Working on column: year
Upper bound: 2024.0, Lower Bound: 2008.0
Working on column: mileage
Upper bound: 109842.375, Lower Bound: -49542.625
Working on column: price
Upper bound: 39726.5, Lower Bound: -10237.5

Proportion of data remaining: 0.8411934576920926
```

Observing some of the outliers labelled by this method it is clear that they are legitimate observations and key to the variation in this dataset. Therefore they are clearly not outliers at all.

A more proactive way of managing outliers is to label any extreme values within the data so that the analysis can be run with and without these values. Choosing to label the top percentile 0.5% of values, the outliers are limited to only the most extreme values in this dataset.

```python
def label_outliers(df,columns):
    # Create outlier column and set values to False
    df['outlier'] = False

    # Loop through specified columns
    for col in columns:
        print('Working on column: {}'.format(col))

        # Set bounds of lower and upper 0.5%
        lower, upper = np.percentile(df[col], [0.5, 99.5])

        print('Upper bound: {}, Lower Bound: {}'.format(upper, lower))

        # Print the number of outliers
        print('Number of values labelled: ', len(df.loc[(df[col] > upper) | (df[col]
                                               < lower)]))

        # Set outlier column to true if outlier
        df.loc[(df[col] > upper) | (df[col] < lower), 'outlier'] = True

    return df

# Run function to label outliers and assign to cars dataframe
cars = label_outliers(cars, ['year','mileage', 'price'])

# Get proportion of outliers
cars['outlier'].value_counts(normalize=True)
```

```
Output:
Working on column: year
Upper bound: 2021.0, Lower Bound: 1999.0
Number of values labelled:  1772
Working on column: mileage
Upper bound: 158000.0, Lower Bound: 0.0
Number of values labelled:  1953
Working on column: price
Upper bound: 122990.0, Lower Bound: 899.0
Number of values labelled:  3923
[186]:
False    0.98167
True     0.01833
Name: outlier, dtype: float64
```

Visualising the outliers in a pair-plot grid helps see how the data has been affected with regards to the extreme values. It is clear to see in Figure 7 that only a small portion of the data have been labelled and the central clusters remain unaffected.

```python
# Visualise numeric data split by outlier
sns.pairplot(data=cars[['price', 'mileage', 'year','outlier']].sample(10000,
                                                        random_state=0),
             hue='outlier', height=4);
```

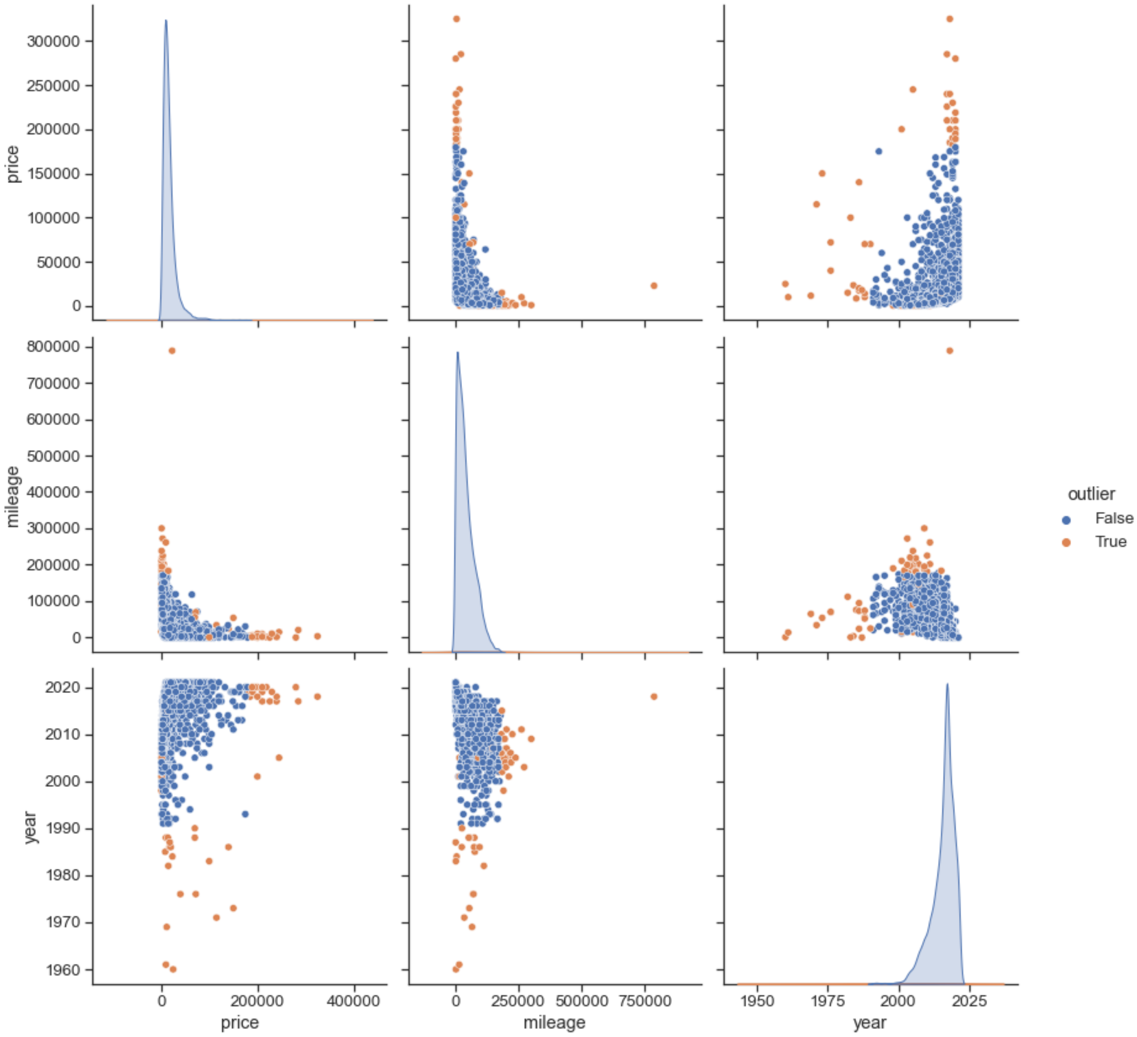

Figure 7: Pair-plot of numeric data split by outlier labels.

4.1.7 Noisy Registrations

Looking closely at the registration data also reveals noisy data, example, the registrations that occur less than 5 times in the data are printed below. See some of the registrations are in lower case, where others are solely in upper case.

```python
# Get count of registrations that appear less than 5 times
```

```
reg_count = cars['reg'].value_counts()
print(reg_count.loc[reg_count < 5].index)
```

```
Output:
Index(['95', '94', '37', 'k', '38', 's', 'CA', '723xuu', 'FW', 'm', '85', 'p'], dtype
='object')
```

This can quickly be rectified once confirmed it is true, so check registrations that contain only letters, and consequently change all registrations to uppercase only.

```
# Transform and transpose into dataframe
pd.DataFrame(
    # Get frequency table of registrations that contain alphabetic only.
    cars[cars['reg'].str.isalpha().fillna(False)]['reg'].value_counts()
).transpose()
```

	Y	X	W	R	P	T	V	S	N	M	L	H	G	J	K	F	E	D	C	B	A	CA	m	FW	s	k	p
reg	348	271	248	241	206	173	154	151	144	125	118	114	106	101	88	81	66	48	47	43	28	1	1	1	1	1	1

Table 6: Frequency table of alphabetic data in registration feature.

```
# Convert registration to upper case
cars['reg'] = cars['reg'].str.upper()
```

In fact, the registration feature doesn't seem to have much order at all, and may all be considered noisy data later in this project. Think about how registration and year are connected, the registration code is a representation of what year actually is, however year is already coded in a numeric fashion, which gives order and therefore more value.

4.2 Feature Engineering

4.2.1 Binning

One variation of feature engineering for numeric data is to split the data into categorical bins. For example, in the following code the mileage feature is split into 5 equal bins containing the bottom 20% of data all the way to the top 20% of data.

```
# Split mileage data into 5 equally spaced bins.
mileage_labels = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
cars['mileage_bins'] = pd.qcut(cars['mileage'], q=[0, 0.2, 0.4, 0.6, 0.8, 1], labels=
                                    mileage_labels)

# Get size, min and max of the bin groupings
cars.groupby('mileage_bins')['mileage'].agg(['size','min','max'])
```

mileage_bins	size	min	max
Very Low	78679	0.00000	8211.00000
Low	78675	8212.00000	21702.00000
Medium	78673	21703.00000	38464.00000
High	79068	38465.00000	66000.00000
Very High	78283	66002.00000	999999.00000

Table 7: Size, min and max of mileage bin groupings.

Similar completion of binning has been completed for the year and price columns for analysis later in this project.

4.3 Transformations

4.3.1 Logarithmic Transformations

Often times logarithmic transformations of positively skewed data can make it appear more normally distributed and have a positive effect on the correlation between features. Figure 8 shows the transformation on the price feature now following a more normal distribution. It achieves this by shrinking larger values more than smaller values.

```
# Transform price using log base 10
cars['price_'] = np.log10(cars['price'])

# Configure subplots
fig, axs = plt.subplots(1,2, figsize=(14, 8))

# Create histogram of price distribution
sns.histplot(x='price', data=cars, bins=20, ax=axs[0]);
axs[0].set_title('Histogram of Price Distribution (A)');

# Create histogram of transformed price distribution
sns.histplot(x='price_', data=cars, bins=20, ax=axs[1]);
axs[1].set_title('Histogram of Log Transformed Price Distribution (B)');
```

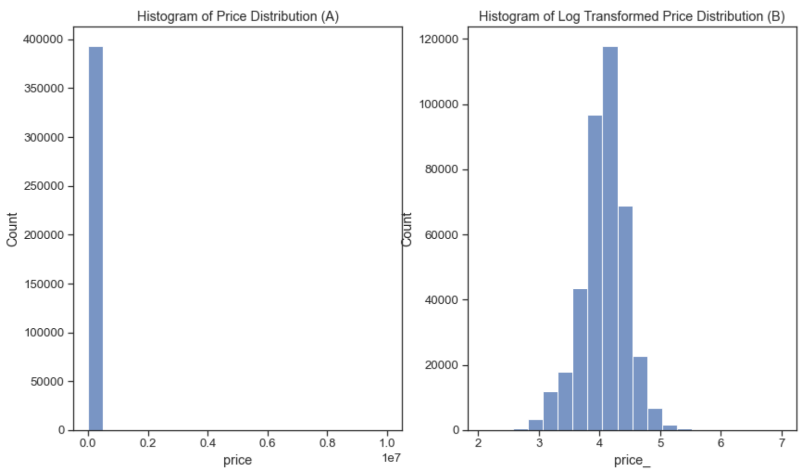

Figure 8: Distribution of price and log transformed price.

4.3.2 Square Root Transformations

Similar to the logarithmic transformation, the square root transformation can change the distribution of values by shrinking large values. This is applied primarily to negatively skewed features, in this case mileage.

```
# Transform mileage using square root
cars['mileage_'] = np.sqrt(cars['mileage'])

# Configure subplots
fig, axs = plt.subplots(1,2, figsize=(14, 8))

# Create histogram of mileage distribution
sns.histplot(x='mileage', data=cars, bins=20, ax=axs[0]);
axs[0].set_title('Histogram of Mileage Distribution (A)');

# Create histogram of transformed mileage distribution
sns.histplot(x='mileage_', data=cars, bins=20, ax=axs[1]);
axs[1].set_title('Histogram of Square Root Transformed Mileage Distribution (B)');
```

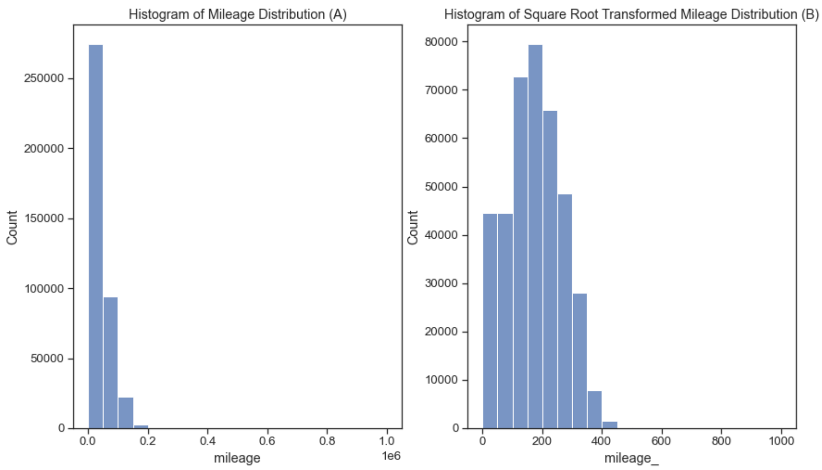

Figure 9: Distribution of mileage and square root transformed mileage.

4.3.3 Condition Transformation

Condition is a binary feature consisting of "NEW" and "USED" vehicle listings. This can easily be transformed into a numeric feature consisting of True or False based on if a vehicle is new or not. In Python, True and False correspond to 1 and 0 respectively, creating a numeric feature to be analysed.

```
# Create 'new' column to consist of True and False (1 and 0)
cars['new'] = cars['condition'] == 'NEW'

# Get counts of new and used
cars['new'].value_counts()
```

```
Output:
False     368891
True       24487
Name: new, dtype: int64
```

4.4 Sub-setting

4.4.1 Feature Selection - Make

Creating small subsets of data using specific makes of vehicle it becomes easier to visualise the relationship between variables.

First, find two commonly occurring makes in the data with variety in the group means.

```
# Groupby function to show top 5 most frequently occurring makes
cars.groupby('make')['price'].agg(['size', 'mean']).sort_values(by='size', ascending=
                                             False)[0:5]
```

make	size	mean
BMW	37228	19825.16751
Audi	34752	20193.59361
Volkswagen	33606	13853.26968
Vauxhall	33084	8032.80758
Mercedes-Benz	31591	21415.24903

Table 8: Top 5 results of grouped make data with mean of price.

```
# Create subset of vauxhall data
vauxhall = cars.loc[(cars['make'] == 'Vauxhall')]

# Create subset of Mercedes data
mercedes = cars.loc[(cars['make'] == 'Mercedes-Benz')]

# Vauxhall correlation with price
print(vauxhall.corr()[['price','price_']])

# Mercedes correlation with price
print(mercedes.corr()[['price', 'price_']])
```

Vauxhall	price	price_
mileage	-0.70710	-0.79666
year	0.79000	0.90684
price	1.00000	0.90311
car_van	0.03190	0.01712
outlier	NaN	NaN
age	-0.79000	-0.90684
new	0.49321	0.29276
age_	-0.87067	-0.86274
price_	0.90311	1.00000
mileage_	-0.77986	-0.79558

Mercedes-Benz	price	price_
mileage	-0.62182	-0.75437
year	0.62735	0.82654
price	1.00000	0.87310
car_van	0.08422	0.07056
outlier	NaN	NaN
age	-0.62735	-0.82654
new	0.32525	0.24838
age_	-0.69441	-0.78532
price_	0.87310	1.00000
mileage_	-0.68453	-0.75580

Table 9: Correlation between Vauxhall, Mercedes-Benz and price.

From Table 8, create subsets of data for Vauxhall and Mercedes-Benz as the means have the largest difference. Table 9 below shows the correlation between all numeric variables with price and the log transformed price.

In tables for both makes, the correlation is considerably higher with the log transformed price in year and mileage.

Now confident that make is a good predictor of the price of the vehicle, with 110 unique makes, reduce the cardinality by analysing the means of groups. Finding the makes that have less than 5,000 observations counts 82 unique values. Grouping these by the mean value of the group allows us to reduce the cardinality while preserving some of the information within price groups. The code below allows us to reduce the number of makes from 110 to only 26, with 5 added groups based on mean price of the make.

```
# View grouped means of makes and size
makes = cars.groupby('make')['price'].agg(['size','mean','median'])

# Check groups with less than 5000 observations
makes.loc[makes['size'] < 5000]

# Transform makes with less than 5000 observations to groups
makes_transform = pd.qcut(makes.loc[makes['size'] < 5000]['mean'],
                          q=[0, 0.2, 0.4, 0.6, 0.8, 1], labels=['Very Low Range', '
                                                                  Low Range', 'Mid
                                                                   Range', 'High
                                                                  End', 'Very High
                                                                   End'])

# Transform new column to new groups
cars['make_new'] = cars['make'].map(makes_transform.astype('str'))

# Copy make to the new column for makes > 5000 observations
cars['make_new'] = cars['make_new'].fillna(cars['make'])

# Check group means again
cars.groupby('make_new')['price'].agg(['size','mean','median']).sort_values(by='
                                          median', ascending=False)
```

make_new	size	mean	median
Very High End	8293	96878.05836	59950.00000
High End	731	46643.60876	46995.00000
Land Rover	14595	35312.22275	29970.00000
Jaguar	7393	26031.53997	21950.00000
Volvo	11067	24138.40643	20000.00000
Mid Range	7254	20905.62696	18762.50000
Mercedes-Benz	31591	21415.24903	18138.00000
BMW	37228	19825.16751	16600.00000
Audi	34752	20193.59361	16450.00000
Volkswagen	33606	13853.26968	12499.00000
SEAT	9251	12616.10756	11999.00000
SKODA	11902	13428.01647	11990.00000
Mazda	6791	12956.51362	11980.00000
Kia	12195	12514.51857	11700.00000
MINI	11321	12308.48238	11495.00000
Low Range	15965	12524.27961	10990.00000
Nissan	19688	11326.38668	10495.00000
Hyundai	11128	11595.49676	10299.00000
Honda	9381	11002.24027	9999.00000
Toyota	17541	11344.51314	9618.00000
Renault	12043	10022.93158	9195.00000
Peugeot	15026	9859.08645	8000.00000
Vauxhall	33084	8032.80758	7500.00000
Citroen	10309	8177.14803	7450.00000
Fiat	8535	7829.40187	6975.00000
Very Low Range	2707	5518.33986	4295.00000

Table 10: Grouped price of new makes feature with reduced cardinality.

Table 10 then shows the newly grouped mean price of makes, which keeps the highest mean vehicle makes in top bands, meaning even with the small sizes, the information can be kept in case it is valuable to modelling.

4.4.2 Feature Selection - Colour

With categorical features, it can be more difficult to make statistical inferences when the target variable is continuous. However, we discussed ANOVA (analysis of variance) earlier in 4.1.3 as one technique for analysing the difference in grouped means.

Before using this again, colour has a high cardinality of 22 unique variables, and for visualisation and modelling purposes, this should be reduced as necessary in line with computing power. As discussed in 3.2.4, Categorical Data Distributions, color is made up primarily of only 6 colours, nearly 94% of the total data.

```
# Proportion of data made up by first six colours
print(cars['colour'].value_counts(normalize=True)[0:6])
cars['colour'].value_counts(normalize=True)[0:6].sum()
```

```
Output:
Black     0.21548
White     0.17554
Grey      0.16841
Blue      0.14859
Silver    0.12413
Red       0.10494
Name: colour, dtype: float64
Total proportion: 0.9370885001194779
```

Then, keeping the first 6 colours, reduce cardinality by changing the other colours to be the same string instance "Other".

```
# Get colour grouped by frequency
colour = cars.groupby('colour')['price'].size().sort_values(ascending=False)
colours = colour.index[0:6]

# Set remaining colours to 'Other'
cars.loc[~cars['colour'].isin(colours), 'colour'] = 'Other'

# Out put new means of groups
cars.groupby('colour')['price'].mean()
```

```
Output:
colour
Black     18517.86068
Blue      16717.40495
Grey      19856.93695
Other     19067.87219
Red       15052.04915
Silver    13314.39011
White     16677.45280
Name: price, dtype: float64
```

Now it's time to see if the data contained in colour provides meaningful insight into the price of the vehicle, using one-way ANOVA. Remember that the target variable needs to follow a normal-distribution, therefore use "price_".

Null Hypothesis: There is no difference in the means of the groups. Alternate Hypothesis: There is a difference in the means of the groups.

```
# Create empty list
colours_data = []

# Loop through unique colour variables and append individual dataframes containing
                                            price_ dataframe
for i in cars['colour'].unique():
     colours_data.append(cars.loc[cars['colour'] == i][['price_']])

f_value, p_value = stats.f_oneway(colours_data[0], colours_data[1], colours_data[2],
                                  colours_data[3], colours_data[4], colours_data[5],
                                  colours_data[6])

print('F-value: {}, P-Value: {:2f}'.format(f_value[0], p_value[0]))
```

```
Output:
F-value: 2275.5970882636866, P-Value: 0.000000
```

Since P-Value is smaller than 0.05, reject the null hypothesis at the 95% confidence level that there is no difference in means. Building on this is the Tukey Algorithm, which compares individual groups with each other individual. Running this algorithm to compare group means, find the results in Table 11. From this, see that all groups reject the null hypothesis except the pair "Black-White".

This suggests there is significant difference in the means and can be selected for modelling.

```
# Import Tukey Algorithm
from statsmodels.stats.multicomp import pairwise_tukeyhsd

print(pairwise_tukeyhsd(endog=cars['price_'], groups=cars['colour'], alpha=0.05))
```

```
Output:
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
 Black   Blue  -0.0511  0.001 -0.0569 -0.0453   True
 Black   Grey   0.0462  0.001  0.0406  0.0518   True
 Black  Other  -0.0623  0.001 -0.0701 -0.0545   True
 Black    Red  -0.1001  0.001 -0.1066 -0.0936   True
 Black Silver   -0.176  0.001 -0.1821 -0.1698   True
 Black  White   0.0054   0.06 -0.0001   0.011  False
  Blue   Grey   0.0973  0.001  0.0912  0.1034   True
  Blue  Other  -0.0112 0.0011 -0.0194  -0.003   True
  Blue    Red   -0.049  0.001 -0.0559 -0.0421   True
  Blue Silver  -0.1249  0.001 -0.1315 -0.1182   True
  Blue  White   0.0565  0.001  0.0505  0.0626   True
  Grey  Other  -0.1085  0.001 -0.1166 -0.1005   True
  Grey    Red  -0.1463  0.001 -0.1531 -0.1395   True
  Grey Silver  -0.2222  0.001 -0.2286 -0.2157   True
  Grey  White  -0.0408  0.001 -0.0467 -0.0349   True
 Other    Red  -0.0378  0.001 -0.0465 -0.0291   True
 Other Silver  -0.1137  0.001 -0.1221 -0.1052   True
 Other  White   0.0677  0.001  0.0597  0.0757   True
   Red Silver  -0.0759  0.001 -0.0831 -0.0686   True
   Red  White   0.1055  0.001  0.0988  0.1123   True
Silver  White   0.1814  0.001   0.175  0.1878   True
-----------------------------------------------------
```

Table 11: Results of Tukey group difference in means.

4.4.3 Data Sampling

Bootstrapping is the method of repeatedly taking samples from the dataset and comparing summary statistics to gain confidence in the mean of the data.

In this example looking at "NEW" vehicles and testing the suitability of the mean value of price **34584.73**. Figure 10 shows the full code, as provided by Luciano Gerber, to obtaining the bootstrap samples and mean boundaries based on the 95% confidence level. This means we can be 95% confident that the real mean of price being between the values of **34334.367** and **34838.192**.

```
# Print results
print('Mean of original data: {}'.format(np.mean(original_data)))
print('Results at 95% confidence interval')
print('[Lower Bound, Upper Bound]: {}'.format(np.percentile(bootstrap_means, [2.5, 97
                                           .5])))
```

```
Output:
Mean of original data: 34584.730836770534
Results at 95% confidence interval
[Lower Bound, Upper Bound]: [34334.36700903 34838.19179667]
```

```python
# Set random seed for sampling
rng = np.random.default_rng(seed=0)

# Subset data for new cars
new_price = np.array(cars.loc[cars['condition'] == 'NEW','price'])

# Copy of new_price
original_data = new_price

# Define number of samples, define numpy array for samples
num_samples = 10000
bootstrap_samples = np.empty((num_samples, len(original_data)))

# Loop through samples array and choose random samples
for i in range(0, num_samples):
    bootstrap_samples[i] = rng.choice(original_data, len(original_data))

# Plot means of the bootstrap samples
fig, ax = plt.subplots(2,1, figsize=(12,5))
sns.stripplot(x=bootstrap_means, jitter=0.2, alpha=0.5, ax=ax[0]);

# Plot confidence intervals on top of histogram
sns.histplot(x=bootstrap_means, stat="proportion", ax=ax[1])
sns.rugplot(x=bootstrap_means, ax=ax[1], height=0.05, color="aqua", linewidth=0.5);

ax[1].axvline(bootstrap_means.mean(), color="lightsalmon", linewidth=0.5)
ax[1].axvline(np.percentile(bootstrap_means, 2.5), color="fuchsia", linewidth=0.5)
ax[1].axvline(np.percentile(bootstrap_means, 50), color="fuchsia", linewidth=0.5)
ax[1].axvline(np.percentile(bootstrap_means, 97.5), color="fuchsia", linewidth=0.5);
ax[1].set_xlabel('Bootstramp Mean Value')
```

Code provided by Gerber, L (November 2022)

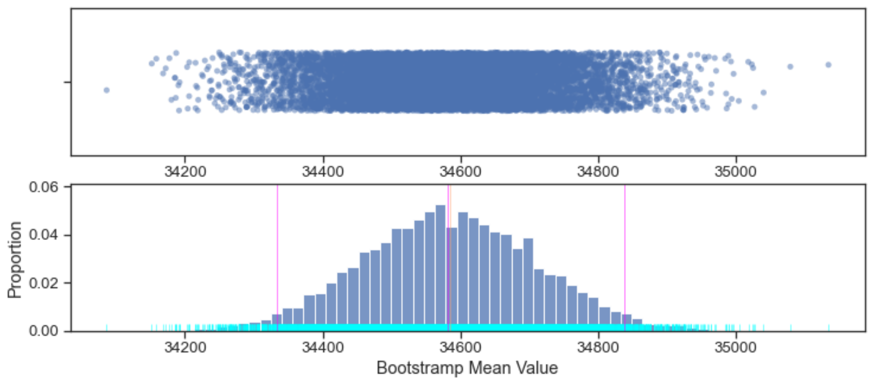

Figure 10: Bootstrapping sampling for gaining confidence in mean of new vehicles.

From the histogram in Figure 10, it is obvious that most of the means of our samples lie between these values and we can say the mean value is fairly accurate.

5 Association and Group Differences Analysis

5.1 Quantitative-Quantitative

5.1.1 Correlation

Everything completed so far would be worthless if some of the changes do not create a positive effect on the relationship of features with price, which is the main aim of this project. As a basis for evaluation, here is a heat-map of correlation values between variables for the selected features in the dataset for comparison.

```
# Initiate new dataframe for visualising relationships
features = ['mileage_', 'colour', 'makes_group', 'new', 'age_', 'price_', 'type', '
                                                type', 'car_van', 'fuel']
cars_vis = cars[features].copy()

# Create heatmap of selected features
sns.heatmap(cars_vis.corr(), annot=True, cmap='Blues');
```

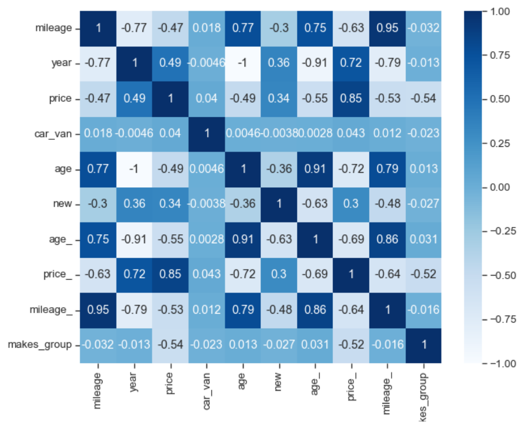

Figure 11: Annotated correlation heatmap between numeric features.

There is high correlation between the log transformed price and a number of features such as year and mileage which could benefit any modelling based from this pre-processing.

5.1.2 Makes, Year and Price

Visualising quantitative features proves difficult when the relationships depend so much on the categorical counterparts, therefore to visualise the strong relationship between year and price, subset the data.

Building on the analysis of correlation completed in section 4.4.1 for make, return to the subsets of data for Vauxhall and Mercedes. Outliers have been excluded for this to help visualise relationships.

```
# Create subset of vauxhall data
vauxhall = cars.loc[(cars['make'] == 'Vauxhall') & (cars['outlier'] == False)]
```

```
# Create subset of Mercedes data
mercedes = cars.loc[(cars['make'] == 'Mercedes-Benz') & (cars['outlier'] == False)]

# Combine the two for later plots
makes = mercedes.append(vauxhall)
```

```
# Create figure for subplots
fig, axs = plt.subplots(2,2, figsize=(12,12), constrained_layout=True)

# Creat boxplot for vauxhall data
sns.boxplot(x='year_bins', y='price', data=vauxhall, showfliers=False, ax=axs[0,0]);
axs[0,0].set_title('Vauxhall - Year Bins Against Price (A)');
axs[0,0].set_ylim(0, 75000);

# Create boxplot for mercedes data
sns.boxplot(x='year_bins', y='price', data=mercedes, showfliers=False, ax=axs[0,1]);
axs[0,1].set_title('Mercedes - Year Bins Against Price (B)');
axs[0,1].set_ylim(0, 75000);

# Creat regression plot for vauxhall data
sns.regplot(x='year', y='price', data=vauxhall.sample(5000, random_state=0), ax=axs[1
                                                    ,0], scatter_kws={'alpha': 0.2});
axs[1,0].set_title('Vauxhall - Year Against Price (C)');
axs[1,0].set_ylim(-10000,120000);


# Create regression plot for mercedes data
sns.regplot(x='year', y='price', data=mercedes.sample(5000, random_state=0), ax=axs[1
                                                    ,1], scatter_kws={'alpha': 0.2});
axs[1,1].set_title('Mercedes - Year Against Price (D)');
axs[1,1].set_ylim(-10000, 120000);
```

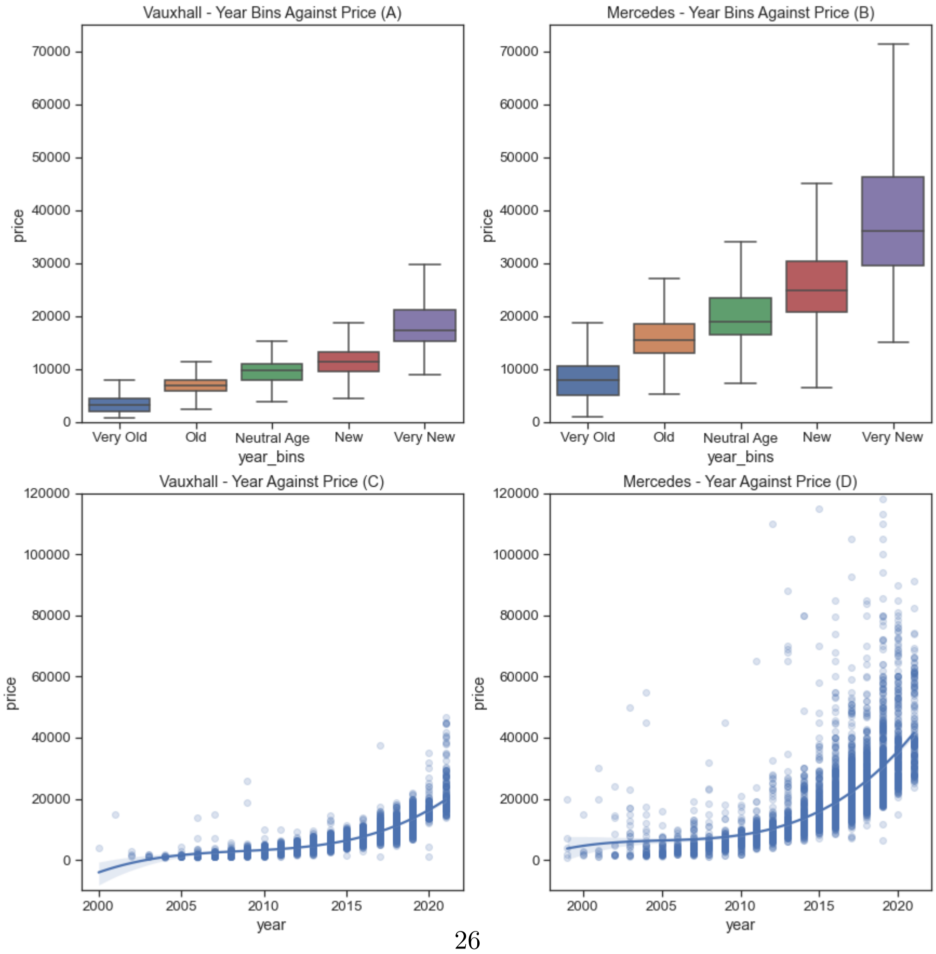

Figure 12: Box and scatter plots of year against price.

Figure 12 shows the relationship between year and price when sub-setting specific makes of vehicle, in this case, Vauxhall and Mercedes. The relationship becomes much clearer in both the box-plot and the scatter-plot for individual makes. The order of the plot has been increased to 3 as the relationship does not appear linear, but rather polynomial fit best by a cubic fit. To interpret this, as vehicles age, the depreciation slows.

However, analysing the relationship of the same models with the log transformed price provides a much more linear relationship as shown in Figure 13.

```
# Create regression plot split by makes
g = sns.lmplot(x='year', y='price_', col='make', hue='make',
           data=makes.sample(10000, random_state=0), scatter_kws={'alpha': 0.05},
           order=1, height=6);
```

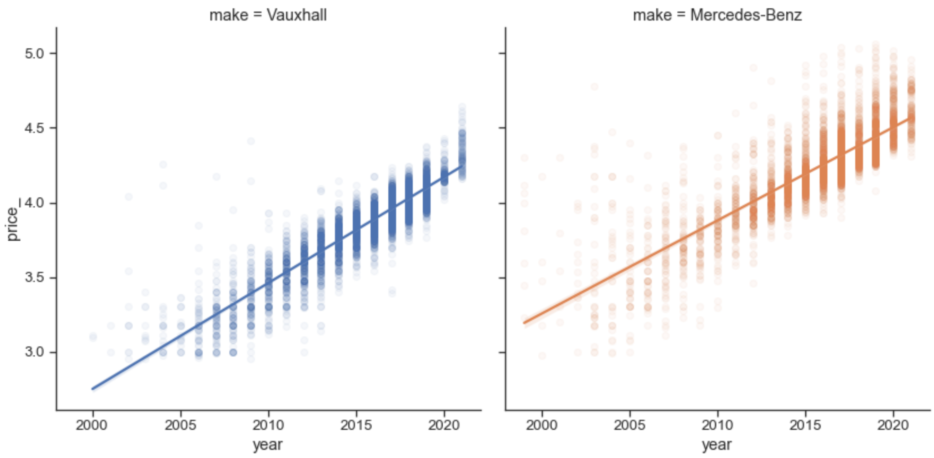

Figure 13: Regression plots of year against log transformed price for makes.

5.2 Quantitative-Categorical

5.2.1 Condition and Price

The relationship between condition and price may seem trivial, where new vehicles should be higher in price, however, trying to put this into perspective visually can be beneficial.

Figure 14 shows the difference in ranges and means of the groups of data within the condition feature. There is a clear bias of data towards the "USED" subset, with many more points labelled in the strip-plot from the sample of 5000. Even so, the ranges and means have a clear visual difference.

The kernel density plot visualises the proportion of data that falls in price ranges, similar to a histogram but not as sensitive to the difference in the frequencies of "NEW" and "USED" vehicles. Notice that there is hardly any "NEW" vehicles in the first portion of the graph, where prices begin at the minimum 7815. This backs up the foundations of the knowledge that "NEW" vehicles have a higher price and also that there is a much higher density of vehicles in the higher price ranges of larger than 40,000.

This could help businesses to value vehicles when buying or selling and consequently prevent major losses on used vehicles.

```
# Create figure for subplots
fig, axs = plt.subplots(1,2, figsize=(15,6), constrained_layout=True);

# Create boxplot and overlay stripplot of condition vs price
sns.boxplot(x='price', y='condition', data=cars_vis, palette='Blues', ax=axs[0]);
sns.stripplot(x='price', y='condition', data=cars_vis.sample(5000), alpha=0.5,
                                          palette='Reds', ax=axs[0]);
axs[0].set_title('Strip-Plot Overlay on Box-Plot of Condition vs Price')

# Create density plots of price by condition
sns.kdeplot(x='price', data=new, ax=axs[1]);
sns.kdeplot(x='price', data=used, ax=axs[1]);
axs[1].legend(['NEW','USED']);
axs[1].set_title('Density Plot of Condition vs Price')
```

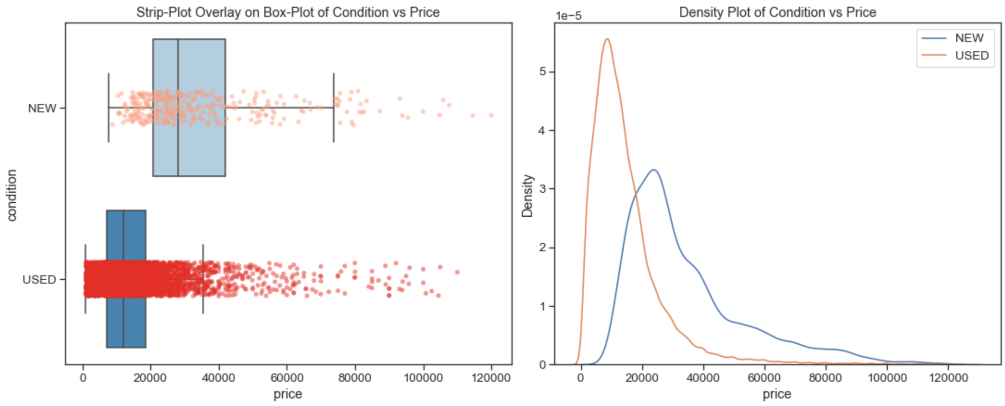

Figure 14: Box, strip and kernel density plots of condition against price.

5.3 Categorical-Categorical

5.3.1 Year Bins and Mileage Bins

Let's see how the average price of vehicles change when adjusting the grouped features of years (year bins) and mileage (mileage bins). To achieve this, use a group-by function to build dataframe of values for the two groups. Remembering that the summary statistic mean can be easily skewed by outliers, remove them for visualisation.

```
# Create groupby function of mean price for year and mileage bins
year_mileage = cars.loc[cars['outlier'] == False].groupby(['year_bins', 'mileage_bins
                                          '])[['price']].agg(['mean']).unstack('
                                          mileage_bins')
print(year_mileage)
```

mileage_bins year_bins	Very Low	Low	Medium	High	Very High
Very Old	24553.54067	19863.97275	11758.44527	7848.45611	5412.50014
Old	21044.68448	15208.45720	13566.82535	12740.01742	10871.22431
Neutral Age	18771.64337	15574.56166	16039.89544	15766.54535	13454.03598
New	23618.74579	19789.55553	17968.88318	16739.41534	13863.34513
Very New	31834.75074	27531.82399	23636.03333	9983.03125	8499.89744

Table 12: Mean price for grouped data by year bins and mileage bins.

From the graph in Figure 15, it can clearly be seen that vehicle prices decrease steadily based on age, but perhaps not as much as first thought. Interestingly, "Very New" vehicles with "High" or "Very High" mileage seem to have decreased value in comparison with counterparts in different year bins. On the other end of the spectrum, vehicles that are "Very Old" with "Very Low" mileage are rare and actually appear to increase in value. This could show that the relationship between these values is not linear and may not favour a linear model.

```
# Plot the groupby function of year and mileage bins
year_mileage.plot(kind='barh');
plt.xlabel('Price');
plt.title('Price of Year Bins per Mileage Bin');
```

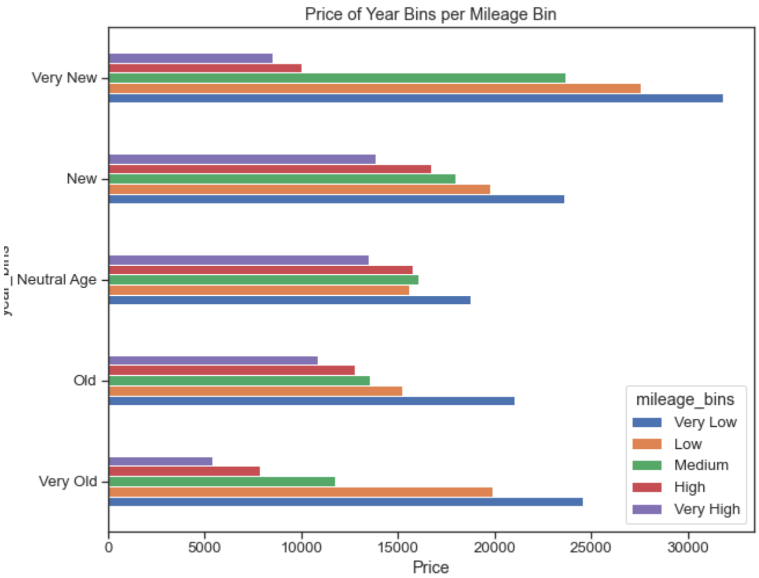

Figure 15: Plot to show change in price per year and mileage bins.

5.3.2 Fuel Type and Year Bins

What is the prevalence of differing fuel types throughout recent years? Find this out by grouping fuel types and year bins, then normalising the data to find what proportion of each fuel type exists in each year.

```
# Look at data from 2011 onwards
years = cars.loc[cars['year'] >= 2011]

# Groupby fuel, year and normalize to proportion
fuel_type = (years.value_counts(['fuel', 'year'])/years.value_counts('year')).unstack
                                                    ()

# Plot heatmap of proportional values
sns.heatmap(fuel_type, cmap='vlag', annot=True, fmt='.2f');
```

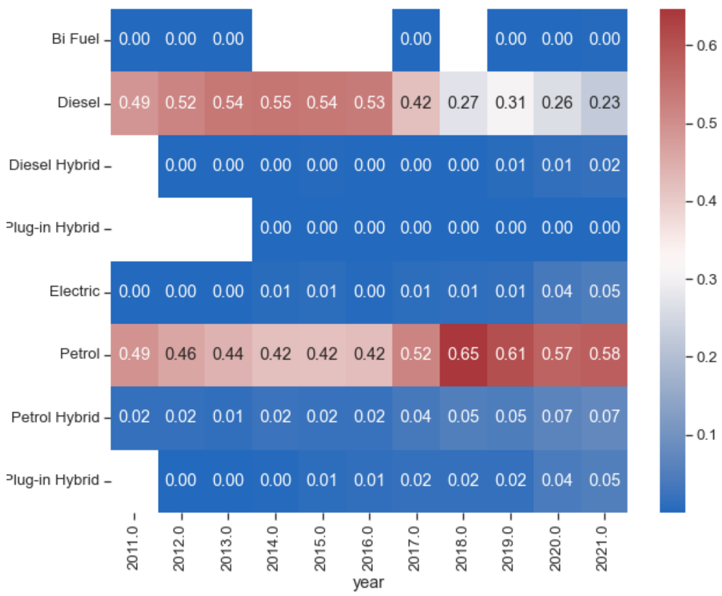

Figure 16: Heatmap of changes in fuel types over years.

Deriving Figure 16 shows that there have been small increases in the listings of Electric and Hybrid Vehicles, as one might expect with them being a more recent invention. Perhaps more surprisingly is that Diesel vehicles appear in higher proportions for years prior to 2017, so that the listings contain a higher proportion of newer Petrol vehicles.

Compare this with the grouped means of Fuel Types in Table , where Petrol ranks among the lowest price mean, and it becomes clearer that this could have an affect on the overall summary of the dataset.

```
# Mean price per fuel type groupings
cars.groupby('fuel')['price'].mean()
```

fuel	price
Diesel Hybrid	39601.98345
Petrol Plug-in Hybrid	35517.33015
Diesel Plug-in Hybrid	35464.17391
Electric	32671.74172
Petrol Hybrid	20146.84684
Petrol	16592.79505
Diesel	16392.13537
Bi Fuel	14820.73171

Table 13: Grouped price means of fuel types

5.3.3 Fuel Type and Condition

Following on from Table 13, Petrol and Diesel vehicles have a very similar grouped mean price. But how do these compare over time? In Table 14, Diesel cars appear to lose much more value when the condition changes from "NEW" to "USED".

```
# Get groupby function for fuel and condition
fuel_con = cars_vis.groupby(['fuel','condition'])['price'].mean().unstack()
print(fuel_con)
```

condition fuel	NEW	USED
Bi Fuel	13557.28889	16793.73611
Diesel	45686.81183	15439.24766
Diesel Hybrid	55464.42512	32554.92857
Diesel Plug-in Hybrid	56511.11111	34381.76000
Electric	41398.27129	28552.77111
Petrol	28247.71519	13531.95579
Petrol Hybrid	29840.65682	18155.21941
Petrol Plug-in Hybrid	47993.64770	29395.87728

Table 14: Mean price of grouped values for fuel and condition.

Figure 17 shows a bar plot of the difference in mean prices between new and used vehciles for fuel types. Petrol vehicles do not fall in price nearly as much as the Diesel vehicles.

```
# Get difference in mean prices
fuel_con['difference'] = fuel_con['NEW'] - fuel_con['USED']

# Create barplot of results
fuel_con['difference'].plot(kind='barh');
```

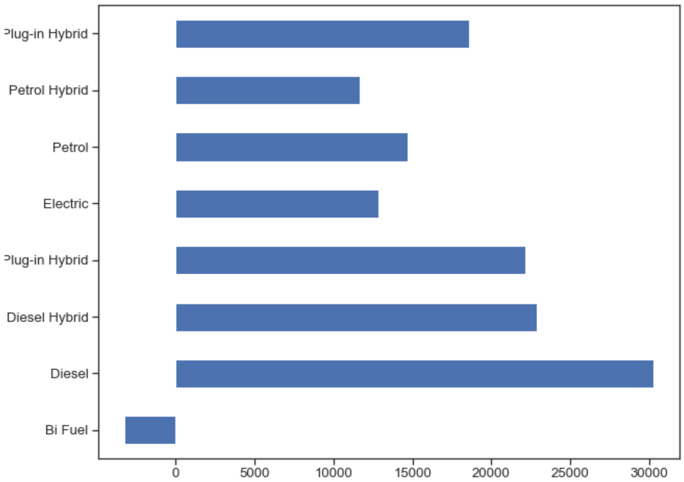

Figure 17: Difference in price between new and used vehicles per fuel type.

Figure 18 shows how the mean price of Diesel and Petrol vehicles changes with the year of registration for the vehicle. Diesel vehicles begin at a higher price mean, soon falling to below the value of petrol vehicles as they age. The box-plot also shows that the mean price of new Diesel vehicles is much different from the Petrol counterparts.

```python
# Create new dataframe of petrol and diesel vehicles
pet_dies = cars_vis[cars_vis['fuel'].isin(['Petrol','Diesel'])]

# Groupby fuel and year for mean price
pet_diesel = pet_dies.groupby(['fuel','year'])['price'].mean().unstack('fuel')

# Create figure for subplots
fig, axs = plt.subplots(1,2, figsize=(16, 8))

# Plot visual for fuel and condition
sns.boxplot(x='fuel', y='price', data=pet_dies, hue='condition', showfliers=False, ax
    =axs[1]);
sns.lineplot(data=data,markers=True, dashes=False, ax=axs[0]);
```

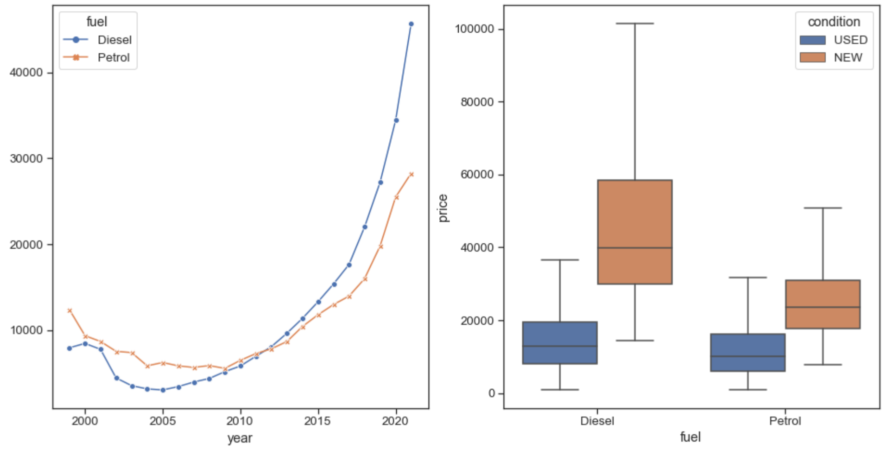

Figure 18: Line and box plot of changes in price for fuel type.

5.4 Conclusion

This project builds a foundation of relationships between variables and by no means touches every base. There may well be more undiscovered correlations, given infinite time.

Promising predictive variables of price are noted with a majority of features in the dataset being able to contribute to any future models built. The most interesting predictors of price are the year, mileage, condition and make.

5.5 References

Grant, P. (2012) *How to find outliers with IQR using python, Built In.*
Available at: https://builtin.com/data-science/how-to-find-outliers-with-iqr
Gerber, L. (November 2022) *Resampling Statistics Google Colab Notebook*, Manchester Metropolitan University

Manchester Metropolitan University