What Are We Looking For?

We assume "overpriced cars" to mean
there exists a car C with features X and MSRP Y
such that
Y > Y' for a "similar" car C' with "similar" features X'

i.e. Cars are overpriced when there exists a similar car that's cheaper.

The same logic follows for underpriced cars.

Inital Thoughts

My very first thought was to conduct some exploratory data analysis whereby I find the features which contribute most to MSRP, i.e. what factors contribute to a car's price.

Then use these features to rank all the cars by MSRP and their "intrinsic" value, and look at the difference in rankings between each car in these two rankings. The issue is how to calculate intrinsic value.

I considered fitting multivariate curves (Gaussians/polynomials) to MSRP, calculating a score, and using it to find outlier cars that could correspond to under/overpriced cars. For example in the gaussian case, we might consider "score" as the probability of seeing this car given all the other cars.

Another approach that I thought about was to use PCA to reduce the dimensionality of the data and visualize to confirm whether these are true outliers in the data.

In all of these cases, we would like to convert categorical/ordinal features to numeric ones.

In the end, I decided to create a model to predict the MSRP and look at the relative difference between the predicted and actual price of cars. In brief, this involved creating an XGBoost model to predict MSRP and using K-fold cross-validation to find the most under and overpriced cars in each fold.

Although it does involve creating a model, which I was trying to avoid, the model allows us to perform cross-validation which allows us to verify whether the car is incorrectly priced, whereas the other methods outlined can not easily do this.

My Approach

My approach consisted of the followed steps:

- **Exploring the data**
  - Correlating all features to MSRP using (absolute) Pearson correlation coefficient, *r*
  - (Considered using Kendall Tau correlation coefficient but MSRP is not inherently ordinal. Although I could've converted all MSRPs to rankings, making it ordinal, I felt this was unnecessary)
- **Cleaning data**
  - Dropping useless features
  - Drop features that have a high proportion of missing values
  - Filling missing values (if low proportion of missing values)

    - Converting non-numeric features to numeric features (one-hot encoding)
- **Creating a prototype model**
    - Just to see if my idea will work
    - Split the data into train and test sets (validation sets came later)
    - Train XGBoost Regressor model to predict MSRP
        - Using features that correlate to actual MSRP above 0.4 (i.e. *{|r|>0.4 for r(feature, MSRP) in features}*) as this is considered moderate to strong correlation in most academic papers
    - This model does not have to predict MSRP perfectly, since we're trying to model intrinsic value (not actual value), thus minimal hyperparameter tuning is performed.
        - A "low enough" error will suffice for this problem
- **5-fold cross-validation**
    - Split the data into 5 sets
    - Use 4 sets of the data to train, and test on the last set
    - Then find 10 most under/overpriced cars each K fold set (take note of it)
    - Repeat for all 5 sets
- **Most under/overpriced cars have the highest/lowest relative difference in MSRP among all K folds**
- **Create Final Model**
    - We need to create a final model as we can not use one of the five k fold models since we'd be testing on our train set
    - Instead, we take the 50 most under and over-priced cars (100 cars) and test on them, using all other cars as training data points
    - We then use this model's final predictions and explain
- **Explain the model**
    - Use SHAP to see why the model predicted what it did, compared to the average predicted MSRP
    - SHAP is a game theoric approach to explaining a model's output, it is based on if players (features) of a game (model) are/are not present, i.e. it figures out the contribution of each feature for any given prediction

We make a major assumption when creating a model: **We assume that the model's prediction will be closer to the car's "true intrinsic" value compared to its true MSRP.**

This approach has the flaw whereby a model's incorrect predictions can easily be seen as outliers.

We try to mitigate this by considering relative difference over the absolute difference.

Whereby if the model predicted `pred_msrp` =**£1,000** price for `true_msrp` =**£1,100** car, The relative difference would be (`true_msrp/pred_msrp`) 1,100/1,000 = 1.1

This means that the car in question is overpriced by 10%. We choose to use this relative method as price and utility of purchase are logarithmic in nature.

The feeling of happiness when receiving £100,000 when you're homeless is very different from receiving £100,000 when you're a billionaire;

This same logic applies to buying items and to model this we use relative price differences.

Dependancies and Imports

Uncomment the cell below and run to download and install all dependencies. Please note, this notebook assumes you are using Conda/MiniConda for your Python instance.

If you do not have access to this, there is a html version of this notebook that can be viewed.

In [1]:
```
# # This cell will install all libraries needed for this Jupyter Notebook.
# !pip install pandas
# !pip install numpy
# !pip install sklearn
# !pip install matplotlib
# !pip install seaborn
# !pip install xgboost
# !conda install -c conda-forge shap
```

In [2]:
```
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import shap
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import mean_absolute_error

shap.initjs()
pd.set_option('display.max_columns', None) # see all columns
sns.set(color_codes=True)
sns.set_theme(style="whitegrid")
%matplotlib inline
```

Exploring the Data

We will now read in the `car_data_cleaned.csv` and inspect the head of the dataframe containing the csv.

In [3]:
```
# Load and show first five rows of data
df = pd.read_csv("car_data_cleaned.csv")
df.head()
```

Out[3]:

Unnamed: 0	Car_Make_Model_Style	MSRP	Style Name	Drivetrain	Passenger Capacity	Passenger Doors	Body Style	EPA Classification

---

	Unnamed: 0	Car_Make_Model_Style	MSRP	Style Name	Drivetrain	Passenger Capacity	Passenger Doors	Body Style	EPA Classification
0	0	2019 Acura RDX Specs: FWD w/Technology Pkg	40600.0	FWD w/Technology Pkg	Front Wheel Drive	5	4	Sport Utility	Small Sport Utility Vehicles 2WD
1	1	2019 Acura RDX Specs: FWD w/Advance Pkg	45500.0	FWD w/Advance Pkg	Front Wheel Drive	5	4	Sport Utility	Small Sport Utility Vehicles 2WD
2	2	2019 Acura RDX Specs: FWD w/A-Spec Pkg	43600.0	FWD w/A-Spec Pkg	Front Wheel Drive	5	4	Sport Utility	Small Sport Utility Vehicles 2WD
3	3	2019 Acura RDX Specs: FWD	37400.0	FWD	Front Wheel Drive	5	4	Sport Utility	Small Sport Utility Vehicles 2WD
4	4	2019 Acura RDX Specs: AWD w/Technology Pkg	42600.0	AWD w/Technology Pkg	All Wheel Drive	5	4	Sport Utility	Small Sport Utility Vehicles 4WD

We will now describe the data. We see that MSRP has a mean of **46,757.18** with a minimum of **11,990.00** and a maximum of **548,800.00**.

In [4]:
```
# Describe data
df.describe()
```

Out[4]:

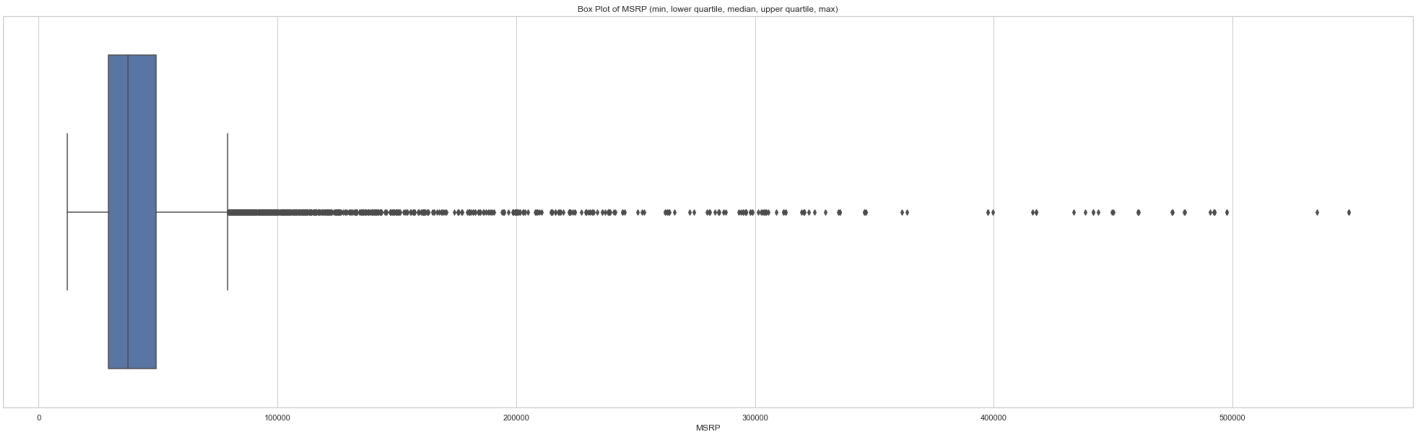
	Unnamed: 0	MSRP	Passenger Capacity	Passenger Doors	Base Curb Weight (lbs)	Front Hip Room (in)	Front Leg Room (in)	Secon Shoulde Room (in
count	11457.000000	11408.000000	11457.000000	11457.000000	7516.000000	8628.000000	10892.000000	9744.00000
mean	5728.000000	46757.176893	5.038841	3.615344	3689.308010	56.967216	42.212651	57.46161
std	3307.495351	39978.632981	1.458029	0.765427	641.898088	4.144232	1.571020	5.20019
min	0.000000	11990.000000	1.000000	2.000000	1808.000000	45.400000	35.800000	39.70000
25%	2864.000000	29120.000000	5.000000	4.000000	3263.000000	54.100000	41.200000	54.50000
50%	5728.000000	37442.500000	5.000000	4.000000	3635.000000	55.600000	41.900000	56.40000

	Unnamed: 0	MSRP	Passenger Capacity	Passenger Doors	Base Curb Weight (lbs)	Front Hip Room (in)	Front Leg Room (in)	Secon Shoulde Room (in
75%	8592.000000	49196.250000	5.000000	4.000000	4123.000000	60.730000	42.900000	60.50000
max	11456.000000	548800.000000	15.000000	4.000000	8591.000000	67.600000	63.900000	71.40000

A better way to visualise the distribution of MSRPs in the dataset is with a box plot. The box plot below shows that the median MSRP falls around **46,757** with many cars being above this number.

This means that there are many expensive cars in the data set and very few cheap cars. However, this does not necessarily imply that there are many over-priced cars and few under-priced cars.

In [5]:
```
msrp_box_plot = sns.boxplot(x=df["MSRP"])
msrp_box_plot.set_title("Box Plot of MSRP (min, lower quartile, median, upper quartile, ma
msrp_box_plot
fig = plt.gcf()
fig.set_size_inches(35, 10)
```



Now let's start the cleaning process, let's have a look at all columns and their data types. We see below that there are a lot of columns, not all of which will be useful for the purposes of prediction.

We can also see that some of these columns are of datatype `object`. This means that the columns are **categorical or otherwise.**

XGBoost and Pandas both have features to deal with these columns, however, we have chosen to manually deal with them...

In [6]:
```
# Name and datatype of all columns
sorted([(col, str(dtype)) for (col, dtype) in zip(df.columns, df.dtypes)], key=lambda t: t
```

Out[6]:
```
[('MSRP', 'float64'),
 ('Base Curb Weight (lbs)', 'float64'),
 ('Front Hip Room (in)', 'float64'),
 ('Front Leg Room (in)', 'float64'),
 ('Second Shoulder Room (in)', 'float64'),
 ('Passenger Volume', 'float64'),
 ('Second Head Room (in)', 'float64'),
 ('Front Shoulder Room (in)', 'float64'),
 ('Second Hip Room (in)', 'float64'),
 ('Front Head Room (in)', 'float64'),
 ('Second Leg Room (in)', 'float64'),
 ('Wheelbase (in)', 'float64'),
```

```
('Track Width, Front (in)', 'float64'),
('Width, Max w/o mirrors (in)', 'float64'),
('Track Width, Rear (in)', 'float64'),
('Height, Overall (in)', 'float64'),
('Fuel Tank Capacity, Approx (gal)', 'float64'),
('Fuel Economy Est-Combined (MPG)', 'float64'),
('EPA Fuel Economy Est - City (MPG)', 'float64'),
('EPA Fuel Economy Est - Hwy (MPG)', 'float64'),
('First Gear Ratio (:1)', 'float64'),
('Fourth Gear Ratio (:1)', 'float64'),
('Second Gear Ratio (:1)', 'float64'),
('Reverse Ratio (:1)', 'float64'),
('Fifth Gear Ratio (:1)', 'float64'),
('Trans Type', 'float64'),
('Third Gear Ratio (:1)', 'float64'),
('Final Drive Axle Ratio (:1)', 'float64'),
('Turning Diameter - Curb to Curb', 'float64'),
('Basic Miles/km', 'float64'),
('Basic Years', 'float64'),
('Corrosion Years', 'float64'),
('Drivetrain Miles/km', 'float64'),
('Drivetrain Years', 'float64'),
('Roadside Assistance Miles/km', 'float64'),
('Roadside Assistance Years', 'float64'),
('Maximum Alternator Capacity (amps)', 'float64'),
('Gears', 'float64'),
('Net Horsepower', 'float64'),
('Net Horsepower RPM', 'float64'),
('Net Torque', 'float64'),
('Net Torque RPM', 'float64'),
('Displacement (L)', 'float64'),
('Displacement (cc)', 'float64'),
('Rear Tire Width', 'float64'),
('Front Tire Width', 'float64'),
('Rear Wheel Size', 'float64'),
('Front Wheel Size', 'float64'),
('Tire Width Ratio', 'float64'),
('Wheel Size Ratio', 'float64'),
('Tire Ratio', 'float64'),
('Year', 'float64'),
('Unnamed: 0', 'int64'),
('Passenger Capacity', 'int64'),
('Passenger Doors', 'int64'),
('Car_Make_Model_Style', 'object'),
('Style Name', 'object'),
('Drivetrain', 'object'),
('Body Style', 'object'),
('EPA Classification', 'object'),
('Fuel System', 'object'),
('Steering Type', 'object'),
('Front Wheel Material', 'object'),
('Suspension Type - Front', 'object'),
('Suspension Type - Rear', 'object'),
('Air Bag-Frontal-Driver', 'object'),
('Air Bag-Frontal-Passenger', 'object'),
('Air Bag-Passenger Switch (On/Off)', 'object'),
('Air Bag-Side Body-Front', 'object'),
('Air Bag-Side Body-Rear', 'object'),
('Air Bag-Side Head-Front', 'object'),
('Air Bag-Side Head-Rear', 'object'),
('Brakes-ABS', 'object'),
('Child Safety Rear Door Locks', 'object'),
('Daytime Running Lights', 'object'),
('Traction Control', 'object'),
('Night Vision', 'object'),
('Rollover Protection Bars', 'object'),
```

```
('Fog Lamps', 'object'),
('Parking Aid', 'object'),
('Tire Pressure Monitor', 'object'),
('Back-Up Camera', 'object'),
('Stability Control', 'object'),
('Other Features', 'object'),
('Corrosion Miles/km', 'object'),
('Engine Configuration', 'object'),
('Engine Class', 'object')]
```

In [7]:

```
# Name and datatype of all columns such that...
# columns are numeric
numeric_col_dtype = [(col, str(dtype)) for (col, dtype) in zip(df.columns, df.dtypes) if '
# columns are NOT numeric
non_numeric_col_dtype = [(col, str(dtype)) for (col, dtype) in zip(df.columns, df.dtypes)
```

Let's have a look at the numeric columns:

We see that **Unnamed: 0** is considered numeric when really it is an index, we decided to keep this column as it is useful for joining operations later on. We do not use this column for prediction purposes.

We can also see that some of these columns have `NaN` as a value (e.g. **Maximum Alternator Capacity (amps)**), these are missing values and will need to be removed/replaced.

In [8]:

```
# View first five rows of data, only show numeric columns
df[[col for (col, dtype) in numeric_col_dtype]].head() # just to inspect the values
```

Out[8]:

	Unnamed: 0	MSRP	Passenger Capacity	Passenger Doors	Base Curb Weight (lbs)	Front Hip Room (in)	Front Leg Room (in)	Second Shoulder Room (in)	Passenger Volume	Second Head Room (in)	Front Shoulder Room (in)	Seco H Roo (i
0	0	40600.0	5	4	3790.0	55.0	41.6	56.6	104.0	38.3	59.7	49
1	1	45500.0	5	4	3829.0	55.0	41.6	56.6	104.0	38.3	59.7	49
2	2	43600.0	5	4	3821.0	55.0	41.6	56.6	104.0	38.3	59.7	49
3	3	37400.0	5	4	3783.0	55.0	41.6	56.6	104.0	38.3	59.7	49
4	4	42600.0	5	4	4026.0	55.0	41.6	56.6	104.0	38.3	59.7	49

Similarly, let's have a look at the non-numeric columns:

Some of these columns will have no impact on the prediction of MSRP, for example, **Style Name**, is an arbitrary name given to styles of the car. No doubt, this does play some impact on pricing but we will not be using any NLP algorithms to extract its impact. Instead, we chose to simply drop these useless columns.

We also see that some columns contain values that are exclusive *Yes/No* style answers, these can be transformed into 0s and 1s (not one-hot encoding).

Whereas other columns such as **Engine Configuration** are a finite set of possible values (*I, V, W, etc.*), columns such as these are a perfect candidate for one-hot encoding. This is a method, whereby a single categorical column is transformed to *C* numerical columns of 0s and 1s (where *C* is the number of unique values or categories within that column). Note that one-hot encoding only makes sense when there are more than two unique non-numerical values:

An example of one-hot encoding is as follows:

e.g.

categorical_column
A
B
C

->

categorical_column_A	categorical_column_B	categorical_column_C
1	0	0
0	1	0
0	0	1

This is done so that algorithms such as XGBoost can utilise useful information, even if it is non-numerical.

In [9]:
```
# View first five rows of data, only show non-numeric columns
df[[col for (col, dtype) in non_numeric_col_dtype]].head() # just to inspect the values
```

Out[9]:

	Car_Make_Model_Style	Style Name	Drivetrain	Body Style	EPA Classification	Fuel System	Steering Type	Front Wheel Material	Suspension Type - Front
0	2019 Acura RDX Specs: FWD w/Technology Pkg	FWD w/Technology Pkg	Front Wheel Drive	Sport Utility	Small Sport Utility Vehicles 2WD	Gasoline Direct Injection	Rack-Pinion	Aluminum	Strut
1	2019 Acura RDX Specs: FWD w/Advance Pkg	FWD w/Advance Pkg	Front Wheel Drive	Sport Utility	Small Sport Utility Vehicles 2WD	Gasoline Direct Injection	Rack-Pinion	Aluminum	Strut
2	2019 Acura RDX Specs: FWD w/A-Spec Pkg	FWD w/A-Spec Pkg	Front Wheel Drive	Sport Utility	Small Sport Utility Vehicles 2WD	Gasoline Direct Injection	Rack-Pinion	Aluminum	Strut
3	2019 Acura RDX Specs: FWD	FWD	Front Wheel Drive	Sport Utility	Small Sport Utility Vehicles 2WD	Gasoline Direct Injection	Rack-Pinion	Aluminum	Strut
4	2019 Acura RDX Specs: AWD w/Technology Pkg	AWD w/Technology Pkg	All Wheel Drive	Sport Utility	Small Sport Utility Vehicles 4WD	Gasoline Direct Injection	Rack-Pinion	Aluminum	Strut

We can deal with missing values by dropping rows, or filling them with a value (e.g. `0` or mean of that column).

Sometimes it is better to drop the entire column if the proportion of missing values is high.

Now, let's have a look at the proportion of missing values per column:

We see that columns such as **Passenger Volume** have a high proportion of missing values at around **48.3%**.

The cell below shows a tuple array of the form List(Tuple(**Column Name**, **Proportion of Missing Values**)):

In [10]:
```
prop_missing_vals_per_col = [(col, 1-df[col].notna().mean()) for col in df.columns]
prop_missing_vals_per_col = sorted(prop_missing_vals_per_col, key=lambda t: t[1], reverse=
prop_missing_vals_per_col[:25] # first 25 features ordered by the proportion of missing va
```

Out[10]:
```
[('Passenger Volume', 0.4837217421663612),
 ('Base Curb Weight (lbs)', 0.34398184516016406),
 ('Final Drive Axle Ratio (:1)', 0.34179977306450204),
 ('Second Hip Room (in)', 0.3267871170463472),
 ('EPA Classification', 0.3197172034564022),
 ('Track Width, Rear (in)', 0.2765121759622938),
 ('Track Width, Front (in)', 0.2764248930784673),
 ('Maximum Alternator Capacity (amps)', 0.2759011957755084),
 ('Front Hip Room (in)', 0.24692327834511651),
 ('EPA Fuel Economy Est - Hwy (MPG)', 0.1728201099764336),
 ('Fuel Economy Est-Combined (MPG)', 0.16374269005847952),
 ('Second Shoulder Room (in)', 0.14951557999476306),
 ('Second Head Room (in)', 0.14637339617700973),
 ('Second Leg Room (in)', 0.14471502138430659),
 ('Fifth Gear Ratio (:1)', 0.13921619970323817),
 ('Gears', 0.13345552937069038),
 ('Fourth Gear Ratio (:1)', 0.133193680719211),
 ('Third Gear Ratio (:1)', 0.13301911495155805),
 ('Second Gear Ratio (:1)', 0.13275726630007856),
 ('EPA Fuel Economy Est - City (MPG)', 0.12926595094701931),
 ('First Gear Ratio (:1)', 0.12725844461901026),
 ('Reverse Ratio (:1)', 0.06162171598149602),
 ('Front Shoulder Room (in)', 0.05402810508859213),
 ('Front Head Room (in)', 0.05385353932093917),
 ('Turning Diameter - Curb to Curb', 0.05298071048267439)]
```

Perhaps these columns with a high proportion of missing values are important to MSRP.

Let's have a look at which columns correlate to MSRP the most.

We use the absolute Pearson correlation coefficient as it can indicate both negative and positive correlations with MSRP.

It is usually used with continuous variables, which is ideal for MSRP - however, not all other features are continuous so these correlations are just to guide model selection.

Correlations above 0.3-0.4 are considered moderate, so correlations above this threshold are highly linked with increasing/decreasing MSRP.

We see that features related to a car's performance are strongly linked with its MSRP, with **Net Horsepower** being the most indicative to a car's price. Whereas other factors such as **Roadside Assistance Years** play little to no part in a car's MSRP.

The cell below shows a tuple array of the form List(Tuple(**Column Name**, **Absolute Pearson Correlation Coefficient with MSRP**)):

In [11]:
```
# Correlate each column with MSRP
msrp_correlations = [(col, abs(df["MSRP"].corr(df[col]))) for (col, _) in numeric_col_dtyp

# Order by most to least correlated.
msrp_correlations.sort(key=lambda t: t[1], reverse=True)

corr_threshold = 0.3
msrp_correlations_threshold = [(col, corr) for (col, corr) in msrp_correlations if corr>cc
msrp_correlations_threshold
```

Out[11]:
```
[('Net Horsepower', 0.6756857921105897),
 ('Net Torque', 0.6251060538650953),
 ('Tire Width Ratio', 0.5773850457839491),
 ('Basic Miles/km', 0.5744428500291611),
 ('Rear Tire Width', 0.5423069552508987),
 ('Rear Wheel Size', 0.5086533378210475),
 ('Front Wheel Size', 0.4913105212659206),
 ('Base Curb Weight (lbs)', 0.4609164441195924),
 ('Displacement (L)', 0.4008159294713095),
 ('Displacement (cc)', 0.38614409533137983),
 ('EPA Fuel Economy Est - Hwy (MPG)', 0.37292374229702585),
 ('EPA Fuel Economy Est - City (MPG)', 0.3675740000552493),
 ('Fuel Economy Est-Combined (MPG)', 0.364829722632772),
 ('Tire Ratio', 0.36442895728548086),
 ('Track Width, Rear (in)', 0.3484596284734415),
 ('Track Width, Front (in)', 0.3443527775078663),
 ('Front Tire Width', 0.33933231084205284),
 ('Maximum Alternator Capacity (amps)', 0.30188673802529525)]
```

The cell below shows a tuple array of the form List(Tuple(**Column Name**, **Absolute Pearson Correlation Coefficient with MSRP**, **Proportion of Missing Values**)) sorted by descending Absolute Pearson Correlation Coefficient with MSRP

In [12]:
```
sorted([(col, abs(df["MSRP"].corr(df[col])), 1-df[col].notna().mean()) for (col, _) in nun
```

Out[12]:
```
[('MSRP', 1.0, 0.004276861307497604),
 ('Net Horsepower', 0.6756857921105897, 0.0033167495854062867),
 ('Net Torque', 0.6251060538650953, 0.00497512437810943),
 ('Tire Width Ratio', 0.5773850457839491, 8.728288382653382e-05),
 ('Basic Miles/km', 0.5744428500291611, 0.0),
 ('Rear Tire Width', 0.5423069552508987, 8.728288382653382e-05),
 ('Rear Wheel Size', 0.5086533378210475, 0.0),
 ('Front Wheel Size', 0.4913105212659206, 0.0),
 ('Base Curb Weight (lbs)', 0.4609164441195924, 0.34398184516016406),
 ('Displacement (L)', 0.4008159294713095, 0.013179715457798724),
 ('Displacement (cc)', 0.38614409533137983, 0.038317185999825476),
 ('EPA Fuel Economy Est - Hwy (MPG)', 0.37292374229702585, 0.1728201099764336),
 ('EPA Fuel Economy Est - City (MPG)',
  0.3675740000552493,
  0.12926595094701931),
 ('Fuel Economy Est-Combined (MPG)', 0.364829722632772, 0.16374269005847952),
 ('Tire Ratio', 0.36442895728548086, 0.0006982630706118265),
 ('Track Width, Rear (in)', 0.3484596284734415, 0.2765121759622938),
 ('Track Width, Front (in)', 0.3443527775078663, 0.2764248930784673),
 ('Front Tire Width', 0.33933231084205284, 8.728288382653382e-05),
 ('Maximum Alternator Capacity (amps)',
  0.30188673802529525,
  0.2759011957755084),
 ('Fifth Gear Ratio (:1)', 0.26911579525358936, 0.13921619970323817),
 ('Drivetrain Years', 0.2642808687448104, 0.0),
 ('Wheel Size Ratio', 0.26195329380148097, 0.0),
 ('Fourth Gear Ratio (:1)', 0.258333400651743, 0.133193680719211),
 ('Front Hip Room (in)', 0.24575824291595905, 0.24692327834511651),
```

```
('Second Hip Room (in)', 0.24179771601465636, 0.3267871170463472),
('Trans Type', 0.23835103978399796, 0.0007855459544383603),
('Drivetrain Miles/km', 0.23032229291438355, 0.0),
('Width, Max w/o mirrors (in)', 0.22876669334124766, 0.008030025312036337),
('Gears', 0.22783447089544206, 0.13345552937069038),
('Third Gear Ratio (:1)', 0.2191728369600342, 0.13301911495155805),
('Second Gear Ratio (:1)', 0.21501697137193082, 0.13275726630007856),
('First Gear Ratio (:1)', 0.19059258947078211, 0.12725844461901026),
('Fuel Tank Capacity, Approx (gal)',
 0.18367856830514387,
 0.013179715457798724),
('Second Leg Room (in)', 0.18314938389982494, 0.14471502138430659),
('Passenger Doors', 0.1794057972711243, 0.0),
('Height, Overall (in)', 0.16777770694383365, 0.006458933403159617),
('Final Drive Axle Ratio (:1)', 0.15748682157232752, 0.34179977306450204),
('Passenger Capacity', 0.12122499817507172, 0.0),
('Net Torque RPM', 0.11360793493175475, 0.0054988216810683),
('Passenger Volume', 0.1081176024377934, 0.4837217421663612),
('Basic Years', 0.08742425847866672, 0.0),
('Net Horsepower RPM', 0.08715702149079634, 0.015099938901981358),
('Second Shoulder Room (in)', 0.08606596780927761, 0.14951557999476306),
('Turning Diameter - Curb to Curb', 0.07055319028924291, 0.05298071048267439),
('Front Shoulder Room (in)', 0.06435557178185893, 0.05402810508859213),
('Roadside Assistance Miles/km', 0.06320939734784455, 0.032294667015798195),
('Year', 0.05519547584683274, 0.0),
('Front Head Room (in)', 0.05469977871567715, 0.05385353932093917),
('Corrosion Years', 0.05381223534852794, 0.005324255913415343),
('Front Leg Room (in)', 0.05108735213087664, 0.049314829361962076),
('Unnamed: 0', 0.04971049274540194, 0.0),
('Reverse Ratio (:1)', 0.04722444402209958, 0.06162171598149602),
('Wheelbase (in)', 0.03562125544617259, 0.0014838090250501867),
('Second Head Room (in)', 0.011675531925192505, 0.14637339617700973),
('Roadside Assistance Years', 0.006328245507303729, 0.04573623112507641)]
```

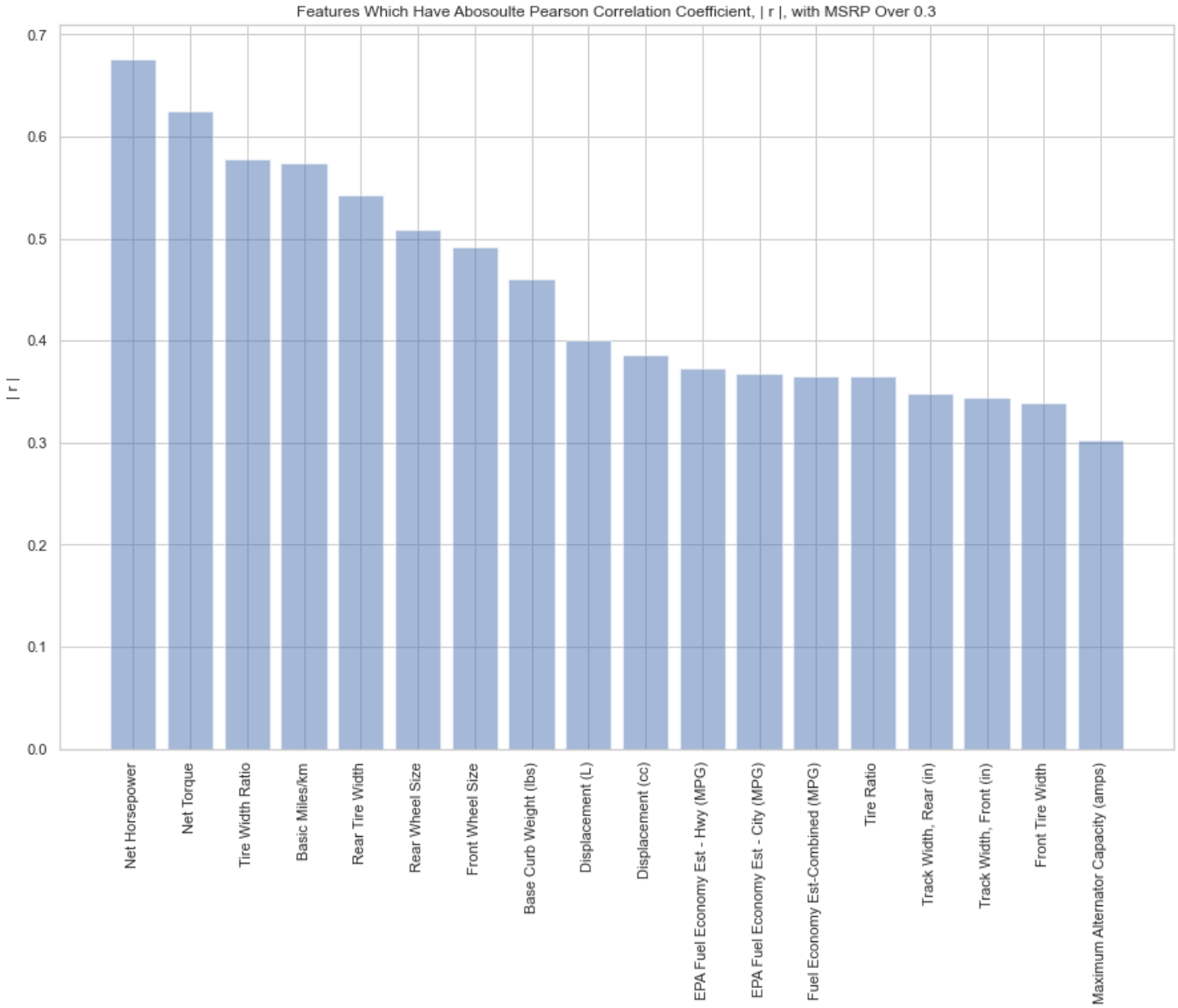
*Note that the feature **Base Curb Weight** (weight of the car w/ full tank of fuel) is heavily correlated with MSRP (0.461). Despite being a good indicator of MSRP, we chose to drop this column as it had over 34% missing values - this would normally still be fine, but these missing values were prevalent in the most expensive cars in the dataset (Lamborghinis, etc) which might be overpriced. Removing these would run the risk of removing the most overpriced cars, thus we remove the column instead of the row.*

To better visualise these correlations, see the plot below:

In [13]:

```
objects = [col for (col, corr) in msrp_correlations_threshold]
y_pos = np.arange(len(objects))
performance = [corr for (_, corr) in msrp_correlations_threshold]

plt.bar(y_pos, performance, align='center', alpha=0.5, color="b")
plt.xticks(y_pos, objects, rotation='vertical')
plt.ylabel('| r |')
plt.title(f'Features Which Have Abosoulte Pearson Correlation Coefficient, | r |, with MSR
fig = plt.gcf()
fig.set_size_inches(15, 10)
```



Cleaning the Data

In [14]:
```
# Reload data, just incase I changed something
df = pd.read_csv("car_data_cleaned.csv")
```

We remove any rows which have missing data for MSRP, as this is the variable we are trying to predict.

In [15]:
```
# Remove rows where MSRP is N/A
df = df[df["MSRP"].notna()]
```

We remove any columns where the proportion of missing values > 0.2 (only columns with less than 20% missing data will be kept).

This is done as missing values can severely damage a model's performance.

We do not conduct full feature engineering as we are only trying to find outlier cars.

If we were trying to predict price as accurately as possible, we would make more of an effort to keep as much useful data as possible.

In [16]:
```
# Remove columns where proportion of missing values > 0.2 (only columns with less than 20
for col, prop_nas in prop_missing_vals_per_col:
    if prop_nas > 0.2:
        try:
            df = df.drop(col, axis=1)
        except KeyError:
            pass
```

We replace all missing values within numeric columns with `0`s.

We briefly experimented with replacing all missing values within numeric columns with the mean of the column being considered, however, this heavily hurt the model's performance, and thus we fill with `0`s instead.

Furthermore, XGBoost uses a sparse matrix representation to do gradient boosting, this means that `0`s in the data will help training time - allowing us to increase the number of estimator trees, lower learning rates, etc. Which may make up for missing values in the data.

In future work, we would like to experiment with filling missing values and assessing model performance

In [17]:
```
# # Fill numeric columns with 0s
# filler = 0
# for (col, dtype) in zip(df.columns, df.dtypes):
#     if "obj" not in str(dtype) and col != "Unnamed: 0": # if numeric column
#         if abs(df["MSRP"].corr(df[col])) > 0.3:
#             df[col].fillna(value=df[col].mean(), inplace=True)
#         else:
#             df[col].fillna(value=filler, inplace=True)
```

In [18]:
```
# Fill numeric columns with 0s
filler = 0
for (col, dtype) in zip(df.columns, df.dtypes):
    if "obj" not in str(dtype) and col != "Unnamed: 0": # if numeric column
        df[col].fillna(value=filler, inplace=True)
```

We also drop useless columns which have little to no impact on MSRP.

If they do have an impact on price, it is not feasible to one-hot encode them as there are too many unique values and we want to keep dimensionality low for computability purposes.

In [19]:
```
# Drop "Style Name" and "Other Features" columns as they add no useful information
df = df.drop("Style Name", axis=1)
df = df.drop("Other Features", axis=1)
```

We also convert `Yes / No` columns to `1 / 0` columns. This is done in place and does not increase the dimensionality of the data.

In [20]:
```
yes_no_cols = [col for col in df.columns if "Yes" in df[col].unique() or "No" in df[col].u
for col in yes_no_cols:
    df[col] = df[col].eq("Yes").mul(1)
```

```
elementwise comparison failed; returning scalar instead, but in the future will perform el
```

ementwise comparison

For all other non numeric columns, we one-hot encode them as discussed above

In [21]:
```
# # Every other non-numeric column is useful, so we create a 1-k encoding for them to conv
df = pd.get_dummies(df, dummy_na=False, columns=[col for (col, dtype) in zip(df.columns, d
```

Finally, we drop duplicates and visualise the sorted dataframe

In [22]:
```
# Drop rows with missing values (there shouldn't be any) and drop duplicate row entires
df = df.dropna(axis=0)
df = df.drop_duplicates()
# Sort by MSRP and show
df.sort_values(by="MSRP", ascending=False)
```

Out[22]:

	Unnamed: 0	Car_Make_Model_Style	MSRP	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Front Shoulder Room (in)	Fron Hea Roon (in
6426	6426	2014 Lamborghini Aventador Specs: 2-Door Conve...	548800.0	2	2	0.0	0.0	0.0	0.0	0.0
6422	6422	2015 Lamborghini Aventador Specs: 2-Door Conve...	548800.0	2	2	0.0	0.0	0.0	0.0	0.0
6417	6417	2016 Lamborghini Aventador Specs: 2-Door Conve...	535500.0	2	2	0.0	0.0	0.0	0.0	0.0
6420	6420	2015 Lamborghini Aventador Specs: 2-Door Coupe...	497650.0	2	2	0.0	0.0	0.0	0.0	0.0
6424	6424	2014 Lamborghini Aventador Specs: 2-Door Coupe...	497650.0	2	2	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
2257	2257	2014 Chevrolet Spark Specs: 5-Door HB Manual LS	12170.0	4	4	42.0	49.7	37.3	50.7	39.4
8418	8418	2018 Nissan Versa Specs: S Manual	12110.0	5	4	41.8	51.9	36.6	51.7	39.8
8431	8431	2016 Nissan Versa Specs: 4-Door Sedan Manual 1...	11990.0	5	4	41.8	51.9	36.6	51.7	39.8
8449	8449	2014 Nissan Versa Specs: 4-Door Sedan Manual 1...	11990.0	5	4	41.8	51.9	36.6	51.7	39.8
8441	8441	2015 Nissan Versa Specs: 4-Door Sedan Manual 1...	11990.0	5	4	41.8	51.9	36.6	51.7	39.8

11408 rows × 208 columns

In [23]:
```
# Describe cleaned data
df.describe()
```

Out[23]:

	Unnamed: 0	MSRP	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Fr Shoul Room
count	11408.00000	11408.000000	11408.000000	11408.000000	11408.000000	11408.000000	11408.000000	11408.000
mean	5717.44872	46757.176893	5.040410	3.615796	40.137631	48.876646	32.692007	56.024
std	3303.43598	39978.632981	1.460096	0.764964	9.256432	21.050191	13.649435	14.059
min	0.00000	11990.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000
25%	2856.75000	29120.000000	5.000000	4.000000	41.100000	52.800000	36.700000	55.600
50%	5712.50000	37442.500000	5.000000	4.000000	41.800000	55.600000	37.900000	57.600
75%	8578.25000	49196.250000	5.000000	4.000000	42.800000	58.900000	39.100000	62.000
max	11456.00000	548800.000000	15.000000	4.000000	63.900000	71.400000	65.200000	74.800

Training Data

We remove the labels, indices, and any other columns that may unfairly allow the model to predict price.

We then prepare training data `X`, which can be seen below:

In [24]:
```
# X data
X = df[df.columns[3:]]
X = (X-X.min())/(X.max()-X.min()) # Normalise
# X = df[[c for (c, rr) in msrp_correlations_threshold]]
X
```

Out[24]:

	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Front Shoulder Room (in)	Front Head Room (in)	Second Leg Room (in)	Wheelbase (in)	Width, Max w/o mirrors (in)	He Ov
0	0.285714	1.0	0.651017	0.792717	0.587423	0.798128	0.622642	0.608558	0.608427	0.856816	0.59
1	0.285714	1.0	0.651017	0.792717	0.587423	0.798128	0.622642	0.608558	0.608427	0.856816	0.59
2	0.285714	1.0	0.651017	0.792717	0.587423	0.798128	0.622642	0.608558	0.608427	0.856816	0.59
3	0.285714	1.0	0.651017	0.792717	0.587423	0.798128	0.622642	0.608558	0.608427	0.856816	0.59
4	0.285714	1.0	0.651017	0.792717	0.587423	0.798128	0.622642	0.608558	0.608427	0.856816	0.59
...	...	...	...	...	...	...	...	...	...	...	

	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Front Shoulder Room (in)	Front Head Room (in)	Second Leg Room (in)	Wheelbase (in)	Width, Max w/o mirrors (in)	He Ov
11452	0.285714	1.0	0.655712	0.773109	0.576687	0.762032	0.608491	0.530903	0.613483	0.840779	0.54
11453	0.285714	1.0	0.655712	0.773109	0.573620	0.762032	0.608491	0.530903	0.614045	0.840779	0.54
11454	0.285714	1.0	0.655712	0.773109	0.573620	0.762032	0.608491	0.530903	0.614045	0.840779	0.54
11455	0.285714	1.0	0.655712	0.773109	0.573620	0.762032	0.608491	0.530903	0.614045	0.840779	0.52
11456	0.285714	1.0	0.655712	0.773109	0.573620	0.762032	0.608491	0.530903	0.614045	0.840779	0.52

11408 rows × 205 columns

Similarly, we use the **MSRP** column as our `Y` target vector.

We also show the first three columns of the cleaned dataset to more easily identify the cars.

Non-numeric columns **Unnamed: 0** and **Car_Make_Model_Style** are kept to easily identify the cars, but these are not used in training.

In [25]:
```
# Y targets
Y = df[df.columns[2]]
print(Y)
df[df.columns[:3]]
```

```
0         40600.0
1         45500.0
2         43600.0
3         37400.0
4         42600.0
           ...
11452     45700.0
11453     41200.0
11454     44850.0
11455     41000.0
11456     44650.0
Name: MSRP, Length: 11408, dtype: float64
```

Out[25]:

	Unnamed: 0	Car_Make_Model_Style	MSRP
0	0	2019 Acura RDX Specs: FWD w/Technology Pkg	40600.0
1	1	2019 Acura RDX Specs: FWD w/Advance Pkg	45500.0
2	2	2019 Acura RDX Specs: FWD w/A-Spec Pkg	43600.0
3	3	2019 Acura RDX Specs: FWD	37400.0
4	4	2019 Acura RDX Specs: AWD w/Technology Pkg	42600.0
...	...	...	...
11452	11452	2018 Volvo V60 Cross Country Specs: T5 AWD Pla...	45700.0
11453	11453	2016 Volvo V60 Cross Country Specs: 4-Door Wag...	41200.0
11454	11454	2016 Volvo V60 Cross Country Specs: 4-Door Wag...	44850.0
11455	11455	2015 Volvo V60 Cross Country Specs: 2015.5 4-D...	41000.0
11456	11456	2015 Volvo V60 Cross Country Specs: 2015.5 4-D...	44650.0

11408 rows × 3 columns

Prototype Model

The prototype model is used just to see if the model works. We use an XGBRegressor which is a tree ensemble algorithm used for regression problems.

We chose to use this model as it is a scalable machine learning algorithm that has proven itself in many Machine Learning competitions (Kaggle). It uses more and more accurate approximations to find the best tree model, addressing the shortfalls of its predictions at every step, while remaining extremely interpretable.

We use a random seed of 42 and a test set size of 20%, i.e we train on 80% of the data and test on the remaining 20%.

We choose not to use validation sets as we do not care about fine-tuning the model's predictions as we may overfit and miss the outlier cars.

In [26]:
```
# split data into train and test sets
seed = 42
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_stat
print(f"{X_train.shape=},\n{X_test.shape=},\n{y_train.shape=},\n{y_test.shape=}")
# X_train, X_test, y_train, y_test = X_test, X_train, y_test, y_train
```

```
X_train.shape=(9126, 205),
X_test.shape=(2282, 205),
y_train.shape=(9126,),
y_test.shape=(2282,)
```

In [27]:
```
# fit model no training data
model = XGBRegressor(n_estimators=1500, learning_rate=0.05)
model.fit(X_train,
          y_train,
          early_stopping_rounds=5,
          eval_set=[(X_test, y_test)],
          verbose=False)
```

Out[27]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=1500, n_jobs=16, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [28]:
```
# make predictions for test data
y_pred = model.predict(X_test)
print(f"{mean_absolute_error(y_test, y_pred)=}")
```

```
mean_absolute_error(y_test, y_pred)=1675.9682026628232
```

To visualise the model's predictions, we will sort all cars by their MSRP and then predict the value of each of these cars and plot.

The plot below shows every car (*in 20% test set*) ordered by **true MSRP** (`blue`) and that car's **predicted MSRP** (`orange`).

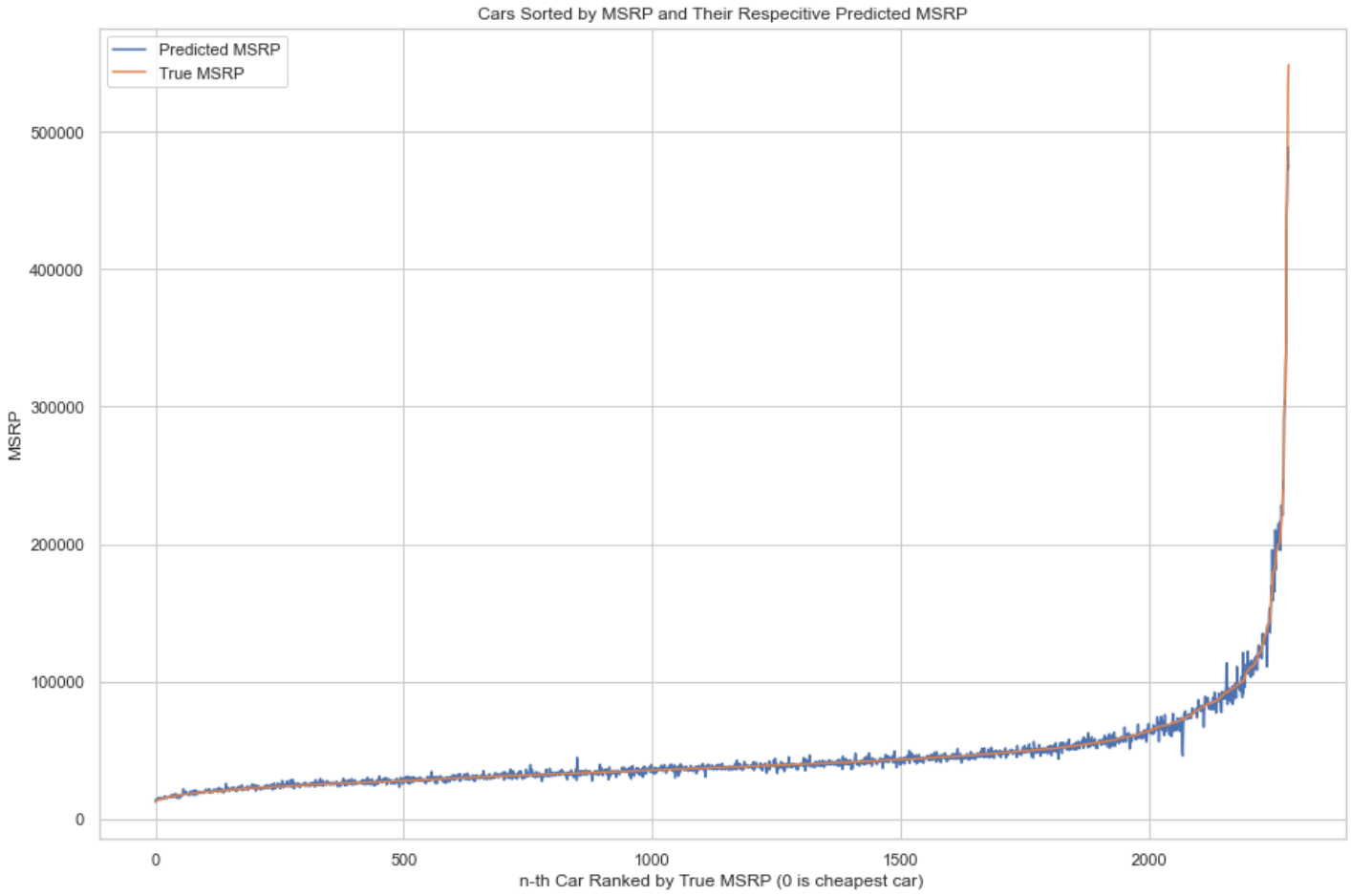
This allows us to easily visualise the absolute error for each car (not relative).

However absolute is not a good indicator of price in the eyes of humans as discussed prior.

We can also clearly see that the distribution of car prices is exponential.

In [29]:
```
true_pred_ys = sorted(list(zip(y_test, y_pred)), key=lambda t: t[0])
x_ax = range(len(true_pred_ys))
```

In [30]:
```
plt.plot(x_ax, [pred_y for (_, pred_y) in true_pred_ys], label="Predicted MSRP")
plt.plot(x_ax, [true_y for (true_y, _) in true_pred_ys], label="True MSRP")
plt.title("Cars Sorted by MSRP and Their Respecitive Predicted MSRP")
plt.ylabel("MSRP")
plt.xlabel("n-th Car Ranked by True MSRP (0 is cheapest car)")
fig = plt.gcf()
fig.set_size_inches(15, 10)
plt.legend()
plt.show()
```



A good way to visualise exponential data is with logarithmic scales, i.e. transforming all data points, `d` , to `log(d)`

The plot below shows the same cars ordered by **log** MSRP and its respective **log** predicted MSRP.

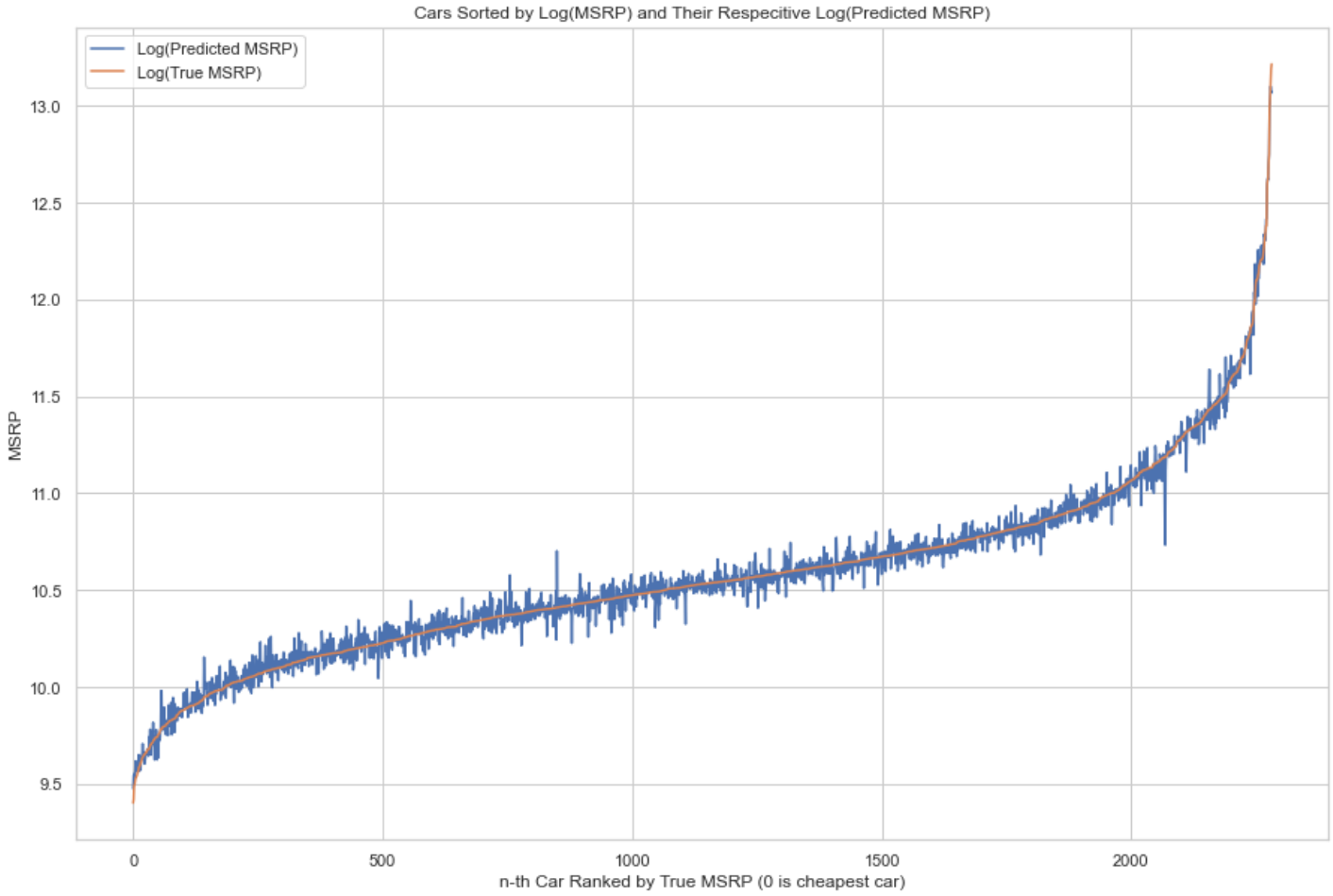
Again, this does not show the relative difference, however, it allows us to better visualise the absolute differences. We see that the model is good at predicted the price of expensive cars. However, it is bad at predicting the price of cheap cars in terms of absolute price.

We can also use this plot as an indication of the model variance, which is relatively high - but the model is only being tested on a fifth of the total data.

We can use cross-validation to reduce the variance of these predications.

In [31]:
```
plt.plot(x_ax, [np.log(pred_y) for (_, pred_y) in true_pred_ys], label="Log(Predicted MSRP
plt.plot(x_ax, [np.log(true_y) for (true_y, _) in true_pred_ys], label="Log(True MSRP)")
plt.title("Cars Sorted by Log(MSRP) and Their Respecitive Log(Predicted MSRP)")
plt.ylabel("MSRP")
plt.xlabel("n-th Car Ranked by True MSRP (0 is cheapest car)")
fig = plt.gcf()
fig.set_size_inches(15, 10)

plt.legend()
plt.show()
```



The plot below shows the cars ordered by true MSRP, but this time the blue line shows the relative difference (ratio).

This graph debunks our hypothesis that the model is predicting well on more expensive cars, **the model seems to be equally bad everywhere**.

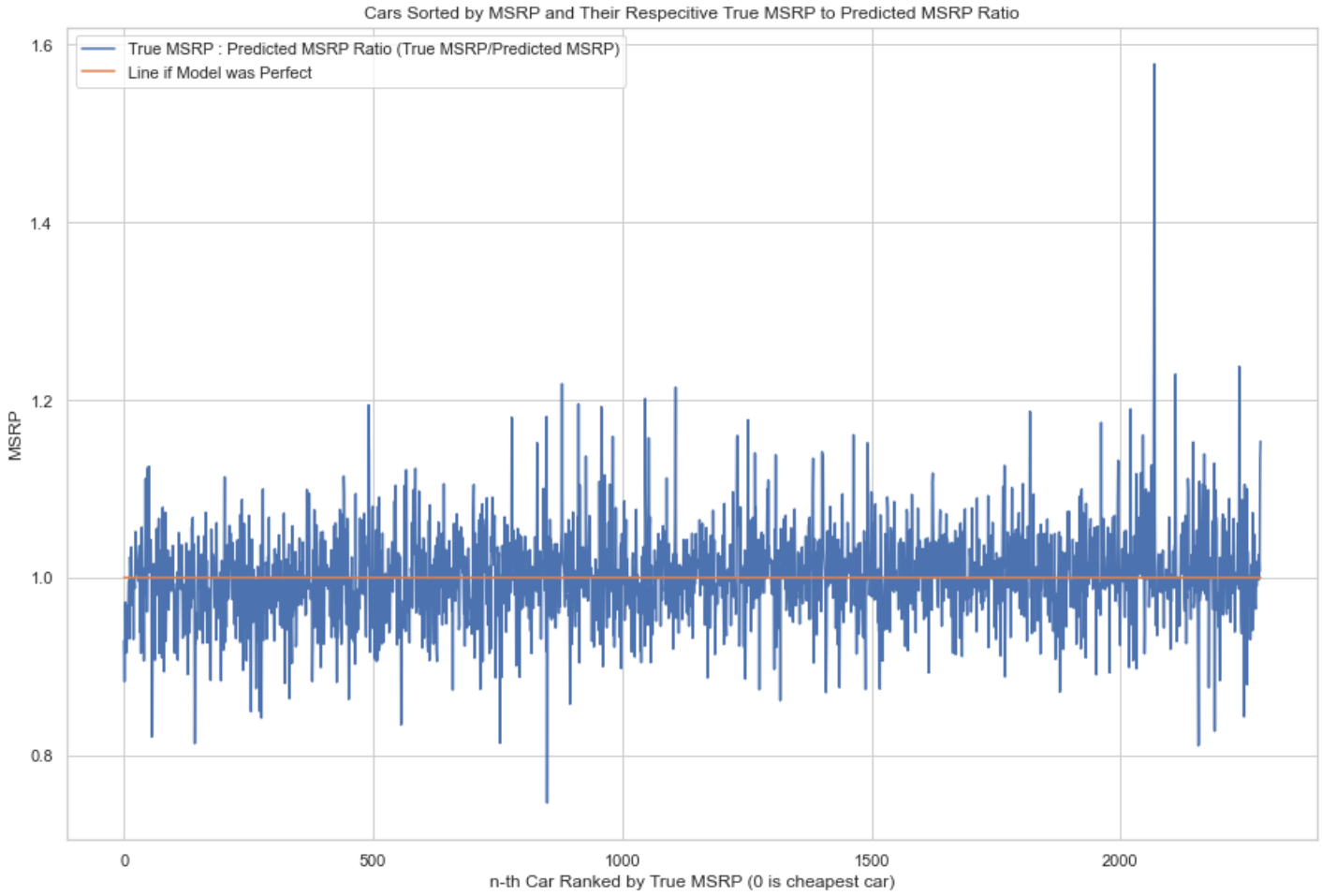
This is actually not a bad thing as we do not need to consider class weights/undersampling to try and un-skew the data.

The two data points furthest away from the orange line, above and below, are the most over and underpriced cars in this test set.

In [32]:
```
plt.plot(x_ax, [true_y/pred_y for (true_y, pred_y) in true_pred_ys], label="True MSRP : Pr
plt.plot(x_ax, [1 for _ in true_pred_ys], label="Line if Model was Perfect")

plt.title("Cars Sorted by MSRP and Their Respecitive True MSRP to Predicted MSRP Ratio")
plt.ylabel("MSRP")
plt.xlabel("n-th Car Ranked by True MSRP (0 is cheapest car)")
fig = plt.gcf()
fig.set_size_inches(15, 10)

plt.legend()
plt.show()
```



Now let's have a look at what these two data points were.

First, we will look at the most over-priced car within **this test set**.

This car is a **2015 Chevrolet Camaro Specs: 2-Door Coupe Z/28**.

We deem this car as over-priced as we predicted a price of **45,825** but the actual price of the car is **72,305**. This is a **57.8%** mark-up!

In [33]:
```
i_test_pred_ratio = [(index, round(y_test), round(y_pred), y_test/y_pred) for (index, y_te
i_test_pred_ratio.sort(key=lambda t: t[3], reverse=True)
top_k = 10

print(f"Top {top_k} most OVER-priced cars, (df_index,  y_test,  y_pred,  y_test/y_pred):")
print(str(i_test_pred_ratio[:top_k]).replace("),", "),\n"))
```

```
overpriced_index, overpriced_true_msrp, overpriced_pred_msrp, overpriced_ratio = i_test_pr
print(f"i.e. Most over-priced car in test set is:\n{overpriced_index=} car has max percent

df[df["Unnamed: 0"] == overpriced_index]
```

```
Top 10 most OVER-priced cars, (df_index,  y_test,  y_pred,  y_test/y_pred):
[(2016, 72305, 45825, 1.5778517287801384),
 (2024, 72305, 50758, 1.424517995909627),
 (9853, 137000, 110708, 1.237490922232065),
 (6508, 82150, 66851, 1.2288499258613823),
 (6059, 33645, 27621, 1.2180812327098833),
 (5954, 36995, 30474, 1.2139783789839529),
 (5974, 35995, 29961, 1.2013823813843305),
 (5809, 34095, 28523, 1.1953399883538054),
 (7242, 27500, 23032, 1.1939662643475228),
 (5821, 34695, 29105, 1.1920484205249717)]
i.e. Most over-priced car in test set is:
overpriced_index=2016 car has max percentage increase of 1.578 from predicted MSRP (45825)
to true MSRP (72305)
```

Out[33]:

	Unnamed: 0	Car_Make_Model_Style	MSRP	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Front Shoulder Room (in)	Front Head Room (in)
2016	2016	2015 Chevrolet Camaro Specs: 2-Door Coupe Z/28	72305.0	4	2	42.4	50.42	35.3	56.9	37.4

Similarly, we deem **2018 Jeep Grand Cherokee Specs: Upland 4x4** to be the most under-priced car in **this test set** as the predicted price was **74.7%** of the actual price of the car.

Of course, these are just the outliers within our 20% test set.

In [34]:

```
i_test_pred_ratio.sort(key=lambda t: t[3], reverse=False)
print(f"Top {top_k} most UNDER-priced cars, (df_index,  y_test,  y_pred,  y_test/y_pred):"
print(str(i_test_pred_ratio[:top_k]).replace("),", "),\n"))

underpriced_index, underpriced_true_msrp, underpriced_pred_msrp, underpriced_ratio = i_tes
print(f"i.e. {underpriced_index=} car has max percentage increase of {underpriced_ratio:.3
df[df["Unnamed: 0"] == underpriced_index]
```

```
Top 10 most UNDER-priced cars, (df_index,  y_test,  y_pred,  y_test/y_pred):
[(5874, 33195, 44463, 0.7465705574680249),
 (5622, 92000, 113437, 0.8110257071595264),
 (5625, 92000, 113221, 0.8125698819758806),
 (2851, 20895, 25692, 0.8132815058552219),
 (5852, 31945, 39251, 0.8138611321179834),
 (2906, 17740, 21610, 0.8209298945710718),
 (2899, 17740, 21507, 0.824853342080545),
 (8503, 99990, 120863, 0.8273028451657121),
 (10786, 28700, 34395, 0.8344344115154493),
 (4196, 24070, 28582, 0.8421437606717175)]
i.e. underpriced_index=5874 car has max percentage increase of 0.747 from predicted MSRP
(44463) to true MSRP (33195)
```

Out[34]:

	Unnamed: 0	Car_Make_Model_Style	MSRP	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Front Shoulder Room (in)	Front Head Room (in)

	Unnamed: 0	Car_Make_Model_Style	MSRP	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)	Front Shoulder Room (in)	Front Head Room (in)
5874	5874	2018 Jeep Grand Cherokee Specs: Upland 4x4	33195.0	5	4	40.3	58.0	39.2	58.7	39.9

---

k-fold Cross-Validation Model

In this section, we will use 5-fold cross-validation which will create 5 different models which are tested on 5 unique test sets.

We take note of the 10 most overpriced cars and the 10 most underpriced cars when testing on each set.

*These 20*5 cars are then used for the final model and SHAP explanations.*

The procedure for K-fold cross-validation is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
   3.1 Take the group as a holdout or test data set
   3.2 Take the remaining groups as a training data set
   3.3 Fit a model on the training set and evaluate it on the test set
   3.4 Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

We use k=5. This means that use 80% (4/5) of the data to train our model, and 20% (1/5) to test the model. We then pick the 10 most under and over-priced cars and take note of them.

We then consider the other 4/5 sets of data to test and train on. This is analogous to a rotating pie.

In [35]:
```
kf = KFold(n_splits=5, random_state=seed, shuffle=True)
kf.get_n_splits(X)

i_test_pred_ratio = []
overpriced_in_kfold = []
underpriced_in_kfold = []

for kfold_number, (train_index, test_index) in enumerate(kf.split(X)):
     X_train, X_test = X.iloc[train_index], X.iloc[test_index]
     y_train, y_test = Y.iloc[train_index], Y.iloc[test_index]

     # fit model no training data
     model = XGBRegressor(n_estimators=1500, learning_rate=0.05)
     model.fit(X_train,
```

```
                y_train,
                early_stopping_rounds=5,
                eval_set=[(X_test, y_test)],
                verbose=False)

    # make predictions for test data
    y_pred = model.predict(X_test)
    print(f"kfold={kfold_number+1} -> {mean_absolute_error(y_test, y_pred)=}")

    i_test_pred_ratio += [(index, f"{y_test:.2f}", f"{y_pred:.2f}", y_test/y_pred) for (i
    i_test_pred_ratio.sort(key=lambda t: t[3], reverse=True)
```

```
kfold=1 -> mean_absolute_error(y_test, y_pred)=1675.9682026628232
kfold=2 -> mean_absolute_error(y_test, y_pred)=1995.9642703083919
kfold=3 -> mean_absolute_error(y_test, y_pred)=2315.041644699003
kfold=4 -> mean_absolute_error(y_test, y_pred)=2067.5489861046417
kfold=5 -> mean_absolute_error(y_test, y_pred)=2292.854461362204
```

After 5-fold cross-validation, and taking note of the top 10 most under and overpriced cars in each fold, we now know the most over and underpriced cars and we can rank them.

But now we have the issue that we can not easily explain our model's output with SHAP.

If we pick any of the 5 models we have just created, we run the risk of testing on training datapoints. So instead we create a new final model which uses these outlier cars as test datapints.

But first, let's have a look at what the 5-fold models outputted as the most under/overpriced cars:

Rankings

We see that the most overpriced car over the 5 models is **2018 Mercedes-Benz G Class Specs: G 550 4x4 Squared** which had a **True:Predicted MSRP Ratio** at `1.86`.

This means that this car is almost double the price of what it should be.

This somewhat makes sense the G 550 or G-Wagon is a common status symbol for wealthy individuals.

We also see that the most underpriced car over the 5 models is **2016 Buick Regal Specs: 4-Door Sedan Sport Touring**.

This again makes sense as this car is often used as a Police vehicle in the USA.

Being underpriced (**True:Predicted MSRP Ratio** at `0.66`) means it's great value for money which is important for Police cars as they tend to do hundreds of thousands of miles.

In [36]:
```
df_outlier_cars = pd.DataFrame(i_test_pred_ratio, columns =['Index', 'True MSRP', 'Predict
df_outlier_cars = df_outlier_cars.set_index('Index').join(df.set_index("Unnamed: 0"))
df_outlier_cars.drop("MSRP", axis=1, inplace=True)
df_outlier_cars = df_outlier_cars.sort_values(by=["True:Predicted MSRP Ratio"], ascending=
df_outlier_cars
```

Out[36]:

Index	True MSRP	Predicted MSRP	True:Predicted MSRP Ratio	Car_Make_Model_Style	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)
7283	227300.00	122112.64	1.861396	2018 Mercedes-Benz G Class Specs: G 550 4x4 Sq...	5	4	52.5	56.30	40.1
5792	187500.00	115664.52	1.621068	2019 Jaguar XE SV Specs: Project 8 AWD	4	4	41.5	54.70	37.3
8770	293200.00	182634.94	1.605388	2018 Porsche 911 Specs: GT2 RS Coupe	2	2	0.0	0.00	0.0
2016	72305.00	45824.96	1.577852	2015 Chevrolet Camaro Specs: 2-Door Coupe Z/28	4	2	42.4	50.42	35.3
5615	165000.00	107792.48	1.530719	2016 Jaguar F-Type Specs: 2-Door Convertible A...	2	2	43.0	0.00	0.0
...	...	...	...	...	...	...	...	...	...
5619	99000.00	136298.69	0.726346	2015 Jaguar F-Type Specs: 2-Door Coupe V8 R	2	2	43.0	0.00	0.0
2630	38995.00	53808.56	0.724699	2019 Dodge Challenger Specs: R/T Scat Pack Wid...	5	2	42.0	53.90	37.1
9014	35985.00	50275.66	0.715754	2016 Ram 1500 Specs: 2WD Reg Cab 120.5" Sport	2	2	41.0	0.00	0.0
9560	325000.00	471744.94	0.688932	2019 Rolls-Royce Cullinan Specs: Sport Utility	5	4	0.0	0.00	0.0
1294	28565.00	43520.59	0.656356	2016 Buick Regal Specs: 4-Door Sedan Sport Tou...	5	4	0.0	0.00	0.0

11408 rows × 209 columns

Looking at the top 10 most overpriced cars (in each fold) shows the trend that all of these cars are relatively high-performance sports cars.

We saw that Net Horsepower was the most indicative of a car's price and these are all fast cars or have big engines.

We also see the trend of electric/hybrid cars being classed as overpriced. This makes sense as with these cars, we are paying for the alternative fuel source and not the car specifically (longevity).

In [37]:
```
df_outlier_cars[:top_k]
```

Out[37]:

Index	True MSRP	Predicted MSRP	True:Predicted MSRP Ratio	Car_Make_Model_Style	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)
7283	227300.00	122112.64	1.861396	2018 Mercedes-Benz G Class Specs: G 550 4x4 Sq...	5	4	52.5	56.30	40.1
5792	187500.00	115664.52	1.621068	2019 Jaguar XE SV Specs: Project 8 AWD	4	4	41.5	54.70	37.3
8770	293200.00	182634.94	1.605388	2018 Porsche 911 Specs: GT2 RS Coupe	2	2	0.0	0.00	0.0
2016	72305.00	45824.96	1.577852	2015 Chevrolet Camaro Specs: 2-Door Coupe Z/28	4	2	42.4	50.42	35.3
5615	165000.00	107792.48	1.530719	2016 Jaguar F-Type Specs: 2-Door Convertible A...	2	2	43.0	0.00	0.0
2024	72305.00	50757.52	1.424518	2014 Chevrolet Camaro Specs: 2-Door Coupe Z/28	4	2	42.4	50.42	35.3
6752	120440.00	84633.90	1.423070	2016 Lexus LS Specs: 4-Door Sedan Hybrid	5	4	44.0	56.40	38.0
6758	120440.00	85460.17	1.409311	2015 Lexus LS Specs: 4-Door Sedan Hybrid	5	4	44.0	56.40	38.0
827	163300.00	116327.32	1.403797	2019 BMW i8 Specs: Roadster	2	2	43.1	0.00	0.0
6762	120060.00	85749.89	1.400118	2014 Lexus LS Specs: 4-Door Sedan Hybrid	5	4	44.0	56.40	38.0

Looking at the top 10 most underpriced cars (in each fold) shows an odd trend.

Some of these cars are known for being cheap and great value for money. For example, the **2019 Dodge Challenger R/T Scat Pack Wide Body** is known for being great as they offer unrivaled horsepower for the money you pay, the essence of American muscle.

Strangely, the model deems cars such as the **2019 Rolls-Royce Cullinan Specs: Sport Utility** as under-priced, even though this again is a luxury car and by no means cheap. In addition, one would be paying for the Rolls-Royce brand name.

The model predicted the car to be more expensive than it actually is. We do not believe that the **2019 Rolls-Royce Cullinan Specs: Sport Utility** is actually overpriced, instead these are outlier predictions.

This is the main issue with this approach, if a car is predicted badly it can easily be recognised as an over/underpriced car. This is caused by our choice of metric, relative difference (`true_msrp/pred_msrp`) - it's not perfect!

In instances like these, we use human expertise and domain knowledge combined with more rigorous cross-validation to iron out these outliers.

But since we are specifically concerned with outlier cars and not predicting as accurately as possible, we will not do this.

Instead, we will create a final model, whereby we use the top/bottom 50 cars to test on and train on the rest in our final model

In [38]:
```
df_outlier_cars[-top_k::][::-1]
```

Out[38]:

Index	True MSRP	Predicted MSRP	True:Predicted MSRP Ratio	Car_Make_Model_Style	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)
1294	28565.00	43520.59	0.656356	2016 Buick Regal Specs: 4-Door Sedan Sport Tou...	5	4	0.0	0.0	0.0
9560	325000.00	471744.94	0.688932	2019 Rolls-Royce Cullinan Specs: Sport Utility	5	4	0.0	0.0	0.0
9014	35985.00	50275.66	0.715754	2016 Ram 1500 Specs: 2WD Reg Cab 120.5" Sport	2	2	41.0	0.0	0.0
2630	38995.00	53808.56	0.724699	2019 Dodge Challenger Specs: R/T Scat Pack Wid...	5	2	42.0	53.9	37.1
5619	99000.00	136298.69	0.726346	2015 Jaguar F-Type Specs: 2-Door Coupe V8 R	2	2	43.0	0.0	0.0
9093	35705.00	48814.97	0.731435	2015 Ram 1500 Specs: 2WD Reg Cab 120.5" Sport	2	2	41.0	0.0	0.0
5873	30895.00	41787.95	0.739328	2018 Jeep Grand Cherokee Specs: Altitude 4x2	5	4	40.3	58.0	39.2
8934	37095.00	50109.55	0.740278	2018 Ram 1500 Specs: Sport 4x2 Reg Cab 6'4" Box	2	2	41.0	0.0	0.0
195	73995.00	99168.26	0.746156	2019 Alfa Romeo Giulia Specs: Quadrifoglio RWD	5	4	42.4	53.6	37.6
5874	33195.00	44463.31	0.746571	2018 Jeep Grand Cherokee Specs: Upland 4x4	5	4	40.3	58.0	39.2

Final Model and Results

As mentioned prior, this is our final model. We will use the top 50 most overpriced and underpriced cars in the ranking from our k-fold model (100 cars in total) to test out the model.

This means that we will use the rest of the data to train on. I.e. this is a specific case of 99% training and 1% testing.

This will be our final model and we will use Shapely values to explain why the model predicted what it did (compared to its baseline prediction)

In [39]:
```
top_k = 50

# get indexes of top and bottom 50 cars in previous ranking (seen above)
indices_of_top_k_min = list(df_outlier_cars[-top_k::][::-1].index)
indices_of_top_k_max = list(df_outlier_cars[:top_k].index)
indices_of_top_k_min_and_max = indices_of_top_k_min + indices_of_top_k_max
```

In [40]:
```
# split data
X_train = X[X.index.isin([x_i for x_i in X.index if x_i not in indices_of_top_k_min_and_ma
y_train = Y[Y.index.isin([y_i for y_i in Y.index if y_i not in indices_of_top_k_min_and_ma
X_test = X.loc[indices_of_top_k_min_and_max]
y_test = Y.loc[indices_of_top_k_min_and_max]

# train model
model = XGBRegressor(n_estimators=1500, learning_rate=0.05)
model.fit(X_train,
          y_train,
          verbose=False)

y_pred = model.predict(X_test)
print(f"{mean_absolute_error(y_test, y_pred)=}")

i_test_pred_ratio_top_k = [(index, f"{y_test:.2f}", f"{y_pred:.2f}", y_test/y_pred) for (i

# push predictions into dataframe and join to og table
df_outlier_cars_top_k = pd.DataFrame(i_test_pred_ratio_top_k, columns =['Index', 'True MSR
df_outlier_cars_top_k = df_outlier_cars_top_k.set_index('Index').join(df.set_index("Unname
df_outlier_cars_top_k.drop("MSRP", axis=1, inplace=True)
df_outlier_cars_top_k = df_outlier_cars_top_k.sort_values(by=["True:Predicted MSRP Ratio"]
df_outlier_cars_top_k
```

```
mean_absolute_error(y_test, y_pred)=20598.580751953126
```

Out[40]:

Index	True MSRP	Predicted MSRP	True:Predicted MSRP Ratio	Car_Make_Model_Style	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)
5885	33195.00	47957.96	0.692169	2018 Jeep Grand Cherokee Specs: Altitude 4x4	5	4	40.3	58.0	39.2
5874	33195.00	47957.96	0.692169	2018 Jeep Grand Cherokee Specs: Upland 4x4	5	4	40.3	58.0	39.2

Index	True MSRP	Predicted MSRP	True:Predicted MSRP Ratio	Car_Make_Model_Style	Passenger Capacity	Passenger Doors	Front Leg Room (in)	Second Shoulder Room (in)	Second Head Room (in)
2630	38995.00	56335.17	0.692196	2019 Dodge Challenger Specs: R/T Scat Pack Wid...	5	2	42.0	53.9	37.1
9560	325000.00	468571.38	0.693598	2019 Rolls-Royce Cullinan Specs: Sport Utility	5	4	0.0	0.0	0.0
5873	30895.00	44045.43	0.701435	2018 Jeep Grand Cherokee Specs: Altitude 4x2	5	4	40.3	58.0	39.2
...	...	...	...	...	...	...	...	...	...
9860	140000.00	101275.40	1.382369	2018 Tesla Model X Specs: P100D AWD	5	4	41.2	56.8	40.9
5615	165000.00	118558.55	1.391717	2016 Jaguar F-Type Specs: 2-Door Convertible A...	2	2	43.0	0.0	0.0
8770	293200.00	193417.28	1.515894	2018 Porsche 911 Specs: GT2 RS Coupe	2	2	0.0	0.0	0.0
5792	187500.00	119680.59	1.566670	2019 Jaguar XE SV Specs: Project 8 AWD	4	4	41.5	54.7	37.3
7283	227300.00	120390.77	1.888018	2018 Mercedes-Benz G Class Specs: G 550 4x4 Sq...	5	4	52.5	56.3	40.1

100 rows × 209 columns

The cell above shows our final rankings for the 100 cars we tested on.

Explaining the Model

We will now use SHAP to explain why the model predicted what it did compared to the model's baseline rate.

SHAP (SHapley Additive exPlanations) is a game-theoretic approach to explain the output of any machine learning model.

This method was developed in Economics to find optimal payments for games given a player's effort - this is analogous to feature importance on a prediction in an ML algorithm.

Please note that this is completely dependant on the model, this is not a ground truth!

It answers the question: **Why did my model predict what it did compare to it's average prediction?**

It does NOT answer the question: **Why did my model predict what it did?**

In [41]:
```
explainer = shap.TreeExplainer(model)

shap_values = explainer(X_test,
                        y=y_test,
                        check_additivity=True)

shap_i_test_pred_ratio_top_k = [(shap_index, index, true_msrp, pred_msrp, ratio) for shap_
shap_i_test_pred_ratio_top_k.sort(key=lambda t: t[4], reverse=True)
# shap_i_test_pred_ratio_top_k
```

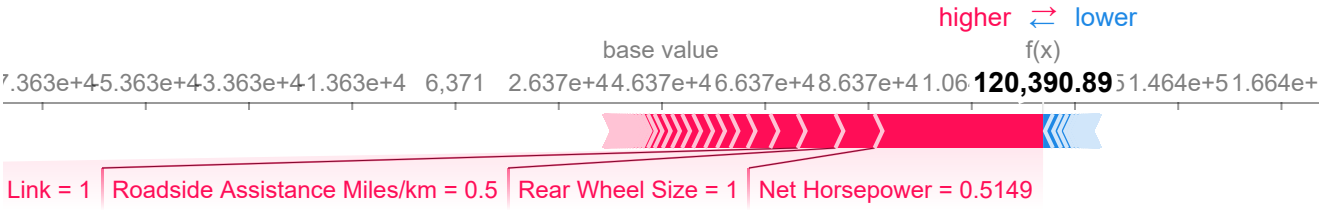
```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

Most Over-Priced Cars

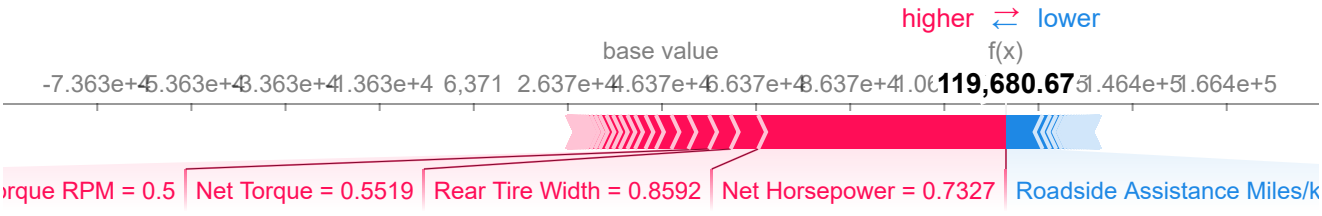
The below cell shows the top 3 most overpriced cars and their force plot. The force plot is a simple way of showing why the prediction is what it is starting from the mean prediction.

In [42]:
```
for shap_index, index, true_msrp, pred_msrp, ratio in shap_i_test_pred_ratio_top_k[:3]:
    name_of_car = df.loc[index]["Car_Make_Model_Style"]
    # print(f"Car = {name_of_car}, \nTrue MSRP = {true_msrp}, \nPredicted MSRP = {pred_msr
    print(f"\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t{name_of_car}")
    plt = shap.force_plot(shap_values[shap_index])
    display(plt)
```

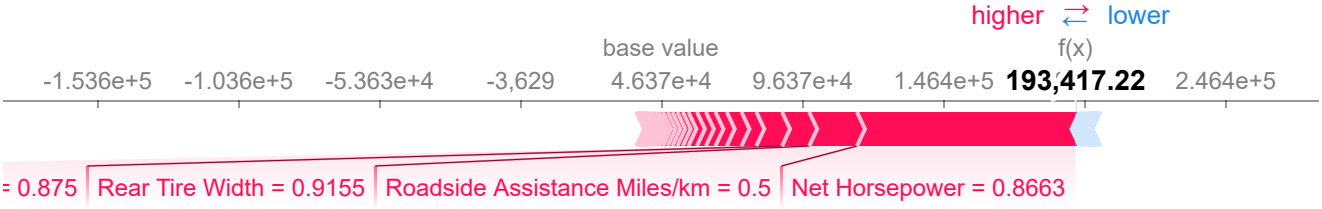
2018 Mercedes-Benz G Class Specs: G 550 4x4 Square
d SUV



2019 Jaguar XE SV Specs: Project 8 AWD



2018 Porsche 911 Specs: GT2 RS Coupe



The Single Most Over-Priced Car,

2018 Mercedes-Benz G Class Specs: G 550 4x4 Squared SUV

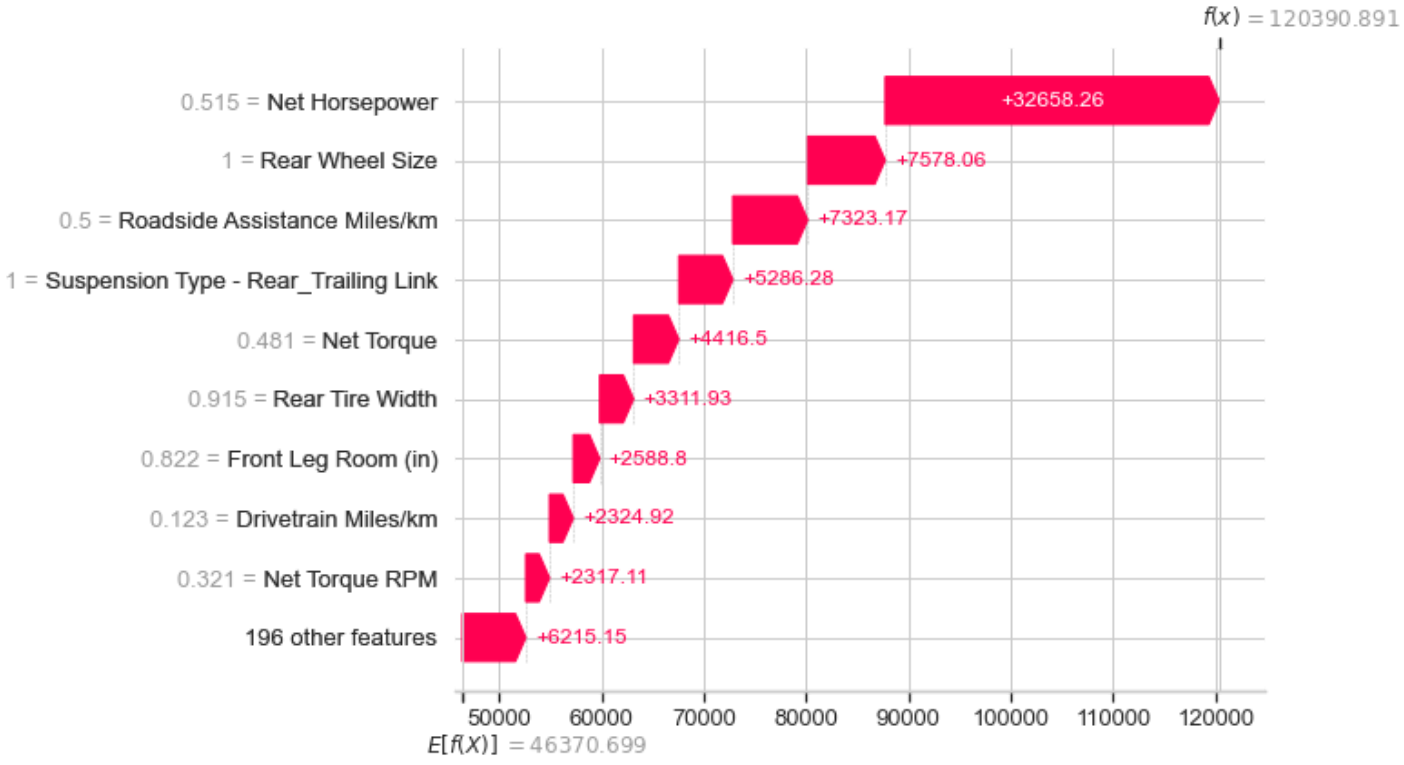
The below cell shows an expanded force plot or waterfall plot of the most overpriced car in the dataset.

We see that Net Horsepower was the largest contributing factor to the G-Wagon's price, followed by its Real Wheel Size.

This is the 4x4 model which some consider a "monster-truck" as it is very high off the ground, we saw earlier that Wheel Size is heavily correlated with a car's MSRP and this car has huge wheels.

In [43]:
```
shap_index, index, true_msrp, pred_msrp, ratio = shap_i_test_pred_ratio_top_k[0]
name_of_car = df.loc[index]["Car_Make_Model_Style"]
print(f"Car = {name_of_car}, \nTrue MSRP = {true_msrp}, \nPredicted MSRP = {pred_msrp}, \r
shap.plots.waterfall(shap_values[shap_index])
```

```
Car = 2018 Mercedes-Benz G Class Specs: G 550 4x4 Squared SUV,
True MSRP = 227300.00,
Predicted MSRP = 120390.77,
Ratio (True/Predicted) = 1.888018437874736
```



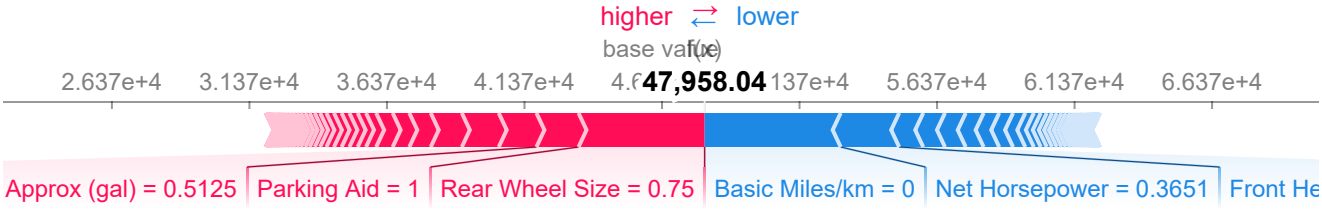
Most Under-Priced Cars

Similarly, the below cell shows the top 3 most under-priced cars and their respective force plots.

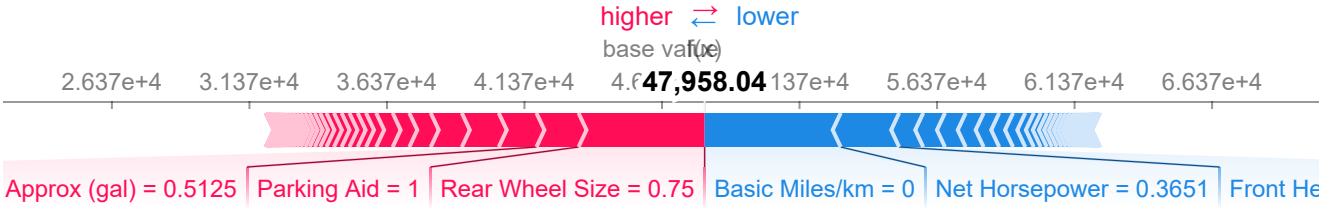
In [44]:

```
for shap_index, index, true_msrp, pred_msrp, ratio in shap_i_test_pred_ratio_top_k[-3:][::
    name_of_car = df.loc[index]["Car_Make_Model_Style"]
    print(f"Car = {name_of_car}, \nTrue MSRP = {true_msrp}, \nPredicted MSRP = {pred_msrp}
    # print(f"\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t{name_of_car}")
    plt = shap.force_plot(shap_values[shap_index])
    display(plt)
```

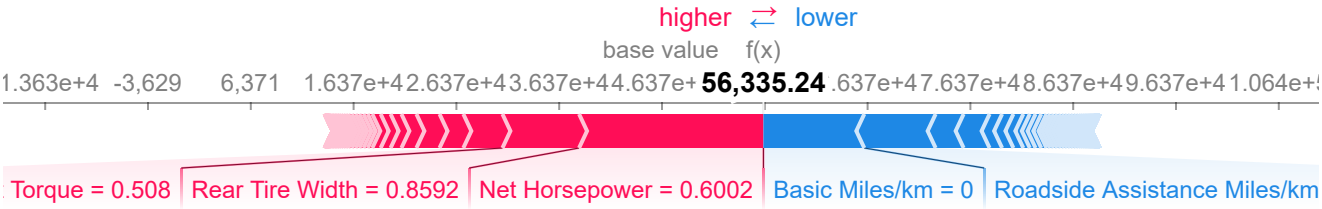
```
Car = 2018 Jeep Grand Cherokee Specs: Altitude 4x4,
True MSRP = 33195.00,
Predicted MSRP = 47957.96,
Ratio (True/Predicted) = 0.6921687672886009
```




```
Car = 2018 Jeep Grand Cherokee Specs: Upland 4x4,
True MSRP = 33195.00,
Predicted MSRP = 47957.96,
Ratio (True/Predicted) = 0.6921687672886009
```




```
Car = 2019 Dodge Challenger Specs: R/T Scat Pack Widebody RWD,
True MSRP = 38995.00,
Predicted MSRP = 56335.17,
Ratio (True/Predicted) = 0.6921963918103721
```



The Single Most Over-Priced Car,

2018 Jeep Grand Cherokee Specs: Altitude 4x4

We can see the waterfall plot of the single most underpriced car in the dataset. Unusually, the car's price is most affected by Basic Miles/km.

This was one of the columns which were set to 0 if it had a missing value, perhaps this may have played an important role in skewing some of these cars e.g. Cullinan.

The Jeep is also a large car, with its rear wheels falling at the 75th percentile, which should've driven up its price.

Overall, the reasons for this car being cheap are not so clear, however, the Jeep brand is again a car that is well known for its value for money and versatility.

In [45]:
```
shap_index, index, true_msrp, pred_msrp, ratio = shap_i_test_pred_ratio_top_k[-1]
name_of_car = df.loc[index]["Car_Make_Model_Style"]
print(f"Car = {name_of_car}, \nTrue MSRP = {true_msrp}, \nPredicted MSRP = {pred_msrp}, \r
shap.plots.waterfall(shap_values[shap_index])
```

```
Car = 2018 Jeep Grand Cherokee Specs: Altitude 4x4,
True MSRP = 33195.00,
Predicted MSRP = 47957.96,
Ratio (True/Predicted) = 0.6921687672886009
```