In [1]:

```python
from IPython.display import Image, Latex
Image(filename='declar.jpg',width=900)
```

Out[1]:

Declaration

By including this statement, we the authors of this work, verify that:

- We hold a copy of this assignment that we can produce if the original is lost or damaged.
- We hereby certify that no part of this assignment/product has been copied from any other student's work or from any other source except where due acknowledgement is made in the assignment.
- No part of this assignment/product has been written/produced for us by another person except where such collaboration has been authorised by the subject lecturer/tutor concerned.
- We are aware that this work may be reproduced and submitted to plagiarism detection software programs for the purpose of detecting possible plagiarism (which may retain a copy on its database for future plagiarism checking).
- We hereby certify that we have read and understand what the School of Computer, Data and Mathematical Sciences defines as minor and substantial breaches of misconduct as outlined in the learning guide for this unit.

In [2]:

```python
Image(filename='title.jpg',width=1000)
```

Out[2]:

WESTERN SYDNEY
UNIVERSITY

Modeling of CO Emissions from Cars

Christine Al-Thifairy (97074755)

Elisabeth Grasia Putri (611410005)

Kin Man Lam (15823898)

Assignment 2 for COMP7023 Predictive Analytics

School of Computer, Data and Mathematical Sciences,

Western Sydney University

Spring, 2022

Modeling of CO Emissions From Cars

1 The Dataset

This assignment was developed based on the dataset provided by Vehicle Certification Agency / UK which includes 41 car manufacturers.

The dataset is uploaded to our group GitHub repo under the file name "Euro_6_latest_07-10-2022.zip".

Most of data are based on the new European driving cycle, Worldwide harmonized Light vehicles Test Procedure (WLTP).

The data is applied to the analysis of emissions including CO, THC, NOx, THC + NO, Particulates, and Noise emissions. Our focus in this report will be on CO emissions.

2 Evaluation Metrics

We will evaluate the performance of the models using the following metrics:

- MSE (Mean Square Error) to evaluate regression models.
- Confusion Matrix, and Accuracy for binary classification model.

3 Data Preprocessing

3.1 - Data Cleaning

First we import Python libraries and read the dataset.

In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold

# read dataset file

df = pd.read_csv('Euro_6_latest.csv', delimiter=',', encoding='ISO-8859-1')
```

First, let's check the dim of the dataset.

In [4]:
```
print('Dataset dim is ',df.shape)
```

```
Dataset dim is  (4657, 44)
```

In [5]:
```
%%html
<style>
  xtable {margin-left: 0 !important;}
  table.a td, table.a th {text-align: left !important}
  table.a th:first-child, table.a td:first-child {width: 25%}
</style>
```

Initial cleaning includes remove unwanted columns including:

1- Drop any column with more than 70% missing values

2- Drop the following unnecessary columns:

Columns	**Reason**
Transmission	This is a low-level category of car transmissions with 39 categories. We are going to use 'Manual or Automatic' which is a high-level category of vehicle transmissions with only 3 values.
Euro Standard Testing Scheme Date of change	Those columns are not relevant to prediction and won't help the model learning new trends
WLTP Imperial Low WLTP Imperial Medium WLTP Imperial High WLTP Imperial Extra High WLTP Imperial Combined WLTP Imperial Combined (Weighted)	Imperial measurements, we are using metric measurements.
Diesel VED Supplement	This is only relevant to Diesel cars
Emissions NOx [mg/km] THC Emissions [mg/km] Noise Level dB(A)	These features are targets (emissions). We are only predicting CO emission, therefore no need to keep the others.

In [6]:
```
# Remove a funny column called Unnamed
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

# Remove features with more than 70% missing values
limitPer = len(df) * .70
df = df.dropna(thresh=limitPer, axis=1).copy()


# drop unnecessary columns
df = df.drop(['Transmission','Euro Standard','Testing Scheme','Date of change
              'WLTP Imperial Medium','WLTP Imperial High','WLTP Imperial Ext
              'WLTP Imperial Combined','WLTP Imperial Combined (Weighted)' ,
               'Diesel VED Supplement','Emissions NOx [mg/km]',
              'THC Emissions [mg/km]','Noise Level dB(A)'], axis=1).copy()
df.shape
```

Out[6]: (4657, 18)

Next, we give 'Manual or Automatic' column a new name 'Transmission' which is less verbos.

First, We look at the labels in 'Manual or Automatic' feature.
There are three labels in this column:

- 'Manual'
- 'Automatic'
- 'Electric - Not Applicable'

We going to create a new column called *Transmission* with all values from *'Manual or Automatic'* column. However, we replace the label *'Electric - Not Applicable'* with one world *'Electric'* which is less verbos.

In [7]:

```
# Create a new column 'Transmission', assign it a value of Automatic where th
# first character of Transmission is A
# or Manual if the first letter is M or Electric if first letter is E
AUTOMATIC = "Automatic"
MANUAL = "Manual"
ELECTRIC = "Electric"
df.loc[df['Manual or Automatic'].str.startswith('A'),'Transmission'] = AUTOMA
df.loc[df['Manual or Automatic'].str.startswith('M'),'Transmission'] = MANUAL
df.loc[df['Manual or Automatic'].str.startswith('E'),'Transmission'] = ELECTR
```

3.2 - Merge Low-Level Categories To Upper-Level Categories

To simplify the encoding process and reduce number of columns, we are going to merge few labels together in both *Fuel Type* and *Powertrain* columns.

First, let's examin the original labels in both columns:

In [8]:

```
print('Fuel Type labels are:\n\n',df['Fuel Type'].unique(),'\n')
print('Powertrain labels are \n\n',df['Powertrain'].unique())
```

```
Fuel Type labels are:

 ['Petrol' 'Diesel' 'Electricity / Petrol' 'Petrol Electric' 'Electricity'
 'Petrol / LPG' 'Diesel Electric' 'Electricity / Diesel']

Powertrain labels are

 ['Internal Combustion Engine (ICE)'
 'Plug-in Hybrid Electric Vehicle (PHEV)'
 'Mild Hybrid Electric Vehicle (MHEV)'
 'Battery Electric Vehicle (BEV) / Pure Electric Vehicle / Electric Vehicle (E
V)'
 'Hybrid Electric Vehicle (HEV)' 'Micro Hybrid']
```

Fuel Type Column:

A new column called just *Fuel* to replace the original *Fuel Type* is added with only four labels **Pertrol, Diesel, Hybrid,** and **Electric**.

- Petrol, Petrol / LPG ---> Petrol
- Diesel ---> Diesel
- Electricity ---> Electric
- Electricity / Petrol, Petrol Electric, Diesel Electric, Electricity / Diesel ---> hybrid

Powertrain Column:

A new column called *PT* to replace the original Powertrain column with only 3 labels **ICE, EV, and Hybrid**

- Internal Combustion Engine (ICE) ---> ICE
- Plug-in Hybrid Electric Vehicle (PHEV), Mild Hybrid Electric Vehicle (MHEV), Hybrid Electric Vehicle (HEV)','Micro Hybrid ---> Hybrid
- Battery Electric Vehicle (BEV) / Pure Electric Vehicle / Electric Vehicle (EV) ---> EV

The new labels are less verbos and easy to encode.

In [9]:
```
pertrol = ['Petrol','Petrol / LPG']
diesel = ['Diesel']
electric = ['Electricity']
hybrid = ['Electricity / Petrol','Petrol Electric','Diesel Electric','Electri
#df = df.copy()
df.loc[df['Fuel Type'].isin(pertrol), 'Fuel'] = 'Petrol'
df.loc[df['Fuel Type'].isin(diesel), 'Fuel'] = 'Diesel'
df.loc[df['Fuel Type'].isin(electric), 'Fuel'] = 'Electric'
df.loc[df['Fuel Type'].isin(hybrid), 'Fuel'] = 'Hybrid'
```

In [10]:
```
ice = ['Internal Combustion Engine (ICE)']
hybrid = ['Plug-in Hybrid Electric Vehicle (PHEV)',
          'Mild Hybrid Electric Vehicle (MHEV)',
          'Hybrid Electric Vehicle (HEV)','Micro Hybrid']
ev = ['Battery Electric Vehicle (BEV) / Pure Electric Vehicle / Electric Vehi

df.loc[df['Powertrain'].isin(ice), 'PT'] = 'ICE'
df.loc[df['Powertrain'].isin(ev), 'PT'] = 'EV'
df.loc[df['Powertrain'].isin(hybrid), 'PT'] = 'Hybrid'
```

And we drop the original columns Powertrain, Fuel Type, and Manual or Automatic.

In [11]:
```
# Drop the old columns
df = df.drop(['Powertrain','Fuel Type','Manual or Automatic'], axis=1)
```

And rename PT back to Powertrain

In [12]:
```
df.rename(columns={'PT':'Powertrain'}, inplace=True)
```

3.3 - Handle Missing Values And Zeros

We check missing values in each of the remaining columns

In [13]:

```
for (columnName, columnData) in df.iteritems():
   print(columnName, ' Num of empty cells : ', columnData.isnull().sum())
```

```
Manufacturer  Num of empty cells :  0
Model  Num of empty cells :  0
Description  Num of empty cells :  0
Engine Capacity  Num of empty cells :  2
Engine Power (PS)  Num of empty cells :  221
Engine Power (Kw)  Num of empty cells :  89
WLTP Metric Low  Num of empty cells :  5
WLTP Metric Medium  Num of empty cells :  5
WLTP Metric High  Num of empty cells :  5
WLTP Metric Extra High  Num of empty cells :  6
WLTP Metric Combined  Num of empty cells :  15
WLTP Metric Combined (Weighted)  Num of empty cells :  1000
WLTP CO2  Num of empty cells :  2
WLTP CO2 Weighted  Num of empty cells :  1209
Emissions CO [mg/km]  Num of empty cells :  108
Transmission  Num of empty cells :  0
Fuel  Num of empty cells :  0
Powertrain  Num of empty cells :  0
```

Engine Capacity has two missing values which correspond to rows of Electic cars. Most likely because it is not applicable for Electric cars. So, we replace those missing vlaues with zero.

In [14]:

```
df['Engine Capacity'] = df['Engine Capacity'].fillna(0)
```

Engine Power (PS) column has 221 missing values. We will use ffill (forward filling) and bfill (backward filling) per group to fill the missing values of Engine Power (PS). We group by car manufacturer and Model. So, the function will search for another car from same manufacturer and same model and has value in that column and copies that value to the other one with missing value.

Engine Power (Kw) has 89 missing values. We apply the same method above to fill the missing values.

In [15]:

```
df['Engine Power (PS)'] = df.groupby(['Manufacturer','Model'], sort=False) \
    ['Engine Power (PS)'].apply(lambda x: x.ffill().bfill())

# Same with Engine Power (Kw)
df['Engine Power (Kw)'] = df.groupby(['Manufacturer','Model'], sort=False) \
    ['Engine Power (Kw)'].apply(lambda x: x.ffill().bfill())
```

Some of the features has many zeros. For example, *WLTP Metric Combined (Weighted)* and *WLTP CO2*.

Let's count how many zeros in WLTP Metric Combined (Weighted) and WLTP CO2 Weighted columns

In [16]:

```
print('WLTP Metric Combined (Weighted) zeros =', len(df) - np.count_nonzero(d
print('WLTP CO2 Weighted zeros =', len(df) - np.count_nonzero(df["WLTP CO2 We
```

```
WLTP Metric Combined (Weighted) zeros = 3472
WLTP CO2 Weighted zeros = 3098
```

There are too many zeros values in both columns. So we are going to drop those 2 columns

In [17]:

```
df = df.drop(['WLTP Metric Combined (Weighted)','WLTP CO2 Weighted'], axis=1)
```

For the rest of the features, we are going to fill missing values with mean of each column.

We did not replace 0 values with the mean because that is not correct, for example electric cars may have 0 CO emissions.

In [18]:

```
df['WLTP Metric Combined'].fillna((df['WLTP Metric Combined'].mean()), inplac
df['WLTP Metric Low'].fillna((df['WLTP Metric Low'].mean()), inplace=True)
df['WLTP Metric Medium'].fillna((df['WLTP Metric Medium'].mean()), inplace=Tr
df['WLTP Metric High'].fillna((df['WLTP Metric High'].mean()), inplace=True)
df['WLTP Metric Extra High'].fillna((df['WLTP Metric Extra High'].mean()), in
df['WLTP CO2'].fillna((df['WLTP CO2'].mean()), inplace=True)
df['Emissions CO [mg/km]'].fillna((df['Emissions CO [mg/km]'].mean()), inplac
df.isnull().sum(axis = 0)
```

Out[18]:

```
Manufacturer              0
Model                     0
Description               0
Engine Capacity           0
Engine Power (PS)       177
Engine Power (Kw)        89
WLTP Metric Low           0
WLTP Metric Medium        0
WLTP Metric High          0
WLTP Metric Extra High    0
WLTP Metric Combined      0
WLTP CO2                  0
Emissions CO [mg/km]      0
Transmission              0
Fuel                      0
Powertrain                0
dtype: int64
```

There are still 177 missing values in Engine Power (PS) and 89 in Engine Power (Kw). We looked at the original dataset csv file. It turns out those car models have no data. Therefore we are going to drop those models.

In [19]:

```
df = df.dropna().copy()
```

Finally, let's check the size of the dataset after all the preprocessing steps above.

In [20]:

```
print('The dim of the dataset is ',df.shape, ' and original dim is (4657, 19)
```

```
The dim of the dataset is  (4467, 16)  and original dim is (4657, 19)
```

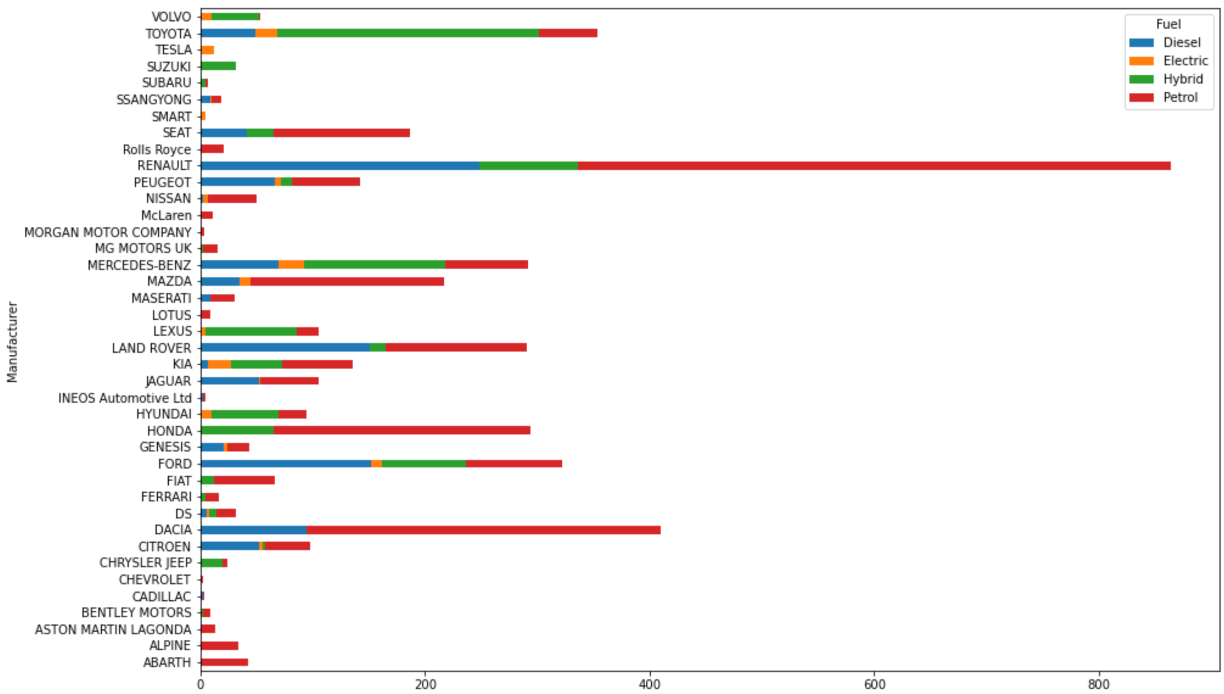
4 Data Exploration

4.1 - Total Cars By Car Manufacturers

First we plot number of cars in the dataset by manufacturers and type of fuel. There are more models of some cars than others, especially the european models. It is obvious that Petrol still the dominant type of petrol. However, many car manufacturers are producing hybrid models.

In [21]:

```
from IPython.core.display import display, HTML
display(HTML("<style>div.output_scroll { height: 44em; }</style>"))
plt.rcParams["figure.figsize"] = (15,10)
cols = ['Manufacturer', 'Fuel']
df1 = df[cols].copy()
df1.groupby(['Manufacturer','Fuel']).size().unstack().plot(sort_columns='Manu
        kind='barh', stacked=True,width=0.5,linewidth=0.5);
```



4.2 - CO Emission By Car Manufacturer

Next, we plot the CO emission value against each car. Some of the cars show higher than others. Sometimes, this is because of number of models used for each cars is higher than other.

Fiat and Lotus are generating more CO emission than other cars in the dataset.

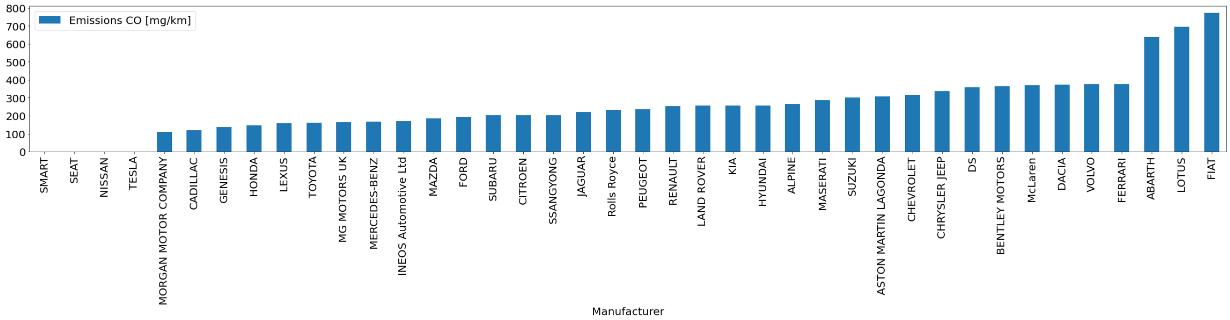
Note: Nissan CO Emission figures are recorded as zero in the original dataset.

In [22]:

```
cols = ['Manufacturer', 'Emissions CO [mg/km]']
plt.rcParams["figure.figsize"] = (40,5)

df2 = df[cols].copy()
df2 = df2.groupby('Manufacturer', as_index=False)['Emissions CO [mg/km]'].mea
```

```python
df2.sort_values('Emissions CO [mg/km]',inplace=True)
df2.plot(kind='bar',x='Manufacturer',y='Emissions CO [mg/km]',fontsize=20)
plt.legend(fontsize = 20)
plt.xlabel('Manufacturer', fontsize=20)
```

Out[22]:
```
Text(0.5, 0, 'Manufacturer')
```

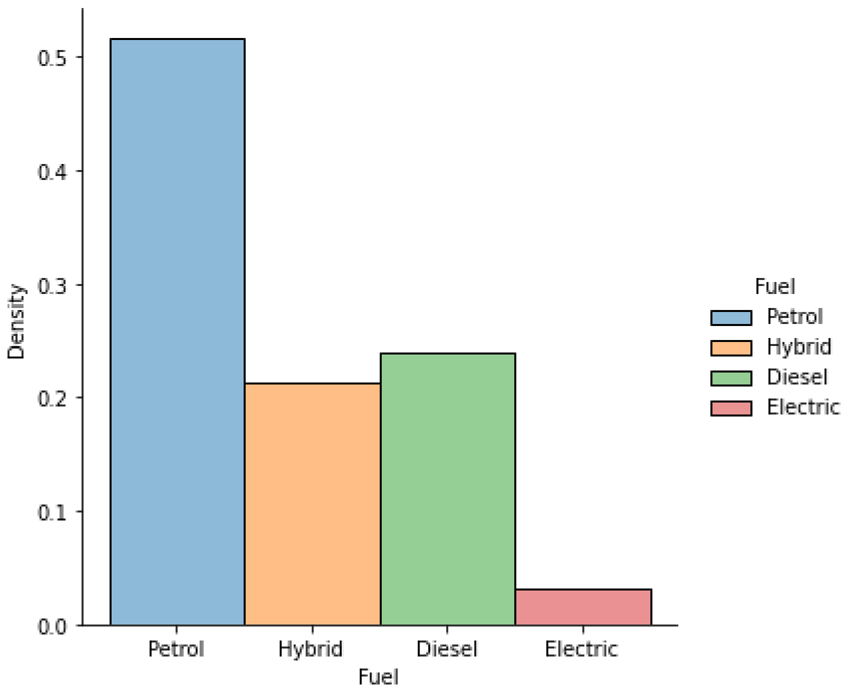


4.3 - Percentage Of Cars By Fuel Type

About half of the cars in the dataset are running on petrol.

In [23]:
```python
sns.displot(data=df1, x="Fuel",stat='density',kind='hist', hue=df['Fuel'])
```

Out[23]:
```
<seaborn.axisgrid.FacetGrid at 0x7f7abac88d90>
```



4.4 - Pairplot

Next, we look at the pairplot of some of the main features. It is clear that hybrid and electric cars act like outliners which is because of their low emissons and engine size related figures. Generally speaking, the dataset has many outliners.</br>

It is also clear petrol and diesel have higher WLTP figures than the rest.

In [24]:
```python
sns.pairplot(df,vars=['Engine Capacity','Engine Power (PS)','Engine Power (Kw
                      'WLTP Metric Low','WLTP Metric Extra High','WLTP CO2',
                      'Emissions CO [mg/km]'],hue='Fuel',
                       diag_kind='hist')
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x7f7abac55cd0>

4.5 - Mean and STD

Note how each feature covers a very different range, therefore we need to normalise the dataset when building the prediction models.

In [25]:

```
df.describe().transpose()[['mean', 'std']]
```

Out[25]:

	mean	std
Engine Capacity	1744.287665	894.872988
Engine Power (PS)	175.893217	109.439872
Engine Power (Kw)	131.782628	83.890608
WLTP Metric Low	8.307146	4.909390
WLTP Metric Medium	6.399704	4.837777
WLTP Metric High	5.861178	4.419764
WLTP Metric Extra High	7.056735	3.763594
WLTP Metric Combined	6.737001	4.058259

	mean	std
WLTP CO2	151.679009	57.911991
Emissions CO [mg/km]	232.946411	197.198821

5 Encoding

Let's look at the result df and see how many categorical features we have:

In [26]:

```
for column in df:
    print("{} | {} | {}".format(
        df[column].name, len(df[column].unique()), df[column].dtype
    ))
```

```
Manufacturer | 40 | object
Model | 305 | object
Description | 2138 | object
Engine Capacity | 88 | float64
Engine Power (PS) | 196 | float64
Engine Power (Kw) | 179 | float64
WLTP Metric Low | 240 | float64
WLTP Metric Medium | 168 | float64
WLTP Metric High | 132 | float64
WLTP Metric Extra High | 138 | float64
WLTP Metric Combined | 146 | float64
WLTP CO2 | 246 | float64
Emissions CO [mg/km] | 422 | float64
Transmission | 3 | object
Fuel | 4 | object
Powertrain | 3 | object
```

There are three categorical features that we are going to include in our analysis. These are:

- Transmission
- Fuel
- Powertrain

We are using get_dummies() for encoding those features.

We also going to drop Manufacturer, Model, and Description. They are not useful for prediction.

In [27]:

```
dummy_cols = ['Transmission','Fuel','Powertrain']
df_encode = pd.get_dummies(df, columns=dummy_cols).copy()
df_encode = df_encode.drop(['Manufacturer','Model','Description'], axis=1).co
```

And finally we check data types do we have and dim of our final dataset.

In [28]:

```
print(df_encode.dtypes)
print()
print('Dataset dim is ',df_encode.shape)
```

```
Engine Capacity            float64
Engine Power (PS)          float64
Engine Power (Kw)          float64
WLTP Metric Low            float64
WLTP Metric Medium         float64
WLTP Metric High           float64
WLTP Metric Extra High     float64
```

```
WLTP Metric Combined      float64
WLTP CO2                  float64
Emissions CO [mg/km]      float64
Transmission_Automatic      uint8
Transmission_Electric       uint8
Transmission_Manual         uint8
Fuel_Diesel                 uint8
Fuel_Electric               uint8
Fuel_Hybrid                 uint8
Fuel_Petrol                 uint8
Powertrain_EV               uint8
Powertrain_Hybrid           uint8
Powertrain_ICE              uint8
dtype: object

Dataset dim is  (4467, 20)
```

6 Prediction Models

6.1 - SVM Model - Binary Classification

We build a binary classification model to predict if the car produce high or low CO emission. </br>

The new binary target classes are 0 or 1. Any sample data above the mean of the original target will belong to class 0 and considered as high (bad). On the other hand, those below its mean are 1 and considered as low (good).

In [29]:

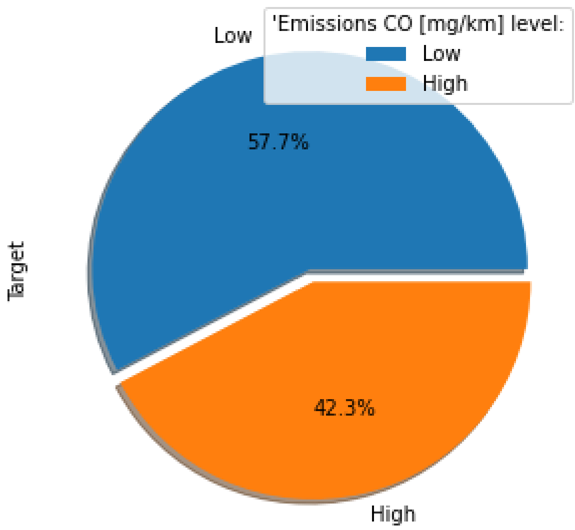
```
# Mean of the target value.
m = df_encode['Emissions CO [mg/km]'].mean()

# make a copy of df_encode
df_svm = df_encode.copy()
df_svm['Target'] = (df_svm['Emissions CO [mg/km]'] <= m).astype('int64')
```

Plot Percentages Of Classes

In [30]:

```
df_svm['Target'].value_counts() / df_svm['Target'].count()
y_labels = ["Low", "High"]
df_svm['Target'].value_counts().plot.pie(explode=[0, 0.06],
                                         figsize=(5,5),
                                         labels = y_labels,
                                         autopct = '%1.1f%%', shadow=True)
# The pie plot shows proportion of each class of the target value.
plt.legend(title = "'Emissions CO [mg/km] level:")
```

Out[30]: `<matplotlib.legend.Legend at 0x7f7ac1280220>`



Define Inputs And Target Variables

Our target variable is the new feature Target which contains 2 classes 0 and 1.
All other numeric features are our input variables.

In [31]:
```
y_2 = df_svm['Target']
X_2 = df_svm.drop(['Emissions CO [mg/km]','Target'],axis=1).copy()
print('Our SVM Input Features\n')
X_2.dtypes
```

```
Our SVM Input Features
```

Out[31]:
```
Engine Capacity             float64
Engine Power (PS)           float64
Engine Power (Kw)           float64
WLTP Metric Low             float64
WLTP Metric Medium          float64
WLTP Metric High            float64
WLTP Metric Extra High      float64
WLTP Metric Combined        float64
WLTP CO2                    float64
Transmission_Automatic        uint8
Transmission_Electric         uint8
Transmission_Manual           uint8
Fuel_Diesel                   uint8
Fuel_Electric                 uint8
Fuel_Hybrid                   uint8
Fuel_Petrol                   uint8
Powertrain_EV                 uint8
Powertrain_Hybrid             uint8
Powertrain_ICE                uint8
dtype: object
```

Splitting The Dataset Into Train And Test

Split train set and test set into 80% to 20%.

In [32]:
```
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_2, y_2, test_si
```

Build SVM Model

We are using MinMaxScaler for normalisation of both train and test data.</br>

In [33]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
import sklearn.metrics as metrics

#MinMax normalization
scaler = MinMaxScaler()

# Scalling the data
X_train_2 = scaler.fit_transform(X_train_2)
X_test_2 = scaler.transform(X_test_2)
```

Check feature scalling

In [34]:
```
df_describe_2 = pd.DataFrame(X_train_2)
df_describe_2.describe().transpose()
```

Out[34]:

	count	mean	std	min	25%	50%	75%	max
0	3573.0	0.258063	0.131438	0.0	0.197066	0.236776	0.295896	1.0
1	3573.0	0.220430	0.136284	0.0	0.142500	0.181250	0.250000	1.0
2	3573.0	0.216014	0.136122	0.0	0.137255	0.174837	0.240196	1.0
3	3573.0	0.148184	0.087956	0.0	0.106952	0.135472	0.163993	1.0
4	3573.0	0.101402	0.069361	0.0	0.077901	0.092210	0.111288	1.0
5	3573.0	0.082307	0.063586	0.0	0.064426	0.075630	0.088235	1.0
6	3573.0	0.123290	0.067509	0.0	0.102967	0.116928	0.137871	1.0
7	3573.0	0.110240	0.067885	0.0	0.086601	0.102941	0.119281	1.0
8	3573.0	0.402077	0.154681	0.0	0.336870	0.384615	0.464191	1.0
9	3573.0	0.620767	0.485264	0.0	0.000000	1.000000	1.000000	1.0
10	3573.0	0.006437	0.079985	0.0	0.000000	0.000000	0.000000	1.0
11	3573.0	0.372796	0.483616	0.0	0.000000	0.000000	1.000000	1.0
12	3573.0	0.238735	0.426370	0.0	0.000000	0.000000	0.000000	1.0
13	3573.0	0.034985	0.183767	0.0	0.000000	0.000000	0.000000	1.0
14	3573.0	0.213826	0.410063	0.0	0.000000	0.000000	0.000000	1.0
15	3573.0	0.512455	0.499915	0.0	0.000000	1.000000	1.000000	1.0
16	3573.0	0.034985	0.183767	0.0	0.000000	0.000000	0.000000	1.0
17	3573.0	0.240134	0.427225	0.0	0.000000	0.000000	0.000000	1.0
18	3573.0	0.724881	0.446637	0.0	0.000000	1.000000	1.000000	1.0

Now we have the data ready for the model.

Create a SVM classifier and use linear as kernel as it is the most common one. First we make prediction with train data, and we also going to look at the accuracy of the model by printing the metrics report and the confusion matrix.

In [35]:

```
svm_1 = svm.SVC(kernel='linear',C=1)

# Fit the model
svm_1.fit(X_train_2, y_train_2)

# Make predictions on train dataset
y_pred_2 = svm_1.predict(X_train_2)
```

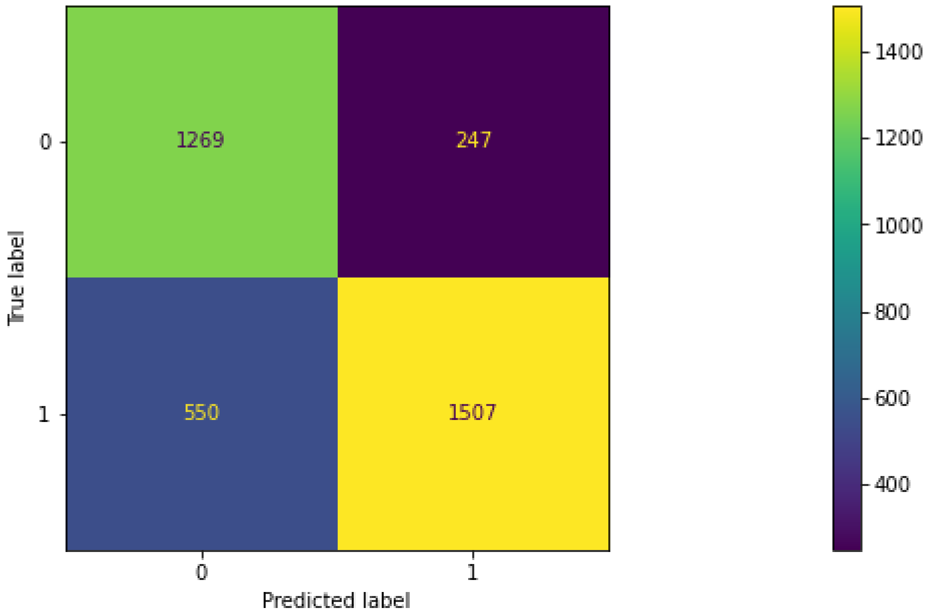
Model Evaluation

10-fold cross validation is used here to check the accuracy of the model.

In [36]:

```
accuracies = cross_val_score(svm_1, X_train_2, y_train_2, cv=10)
print("Train Score:", np.mean(accuracies))
print("confusion_matrix:", confusion_matrix(y_train_2, y_pred_2))
print(metrics.classification_report(y_train_2, y_pred_2, digits=2))
plot_confusion_matrix(svm_1, X_train_2, y_train_2)
plt.show()
```

```
Train Score: 0.774972223526282
confusion_matrix: [[1269  247]
 [ 550 1507]]
              precision    recall  f1-score   support

           0       0.70      0.84      0.76      1516
           1       0.86      0.73      0.79      2057

    accuracy                           0.78      3573
   macro avg       0.78      0.78      0.78      3573
weighted avg       0.79      0.78      0.78      3573
```

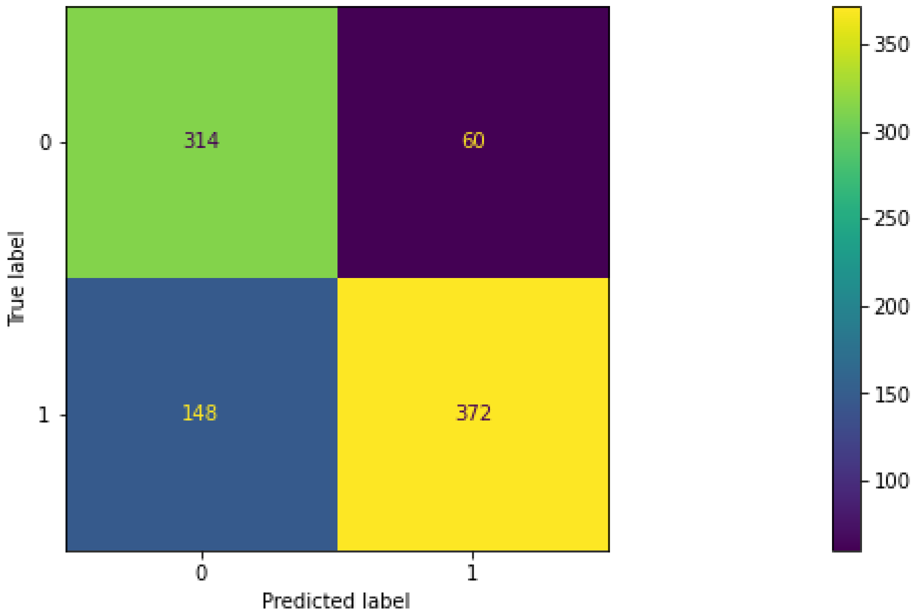


Average score for training data is 77%.
Precision figure for class 0 is 70% and for class 1 is 86% which is not bad. Out of 1516 sample with class 0, 1269 are correct and out of 2057 of class 1, 1507 are correct.

Let's test accuracy with test dataset.

In [37]:
```
y_pred_3 = svm_1.predict(X_test_2)

#Accuracies of the model. 10-fold cross validation is used here.
accuracies2 = cross_val_score(svm_1, X_test_2, y_test_2, cv=10)
print("Test Score:", np.mean(accuracies2))
print("confusion_matrix:", confusion_matrix(y_test_2, y_pred_3))
print(metrics.classification_report(y_test_2, y_pred_3, digits=2))
plot_confusion_matrix(svm_1, X_test_2, y_test_2)
plt.show()
```

```
Test Score: 0.7393008739076155
confusion_matrix: [[314  60]
 [148 372]]
              precision    recall  f1-score   support

           0       0.68      0.84      0.75       374
           1       0.86      0.72      0.78       520

    accuracy                           0.77       894
   macro avg       0.77      0.78      0.77       894
weighted avg       0.79      0.77      0.77       894
```



Average score of test data is 74% which is less than training data. It is normal to have less test score than train score.

precision figures are close with 68% for class 0 and 86% for class 1.

The score is not that great and that could be because there are many outliners in our dataset as you can see from the pairplot figure above.

We are going to use GridSearchCV to find the best hyperparameters C, kernel, degree, and gamma.

GridSearchCV Model

GridSearchCV is used here to search the best parameters for the SVM model.

In [38]:

```
grid = {
    'C':[0.01, 0.1, 1, 10],
    'kernel' : ["linear", "poly", "rbf", "sigmoid"],
    'degree' : [1, 3, 5, 7],
    'gamma' : [0.01, 1]
}

svm_2  = SVC(random_state = 125)
svm_cv = GridSearchCV(svm_2, grid, cv = 10)
svm_cv.fit(X_train_2, y_train_2)

print("Best Parameters:", svm_cv.best_params_)
print("Accuracy on train data", svm_cv.best_score_)
# Print the accuracy on the test data
print("Accuracy on test data:", svm_cv.score(X_test_2, y_test_2))
```

```
Best Parameters: {'C': 10, 'degree': 7, 'gamma': 1, 'kernel': 'poly'}
Accuracy on train data 0.8804868316041501
Accuracy on test data: 0.8780760626398211
```

We use the best parameters for prediction and print the classification report.

In [39]:

```
grid_predictions = svm_cv.predict(X_test_2)

print(classification_report(y_test_2, grid_predictions))
```

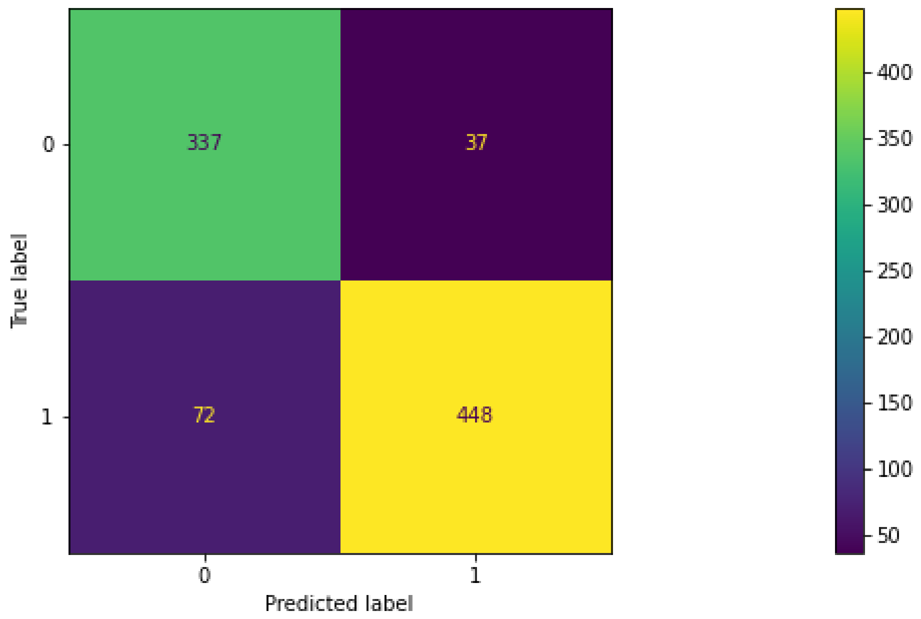
```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       374
           1       0.92      0.86      0.89       520

    accuracy                           0.88       894
   macro avg       0.87      0.88      0.88       894
weighted avg       0.88      0.88      0.88       894
```

The precision figures for both classes are better than the linear SVM model. Class 0 was 68% now it is 82%. Class 1 was 86% and now it is 92%.

Use best parameters for prediction

In [40]:
```
svm_3 = SVC(kernel='poly', degree=7, gamma=1, C=10)
svm_3.fit(X_train_2, y_train_2)
y_pred_2_1 = svm_3.predict(X_test_2)

print("Test Accuracy:\n", accuracy_score(y_test_2, y_pred_2_1))
print("confusion_matrix:\n", confusion_matrix(y_test_2, y_pred_2_1))
print(metrics.classification_report(y_test_2, y_pred_2_1, digits=2))
plot_confusion_matrix(svm_3, X_test_2, y_test_2)
plt.show()
```

```
Test Accuracy:
 0.8780760626398211
confusion_matrix:
 [[337  37]
 [ 72 448]]
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       374
           1       0.92      0.86      0.89       520

    accuracy                           0.88       894
   macro avg       0.87      0.88      0.88       894
weighted avg       0.88      0.88      0.88       894
```



Accuracy score for the new polynomial model has also improved. Out of 374 data samples with class 0, only 37 were misclassified. And out of 520 sample with class 1, only 72 were wrongly classified as 0.

Overall, this binary SVM model is good and produces good results.

6.2 - Neural Network Regression Model

Build the model using Keras deep learning API https://keras.io/

Identify The Input And Target Variables

Our target variable (y) is 'Emissions CO [mg/km]' which is continuous variable. So our NN model is regression.

We use all the numeric features for the prediction models as input variables or X.

In [41]:

```
X = df_encode.loc[:, df_encode.columns != 'Emissions CO [mg/km]']
print()
print('X or input variables are:')
print()
print(X.columns.tolist())
y =  df_encode['Emissions CO [mg/km]']
```

```
X or input variables are:

['Engine Capacity', 'Engine Power (PS)', 'Engine Power (Kw)', 'WLTP Metric Lo
w', 'WLTP Metric Medium', 'WLTP Metric High', 'WLTP Metric Extra High', 'WLTP
Metric Combined', 'WLTP CO2', 'Transmission_Automatic', 'Transmission_Electri
c', 'Transmission_Manual', 'Fuel_Diesel', 'Fuel_Electric', 'Fuel_Hybrid', 'Fue
l_Petrol', 'Powertrain_EV', 'Powertrain_Hybrid', 'Powertrain_ICE']
```

Spliting The Data

In [42]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=.2)
```

Define Keras Sequential Model

The model is using build-in Adam() optimiser.
For measuring the losses, we use mean_square_error metrics.

We are using sklearn.preprocessing.StandardScaler library to scale our train and test data.

Our model has seven hidden layers and one output layer with linear activation function.

In [43]:

```
from keras.models import Sequential
#from keras import utils
import keras
from keras.layers import Dense, Activation
import scikeras
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score
from keras.optimizers import SGD
from keras import initializers
from keras.layers import LeakyReLU

numerics = ['uint8','int16', 'int32', 'int64', 'float16', 'float32', 'float64

X_train_chris = X_train.select_dtypes(include=numerics).to_numpy()
X_test_chris = X_test.select_dtypes(include=numerics).to_numpy()
```

```
y_train_chris = Y_train.to_numpy()
y_test_chris = Y_test.to_numpy()

#Standard normalization
sc = StandardScaler()
X_train_chris = sc.fit_transform(X_train_chris)
X_test_chris = sc.transform(X_test_chris)
y_train_chris = sc.fit_transform(y_train_chris.reshape(len(y_train_chris),1))
y_test_chris = sc.transform(y_test_chris.reshape(len(y_test_chris),1))[:,0]

dim = X_train_chris.shape[1]

# define the keras model
model = Sequential()

# First layer with inputs
model.add(Dense(64, input_shape=(dim,),activation='relu'))

# Second hidden layer
model.add(Dense(32, activation='relu'))

# Third hidden layer
model.add(Dense(132, activation='relu'))

# Fourth hidden layer
model.add(Dense(32))
model.add(LeakyReLU(alpha=0.1))

# Fifth hidden layer
model.add(Dense(32))
model.add(LeakyReLU(alpha=0.1))

# Sixth hidden layer
model.add(Dense(16))
model.add(LeakyReLU(alpha=0.1))

# Seventh hidden layer
model.add(Dense(4))
model.add(LeakyReLU(alpha=0.1))

# Output layer is linear
model.add(Dense(1, activation='linear'))

opt = keras.optimizers.Adam(learning_rate=0.001)
# compile the keras model with adam optimise

model.compile(loss='mean_squared_error', optimizer=opt , metrics='mean_square
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1280

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 132)               4356

 dense_3 (Dense)             (None, 32)                4256

 leaky_re_lu (LeakyReLU)     (None, 32)                0

 dense_4 (Dense)             (None, 32)                1056
```

```
 leaky_re_lu_1 (LeakyReLU)   (None, 32)              0

 dense_5 (Dense)             (None, 16)              528

 leaky_re_lu_2 (LeakyReLU)   (None, 16)              0

 dense_6 (Dense)             (None, 4)               68

 leaky_re_lu_3 (LeakyReLU)   (None, 4)               0

 dense_7 (Dense)             (None, 1)               5

=================================================================
Total params: 13,629
Trainable params: 13,629
Non-trainable params: 0
_________________________________________________________________
```

Fit The Model

In [44]:

```
history = model.fit(X_train_chris, y_train_chris,
        validation_data=(X_test_chris, y_test_chris), epochs=1200, verbose=0)
```

Evaluate The Model

We are using MSE as metric to measure the accurracy of our NN model.

In [45]:

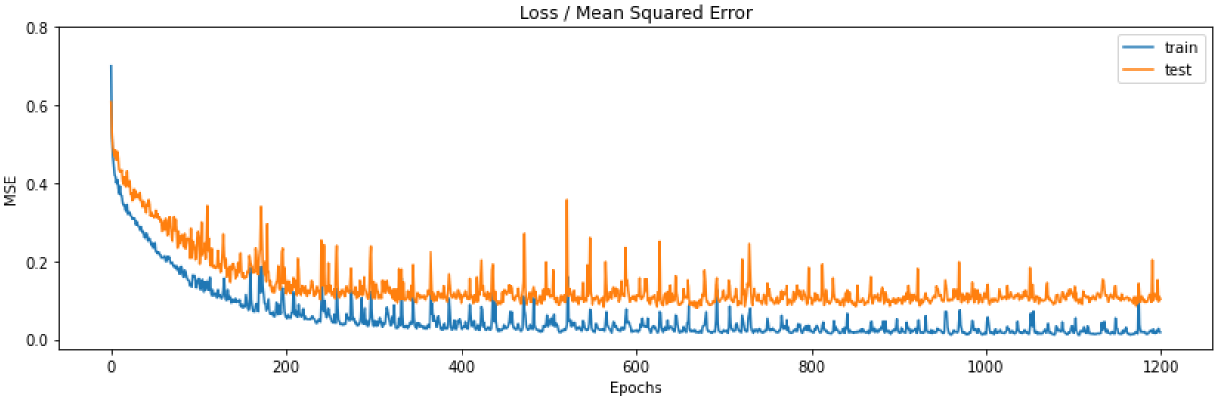
```
train_mse, train_accuracy = model.evaluate(X_train_chris, y_train_chris, verb
test_mse, test_accuracy = model.evaluate(X_test_chris, y_test_chris, verbose=
print('Train MSE: %.5f, Test MSE: %.5f' % (train_mse, test_mse))
```

```
Train MSE: 0.01596, Test MSE: 0.10274
```

Plot Train And Test Losses

In [46]:

```
plt.figure(figsize=(14,4))
plt.title('Loss / Mean Squared Error')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('MSE')
plt.xlabel('Epochs')
plt.yticks(np.arange(0, np.max(history.history['loss'])+0.2, step=0.2))
plt.legend()
plt.show()
```



Keras Model Evaluation

The Loss curves for both training and test data are fluctuating and not smooth. This could be because we have relatively small dataset and little large network (13,629 parameters).

Our train MSE is on average 1% while the test MSE is 8%-17%. It is normal to training MSE lower than test MSE.

This model prediction accuracy is very high and therefore it is a good model.

6.3 - Lasso Regression Model

First, we import the libraries we are going to use to build the Lasso regression model.

In [47]:
```
from sklearn.linear_model import Lasso, LassoCV
from sklearn.preprocessing import scale
from sklearn.model_selection import RepeatedKFold
from sklearn.metrics import mean_squared_error
```

Creating the Training and Test Datasets

In [48]:
```
X_train_lasso, X_test_lasso, y_train_lasso, y_test_lasso = train_test_split(X
```

Data Normalisation

We use StandardScaler() function for scalling the datasets.

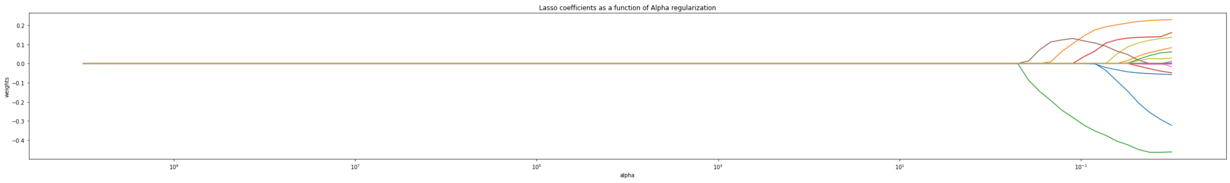
In [49]:
```
sc = StandardScaler()
y_train_lasso  = y_train_lasso.to_numpy()
y_test_lasso = y_test_lasso.to_numpy()
X_train_lasso = sc.fit_transform(X_train_lasso)
X_test_lasso = sc.transform(X_test_lasso)
y_train_lasso = sc.fit_transform(y_train_lasso.reshape(len(y_train_lasso),1))
y_test_lasso = sc.transform(y_test_lasso.reshape(len(y_test_lasso),1))[:,0]
```

Exploring L1 Penalty Values

First, we investigate the size of each feature weight as function of alpha.

In [50]:
```
alphas = 10**np.linspace(10,-2,100)*0.5

# Define the model
lasso = Lasso()
coefs = []

for a in alphas*2:
    lasso.set_params(alpha=a)
    lasso.fit(X_train_lasso, y_train_lasso)
    coefs.append(lasso.coef_)

ax = plt.gca()
ax.plot(alphas*2, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.axis('tight')
plt.xlabel('alpha')
```

```
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of Alpha regularization');
```



Moving from the left to right in the plot, at first, the coefficient estimates approximate towards zero. Then the model starts to have more predictors with high magnitudes of coefficient estimates.

Selecting Optimal Alpha Value

Next, we need to find the optimal value of alpha to use in our model.

use LassoCV function to fit the regression model and find the optimal alpha. We also use RepeatedKFold() to evaluate the lasso model.

We will define a range for alpha from 0 to 1 with increment of 0.01

In [51]:

```
# RepeatedKFold for evaluation of the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

# Define the model
lassocv = LassoCV(alphas=np.arange(0.01, 1, 0.01), cv=cv, n_jobs=-1)

# Fit the model
lassocv.fit(X_train_lasso, y_train_lasso)

# Best alpha
best_alpha = lassocv.alpha_
print('Best lambda that produced the lowest test MSE = ', best_alpha)
print()
print('Coefficients of the model are \n')
pd.Series(lassocv.coef_, index=X.columns)
```

```
Best lambda that produced the lowest test MSE =  0.01

Coefficients of the model are
```

Out[51]:

```
Engine Capacity           -0.323045
Engine Power (PS)          0.082488
Engine Power (Kw)          0.060479
WLTP Metric Low            0.161265
WLTP Metric Medium        -0.000000
WLTP Metric High          -0.000000
WLTP Metric Extra High    -0.016670
WLTP Metric Combined      -0.000000
WLTP CO2                   0.135967
Transmission_Automatic    -0.000000
Transmission_Electric     -0.057536
Transmission_Manual        0.229278
Fuel_Diesel               -0.461778
Fuel_Electric             -0.048394
Fuel_Hybrid                0.009915
Fuel_Petrol                0.000000
Powertrain_EV             -0.003320
Powertrain_Hybrid         -0.000000
Powertrain_ICE             0.028399
dtype: float64
```

We can see few of the coefficients are zero value.

Build The Model

Now we are going to use the best lambda in our model for prediction

In [52]:
```
lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train_lasso, y_train_lasso)
y_pred = lasso.predict(X_train_lasso)
```

Model Evaluation

In [53]:
```
MSE = mean_squared_error(y_train_lasso, y_pred)
print("Train MSE = " ,round(MSE,2) )

y_pred_t = lasso.predict(X_test_lasso)
MSE_test = mean_squared_error(y_test_lasso, y_pred_t)
print("Test MSE = " ,round(MSE_test,2) )
```

```
Train MSE =  0.63
Test MSE =  0.66
```

The model scores 0.63 on the training dataset and 0.67 on the test dataset. The difference is very small but they indicate the model is not really good one.

7 Conclusion and Comparison

We implemented 3 prediction models for this assignment, SVM (binary classification), Neural Network (regression), and Lasso (regression) using the Euro_6_latest.csv dataset to predict CO Emission.

The SVM and NN models performed well and show good results while the Lasso regression model showed very average result. SVM accuracy were 88% on the train data and 87% on the test data.

The NN model MSE were around 2% on the train data and 10%-18% on the test data on average which is very good considering the dataset features have many outliners.

The last model, Regression Lasso did badly. The MSE were 63% for training and 67% for test data. We could not reduce the MSE even when choosing the best alpha.