Importing Libraries

In [1]:
```
import pandas as pd #Data Manipulation and Analysis
import matplotlib.pyplot as plt #plotting
# magic command
%matplotlib widget
import numpy as np #for working with arrays
import seaborn as sns #interactive data visulaization base on matplotlib
import warnings
warnings.filterwarnings('ignore')
import re
from time import time #Timing our operations
import collections
from collections import defaultdict
import spacy #spaCy is a free, open-source library for NLP in Python.
from gensim.models import Word2Vec #NLP functionality
import logging
logging.basicConfig(format = "%(levelname)s - %(asctime)s: %(message)s",datefmt = '%H:
from sklearn.manifold import TSNE #tool to visualize high dimensional data
from numpy import dot #dotproduct
from numpy.linalg import norm #linear algebra ...matrix norms
```

Dataset Overview

This dataset contains different attributes like Make, Model, Year, Engine Fuel Type etc. for different car models. The attributes containing text will be used to predict the similarity using NLP between the different car models. To compare the similarity word embeddings using Gensim is used.

In [2]:
```
#Data Set Import
df = pd.read_csv('data.csv')
df.head()
```

Out[2]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Tuner,Lu Pe
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Pe
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Lu Pe
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Pe
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	

In [3]:

```
print('Shape of initial dataset:', df.shape)
```

```
Shape of initial dataset: (11914, 16)
```

Dataset Preprocessing

Gensim uses list of lists for its working. For this purpose the make and model of the car are combined into one and then other features will also be clubbed in respective list. Once, list for all individual cars are completed, all these lists will be combined into one single list and further processing is done.

In [4]:

```
#New column for combined make and model is created
df['Maker_Model'] = df['Make']+" "+df['Model']
```

In [5]:

```
print(df.shape)
df.head()
```

```
(11914, 17)
```

Out[5]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Tuner,Lu Pe
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Pe
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Lu Pe
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Pe
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	

In [6]:

```
#All the columns containing text are chosen and put in a new dataframe df1
df1 = df[['Engine Fuel Type','Transmission Type','Driven_Wheels','Market Category','Ve
print(df1.shape)
df1.head()
```

(11914, 7)

Out[6]:

	Engine Fuel Type	Transmission Type	Driven_Wheels	Market Category	Vehicle Size	Vehicle Style	Maker_Model
0	premium unleaded (required)	MANUAL	rear wheel drive	Factory Tuner,Luxury,High-Performance	Compact	Coupe	BMW 1 Series M
1	premium unleaded (required)	MANUAL	rear wheel drive	Luxury,Performance	Compact	Convertible	BMW 1 Series
2	premium unleaded (required)	MANUAL	rear wheel drive	Luxury,High-Performance	Compact	Coupe	BMW 1 Series
3	premium unleaded (required)	MANUAL	rear wheel drive	Luxury,Performance	Compact	Coupe	BMW 1 Series
4	premium unleaded (required)	MANUAL	rear wheel drive	Luxury	Compact	Convertible	BMW 1 Series

In [7]:

```
#All the columns combined into one column in df2 dataframe
df2 = df1.apply(lambda x: ','.join(x.astype(str)),axis = 1)
print(df2.shape)
df2.head()
```

(11914,)

Out[7]:
```
0    premium unleaded (required),MANUAL,rear wheel ...
1    premium unleaded (required),MANUAL,rear wheel ...
2    premium unleaded (required),MANUAL,rear wheel ...
3    premium unleaded (required),MANUAL,rear wheel ...
4    premium unleaded (required),MANUAL,rear wheel ...
dtype: object
```

In [8]:
```
#a new pandas dataframe is created of name df_clean containing column clean
df_clean = pd.DataFrame({'clean':df2})
df_clean.head()
```

Out[8]:

	clean
0	premium unleaded (required),MANUAL,rear wheel ...
1	premium unleaded (required),MANUAL,rear wheel ...
2	premium unleaded (required),MANUAL,rear wheel ...
3	premium unleaded (required),MANUAL,rear wheel ...
4	premium unleaded (required),MANUAL,rear wheel ...

In [9]:
```
df_clean.shape
```

Out[9]:
```
(11914, 1)
```

In [10]:
```
#List of list data corpus for Gensim modelling
sent = [row.split(',') for row in df_clean['clean']]
sent[:2]
```

Out[10]:
```
[['premium unleaded (required)',
  'MANUAL',
  'rear wheel drive',
  'Factory Tuner',
  'Luxury',
  'High-Performance',
  'Compact',
  'Coupe',
  'BMW 1 Series M'],
 ['premium unleaded (required)',
  'MANUAL',
  'rear wheel drive',
  'Luxury',
  'Performance',
  'Compact',
  'Convertible',
  'BMW 1 Series']]
```

Model Training

Word Embedding is implemented using Word2Vec technique. It uses two-layer neural network. Input for this is text and output is a set of vectors. Gensim library on the custom corpus is implemented using algorithms like CBOW(Continuous Bag of Words), SG(Skip Gram). Training model is created as below.

In [11]:
```
model = Word2Vec(sent,min_count =1,vector_size=50, workers = 3,window=3,sg=1)
```

1. size: The number of dimensions of the embeddings and the default is 100.
2. window: The maximum distance between a target word and words around the target word. The default window is 5.
3. min_count: The minimum count of words to consider when training the model; words with occurrence less than this count will be ignored. The default for min_count is 5.
4. workers: The number of partitions during training and the default workers is 3.
5. sg: The training algorithm, either CBOW(0) or skip gram (1). The default training alogrithm is CBOW.

In [12]:
```
model.wv['Toyota Camry']
```

Out[12]:
```
array([ 0.02502831,  0.12598681,  0.08280787, -0.09878854, -0.07040344,
       -0.22827502,  0.00432529,  0.2873881 , -0.11326703, -0.07166937,
        0.01355   , -0.00968716,  0.08243357, -0.02705853, -0.05206136,
        0.1628918 ,  0.14276648,  0.2796947 , -0.09449822, -0.25770095,
       -0.09563645, -0.02092389,  0.23033041,  0.05986825,  0.17628908,
        0.0127508 , -0.06394614,  0.35839027, -0.01284829, -0.00500502,
       -0.02917308,  0.0225873 ,  0.06831939,  0.00116526,  0.07883739,
       -0.11637811,  0.16327074, -0.07241298,  0.01589195,  0.06132088,
        0.08909579, -0.02413157, -0.21030034,  0.09350082,  0.3142694 ,
        0.00854845, -0.01886506, -0.13280734, -0.01651406,  0.03528447],
      dtype=float32)
```

Compare Similarities

Euclidean Similarity

In [13]:
```
print('Similarity b/w Porsche 718 Cayman & Nissan Van',model.wv.similarity('Porsche 71
print('Similarity b/w Porsche 718 Cayman & Mercedes-Benz SLK-Class',model.wv.similarit
```

```
Similarity b/w Porsche 718 Cayman & Nissan Van 0.77489597
Similarity b/w Porsche 718 Cayman & Mercedes-Benz SLK-Class 0.93039864
```

In [14]:
```
print('Top 5 similar to Mercedes-Benz SLK-Class: \n',model.wv.most_similar(positive=['
```

```
Top 5 similar to Mercedes-Benz SLK-Class: 
 [('BMW ALPINA B7', 0.9870179295539856), ('Lamborghini Aventador', 0.986897110939025
9), ('Mercedes-Benz CLK-Class', 0.9863013625144958), ('Mercedes-Benz SLS AMG GT', 0.9
856127500534058), ('Lamborghini Murcielago', 0.9855102300643921)]
```

In [15]:
```
print('Top 5 similar to Toyota Camry: \n',model.wv.most_similar(['Toyota Camry'], topr
```

```
Top 5 similar to Toyota Camry: 
 [('Chevrolet Malibu', 0.9885684251785278), ('Oldsmobile Eighty-Eight Royale', 0.9875
094294548035), ('Toyota Avalon', 0.9817401766777039), ('Dodge Dart', 0.98140937089920
04), ('Mazda 6', 0.9799979329109192)]
```

Cosine Similarity

[Ref 1] Euclidian similarity cannot work well for the high-dimensional word vectors, This is because Euclidian similarity will increase the number of dimensions increases even if the word embedding stands for different meanings. Alternatively, we can use cosine similarity to measure the similarity between two vectors. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. Therefore, the cosine similarity captures the angle of the word vectors and not the magnitude. Under cosine similarity, no similarity is expressed as a 90-degree angle while the total similarity of 1 is at 0 degree angle.

In [16]:
```
def cosine_distance(model, word, target_list, num):
    cosine_dict = {}
    word_list = []
    a = model.wv[word]
    for item in target_list:
        if item != word:
            b = model.wv[item]
            cos_sim = dot(a,b)/(norm(a)*norm(b))
            cosine_dict[item] = cos_sim
    dist_sort = sorted(cosine_dict.items(), key = lambda dist: dist[1], reverse = True
    for item in dist_sort:
        word_list.append((item[0], item[1]))
    return word_list[0:num]
```

In [17]:
```
Maker_Model = list(df.Maker_Model.unique())
cosine_distance(model,'Mercedes-Benz SLK-Class', Maker_Model, 5)
```

Out[17]:
```
[('BMW ALPINA B7', 0.9870179),
 ('Lamborghini Aventador', 0.98689705),
 ('Mercedes-Benz CLK-Class', 0.98630136),
 ('Mercedes-Benz SLS AMG GT', 0.9856128),
 ('Lamborghini Murcielago', 0.98551023)]
```

T-SNE Plot

[Ref 1] T-SNE is an useful tool to visualize high-dimensional data by reducing dimensional space while keeping relative pairwise distance between points. It can be said that t-SNE looking for a new data representation where the neighborhood relations are preserved. T-SNE is an useful tool to visualize high-dimensional data by reducing dimensional space while keeping relative pairwise distance between points. It can be said that t-SNE looking for a new data representation where the neighborhood relations are preserved.

In [18]:
```
def display_closestwowords_tsnescatterplot(model, word,size):
    arr = np.empty((0, size), dtype = 'f')
    word_labels = [word]

    close_words = model.wv.similar_by_word(word)

    arr = np.append(arr, np.array([model.wv[word]]), axis = 0)
    for wrd_score in close_words:
        wrd_vector = model.wv[wrd_score[0]]
        word_labels.append(wrd_score[0])
        arr = np.append(arr, np.array([wrd_vector]), axis = 0)

    tsne = TSNE(n_components = 2, random_state = 0)
```

```python
    np.set_printoptions(suppress = True)
    Y = tsne.fit_transform(arr)

    x_coords = Y[:, 0]
    y_coords = Y[:, 1]
    plt.scatter(x_coords, y_coords)

    for label, x, y in zip(word_labels, x_coords, y_coords):
        plt.annotate(label, xy = (x, y), xytext = (0,0), textcoords = 'offset points')
    plt.xlim(x_coords.min()+0.00005, x_coords.max()+0.00005)
    plt.ylim(y_coords.min()+0.00005, y_coords.max()+0.00005)
    plt.show()
```

In [19]:

```python
%matplotlib widget
display_closestwowords_tsnescatterplot(model, 'Porsche 718 Cayman', 50)
```

Figure