In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib notebook
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import re  # For preprocessing
import pandas as pd  # For data handling
from time import time  # To time our operations
from collections import defaultdict  # For word frequency

import spacy  # For preprocessing
from gensim.models import Word2Vec
import logging  # Setting up the loggings to monitor gensim
logging.basicConfig(format="%(levelname)s - %(asctime)s: %(message)s", datefmt=
'%H:%M:%S', level=logging.INFO)
from sklearn.manifold import TSNE
from numpy import dot
from numpy.linalg import norm
```

In [2]:

```
df = pd.read_csv('cardata.csv')
df.head()
```

Out[2]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Nu
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.(
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.(
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.(
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.(
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.(

In [ ]:

```
# Genism word2Vec creates a list of list for training where every document is a
 list and every list is a set of tokens of that document
```

In [3]:

```
#1)Create a new column for Make Model because each make-model is contained in a
 list and every list contains list of features of that make-model.
#2)Generate a list of list for following features: Engine Fuel Type, Transmissio
n Type, Driven_Wheels, Market Category, Vehicle Size and Vehicle Style.

# tip to create a column from the dataframe :  df['column_Name']
df['Maker_Model']= df['Make']+ " " + df['Model']
df_make_model = df['Maker_Model']
df_make_model.head()
```

Out[3]:

```
0    BMW 1 Series M
1      BMW 1 Series
2      BMW 1 Series
3      BMW 1 Series
4      BMW 1 Series
Name: Maker_Model, dtype: object
```

In [4]:

```
# Select features from original dataset to form a new dataframe
df1 = df[['Engine Fuel Type','Transmission Type','Driven_Wheels','Market Categor
y','Vehicle Size', 'Vehicle Style', 'Maker_Model']]

# For each row, combine all the columns into one column where values under each
 column is separated by comma
df2 = df1.apply(lambda x: ','.join(x.astype(str)), axis=1)#apply() works in all
 rows
```

In [5]:

```
df2.head()
```

Out[5]:

```
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 [6]:

```
# Store them in the pandas dataframe with a single column named: clean
df_clean = pd.DataFrame({'clean': df2})
df_clean.head()
```

Out[6]:

	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 [7]:

```
# Create the list of list
sent = [row.split(',') for row in df_clean['clean']]#column names is clean

#only the 1st 2 lists are displayed inside the list
sent[:2]
```

Out[7]:

```
[['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']]
```

In [8]:

```
# Word2vec Model Training using Genism
model = Word2Vec(sent, min_count=1,size= 50,workers=3, window =5, sg = 1)

#size: The number of dimensions of the embeddings and the default is 100.
#window: maximum distance between a target word and words around the target wor
d. The default window size=5.
#min_count: The minimum count of words to consider when training the model; word
s with occurrence less than this count will be ignored. The default for min_coun
t is 5.
#workers(i.e. no of threads): The number of partitions during training and the d
efault workers is 3.
#sg: The training algorithm, either CBOW(0) or skip gram(1). The default trainin
g algorithm is CBOW.
```

```
WARNING - 23:19:10: consider setting layer size to a multiple of 4 f
or greater performance
INFO - 23:19:10: collecting all words and their counts
INFO - 23:19:10: PROGRESS: at sentence #0, processed 0 words, keepin
g 0 word types
INFO - 23:19:10: PROGRESS: at sentence #10000, processed 74060 word
s, keeping 841 word types
INFO - 23:19:10: collected 977 word types from a corpus of 88129 raw
words and 11914 sentences
INFO - 23:19:10: Loading a fresh vocabulary
INFO - 23:19:10: effective_min_count=1 retains 977 unique words (10
0% of original 977, drops 0)
INFO - 23:19:10: effective_min_count=1 leaves 88129 word corpus (10
0% of original 88129, drops 0)
INFO - 23:19:10: deleting the raw counts dictionary of 977 items
INFO - 23:19:10: sample=0.001 downsamples 35 most-common words
INFO - 23:19:10: downsampling leaves estimated 29996 word corpus (3
4.0% of prior 88129)
INFO - 23:19:10: estimated required memory for 977 words and 50 dime
nsions: 879300 bytes
INFO - 23:19:10: resetting layer weights
INFO - 23:19:10: training model with 3 workers on 977 vocabulary and
50 features, using sg=1 hs=0 sample=0.001 negative=5 window=5
INFO - 23:19:10: worker thread finished; awaiting finish of 2 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 1 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 0 more t
hreads
INFO - 23:19:10: EPOCH - 1 : training on 88129 raw words (29976 effe
ctive words) took 0.1s, 543392 effective words/s
INFO - 23:19:10: worker thread finished; awaiting finish of 2 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 1 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 0 more t
hreads
INFO - 23:19:10: EPOCH - 2 : training on 88129 raw words (30030 effe
ctive words) took 0.0s, 629840 effective words/s
INFO - 23:19:10: worker thread finished; awaiting finish of 2 more t
```

```
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 1 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 0 more t
hreads
INFO - 23:19:10: EPOCH - 3 : training on 88129 raw words (29999 effe
ctive words) took 0.1s, 530975 effective words/s
INFO - 23:19:10: worker thread finished; awaiting finish of 2 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 1 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 0 more t
hreads
INFO - 23:19:10: EPOCH - 4 : training on 88129 raw words (29935 effe
ctive words) took 0.1s, 569620 effective words/s
INFO - 23:19:10: worker thread finished; awaiting finish of 2 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 1 more t
hreads
INFO - 23:19:10: worker thread finished; awaiting finish of 0 more t
hreads
INFO - 23:19:10: EPOCH - 5 : training on 88129 raw words (30044 effe
ctive words) took 0.0s, 686973 effective words/s
INFO - 23:19:10: training on a 440645 raw words (149984 effective wo
rds) took 0.3s, 483086 effective words/s
```

In [9]:

```
## We can obtain the word embedding directly from the training model
model['Toyota Camry']

# below we see 5 cols and each col is 10 dim so total=50 dim
# below each value is similarity score between given word and corpus words.
```

Out[9]:

```
array([-0.06306811,  0.14448293,  0.2721026 , -0.01445359, -0.127415
42,
        0.07445106,  0.10775785, -0.11229065,  0.02546714, -0.145302
07,
        0.15541446,  0.05534331, -0.09064761,  0.12286652, -0.099303
54,
       -0.17074354,  0.0019181 , -0.04031203,  0.22337429, -0.053768
63,
       -0.24104367, -0.064581  , -0.00555336, -0.32584286, -0.175725
7 ,
        0.14712977, -0.060808  , -0.00722675,  0.01924545, -0.023145
5 ,
        0.04846632,  0.02611506,  0.20612359,  0.18661937, -0.017802
82,
       -0.15858209, -0.18877473, -0.08904456, -0.14509915, -0.118936
72,
        0.02974049,  0.12514849,  0.28386948,  0.20031066,  0.245452
26,
       -0.00678116, -0.01295323, -0.07010298,  0.08674996,  0.042982
1 ],
      dtype=float32)
```

In [10]:

```
# Compare Similarities using Euclidian distances
# model.similarity('Porsche 718 Cayman', 'Nissan Van') This will give us the Euc
lidian similarity between Porsche 718 Cayman and Nissan Van.

model.similarity('Porsche 718 Cayman', 'Nissan Van') # L2 norm
```

Out[10]:

0.9249482

In [11]:

```
# From the above example, we can tell that Porsche 718 Cayman is more similar wi
th Mercedes-Benz SLK-Class than Nissan Van.
model.similarity('Porsche 718 Cayman', 'Mercedes-Benz SLK-Class')
```

Out[11]:

0.9750005

In [12]:

```
#find 10 nearest neighbors w.r.t the similarity score:

## Show the most similar vehicles for Mercedes-Benz SLK-Class : Default by eculi
dean distance
model.most_similar('Mercedes-Benz SLK-Class')[:10]
```

```
INFO - 23:19:52: precomputing L2-norms of word weight vectors
```

Out[12]:

```
[('Maserati GranSport', 0.9972572326660156),
 ('Mercedes-Benz AMG GT', 0.9960204362869263),
 ('Porsche Boxster', 0.9957455992698669),
 ('Rolls-Royce Phantom Drophead Coupe', 0.995137631893158),
 ('Mercedes-Benz SL-Class', 0.9950829744338989),
 ('Ferrari F12 Berlinetta', 0.9947906732559204),
 ('Mercedes-Benz CLK-Class', 0.9945163130760193),
 ('Mercedes-Benz SLS AMG GT', 0.9943203330039978),
 ('Lotus Exige', 0.9942452311515808),
 ('Lamborghini Murcielago', 0.9940518140792847)]
```

In [13]:

```
# Differences between cosine similarity(distance) VS Euclidean similarity

#Euclidian similarity cannot work well for the high-dimensional word vectors.
#Because Euclidian similarity will increase as the number of dimensions increase
s.
#Cosine similarity measures the cosine of the angle between two vectors projecte
d in a n-dim vector space.
#The cosine similarity captures the angle of the 2 vectors.
```

In [14]:

```
# Code for cosine similarity to generate the most similar make model based on.

def cosine_distance (model, word,target_list , num) :
    cosine_dict ={}
    word_list = []
    a = model[word]
    for item in target_list :
        if item != word :
            b = model [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 = Tru
e) ## in Descedning order
    for item in dist_sort:
        word_list.append((item[0], item[1]))
    return word_list[0:num]
```

In [15]:

```
# only get the unique Maker_Model
>>> Maker_Model = list(df.Maker_Model.unique())
```

In [16]:

```
# Show the most similar Mercedes-Benz SLK-Class by cosine distance
>>> cosine_distance (model,'Mercedes-Benz SLK-Class',Maker_Model,5)
```

Out[16]:

```
[('Maserati GranSport', 0.99725723),
 ('Mercedes-Benz AMG GT', 0.9960205),
 ('Porsche Boxster', 0.9957455),
 ('Rolls-Royce Phantom Drophead Coupe', 0.99513745),
 ('Mercedes-Benz SL-Class', 0.9950829)]
```

In [17]:

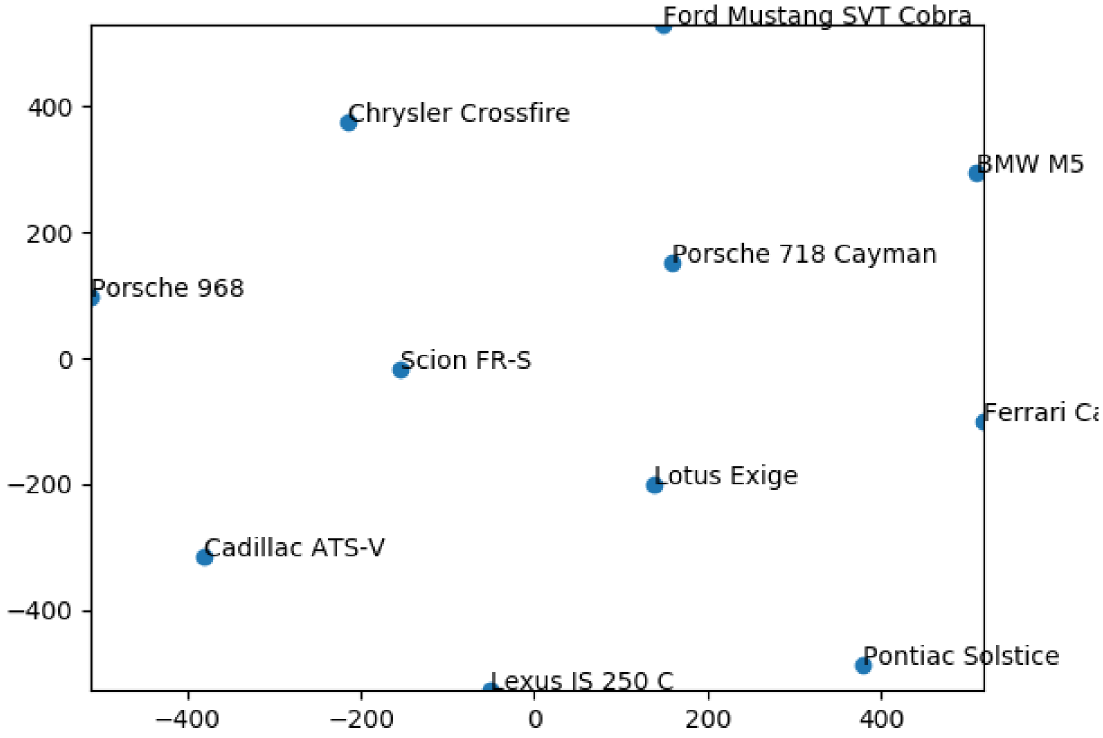
```
# T-SNE Visualizations

#It’s hard to visualize the word embedding directly, with more than 3 dimension
s.
#T-SNE visualize high-dimensional data by dimension reduction technique by keepi
ng relative pairwise distance between points.
#T-SNE looks for a new data representation and neighborhood relations are preser
ved.

#The following code shows how to plot the word embedding with T-SNE plot.
```

In [18]:

```
def display_closestwords_tsnescatterplot(model, word, size):

    arr = np.empty((0,size), dtype='f')
    word_labels = [word]

    close_words = model.similar_by_word(word)

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

```
>>> display_closestwords_tsnescatterplot(model, 'Porsche 718 Cayman', 50)
```



In [ ]:

```

```