In [1]:
```
from nltk.corpus.reader.reviews import Review
import re, nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import StratifiedKFold
from sklearn import metrics
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB

import umap
import plotly.graph_objs as go
import plotly.figure_factory as ff
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [2]:
```
df = pd.read_csv("Car_Reviews.csv", encoding = "ISO-8859-1") # You can also use '
pd.set_option('display.max_colwidth', None) # Setting this so we can see the full
pd.set_option('display.max_columns', None) # to make sure we can see all the colu


# Converting structured categorical features to numerical features
df['Recommend'] = df['Recommend'].map({'Yes':1, 'No':0})
```

In [3]:
```
def cleaner(review): # Cleaning Reviews Column
    soup = BeautifulSoup(review, 'lxml') # removing HTML entities such as ‘&amp’,
    souped = soup.get_text()
    re1 = re.sub(r"(#|$|œ|™|€|â|%|@|http://|https://|www|\\x)\S*", " ", souped) #
    re2 = re.sub("[^A-Za-z]+"," ", re1) # substituting any non-alphabetic charact


    tokens = nltk.word_tokenize(re2)
    lower_case = [t.lower() for t in tokens]

    stop_words = set(stopwords.words('english'))
    filtered_result = list(filter(lambda l: l not in stop_words, lower_case))

    wordnet_lemmatizer = WordNetLemmatizer()
    lemmas = [wordnet_lemmatizer.lemmatize(t) for t in filtered_result]
    return lemmas
```

In [4]:
```
df['cleaned_review'] = df.Review.apply(cleaner)
df = df[df['cleaned_review'].map(len) > 0] # removing rows with cleaned reviews c
print("Printing top 5 rows of dataframe showing original and cleaned reviews....'
print(df[['Review','cleaned_review']].head())
df['cleaned_review'] = [" ".join(row) for row in df['cleaned_review'].values] # j
data = df['cleaned_review']
Y = df['Recommend'] # target column
tfidf = TfidfVectorizer(min_df=.00281, ngram_range=(1,3)) # min_df=.00281 means t
tfidf.fit(data) # learn vocabulary of entire data
data_tfidf = tfidf.transform(data) # creating tfidf values
print("The created tokens: \n", tfidf.get_feature_names())
print("Shape of tfidf matrix: ", data_tfidf.shape)
print(type(data_tfidf))
```

```
'case', 'cash', 'catalytic', 'catalytic converter', 'catch', 'category', 'cau
ght', 'cause', 'caused', 'causing', 'cd', 'cd player', 'celica', 'celica gt',
'cell', 'center', 'center console', 'certain', 'certainly', 'certified', 'cha
in', 'chair', 'champ', 'chance', 'change', 'change oil', 'change tire', 'chan
ged', 'changing', 'charge', 'charged', 'charging', 'cheap', 'cheap car', 'che
ap plastic', 'cheaper', 'cheaply', 'cheaply made', 'check', 'check engine',
'check engine light', 'checked', 'cherokee', 'chevrolet', 'chevy', 'child',
'chip', 'choice', 'choose', 'chose', 'chrome', 'chrysler', 'city', 'city driv
ing', 'city highway', 'city hwy', 'city mpg', 'civic', 'claim', 'claimed', 'c
lass', 'class action', 'class action lawsuit', 'clean', 'clear', 'clearance',
'clearly', 'clicking', 'climate', 'climate control', 'climb', 'clock', 'clos
e', 'closed', 'closer', 'cloth', 'clunk', 'clunking', 'cluster', 'clutch', 'c
lutch transmission', 'co', 'coast', 'code', 'coil', 'cold', 'cold weather',
'college', 'collision', 'color', 'colorado', 'column', 'combination', 'combin
ed', 'come', 'come back', 'comfort', 'comfortable', 'comfortable drive', 'com
fortable long', 'comfortable ride', 'comfortable seat', 'comfortably', 'comf
y', 'coming', 'comment', 'common', 'common problem', 'commute', 'commuter',
'commuter car', 'commuting', 'compact', 'company', 'comparable', 'compare',
'compared', 'comparing', 'comparison', 'compartment', 'competition', 'competi
tor', 'complain', 'complained', 'complaining', 'complaint', 'complete', 'comp
```

In [5]:
```
print("Implementing SVC...")
model = LinearSVC() # kernel = 'linear' and C = 1

# Running cross-validation
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1) # 10-fold cross-v
scores=[]
iteration = 0
for train_index, test_index in kf.split(data_tfidf,Y):
    iteration += 1
    print("Iteration ", iteration)
    X_train, Y_train = data_tfidf[train_index], Y.iloc[train_index]
    X_test, Y_test = data_tfidf[test_index], Y.iloc[test_index]
    model.fit(X_train, Y_train) # Fitting SVC
    Y_pred = model.predict(X_test)
    score = metrics.accuracy_score(Y_test, Y_pred) # Calculating accuracy
    print("Cross-validation accuracy: ", score)
    scores.append(score) # appending cross-validation accuracy for each iteration
svc_mean_accuracy = np.mean(scores)
print("Mean cross-validation accuracy: ", svc_mean_accuracy)
```

```
Implementing SVC...
Iteration  1
Cross-validation accuracy:  0.9288389513108615
Iteration  2
Cross-validation accuracy:  0.9166666666666666
Iteration  3
Cross-validation accuracy:  0.9335205992509363
Iteration  4
Cross-validation accuracy:  0.903558052434457
Iteration  5
Cross-validation accuracy:  0.9138576779026217
Iteration  6
Cross-validation accuracy:  0.9147940074906367
Iteration  7
Cross-validation accuracy:  0.9109653233364574
Iteration  8
Cross-validation accuracy:  0.9109653233364574
Iteration  9
Cross-validation accuracy:  0.9100281162136832
Iteration  10
Cross-validation accuracy:  0.901593252108716
Mean cross-validation accuracy:  0.9144787970051494
```

In [6]:
```
print("Implementing NBC.....")
nbc_clf = MultinomialNB()

# Running cross-validation
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=1) # 10-fold cross-v
scores=[]
iteration = 0
for train_index, test_index in kf.split(data_tfidf,Y):
    iteration += 1
    print("Iteration ", iteration)
    X_train, Y_train = data_tfidf[train_index], Y.iloc[train_index]
    X_test, Y_test = data_tfidf[test_index], Y.iloc[test_index]
    nbc_clf.fit(X_train, Y_train) # Fitting NBC
    Y_pred = nbc_clf.predict(X_test)
    score = metrics.accuracy_score(Y_test, Y_pred) # Calculating accuracy
    print("Cross-validation accuracy: ", score)
    scores.append(score) # appending cross-validation accuracy for each iteration
nbc_mean_accuracy = np.mean(scores)
print("Mean cross-validation accuracy: ", nbc_mean_accuracy)
```

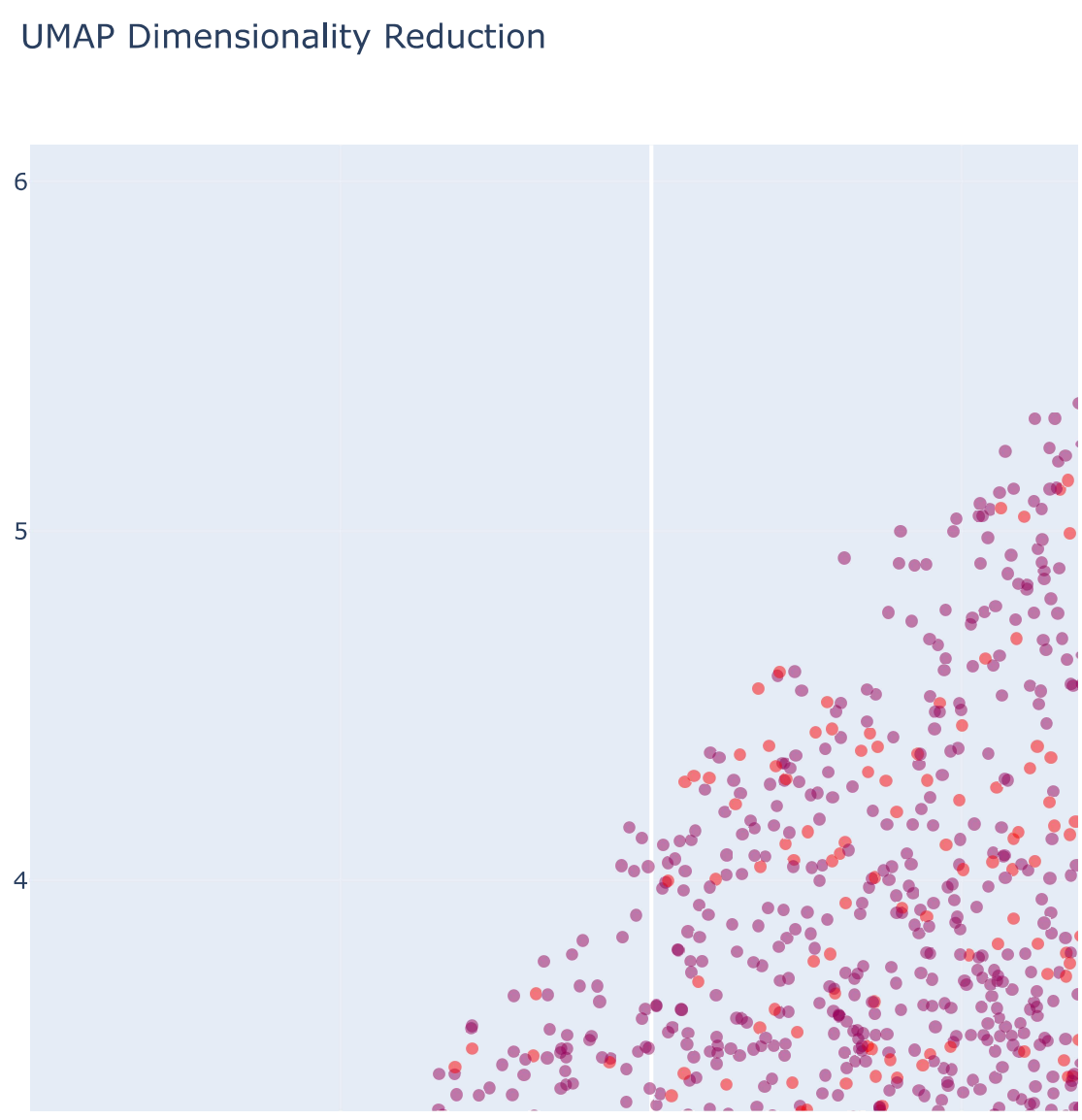
```
Implementing NBC.....
Iteration  1
Cross-validation accuracy:  0.9110486891385767
Iteration  2
Cross-validation accuracy:  0.8979400749063671
Iteration  3
Cross-validation accuracy:  0.9026217228464419
Iteration  4
Cross-validation accuracy:  0.8979400749063671
Iteration  5
Cross-validation accuracy:  0.8923220973782772
Iteration  6
Cross-validation accuracy:  0.8848314606741573
Iteration  7
Cross-validation accuracy:  0.8931583880037488
Iteration  8
Cross-validation accuracy:  0.8940955951265229
Iteration  9
Cross-validation accuracy:  0.8837863167760075
Iteration  10
Cross-validation accuracy:  0.8837863167760075
Mean cross-validation accuracy:  0.8941530736532473
```

In [9]:
```
import umap.umap_ as umap
u = umap.UMAP(n_components=2, n_neighbors=170, min_dist=0.4)
x_umap = u.fit_transform(data_tfidf)

recommend = list(df['Recommend'])
review = list(df['Review'])
vehicle_title = list(df['Vehicle_Title'])

data_ = [go.Scatter(x=x_umap[:,0], y=x_umap[:,1], mode='markers',
                    marker = dict(color=df['Recommend'], colorscale='Rainbow', op
                                  text=[f'recommend: {a}<br>review: {b}<br>vehicle_
                                  hoverinfo='text')]

layout = go.Layout(title = 'UMAP Dimensionality Reduction', width = 1500, height
                    xaxis = dict(title='First Dimension'),
                    yaxis = dict(title='Second Dimension'))
fig = go.Figure(data=data_, layout=layout)
fig.show()
```



In [ ]: