```
#Imports
import pandas as pd
import numpy as np
import itertools
import time
import seaborn as sn
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
auto =pd.read_csv('Auto.csv')
print(auto.shape)
auto.head()
```

```
(397, 9)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130	3504	12.0	70	1
1	15.0	8	350.0	165	3693	11.5	70	1
2	18.0	8	318.0	150	3436	11.0	70	1

```
auto.describe()
```

```
auto.isnull().any()
```

```
mpg             False
cylinders       False
displacement    False
horsepower      False
weight          False
acceleration    False
year            False
origin          False
name            False
dtype: bool
```

```
auto[auto['horsepower'] == '?']
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
32	25.0	4	98.0	?	2046	19.0	71	1
126	21.0	6	200.0	?	2875	17.0	74	1
330	40.9	4	85.0	?	1835	17.3	80	2

```
auto = auto[auto['horsepower'] != '?']
auto['horsepower'] = auto['horsepower'].astype(int)
```

```
print(auto.shape)
auto.head()
```

```
(392, 9)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130	3504	12.0	70	1
1	15.0	8	350.0	165	3693	11.5	70	1
2	18.0	8	318.0	150	3436	11.0	70	1

(a) What is the fundamental idea behind Support Vector Machines?

SVM's main idea if for a compromise between separating the classes and having the widest possible street (few instances on the street). The algorithm separates the data into two destinct classes

(b) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

```
isAboveMedianGPM = (auto["mpg"] > auto["mpg"].median()).map({False: 0, True: 1})
auto["isAboveMedianGPM"] = isAboveMedianGPM
auto.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000
mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.541327
std	7.805007	1.705783	104.644004	38.491160	849.402560	2.758864
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000
25%	17.000000	4.000000	105.000000	75.000000	2225.250000	13.775000
50%	22.750000	4.000000	151.000000	93.500000	2803.500000	15.500000
75%	29.000000	8.000000	275.750000	126.000000	3614.750000	17.025000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000

(c) Fit a linear support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Comment on your results.

```
X = auto.iloc[:,:-2]
y = auto["isAboveMedianGPM"]
X.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130	3504	12.0	70	1
1	15.0	8	350.0	165	3693	11.5	70	1
2	18.0	8	318.0	150	3436	11.0	70	1
3	16.0	8	304.0	150	3433	12.0	70	1
4	17.0	8	302.0	140	3449	10.5	70	1

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,random_state = 0)
```

```
from sklearn import svm
cost = [0.0001,1000]
results={}
for  c in cost:
  svm_clf = Pipeline([
    ("scaler", StandardScaler()),
```

```
    ("linear_svc", LinearSVC(C=c, loss="hinge", random_state=42, max_iter =10000))
  ])
  svm_clf.fit(X_train, y_train)
  y_pred = svm_clf.predict(X_test)
  results[c] = y_pred
  print(y_pred)
  print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
[1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1
 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 0 0
 0 0 1 0 1]
Accuracy:  0.8987341772151899
[1 0 0 1 1 0 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 1 0 1 1
 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0
 1 0 1 0 1]
Accuracy:  1.0
D:\Code\Py\lib\site-packages\sklearn\svm\_base.py:1199: ConvergenceWarning: Liblinear failed to c
  warnings.warn(
```

```
results = pd.DataFrame(results)
results
```

	0.0001	1000.0000
0	1	1
1	1	0
2	0	0
3	1	1
4	1	1
...	...	...
74	0	1
75	0	0
76	1	1
77	0	0
78	1	1

79 rows × 2 columns

```
print("Low mileage predictions for C = 0.0001: ",len(results[results[0.0001]==0]))
print("High mileage predictions for C = 0.0001: ",len(results[results[0.0001]==1]))
print("Low mileage predictions for C = 1000.0: ",len(results[results[1000.0]==0]))
print("High mileage predictions for C = 1000.0: ",len(results[results[1000.0]==1]))
```

```
Low mileage predictions for C = 0.0001:  36
High mileage predictions for C = 0.0001:  43
Low mileage predictions for C = 1000.0:  42
High mileage predictions for C = 1000.0:  37
```

C is the Regularization parameter. The strength of the regularization is inversely proportional to C and therefore a larger the value of C such as 1000, makes the margin smaller. for 0.0001, the decision boundary has a large margin and many misclassifications. In turn 1000 has fewer misclassifications.

(d) Now repeat (c), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
#radial basis kernels
gamma1, gamma2 = 0.1, 5
C1, C2 = 0.001, 1000
hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)
svm_clfs = []
for gamma, C in hyperparams:
  rbf_kernel_svm_clf = Pipeline([
  ("scaler", StandardScaler()),
  ("svm_clf", SVC(kernel="rbf", gamma=gamma, C=C))
  ])
  rbf_kernel_svm_clf.fit(X_train, y_train)
  y_pred = rbf_kernel_svm_clf.predict(X_test)
  print(f"Radial SVM for gamma = {gamma}, C = {C}", y_pred)
```

```
Radial SVM for gamma = 0.1, C = 0.001 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1]
Radial SVM for gamma = 0.1, C = 1000 [1 1 0 1 1 0 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0
 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0
 0 0 1 0 1]
Radial SVM for gamma = 5, C = 0.001 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1]
Radial SVM for gamma = 5, C = 1000 [1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 0 1
 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0
 0 0 1 0 1]
```

```
degree = [3, 10]
#polynomial basis kernels
for i in range(0,2):
  poly_kernel_svm_clf = Pipeline([
  ("scaler", StandardScaler()),
  ("svm_clf", SVC(kernel="poly", degree=degree[i], coef0=1, C=cost[i]))
  ])
  poly_kernel_svm_clf.fit(X_train, y_train.values.ravel())
  y_pred = poly_kernel_svm_clf.predict(X_test)
  print(f'Polynomial SVM for = {degree[i]}, C = {cost[i]}', y_pred)
  print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Polynomial SVM for = 3, C = 0.0001 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1]
```

```
Accuracy:  0.46835443037974683
Polynomial SVM for = 10, C = 1000 [1 1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1
 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0
 0 0 0 0 1]
Accuracy:  0.9240506329113924
```

The smaller the gamma and the larger the cost the better the acuracy of the prediction, because fewer of the datapoints are left inbetween the classes.