In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

Q.no.1. a

The fundamental idea behind Support Vector Machines is to fit the widest possible "street" between the classes (a concept often referred to as large margin classification).
SVM aims to have the largest possible margin between the decision boundary that separates the two classes and the training instances. When performing hard margin classification which works well only for the linearly separable data and when there are no outliers, the SVM strictly imposes that all instances must be off the street and on the right side while when performing soft margin classification, it searches for a compromise between perfectly separating the two classes and having the widest possible street which means that a few instances might end up on the street or even on the wrong side.
Another prominent concept in the SVM is to use kernels when training on nonlinear datasets.

Q.no.1. b

In [2]:
```
# Creating the outcome variable
def categorise(row):
    if row['mpg'] > df['mpg'].median():
        return 1
    else:
        return 0

df = pd.read_csv('Auto.csv')
df['mileage_status'] = df.apply(lambda row: categorise(row), axis=1)
```

In [7]:
```
df.head()
```

Out[7]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mileage
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu	
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320	
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite	
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst	
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino	

Q.no.1. c

In [8]:
```
# Replacing the invalid value in horsepower
def remove_invalid(row):
    if row['horsepower'] == '?':
        return -1
    else:
        return row['horsepower']

# Replacing invalid value in horsepower with median as it is skewed
def fill_invalid(row):
    if row['horsepower'] == -1:
        return df['horsepower'].median()
    else:
        return row['horsepower']

df['horsepower'] = df.apply(lambda row: remove_invalid(row), axis=1)
df['horsepower'] = df['horsepower'].astype('int64')
df['horsepower'] = df.apply(lambda row: fill_invalid(row), axis=1)
df = df.drop(['name'], axis=1) # Dropping the name as it doesnt have significant

X, y = df.drop(['mileage_status'], axis = 1), df['mileage_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, randon
```

In [9]:
```
X_train.dtypes
```

Out[9]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
year              int64
origin            int64
dtype: object
```

In [13]:
```
accuracy = {}
values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
for cost in values:
    svm_clf = Pipeline([
        ("transformer", ColumnTransformer(transformers = [('standarscaler',
                                                            StandardScaler(),
                                                            ['mpg', 'cylinders', '
                                                             'weight', 'accelerati
                                          remainder='passthrough')),
        ("linear_svc", LinearSVC(C=cost, loss="hinge", random_state=42)),
    ])
    svm_clf.fit(X_train, y_train)
    y_pred = svm_clf.predict(X_test)
    accuracy[cost] = accuracy_score(y_test, y_pred)

accuracy # accuracy is used because the outcome variable is balanced
```

```
C:\Users\joshi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\joshi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\joshi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\joshi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[13]:
```
{0.001: 0.875,
 0.01: 0.9083333333333333,
 0.1: 0.9833333333333333,
 1: 0.975,
 10: 0.9916666666666667,
 100: 0.9916666666666667,
 1000: 0.9916666666666667}
```

As we increase the value of cost from 0.001 to 0.1, the model which was initially underfitting, starts capturing the essential patterns in the data and hence its accuracy increases. However, as we further increase the cost above 0.1, the model has only a slight increase in accuracy which means it begins overfitting the data because of which the accuracy peaks at 0.99 and does not increase any more.

Q.no.1. d

In [14]:
```
gammas = [0.1, 1, 10]
degrees = [3, 6, 9]
costs = [0.001, 1, 1000]

print("SVM with polynomial basis kernel:")
for degree in degrees:
    for cost in costs:
        svm_poly = Pipeline([
            ("transformer", ColumnTransformer(transformers = [('standarscaler', S
                                                               ['mpg', 'cylinders
                                                                'weight', 'accele
                                              remainder='passthrough')),
            ("svm_clf", SVC(kernel="poly", degree=degree, C=cost)), # coeff has b
        ])
        svm_poly.fit(X_train, y_train)
        y_pred = svm_poly.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        print("Degree: " + str(degree) + " Cost: " + str(cost) + " Accuracy: " +
    print("=" * 20)

print("\n\n\nSVM with radial kernel:")
for gamma in gammas:
    for cost in costs:
        svm_poly = Pipeline([
            ("transformer", ColumnTransformer(transformers = [('standarscaler', S
                                                               ['mpg', 'cylinders
                                                                'weight', 'accele
                                              remainder='passthrough')),
            ("svm_clf", SVC(kernel="rbf", gamma=gamma, C=cost)),
        ])
        svm_poly.fit(X_train, y_train)
        y_pred = svm_poly.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        print("Gamma: " + str(gamma) + " Cost: " + str(cost) + " Accuracy: " + st
    print("=" * 20)
```

```
SVM with polynomial basis kernel:
Degree: 3 Cost: 0.001 Accuracy: 0.4666666666666667
Degree: 3 Cost: 1 Accuracy: 0.925
Degree: 3 Cost: 1000 Accuracy: 0.975
====================
Degree: 6 Cost: 0.001 Accuracy: 0.575
Degree: 6 Cost: 1 Accuracy: 0.8666666666666667
Degree: 6 Cost: 1000 Accuracy: 0.95
====================
Degree: 9 Cost: 0.001 Accuracy: 0.625
Degree: 9 Cost: 1 Accuracy: 0.7833333333333333
Degree: 9 Cost: 1000 Accuracy: 0.9166666666666666
====================



SVM with radial kernel:
Gamma: 0.1 Cost: 0.001 Accuracy: 0.4666666666666667
Gamma: 0.1 Cost: 1 Accuracy: 0.9666666666666667
Gamma: 0.1 Cost: 1000 Accuracy: 0.9833333333333333
```

```
====================
Gamma: 1 Cost: 0.001 Accuracy: 0.4666666666666667
Gamma: 1 Cost: 1 Accuracy: 0.925
Gamma: 1 Cost: 1000 Accuracy: 0.9333333333333333
====================
Gamma: 10 Cost: 0.001 Accuracy: 0.4666666666666667
Gamma: 10 Cost: 1 Accuracy: 0.775
Gamma: 10 Cost: 1000 Accuracy: 0.7666666666666667
====================
```

For the polynomial basis kernel, most suitable value for degree was found to be 3 as it obtained the highest accuracy of 0.975. For all the degrees we have chosen, as we increase the cost, our models become more efficient which means that our model is still not overfitting up until the cost value of 1000. As for the degree, our model appears to be overfitting in degree values above 3 as they have a decrease in accuracy for majority of costs compared to the model with degree 3 for the same costs with an exception for the cost of 0.001.

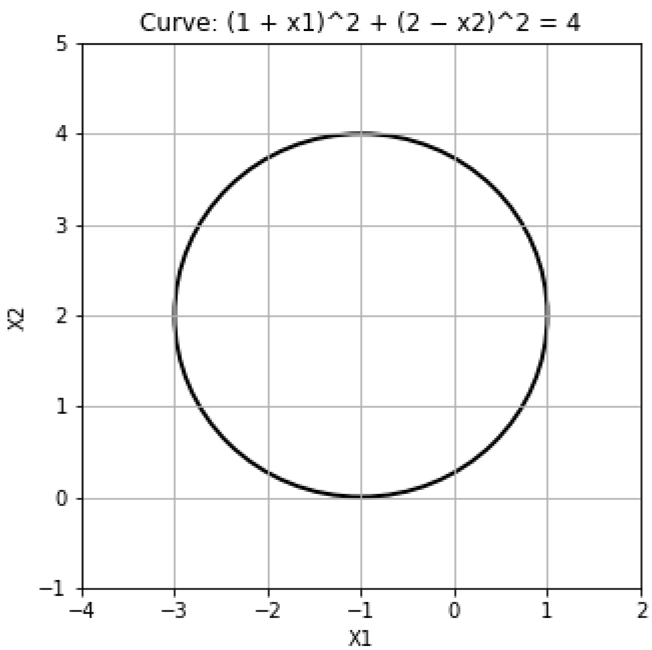
Similarly, for the radial kernel, gamma value of 0.1 appears to be the most suitable as it has the highest accuracy among all the gamma values. For gamma values of 1 and 10, the accuracies are less than that for gamma value of 0.1 except for the cost 0.001, which suggest that our model is overfitting as the decision boundary ends up being more irregular, wiggling around individual instances.

As with all the values of degree or gamma, keeping them constant, our both models (with polynomial and radial kernels) appear to be underfitting for cost of 0.001 and efficiently capturing the pattern in the data as we increase the cost except for gamma value of 10 where it is overfitting at cost 1000.

Finally, our linear kernel model which obtained the maximum accuracy of 0.99 appears to be the most suitable model for our dataset.

Q.no.2. a

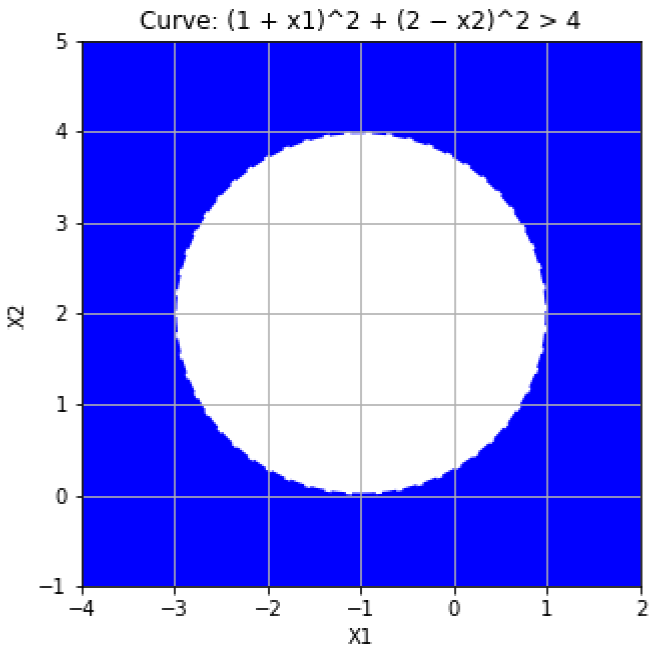
In [58]:
```
circle = plt.Circle((-1, 2), radius=2, facecolor='white', edgecolor='black', line

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot()
ax.add_artist(circle)

ax.set_xlim(-4, 2)
ax.set_ylim(-1, 5)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_title("Curve: (1 + x1)^2 + (2 - x2)^2 = 4")
plt.grid()
plt.show()
```



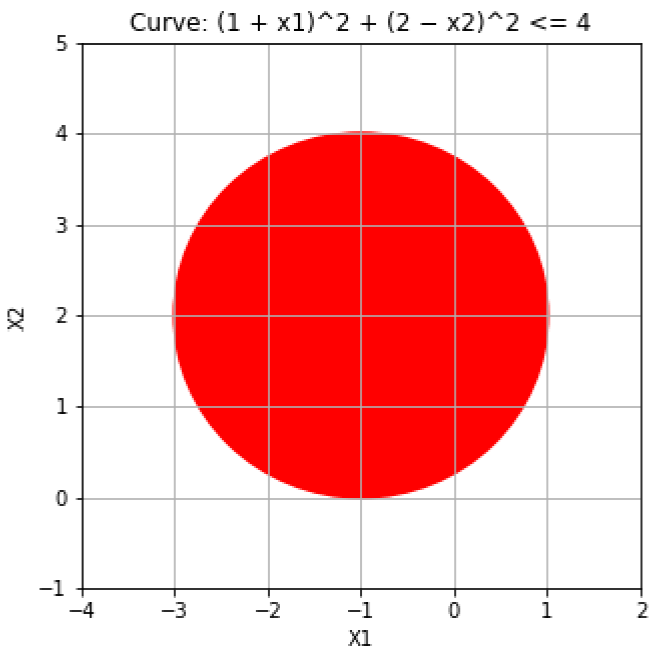
The curve is a circle with center (-1, 2) and radius 2 as it follows the equation (x-h)^2 + (y-k)^2 = r^2 where (h,k) is the center and r is the radius of the circle.

Q.no.2. b

In [87]:
```
circle = plt.Circle((-1, 2), radius=2, facecolor='white', edgecolor='blue', lines

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot()
ax.add_artist(circle)

ax.set_xlim(-4, 2)
ax.set_ylim(-1, 5)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_facecolor('blue')
ax.set_title("Curve: (1 + x1)^2 + (2 - x2)^2 > 4")
plt.grid()
plt.show()
```



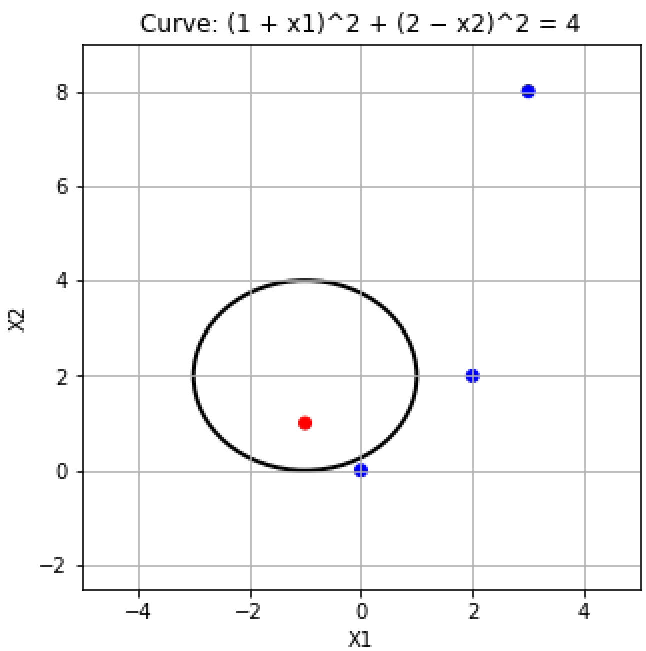
In [85]:
```
circle = plt.Circle((-1, 2), radius=2, facecolor='red', edgecolor='red', linestyl

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot()
ax.add_artist(circle)

ax.set_xlim(-4, 2)
ax.set_ylim(-1, 5)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_title("Curve: (1 + x1)^2 + (2 - x2)^2 <= 4")
plt.grid()
plt.show()
```



The set of points for the equation (1 + x1)^2 + (2 − x2)^2 > 4 lie on the region beyond the circle while the set of points for the equation (1 + x1)^2 + (2 − x2)^2 <= 4 lie inside the circle including the border of the circle which is the reason why the there are solid lines in the second circle compared to the dashed lines in the first circle to mark the border.

Q.no.2. c

In [103]:
```
circle = plt.Circle((-1, 2), radius=2, facecolor='none', edgecolor='black', linew

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot()
ax.add_artist(circle)

ax.set_xlim(-5, 5)
ax.set_ylim(-2.5, 9)
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_title("Curve: (1 + x1)^2 + (2 - x2)^2 = 4")
points_x1 = [0, -1, 2, 3]
points_x2 = [0, 1, 2, 8]
colors = ['b', 'r', 'b', 'b']
plt.scatter(points_x1, points_x2, c=colors)
plt.grid()
plt.show()
```



(0, 0), (2, 2), and (3, 8) lie on the blue class and (-1, 1) lie on the red class as shown in the figure above.