In [1]:
```
import pandas as pd
import time
```

Question 1

In [2]:
```
df=pd.read_csv("./spambase.data")
```

In [3]:
```
df.info()
# all numeric values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 58 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       4600 non-null   float64
 1   0.64    4600 non-null   float64
 2   0.64.1  4600 non-null   float64
 3   0.1     4600 non-null   float64
 4   0.32    4600 non-null   float64
 5   0.2     4600 non-null   float64
 6   0.3     4600 non-null   float64
 7   0.4     4600 non-null   float64
 8   0.5     4600 non-null   float64
 9   0.6     4600 non-null   float64
 10  0.7     4600 non-null   float64
 11  0.64.2  4600 non-null   float64
 12  0.8     4600 non-null   float64
 13  0.9     4600 non-null   float64
 14  0.10    4600 non-null   float64
 15  0.32.1  4600 non-null   float64
 16  0.11    4600 non-null   float64
 17  1.29    4600 non-null   float64
 18  1.93    4600 non-null   float64
 19  0.12    4600 non-null   float64
 20  0.96    4600 non-null   float64
 21  0.13    4600 non-null   float64
 22  0.14    4600 non-null   float64
 23  0.15    4600 non-null   float64
 24  0.16    4600 non-null   float64
 25  0.17    4600 non-null   float64
 26  0.18    4600 non-null   float64
 27  0.19    4600 non-null   float64
 28  0.20    4600 non-null   float64
 29  0.21    4600 non-null   float64
 30  0.22    4600 non-null   float64
 31  0.23    4600 non-null   float64
 32  0.24    4600 non-null   float64
 33  0.25    4600 non-null   float64
 34  0.26    4600 non-null   float64
 35  0.27    4600 non-null   float64
 36  0.28    4600 non-null   float64
 37  0.29    4600 non-null   float64
 38  0.30    4600 non-null   float64
 39  0.31    4600 non-null   float64
 40  0.33    4600 non-null   float64
 41  0.34    4600 non-null   float64
 42  0.35    4600 non-null   float64
 43  0.36    4600 non-null   float64
 44  0.37    4600 non-null   float64
 45  0.38    4600 non-null   float64
 46  0.39    4600 non-null   float64
 47  0.40    4600 non-null   float64
 48  0.41    4600 non-null   float64
 49  0.42    4600 non-null   float64
 50  0.43    4600 non-null   float64
 51  0.778   4600 non-null   float64
```

```
 52  0.44    4600 non-null   float64
 53  0.45    4600 non-null   float64
 54  3.756   4600 non-null   float64
 55  61      4600 non-null   int64
 56  278     4600 non-null   int64
 57  1       4600 non-null   int64
dtypes: float64(55), int64(3)
memory usage: 2.0 MB
```

In [4]:

```
from sklearn.cluster import KMeans
n_centers=[5,10,15,20,25]
time_taken=[]
for i in n_centers:
    start=time.time()
    kmeans_model=KMeans(n_clusters=i)
    kmeans_model.fit(df)
    time_taken.append(time.time()-start)
print(n_centers)
print(time_taken)
```

```
[5, 10, 15, 20, 25]
[0.15382719039916992, 0.1705472469329834, 0.17885327339172363, 0.3304574489593506, 0.4808390140533447]
```

In [5]:

```
n_centers=[5,10,15,20,25]
score=[]
for i in n_centers:
    kmeans_model=KMeans(n_clusters=i)
    kmeans_model.fit(df)
    score.append(kmeans_model.score(df))
print(n_centers)
print(score)
```

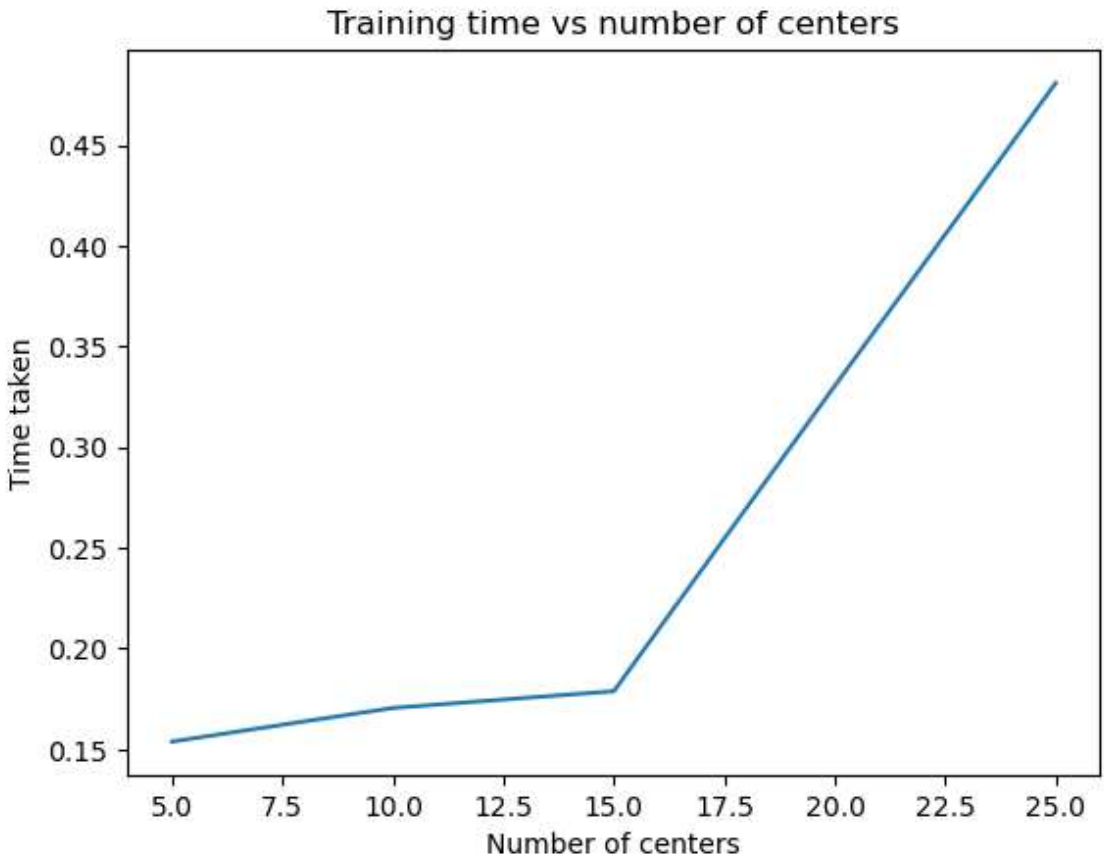
```
[5, 10, 15, 20, 25]
[-256344529.3905624, -76982091.64281926, -36655972.56144245, -21378341.038881607, -15295042.658365007]
```

In [6]:

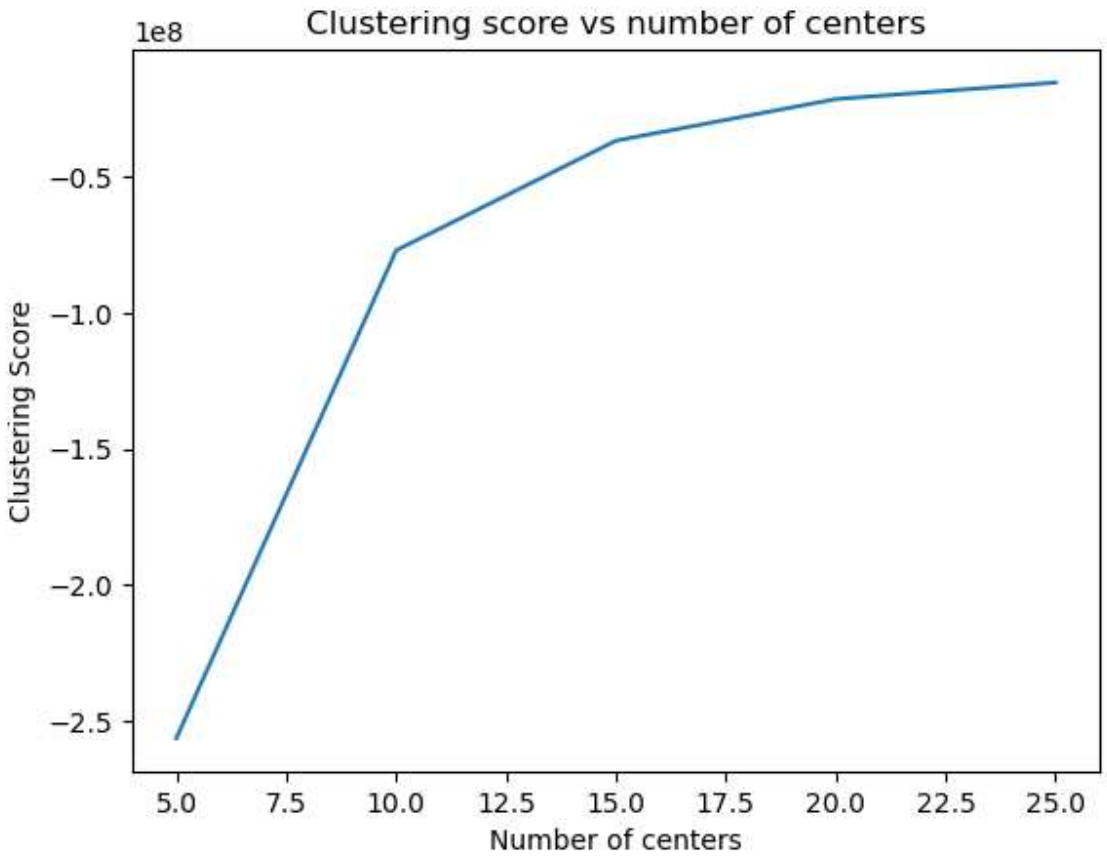
```
n_samples = [100,500,1000,2000,5000]
time_taken2=[]
for i in n_samples:
    start=time.time()
    data=df.sample(n=i,replace=True)
    kmeans_model = KMeans(n_clusters=15)
    kmeans_model.fit(data)
    time_taken2.append(time.time()-start)
print(n_samples)
print(time_taken2)
```

```
[100, 500, 1000, 2000, 5000]
[0.12843060493469238, 0.23240065574645996, 0.2565042972564697, 0.3556063175201416, 0.43860435485839844]
```

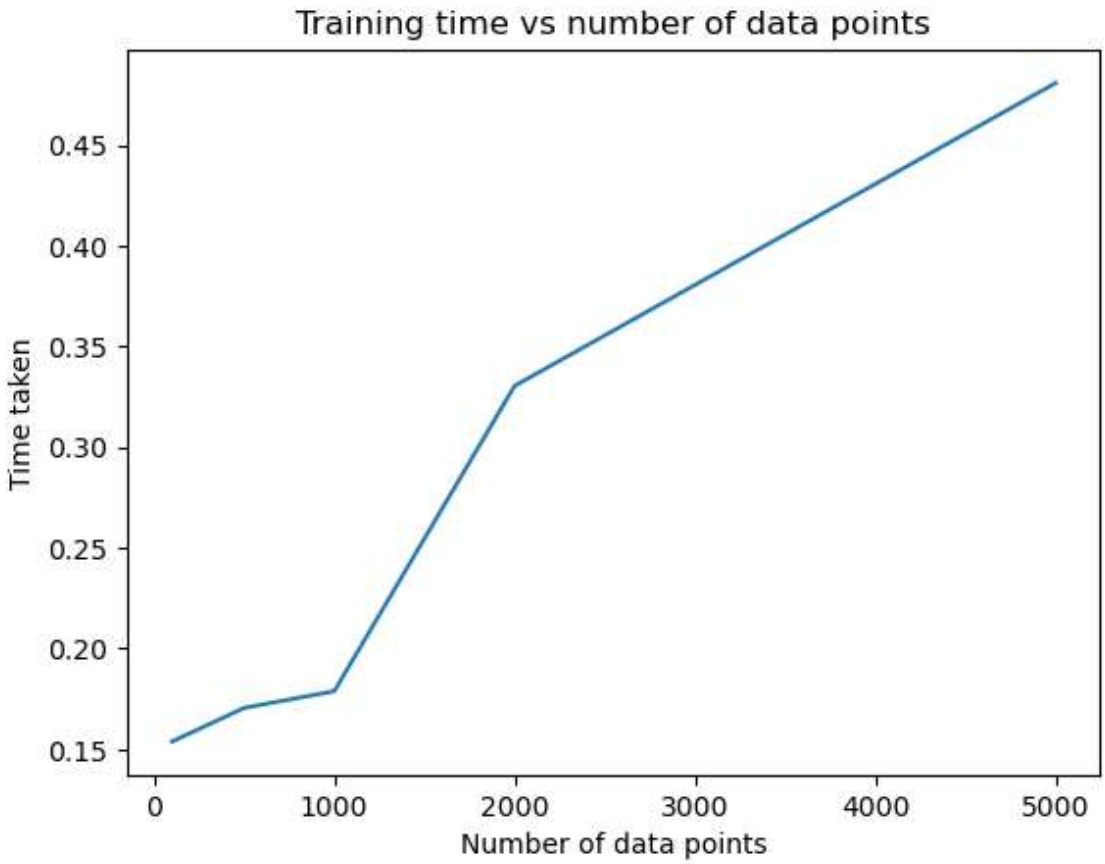
In [7]:
```
import matplotlib.pyplot as plt
plt.plot(n_centers,time_taken)
plt.xlabel('Number of centers')
plt.ylabel('Time taken')
plt.title('Training time vs number of centers')
plt.show()
```



In [8]:
```
import matplotlib.pyplot as plt
plt.plot(n_centers,score)
plt.xlabel('Number of centers')
plt.ylabel('Clustering Score')
plt.title('Clustering score vs number of centers')
plt.show()
```



In [9]:
```
import matplotlib.pyplot as plt
plt.plot(n_samples,time_taken)
plt.xlabel('Number of data points')
plt.ylabel('Time taken')
plt.title('Training time vs number of data points')
plt.show()
```



Question 2

In [10]:

```
from sklearn.datasets import load_iris
iris=load_iris()
X=iris.data
Y=iris.target
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
import time
n = 1000
d = 4
classifier = SVC(C = 100)
print("d,time,Accuracy,Std")
for X_new in [X,X[:,:2],X[:,-2:],X[:,1:],X[:,1:10]]:
    startTime = time.time()
    scores = cross_val_score(classifier, X_new, Y, cv = 10)
    duration = time.time() - startTime
    meanAcc = scores.mean()
    stdAcc = scores.std()
    print(X_new.shape[1], duration, meanAcc, stdAcc)
```

```
d,time,Accuracy,Std
4 0.024004697799682617 0.9733333333333334 0.04422166387140532
2 0.04800605773925781 0.8066666666666666 0.06960204339273703
2 0.031998634338378906 0.9533333333333334 0.059999999999999984
3 0.02400350570678711 0.9533333333333334 0.052068331172711015
3 0.03200030326843262 0.9533333333333334 0.052068331172711015
```

In [11]:

```
from sklearn.datasets import load_iris
iris=load_iris()
X=iris.data
Y=iris.target
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
n = 500
d = 4
rndPer = np.random.permutation(d)
print(f"Useful columns are hidden at {rndPer[:2]}")
X[:, rndPer] = X
k=3
features = []
classifier = SVC(C = 100)
print("d,time,Acc,Std")
while len(features)<k:
    bestScore = -1
    bestFeature = None
    for j in range(d):
        if j in features:
            continue
        X_new = np.concatenate([X[:,features],X[:,j:j+1]],axis=1)
        scores = cross_val_score(classifier, X_new, Y, cv = 10)
        meanAcc = scores.mean()
        stdAcc = scores.std()
        if meanAcc>bestScore:
            bestScore = meanAcc
            bestFeature = j
        features.append(bestFeature)
    print( bestScore, bestFeature)
print("The final set of detected features is", features)
```

```
Useful columns are hidden at [2 3]
d,time,Acc,Std
0.8066666666666666 1
The final set of detected features is [0, 1, 1, 1]
```

In [12]:
```
 sklearn.feature_selection import VarianceThreshold, SelectKBest, SelectPer
100
4
is.data
is.target
e_funcs = [ chi2, f_classif, mutual_info_classif ]
sformers = [VarianceThreshold( threshold = (0.01) )]
sformers +=[SelectKBest( score_func, k = k )
            for score_func in score_funcs
              for k in range(1,d)]
sformers +=[SelectPercentile( score_func, percentile = p )
            for score_func in score_funcs
              for p in range(1,100,20)]

transformer in transformers:
transformer.fit(X, Y)
result = transformer.get_support()
X_new = transformer.transform(X)
transName = str(transformer)
transName = transName[:transName.find("(")]
print(transName,X_new.shape, result)
```

```
VarianceThreshold (150, 4) [ True  True  True  True]
SelectKBest (150, 1) [False False  True False]
SelectKBest (150, 2) [False  True  True False]
SelectKBest (150, 3) [False  True  True  True]
SelectKBest (150, 1) [False False  True False]
SelectKBest (150, 2) [False  True  True False]
SelectKBest (150, 3) [False  True  True  True]
SelectKBest (150, 1) [False  True False False]
SelectKBest (150, 2) [False  True  True False]
SelectKBest (150, 3) [ True  True  True False]
SelectPercentile (150, 0) [False False False False]
SelectPercentile (150, 0) [False False False False]
SelectPercentile (150, 2) [False  True  True False]
SelectPercentile (150, 2) [False  True  True False]
SelectPercentile (150, 3) [ True  True  True False]
SelectPercentile (150, 0) [False False False False]
SelectPercentile (150, 0) [False False False False]
SelectPercentile (150, 2) [False  True  True False]
SelectPercentile (150, 2) [False  True  True False]
SelectPercentile (150, 3) [ True  True  True False]
SelectPercentile (150, 1) [False  True False False]
SelectPercentile (150, 1) [False False  True False]
SelectPercentile (150, 2) [False  True  True False]
SelectPercentile (150, 2) [False  True  True False]
SelectPercentile (150, 3) [False  True  True  True]
```

```
C:\Users\Aaryan Agarwal\anaconda3\lib\site-packages\sklearn\feature_selec
tion\_base.py:96: UserWarning: No features were selected: either the data
is too noisy or the selection test too strict.
  warn(
C:\Users\Aaryan Agarwal\anaconda3\lib\site-packages\sklearn\feature_selec
tion\_base.py:96: UserWarning: No features were selected: either the data
is too noisy or the selection test too strict.
  warn(
C:\Users\Aaryan Agarwal\anaconda3\lib\site-packages\sklearn\feature_selec
tion\_base.py:96: UserWarning: No features were selected: either the data
is too noisy or the selection test too strict.
  warn(
C:\Users\Aaryan Agarwal\anaconda3\lib\site-packages\sklearn\feature_selec
tion\_base.py:96: UserWarning: No features were selected: either the data
is too noisy or the selection test too strict.
  warn(
```

In [13]:

```
import numpy as np
from sklearn.feature_selection import VarianceThreshold, chi2, f_classif,S
n = 100
d = 4
X=iris.data
Y=iris.target
score_funcs = [ chi2, f_classif,]
transformers = [ VarianceThreshold( threshold = (0.01) )]
transformers += [SelectFpr(score_func, alpha = 5e-2 ) for score_func in sc
transformers += [SelectFwe(score_func, alpha = 5e-2 ) for score_func in sc

for transformer in transformers:
    transformer.fit(X, Y)
    result = transformer.get_support()
    X_new = transformer.transform(X)
    transName = str(transformer)
    transName = transName[:transName.find("(")]
    print(transName,X_new.shape, result)
```

```
VarianceThreshold (150, 4) [ True  True  True  True]
SelectFpr (150, 2) [False  True  True False]
SelectFpr (150, 4) [ True  True  True  True]
SelectFwe (150, 2) [False  True  True False]
SelectFwe (150, 4) [ True  True  True  True]
```

Question 3

In [14]:
```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
n = 20
d = 2
X = np.random.rand(n,d)
y = np.zeros((n))
y[ X[:,0] < X[:,1] ] = 1
estimator = SVC(kernel='linear')
score = cross_val_score(estimator, X, y).mean()
print(f"Score = {score:.3f}.")
```

```
Score = 0.900.
```

In [15]:
```
import time
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
n = 20
d = 2
X = np.random.rand(n,d)
y = np.zeros((n))
y[ X[:,0] < X[:,1] ] = 1
for degree in range(1,20):
    startTime = time.time()
    regressor = make_pipeline(PolynomialFeatures(degree), Ridge())
    score = cross_val_score(regressor, X, y).mean()
    duration = time.time() - startTime
    print(f"Degree={degree}, Score = {score:.3f}, duration = {duration:.10
```

```
Degree=1, Score = 0.525, duration = 0.0252645016 seconds
Degree=2, Score = 0.551, duration = 0.0231320858 seconds
Degree=3, Score = 0.540, duration = 0.0160007477 seconds
Degree=4, Score = 0.529, duration = 0.0164315701 seconds
Degree=5, Score = 0.520, duration = 0.0159974098 seconds
Degree=6, Score = 0.515, duration = 0.0160007477 seconds
Degree=7, Score = 0.511, duration = 0.0239932537 seconds
Degree=8, Score = 0.509, duration = 0.0244460106 seconds
Degree=9, Score = 0.508, duration = 0.0240197182 seconds
Degree=10, Score = 0.507, duration = 0.0241019726 seconds
Degree=11, Score = 0.507, duration = 0.0239953995 seconds
Degree=12, Score = 0.507, duration = 0.0240087509 seconds
Degree=13, Score = 0.507, duration = 0.0239818096 seconds
Degree=14, Score = 0.508, duration = 0.0240144730 seconds
Degree=15, Score = 0.508, duration = 0.0240097046 seconds
Degree=16, Score = 0.508, duration = 0.0319907665 seconds
Degree=17, Score = 0.509, duration = 0.0322029591 seconds
Degree=18, Score = 0.509, duration = 0.0320265293 seconds
Degree=19, Score = 0.509, duration = 0.0437183380 seconds
```

In [ ]:
```
```