CS-504 Project

Group-26

Akhil Samineni(G01319367),Naga Tejaswi Veluri (G01335094),Vaishnavi Putcha (G01326950)

Step-1

Importing data

In [1]:
```
# We use pandas to convert a csv file into a DataFrame
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import os
```

In [2]:
```
os.chdir("D:\\study\\Gmu\\Sem 1\cs\\final")
data="bank.csv"
data = pd.read_csv(data, sep=";",header = 0)
data.head()
```

Out[2]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	dura
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	

Data Preprocessing

In [3]:
```
from sklearn import tree
```

In [4]:
```
# scikit-learn needs everything to be numerical for decision trees to work.
# So, we'll map Y,N to 1,0 and levels of education to some scale of 0-2.
# we know we'll get NaN for unexpected values.
d = {'yes': 1, 'no': 0}
data['default'] = data['default'].map(d)
data['housing'] = data['housing'].map(d)
data['loan'] = data['loan'].map(d)
data['y'] = data['y'].map(d)

d = {'unknown' : 0, 'services' : 1, 'management' : 2, 'blue-collar' : 3, 'self-employed
     'entrepreneur': 6, 'admin.': 7, 'student': 8, 'housemaid' : 9, 'retired' : 10, 'un
data['job'] = data['job'].map(d)

d = {'married' : 0, 'single': 1, 'divorced' : 2}
```

```
data['marital'] = data['marital'].map(d)

d = {'unknown' : 0, 'primary': 1, 'secondary' : 2, 'tertiary': 3}
data['education'] = data['education'].map(d)

d = {'unknown' : 0, 'cellular': 1, 'telephone' : 2}
data['contact'] = data['contact'].map(d)

d = {'oct': 10, 'may': 5, 'apr': 4, 'jun': 6, 'feb': 2, 'aug': 8, 'jan': 1, 'jul': 7, '
data['month'] = data['month'].map(d)

d = {'unknown' : 0, 'failure' :1, 'other' : 2, 'success' : 3}
data['poutcome'] = data['poutcome'].map(d)
data.head()
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	cam
0	30	11	0	1	0	1787	0	0	1	19	10	79	
1	33	1	0	2	0	4789	1	1	1	11	5	220	
2	35	2	1	3	0	1350	1	0	1	16	4	185	
3	30	2	0	3	0	1476	1	1	0	3	6	199	
4	59	3	0	2	0	0	1	0	0	5	5	226	

Checking and Removing NULL values

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

In [6]:
```
data.dtypes
```

Out[6]:
```
age          int64
job          int64
marital      int64
education    int64
default      int64
balance      int64
housing      int64
loan         int64
```

```
contact      int64
day          int64
month        int64
duration     int64
campaign     int64
pdays        int64
previous     int64
poutcome     int64
y            int64
dtype: object
```

In [7]:
```
data.describe()
```

Out[7]:

	age	job	marital	education	default	balance	housing	
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	452
mean	41.170095	4.314753	0.498120	2.065915	0.016810	1422.657819	0.566025	
std	10.576211	2.709093	0.695471	0.780906	0.128575	3009.638142	0.495676	
min	19.000000	0.000000	0.000000	0.000000	0.000000	-3313.000000	0.000000	
25%	33.000000	2.000000	0.000000	2.000000	0.000000	69.000000	0.000000	
50%	39.000000	3.000000	0.000000	2.000000	0.000000	444.000000	1.000000	
75%	49.000000	6.000000	1.000000	3.000000	0.000000	1480.000000	1.000000	
max	87.000000	11.000000	2.000000	3.000000	1.000000	71188.000000	1.000000	

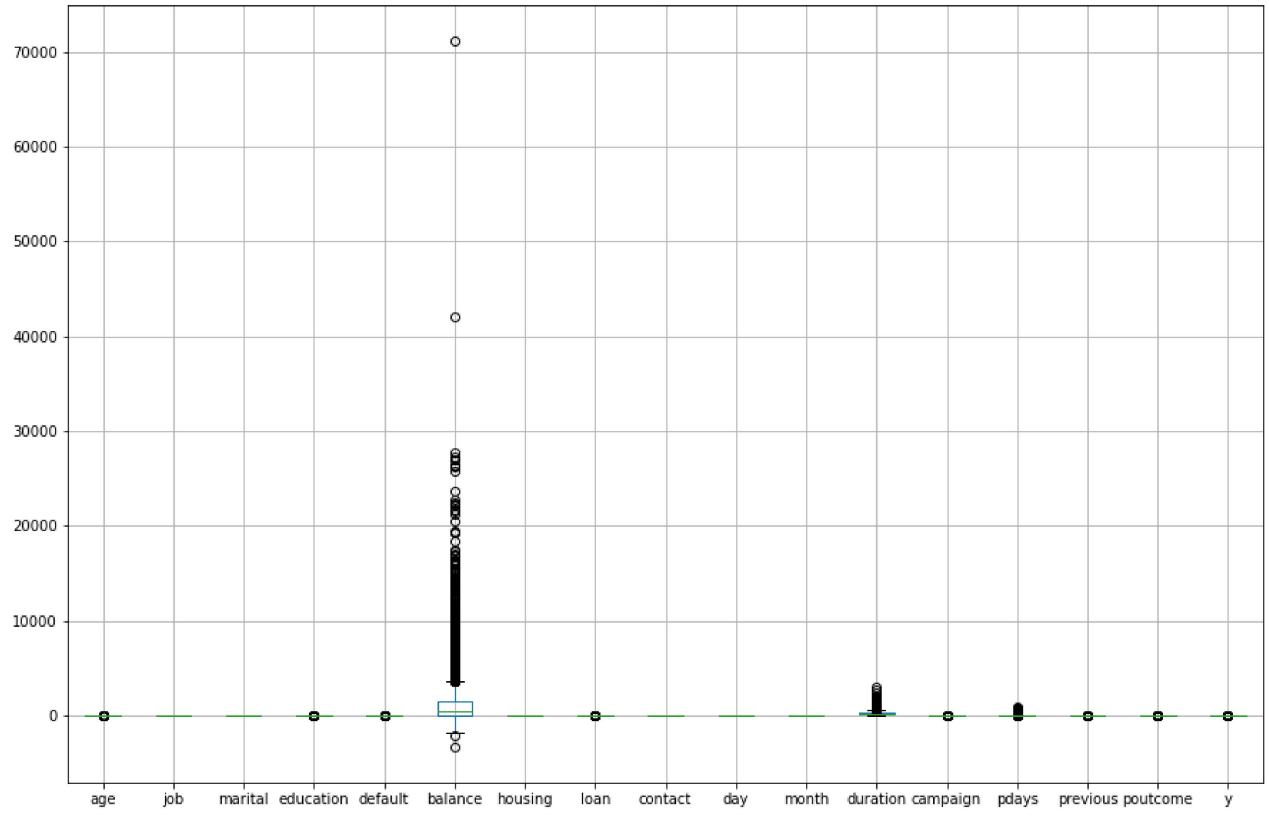
In [8]:
```
features = list(data.columns[:16])
features
```

Out[8]:
```
['age',
 'job',
 'marital',
 'education',
 'default',
 'balance',
 'housing',
 'loan',
 'contact',
 'day',
 'month',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome']
```

In [9]:
```
Y = data["y"]
x = data[features]
```

In [10]:
```
data.boxplot(figsize=(15,10))
```

Out[10]:
```
<AxesSubplot:>
```



Removing outliers for Balance

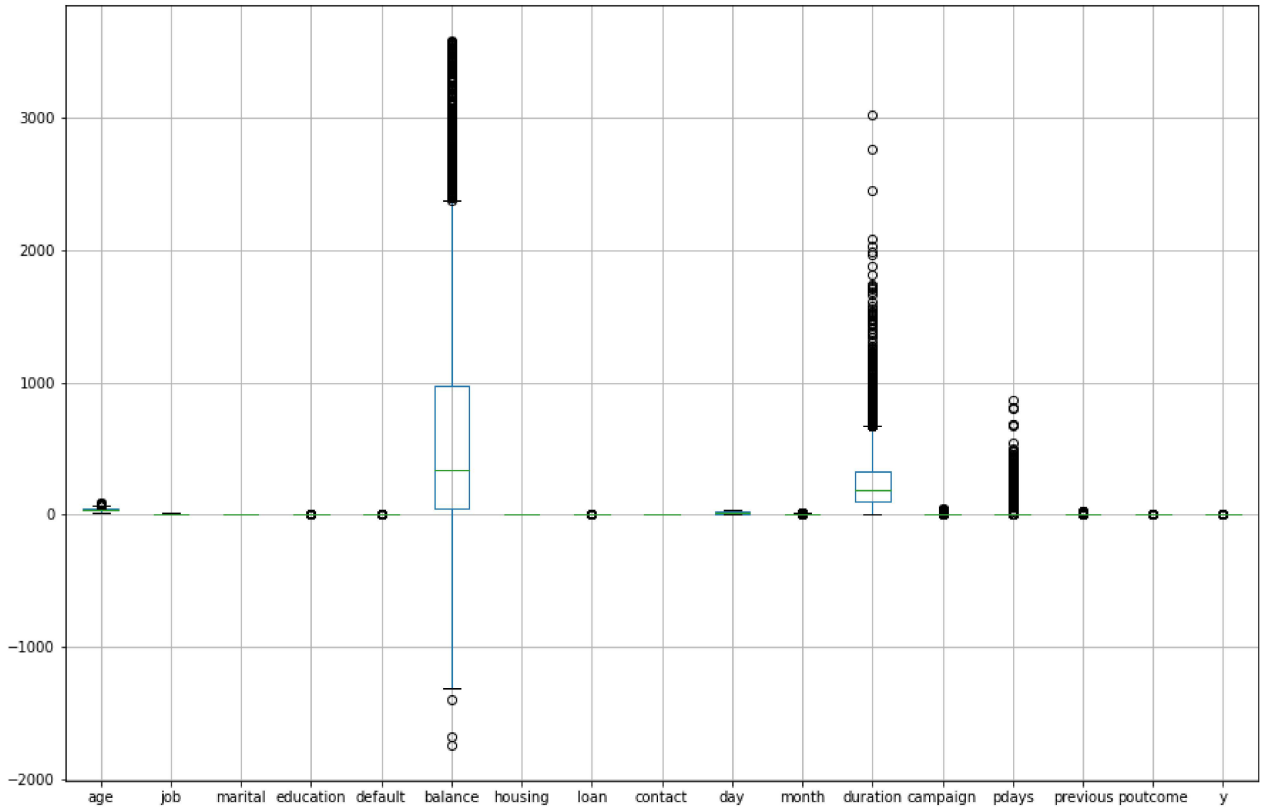
In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
Q1 = np.percentile(data['balance'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(data['balance'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1
# Upper bound
upper = np.where(data['balance'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(data['balance'] <= (Q1-1.5*IQR))

''' Removing the Outliers '''
data.drop(upper[0], inplace = True)
data.drop(lower[0], inplace = True)
```

In [12]:
```
data.boxplot(figsize=(15,10))
```

Out[12]:
```
<AxesSubplot:>
```

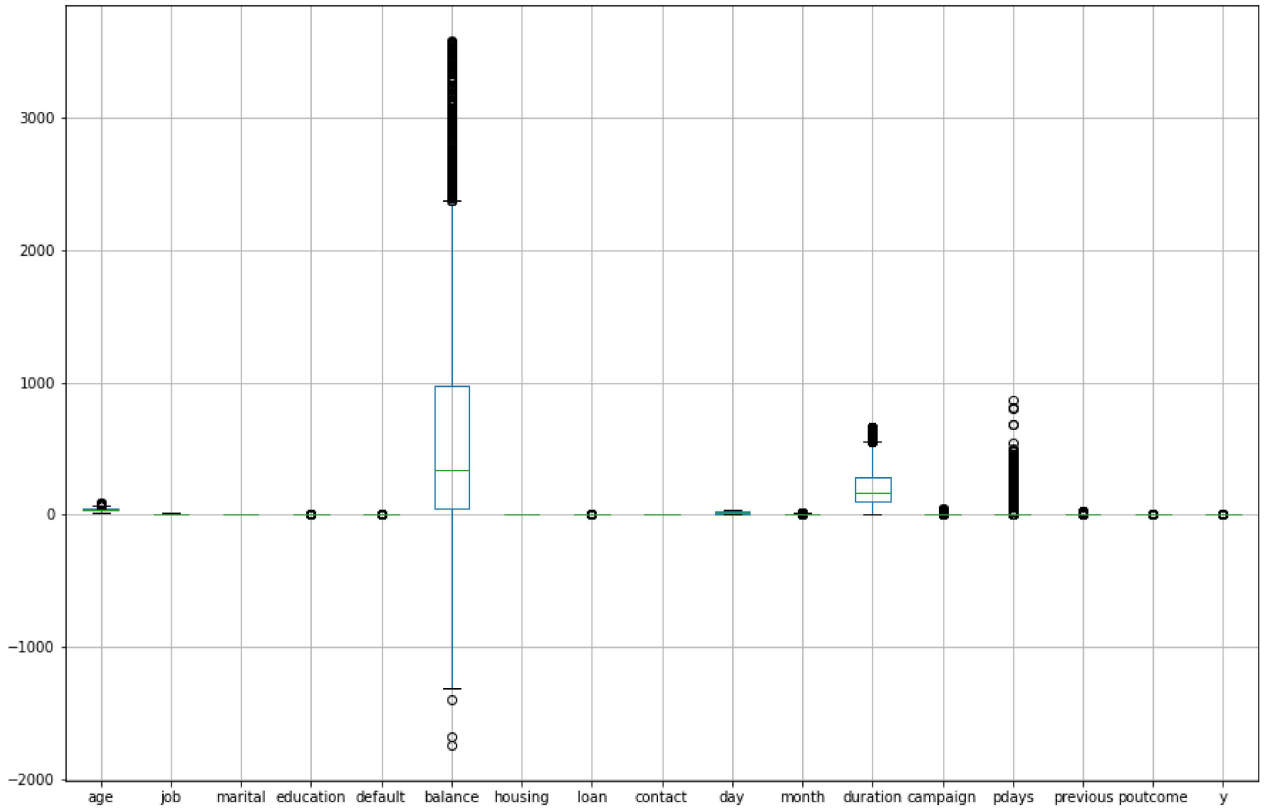


Removing outliers for Duration

In [13]:
```
# Using IQR method to remove outlieres
Q1 = np.quantile(data['duration'],0.25)
Q3 = np.quantile(data['duration'],0.75)
IQR = Q3 - Q1 # defining Inter Quartile Range and developiing upper and lower limit, th
data = data[(data['duration'] > (Q1 - 1.5 * IQR))&(data['duration'] < (Q3 + 1.5 * IQR))
```

In [14]:
```
data.boxplot(figsize=(15,10))
```

Out[14]: <AxesSubplot:>



In [15]:
```
#create a dataframe with all training data except the target column
X = data.drop(columns=['y'])
#check if the target variable has been removed or not
X.head()
```

Out[15]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	can
0	30	11	0	1	0	1787	0	0	1	19	10	79	
2	35	2	1	3	0	1350	1	0	1	16	4	185	
3	30	2	0	3	0	1476	1	1	0	3	6	199	
4	59	3	0	2	0	0	1	0	0	5	5	226	
5	35	2	1	3	0	747	0	0	1	23	2	141	

In [16]:
```
#separate target values
Y = data['y'].values
```

Split data into 75% train and 25% test

In [17]:
```
from sklearn.model_selection import train_test_split
#split dataset into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=
```

Step-2

Decision Tree

In [18]:
```
# Now actually construct the decision tree:

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X,Y)
clf.predict([[10, 1, 4, 0, 2, 3, 5, 6, 0, 0, 0, 0, 0, 0, 0, 0]])
```

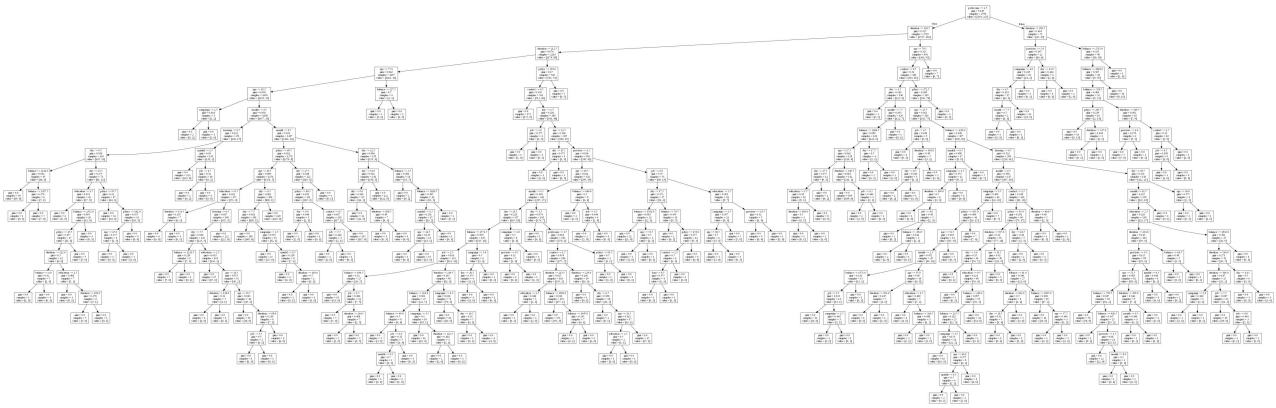
Out[18]:
```
array([1], dtype=int64)
```

In [19]:
```
clf = tree.DecisionTreeClassifier() # Create a decision tree classifer object
clf = clf.fit(X_train,y_train) # Train decision tree Classifer
y_pred_dt = clf.predict(X_test) #Predict the test data
```

In [20]:
```
# Visual representation of Decision Tree
from IPython.display import Image
from six import StringIO
import pydotplus

dot_data = StringIO()
tree.export_graphviz(clf, out_file = dot_data, feature_names = features)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[20]:



In [21]:
```
# Accuracy of decision tree
from sklearn import metrics
print("Decision tree accuracy:",round(metrics.accuracy_score(y_test, y_pred_dt) * 100,2
```

```
Decision tree accuracy: 89.35 %
```

In [22]:
```
# Accuracy of decision tree using 10-fold cross-validation
from sklearn.model_selection import cross_val_score
print(cross_val_score(clf, X, Y, scoring="accuracy", cv = 10))
mean_score = cross_val_score(clf, X, Y, scoring="accuracy", cv = 10)
```

```
[0.88709677 0.8844086  0.88978495 0.89516129 0.8844086  0.91666667
 0.90053763 0.89247312 0.89516129 0.91129032]
```

In [23]:
```
print("Decision Tree accuracy using 10-fold cross-validation: ", round(mean_score.mean(
```

```
Decision Tree accuracy using 10-fold cross-validation:  89.57 %
```

Random Forest

In [24]:
```
#Use ensemble.RandomForestClassifier with n_estimators=10 and use 10-fold cross validat
#to get a measure of the accuracy. Does it perform better than decision tree?
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
```

```
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier(n_estimators=10)
rfc.fit(X_train,y_train)
# predictions
rfc_predict = rfc.predict(X_test)
rfc_cv_score = cross_val_score(rfc, X, Y, cv=10, scoring='roc_auc')
print(rfc_cv_score)
```

```
[0.86661793 0.81676413 0.83425926 0.8248538  0.89371345 0.89605263
 0.85721248 0.82168616 0.8737111  0.87749503]
```

In [25]:
```
# Accuracy of Random Forest
print("Random Forest accuracy: ", round(rfc_cv_score.mean() * 100,2),"%")
```

```
Random Forest accuracy:  85.62 %
```

From the above analysis, we can depict that Decision Tree model (with 89.27% of accuracy) performs comparatively better than Random forest (with 86.16% of accuracy) .

KNN

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier
# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 10)
# Fit the classifier to the data
knn.fit(X_train,y_train)
```

Out[26]:
```
KNeighborsClassifier(n_neighbors=10)
```

In [27]:
```
y_pred = knn.predict(X_test)
print('KNN accuracy: ', round(knn.score(X_test, y_test) * 100,2),"%")
```

```
KNN accuracy:  91.51 %
```

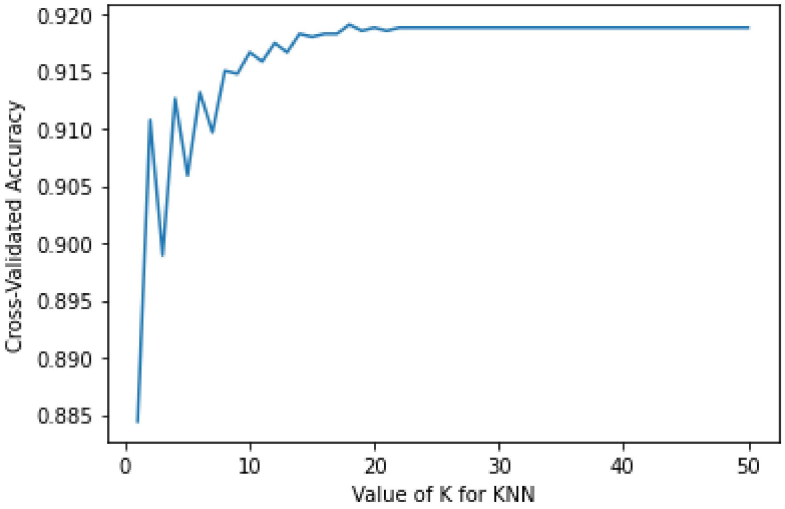
In [28]:
```
#Use neighbors.KNeighborsClassifier with n_neighbors=10 and use 10-fold cross validatio
#get a measure of the accuracy.
# import k-folder
from sklearn.model_selection import cross_val_score
knn = KNeighborsClassifier(n_neighbors = 10)
# X,Y will automatically devided by 5 folder, the scoring I will still use the accuracy
scores = cross_val_score(knn, X, Y, cv=10, scoring='accuracy')
# print all 5 times scores
print(scores)
# then I will do the average about these five scores to get more accuracy score.
print("Mean of scores:",scores.mean())
```

```
[0.91397849 0.91666667 0.91935484 0.91935484 0.91935484 0.91935484
 0.91666667 0.91397849 0.91397849 0.91397849]
Mean of scores: 0.9166666666666666
```

In [29]:
```
#Try different values of K. Write a for loop to run KNN with K values ranging from 1 to
#see if the value of K makes a substantial difference. Make a note of the best performa
# choose k between 1 to 50
k_range = range(1, 51)
k_scores = []
# use iteration to caclulator different k in models, then return the average accuracy b
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, Y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
```

```
print('RMSE value for k= ' , k_range , 'is:', k_scores)
# plot to see clearly
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```

```
RMSE value for k=  range(1, 51) is: [0.8844086021505376, 0.9107526881720431, 0.898924731
1827956, 0.9126344086021504, 0.9059139784946236, 0.9131720430107526, 0.9096774193548386,
0.9150537634408602, 0.914784946236559, 0.9166666666666666, 0.9158602150537634, 0.9174731
182795698, 0.9166666666666666, 0.9182795698924731, 0.918010752688172, 0.918279569892473
1, 0.9182795698924731, 0.9190860215053762, 0.918548387096774, 0.9188172043010752, 0.9185
48387096774, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010
752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.9
188172043010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.918817204
3010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752,
0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.918817
2043010752, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752, 0.91881720430107
52, 0.9188172043010752, 0.9188172043010752, 0.9188172043010752]
```



In [30]:
```
max_value = max(k_scores)
max_index = k_scores.index(max_value)
print("KNN performs well with an accuracy:",round(max_value*100,2),"%. When K value is"
```

```
KNN performs well with an accuracy: 91.91 %. When K value is 17
```

Navie Bayes

In [31]:
```
#Use naive_bayes.GaussianNB and use 10-fold cross validation to get a measure of the ac
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
Acc = cross_val_score(gnb, X, Y, cv=10, scoring="accuracy")
print(Acc)
```

```
[0.83064516 0.84139785 0.8655914  0.86021505 0.84677419 0.89247312
 0.86290323 0.84677419 0.8655914  0.85215054]
```

In [32]:
```
print("Naive Bayes accuracy:",round(Acc.mean() * 100,2),"%")
```

```
Naive Bayes accuracy: 85.65 %
```

In [33]:
```
from sklearn import preprocessing
scale= preprocessing.MinMaxScaler()
data_scale = scale.fit_transform(X)
```

```
X = pd.DataFrame(data_scale, columns =X.columns)
X.head()
```

Out[33]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month
0	0.161765	1.000000	0.0	0.333333	0.0	0.662479	0.0	0.0	0.5	0.600000	0.818182
1	0.235294	0.181818	0.5	1.000000	0.0	0.580536	1.0	0.0	0.5	0.500000	0.272727
2	0.161765	0.181818	0.0	1.000000	0.0	0.604163	1.0	1.0	0.0	0.066667	0.454545
3	0.588235	0.272727	0.0	0.666667	0.0	0.327395	1.0	0.0	0.0	0.133333	0.363636
4	0.235294	0.181818	0.5	1.000000	0.0	0.467467	0.0	0.0	0.5	0.733333	0.090909

Multinomial Navie Bayes

In [34]:
```
from sklearn.naive_bayes import MultinomialNB
Mnb = MultinomialNB()
Acc  = cross_val_score(Mnb, X, Y, cv=10, scoring='accuracy')
print(Acc)
```

```
[0.91935484 0.91935484 0.91935484 0.91935484 0.91935484 0.91935484
 0.91935484 0.91935484 0.91666667 0.91666667]
```

In [35]:
```
print("Multinomial NB accuracy:",round(Acc.mean() * 100,2),"%")
```

```
Multinomial NB accuracy: 91.88 %
```

From the above analysis, we can clearly see that Multinominal Naive Bayes performs better than Gaussian Naive Bayes

Accuracy of all models using 10-fold cross-validation:

Decision Tree accuracy: 89.27 %

Random Forest accuracy: 86.16 %

KNN accuracy: 91.91 %. When K value is 17

Naive Bayes accuracy: 85.65 %

Multinomial NB accuracy: 91.88 %

Hence, after looking at the above accuracy values it canbe observed that KNN has highest accuracy, i.e., 91.91%, which is approximately equal to 92%. Therefore, KNN is the best model for this data set.

References

=> Sun, Q. (2018, May 18). How to deal with cross-validation based on KNN algorithm, compute AUC based on Naive Bayes... Medium. Retrieved December 2, 2021, from https://medium.com/@svanillasun/how-to-deal-with-cross-validation-based-on-knn-algorithm-compute-auc-based-on-naive-bayes-ff4b8284cff4.

=> Allibhai, E. (2018, October 2). Building a K-nearest-neighbors (K-nn) model with Scikit-Learn. Medium. Retrieved December 2, 2021, from https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a.

=> Kumar, N. (n.d.). Implement naive Bayes algorithm using cross validation (cross_val_score) in python. Implement Naive Bayes Algorithm using Cross Validation (cross_val_score) in Python. Retrieved December 2, 2021, from http://theprofessionalspoint.blogspot.com/2019/02/implement-naive-bayes-algorithm-using.html.

=> K-fold cross validation. James LeDoux's Blog. (2019, June 1). Retrieved December 2, 2021, from https://jamesrledoux.com/code/k_fold_cross_validation.

=> Vectorization, multinomial naive Bayes classifier and evaluation. ritchieng.github.io. (n.d.). Retrieved December 2, 2021, from https://www.ritchieng.com/machine-learning-multinomial-naive-bayes-vectorization/.

=> Detect and remove the outliers using Python. GeeksforGeeks. (2021, August 10). Retrieved December 4, 2021, from https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/.

In [ ]: