# Name: Navneet Yadav

# Roll no. 2301560044

# Program: MCA-I ¶

# Subject: AI-ML

---

## Question 1

GitHub: https://github.com/navYadav20/AI-ML

## Classification Modelling

1. Data Cleaning and Regulation
2. Normalization
3. Train Test Splite
4. Classification Methods
   - Logistic Regression, Decision Tree, Random Forest, Navie Byes, SVM, KNN
5. Visualization
6. Conclusion

---

In [3]:
```
import pandas as pd
import numpy as np
```

In [6]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pyplot as plt
```

In [9]:
```
data = pd.read_csv("D:/ai-ml assignment/assignment 4/bank.csv")
```

# Observing Data

In [13]:
```
data.head()
```

Out[13]:

| | age | job | marital | education | default | balance | housing | loan | contact | day | mor |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | m |
| **1** | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | m |
| **2** | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | m |
| **3** | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | m |
| **4** | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | m |

In [14]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  y          45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In [15]:
```
data.describe()
```

Out[15]:

| | age | balance | day | duration | campaign | pdays |
|---|---|---|---|---|---|---|
| **count** | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 |
| **mean** | 40.936210 | 1362.272058 | 15.806419 | 258.163080 | 2.763841 | 40.197828 |
| **std** | 10.618762 | 3044.765829 | 8.322476 | 257.527812 | 3.098021 | 100.128746 |
| **min** | 18.000000 | -8019.000000 | 1.000000 | 0.000000 | 1.000000 | -1.000000 |
| **25%** | 33.000000 | 72.000000 | 8.000000 | 103.000000 | 1.000000 | -1.000000 |
| **50%** | 39.000000 | 448.000000 | 16.000000 | 180.000000 | 2.000000 | -1.000000 |
| **75%** | 48.000000 | 1428.000000 | 21.000000 | 319.000000 | 3.000000 | -1.000000 |
| **max** | 95.000000 | 102127.000000 | 31.000000 | 4918.000000 | 63.000000 | 871.000000 |

In [16]:
```
# we delete columns whose data type is text
data1= data.drop(["job","education","contact","month","poutcome"],axis=1)
```

In [17]:
```
data1["default"] = [0 if each== "no" else 1 for each in data1.default]
```

In [18]:
```
data1["loan"] = [0 if each== "no" else 1 for each in data1.loan]
```

In [19]:
```
data1["y"] = [0 if each== "no" else 1 for each in data1.y]
```

In [20]:
```
data1["marital"] = [1 if each == "married" else 0 if each == "single" else
```

In [21]:
```
data1["housing"] = [1 if each == "yes" else 0  for each in data1.housing]
```

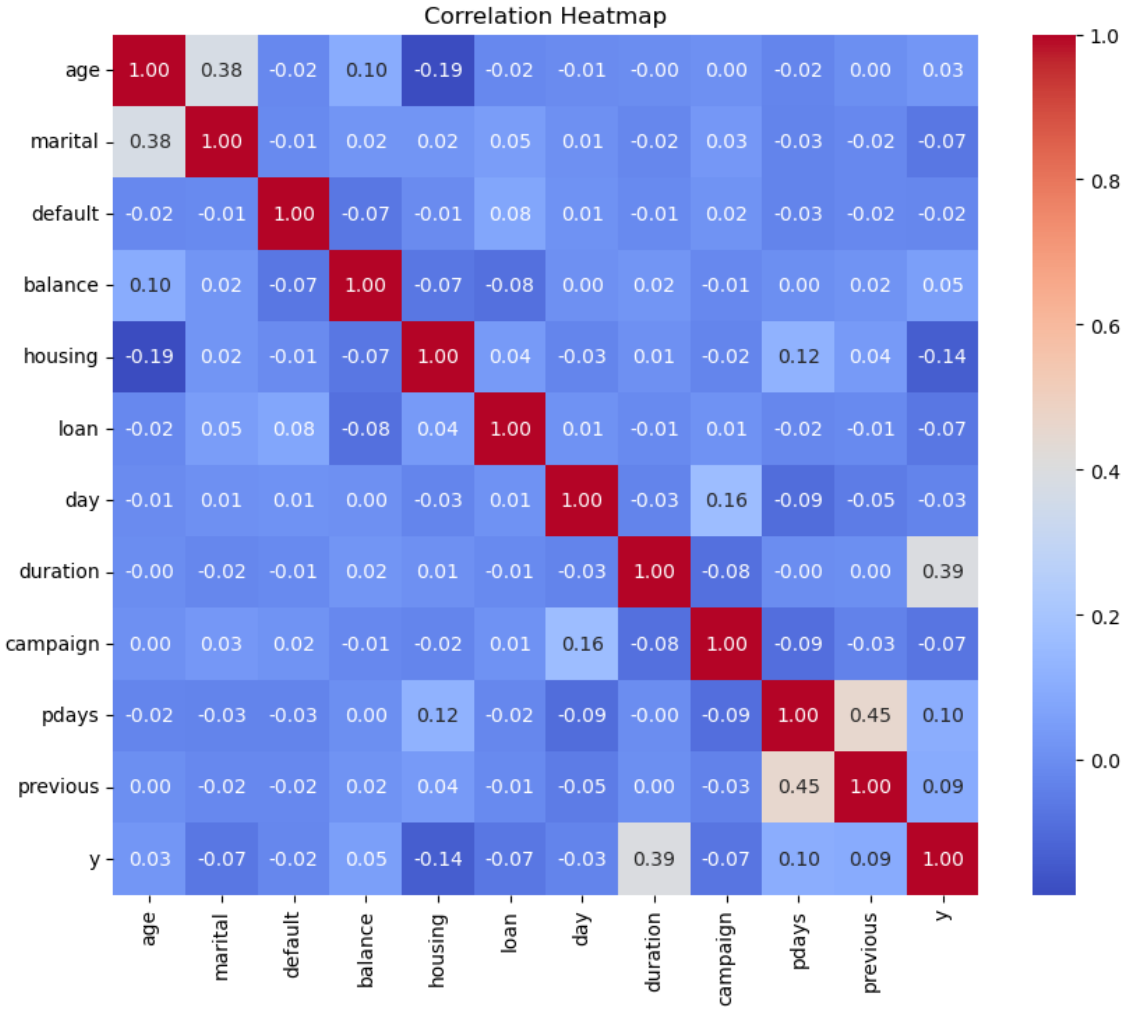
In [22]:
```
data1.head()
```

Out[22]:

| | age | marital | default | balance | housing | loan | day | duration | campaign | pdays | previous |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 58 | 1.0 | 0 | 2143 | 1 | 0 | 5 | 261 | 1 | -1 | 0 |
| **1** | 44 | 0.0 | 0 | 29 | 1 | 0 | 5 | 151 | 1 | -1 | 0 |
| **2** | 33 | 1.0 | 0 | 2 | 1 | 1 | 5 | 76 | 1 | -1 | 0 |
| **3** | 47 | 1.0 | 0 | 1506 | 1 | 0 | 5 | 92 | 1 | -1 | 0 |
| **4** | 33 | 0.0 | 0 | 1 | 0 | 0 | 5 | 198 | 1 | -1 | 0 |

### Correlation heatmap

In [24]:
```
import seaborn as sns
correlation_matrix = data1.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```




## Normalization

In [25]:
```
# Defining to y and x_data values for train data
y = data1.y.values
x_data = data1.drop(["y"],axis=1)
```

In [26]:
```
# normalization
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data))
```

## Train Test Splite

In [27]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =  train_test_split(x,y,test_size=0.15,ran
```

In [28]:
```
# list one save results of models other list save name of model
labelList = []
resultList = []
```

# Classification Methods

## Logictic Regression

In [29]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
print("test accuracy {}".format(lr.score(x_test,y_test)))

# adding result and label to lists
labelList.append("Log_Rec")
resultList.append(lr.score(x_test,y_test))
```

```
test accuracy 0.8802713063992923
```

## Decision Tree

In [30]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)

print("decison tree score : ",dt.score(x_test,y_test))

# adding result and label to lists
labelList.append("Dec_Tree")
resultList.append(dt.score(x_test,y_test))
```

```
decison tree score :  0.8522559716897671
```

## Random Forest

In [31]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100,random_state = 1)
rf.fit(x_train, y_train)
print("Random forest result: ",rf.score(x_test,y_test))

labelList.append("Rand_For")
resultList.append(rf.score(x_test,y_test))
```

```
Random forest result:  0.893246829843704
```

# Navie Byes

In [32]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train,y_train)
print("Accuracy of naive bayes algo: ",nb.score(x_test,y_test))

# adding result and label to lists
labelList.append("Naive_Byes")
resultList.append(nb.score(x_test,y_test))
```

```
Accuracy of naive bayes algo:  0.8641993512238277
```

## SVM Model

In [34]:
```
from sklearn.svm import SVC
svm = SVC(random_state=3)
svm.fit(x_train,y_train)
print("Accuracy of svm algo: ",svm.score(x_test,y_test))

# adding result and label to lists
labelList.append("SVM")
resultList.append(svm.score(x_test,y_test))
```

```
Accuracy of svm algo:  0.8810085520495429
```

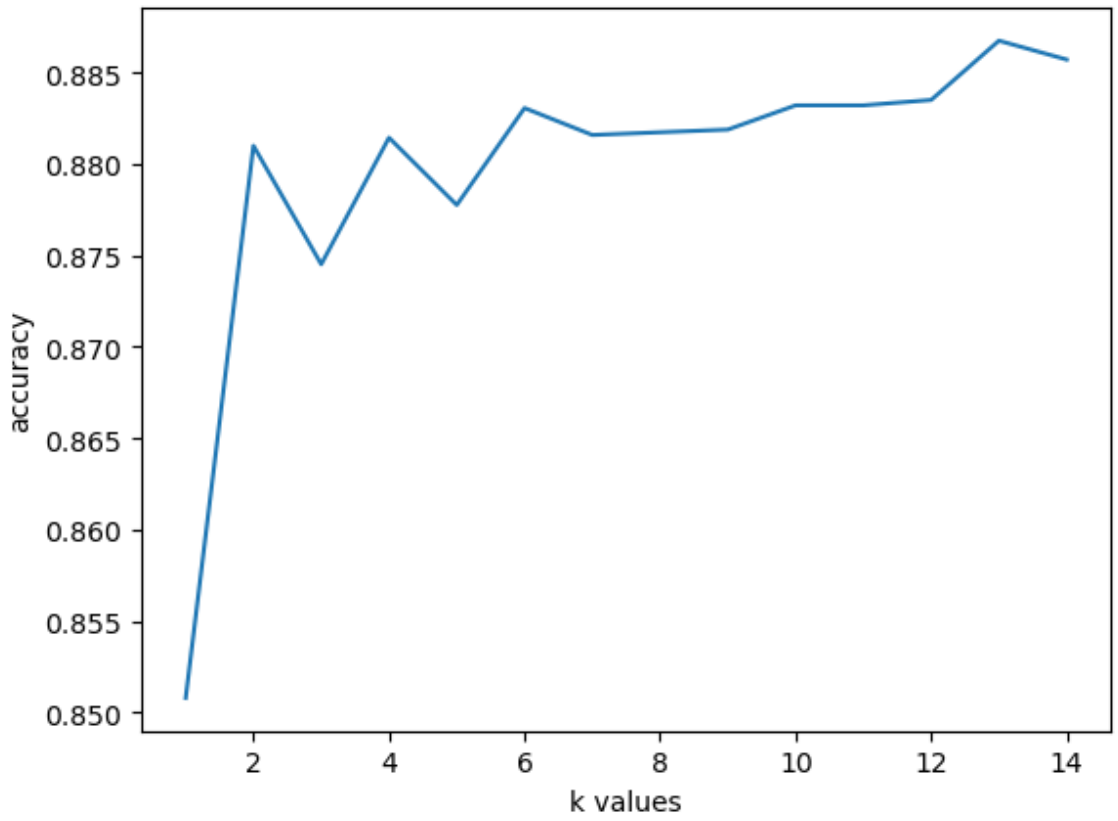
# KNN model

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3) #n_neighbors = k
knn.fit(x_train,y_train)
prediction = knn.predict(x_test)

# score
print(" {} nn score: {} ".format(3,knn.score(x_test,y_test)))
```

```
 3 nn score: 0.8745207903273371
```

- Finding optimum k value between 1 and 15

In [36]:
```
score_list = []
for each in range(1,15):
    knn2 = KNeighborsClassifier(n_neighbors = each) # create a new knn mode
    knn2.fit(x_train,y_train)
    score_list.append(knn2.score(x_test,y_test))

plt.plot(range(1,15),score_list) # x axis is in interval of 1 and 15
plt.xlabel("k values")
plt.ylabel("accuracy")
plt.show()
```

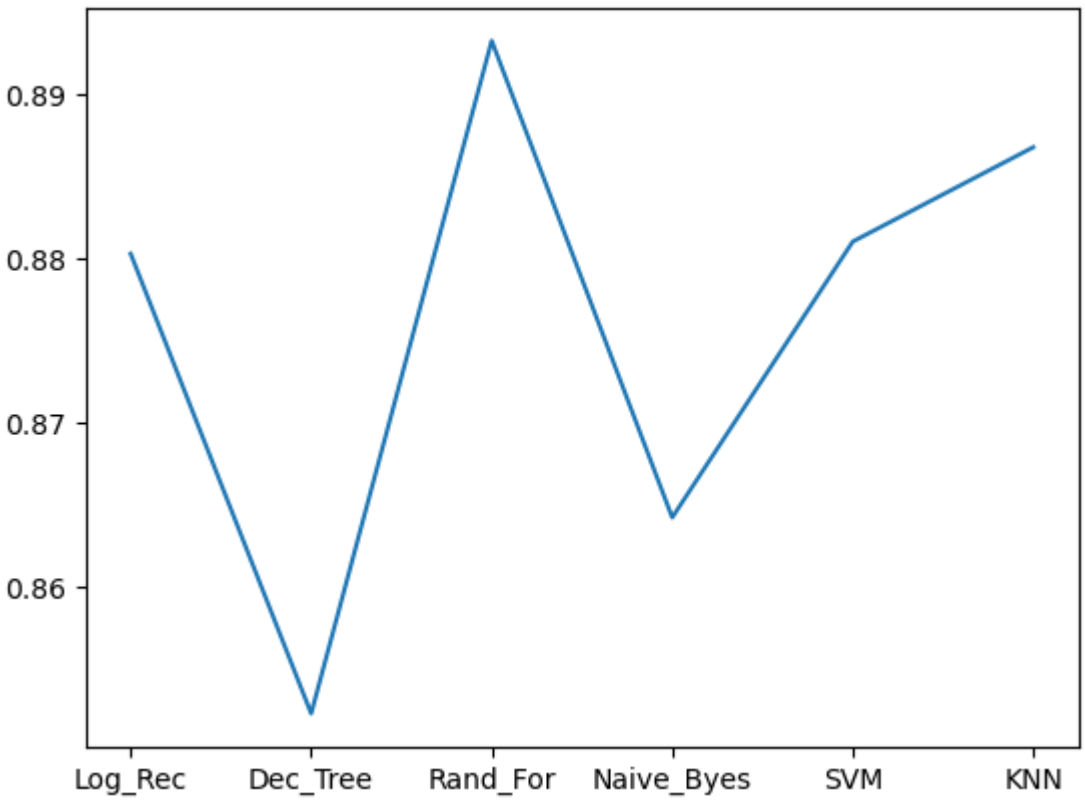



In [37]:
```
# finding max value in a list and it's index.
a = max(score_list) # finding max value in list
b = score_list.index(a)+1 # index of max value.

print("k = ",b," and maximum value is ", a)

# adding result and label to lists
labelList.append("KNN")
resultList.append(a)
```

```
k =  13  and maximum value is  0.8867590681214981
```

# Visualization

In [38]:
```
plt.plot(labelList,resultList)
plt.show()
```




In [39]:
```
zipped = zip(labelList, resultList)
zipped = list(zipped)
```

In [40]:
```
df = pd.DataFrame(zipped, columns=['label','result'])
df
```

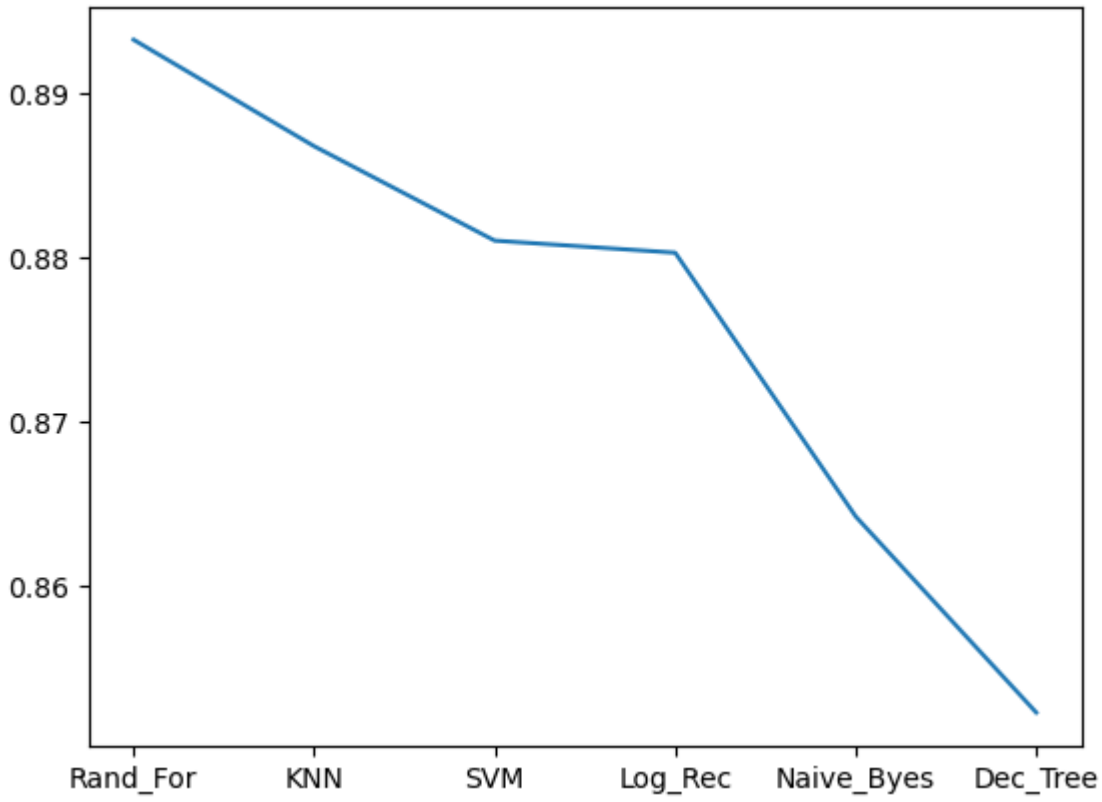
Out[40]:

| | label | result |
|---|---|---|
| **0** | Log_Rec | 0.880271 |
| **1** | Dec_Tree | 0.852256 |
| **2** | Rand_For | 0.893247 |
| **3** | Naive_Byes | 0.864199 |
| **4** | Naive_Byes | 0.864199 |
| **5** | SVM | 0.881009 |
| **6** | SVM | 0.881009 |
| **7** | KNN | 0.886759 |

- results of classification algorithms with sorted and clear chart

In [41]:
```
new_index = (df['result'].sort_values(ascending=False)).index.values
sorted_data = df.reindex(new_index)
```

In [42]:
```
plt.plot(sorted_data.loc[:,"label"],sorted_data.loc[:,"result"])
plt.show()
```




In [43]:
```
sorted_data
```

Out[43]:

| | label | result |
|---|---|---|
| **2** | Rand_For | 0.893247 |
| **7** | KNN | 0.886759 |
| **5** | SVM | 0.881009 |
| **6** | SVM | 0.881009 |
| **0** | Log_Rec | 0.880271 |
| **3** | Naive_Byes | 0.864199 |
| **4** | Naive_Byes | 0.864199 |
| **1** | Dec_Tree | 0.852256 |

In [ ]:
```

```