Importing libraries

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Reading Data

```python
df=pd.read_csv(r'C:\Users\Nikhil\Downloads\test.csv')
```

```python
df
```

	Id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	1	35	management	married	tertiary	no	231	yes	no	unknown	5	may	139	1	-1	0	unknown
1	2	28	management	single	tertiary	no	447	yes	yes	unknown	5	may	217	1	-1	0	unknown
2	3	42	entrepreneur	divorced	tertiary	yes	2	yes	no	unknown	5	may	380	1	-1	0	unknown
3	4	58	retired	married	primary	no	121	yes	no	unknown	5	may	50	1	-1	0	unknown
4	5	43	technician	single	secondary	no	593	yes	no	unknown	5	may	55	1	-1	0	unknown
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
422	423	26	blue-collar	single	secondary	no	82	yes	no	unknown	9	may	654	1	-1	0	unknown
423	424	40	blue-collar	married	secondary	no	10	yes	no	unknown	9	may	1692	2	-1	0	unknown
424	425	28	blue-collar	single	secondary	no	-197	yes	no	unknown	9	may	2016	2	-1	0	unknown
425	426	33	blue-collar	divorced	primary	no	390	yes	no	unknown	9	may	665	2	-1	0	unknown
426	427	45	blue-collar	married	primary	no	101	yes	no	unknown	12	may	460	5	-1	0	unknown

427 rows × 17 columns

Data Cleaning

Checking duplicate

```python
df.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
422    False
423    False
424    False
425    False
426    False
Length: 427, dtype: bool
```

columns names

```python
df.columns
```

```
Index(['Id', 'age', 'job', 'marital', 'education', 'default', 'balance',
       'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign',
       'pdays', 'previous', 'poutcome'],
      dtype='object')
```

Rows in each columns

```python
df.count()
```

```
Id           427
age          427
job          427
marital      427
education    427
default      427
balance      427
housing      427
loan         427
contact      427
day          427
month        427
duration     427
campaign     427
pdays        427
previous     427
poutcome     427
dtype: int64
```

shape of dataset

```python
df.shape
```

```
(427, 17)
```

checking null values

```python
df.isna().sum()
```

```
Id           0
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
dtype: int64
```

droping unnecessary column

```python
df.drop(['contact','month','pdays','previous','poutcome','housing','campaign','day','duration'],axis=1,inplace=True)
```

```python
df.drop(['Id'],axis=1,inplace=True)
```

Top 5 rows

```python
df.head()
```

	age	job	marital	education	default	balance	loan
0	35	management	married	tertiary	no	231	no
1	28	management	single	tertiary	no	447	yes
2	42	entrepreneur	divorced	tertiary	yes	2	no
3	58	retired	married	primary	no	121	no
4	43	technician	single	secondary	no	593	no

checking unique values in all columns

```python
df.nunique()
```

```
age           40
job           12
marital        3
education      4
default        2
balance      310
loan           2
dtype: int64
```

checking unique values for particular column

```python
df['education'].unique()
```

```
array(['tertiary', 'primary', 'secondary', 'unknown'], dtype=object)
```

```python
edu=df.education.value_counts()
edu
```

```
secondary    242
tertiary      87
primary       58
unknown       40
Name: education, dtype: int64
```

```python
df['job'].unique()
```

```
array(['management', 'entrepreneur', 'retired', 'technician', 'admin.',
       'services', 'blue-collar', 'self-employed', 'unemployed',
       'housemaid', 'student', 'unknown'], dtype=object)
```

```python
df['marital'].unique()
```

```
array(['married', 'single', 'divorced'], dtype=object)
```

```python
df['default'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```python
df['loan'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

checking types of columns

```python
df.dtypes
```

```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
loan         object
dtype: object
```

stastical information

```python
df.describe()
```

	age	balance
count	427.000000	427.000000
mean	43.971897	640.302108
std	9.589908	2722.544765
min	22.000000	-674.000000
25%	36.000000	23.500000
50%	43.000000	168.000000
75%	53.000000	446.000000
max	61.000000	45248.000000

Prediction

Converting data types into integer

```python
df.job=df.job.replace({'management':1,'entrepreneur':2,'retired':3,'technician':4,'admin.':5,'services':6,'blue-collar':7,'self-employed':8,'unemployed':9,'hou
```

```python
df.marital=df.marital.replace({'married':1,'single':2,'divorced':3,'unknown':4})
```

```python
df.education=df.education.replace({'primary':1,'secondary':2,'tertiary':3,'unknown':4})
```

```python
df.loan=df.loan.replace({'yes':1,'no':2})
```

```python
df.default=df.default.replace({'yes':1,'no':2})
```

```python
df.head()
```

	age	job	marital	education	default	balance	loan
0	35	1	1	3	2	231	2
1	28	1	2	3	2	447	1
2	42	2	3	3	1	2	2
3	58	3	1	1	2	121	2
4	43	4	2	2	2	593	2

```python
df.dtypes
```

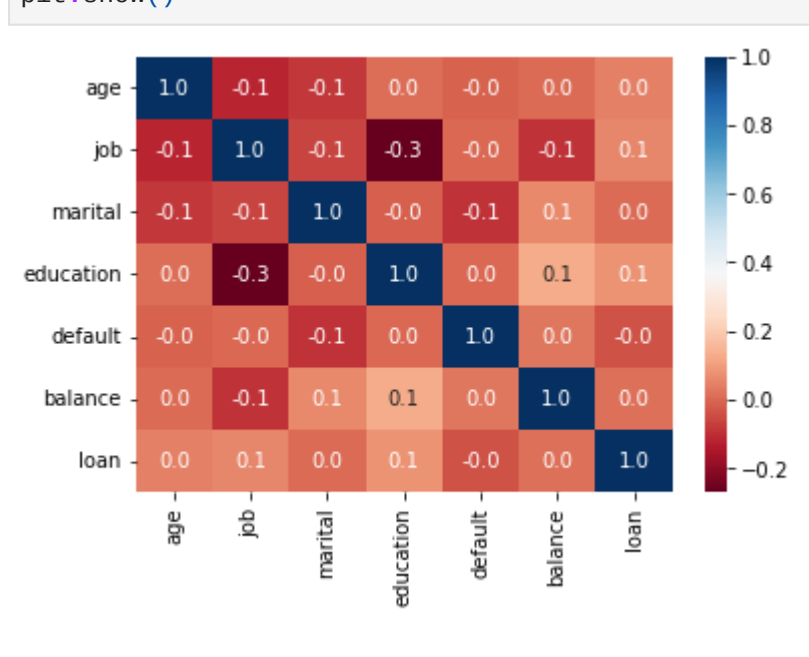
```
age          int64
job          int64
marital      int64
education    int64
default      int64
balance      int64
loan         int64
dtype: object
```

Heat Map

```python
df.corr()
```

	age	job	marital	education	default	balance	loan
age	1.000000	-0.119984	-0.085721	0.010033	-0.010437	0.004254	0.043394
job	-0.119984	1.000000	-0.064153	-0.270193	-0.001241	-0.092210	0.056652
marital	-0.085721	-0.064153	1.000000	-0.015460	-0.095898	0.058969	0.006856
education	0.010033	-0.270193	-0.015460	1.000000	0.000636	0.127697	0.078441
default	-0.010437	-0.001241	-0.095898	0.000636	1.000000	0.022897	-0.037391
balance	0.004254	-0.092210	0.058969	0.127697	0.022897	1.000000	0.014398
loan	0.043394	0.056652	0.006856	0.078441	-0.037391	0.014398	1.000000

```python
sns.heatmap(df.corr(),cmap='RdBu',annot=True,fmt='.1f')
plt.show()
```



We can see there is a positive correlation between age & loan This makes sense since, The greater age results in a greater chance of loan.

In addition, we see a negative correlation between job & our predictor i.e-loan.

Seprating dependent and independent variable

```python
x=df.drop(['loan'],axis=1)
y=df.loan
```

```python
x
```

	age	job	marital	education	default	balance
0	35	1	1	3	2	231
1	28	1	2	3	2	447
2	42	2	3	3	1	2
3	58	3	1	1	2	121
4	43	4	2	2	2	593
...	...	...	...	...	...	...
422	26	7	2	2	2	82
423	40	7	1	2	2	10
424	28	7	2	2	2	-197
425	33	7	3	1	2	390
426	45	7	1	1	2	101

427 rows × 6 columns

```python
y
```

```
0      2
1      1
2      2
3      2
4      2
      ..
422    2
423    2
424    2
425    2
426    2
Name: loan, Length: 427, dtype: int64
```

```python
y.value_counts()
```

```
2    372
1     55
Name: loan, dtype: int64
```

Importing train-test split

```python
from sklearn.model_selection import train_test_split
```

```python
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

```python
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(341, 6)
(86, 6)
(341,)
(86,)
```

1-Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
```

```python
model=LogisticRegression()
```

```python
model
```

```
LogisticRegression()
```

```python
model.fit(x_train,y_train)
```

```
LogisticRegression()
```

```python
model.score(x_train,y_train)*100
```

```
86.21700879765396
```

```python
model.score(x_test,y_test)*100
```

```
90.69767441860465
```

```python
y_predict=model.predict(x_test)
```

```python
y_predict
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
      dtype=int64)
```

```python
df1=pd.DataFrame({'Actual':y_test,'Predicted':y_predict})
```

```python
df1
```

	Actual	Predicted
61	2	2
94	2	2
178	2	2
145	2	2
229	2	2
...	...	...
11	2	2
359	2	2
10	2	2
389	2	2
172	2	2

86 rows × 2 columns

Checking accuracy

```python
from sklearn.metrics import accuracy_score
```

```python
test_acc = accuracy_score(y_test,y_predict)*100
```

```python
test_acc
```

```
90.69767441860465
```

importing confusion matrix

```python
from sklearn.metrics import confusion_matrix
```

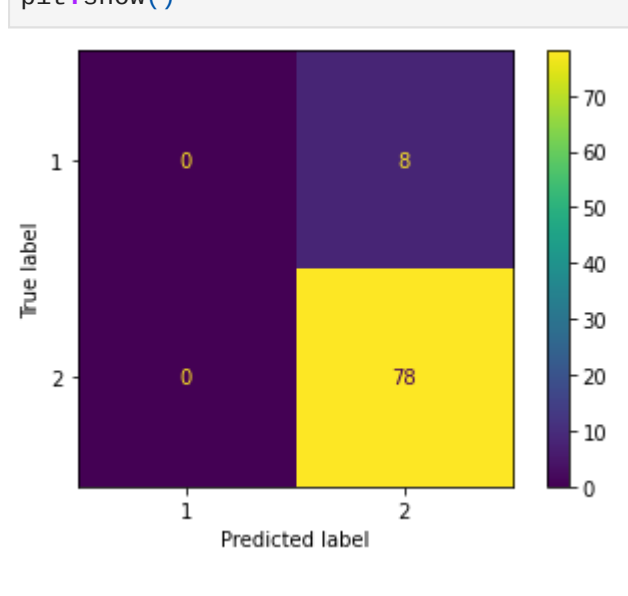
```python
performance=confusion_matrix(y_test,y_predict)
```

```python
performance
```

```
array([[ 0,  8],
       [ 0, 78]], dtype=int64)
```

```python
from sklearn.metrics import plot_confusion_matrix
```

```python
plot_confusion_matrix(model,x_test,y_test)
plt.show()
```



0 is the amount of True Positives in our data, while 78 is the amount of True Negatives.

8 & 0 are the number of errors.

There are 8 type 1 error (False Positives)- You predicted positive and it's false.

There are 0 type 2 error (False Negatives)- You predicted negative and it's false.

Hence if we calculate the accuracy its # Correct Predicted/ # Total. In other words, where TP, FN, FP and TN represent the number of true positives, false negatives, false positives and true negatives.

(TP + TN)/(TP + TN + FP + FN). (0 + 78)/(0+78+8+0) = 0.9069 = 90.69% accuracy

Accuracy = 90.69%

AS we are getting near by same accuracy in both training and testing so our model is balkanced fit.

Classification report

```python
from sklearn.metrics import classification_report
```

```python
performance=classification_report(y_test,y_predict)
```

```python
print(performance)
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         8
           2       0.91      1.00      0.95        78

    accuracy                           0.91        86
   macro avg       0.45      0.50      0.48        86
weighted avg       0.82      0.91      0.86        86
```

2-Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier
```

```python
model_dt=DecisionTreeClassifier()
```

```python
model_dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier()
```

```python
model_dt.score(x_train,y_train)*100
```

```
100.0
```

```python
model_dt.score(x_test,y_test)*100
```

```
75.5813953488372
```

```python
from sklearn.metrics import accuracy_score
```

```python
y_predict=model_dt.predict(x_test)
```

```python
y_predict
```

```
array([2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2,
       2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
      dtype=int64)
```

```python
test_acc = accuracy_score(y_test,y_predict)*100
```

```python
test_acc
```

```
75.5813953488372
```

3-Random Forest

```python
from sklearn.ensemble import RandomForestClassifier
```

```python
model_rf=RandomForestClassifier(n_estimators=100)
```

```python
model_rf.fit(x_train,y_train)
```

```
RandomForestClassifier()
```

```python
model_rf.score(x_train,y_train)*100
```

```
100.0
```

```python
model_rf.score(x_test,y_test)*100
```

```
90.69767441860465
```

```python
from sklearn.metrics import accuracy_score
```

```python
y_predict=model_rf.predict(x_test)
```

```python
y_predict
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
      dtype=int64)
```

```python
test_acc = accuracy_score(y_test,y_predict)*100
```

```python
test_acc
```

```
90.69767441860465
```

4-KNeighborsClassifier

```python
from sklearn.neighbors import KNeighborsClassifier
```

```python
model_knn=KNeighborsClassifier(n_neighbors=10)
```

```python
model_knn.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=10)
```

```python
model_knn.score(x_train,y_train)*100
```

```
86.21700879765396
```

```python
model_knn.score(x_test,y_test)*100
```

```
89.53488372093024
```

```python
from sklearn.metrics import accuracy_score
```

```python
y_predict=model_knn.predict(x_test)
```

```python
y_predict
```

```
array([2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
      dtype=int64)
```

```python
test_acc = accuracy_score(y_test,y_predict)*100
```

```python
test_acc
```

```
89.53488372093024
```

Observation

From comparing the 4 models, we can conclude that Logistic regression and RandomForestClassifier Model has highest accuracy. With an accuracy of 90.69%.

We have precision, recall, f1-score and support:

Precision : be "how many are correctly classified among that class"

Recall : "how many of this class you find over the whole number of element of this class"

F1-score : harmonic mean of precision and recall values. F1 score reaches its best value at 1 and worst value at 0. F1 Score = 2 x ((precision x recall) / (precision + recall))

Support: of samples of the true response that lie in that class.

Analysis of result

We used precision, F1-score, recall and accuracy evaluation metrics for evaluating our models.

False Positive(FP) is when a model incorrectly predicts a positive outcome.

False Negative(FN) is when a model incorrectly predicts the negative outcome.

True Positive(TP) is when model correctly predicts a positive outcome.

True Negative(TN) is when a model correctly predicts a negative outcome.

Precision=TP/(TP+FP) =

Recall = TP / (TP + FN)

F1 score = 2 *precision* Recall / (precision + Recall)

Machine Learning Models - Accuracy

1 - Logistic Regression - 74.50

2 - Decision Tree - 82.43

3 - Random Forest - 82.98

4 - KNeighborsClassifier - 82.50

5 - Naive bayes - 72.72

Conclusions

Our Random Forest algorithmand Logistic regression yields the highest accuracy, 90.69 %. Any accuracy above 80% is considered good. Thus, 90.69 % is the ideal accuracy! and Random forest model has and Logistic Regression Highest testing score i.e-90.69% and score of train data is 86%.

Out of the 17 features we examined the 6 features i.e-age,job,marital,education,default,balance features that helped us classify between a yes and no (loan)

As we are getting high accuracy in training as well as testing dataset,it is example of Right fit.