In [94]:
```
import numpy as np
import pandas as pd
```

In [95]:
```
df= pd.read_csv('bank-market.csv')
```

In [96]:
```
df.head()
```

Out[96]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no

In [97]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        4521 non-null   int64 
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64 
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64 
 10  month      4521 non-null   object
 11  duration   4521 non-null   int64 
 12  campaign   4521 non-null   int64 
 13  pdays      4521 non-null   int64 
 14  previous   4521 non-null   int64 
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

In [98]:
```
#getting columns with categorical data
cat_cols = [c for c in df.columns if df[c].dtypes=='O']  # Make lists with categorical and numerical variables:
num_cols = [c for c in df.columns if df[c].dtypes!='O']
cat_cols
```

Out[98]:
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'poutcome',
 'y']
```

Conversion to numerics

Converting job into binary

In [99]:
```
df.job.value_counts()
```

Out[99]:
```
management       969
blue-collar      946
technician       768
admin.           478
services         417
retired          230
self-employed    183
entrepreneur     168
unemployed       128
housemaid        112
student           84
unknown           38
Name: job, dtype: int64
```

In [100]:
```
df['job'] = df['job'].map({'management':1, 'blue-collar':2, 'technician':3, 'admin.':4, 'services':5, 'retired':6, 'self-employe
```

Converting marital into binary

In [101]:
```
df.marital.value_counts()
```

Out[101]:
```
married     2797
single      1196
divorced     528
Name: marital, dtype: int64
```

In [102]:
```
Counts_val = df['marital'].value_counts()
mask = df['marital'].isin(Counts_val[Counts_val<1197].index)
df['marital'][mask] = 'other'
print(pd.value_counts(df['marital']))
```

```
married    2797
other      1724
Name: marital, dtype: int64

C:\Users\Huzefa\AppData\Local\Temp\ipykernel_13668\58358692.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['marital'][mask] = 'other'
```

In [103]:
```
df['marital'] = df['marital'].map({'married':1, 'other':0}) # Re-code the 'marital' parameter as binary:
```

Converting education into binary

In [104]:
```
df.education.value_counts()
```

Out[104]:
```
secondary    2306
tertiary     1350
primary       678
unknown       187
Name: education, dtype: int64
```

In [105]:
```
df['education'] = df['education'].map({'secondary':2, 'tertiary':3, 'primary':1, 'unknown':0}) # Re-code the 'education' paramete
```

Converting default into binary

In [106]:
```
df.default.value_counts()
```

Out[106]:
```
no     4445
yes      76
Name: default, dtype: int64
```

In [107]:
```
df['default'] = df['default'].map({'yes':1, 'no':0}) # Re-code the 'default' parameter as binary:
```

Converting housing into binary

In [108]:
```
df.housing.value_counts()
```

Out[108]:
```
yes    2559
no     1962
Name: housing, dtype: int64
```

In [109]:
```
df['housing'] = df['housing'].map({'yes':1, 'no':0}) # Re-code the 'housing' parameter as binary:
```

Converting loan into binary

In [110]:
```
df.loan.value_counts()
```

Out[110]:
```
no     3830
yes     691
Name: loan, dtype: int64
```

In [111]:
```
df['loan'] = df['loan'].map({'yes':1, 'no':0}) # Re-code the 'loan' parameter as binary:
```

Converting contact into binary

In [112]:
```
df.contact.value_counts()
```

Out[112]:
```
cellular     2896
unknown      1324
telephone     301
Name: contact, dtype: int64
```

In [113]:
```
df['contact'] = df['contact'].map({'cellular':1, 'unknown':0, 'telephone':2}) # Re-code the 'contact' parameter as binary:
```

Converting month into binary

In [114]:
```
df.month.value_counts()
```

Out[114]:
```
may    1398
jul     706
aug     633
jun     531
nov     389
apr     293
feb     222
jan     148
oct      80
sep      52
mar      49
dec      20
Name: month, dtype: int64
```

In [115]:
```
# Re-code the 'month' parameter as binary:
df['month'] = df['month'].map({'may':5, 'jul':7, 'aug':8, 'jun':6, 'nov':11, 'apr':4, 'feb':2, 'jan':1, 'oct':10, 'sep':9, 'mar'
```

Converting y into binary

In [116]:
```
df.y.value_counts()
```

Out[116]:
```
no     4000
yes     521
Name: y, dtype: int64
```

In [117]:
```
df['y'] = df['y'].map({'yes':1, 'no':0}) # Re-code the 'y' parameter as binary:
```

Converting poutcome into binary

In [118]:
```
df.poutcome.value_counts()
```

Out[118]:
```
unknown    3705
failure     490
other       197
success     129
Name: poutcome, dtype: int64
```

In [119]:
```
df['poutcome'] = df['poutcome'].map({'unknown':0, 'failure':-1, 'other':0, 'success':1}) # Re-code the 'poutcome' parameter as bi
```

In [120]:
```
df.poutcome.value_counts()
```

Out[120]:
```
 0    3902
-1     490
 1     129
Name: poutcome, dtype: int64
```

Logistic Regression

Logistic Regression : Hold Out method (30%-70%)

In [121]:
```
X = df.drop('y', axis=1)
```

In [122]:
```
y = df['y']
```

In [123]:
```
from sklearn.model_selection import train_test_split
```

In [124]:
```
trainX, testX, trainY, testY = train_test_split(X,y, test_size=0.3, random_state=42)
```

In [125]:
```
from sklearn.linear_model import LogisticRegression  #import
```

In [126]:
```
modelLogR = LogisticRegression()     # object
```

In [127]:
```
modelLogR.fit(trainX,trainY)
```

```
C:\Users\Huzefa\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/line
ar_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[127]:
```
LogisticRegression()
```

In [128]:
```
preLogR = modelLogR.predict(testX)
```

In [129]:
```
from sklearn.metrics import classification_report, accuracy_score
```

In [130]:
```
accuracy_score(preLogR,testY)
```

Out[130]:
```
0.8887251289609432
```

In [131]:
```
print(classification_report(testY,preLogR))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1205
           1       0.51      0.17      0.26       152

    accuracy                           0.89      1357
   macro avg       0.71      0.58      0.60      1357
weighted avg       0.86      0.89      0.86      1357
```

Logistic Regression : Simple K-fold Method

In [132]:
```
# evaluate a logistic regression model using k-fold cross-validation
import numpy as np
import pandas as pd
#from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold
from sklearn.linear_model import LogisticRegression
```

In [133]:
```
# Simple Kfold
kf1 = KFold(n_splits=10, random_state=42, shuffle=True)
# Stratified K-fold
Skf1 = StratifiedKFold(n_splits=10, random_state=42, shuffle=True)
```

In [134]:
```
scores_cv1_accuracy = cross_val_score(modelLogR, X, y, scoring='accuracy', cv=kf1, n_jobs=-1)
scores_cv1_precision = cross_val_score(modelLogR, X, y, scoring='precision', cv=kf1, n_jobs=-1)
scores_cv1_recall = cross_val_score(modelLogR, X, y, scoring='recall', cv=kf1, n_jobs=-1)
```

In [135]:
```
# --- Simple Kfold ---
print("Accuracy :",scores_cv1_accuracy.mean())
print("Precision :",scores_cv1_precision.mean())
print("Recall :",scores_cv1_recall.mean())
```

```
Accuracy : 0.8847589325831722
Precision : 0.49279054279054285
Recall : 0.1560202971416586
```

Logistic Regression : Stratified K-fold method

In [136]:
```
scores_skf1_accuracy = cross_val_score(modelLogR, X, y, scoring='accuracy', cv=Skf1, n_jobs=-1)
scores_skf1_precision = cross_val_score(modelLogR, X, y, scoring='precision', cv=Skf1, n_jobs=-1)
scores_skf1_recall = cross_val_score(modelLogR, X, y, scoring='recall', cv=Skf1, n_jobs=-1)
```

In [137]:
```
# --- stratified Kfold ----
print("Accuracy :",scores_skf1_accuracy.mean())
print("Precision :",scores_skf1_precision.mean())
print("Recall :",scores_skf1_recall.mean())
```

```
Accuracy : 0.8847613745140557
Precision : 0.5139573906485673
Recall : 0.15348330914368652
```

KNN

KNN : Hold Out method (30%-70%)

In [138]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [139]:
```
modelKNN = KNeighborsClassifier(n_neighbors=12)
```

In [140]:
```
modelKNN.fit(trainX,trainY)
```

Out[140]:
```
KNeighborsClassifier(n_neighbors=12)
```

In [141]:
```
preKNN = modelKNN.predict(testX)
```

```
C:\Users\Huzefa\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction fun
ctions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, t
his behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will
be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [143]:
```
from sklearn.metrics import classification_report
```

In [144]:
```
print(classification_report(testY,preKNN))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1205
           1       0.50      0.15      0.23       152

    accuracy                           0.89      1357
   macro avg       0.70      0.57      0.59      1357
weighted avg       0.86      0.89      0.86      1357
```

KNN : Simple K-fold Method

In [145]:
```
# Simple Kfold
kf2 = KFold(n_splits=8, random_state=42, shuffle=True)
# Stratified K-fold
Skf2 = StratifiedKFold(n_splits=8, random_state=42, shuffle=True)
```

In [146]:
```
# --- Simple Kfold ---
scores_cv2_accuracy = cross_val_score(modelKNN, X, y, scoring='accuracy', cv=kf2, n_jobs=-1)
scores_cv2_precision = cross_val_score(modelKNN, X, y, scoring='precision', cv=kf2, n_jobs=-1)
scores_cv2_recall = cross_val_score(modelKNN, X, y, scoring='recall', cv=kf2, n_jobs=-1)
```

In [147]:
```
# --- Simple Kfold ---
print("Accuracy :",scores_cv2_accuracy.mean())
print("Precision :",scores_cv2_precision.mean())
print("Recall :",scores_cv2_recall.mean())
```

```
Accuracy : 0.8867479439632259
Precision : 0.5513418571163136
Recall : 0.11926851478011112
```

KNN : Stratified K-fold method

In [148]:
```
# --- stratified Kfold ----
scores_skf2_accuracy = cross_val_score(modelKNN, X, y, scoring='accuracy', cv=Skf2, n_jobs=-1)
scores_skf2_precision = cross_val_score(modelKNN, X, y, scoring='precision', cv=Skf2, n_jobs=-1)
scores_skf2_recall = cross_val_score(modelKNN, X, y, scoring='recall', cv=Skf2, n_jobs=-1)
```

In [149]:
```
# --- stratified Kfold ----
print("Accuracy :",scores_skf2_accuracy.mean())
print("Precision :",scores_skf2_precision.mean())
print("Recall :",scores_skf2_recall.mean())
```

```
Accuracy : 0.8852023984489822
Precision : 0.5125254953379953
Recall : 0.10174825174825175
```

ML Algo.	Splitting	Accuracy	Precision	Recall	F1-Score
Logistic Regression	Hold Out method (30%-70%)	89%	51%	17%	26%
	Simple K-fold Method	88%	49%	16%	
	Stratified K-fold method	88%	51%	15%	
KNN	Hold Out method (30%-70%)	89%	50%	15%	23%
	Simple K-fold Method	89%	55%	12%	
	Stratified K-fold method	89%	51%	10%	