

Bank Marketing Analysis

```
In [1]: #importing the library
import numpy as np
import pandas as pd
```

Data Collection

```
In [2]: #data loading in panda
data=pd.read_csv('bank.csv')
```

```
In [3]: #check first five rows of the dataset
data.head()
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	mon
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	m
1	56	admin.	married	secondary	no	45	no	no	unknown	5	m
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	m
3	55	services	married	secondary	no	2476	yes	no	unknown	5	m
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	m

```
In [4]: #check last five rows pf the dataset
data.tail()
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	20
11158	39	services	married	secondary	no	733	no	no	unknown	16
11159	32	technician	single	secondary	no	29	no	no	cellular	19
11160	43	technician	married	secondary	no	0	no	yes	cellular	8
11161	34	technician	married	secondary	no	0	no	no	cellular	9

```
In [3]: #check basic infomation of the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype  
---  --
 0   age             11162 non-null  int64  
 1   job             11162 non-null  object  
 2   marital         11162 non-null  object  
 3   education       11162 non-null  object  
 4   default         11162 non-null  object  
 5   balance         11162 non-null  int64  
 6   housing         11162 non-null  object  
 7   loan            11162 non-null  object  
 8   contact         11162 non-null  object  
 9   day             11162 non-null  int64  
10  month           11162 non-null  object  
11  duration        11162 non-null  int64  
12  campaign        11162 non-null  int64  
13  pdays           11162 non-null  int64  
14  previous        11162 non-null  int64  
15  poutcome        11162 non-null  object  
16  deposit         11162 non-null  object  
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
```

```
In [4]: #check columns name of the dataset
data.columns
```

```
Out[4]: Index(['age', 'job', 'marital', 'education', 'default', 'balance',
              'housing',
              'loan', 'contact', 'day', 'month', 'duration', 'campaign',
              'pdays',
              'previous', 'poutcome', 'deposit'],
              dtype='object')
```

In [5]: `data.head(10)`

Out [5]:

	age	job	marital	education	default	balance	housing	loan	contact	day
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5
1	56	admin.	married	secondary	no	45	no	no	unknown	5
2	41	technician	married	secondary	no	1270	yes	no	unknown	5
3	55	services	married	secondary	no	2476	yes	no	unknown	5
4	54	admin.	married	tertiary	no	184	no	no	unknown	5
5	42	management	single	tertiary	no	0	yes	yes	unknown	5
6	56	management	married	tertiary	no	830	yes	yes	unknown	6
7	60	retired	divorced	secondary	no	545	yes	no	unknown	6
8	37	technician	married	secondary	no	1	yes	no	unknown	6
9	28	services	single	secondary	no	5090	yes	no	unknown	6

In [6]: *#check mathamatic realtionship of the dataset*
`data.describe()`

Out [6]:

	age	balance	day	duration	campaign	pday:
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
mean	41.231948	1528.538524	15.658036	371.993818	2.508421	51.330400
std	11.913369	3225.413326	8.420740	347.128386	2.722077	108.758280
min	18.000000	-6847.000000	1.000000	2.000000	1.000000	-1.000000
25%	32.000000	122.000000	8.000000	138.000000	1.000000	-1.000000
50%	39.000000	550.000000	15.000000	255.000000	2.000000	-1.000000
75%	49.000000	1708.000000	22.000000	496.000000	3.000000	20.750000
max	95.000000	81204.000000	31.000000	3881.000000	63.000000	854.000000

1. Age / Age
2. Job / Job
3. Marital Status / Marital Status
4. Education / Education Level
5. Default / Having a previously broken credit
6. Housing / home loan?
7. Loan / Personal Loan?
8. Contact / Was the customer contacted on his home or mobile phone?
9. Month: Last month of contact
10. Day: The day of the contacted.
11. Duration: Talk time on last call
12. Campaign: The number of contacts reaching the customer during the current campaign (including the last contact)
13. Pdays: The number of days since the previous campaign, if reached (-1 if it was never reached before)
14. Previous: The number of contacts that reached the customer before this campaign
15. Poutcome: Previous campaign success, failure or failure

Univariate Variable Analysis

- Categorical Variables: job, marital, default, education, housing, loan, contact, poutcome, month, deposit, day
- Numerical Variables: age, campaign, duration, pdays, balance, previous

Categorical Variable

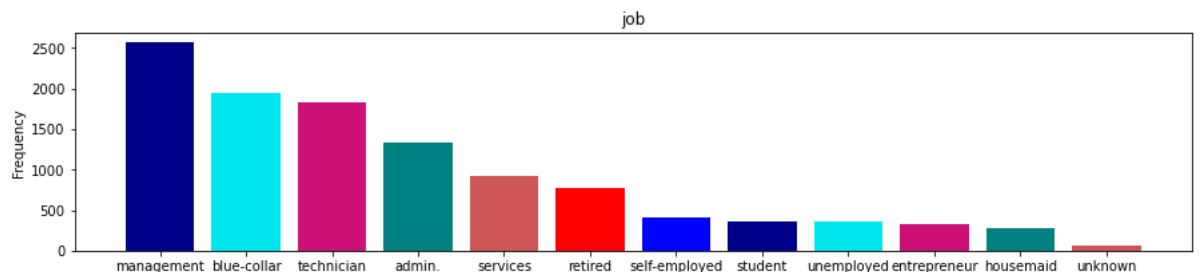
```
In [7]: import matplotlib.pyplot as plt
```

In [8]:

```
def bar_plot(variable):
    var = data[variable]
    varValue = var.value_counts()
    plt.figure(figsize=(15,3))
    plt.bar(varValue.index, varValue,color=['#00008b','#00e5ee','#c
    plt.xticks(varValue.index, varValue.index.values)
    plt.ylabel("Frequency")
    plt.title(variable)

    plt.show()
    print("{}: \n {}".format(variable,varValue))
```

```
In [9]: categoryc = ["job","marital","education", "housing", "loan","contac
for c in categoryc:
    bar_plot(c)
```

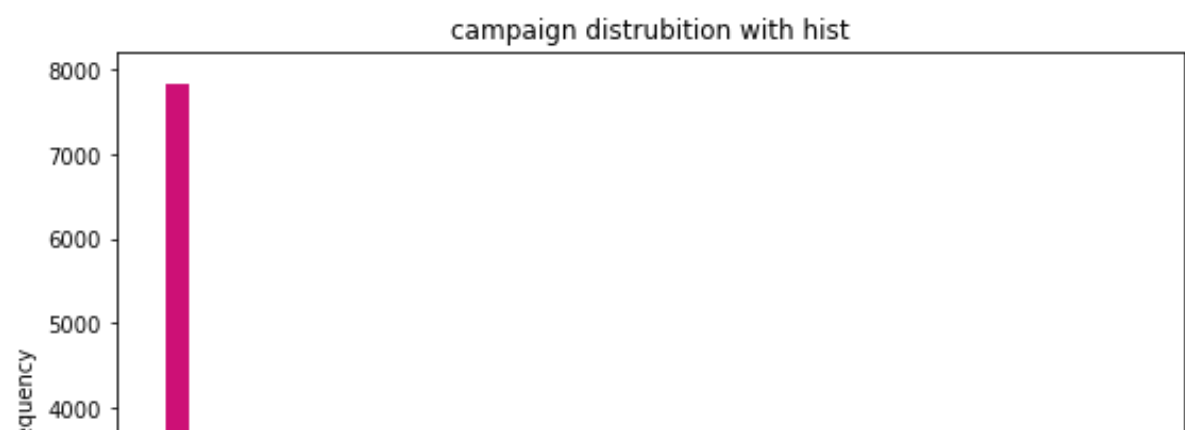
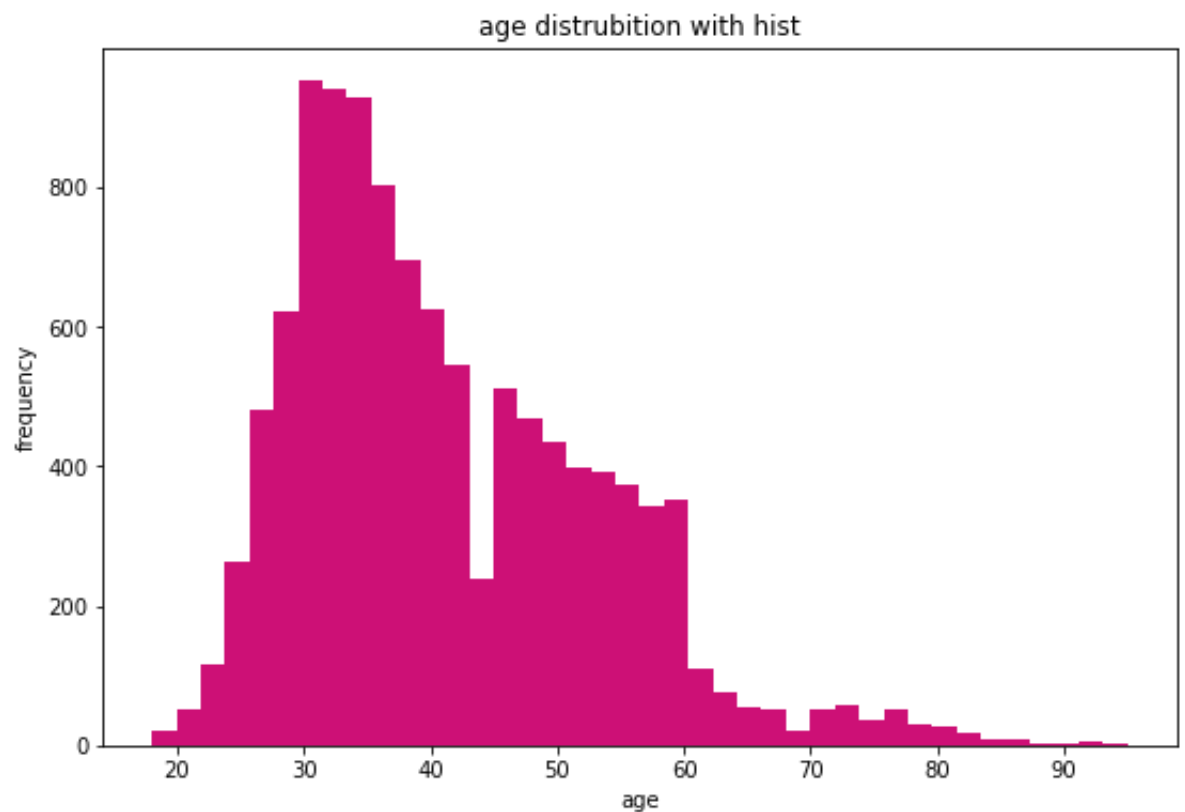


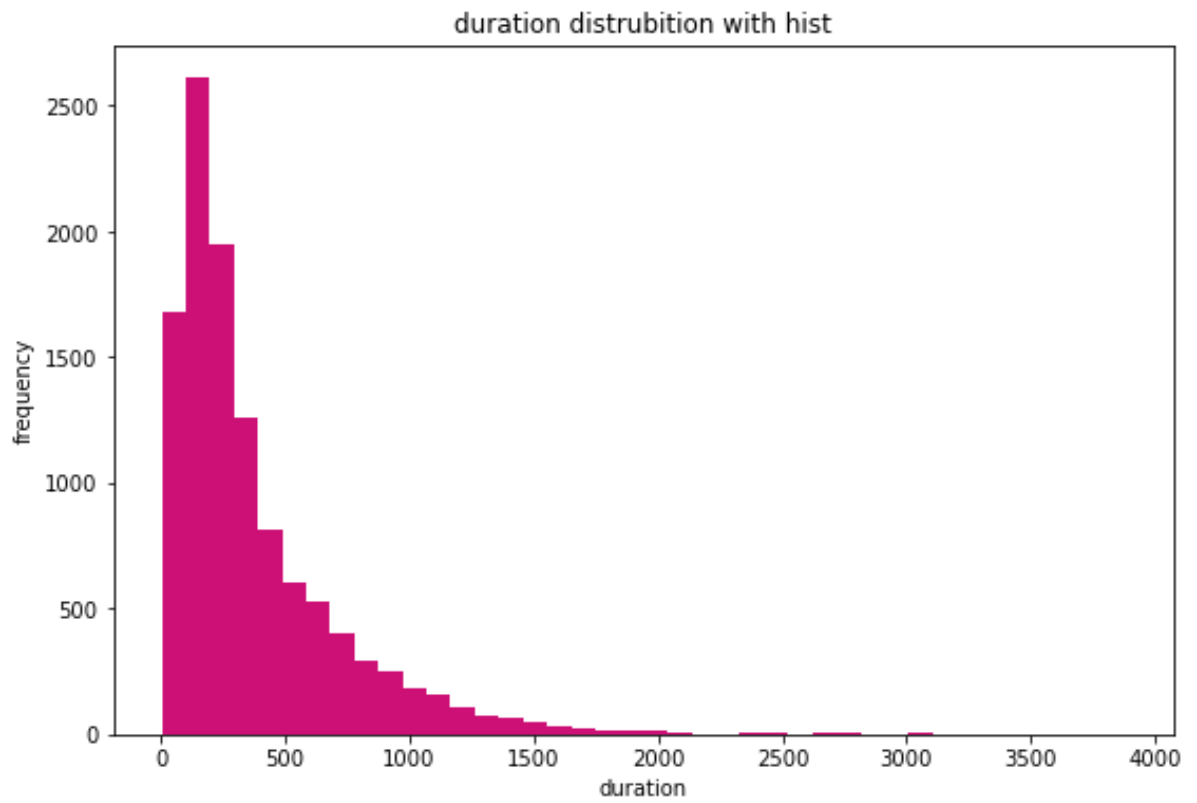
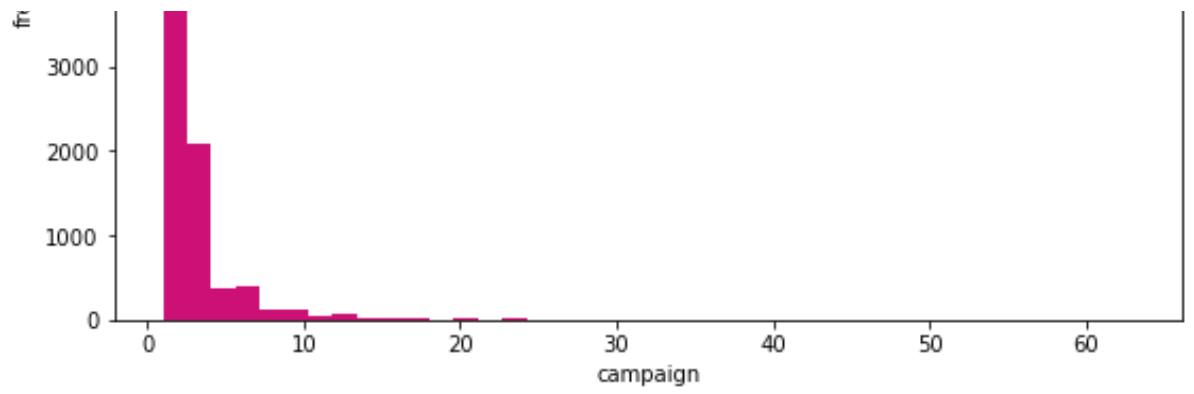
```
job:
  management      2566
blue-collar      1944
technician       1823
admin.           1334
services          923
retired           778
self-employed    405
student           360
unemployed        357
entrepreneur     328
```

Numerical Variable

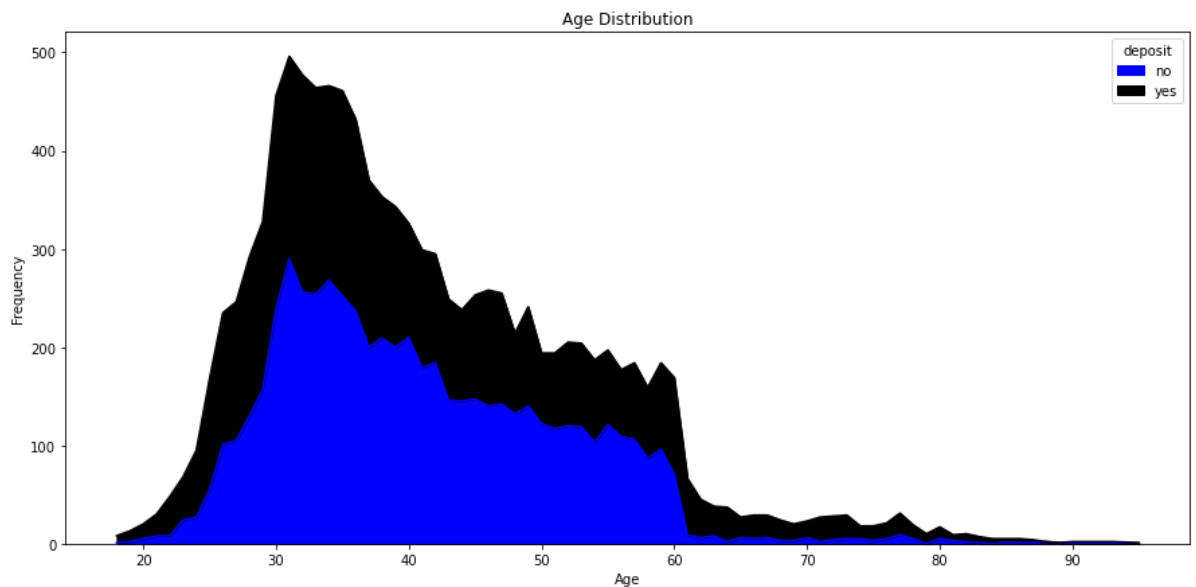
```
In [10]: def plot_hist(variable):  
    plt.figure(figsize=(9,6))  
    plt.hist(data[variable], bins=40,color='#cd1076')  
    plt.xlabel(variable)  
    plt.ylabel("frequency")  
    plt.title("{} distrubition with hist".format(variable))  
    plt.show()
```

```
In [11]: numericVar = ["age","campaign","duration"]  
    for n in numericVar:  
        plot_hist(n)
```



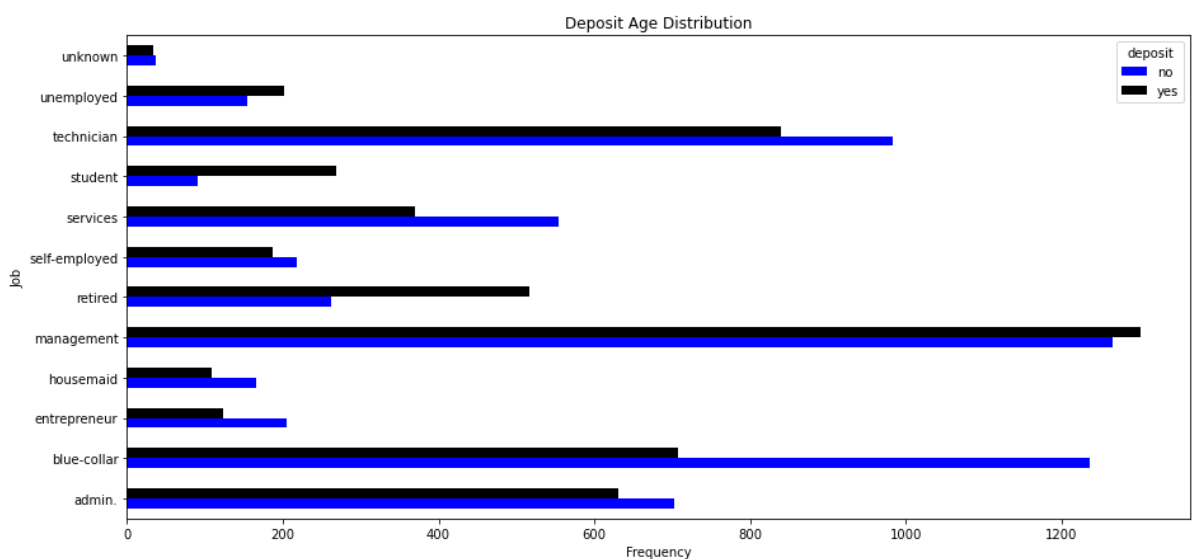


```
In [12]: pd.crosstab(data.age,data.deposit).plot(kind="area",figsize=(15,7),
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



The number of people who are 25 to 40 years old with a time deposit account is high.

```
In [13]: pd.crosstab(data.job,data.deposit).plot(kind="barh",figsize=(15,7),
plt.title('Deposit Age Distribution')
plt.xlabel('Frequency')
plt.ylabel('Job')
plt.show()
```



In people at the executive level have more deposit accounts.

Outlier Detection

```
In [14]: from collections import Counter
def detect_outliers(data, features):
    outlier_indices = []
    for c in features:
        # 1st quartile
        Q1 = np.percentile(data[c], 25)
        # 3rd quartile
        Q3 = np.percentile(data[c], 75)
        # IQR
        IQR = Q3 - Q1
        # Outlier step
        outlier_step = IQR * 1.5
        # detect outlier and their indeces
        outlier_list_col = data[(data[c] < Q1 - outlier_step) | (data[c] > Q3 + outlier_step)]
        # store indeces
        outlier_indices.extend(outlier_list_col)

    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list(i for i, v in outlier_indices.items() if v > 1)

    return multiple_outliers
```

```
In [15]: data.loc[detect_outliers(data, ['age', 'day', 'duration', 'campaign', 'previous_employment', 'housing', 'loan', 'contact', 'day', 'month'])]
```

```
Out[15]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month
3945	84	retired	married	tertiary	no	4761	no	no	telephone	9	september

I do not include this row

```
In [16]: data = data.drop([3945], axis=0)
```

Missing Value

```
In [17]: data.isnull().sum()
```

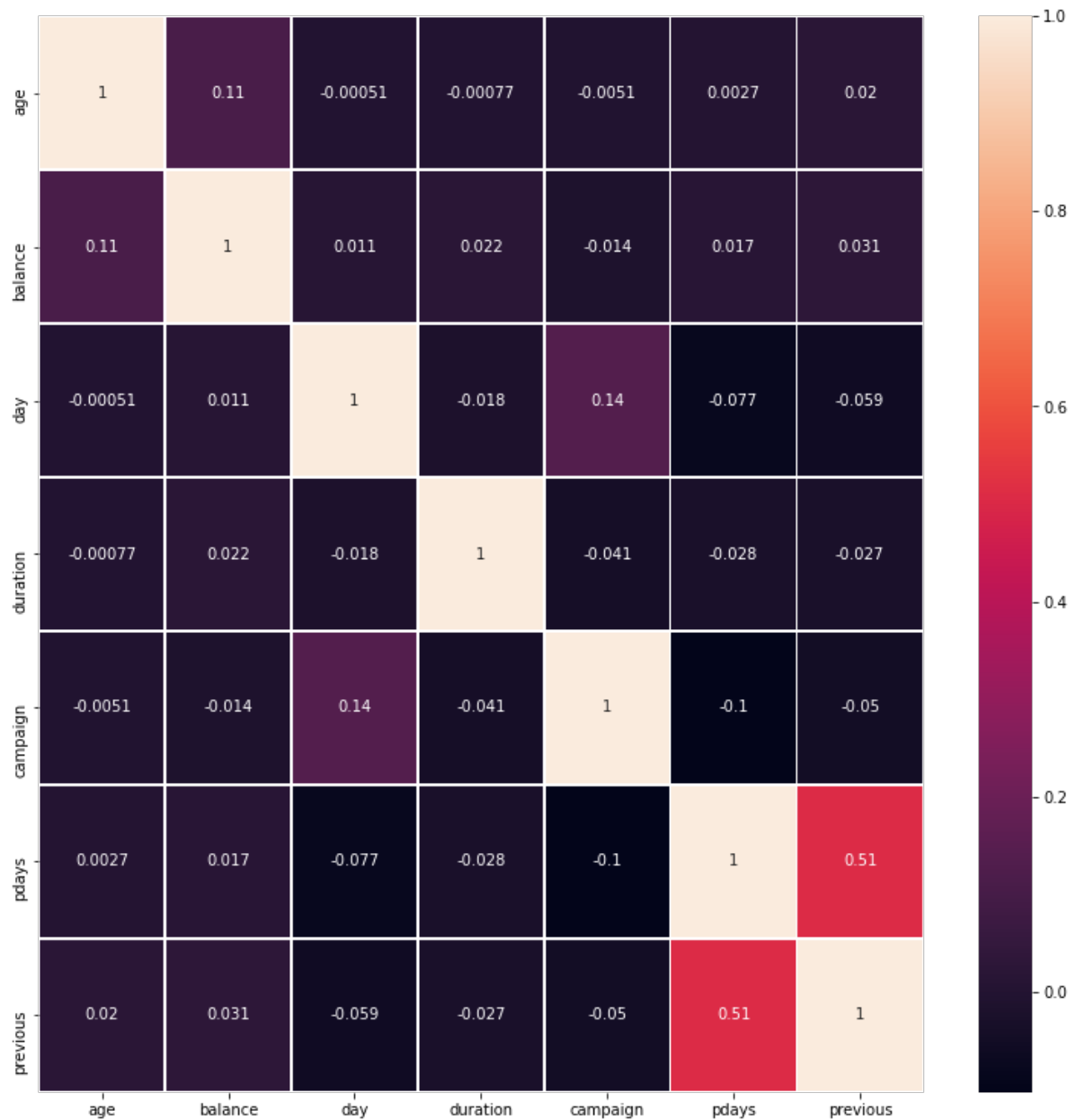
```
Out[17]: age          0  
         job          0  
         marital      0  
         education    0  
         default      0  
         balance      0  
         housing      0  
         loan         0  
         contact      0  
         day          0  
         month        0  
         duration     0  
         campaign     0  
         pdays       0  
         previous     0  
         poutcome     0  
         deposit      0  
         dtype: int64
```

No missing value..

Correlation matrix

```
In [18]: import seaborn as sns
fig, ax = plt.subplots(figsize=(13,13)) # Sample figsize in
sns.heatmap(data.corr(), annot=True, linewidths=.5, ax=ax)
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f705038d390>



*** Calculated correlation between two variables (r) gets a value between -1 and 1.**

- No correlaiton $r=0$
- Very weak correlation: $r<0.20$
- Weak correlation: between 0.20-0.49
- Moderate correlation: between 0.5-0.79
- Strong correlation: between 0.8-0.99
- Perfect correlation: $r=1$

Looking at it, there is a moderate correlation between the *days* and the *previous* ones. ($r=0.51$)

Data Manipulation

I do not include the *Duration column* in the dataset, as it is unknown data at the time of the prediction.

duration: Talk Time on Last Call

```
In [19]: data=data.drop(['duration'],axis=1)
```

```
In [20]: data.head()
```

```
Out[20]:
```

	age	job	marital	education	default	balance	housing	loan	contact	day	mon
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	m
1	56	admin.	married	secondary	no	45	no	no	unknown	5	m
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	m
3	55	services	married	secondary	no	2476	yes	no	unknown	5	m
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	m

One-Hot Encoding

One Hot Encoding means that categorical variables are represented as binary.

```
In [21]: columns=data.select_dtypes(include=[object]).columns
data=pd.concat([data,pd.get_dummies(data[columns])],axis=1)
data=data.drop(['job','marital','education','default','housing','lo
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11161 entries, 0 to 11161
Data columns (total 52 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   age                                   11161 non-null  int64
1   balance                             11161 non-null  int64
2   campaign                             11161 non-null  int64
3   pdays                               11161 non-null  int64
4   previous                             11161 non-null  int64
5   deposit                             11161 non-null  object
6   job_admin.                           11161 non-null  uint8
7   job_blue-collar                      11161 non-null  uint8
8   job_entrepreneur                     11161 non-null  uint8
9   job_housemaid                        11161 non-null  uint8
10  job_management                       11161 non-null  uint8
11  job_retired                          11161 non-null  uint8
12  job_self-employed                    11161 non-null  uint8
13  job_services                         11161 non-null  uint8
14  job_student                          11161 non-null  uint8
15  job_technician                       11161 non-null  uint8
16  job_unemployed                       11161 non-null  uint8
17  job_unknown                          11161 non-null  uint8
18  marital_divorced                     11161 non-null  uint8
19  marital_married                      11161 non-null  uint8
20  marital_single                       11161 non-null  uint8
21  education_primary                    11161 non-null  uint8
22  education_secondary                  11161 non-null  uint8
23  education_tertiary                   11161 non-null  uint8
24  education_unknown                    11161 non-null  uint8
25  default_no                           11161 non-null  uint8
26  default_yes                          11161 non-null  uint8
27  housing_no                           11161 non-null  uint8
28  housing_yes                          11161 non-null  uint8
29  loan_no                              11161 non-null  uint8
30  loan_yes                             11161 non-null  uint8
31  contact_cellular                     11161 non-null  uint8
32  contact_telephone                    11161 non-null  uint8
33  contact_unknown                      11161 non-null  uint8
34  month_apr                            11161 non-null  uint8
35  month_aug                            11161 non-null  uint8
36  month_dec                            11161 non-null  uint8
```

```

37 month_feb          11161 non-null uint8
38 month_jan          11161 non-null uint8
39 month_jul          11161 non-null uint8
40 month_jun          11161 non-null uint8
41 month_mar          11161 non-null uint8
42 month_may          11161 non-null uint8
43 month_nov          11161 non-null uint8
44 month_oct          11161 non-null uint8
45 month_sep          11161 non-null uint8
46 poutcome_failure  11161 non-null uint8
47 poutcome_other     11161 non-null uint8
48 poutcome_success   11161 non-null uint8
49 poutcome_unknown   11161 non-null uint8
50 deposit_no         11161 non-null uint8
51 deposit_yes        11161 non-null uint8
dtypes: int64(5), object(1), uint8(46)
memory usage: 1.1+ MB

```

Out [21]:

	age	balance	campaign	pdays	previous	deposit	job_admin.	job_blue-collar	job_entrepreneur
0	59	2343	1	-1	0	yes	1	0	
1	56	45	1	-1	0	yes	1	0	
2	41	1270	1	-1	0	yes	0	0	
3	55	2476	1	-1	0	yes	0	0	
4	54	184	2	-1	0	yes	1	0	

5 rows × 10 columns

Others..

1. The *pdays* data indicates how many times the customer has been contacted before.

Updated as follows.

if the **pdays = 0**, it indicates that it has not been contacted before

if the **pdays = 1**, it indicates that it was contacted earlier

```
In [22]: def pdayswork(pdays):
          if(pdays == -1):
              return(0)
          elif(pdays >= 0):
              return(1)
          data['pdays2'] = data['pdays'].apply(pdayswork)
```

2. For a single target column

```
In [23]: data=data.drop(['deposit_no', 'deposit_yes'],axis=1)
```

```
In [24]: def deposit1(deposit):
          if(deposit=='yes'):
              return(1)
          elif(deposit=='no'):
              return(0)
          data['depositNew'] = data['deposit'].apply(deposit1)
```

```
In [25]: data=data.drop(['deposit'],axis=1)
```

In this way, our target column, whose data type is object, turned into numerical values. And new target column name is *depositNew*. Also as this is a classification problem, the target column can remain as an object. But I chose to convert it to int data type.

the current state of our data set.

```
In [26]: data.head()
```

```
Out [26]:
```

	age	balance	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepreneur	job_l
0	59	2343	1	-1	0	1	0		0
1	56	45	1	-1	0	1	0		0
2	41	1270	1	-1	0	0	0		0
3	55	2476	1	-1	0	0	0		0
4	54	184	2	-1	0	1	0		0

5 rows × 51 columns

```
In [27]:
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11161 entries, 0 to 11161
Data columns (total 51 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   age                                   11161 non-null  int64
1   balance                             11161 non-null  int64
2   campaign                            11161 non-null  int64
3   pdays                               11161 non-null  int64
4   previous                            11161 non-null  int64
5   job_admin.                          11161 non-null  uint8
6   job_blue-collar                    11161 non-null  uint8
7   job_entrepreneur                   11161 non-null  uint8
8   job_housemaid                      11161 non-null  uint8
9   job_management                     11161 non-null  uint8
10  job_retired                        11161 non-null  uint8
11  job_self-employed                  11161 non-null  uint8
12  job_services                       11161 non-null  uint8
13  job_student                        11161 non-null  uint8
14  job_technician                     11161 non-null  uint8
15  job_unemployed                     11161 non-null  uint8
16  job_unknown                        11161 non-null  uint8
17  marital_divorced                   11161 non-null  uint8
18  marital_married                    11161 non-null  uint8
19  marital_single                     11161 non-null  uint8
20  education_primary                  11161 non-null  uint8
21  education_secondary                 11161 non-null  uint8
22  education_tertiary                 11161 non-null  uint8
23  education_unknown                  11161 non-null  uint8
24  default_no                         11161 non-null  uint8
25  default_yes                        11161 non-null  uint8
26  housing_no                         11161 non-null  uint8
27  housing_yes                        11161 non-null  uint8
28  loan_no                            11161 non-null  uint8
29  loan_yes                           11161 non-null  uint8
30  contact_cellular                   11161 non-null  uint8
31  contact_telephone                  11161 non-null  uint8
32  contact_unknown                    11161 non-null  uint8
33  month_apr                          11161 non-null  uint8
34  month_aug                          11161 non-null  uint8
35  month_dec                          11161 non-null  uint8
36  month_feb                          11161 non-null  uint8
37  month_jan                          11161 non-null  uint8
38  month_jul                          11161 non-null  uint8
39  month_jun                          11161 non-null  uint8
40  month_mar                          11161 non-null  uint8
41  month_may                          11161 non-null  uint8
42  month_nov                          11161 non-null  uint8
43  month_oct                          11161 non-null  uint8
44  month_sep                          11161 non-null  uint8
45  poutcome_failure                   11161 non-null  uint8
```



```

46  poutcome_other          11161 non-null  uint8
47  poutcome_success        11161 non-null  uint8
48  poutcome_unknown        11161 non-null  uint8
49  pdays2                  11161 non-null  int64
50  depositNew              11161 non-null  int64
dtypes: int64(7), uint8(44)
memory usage: 1.1 MB

```

Data Normalization

StandartScaler, normalizes the data with a standard deviation of 1 with an average of 0.

The target column is not normalized.

```

In [28]: from sklearn.preprocessing import StandardScaler
X = data.iloc[:, 0:50]
Y = data.iloc[:, 50]
nd = StandardScaler()
nd.fit(X)
X = nd.transform(X)
print(X)

```

```

[[ 1.4926218  0.25261499 -0.55420079 ... -0.32579855  0.58352347
 -0.58379938]
 [ 1.24065834 -0.4598839  -0.55420079 ... -0.32579855  0.58352347
 -0.58379938]
 [-0.01915895 -0.08007052 -0.55420079 ... -0.32579855  0.58352347
 -0.58379938]
 ...
 [-0.77504932 -0.46484473 -0.18682923 ... -0.32579855  0.58352347
 -0.58379938]
 [ 0.14881669 -0.47383623 -0.18682923 ... -0.32579855 -1.71372713
  1.71291719]
 [-0.60707368 -0.47383623 -0.55420079 ... -0.32579855  0.58352347
 -0.58379938]]

```

Algorithm Works

```
In [29]: from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import f1_score
X = data.iloc[:, 0:50]
Y = data.iloc[:, 50]
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size=0.3)

accuracies = {}
kappaScores= {}
f1scores={}
```

Logistic Regression

Logistic regression is a predictive linear model that aims to explain the relationship between a dependent binary variable and one or more independent variables. The output of Logistic Regression is a number between *0 and 1* which you can think about as being the probability that a given class is true or not.

```
In [30]: from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(random_state=101,multi_class='ovr',solver='libsvm')
lr.fit(X_train,y_train)
prediction = lr.predict(X_test)
```

```
In [31]: print(classification_report(y_test,prediction))
acc = accuracy_score(y_test,prediction)*100
print("Logistic Regression accuracy:",acc)
accuracies['Logistic Regression']=acc

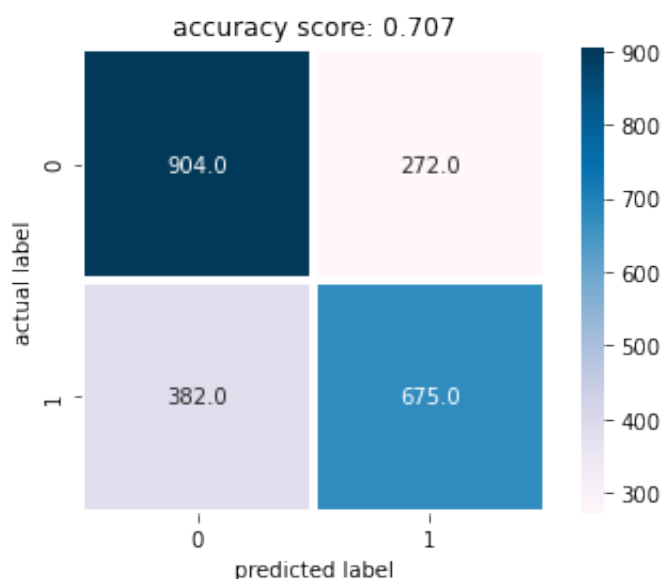
f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['Logistic Regression']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Logistic Regression']=cohen_kappa
```

	precision	recall	f1-score	support
0	0.70	0.77	0.73	1176
1	0.71	0.64	0.67	1057
accuracy			0.71	2233
macro avg	0.71	0.70	0.70	2233
weighted avg	0.71	0.71	0.71	2233

Logistic Regression accuracy: 70.71204657411553
 F1-Score: 67.36526946107784
 Cohen Kappa score: 40.94632616436728

```
In [32]: score=round(accuracy_score(y_test,prediction),3)
cm= confusion_matrix
cm1=cm(y_test,prediction)
sns.heatmap(cm1, annot=True,fmt=".1f",linewidths=3,square=True, cma
plt.ylabel('actual label')
plt.xlabel('predicted label')
plt.title('accuracy score: {0}'.format(score),size=12)
plt.show()
```



Random Forest

```
In [33]: from sklearn.ensemble import RandomForestClassifier
```

```
In [34]: clf = RandomForestClassifier(n_estimators=100, max_depth=12,
                                     random_state=50)

clf.fit(X_train,y_train)

prediction = clf.predict(X_test)
```

```
In [35]: acc = accuracy_score(y_test,prediction)*100
print("Random Forest accuracy:",acc)
accuracies['Random Forest']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['Random Forest']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Random Forest']=cohen_kappa
```

Random Forest accuracy: 72.05553067622034
F1-Score: 66.5236051502146
Cohen Kappa score: 43.27305059532291

Naive Bayes

```
In [36]: from sklearn.naive_bayes import GaussianNB
```

```
In [37]: nb=GaussianNB()
nb.fit(X_train,y_train)
naiveb=nb.predict(X_test)
prediction= nb.predict(X_test)
```

```
In [38]: acc = accuracy_score(y_test,prediction)*100
print("Naive Bayes accuracy:",acc)
accuracies['Naive Bayes']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['Naive Bayes']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Naive Bayes']=cohen_kappa
```

Naive Bayes accuracy: 68.3833407971339
F1-Score: 62.08378088077337
Cohen Kappa score: 35.812328283001015

Stochastic Gradient Descent Classifier

```
In [39]: from sklearn.linear_model import SGDClassifier
```

```
In [40]: sgd=SGDClassifier(loss='modified_huber',shuffle=True,random_state=1
                        ,max_iter=100,eta=0.2,learning_rate='optimal')
sgd.fit(X_train,y_train)
prediction=sgd.predict(X_test)
```

```
In [41]: acc = accuracy_score(y_test,prediction)*100
print("SGD Classifier accuracy:",acc)
accuracies['SGDC']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['SGDC']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['SGDC']=cohen_kappa
```

SGD Classifier accuracy: 65.42767577250336
F1-Score: 61.361361361361354
Cohen Kappa score: 30.271249787643693

KNN

```
In [42]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [43]: knn= KNeighborsClassifier(n_neighbors = 4,algorithm='ball_tree')
knn.fit(X_train, y_train)
prediction=knn.predict(X_test)
```

```
In [44]: acc = accuracy_score(y_test,prediction)*100
print("Knn accuracy:",acc)
accuracies['KNN']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['KNN']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['KNN']=cohen_kappa
```

Knn accuracy: 61.12852664576802
F1-Score: 49.651972157772626
Cohen Kappa score: 20.55250457646862

Decision Tree

```
In [45]: from sklearn.tree import DecisionTreeClassifier
```

```
In [46]: dtree= DecisionTreeClassifier(criterion='gini',max_depth=10,random_
dtree.fit(X_train, y_train)
prediction=dtree.predict(X_test)
```

```
In [47]: acc = accuracy_score(y_test,prediction)*100
print("Decision Tree accuracy:",acc)
accuracies['Decision Tree']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['Decision Tree']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Decision Tree']=cohen_kappa
```

Decision Tree accuracy: 70.53291536050156
F1-Score: 62.61363636363636
Cohen Kappa score: 39.879244072476475

Neural Network - Perceptron

```
In [48]: from sklearn.linear_model import Perceptron
```

```
In [49]: pr = Perceptron(alpha=0.07,max_iter=100, random_state=100,penalty='l2')
pr.fit(X_train, y_train)
prediction = pr.predict(X_test)
```

```
In [50]: acc = accuracy_score(y_test, prediction)*100
print("Perceptron accuracy:",acc)
accuracies['Perceptron']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['Perceptron']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Perceptron']=cohen_kappa
```

Perceptron accuracy: 61.262875055978505
F1-Score: 47.41641337386018
Cohen Kappa score: 20.520826432474315

Gradient Boosting Classifier

```
In [51]: from sklearn.ensemble import GradientBoostingClassifier
```

```
In [52]: clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
max_depth=2, random_state=0)
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
```

```
In [53]: acc = accuracy_score(y_test, prediction)*100
print("Gradient Boosting Classifier accuracy:",acc)
accuracies['Gradient Boosting']=acc

f1=f1_score(y_test,prediction)*100
print("F1-Score: ",f1)
f1scores['Gradient Boosted']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Gradient Boosting']=cohen_kappa
```

Gradient Boosting Classifier accuracy: 70.98074339453649
F1-Score: 66.14420062695925
Cohen Kappa score: 41.23359639746186

Xgboost Classifier

```
In [54]: from xgboost import XGBClassifier
```

```
In [55]: xgb =XGBClassifier(n_estimators=100, learning_rate=0.08, gamma=0, s
                                colsample_bytree=1, max_depth=7)
xgb.fit(X_train,y_train)
prediction = xgb.predict(X_test)
```

```
In [56]: acc = accuracy_score(y_test, prediction)*100
print("Xgboost Classifier accuracy:",acc)
accuracies['Xgboost Classifier']=acc

f1=f1_score(y_test,prediction)*100
print("F1 Score: ",f1)
f1scores['Xgboost Classifier']=f1

cohen_kappa = cohen_kappa_score(y_test, prediction)*100
print('Cohen Kappa score: ',cohen_kappa)
kappaScores['Xgboost Classifier']=cohen_kappa
```

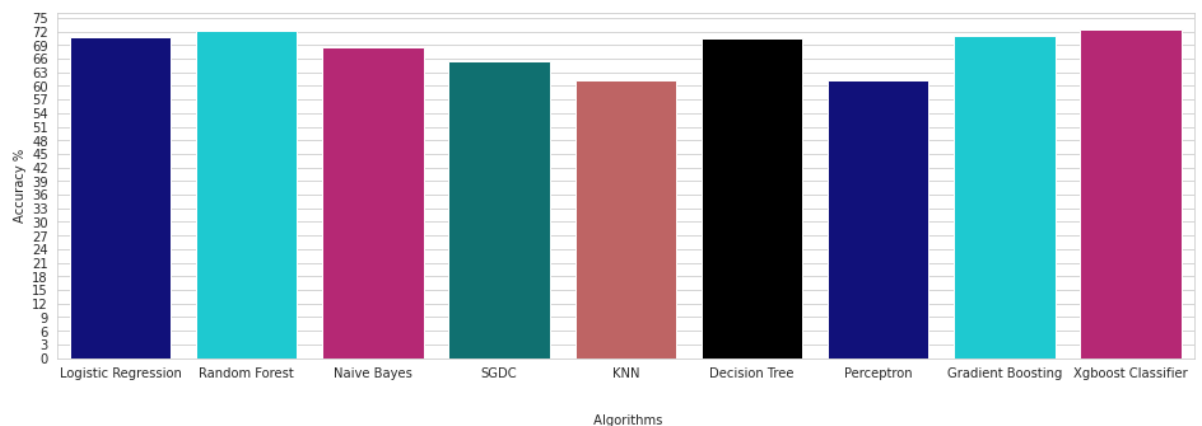
Xgboost Classifier accuracy: 72.50335871025526
F1 Score: 67.5475687103594
Cohen Kappa score: 44.25775091212313

Comparison of accuracies

Accuracy is a metric used to measure the success of a model but is not sufficient by itself.

```
In [57]: colors = ["#00008b", "#00e5ee", "#cd1076", "#008080", "#cd5555", 'black']

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,3))
plt.ylabel("Accuracy %")
plt.xlabel("\n\n Algorithms")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()),
plt.show()
```



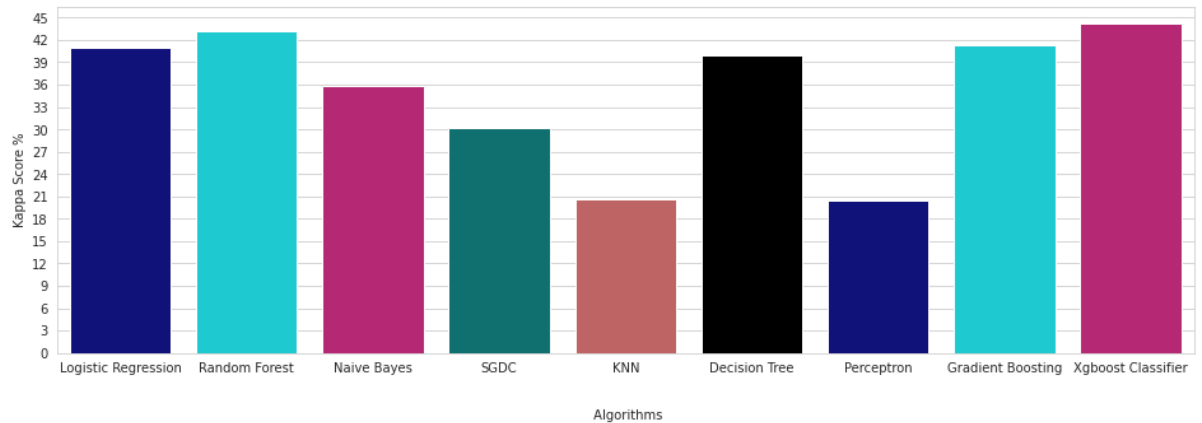
Comparison of Kappa Scores

Cohen's kappa, (κ), symbolized by the lowercase Greek letter, is a powerful statistic useful for testing reliability. Similar to the correlation coefficients, between -1 and +1; where 0 represents the availability that can be expected from random chance, and 1 represents the perfect match between raters.

- 0 indicates no information agreement
- 0.01-0.20 **Slight agreement**
- 0.21-0.40 **Fair agreement**
- 0.41-0.60 **Moderate agreement**
- 0.61-0.80 **Substantial agreement**
- 0.81-1.00 **Almost perfect agreement**

```
In [58]: colors = ["#00008b", "#00e5ee", "#cd1076", "#008080", "#cd5555", 'black']

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,3))
plt.ylabel("Kappa Score %")
plt.xlabel("\n\n Algorithms")
sns.barplot(x=list(kappaScores.keys()), y=list(kappaScores.values()))
plt.show()
```



Comparison of F1 Scores

The F1 Score value shows us the harmonic mean of the Precision and Recall values.

The main reason for using the F1 Score value instead of Accuracy is not to make an incorrect model selection in non-uniform data sets.

```
In [59]: colors = ["#00008b", "#00e5ee", "#cd1076", "#008080", "#cd5555", 'black']

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,3))
plt.ylabel("F1 Score %")
plt.xlabel("\n\n Algorithms")
sns.barplot(x=list(f1scores.keys()), y=list(f1scores.values()), palette=colors)
plt.show()
```

