```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

\+ Code + Text

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

```
data = '/content/drive/MyDrive/Data/weather.csv'
df = pd.read_csv(data)
```

```
df.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	W
0	11/1/2007	Canberra	8.0	24.3	0.0	3.4	6.3	NW	
1	11/2/2007	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	
2	11/3/2007	Canberra	13.7	23.4	3.6	5.8	3.3	NW	
3	11/4/2007	Canberra	13.3	15.5	39.8	7.2	9.1	NW	
4	11/5/2007	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	

```
df.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSp
count	366.000000	366.000000	366.000000	366.000000	363.000000	364.000000	359.
mean	7.265574	20.550273	1.428415	4.521858	7.909366	39.840659	9.
std	6.025800	6.690516	4.225800	2.669383	3.481517	13.059807	7.
min	-5.300000	7.600000	0.000000	0.200000	0.000000	13.000000	0.
25%	2.300000	15.025000	0.000000	2.200000	5.950000	31.000000	6.
50%	7.450000	19.650000	0.000000	4.200000	8.600000	39.000000	7.
75%	12.500000	25.500000	0.200000	6.400000	10.500000	46.000000	13.
max	20.900000	35.800000	39.800000	13.800000	13.600000	98.000000	41.

```
df.size
```

```
8784
```

```
df.shape
```

```
(366, 24)
```

```
col_names = df.columns

col_names
```

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
      dtype='object')
```

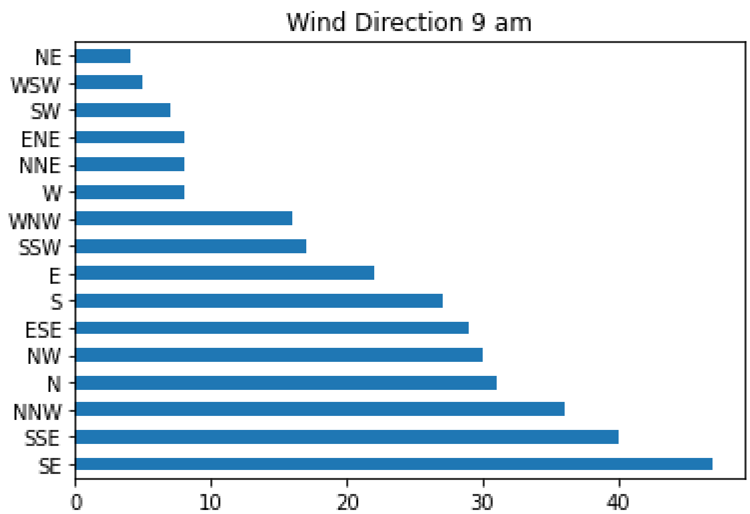
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 24 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           366 non-null    object 
 1   Location       366 non-null    object 
 2   MinTemp        366 non-null    float64
 3   MaxTemp        366 non-null    float64
 4   Rainfall       366 non-null    float64
 5   Evaporation    366 non-null    float64
 6   Sunshine       363 non-null    float64
 7   WindGustDir    363 non-null    object 
 8   WindGustSpeed  364 non-null    float64
 9   WindDir9am     335 non-null    object 
 10  WindDir3pm     365 non-null    object 
 11  WindSpeed9am   359 non-null    float64
 12  WindSpeed3pm   366 non-null    int64  
 13  Humidity9am    366 non-null    int64  
 14  Humidity3pm    366 non-null    int64  
 15  Pressure9am    366 non-null    float64
 16  Pressure3pm    366 non-null    float64
 17  Cloud9am       366 non-null    int64  
 18  Cloud3pm       366 non-null    int64  
 19  Temp9am        366 non-null    float64
 20  Temp3pm        366 non-null    float64
 21  RainToday      366 non-null    object 
 22  RISK_MM        366 non-null    float64
 23  RainTomorrow   366 non-null    object 
dtypes: float64(12), int64(5), object(7)
memory usage: 68.8+ KB
```

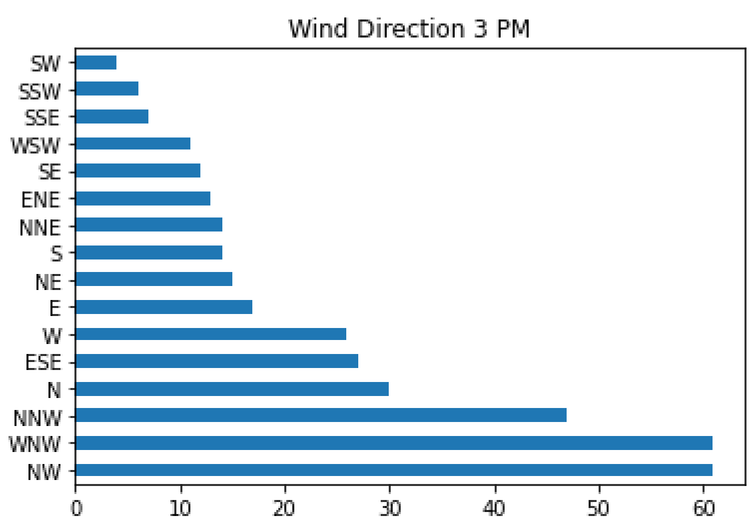
```
categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :', categorical)
```

```
There are 7 categorical variables

The categorical variables are : ['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindD
```

```
df.WindDir9am.value_counts().plot(kind = 'barh')
plt.title("Wind Direction 9 am")
plt.show()
```




```
df.WindDir3pm.value_counts().plot(kind="barh")
plt.title("Wind Direction 3 PM")
plt.show()
```



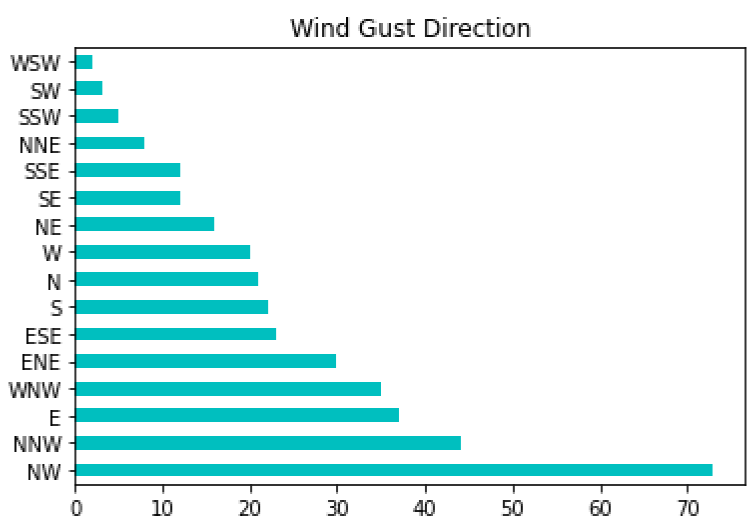

```
df['RainToday']=df['RainToday'].apply(lambda x:1 if x == "Yes" else 0)
```

```
df['RainTomorrow']=df['RainTomorrow'].apply(lambda x:1 if x == "Yes" else 0)
```

```
df.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	W
0	11/1/2007	Canberra	8.0	24.3	0.0	3.4	6.3	NW	
1	11/2/2007	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	
2	11/3/2007	Canberra	13.7	23.4	3.6	5.8	3.3	NW	
3	11/4/2007	Canberra	13.3	15.5	39.8	7.2	9.1	NW	
4	11/5/2007	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	

```
df.WindGustDir.value_counts().plot(kind = "barh",color = 'c')
plt.title("Wind Gust Direction")
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
df=df.dropna()
```

```
df.shape
```

```
(328, 24)
```

```
df.WindGustDir = le.fit_transform(df.WindGustDir)
```

```
df.WindGustDir = le.fit_transform(df.WindGustDir)
```

```
df.WindDir3pm = le.fit_transform(df.WindDir3pm)
```

```
df.WindDir9am = le.fit_transform(df.WindDir9am)
```

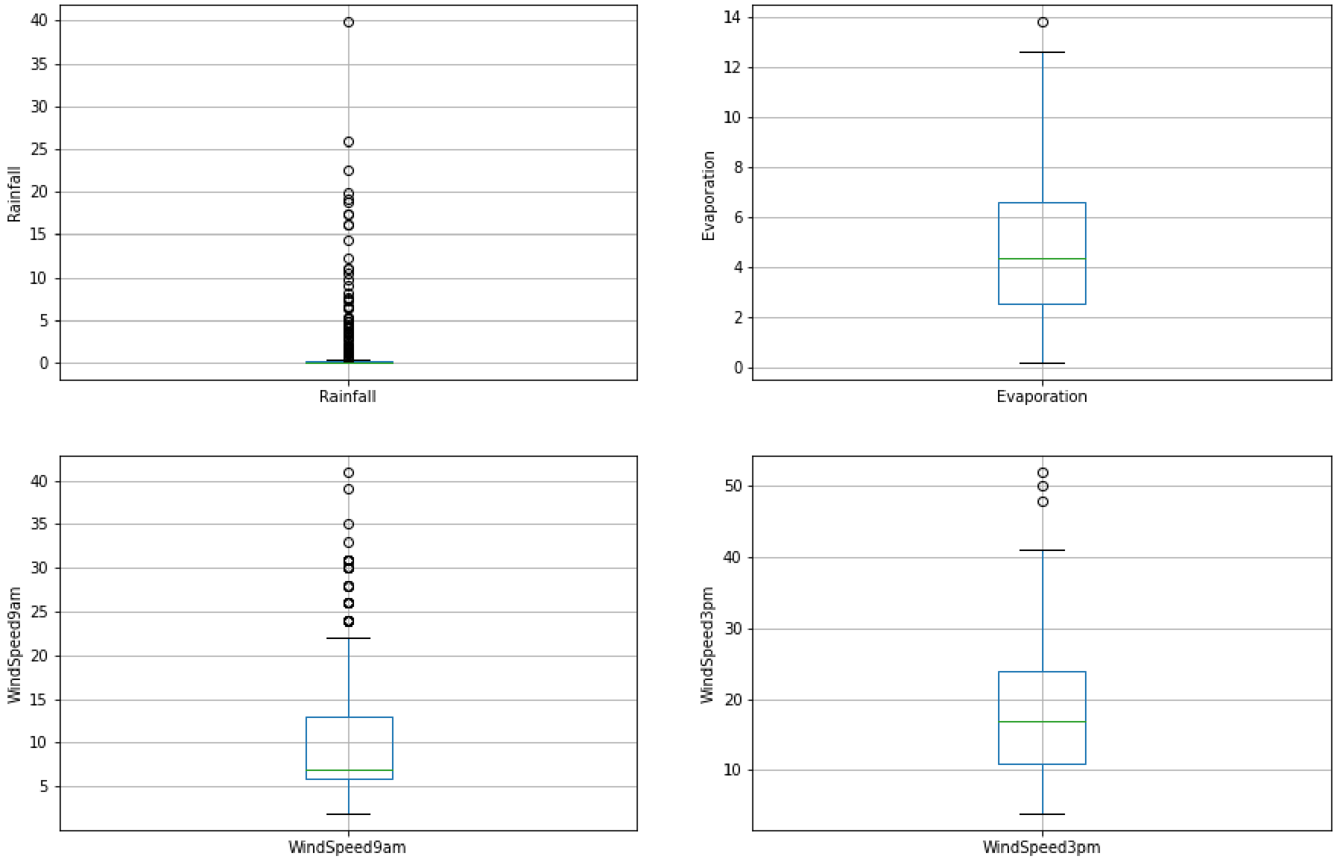
```
df.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGust
count	328.000000	328.000000	328.000000	328.000000	328.000000	328.000000	328.0
mean	7.742988	20.897561	1.440854	4.702439	8.014939	6.192073	40.3
std	5.945199	6.707310	4.289427	2.681183	3.506646	4.337765	13.1
min	-5.300000	7.600000	0.000000	0.200000	0.000000	0.000000	13.0
25%	2.850000	15.500000	0.000000	2.550000	6.000000	2.000000	31.0
50%	7.900000	20.400000	0.000000	4.400000	8.750000	6.500000	39.0
75%	12.800000	25.800000	0.200000	6.600000	10.700000	8.000000	46.0
max	20.900000	35.800000	39.800000	13.800000	13.600000	15.000000	98.0

```
plt.figure(figsize=(15,10))


plt.subplot(2, 2, 1)
fig = df.boxplot(column='Rainfall')
fig.set_title('')
fig.set_ylabel('Rainfall')


plt.subplot(2, 2, 2)
fig = df.boxplot(column='Evaporation')
fig.set_title('')
fig.set_ylabel('Evaporation')


plt.subplot(2, 2, 3)
fig = df.boxplot(column='WindSpeed9am')
fig.set_title('')
fig.set_ylabel('WindSpeed9am')


plt.subplot(2, 2, 4)
fig = df.boxplot(column='WindSpeed3pm')
fig.set_title('')
fig.set_ylabel('WindSpeed3pm')
```

```
fig.set_ylabel('WindSpeed3pm')
```

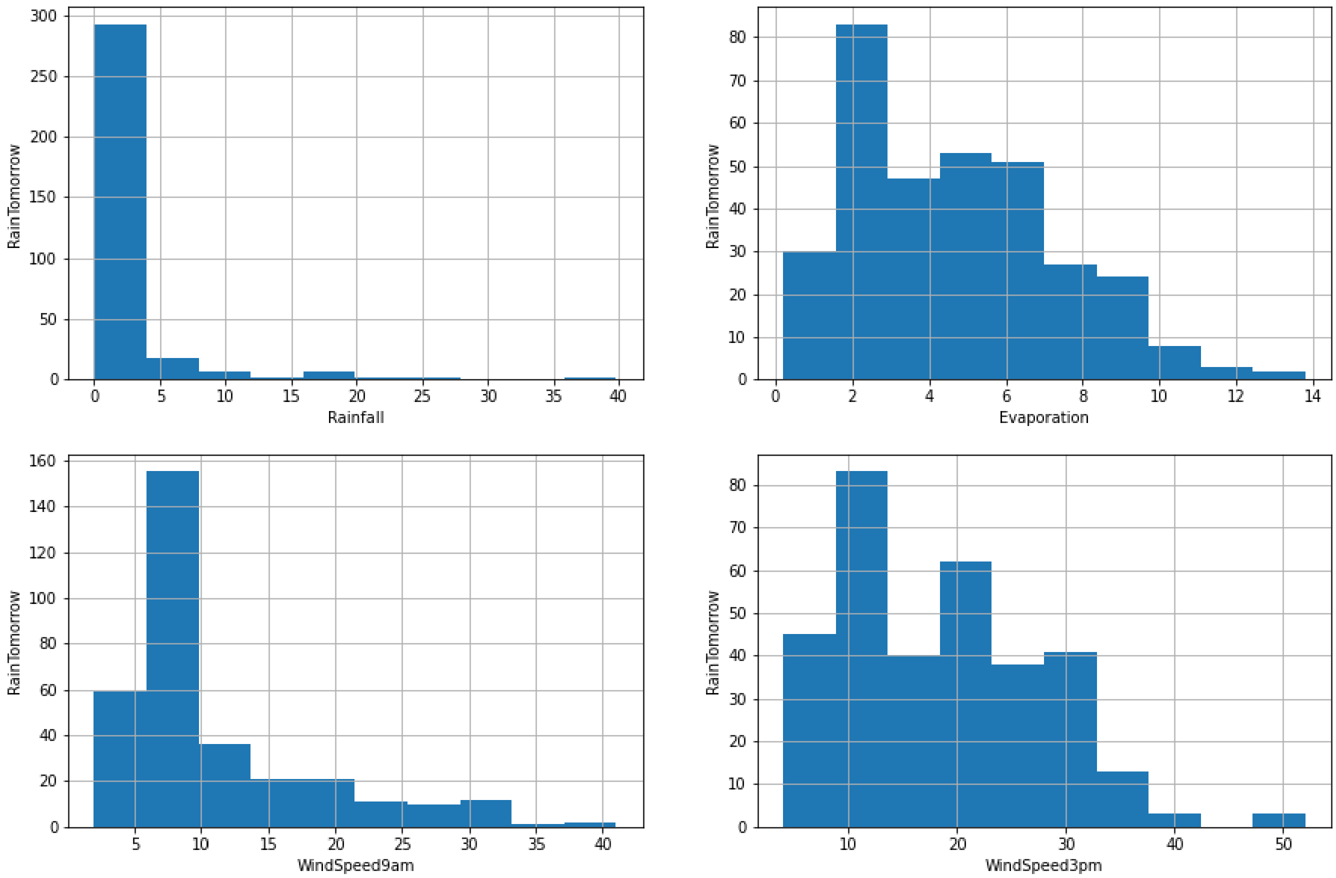
Text(0, 0.5, 'WindSpeed3pm')




```
plt.figure(figsize=(15,10))


plt.subplot(2, 2, 1)
fig = df.Rainfall.hist(bins=10)
fig.set_xlabel('Rainfall')
fig.set_ylabel('RainTomorrow')


plt.subplot(2, 2, 2)
fig = df.Evaporation.hist(bins=10)
fig.set_xlabel('Evaporation')
fig.set_ylabel('RainTomorrow')


plt.subplot(2, 2, 3)
fig = df.WindSpeed9am.hist(bins=10)
```

```
fig.set_xlabel('WindSpeed9am')
fig.set_ylabel('RainTomorrow')


plt.subplot(2, 2, 4)
fig = df.WindSpeed3pm.hist(bins=10)
fig.set_xlabel('WindSpeed3pm')
fig.set_ylabel('RainTomorrow')
```

```
Text(0, 0.5, 'RainTomorrow')
```




```
IQR = df.Rainfall.quantile(0.75) - df.Rainfall.quantile(0.25)
Lower_fence = df.Rainfall.quantile(0.25) - (IQR * 3)
Upper_fence = df.Rainfall.quantile(0.75) + (IQR * 3)
print('Rainfall outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerbound
```

```
Rainfall outliers are values < -0.6000000000000001 or > 0.8
```

```
IQR = df.Evaporation.quantile(0.75) - df.Evaporation.quantile(0.25)
```

```
Lower_fence = df.Evaporation.quantile(0.25) - (IQR * 3)
Upper_fence = df.Evaporation.quantile(0.75) + (IQR * 3)
print('Evaporation outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerbo
```

```
Evaporation outliers are values < -9.599999999999996 or > 18.749999999999996
```

```
IQR = df.WindSpeed9am.quantile(0.75) - df.WindSpeed9am.quantile(0.25)
Lower_fence = df.WindSpeed9am.quantile(0.25) - (IQR * 3)
Upper_fence = df.WindSpeed9am.quantile(0.75) + (IQR * 3)
print('WindSpeed9am outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerb
```

```
WindSpeed9am outliers are values < -15.0 or > 34.0
```

```
IQR = df.WindSpeed3pm.quantile(0.75) - df.WindSpeed3pm.quantile(0.25)
Lower_fence = df.WindSpeed3pm.quantile(0.25) - (IQR * 3)
Upper_fence = df.WindSpeed3pm.quantile(0.75) + (IQR * 3)
print('WindSpeed3pm outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerb
```

```
WindSpeed3pm outliers are values < -28.0 or > 63.0
```

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

```
x = df.drop(['Date','Location','RainTomorrow'],axis=1)
```

```
y = df['RainTomorrow']
```

```
train_x , train_y ,test_x , test_y = train_test_split(x,y , test_size = 0.2,random_state = 2)
```

```
train_x.shape
```

```
(262, 21)
```

```
train_y.shape
```

```
(66, 21)
```

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

```
model = LogisticRegression()
```

```
model.fit(train_x , test_x)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Convergenc
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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
predict = model.predict(train_y)
```

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

```
accuracy_score(predict , test_y)
```

```
0.9696969696969697
```

```
from sklearn.svm import SVC
```

```
model1 = SVC()
```

```
model1.fit(train_x , test_x)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
predict1 = model1.predict(train_y)
```

```
accuracy_score(predict1 , test_y)
```

```
0.8333333333333334
```

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

```
clf = KNeighborsClassifier()
```

```
clf.fit(train_x , test_x)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
predict3= clf.predict(train_y)
```

```
accuracy_score(predict3 , test_y)
```

```
0.8939393939393939
```

```
clf1 = KNeighborsClassifier(n_neighbors=4)
```

```
clf1.fit(train_x , test_x)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

```
predict4 = clf.predict(train_y)
```

```
accuracy_score(predict4 , test_y)
```

```
0.8939393939393939
```

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

```
rf = RandomForestClassifier(max_depth=4)
```

```
rf.fit(train_x , test_x)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=4, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
predict5=rf.predict(train_y)
```

```
accuracy_score(predict5,test_y)
```

```
1.0
```

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

```
model = DecisionTreeClassifier(criterion = "entropy",splitter = "best")
```

```
model.fit(train_x , test_x)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
predict6 = model.predict(train_y)
```

```
accuracy_score(predict6,test_y)
```

```
1.0
```

```
from sklearn.naive_bayes import GaussianNB
```

```
model = GaussianNB()
```

```
model.fit(train_x , test_x)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
predict7 = model.predict(train_y)
```

```
accuracy_score(predict7,test_y)
```

```
0.9848484848484849
```

```
from sklearn.neural_network import MLPClassifier
```

```
model = MLPClassifier(hidden_layer_sizes=(10,),
                        max_iter = 5000,
                        activation = 'logistic',
                        solver = 'sgd',
                        learning_rate_init = 0.001
                        )
```

```
model.fit(train_x , test_x)
```

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(10,), learning_rate='constant',
```

```
                    learning_rate_init=0.001, max_fun=15000, max_iter=5000,
                    momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
                    power_t=0.5, random_state=None, shuffle=True, solver='sgd',
                    tol=0.0001, validation_fraction=0.1, verbose=False,
                    warm_start=False)
```

```
predict8 = model.predict(train_y)
```

```
accuracy_score(predict8,test_y)
```

```
0.8333333333333334
```

```
from sklearn.cluster import KMeans
```

```
model = KMeans(n_clusters = 3, init = "random", algorithm = "full")
```

```
model.fit(train_x , test_x)
```

```
KMeans(algorithm='full', copy_x=True, init='random', max_iter=300, n_clusters=3,
       n_init=10, n_jobs=None, precompute_distances='auto', random_state=None,
       tol=0.0001, verbose=0)
```

```
predict9 = model.predict(train_y)
```

```
accuracy_score(predict9,test_y)
```

```
0.15151515151515152
```

```
from sklearn.linear_model import LinearRegression
```

```
model=LinearRegression()
```

```
model.fit(train_x , test_x)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
predict10 = model.predict(train_y)
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt
print(sqrt(mean_squared_error(predict10,test_y)))
```

```
0.25684667532443894
```

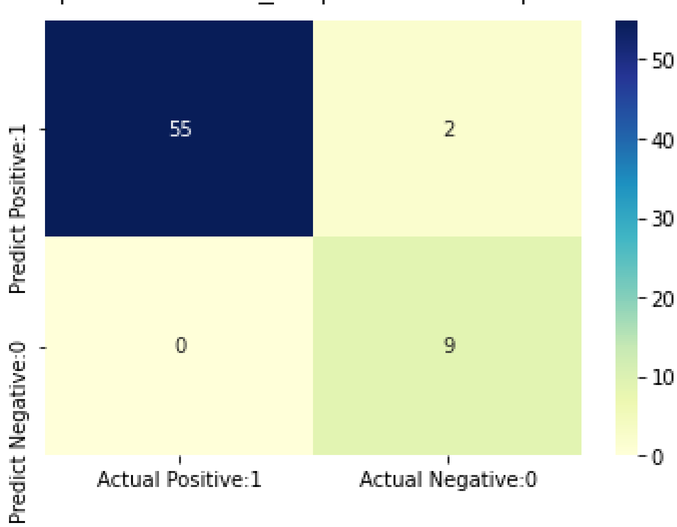
```
from sklearn.metrics import confusion_matrix

cm =confusion_matrix(predict,test_y)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[55  2]
 [ 0  9]]

True Positives(TP) =  55

True Negatives(TN) =  9

False Positives(FP) =  2

False Negatives(FN) =  0
```

```
import seaborn as sns
```

```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a69177350>
```



```
from sklearn.metrics import classification_report

print(classification_report(predict,test_y))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        57
           1       0.82      1.00      0.90         9

    accuracy                           0.97        66
   macro avg       0.91      0.98      0.94        66
weighted avg       0.98      0.97      0.97        66
```

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

```
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.9697
```

```
classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0303
```

```
precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9649
```

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 1.0000
```

```
true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 1.0000
```

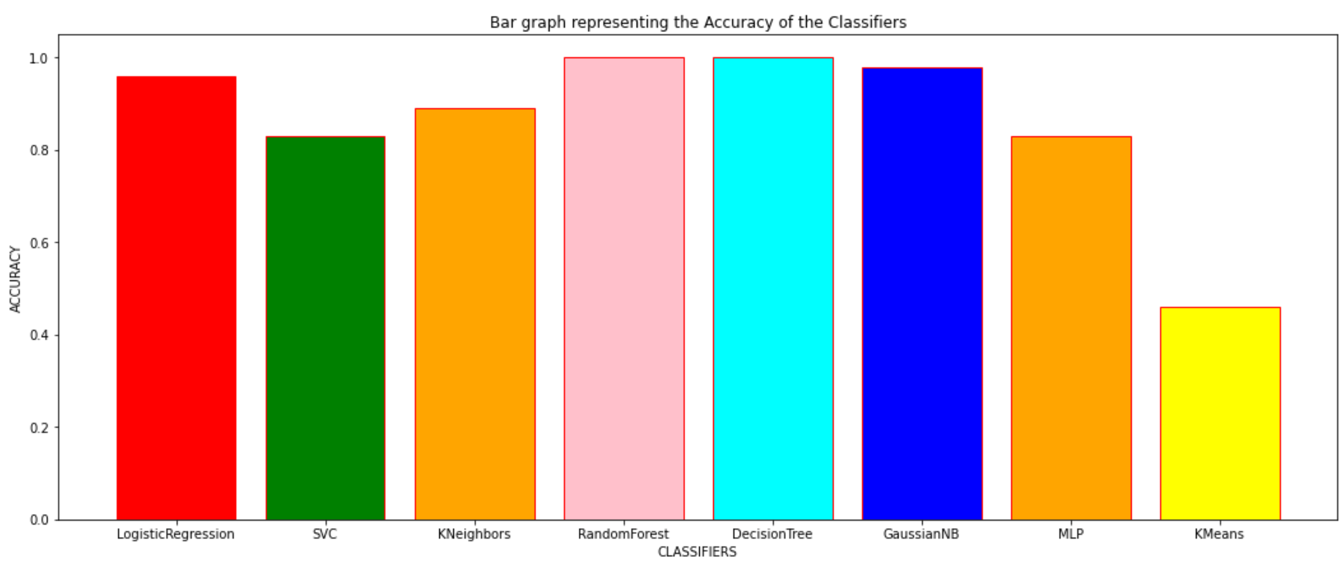
```
false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.1818
```

```
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.8182
```

```
import matplotlib.pyplot as plt

plt.figure(figsize = (18,7))
classifiers = ['LogisticRegression', 'SVC', 'KNeighbors', 'RandomForest', 'DecisionTree', 'Ga
             'MLP', 'KMeans']

accuracy = [0.96, 0.83, 0.89, 1.0, 1.0, 0.98, 0.83, 0.46]

c = ["red", "green", "orange", "pink", "cyan", "blue", "orange", "yellow"]

plt.bar(classifiers, accuracy, width= 0.8, align='center',color= c, edgecolor = 'red')

i = 0
j = 2.0

for i in range(len(classifiers)):
    plt.annotate(accuracy[i], (-0.1 + i, accuracy[i] + j))



plt.title("Bar graph representing the Accuracy of the Classifiers")

plt.xlabel('CLASSIFIERS')
plt.ylabel('ACCURACY')

plt.savefig('Accuracy.png')

plt.show()
```


Bar graph representing the Accuracy of the Classifiers
ACCURACY
CLASSIFIERS
1.0
0.8
0.6
0.4
0.2
0.0
LogisticRegression
SVC
KNeighbors
RandomForest
DecisionTree
GaussianNB
MLP
KMeans