

### Question 3:

At this question, the *insurance.csv* dataset is used to find the relationship between the number of features and smoking people by using a Decision Tree, Random Forest, Logistic Regression, and k-Nearest Neighbors classifiers.

#### Loading dataset:

For uploading dataset we used command below:

```
data = pandas.read_csv("insurance.csv")
```

, that shows this dataset has the shape of (1338, 7), To explore the type of dataset's elements we used the command below:

```
data.head(5)
```

that shows the first five rows as shown in Figure 1:

age	sex	bmi	children	smoker	region	charges
19	female	27.900	0	yes	southwest	16884.92400
18	male	33.770	1	no	southeast	1725.55230
28	male	33.000	3	no	southeast	4449.46200
33	male	22.705	0	no	northwest	21984.47061
32	male	28.880	0	no	northwest	3866.85520

Figure 1: first five elements from the dataset.

#### Data Processing:

The purpose of data processing is to clean the data from unrelated data that can affect the classification negatively. The first process is to find if the data has NaN value elements, we use *isnull().sum()* to find the sum of NaN values elements as shown in the command below:

```
data.isnull().sum()
```

The command output there is not any NaN value element in every dataset's columns as shown in Figure 2:

```
age      0
sex      0
bmi      0
children 0
smoker   0
region   0
charges  0
dtype: int64
```

Figure 2: Summing of NaN value Element.

By using the command:

`data.columns` the dataset index is repectively :

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

To re-arrange the dataset indexes we used this command:

```
data=data[['age','bmi','children','sex','region','charges','smoker']]
```

that mved the *smoker* column to the end of the dataset matrix.

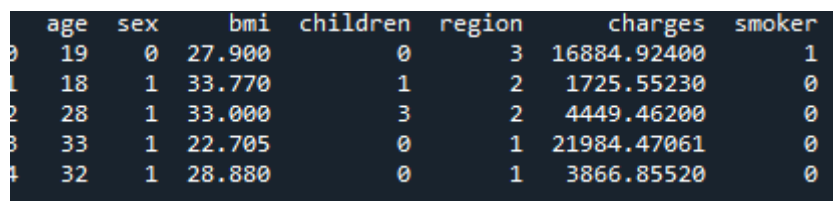
From Figure 1 we can note that, the columns *sex*, *region*, and *smoker* are not a numerical element, that we have to re-present them as numbers, therefore, we used *LabelEncoder()* to achieve this, as shown in commands below:

```
categ = [ 'sex','region','smoker']
```

```
le = LabelEncoder()
```

```
data[categ] = data[categ].apply(le.fit_transform)
```

the new encoded dataset as shown in Figure 3:



	age	sex	bmi	children	region	charges	smoker
0	19	0	27.900	0	3	16884.92400	1
1	18	1	33.770	1	2	1725.55230	0
2	28	1	33.000	3	2	4449.46200	0
3	33	1	22.705	0	1	21984.47061	0
4	32	1	28.880	0	1	3866.85520	0

Figure 3: first five rows of the new encoded dataset.

### Data splitting:

The insurance dataset has 7 columns, that are indexed re-arranged, and encoded to numbers.

For splitting the features columns and training label we used the command below:

```
X, y = data[:, :6], data[:, -1]
```

, which takes the first 6 columns as features and the last column as a label.

The Label has two classes: '0' class for not smoking and person '1' for smoking person.

Before training we have to splite the dataset to train and test dataset by using the command below:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=5)
```

The train dataset ratio is 70% from the original dataset with (936, 6) and train label (936, 1), the test dataset ratio is 30% with (402, 6) and test label (402, 1).

For the four methods the same dataset will be used and each method's results will be taken individually.

## 1) Decision Tree:

For this classifier we used *DecisionTreeClassifier* classifier with '0' random state and '4' max depth as shown in command below:

```
classifier_DecisionTree = DecisionTreeClassifier(**{'random_state': 0, 'max_depth': 4})
```

To train the classifier we used the *fit* command as shown below:

```
classifier_DecisionTree.fit(X_train, y_train)
```

For showing the classifier performance we predicted the test dataset, after that we will compare the results with real test dataset labels.

```
y_test_pred = classifier_DecisionTree.predict(X_test)
```

To show the Precision, Recall ,F1-Score And Accuracy values we used the command: *classification\_report(y\_test, y\_test\_pred, target\_names=class\_names)* Where *y\_test* is the true test dataset labels and *y\_test\_pred* predicted labels through the classifier.

classifier_DecisionTree performance on test dataset				
	precision	recall	f1-score	support
Class-0	0.98	0.97	0.98	318
Class-1	0.91	0.94	0.92	84
accuracy			0.97	402
macro avg	0.95	0.96	0.95	402
weighted avg	0.97	0.97	0.97	402

Figure 4: Precision, Recall, F1-Score And Accuracy For Decision Tree Classifier

The confusion matrix is shown in figure 5:

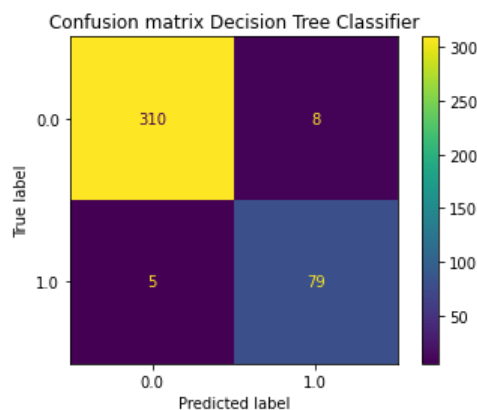


Figure 5: confusion matrix for Decision Tree Classifier.

## 2) Random Forest:

To classify the labels according to the input features first we have to define the classifier which in this state situation is RandomForestClassifier classifier as shown in the command below:

```
classifier_RandomForest = RandomForestClassifier(n_jobs=2, random_state=0)
```

For training the input feature dataset ( $X_{train}$ ) with its labels ( $y_{train}$ ) we used the fit function as shown in the command below:

```
classifier_RandomForest.fit(X_train, y_train)
```

Predicting the labels by the classifier through the test dataset is shown below in command:

```
y_test_pred = classifier_RandomForest.predict(X_test)
```

To show the Precision, Recall ,F1-Score And Accuracy values we used the command: `classification_report(y_test, y_test_pred, target_names=class_names)` Where  $y_{test}$  is the true test dataset labels and  $y_{test\_pred}$  predicted labels through the classifier.

classifier_RandomForest performance on test dataset				
	precision	recall	f1-score	support
Class-0	0.97	0.98	0.98	318
Class-1	0.93	0.89	0.91	84
accuracy			0.96	402
macro avg	0.95	0.94	0.94	402
weighted avg	0.96	0.96	0.96	402

Figure 5: Precision, Recall, F1-Score And Accuracy For Random Forest Classifier

The confusion matrix is shown in figure 6:

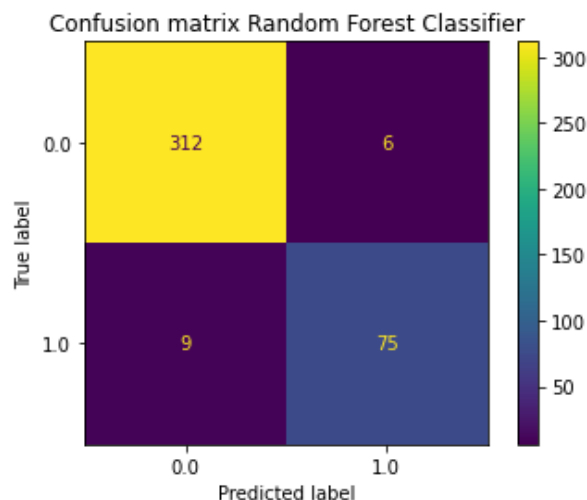


Figure 6: confusion matrix for Random Forest Classifier.

### 3) Logistic Regression:

For logistic regression the linear model classifier LogisticRegression will be used as shown in command below:

```
classifier_logisticREG = linear_model.LogisticRegression(solver='liblinear', C=1)
```

For training classifier by train dataset and its labels, command below is used:

```
classifier_logisticREG.fit(X_train, y_train)
```

After training data, we test our classifier by test dataset by command below:

```
y_test_pred = classifier_logisticREG.predict(X_test)
```

To show the Precision, Recall ,F1-Score And Accuracy values we used the command: `classification_report(y_test, y_test_pred, target_names=class_names)` Where `y_test` is the true test dataset labels and `y_test_pred` predicted labels through the classifier.

classifier_logisticREG performance on test dataset				
	precision	recall	f1-score	support
Class-0	0.95	0.98	0.96	318
Class-1	0.93	0.79	0.85	84
accuracy			0.94	402
macro avg	0.94	0.88	0.91	402
weighted avg	0.94	0.94	0.94	402

Figure 7: Precision, Recall, F1-Score And Accuracy For Logistic Regression Classifier

The confusion matrix as shown in figure 8:

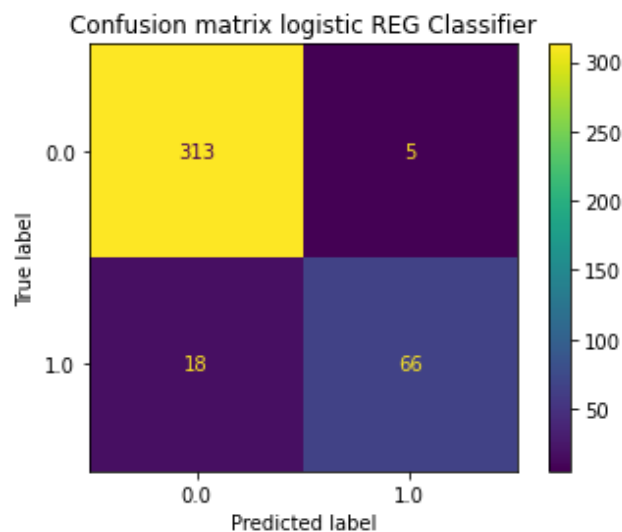


Figure 8: confusion matrix for Logistic Regression Classifier.

#### 4) k-Nearest Neighbors classifier:

In this method *neighbors.KNeighborsClassifier* classifier is used with step size 12 and the distance weight type as shown in the command below:

```
KNeighbors_Classifier = neighbors.KNeighborsClassifier(12, weights='distance')
```

For training we trained the input data with its labels by command shown below:

```
KNeighbors_Classifier.fit(X_train, y_train)
```

After training data, we test our classifier by test dataset by command below:

```
y_test_pred = classifier_KNeighborsClassifier.predict(X_test)
```

for showing the Precision, Recall, F1-Score And Accuracy values we used the command: *classification\_report*(*y\_test*, *y\_test\_pred*, *target\_names*=*class\_names*)  
Where *y\_test* is the true test dataset labels and *y\_test\_pred* predicted labels through the classifier.

Classifier_KNeighborsClassifier performance on test dataset				
	precision	recall	f1-score	support
Class-0	0.94	0.96	0.95	318
Class-1	0.83	0.75	0.79	84
accuracy			0.92	402
macro avg	0.88	0.85	0.87	402
weighted avg	0.91	0.92	0.91	402

Figure 9: Precision, Recall, F1-Score and Accuracy for k-Nearest Neighbors Classifier

The confusion matrix as shown in figure 8:

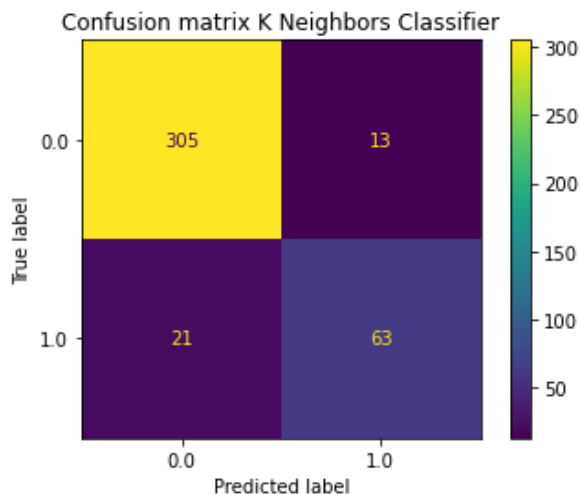


Figure 10: confusion matrix for k-Nearest Neighbors Classifier.