Name: **Sam Louvall**

Course: **CS 5402**

Assignment: **Programming Assignment 4**

Date: **07-13-2021**

Concept Description:

Use K-Nearest neighbor to classify and compare attributes to see what gives some one chronic heart disease.

Data Collection:

The data has be provided by the client.

In [1]:
```
#For data managment
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.linear_model import LinearRegression
import seaborn as sns
from matplotlib import pyplot as plt
#For Simple Linear Regression
import statsmodels.api as sm
```

In [2]:
```
#reading in the file
df = pd.read_csv(r'C:\Users\samlo\DataMining\PA4\heart-disease-data.csv')
```

Example Description:

Age:

Ratio attribute that describes the age of the person.

cigsPerDay:

Ratio attribute that describes the amount of cigs per day.

totChol:

Ratio attribute that describes the total cholesterol of the person.

sysBP:

Ratio attribute that describes the systolic blood pressure of the person.

diaBP:

Ratio attribute that describes the diabolic blood pressure of the person.

BMI:

Ratio attribute that describes the body mass index of the person.

heartRate:

Ratio attribute that describes the heart rate of the person.

Glucose:

Ratio attribute that describes the blood glucose of the person.

CHD:

Binary attribute that describes if the person has chronic heart disease or not.

Data Import and Wrangling:

Due to an attribute missing some values, I had to use simple linear regression to find the missing values.

In [4]:
```
#separating the NULL values from the data
test_data = df[df['BMI'].isnull()]
test_data
```

Out[4]:

	age	cigsPerDay	totChol	sysBP	diaBP	BMI	heartRate	glucose	CHD
91	40	0	205	100.0	60.0	NaN	60	72	1
1048	64	3	221	148.0	85.0	NaN	90	80	1
1446	40	0	164	135.0	75.0	NaN	75	85	0
1456	40	20	266	101.0	73.0	NaN	70	64	0
1474	70	0	107	143.0	93.0	NaN	68	62	1
1783	63	20	213	163.0	94.0	NaN	76	69	1
1844	35	0	274	104.0	61.0	NaN	60	68	0
1861	58	1	244	160.5	98.0	NaN	86	69	0
1881	65	0	240	235.0	100.0	NaN	68	297	1
1955	53	0	240	109.0	79.0	NaN	92	80	1
2281	39	0	229	119.0	63.5	NaN	76	83	0
2447	38	20	215	110.0	80.0	NaN	100	73	0
2633	64	20	225	120.0	75.0	NaN	70	94	0
2776	61	0	300	150.5	89.0	NaN	68	72	1

```
#dropping the NULl values from the data frame and considering it the train data
df.dropna(inplace = True)
df
```

5]:

	age	cigsPerDay	totChol	sysBP	diaBP	BMI	heartRate	glucose	CHD
0	39	0	195	106.0	70.0	26.97	80	77	0
1	46	0	250	121.0	81.0	28.73	95	76	0
2	48	20	245	127.5	80.0	25.34	75	70	0
3	61	30	225	150.0	95.0	28.58	65	103	1
4	46	23	285	130.0	84.0	23.10	85	85	0
...	...	...	...	...	...	...	...	...	...
3809	68	0	176	168.0	97.0	23.14	60	79	1
3810	50	1	313	179.0	92.0	25.97	66	86	1
3811	51	43	207	126.5	80.0	19.71	65	68	0
3812	48	20	248	131.0	72.0	22.00	84	86	0
3813	52	0	269	133.5	83.0	21.47	80	107	0

3800 rows × 9 columns

```
#creating "X_train" and a "Y_train" from the data frame
y_train = df["BMI"]
y_train
```

```
0       26.97
1       28.73
2       25.34
3       28.58
4       23.10
        ...
3809    23.14
3810    25.97
3811    19.71
3812    22.00
3813    21.47
Name: BMI, Length: 3800, dtype: float64
```

```
#X_train means "dataset except df['BMI'] features with Non NULL Values"
x_train = df.drop('BMI',axis=1)
```

```
#Building the model
lr = LinearRegression()
#train the model on train data set
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

In [9]:

```
#creating the X_test from the test_data
x_test = test_data.drop("BMI",axis=1)
x_test
```

Out[9]:

	age	cigsPerDay	totChol	sysBP	diaBP	heartRate	glucose	CHD
91	40	0	205	100.0	60.0	60	72	1
1048	64	3	221	148.0	85.0	90	80	1
1446	40	0	164	135.0	75.0	75	85	0
1456	40	20	266	101.0	73.0	70	64	0
1474	70	0	107	143.0	93.0	68	62	1
1783	63	20	213	163.0	94.0	76	69	1
1844	35	0	274	104.0	61.0	60	68	0
1861	58	1	244	160.5	98.0	86	69	0
1881	65	0	240	235.0	100.0	68	297	1
1955	53	0	240	109.0	79.0	92	80	1
2281	39	0	229	119.0	63.5	76	83	0

```
#Applying The trained model on the X_test
y_pred = lr.predict(x_test)
y_pred
```

```
]: array([22.98763624, 26.473629  , 24.73366214, 24.06577123, 26.91661014,
       27.07762383, 23.12512159, 27.83466903, 30.80470004, 25.58441273,
       23.52825446, 24.81620083, 24.80845108, 27.16446998])
```

```
#replace the missing values with predicted values
test_data.loc[test_data.BMI.isnull(),'BMI'] = y_pred
```

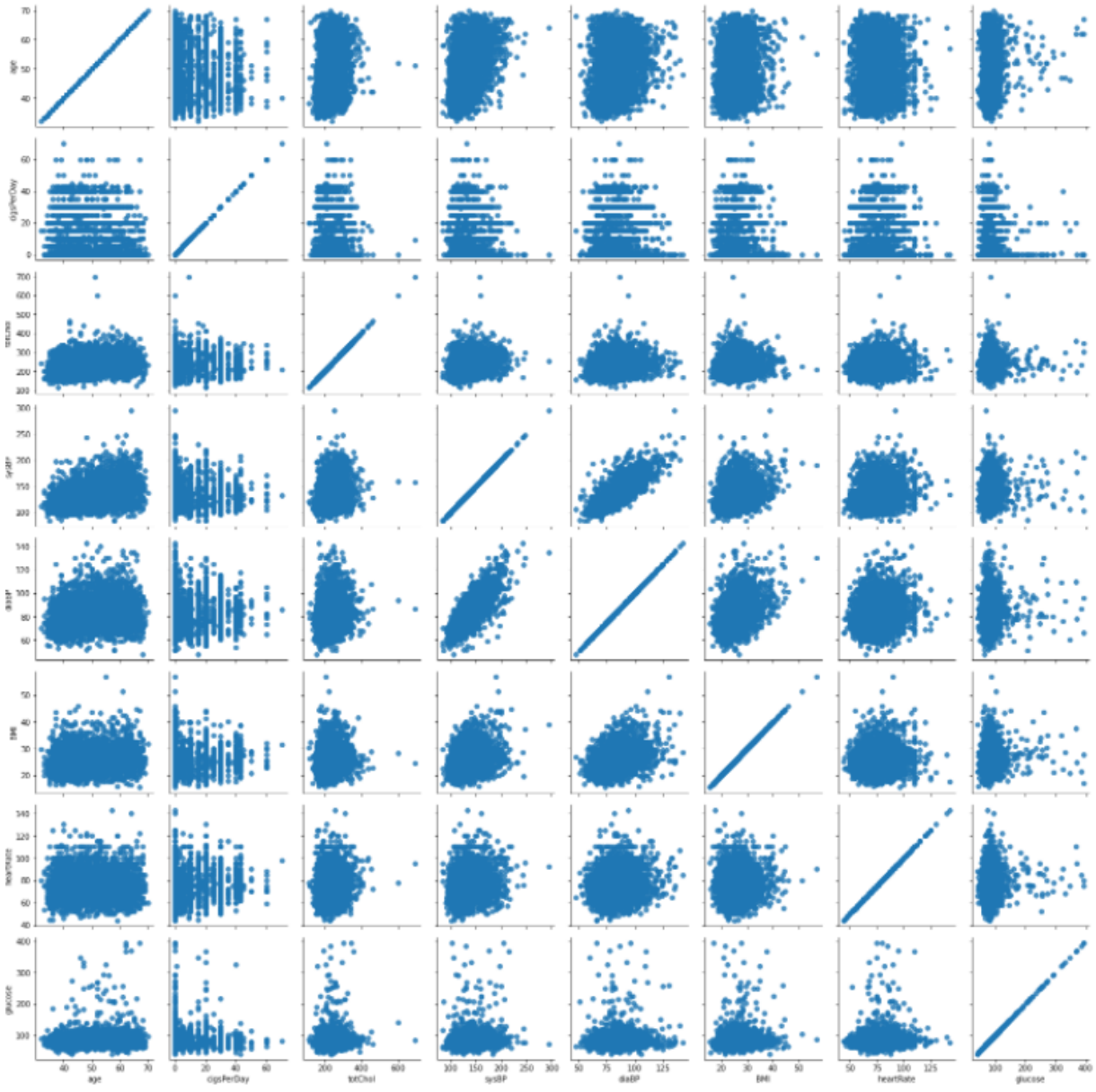
```
C:\Users\samlo\anaconda3\lib\site-packages\pandas\core\indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

Out[13]:

	age	cigsPerDay	totChol	sysBP	diaBP	BMI	heartRate	glucose	CHD
91	40	0	205	100.0	60.0	22.987636	60	72	1
1048	64	3	221	148.0	85.0	26.473629	90	80	1
1446	40	0	164	135.0	75.0	24.733662	75	85	0
1456	40	20	266	101.0	73.0	24.065771	70	64	0
1474	70	0	107	143.0	93.0	26.916610	68	62	1
1783	63	20	213	163.0	94.0	27.077624	76	69	1
1844	35	0	274	104.0	61.0	23.125122	60	68	0
1861	58	1	244	160.5	98.0	27.834669	86	69	0
1881	65	0	240	235.0	100.0	30.804700	68	297	1
1955	53	0	240	109.0	79.0	25.584413	92	80	1
2281	39	0	229	119.0	63.5	23.528254	76	83	0

From the images provided, I separated the missing BMI values from the dataset. Next, I created a new data set without the BMI attribute. After that I created a Y_train set from the BMI values that were available. I also created an X_train data set with all values present without BMI. Then I used the SKlearn module to use Linear Regression to build the model. I then used the model to predict the missing BMI values. I did replace the missing BMI values manualy into a new dataset. I then used KNN on the new dataset with no missing values.

Exploratory Data Analysis:



I wanted to explore the data in an easy visual way. So I thought comparing each attribute to each other was a good way to analyze each attribute.

Mining or Analytics:

Splitting the data into an 80% %20 split.

```
#Partitioning the data into an 80 / 20 split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,train_size = .80)
```

K-Nearest Neighbor model. With three different values of k.

K=3

```
#Creating the k-nearest neighbor

model = KNeighborsClassifier(n_neighbors = 3)
model.fit(X_train,Y_train['CHD'])
y_pred = model.predict(X_test)
```

```
pred = pd.DataFrame(y_pred,)
pred.rename(columns = {0:"pred"},inplace = True)
results = pd.concat([pred,Y_test.reset_index(drop=True)],axis=1)
results
```

	pred	CHD
0	0	0
1	0	0
2	0	1
3	0	0
4	0	1
...	...	...
758	0	0
759	0	0
760	0	0
761	0	0
762	0	0

763 rows × 2 columns

```
print(classification_report(Y_test,y_pred))
```

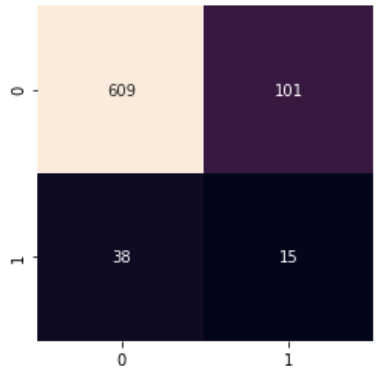
```
              precision    recall  f1-score   support

           0       0.86      0.94      0.90       647
           1       0.28      0.13      0.18       116

    accuracy                           0.82       763
   macro avg       0.57      0.54      0.54       763
weighted avg       0.77      0.82      0.79       763
```

```
print(confusion_matrix(Y_test,y_pred))
```

```
[[609  38]
 [101  15]]
```

```
map = confusion_matrix(Y_test,y_pred)
sns.heatmap(map.T,square=True,annot=True,fmt='d',cbar = False)
```

26]: <AxesSubplot:>




```
from sklearn.metrics import accuracy_score
print("Accuracy of prediction using KNN =",accuracy_score(y_pred,Y_test)*100)
```

```
Accuracy of prediction using KNN = 81.7824377457405
```

K=10

```
#Next is K = 10;
#Creating the k-nearest neighbor

model = KNeighborsClassifier(n_neighbors = 10)
model.fit(X_train,Y_train['CHD'])
y_pred = model.predict(X_test)
```

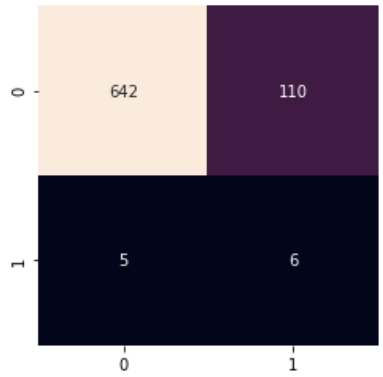
```
pred = pd.DataFrame(y_pred,)
pred.rename(columns = {0:"pred"},inplace = True)
results = pd.concat([pred,Y_test.reset_index(drop=True)],axis=1)
results
```

	pred	CHD
0	0	0
1	0	0
2	0	1
3	0	0
4	0	1
...	...	...
758	0	0
759	0	0
760	0	0
761	0	0
762	0	0

763 rows × 2 columns

```
print(classification_report(Y_test,y_pred))
print(confusion_matrix(Y_test,y_pred))
map = confusion_matrix(Y_test,y_pred)
sns.heatmap(map.T,square=True,annot=True,fmt='d',cbar = False)
from sklearn.metrics import accuracy_score
print("Accuracy of prediction using KNN =",accuracy_score(y_pred,Y_test)*100)
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92       647
           1       0.55      0.05      0.09       116

    accuracy                           0.85       763
   macro avg       0.70      0.52      0.51       763
weighted avg       0.81      0.85      0.79       763

[[642   5]
 [110   6]]
Accuracy of prediction using KNN = 84.92791612057667
```



K=50

```
#Next will be k = 50;

model = KNeighborsClassifier(n_neighbors = 50)
model.fit(X_train,Y_train['CHD'])
y_pred = model.predict(X_test)
```

```
pred = pd.DataFrame(y_pred,)
pred.rename(columns = {0:"pred"},inplace = True)
results = pd.concat([pred,Y_test.reset_index(drop=True)],axis=1)
results
```

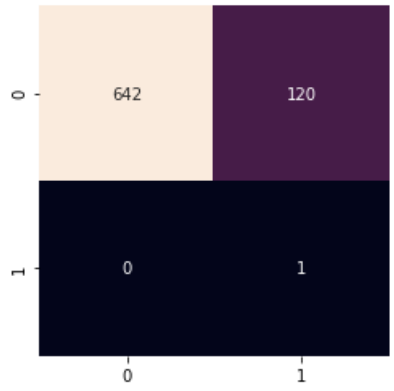
:

	pred	CHD
0	0	1
1	0	0
2	0	1
3	0	1
4	0	0
...	...	...
758	0	0
759	0	0
760	0	0
761	0	0
762	0	0

763 rows × 2 columns

```
print(classification_report(Y_test,y_pred))
print(confusion_matrix(Y_test,y_pred))
map = confusion_matrix(Y_test,y_pred)

sns.heatmap(map.T,square=True,annot=True,fmt='d',cbar = False)
from sklearn.metrics import accuracy_score
print("Accuracy of prediction using KNN =",accuracy_score(y_pred,Y_test)*100)
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91       642
           1       1.00      0.01      0.02       121

    accuracy                           0.84       763
   macro avg       0.92      0.50      0.47       763
weighted avg       0.87      0.84      0.77       763

[[642   0]
 [120   1]]
Accuracy of prediction using KNN = 84.27260812581913
```



Evaluation:

```
print(classification_report(Y_test,y_pred))
```

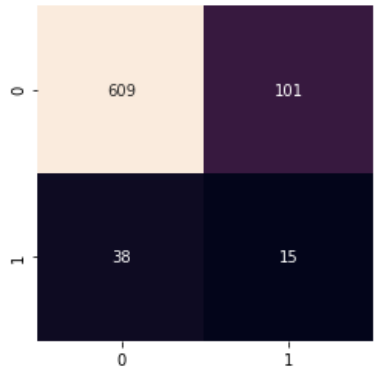
```
              precision    recall  f1-score   support

           0       0.86      0.94      0.90       647
           1       0.28      0.13      0.18       116

    accuracy                           0.82       763
   macro avg       0.57      0.54      0.54       763
weighted avg       0.77      0.82      0.79       763
```

```
print(confusion_matrix(Y_test,y_pred))
```

```
[[609  38]
 [101  15]]
```

```
map = confusion_matrix(Y_test,y_pred)
sns.heatmap(map.T,square=True,annot=True,fmt='d',cbar = False)
```

26]: <AxesSubplot:>

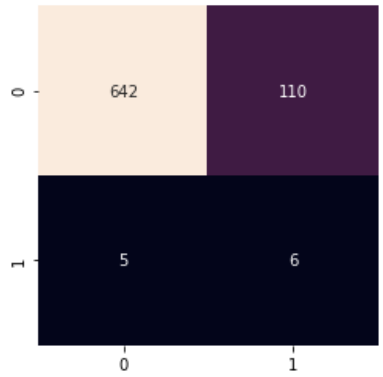



```
from sklearn.metrics import accuracy_score
print("Accuracy of prediction using KNN =",accuracy_score(y_pred,Y_test)*100)
```

```
Accuracy of prediction using KNN = 81.7824377457405
```

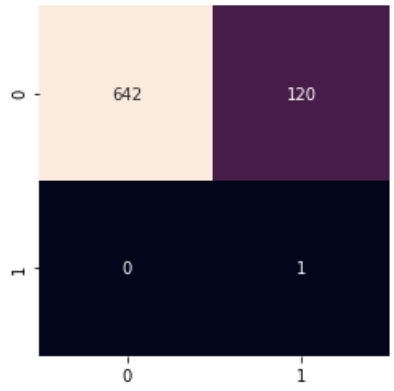
```
print(classification_report(Y_test,y_pred))
print(confusion_matrix(Y_test,y_pred))
map = confusion_matrix(Y_test,y_pred)
sns.heatmap(map.T,square=True,annot=True,fmt='d',cbar = False)
from sklearn.metrics import accuracy_score
print("Accuracy of prediction using KNN =",accuracy_score(y_pred,Y_test)*100)
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92       647
           1       0.55      0.05      0.09       116

    accuracy                           0.85       763
   macro avg       0.70      0.52      0.51       763
weighted avg       0.81      0.85      0.79       763

[[642   5]
 [110   6]]
Accuracy of prediction using KNN = 84.92791612057667
```



```
print(classification_report(Y_test,y_pred))
print(confusion_matrix(Y_test,y_pred))
map = confusion_matrix(Y_test,y_pred)

sns.heatmap(map.T,square=True,annot=True,fmt='d',cbar = False)
from sklearn.metrics import accuracy_score
print("Accuracy of prediction using KNN =",accuracy_score(y_pred,Y_test)*100)
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91       642
           1       1.00      0.01      0.02       121

    accuracy                           0.84       763
   macro avg       0.92      0.50      0.47       763
weighted avg       0.87      0.84      0.77       763

[[642   0]
 [120   1]]
Accuracy of prediction using KNN = 84.27260812581913
```



After using a confusion matrix and looking at the accuracy of each value of k, it appears that using k = 10 was the most accurate. With an accuracy of 84% and having the most true positives and true negatives. K = 50 did have the most precision in guessing if they did have chronic heart disease.

Results:

After looking at the results of the confusion matrices and accuracy scores. I would say the best model for finding if someone has chronic heart disease is k = 10.

Reference:

Youtube teaching about KNN

https://www.youtube.com/watch?v=_ukYsNbZy8Q

seaborn : python visualization

http://seaborn.pydata.org/introduction.html

Scikitlearn visualization

https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html

More seaborn visualization

https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html