Healthcare

**Objective:**

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

Build a model to accurately predict whether the patients in the dataset have diabetes or not.

**Data Exploration:**

1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose

• Blood Pressure

• Skin Thickness

• Insulin

• BMI

2. Visually explore these variables using histograms. Treat the missing values accordingly.

3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

**Dataset names:**

health care diabetes.csv

**Source code:**

In [250]:

*# Import Libraries*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**import** **seaborn** **as** **sns**

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib** **import** style

%**matplotlib** inline

In [251]:

*# Importing Datasets*

hcare\_data = pd.read\_csv("health care diabetes.csv")

In [252]:

hcare\_data.shape

Out[252]:

(768, 9)

In [253]:

hcare\_data.columns

Out[253]:

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],

dtype='object')

In [254]:

hcare\_data.info()

Output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 SkinThickness 768 non-null int64

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

In [255]:

hcare\_data.head(10)

Out[255]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| **5** | 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |
| **6** | 3 | 78 | 50 | 32 | 88 | 31.0 | 0.248 | 26 | 1 |
| **7** | 10 | 115 | 0 | 0 | 0 | 35.3 | 0.134 | 29 | 0 |
| **8** | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| **9** | 8 | 125 | 96 | 0 | 0 | 0.0 | 0.232 | 54 | 1 |

In [256]:

hcare\_data.Insulin.mean()

Out[256]:

79.79947916666667

In [257]:

hcare\_data.BloodPressure.mean()

Out[257]:

69.10546875

In [258]:

hcare\_data.describe()

Out[258]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

In [259]:

zero\_list = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']

zero\_data = (hcare\_data[zero\_list] == 0).sum()

zero\_data

Out[259]:

Glucose 5

BloodPressure 35

SkinThickness 227

Insulin 374

BMI 11

dtype: int64

In [260]:

*# Replacing the zero values with Nan*

hcare\_data[zero\_list] = hcare\_data[zero\_list].replace(0, np.nan)

hcare\_data.isnull().sum()

*# Replacing the Nan values with mean values*

hcare\_data.fillna(hcare\_data.mean(), inplace = **True**)

hcare\_data.isnull().sum()

hcare\_data.describe()

Out[260]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 3.845052 | 121.686763 | 72.405184 | 29.153420 | 155.548223 | 32.457464 | 0.471876 | 33.240885 | 0.348958 |
| **std** | 3.369578 | 30.435949 | 12.096346 | 8.790942 | 85.021108 | 6.875151 | 0.331329 | 11.760232 | 0.476951 |
| **min** | 0.000000 | 44.000000 | 24.000000 | 7.000000 | 14.000000 | 18.200000 | 0.078000 | 21.000000 | 0.000000 |
| **25%** | 1.000000 | 99.750000 | 64.000000 | 25.000000 | 121.500000 | 27.500000 | 0.243750 | 24.000000 | 0.000000 |
| **50%** | 3.000000 | 117.000000 | 72.202592 | 29.153420 | 155.548223 | 32.400000 | 0.372500 | 29.000000 | 0.000000 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 155.548223 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

In [261]:

*#hcare\_data.info()*

*#hcare\_data['Glucose','BloodPressure','SkinThickness','Insulin'] = hcare\_data['Glucose','BloodPressure','SkinThickness','Insulin'].astype('int64')*

hcare\_data['Glucose'] = hcare\_data['Glucose'].astype('int64')

hcare\_data['BloodPressure'] = hcare\_data['BloodPressure'].astype('int64')

hcare\_data['SkinThickness'] = hcare\_data['SkinThickness'].astype('int64')

hcare\_data['Insulin'] = hcare\_data['Insulin'].astype('int64')

hcare\_data.info()

hcare\_data.head(10)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 768 entries, 0 to 767

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Pregnancies 768 non-null int64

1 Glucose 768 non-null int64

2 BloodPressure 768 non-null int64

3 SkinThickness 768 non-null int64

4 Insulin 768 non-null int64

5 BMI 768 non-null float64

6 DiabetesPedigreeFunction 768 non-null float64

7 Age 768 non-null int64

8 Outcome 768 non-null int64

dtypes: float64(2), int64(7)

memory usage: 54.1 KB

Out[261]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148 | 72 | 35 | 155 | 33.600000 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 155 | 26.600000 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 29 | 155 | 23.300000 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.100000 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.100000 | 2.288 | 33 | 1 |
| **5** | 5 | 116 | 74 | 29 | 155 | 25.600000 | 0.201 | 30 | 0 |
| **6** | 3 | 78 | 50 | 32 | 88 | 31.000000 | 0.248 | 26 | 1 |
| **7** | 10 | 115 | 72 | 29 | 155 | 35.300000 | 0.134 | 29 | 0 |
| **8** | 2 | 197 | 70 | 45 | 543 | 30.500000 | 0.158 | 53 | 1 |
| **9** | 8 | 125 | 96 | 29 | 155 | 32.457464 | 0.232 | 54 | 1 |

In [262]:

*# Visually exploring these variables using histograms.*

plt.style.use('ggplot')

plt.figure(figsize=(1000,800))

hcare\_data.hist(bins='auto',color='blue')

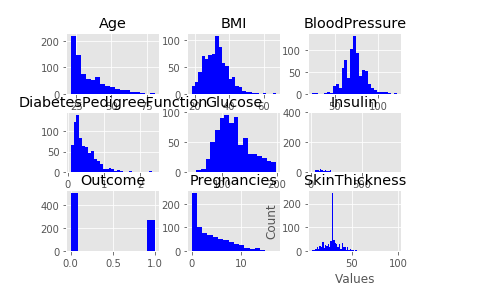
plt.xlabel('Values')

plt.ylabel('Count')

plt.show()

Output:

<Figure size 72000x57600 with 0 Axes>



In [263]:

*# Count (frequency) plot describing the data types and the count of variables.*

h\_datatypes = hcare\_data.dtypes

h\_datatypes

h\_datatypes.to\_csv('h\_dtypes.csv',header=["Data\_Types"],index=**False**)

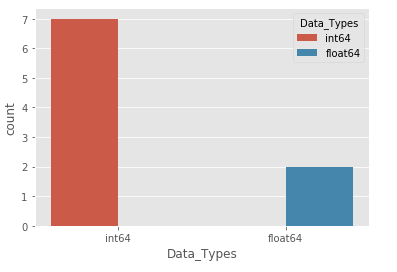
h\_dtypes = pd.read\_csv('h\_dtypes.csv')

*#h\_dtypes.head()*

h\_data\_plt = sns.countplot(x="Data\_Types", hue = "Data\_Types", data=h\_dtypes)

plt.show()

Output:



**Data Exploration:**

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

In [281]:

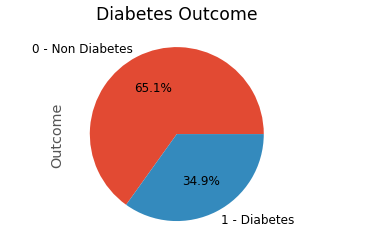
*# Plotting the count of outcomes by their value*

labels = '0 - Non Diabetes', '1 - Diabetes'

hcare\_data['Outcome'].value\_counts().plot.pie(autopct='**%1.1f%%**', labels = labels)

plt.title("Diabetes Outcome")

plt.show()



In [282]:

*# Scatter charts between the pair of variables to understand the relationships*

sns.pairplot(hcare\_data, hue = 'Outcome')

Out[282]:

<seaborn.axisgrid.PairGrid at 0x24f984b2308>





In [283]:

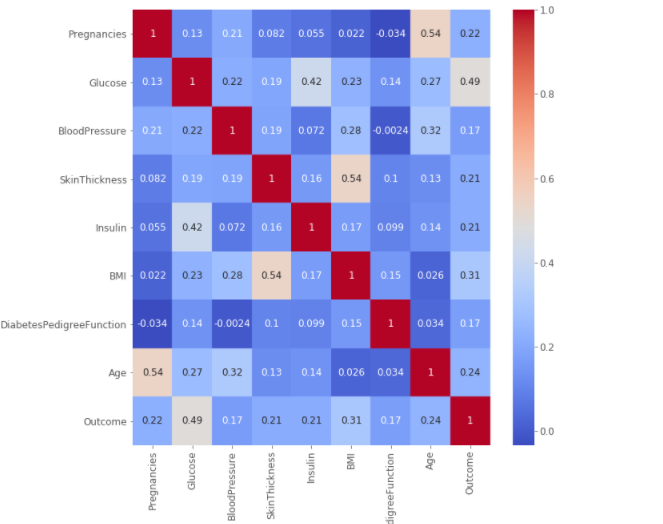
*# Performing correlation analysis visualizing it with heat map*

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

sns.heatmap(hcare\_data.corr(), annot = **True**, cmap = 'coolwarm')

Out[283]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x24f9a5524c8>



In [284]:

hcare\_data.columns

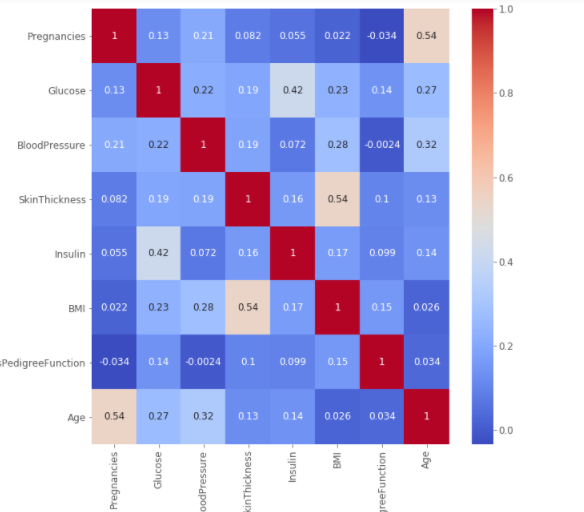
corr\_cols = hcare\_data[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction', 'Age']]

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

sns.heatmap(corr\_cols.corr(), annot = **True**, cmap = 'coolwarm')

Out[284]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x24f9a7ab488>



**Data Modeling:**

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

In [285]:

*#Train and Test Split the data*

**from** **sklearn.model\_selection** **import** train\_test\_split

*#Splitting the data*

X = hcare\_data[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction', 'Age']]

y = hcare\_data['Outcome']

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X , y, test\_size=0.2)

In [286]:

*# Strategies for model building*

*# k-fold cross validation evaluation of xgboost model*

**import** **xgboost**

**from** **numpy** **import** loadtxt

**from** **xgboost** **import** XGBClassifier

**from** **sklearn.model\_selection** **import** KFold

**from** **sklearn.model\_selection** **import** cross\_val\_score

*# CV model*

model = xgboost.XGBClassifier()

kfold = KFold(n\_splits=10, random\_state=7)

results = cross\_val\_score(model, X, y, cv=kfold)

print("Accuracy: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

Output:

Accuracy: 73.96% (4.56%)

In [287]:

*# Applying an appropriate classification algorithm to build a model and comparing with the results from KNN algorithm.*

*# Predict the accuracy using random forest classifier.*

**from** **sklearn.ensemble** **import** RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=100)

rfc.fit(X\_train,y\_train)

y\_pred = rfc.predict(X\_test)

*# Imported scikit-learn metrics module for accuracy calculation*

**from** **sklearn** **import** metrics

print("Accuracy: **%.2f%%** " % ((metrics.accuracy\_score(y\_test, y\_pred))\*100))

Output:

Accuracy: 74.68%

In [299]:

*# Applying Logistic Regression*

**from** **sklearn.linear\_model** **import** LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

print ("Accuracy : **%.2f%%**" % (metrics.accuracy\_score(y\_test, y\_pred)\*100))

Output:

Accuracy : 76.62%

In [300]:

*# Applying KNN Algorithm*

**from** **sklearn.neighbors** **import** KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train)

pred\_matrix = knn.predict(X\_test)

print(pred\_matrix)

acc\_res = knn.score(X\_test, y\_test)

print("Accuracy: **%.2f%%** " % (acc\_res\*100))

Output:

[0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1

0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 0

0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0

0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 1

0 0 1 0 0 1]

Accuracy: 69.48%

**Data Modeling:**

1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

In [290]:

*# Classification report by analyzing sensitivity, specificity, AUC (ROC curve)*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.metrics** **import** roc\_curve

**from** **sklearn.metrics** **import** roc\_auc\_score

*# Cross-validated AUC*

cross\_val\_score(classifier, X, y, cv=10, scoring='roc\_auc').mean()

Out[290]:

Output:

0.8288831908831907

In [291]:

*# Confusion Matrix*

confusion\_mat = metrics.confusion\_matrix(y\_test, y\_pred)

confusion\_mat

Out[291]:

array([[80, 10],

[26, 38]], dtype=int64)

In [292]:

print('True', y\_test.values[0:25])

print('Pred', y\_pred[0:25])

TP = confusion\_mat[1, 1]

TN = confusion\_mat[0, 0]

FP = confusion\_mat[0, 1]

FN = confusion\_mat[1, 0]

Output:

True [0 0 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 0 0]

Pred [0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0]

In [293]:

sensitivity = TP / float(FN + TP)

print(sensitivity)

print(metrics.recall\_score(y\_test, y\_pred))

Output:

0.59375

0.59375

In [294]:

specificity = TN / (TN + FP)

print(specificity)

Output:

0.8888888888888888

In [295]:

false\_positive\_rate = FP / float(TN + FP)

print(false\_positive\_rate)

print(1 - specificity)

Output:

0.1111111111111111

0.11111111111111116

In [296]:

precision = TP / float(TP + FP)

print(precision)

print(metrics.precision\_score(y\_test, y\_pred))

Output:

0.7916666666666666

0.7916666666666666

In [297]:

fpr, tpr, thresholds = metrics.roc\_curve(y\_test, y\_pred)

plt.plot(fpr, tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

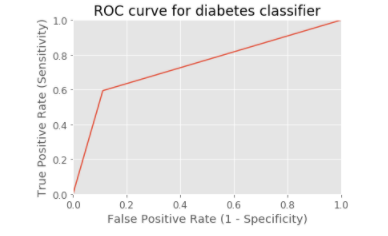
plt.rcParams['font.size'] = 12

plt.title('ROC curve for diabetes classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.grid(**True**)



**Insights:**

* Missing values are handled.
* According to the outcome there are 65.1% diabetic patients.
* There is a positive correlation as the coefficient is 1.
* Accuracy of k-fold cross validation evaluation of xgboost model is 73.96%
* Accuracy of random forest classifier is 74.68%.
* Accuracy for Applying Logistic Regression is 76.62%
* Accuracy after applying KNN Algorithm is 69.48%
* KNN is comparitively slower than Logistic Regression. In KNN, you run the algorithm n times, the results will be the same but in Logistic Regression, you run the algorithm many times you'll see the results varying.
* Sensitivity(True Positive Rate or Recall) = TP/(TP+FN)
* False Negative Rate(FNR) = FN/(TP+FN)
* Specificity(True Negative Rate) = TN/(TN+FP)
* False Positive Rate(FPR) = FP/(TN+FP) = 1-Specificity
* The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.
* When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values.