```
In [49]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
```

```
In [50]: df=pd.read_csv(r'C:\Users\sutharsan\Downloads\capstone\project cp 1\Project_2\Project 2\Healthcare - Diabetes\health care diabetes.csv')
```

```
In [51]: df.head()
```

Out[51]:

	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

Project Task: Week 1

```
In [52]: df.describe()    # 1.1. Perform descriptive analysis
```

Out[52]:

	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 [53]: #  Understand the variables and their corresponding values.
         #On the columns below, a value of zero does not make sense and thus indicates missing value:

         df['Glucose']=df['Glucose'].replace(0,df['Glucose'].median())
         df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].median())
         df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].mean())
         df['Insulin']=df['Insulin'].replace(0,df['Insulin'].median())
         df['BMI']=df['BMI'].replace(0,df['BMI'].mean())
```

```
In [54]: df.head()
```

Out[54]:

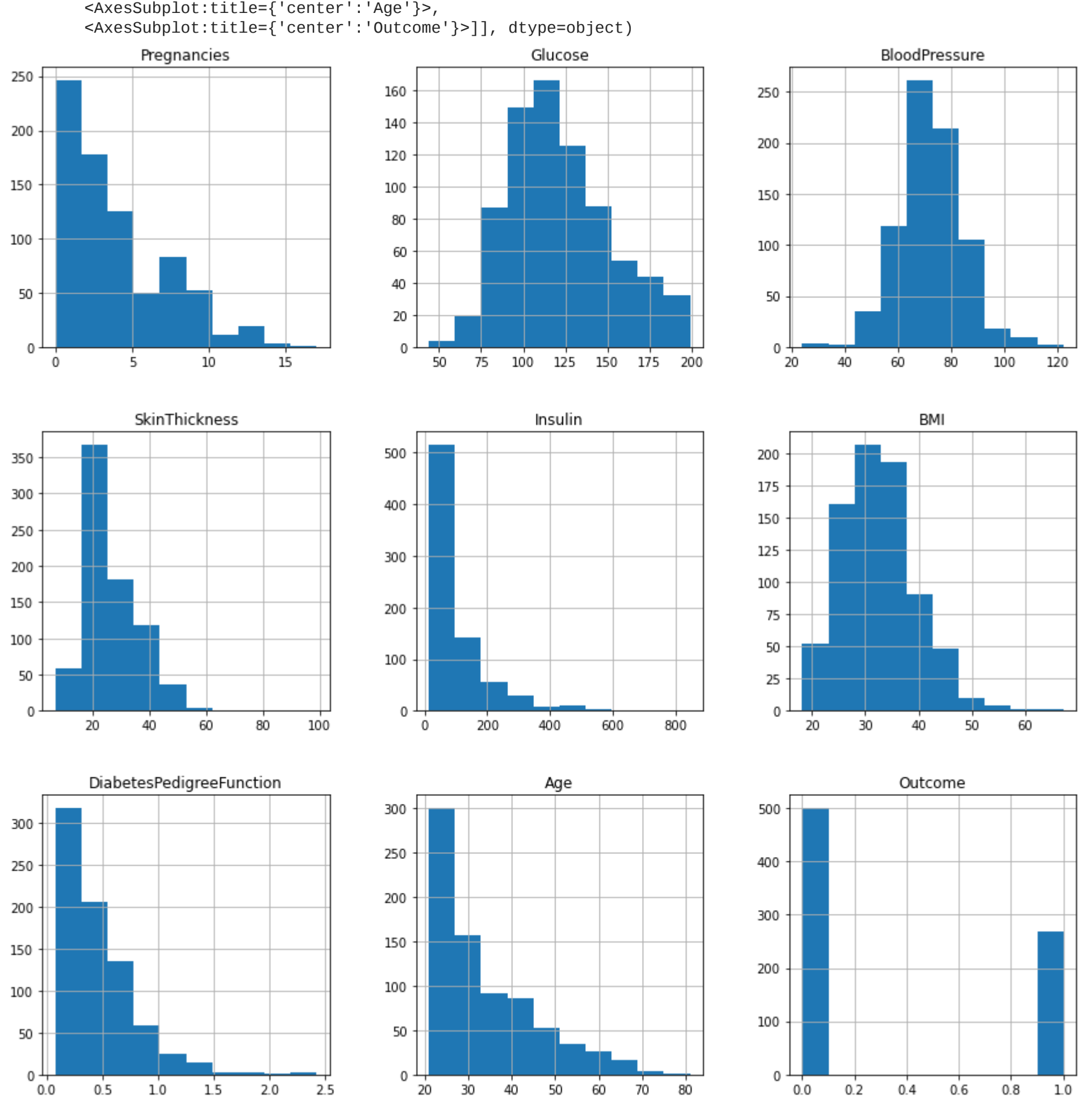
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35.000000	30.5	33.6	0.627	50	1
1	1	85	66	29.000000	30.5	26.6	0.351	31	0
2	8	183	64	20.536458	30.5	23.3	0.672	32	1
3	1	89	66	23.000000	94.0	28.1	0.167	21	0
4	0	137	40	35.000000	168.0	43.1	2.288	33	1

```
In [55]: df.isnull().any()   # Treat the missing values accordingly.
```

```
Out[55]: Pregnancies                 False
         Glucose                     False
         BloodPressure               False
         SkinThickness               False
         Insulin                     False
         BMI                         False
         DiabetesPedigreeFunction    False
         Age                         False
         Outcome                     False
         dtype: bool
```

```
In [56]: #2. Visually explore these variables using histograms.
         df.hist(figsize=(15,15))
```

```
Out[56]: array([[<AxesSubplot:title={'center':'Pregnancies'}>,
                 <AxesSubplot:title={'center':'Glucose'}>,
                 <AxesSubplot:title={'center':'BloodPressure'}>],
                [<AxesSubplot:title={'center':'SkinThickness'}>,
                 <AxesSubplot:title={'center':'Insulin'}>,
                 <AxesSubplot:title={'center':'BMI'}>],
                [<AxesSubplot:title={'center':'DiabetesPedigreeFunction'}>,
                 <AxesSubplot:title={'center':'Age'}>,
                 <AxesSubplot:title={'center':'Outcome'}>]], dtype=object)
```

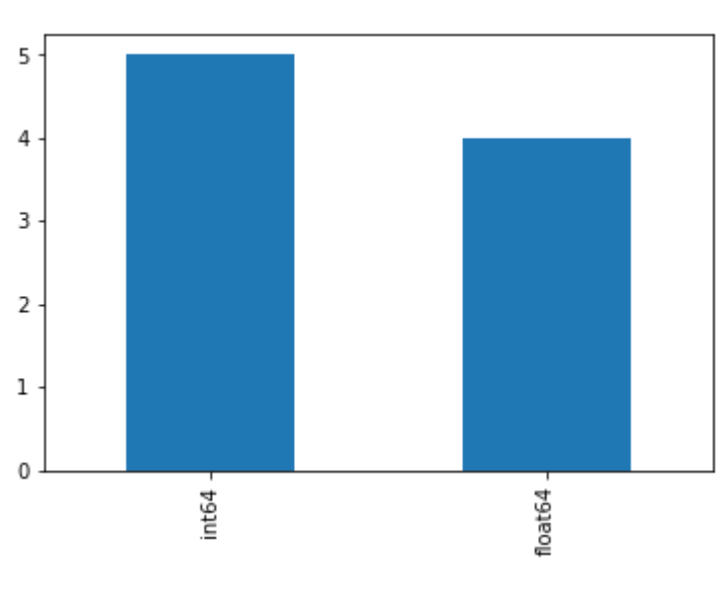



```
In [57]: df.dtypes
```

```
Out[57]: Pregnancies                   int64
         Glucose                       int64
         BloodPressure                 int64
         SkinThickness               float64
         Insulin                     float64
         BMI                         float64
         DiabetesPedigreeFunction    float64
         Age                           int64
         Outcome                       int64
         dtype: object
```

```
In [58]: # df_count=df.dtypes.value_counts()
         df_count.plot(kind='bar')
```

Out[58]: <AxesSubplot:>



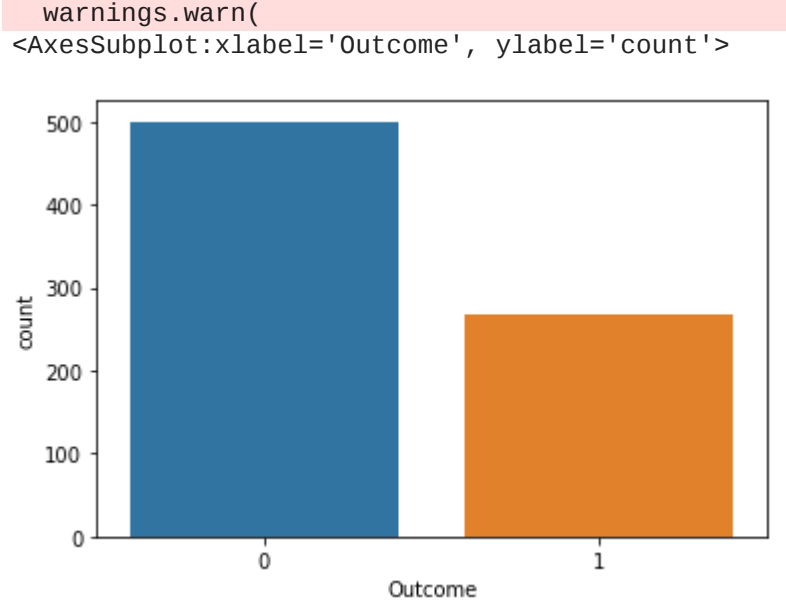
Project Task: Week 2

Data Exploration:

```
In [59]: #1.Check the balance of the data by plotting the count of outcomes by their value.
         import seaborn as sns
         sns.countplot(df['Outcome'])
```

```
C:\Users\sutharsan\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[59]: <AxesSubplot:xlabel='Outcome', ylabel='count'>




```
In [60]: from sklearn.model_selection import train_test_split
         y=df['Outcome']
         x=df.drop(['Outcome'],axis=1)
         x_train,x_test,y_train ,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
```

```
In [61]: # imbalance so use SMOTE

         from imblearn.over_sampling import SMOTE
         sm = SMOTE(random_state = 2)
         x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())
         print("After smote '1' {}".format(sum(y_train_res == 1)))
         print("After smote'0': {}".format(sum(y_train_res == 0)))
```

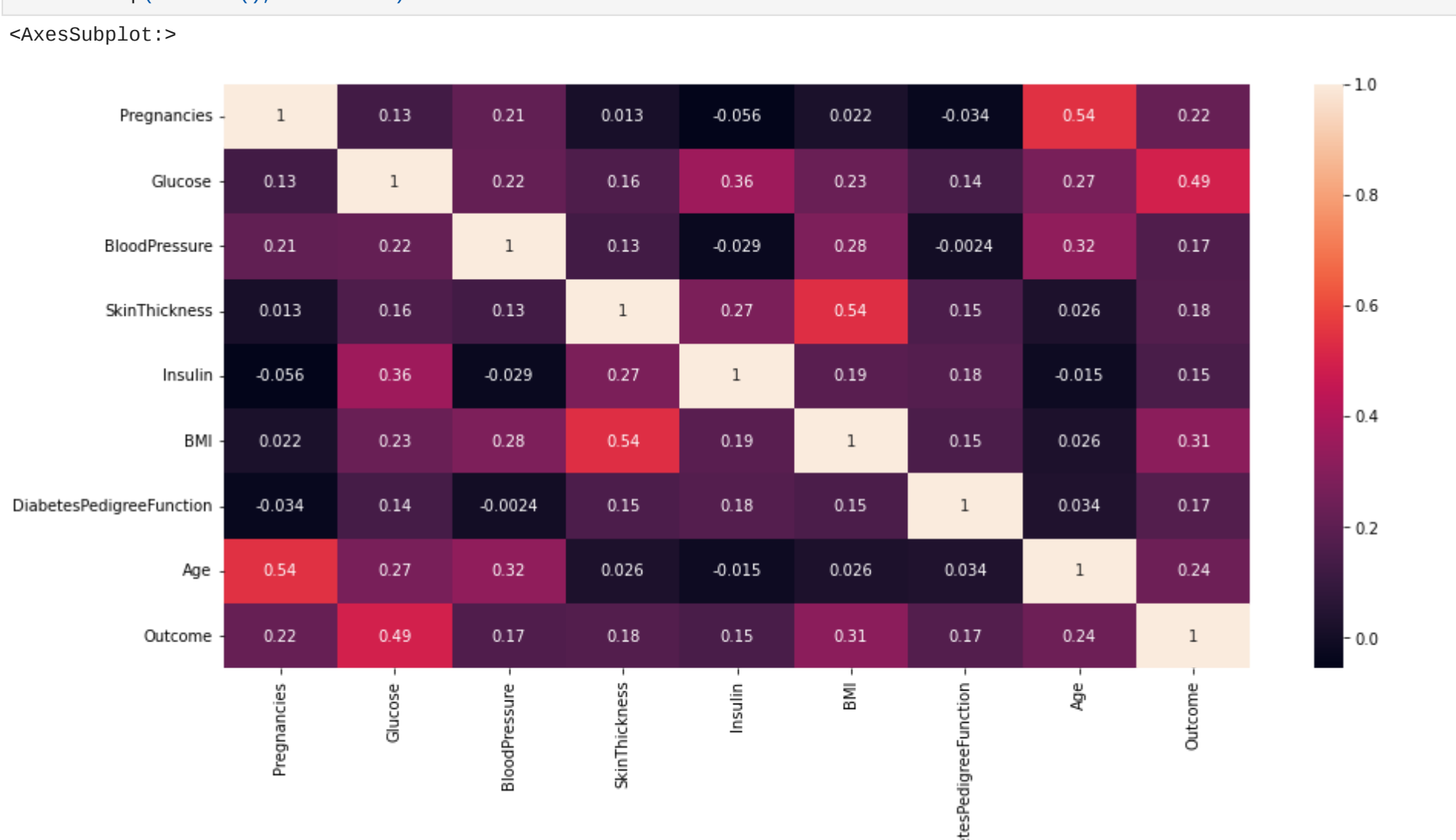
```
After smote '1' 354
After smote'0': 354
```

```
In [62]: sns.pairplot(df) #2.Create scatter charts between the pair of variables to understand the relationships.
```

Out[62]: <seaborn.axisgrid.PairGrid at 0xc83b6a0>


```
In [63]: # Describe your findings.
         plt.figure(figsize=(15,7))
         sns.heatmap(df.corr(),annot=True)
```

Out[63]: <AxesSubplot:>




```
In [64]:  # gulcose level affect outcome
          # pregnancy and age has correlation
          # skin thickness and bmi too
```

Project Task: Week 3

Data Modeling:

```
In [65]: #1. Devise strategies for model building.
         #It is important to decide the right validation framework. Express your thought process.
         from sklearn.linear_model import LogisticRegression # logistic reg is better binary classification
         from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score
         lr=LogisticRegression()
         lr.fit(x_train,y_train)
         score=lr.score(x_train,y_train)
         y_pred_lr=lr.predict(x_test)
         acc=accuracy_score(y_test,y_pred_lr)
         print('acc is :',acc *100)
```

```
acc is : 78.78787878787878
```

```
C:\Users\sutharsan\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
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
  n_iter_i = _check_optimize_result(
```

```
In [66]: from sklearn.neighbors import KNeighborsClassifier
         knn= KNeighborsClassifier(n_neighbors=6)
         knn.fit(x_train, y_train)
         score=knn.score(x_train,y_train)
         y_pred_knn=knn.predict(x_test)
         acc=accuracy_score(y_test,y_pred_knn)
         print('acc is :',acc *100)
```

```
acc is : 75.75757575757575
```

```
In [67]: # 2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.naive_bayes import GaussianNB
         from sklearn.svm import SVC
```

```
In [68]: classification_models = []
         classification_models.append(('Kernel SVM', SVC(kernel = 'rbf',gamma='scale')))
         classification_models.append(('Naive Bayes', GaussianNB()))
         classification_models.append(('Decision Tree', DecisionTreeClassifier(criterion = "entropy")))
         classification_models.append(('Random Forest', RandomForestClassifier(n_estimators=100, criterion="entropy")))
```

```
In [69]: for name, model in classification_models:
           kfold = KFold(n_splits=10, random_state=(7), shuffle=(True))
           result = cross_val_score(model, x, y, cv=kfold, scoring='accuracy')
           print("%s: Mean Accuracy = %.2f%% - SD Accuracy = %.2f%%" % (name, result.mean()*100, result.std()*100))
```

```
Kernel SVM: Mean Accuracy = 76.31% - SD Accuracy = 3.65%
Naive Bayes: Mean Accuracy = 75.14% - SD Accuracy = 3.59%
Decision Tree: Mean Accuracy = 67.85% - SD Accuracy = 4.00%
Random Forest: Mean Accuracy = 76.69% - SD Accuracy = 4.75%
```

Project Task: Week 4

Data Modeling:

```
In [70]: #1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.
         print(confusion_matrix(y_test,y_pred_lr))
```

```
[[131  15]
 [ 34  51]]
```

```
In [71]: print(classification_report(y_test,y_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       146
           1       0.77      0.60      0.68        85

    accuracy                           0.79       231
   macro avg       0.78      0.75      0.76       231
weighted avg       0.79      0.79      0.78       231
```

```
In [72]: roc_auc_score(y_test,y_pred_lr)
```

```
Out[72]: 0.7486301369863014
```

```
In [73]: from sklearn import metrics
         precision = metrics.precision_score(y_test, y_pred_lr)
         print("Precision score:",precision)  #specificity,
         recall = metrics.recall_score(y_test, y_pred_lr)
         print("Recall score:",recall)    # sensitivity
```

```
Precision score: 0.7727272727272727
Recall score: 0.6
```

```
In [74]: print(confusion_matrix(y_test,y_pred_knn))
```

```
[[128  18]
 [ 38  47]]
```

```
In [75]: print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       146
           1       0.72      0.55      0.63        85

    accuracy                           0.76       231
   macro avg       0.75      0.71      0.72       231
weighted avg       0.75      0.76      0.75       231
```

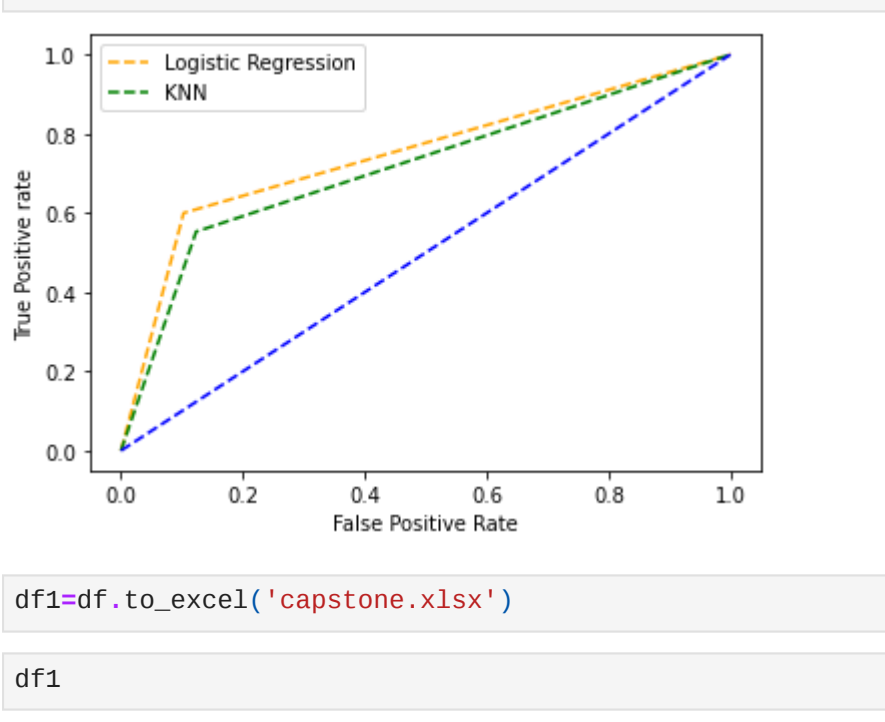
```
In [76]: roc_auc_score(y_test,y_pred_lr)
```

```
Out[76]: 0.7486301369863014
```

```
In [77]: from sklearn.metrics import roc_curve
         fpr1, tpr1, thresh1 = roc_curve(y_test, y_pred_lr, pos_label=1)
         fpr2, tpr2, thresh2 = roc_curve(y_test, y_pred_knn, pos_label=1)
         prob = [0 for i in range(len(y_test))]
         p_fpr, p_tpr, _ = roc_curve(y_test, prob, pos_label=1)
         auc_score1 = roc_auc_score(y_test, y_pred_lr)
         auc_score2 = roc_auc_score(y_test, y_pred_knn)
         print(auc_score1, auc_score2)
```

```
0.7486301369863014 0.7148267526188558
```

```
In [78]: plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
         plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
         plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive rate')
         plt.legend(loc='best')
         plt.show();
```




```
In [79]: df1=df.to_excel('capstone.xlsx')
```

```
In [82]: df1
```

```
In [ ]:
```