In [ ]:
```
"DIABETES PATIENTS"
```

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#loading dataset
diab = pd.read_csv('C:\\Users\\hp221\\OneDrive\\Desktop\\Document\\python310\\diabetes.csv')
diab
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

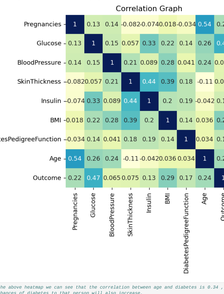
In [3]:
```
#checking for null values
diab.isna().sum()
```

Out[3]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [ ]:
```
#so we can see that there is not any null value present in given dataset.
```

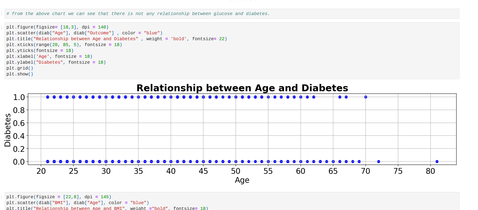
In [4]:
```
#creating correlation graph of dataset between each columns
plt.figure(figsize = (8,5), dpi = 130)
plt.title("Correlation Graph", fontsize= 12)
sns.heatmap(diab.corr(), annot= True, cmap = "YlGnBu")
```

Out[4]:
```
<Axes: title={'center': 'Correlation Graph'}>
```



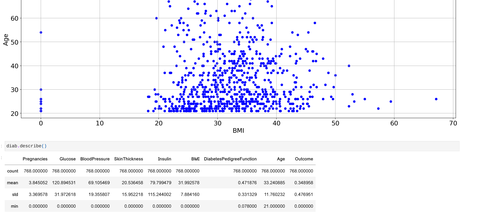
In [ ]:
```
#from the above heatmap we can see that the correlation between age and diabetes is 0.34 , it's highly correlated means if age of person will increase ,
#then chances of diabetes to that person will also increase.
```

In [5]:
```
plt.figure(figsize= (16,3), dpi = 140)
plt.scatter(diab["Glucose"], diab["Outcome"] , color = "blue")
plt.title("Relationship between Glucose and Diabetes" , weight = 'bold', fontsize= 22)
plt.xticks(range(0, 205, 10), fontsize = 18)
plt.yticks(range(0,1.2,0.2))
plt.xlabel("Glucose", fontsize = 18)
plt.ylabel("Diabetes", fontsize = 18)
plt.grid()
plt.show()
```

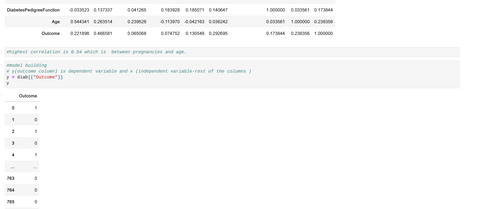


In [ ]:
```
# from the above chart we can see that there is not any relationship between glucose and diabetes.
```

In [6]:
```
plt.figure(figsize= (16,3), dpi = 140)
plt.scatter(diab["Age"], diab["Outcome"] , color = "blue")
plt.title("Relationship between Age and Diabetes" , weight = 'bold', fontsize= 22)
plt.xticks(range(20, 85, 5), fontsize = 18)
plt.yticks(range(0,1.2,0.2))
plt.xlabel("Age", fontsize = 18)
plt.ylabel("Diabetes", fontsize = 18)
plt.grid()
plt.show()
```



In [22]:
```
plt.figure(figsize = (22,6), dpi = 105)
plt.scatter(diab["BMI"], diab["Age"], color = "blue")
plt.title("Relationship between Age and BMI", weight = "bold", fontsize= 18)
plt.xticks(range(0,80,10), fontsize = 18)
plt.yticks(range(20,90, 10), fontsize = 18)
plt.xlabel("BMI", fontsize = 18)
plt.ylabel("Age", fontsize = 18)
plt.grid()
plt.show()
```



In [23]:
```
diab.describe()
```

Out[23]:

	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 [24]:
```
diab.corr()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

In [ ]:
```
#highest correlation is 0.54 which is  between pregnancies and age.
```

In [9]:
```
#model building
# y(outcome column) is dependent variable and x (independent variable-rest of the columns )
y = diab[["Outcome"]]
y
```

Out[9]:

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

768 rows × 1 columns

In [10]:
```
x = diab.loc[:, diab.columns!= 'Outcome']
x
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

In [11]:
```
#importing train_test_split
from sklearn.model_selection import train_test_split
```

In [12]:
```
# dividing the dataset into TRN(training) and TRN(testing)
x_train, x_test, y_train, y_test = train_test_split(x,y,train_size = 0.70, random_state = 42)
```

In [13]:
```
#printing no. of rows and columns used for training and testing
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Out[13]:
```
(537, 8)
(537, 1)
(231, 8)
(231, 1)
```

In [14]:
```
#importing logistic regression model
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
```

In [15]:
```
#training the model
model.fit(x_train, y_train)
```

Out[15]:
```
C:\Users\hp221\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\validation.py:1183: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\hp221\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear_model\_logistic.py:460: 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(
```

```
▾ LogisticRegression
LogisticRegression()
```

In [16]:
```
#testing the model
prediction = model.predict(x_test)
prediction
```

Out[16]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0]), dtype=int64)
```

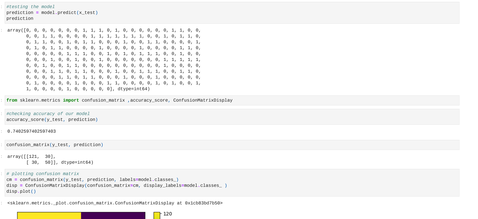
In [18]:
```
from sklearn.metrics import confusion_matrix ,accuracy_score, ConfusionMatrixDisplay
```

In [21]:
```
#checking accuracy of our model
accuracy_score(y_test, prediction)
```

Out[21]:
```
0.7402597402597402
```

In [22]:
```
confusion_matrix(y_test, prediction)
```

Out[22]:
```
array([[121,  30],
       [ 30,  50]], dtype=int64)
```

In [23]:
```
# plotting confusion matrix
cm = confusion_matrix(y_test, prediction, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_ )
disp.plot()
```

Out[23]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c083bd7550>
```