In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardSc
import warnings
from IPython.display import Image

```

In [6]:
```
display(Image(filename='maxresdefault.jpg', width=600, height=400))

```

In [7]:
```
df = pd.read_csv(r"C:\Users\anike\OneDrive\Desktop\diabetes2.csv")
```

In [8]:
```
1 df
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6.0	148.0	72	35	0	33.6	
1	1.0	85.0	66	29	0	26.6	
2	8.0	183.0	64	0	0	23.3	
3	1.0	NaN	66	23	94	28.1	
4	0.0	NaN	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10.0	101.0	76	48	180	32.9	
764	2.0	122.0	70	27	0	36.8	
765	5.0	121.0	72	23	112	26.2	
766	1.0	126.0	60	0	0	30.1	
767	1.0	93.0	70	31	0	30.4	

768 rows × 9 columns

In [9]:
```
1 df.head()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6.0	148.0	72	35	0	33.6	0.
1	1.0	85.0	66	29	0	26.6	0.
2	8.0	183.0	64	0	0	23.3	0.
3	1.0	NaN	66	23	94	28.1	0.
4	0.0	NaN	40	35	168	43.1	2.

In [10]:
```
1 df.tail()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
763	10.0	101.0	76	48	180	32.9	
764	2.0	122.0	70	27	0	36.8	
765	5.0	121.0	72	23	112	26.2	
766	1.0	126.0	60	0	0	30.1	
767	1.0	93.0	70	31	0	30.4	

In [11]:
```
1 df.describe()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	748.000000	714.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.822193	120.470588	69.105469	20.536458	79.799479	31.992578	
std	3.359606	31.850410	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.000000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [12]:
```
1 df.describe()
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	748.000000	714.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.822193	120.470588	69.105469	20.536458	79.799479	31.992578	
std	3.359606	31.850410	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.000000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [13]:
```
1 df.isnull().sum()
```

Out[13]:
```
Pregnancies                 20
Glucose                     54
BloodPressure                0
SkinThickness                0
Insulin                      0
BMI                          0
DiabetesPedigreeFunction     0
Age                          0
Outcome                      0
dtype: int64
```

In [14]:
```
1 df['Pregnancies'].fillna(1.0,inplace = True)
2 df['Glucose'].fillna(1.0,inplace = True)
3
```

In [15]:
```
1 df.isnull().sum()
```

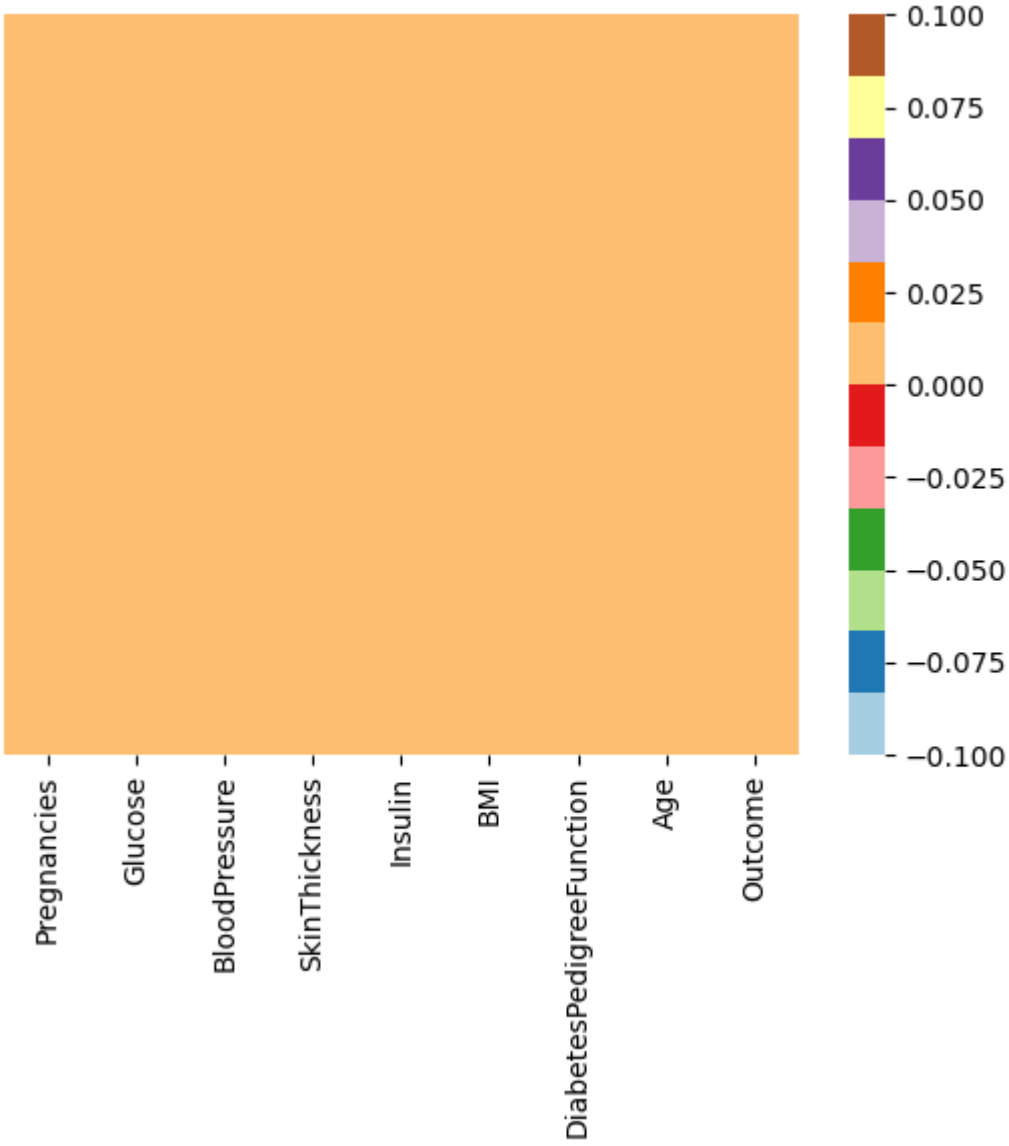
Out[15]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [16]:
```
1 df.dtypes
```

Out[16]:
```
Pregnancies                 float64
Glucose                     float64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [17]:
```
1 sns.heatmap(df.isnull(),yticklabels=False,cmap="Paired")
```

Out[17]: <Axes: >



EDA

In [18]:
```
1 df_1 = df.groupby('BloodPressure')['SkinThickness'].count()
```

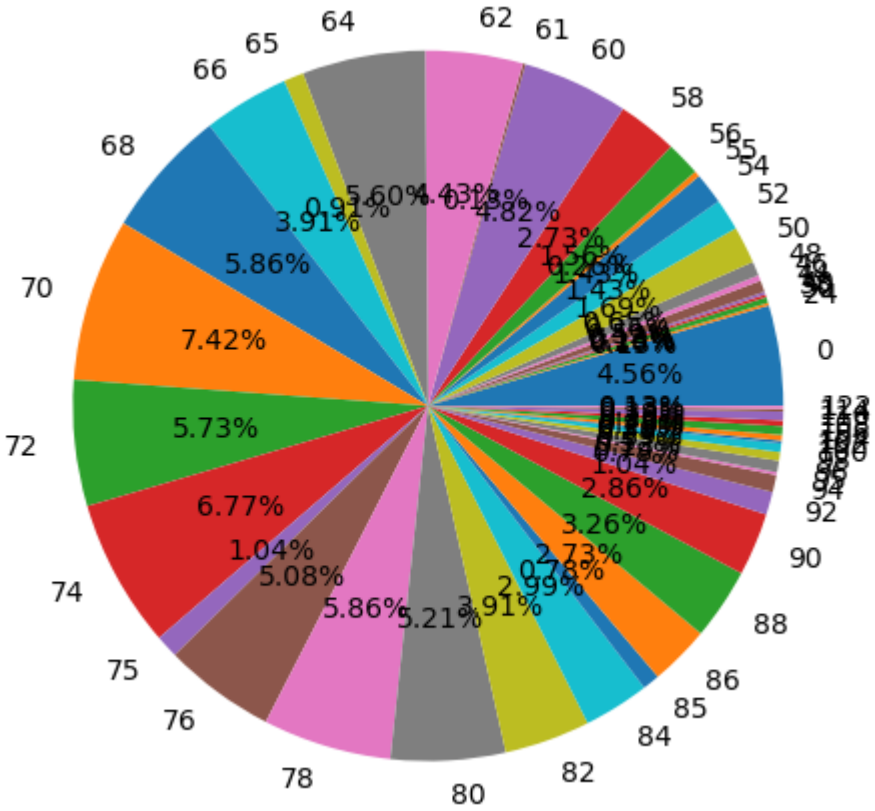
In [19]:
```
1 df_1
```

Out[19]:

	BloodPressure
0	35
24	1
30	2
38	1
40	1
44	4
46	2
48	5
50	13
52	11
54	11
55	2
56	12
58	21
60	37
61	1
62	34
64	43
65	7
66	30
68	45
70	57
72	44
74	52
75	8
76	39
78	45
80	40
82	30
84	23
85	6
86	21
88	25
90	22
92	8
94	6
95	1
96	4
98	3
100	3
102	1
104	2
106	3
108	2
110	3
114	1
122	1

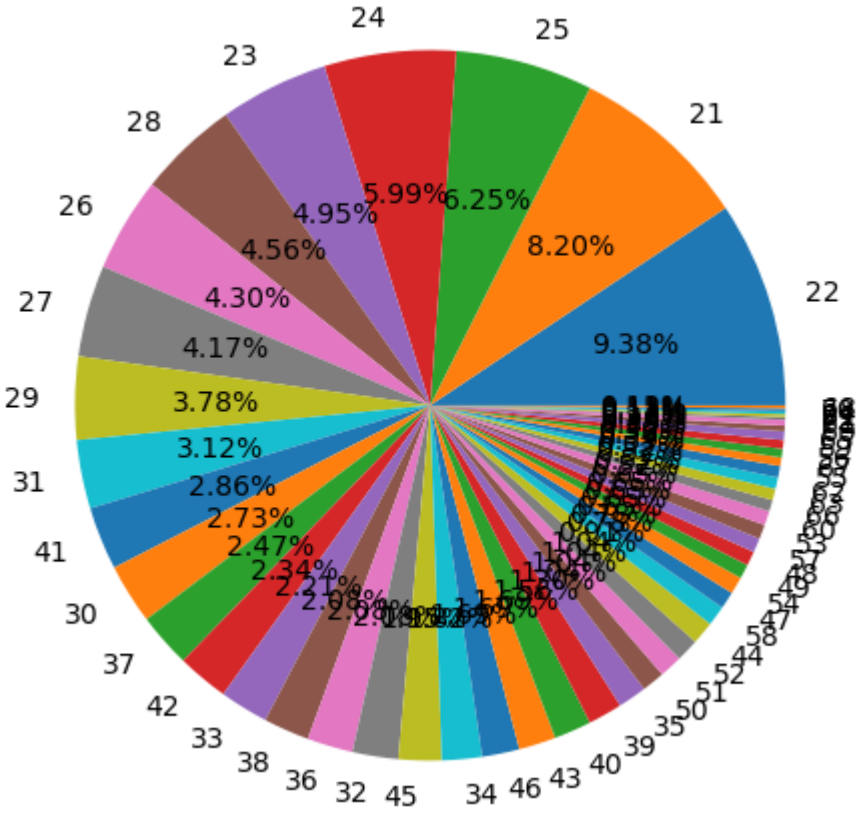
Name: SkinThickness, dtype: int64

In [20]:
```
plt.pie(df_1,labels=df_1.index,autopct="%.2f%%",radius =1.2)
plt.show()
```



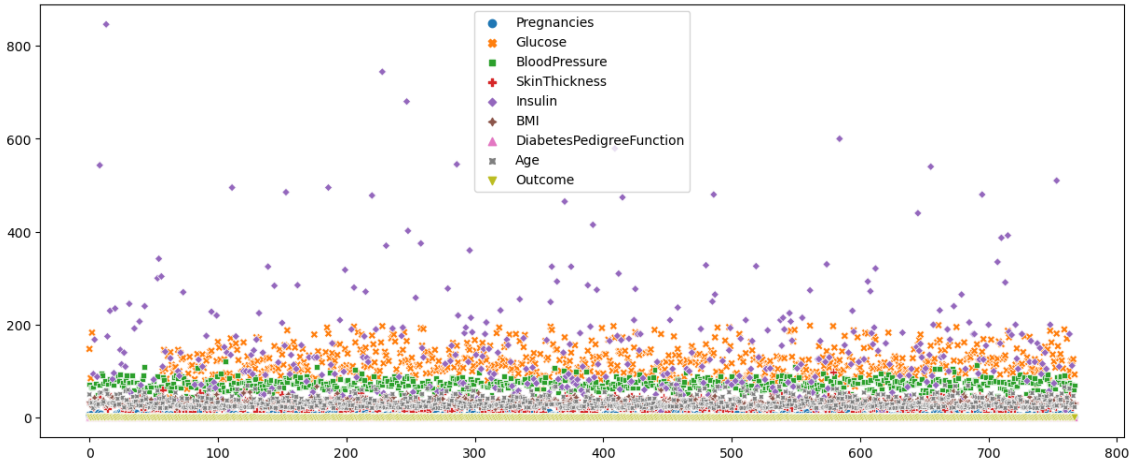
In [22]:
```
ed_2 = df['Age'].value_counts()
```

In [23]:
```
plt.pie(ed_2,labels=ed_2.index,autopct="%.2f%%",radius =1.2)
plt.show()
```



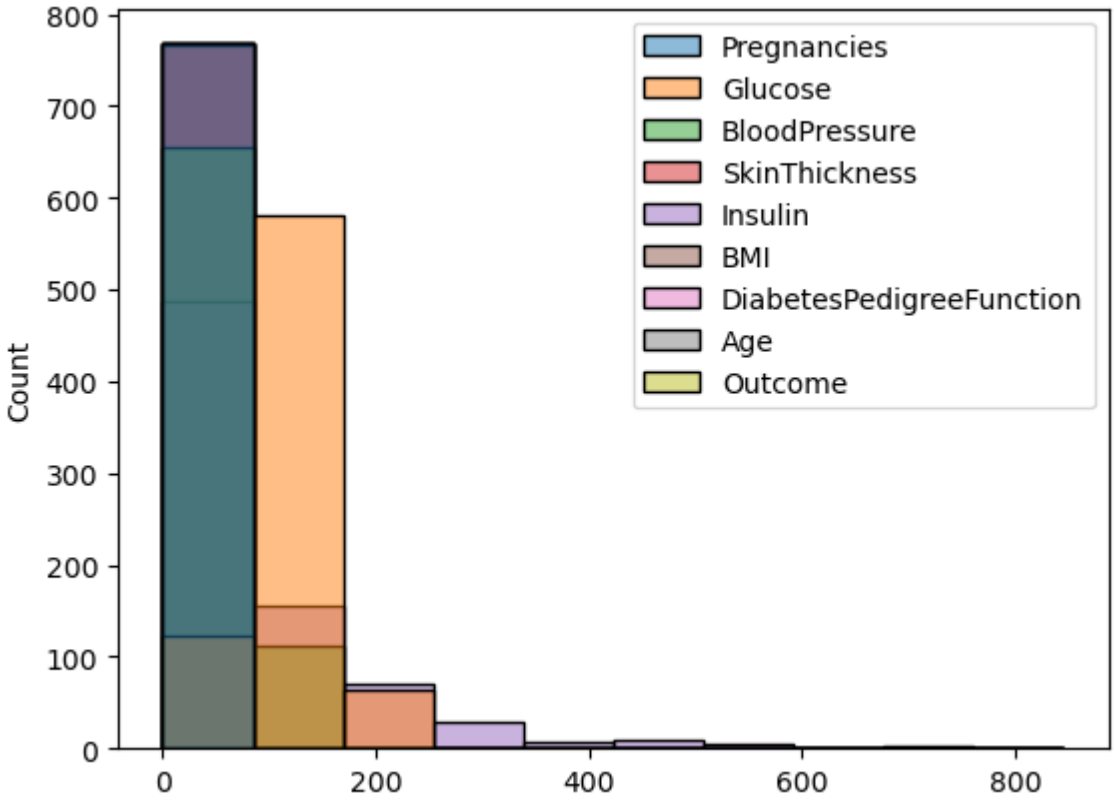
In [25]:
```
fig = plt.gcf();
fig.set_size_inches(15, 6);
sns.scatterplot(df)
```

Out[25]: <Axes: >

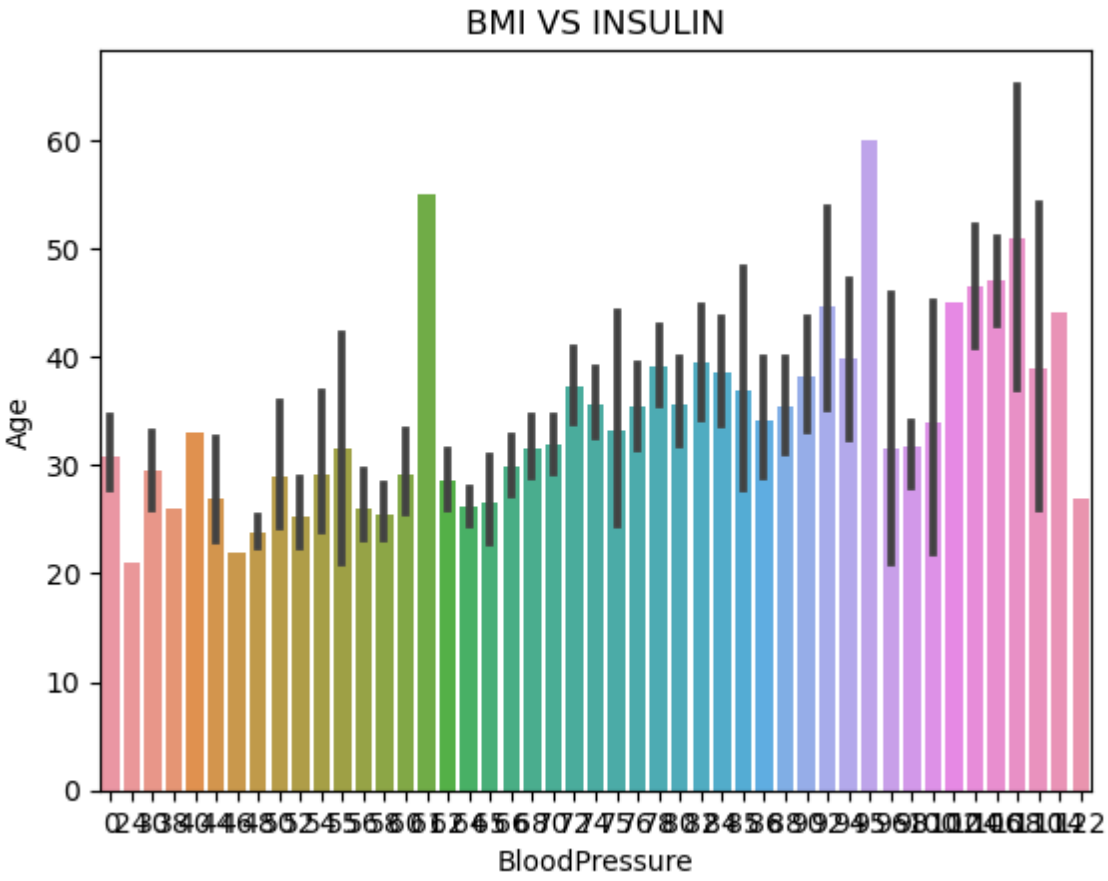


In [26]:
```
sns.histplot(df, bins=10, color='skyblue', kde=False)

```

Out[26]: <Axes: ylabel='Count'>

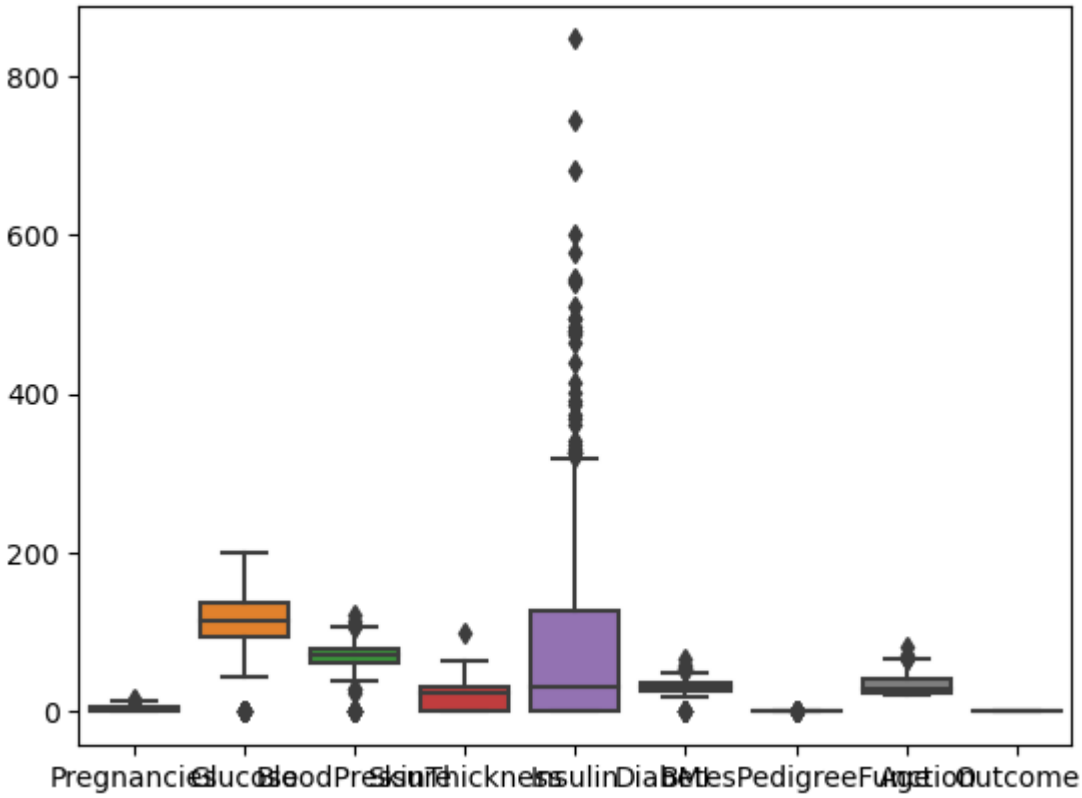


In [29]:
```
sns.barplot(x='BloodPressure',y='Age',data = df, orient='v')
plt.title('BMI VS INSULIN')
plt.show()
```

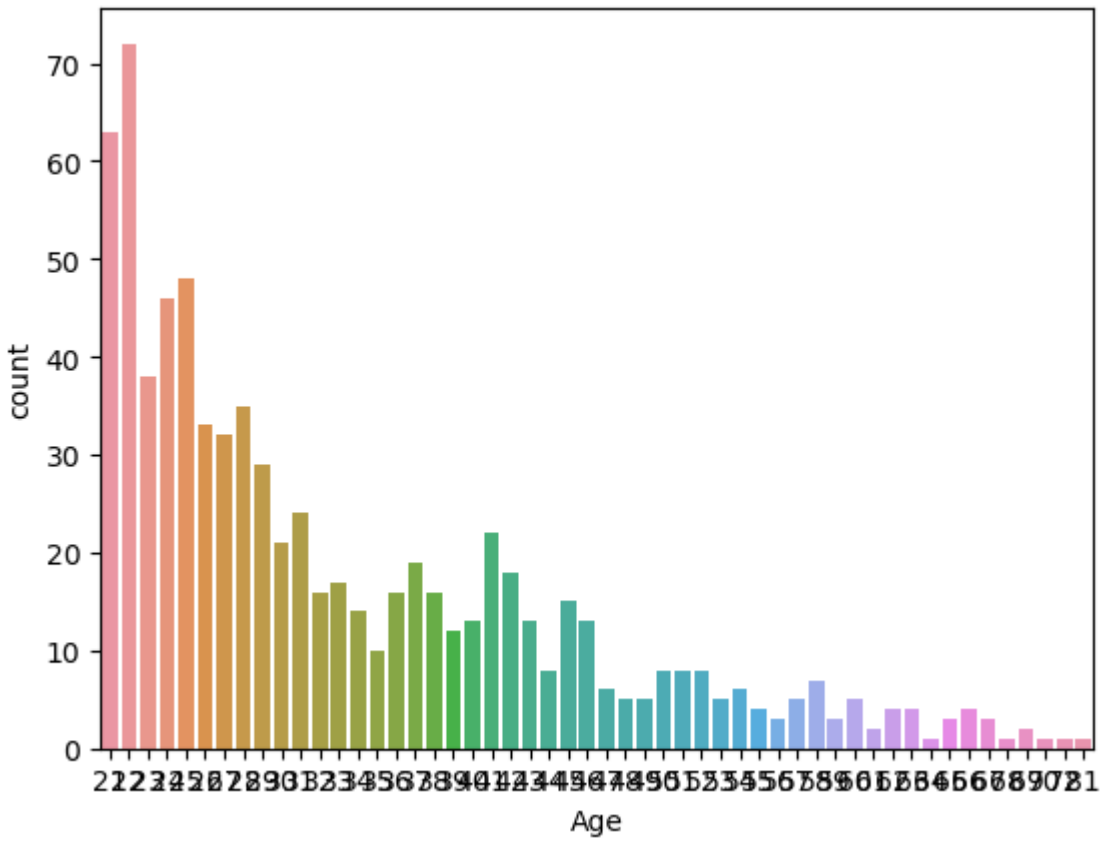


In [26]:
```
sns.boxplot(df)
```

Out[26]: <Axes: >



In [33]:
```
sns.countplot(x=df['Age'])
plt.show()
```



PERFORMING LABEL ENCODING AND CONVERTING ALL CATEGORICAL DATA IN NUMERICAL FORM

In [36]:
```
df.dtypes
```

Out[36]:
```
Pregnancies                 float64
Glucose                     float64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [34]:
```
from sklearn.preprocessing import OneHotEncoder,StandardScaler

```

In [35]:
```
encoder = LabelEncoder()
```

In [38]:
```
df['Pregnancies'] = encoder.fit_transform(df['Pregnancies'])
df['Glucose'] = encoder.fit_transform(df['Glucose'])
df['BMI'] = encoder.fit_transform(df['BMI'])
df['DiabetesPedigreeFunction'] = encoder.fit_transform(df['DiabetesPedi
```

In [39]:
```
numerical_cols=["Pregnancies","Glucose","BloodPressure","SkinThickness'
scaler=StandardScaler()
scaled_cols=pd.DataFrame(scaler.fit_transform(df[numerical_cols]),colur
```

In [40]:
```
x = df.drop(['Outcome'],axis=1)
y = df['Outcome']
```

In [41]:
```
x
```

Out[41]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	87	72	35	0	123	
1	1	24	66	29	0	62	
2	8	122	64	0	0	30	
3	1	1	66	23	94	77	
4	0	1	40	35	168	209	
...	...	...	...	...	...	...	
763	10	40	76	48	180	118	
764	2	61	70	27	0	155	
765	5	60	72	23	112	58	
766	1	65	60	0	0	95	
767	1	32	70	31	0	98	

768 rows × 8 columns

In [42]:
```
y
```

Out[42]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

PERFORMING LOGISTIC REGRESSION

In [43]:
```
from sklearn.linear_model import LogisticRegression

```

In [44]:
```
X_train,X_test,Y_train,Y_test= train_test_split(x,y,test_size=0.2,rando
```

In [45]:
```
log = LogisticRegression()
```

In [46]:
```
y.isnull().sum()
```

Out[46]:
```
0
```

In [47]:
```
log.fit(X_train,Y_train)
```

Out[47]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [48]:
```
print ('Train Score:',log.score(X_train,Y_train))
```

```
Train Score: 0.754071661237785
```

In [49]:
```
print('Test Score:',log.score(X_test,Y_test))
```

```
Test Score: 0.7857142857142857
```

In [50]:
```
pred_train = log.predict(X_train)
pred_test = log.predict(X_test)
```

In [51]:
```
from sklearn import metrics
```

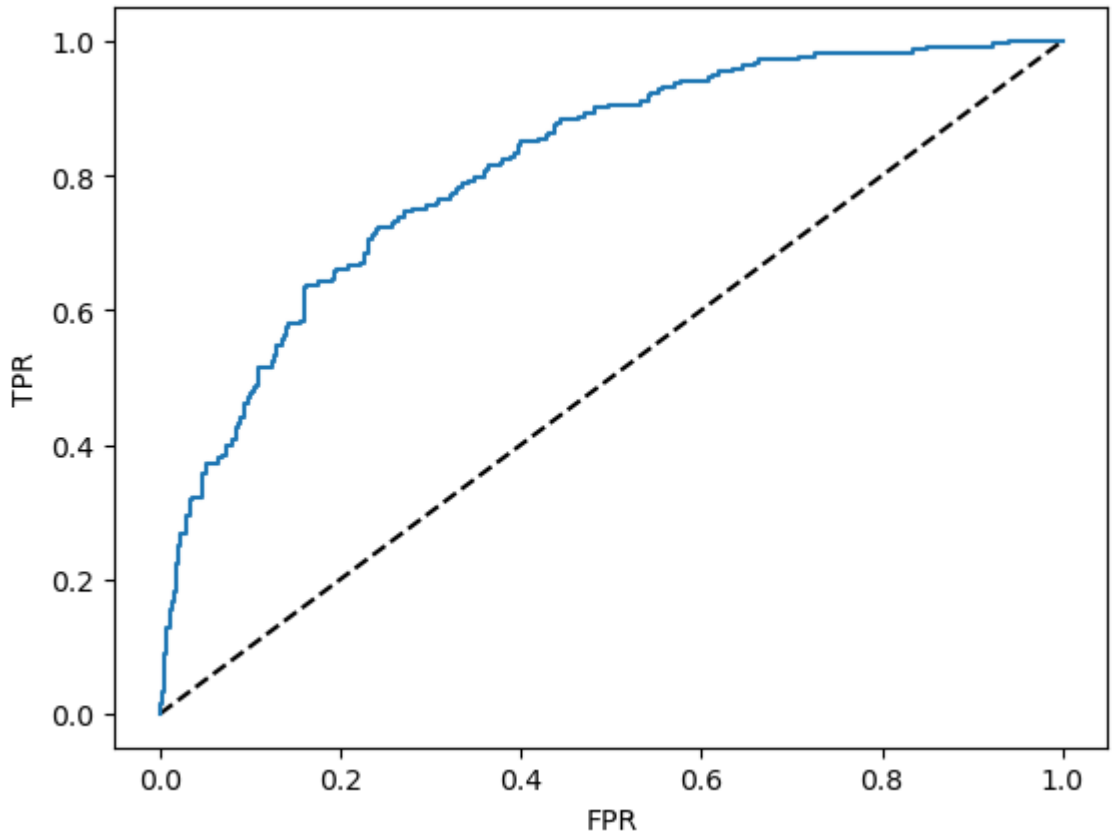
In [52]:
```
print(metrics.classification_report(Y_train,pred_train))
```

```
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       393
           1       0.70      0.56      0.62       221

    accuracy                           0.75       614
   macro avg       0.74      0.71      0.72       614
weighted avg       0.75      0.75      0.75       614
```

In [54]:
```
1 df.dtypes
```

Out[54]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                         int64
DiabetesPedigreeFunction    int64
Age                         int64
Outcome                     int64
dtype: object
```

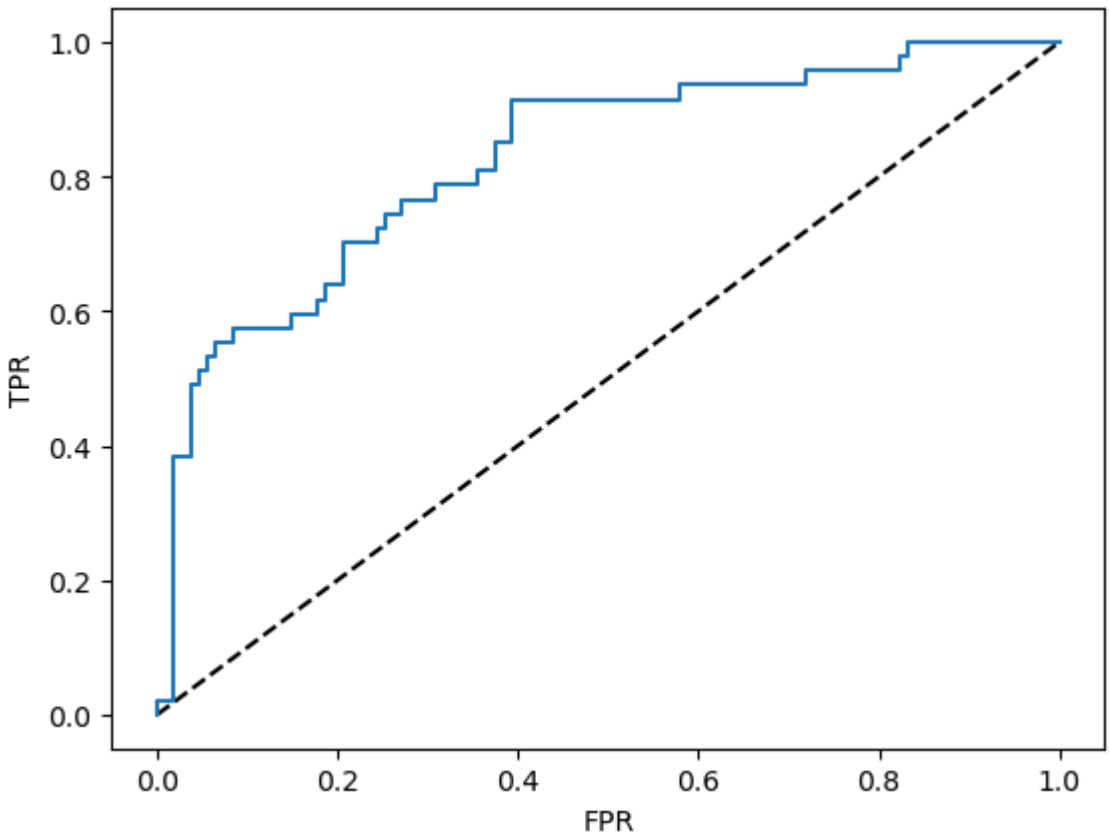
In [55]:
```
1 roc = log.predict_proba(X_train)[:,1]
2 fpr, tpr, threshold = metrics.roc_curve(Y_train, roc)
3 plt.plot([0,1], [0,1],'k--')
4 plt.plot(fpr, tpr, label='logistic')
5 plt.ylabel('TPR')
6 plt.xlabel('FPR')
7 plt.show()
```



In [56]:
```
metrics.roc_curve(Y_train, roc)
```

```
0.30982591, 0.3090899 , 0.30668413, 0.30468587, 0.30421869,
0.30291995, 0.30219937, 0.30151749, 0.29998059, 0.294387  ,
0.29138865, 0.28880267, 0.28796142, 0.28385816, 0.28318489,
0.28169224, 0.28150474, 0.27813976, 0.27811371, 0.27150223,
0.27032174, 0.26921408, 0.26887215, 0.2658929 , 0.2649347 ,
0.26434343, 0.26429818, 0.2639778 , 0.26325922, 0.26145747,
0.26130088, 0.24928567, 0.24686276, 0.24152607, 0.24096859,
0.24042768, 0.24028485, 0.23669244, 0.23547539, 0.23494064,
0.23377006, 0.23180623, 0.22876956, 0.220361  , 0.22035554,
0.21821769, 0.21708358, 0.21290087, 0.21109604, 0.20275716,
0.20216383, 0.18734255, 0.18668639, 0.18038444, 0.17982507,
0.17930759, 0.17878598, 0.17666633, 0.17602899, 0.17507593,
0.17285445, 0.16328505, 0.16240531, 0.16182712, 0.16031687,
0.15470247, 0.15449376, 0.15311126, 0.15299139, 0.15251718,
0.15238728, 0.14955705, 0.14854498, 0.14497683, 0.14317405,
0.13751355, 0.13683936, 0.13254681, 0.13237412, 0.11833758,
0.11658793, 0.11163329, 0.11090897, 0.0771268 , 0.07686514,
0.07094846, 0.06805919, 0.04756147, 0.04747654, 0.04314495,
0.04310462, 0.01880086]))
```

In [57]:
```
roc = log.predict_proba(X_test)[:,1]
fpr, tpr, threshold = metrics.roc_curve(Y_test, roc)
plt.plot([0,1], [0,1],'k--')
plt.plot(fpr, tpr, label='logistic')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
```



In [58]:
```
from sklearn.metrics import matthews_corrcoef

mcc = matthews_corrcoef(Y_test, pred_test)
print('MCC: ',mcc)
```

```
MCC:  0.4756970745542875
```

In [59]:
```
param_grid = {
    'penalty' :['l1','l2'],
    'C' : [0.1,0.5,1,5,10]
}
```

In [60]:
```
from sklearn.model_selection import GridSearchCV
```

In [61]:
```
grid = GridSearchCV(estimator=log, param_grid=param_grid, cv = 5)
```

In [62]:
```
grid.fit(X_train,Y_train)
```

```
C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:458: 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 (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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\Users\anike\anaconda3\anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

In [63]:
```
best_param = grid.best_params_
best_model = grid.best_estimator_
```

In [64]:
```
y_pred = best_model.predict(X_test)
```

In [65]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_sc
```

In [66]:
```
acc = accuracy_score(Y_test, y_pred)
pre = precision_score(Y_test, y_pred)
rec = recall_score(Y_test, y_pred)
f1 = f1_score(Y_test, y_pred)
roc_auc = roc_auc_score(Y_test, y_pred)
```

In [67]:
```
print('Best Param: ', best_param)
print('Accuracy: ', acc)
print('Recall: ', rec)
print('Precision: ', pre)
print('F1 Score: ', f1)
print('AUC-ROC: ', roc_auc)
```

```
Best Param:  {'C': 0.1, 'penalty': 'l2'}
Accuracy:  0.7857142857142857
Recall:  0.574468085106383
Precision:  0.675
F1 Score:  0.6206896551724138
AUC-ROC:  0.7264863790017896
```

In [ ]:
```

```