PIMA Indian Diabetes Dataset for Women above the age of 21

Applying Statistics and Probability concepts using Python programming to PIMA Indian Diabetes Dataset for Women above the age of 21.By using the concepts like data visualization, data cleaning, transformation, descriptive statistics and inferential statistics various insights and inferences are drawn and how data related to diabetes can be leveraged to predict if a person has diabetes or not.

Reading the data

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
df = pd.read_csv("diabetes.csv")
```

In [6]:
```
df.head()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

Using pandas head() method top 5 rows of our data frame are displayed giving a picture of what attributes our data set contains and what values are entered in different columns.

In [7]:
```
# in knowing the number of rows and columns
df.shape
```

Out[7]:
```
(768, 9)
```

Grouping the dataset on the basis of outcome

```
In [8]: df.groupby("Outcome").size()
```

```
Out[8]: Outcome
        0    500
        1    268
        dtype: int64
```

The count of number of positive result =268 and negative result =500

Checking for null values

```
In [9]: df.isna().any()
```

```
Out[9]: Pregnancies                 False
        Glucose                     False
        BloodPressure               False
        SkinThickness               False
        Insulin                     False
        BMI                         False
        DiabetesPedigreeFunction    False
        Age                         False
        Outcome                     False
        dtype: bool
```

As False is displayed for ever column which means no null values are present in our dataset.

Cleaning the dataset

Detecting inaccurate parts of data and then replacing them. The dataset is cleaned for all the columns having unnecesary 0 values which are invalid in some parameters and are replaced first by nan values followed by the mean of the columns. The dataset is then described

In [10]:
```
dataset_nozeros = df.copy()
zero_fields = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
df[zero_fields] = df[zero_fields].replace(0, np.nan)
df[zero_fields] = df[zero_fields].fillna(dataset_nozeros.mean())
df.describe()
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.681605	72.254807	26.606479	118.660163	32.450805	
std	3.369578	30.436016	12.115932	9.631241	93.080358	6.875374	
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	
25%	1.000000	99.750000	64.000000	20.536458	79.799479	27.500000	
50%	3.000000	117.000000	72.000000	23.000000	79.799479	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

Getting the datatypes for columns

In [11]:
```
df.dtypes
```

Out[11]:
```
Pregnancies                 int64
Glucose                   float64
BloodPressure             float64
SkinThickness             float64
Insulin                   float64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

All the columns contain numeric datatype.

Inserting Result Column in the dataset

We are adding Categorical data in order to extend the limit so that better usage of graphs can take place.

Now in order to make clear notation about the results of the tests conducted for the dataset, a column containing the final result in words declaring either the diabetes test result was 'positive' or 'negative' is added taking inferences from the Outcome column already present in the data set

In [12]:
```
Type_new = pd.Series([])
for i in range(len(df)):
    if df["Outcome"][i] == 0:
        Type_new[i]="negative"

    elif df["Outcome"][i] == 1:
        Type_new[i]="positive"

df.insert(8, "Result", Type_new)
```

0:Negative
1:Positive

Inserting BMI Column in the dataset

In order to get clear inferences from the BMI values measured for the patients in the dataset a seperate BMI Inference column has been added specifing whether the individual belongs to the obese , overweight , normal or underweight category taking and clubbing values from the BMI column already present

In [13]:
```
Type_new = pd.Series([])
for i in range(len(df)):
    if df["BMI"][i] < 18.5:
        Type_new[i]="underweight"

    elif df["BMI"][i] < 25:
        Type_new[i]="normal"

    elif df["BMI"][i] < 30:
        Type_new[i]="over weight"

    elif df["BMI"][i] > 29.9:
        Type_new[i]="obese"

df.insert(6,"BMI Inference" , Type_new)
```

BMI<18.5:underweight
BMI<25 :normal
BMI<30 :overweight
BMI>29.9: obese

Display dataset after insertion of columns ¶

Our modified dataset now has been cleaned for any invalid values and contains categorical values of data interpretation too.

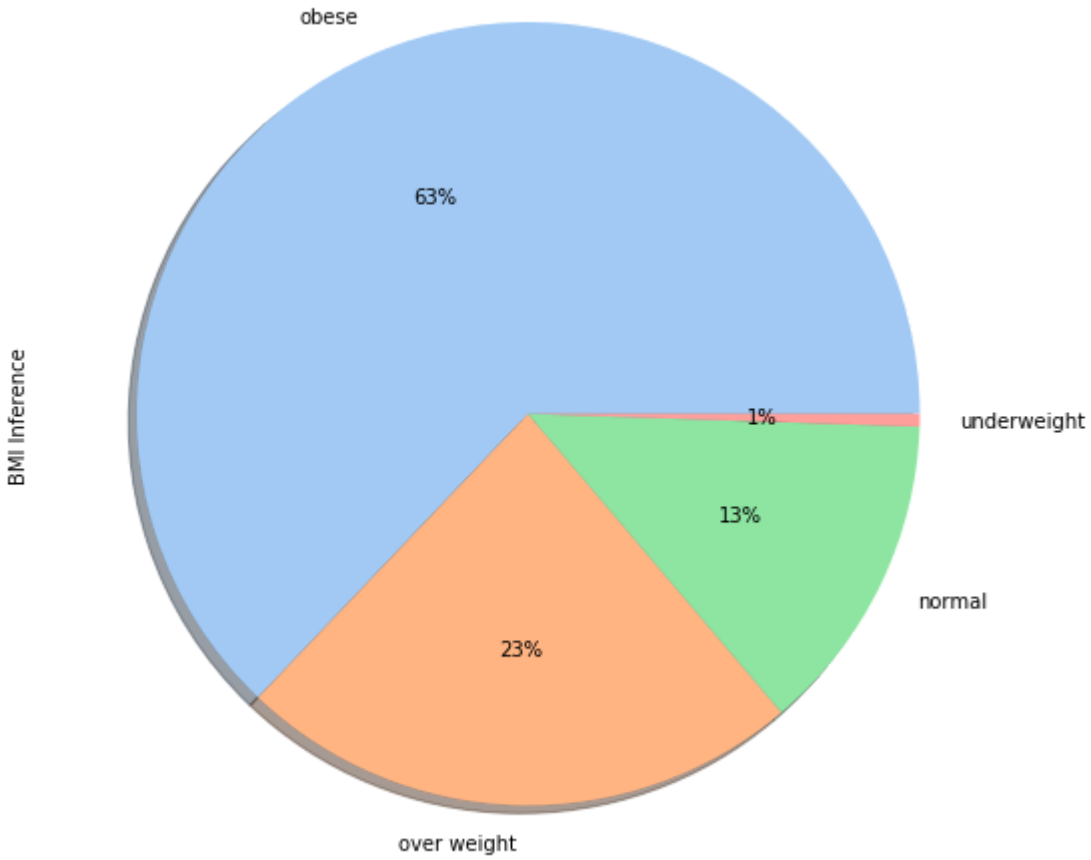
```
In [14]: df.head()
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	BMI Inference	DiabetesPe
0	6	148.0	72.0	35.000000	79.799479	33.6	obese	
1	1	85.0	66.0	29.000000	79.799479	26.6	over weight	
2	8	183.0	64.0	20.536458	79.799479	23.3	normal	
3	1	89.0	66.0	23.000000	94.000000	28.1	over weight	
4	0	137.0	40.0	35.000000	168.000000	43.1	obese	

Pie plot for BMI

In [15]:
```
sns.set_palette("pastel")
bmi_inference = df["BMI Inference"].value_counts()
bmi_inference.plot.pie(y="BMI Inference" , autopct='%1.0f%%', shadow = True ,
figsize=(9,9));
```



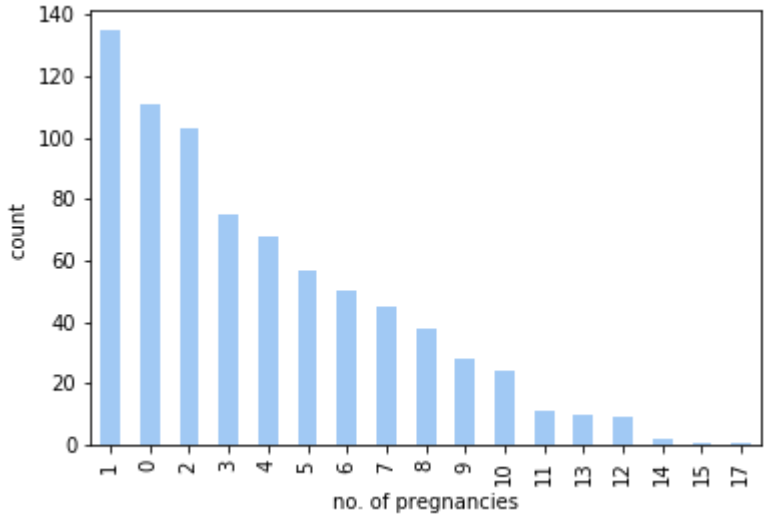
Pie chart are usually used to show the part-whole relationship.

Hence a pie plot has been constructed to depict the percentage of women belonging to each BMI category giving a clear indication that the most percentage i.e. 63% were belonging to the obese category followed by overweight category while the underweight category still prevailed behind at 1%.Whereas Normal and Overweight are 13% and 23% respectively. It means a good part of people are obese.

Count corresponding to pregnancies

In [16]:
```
pregnancies = df["Pregnancies"].value_counts()
pregnancies.plot.bar();
plt.xlabel("no. of pregnancies")
plt.ylabel("count ")
```

Out[16]: Text(0, 0.5, 'count ')

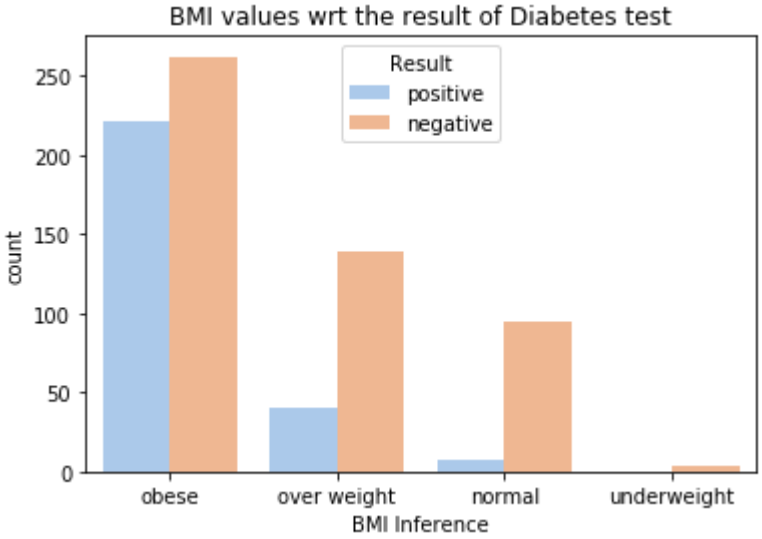


Bar graph are useful for comparison.

The bar graph shown above helps us to infer that the dataset contains the most number women with less number of pregnancies 1,0,2 while the higher number pregnancies are very less among women.

Countplot for BMI wrt to the test results

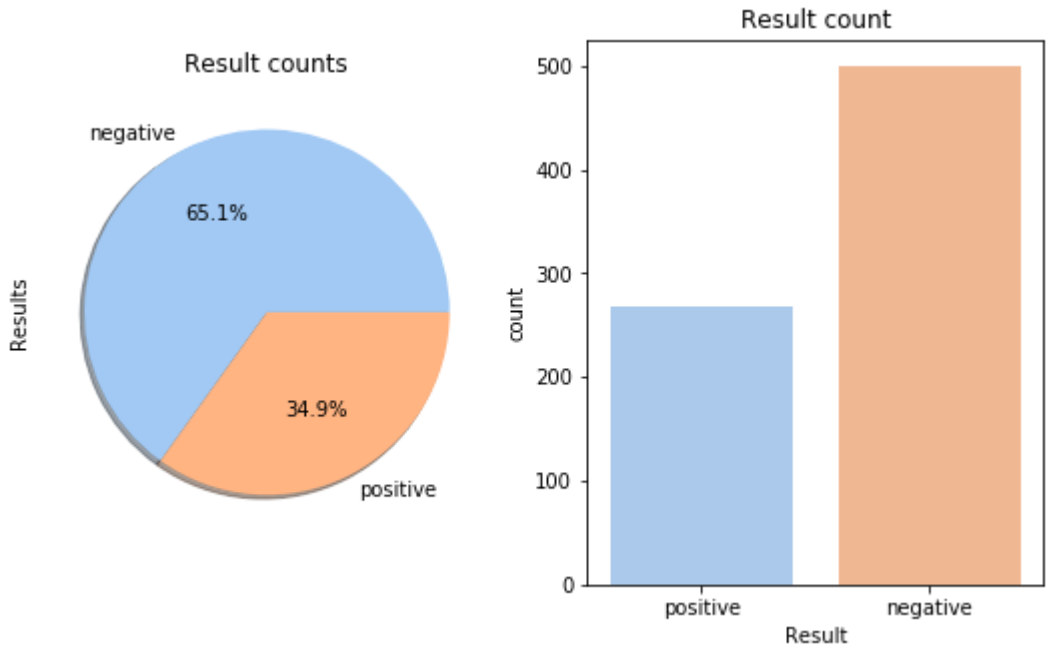
In [17]:
```
sns.countplot(df["BMI Inference"],hue=df["Result"])
plt.title('BMI values wrt the result of Diabetes test')
plt.show()
```



Count plot for the BMI Inferences helps us know exactly that test results are positive for more obese people while others are mostly negative. on the other hand the negative bar is raised higher in each case as the number of people tested negative is a little less than number of people tested positive.
It clearly means that obese women have higher chance of being diabetes positive as compared to the rest BMI categories.

Analysing the Result

In [18]:
```
sns.set_palette("pastel")

f,ax=plt.subplots(1,2,figsize=(9,5))
df['Result'].value_counts().plot.pie(autopct='%1.1f%%',ax=ax[0],shadow=True )
ax[0].set_title('Result counts')
ax[0].set_ylabel('Results')
sns.countplot('Result',data=df,ax=ax[1])
ax[1].set_title('Result count')
plt.show()
```



The result analysis is shown using 2 differnt kinds of plot , pie plot and count plot Both clearly shows that out of 768 women majority of women are diagonised diabetes negative and minority of them are diabetes positive.

Age analysis for the dataset

In [19]:
```
((df.Age >= 20) & (df.Age < 30)).sum()
```

Out[19]: 396

In [20]:
```
((df.Age >= 30) & (df.Age < 40)).sum()
```

Out[20]: 165

In [21]:
```
((df.Age >= 40) & (df.Age < 50)).sum()
```

Out[21]: 118

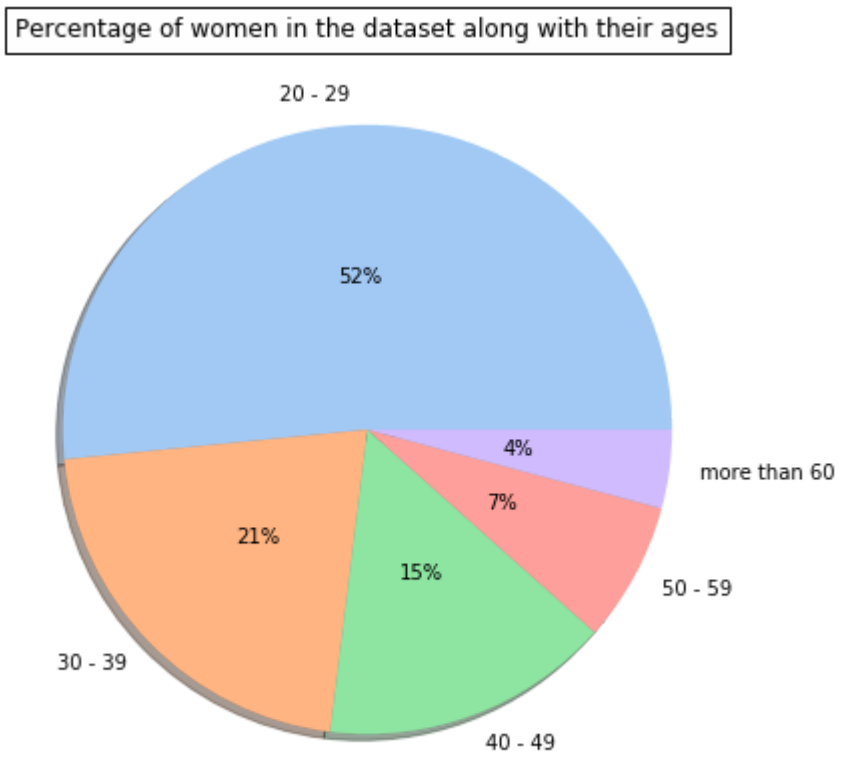
In [179]:
```
((df.Age >= 50) & (df.Age < 60)).sum()
```

Out[179]: 57

In [22]:
```
(df.Age >= 60).sum()
```

Out[22]: 32

In [23]:
```
slices=[
((df.Age >= 20) & (df.Age < 30)).sum(),
((df.Age >= 30) & (df.Age < 40)).sum(),
((df.Age >= 40) & (df.Age < 50)).sum(),
((df.Age >= 50) & (df.Age < 60)).sum(),
(df.Age >= 60).sum()
]
labels = ['20 - 29' , '30 - 39' , '40 - 49' , '50 - 59' , 'more than 60 ']
sns.set_palette("pastel")
plt.pie(slices, labels=labels, autopct='%1.0f%%', pctdistance=.5, labeldistanc
e=1.1,shadow=True)
fig = plt.gcf()
plt.title("Percentage of women in the dataset along with their ages", bbox={'f
acecolor':'2','pad':5})

fig.set_size_inches(7,7)
plt.show()
```



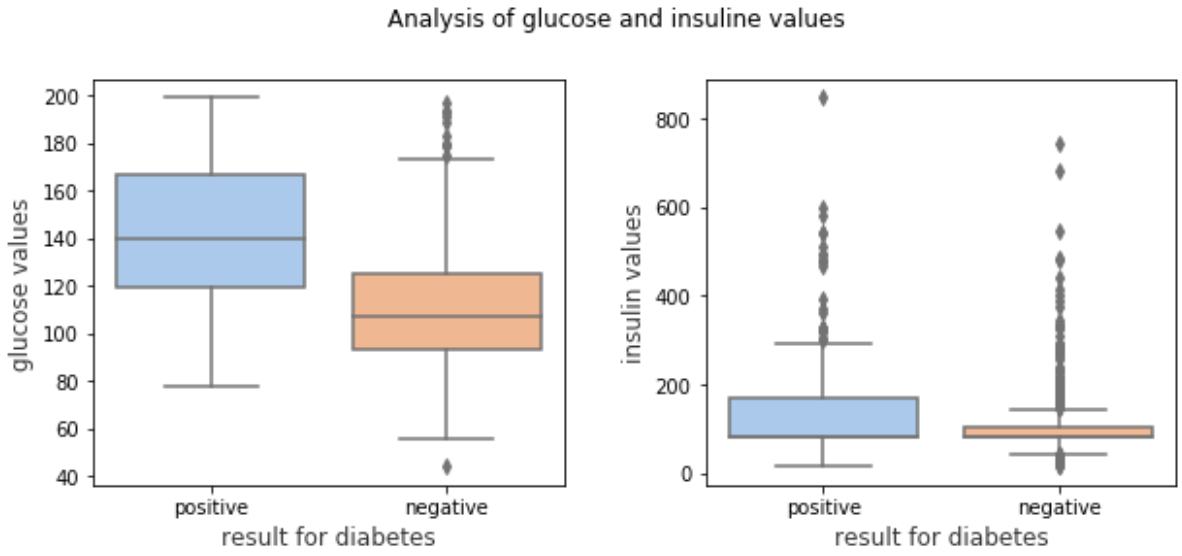
The dataset first has been converted to certain intervals which include the sum the actual values of ages present in the intervals . Then the pie plot commands has been used in order the infer that the dataset contains the most number of women from the age group 20 - 29 while less but still 4% are present having ages more than 60.
The data set we are working on consist of women as young adult in large number.

Inferential Statistics

Box plot for Result wrt to Glucose and Insulin

In [24]:
```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
f.suptitle(" Analysis of glucose and insuline values ")
f.subplots_adjust(top=0.85, wspace=0.3)

sns.boxplot(x="Result", y="Glucose",
            data=df, ax=ax1)
ax1.set_xlabel("result for diabetes",size = 12,alpha=0.8)
ax1.set_ylabel("glucose values",size = 12,alpha=0.8)


sns.boxplot(x="Result", y="Insulin", data=df, ax=ax2)
ax2.set_xlabel("result for diabetes",size = 12,alpha=0.8)
ax2.set_ylabel("insulin values",size = 12,alpha=0.8)
```

Out[24]:
```
Text(0, 0.5, 'insulin values')
```



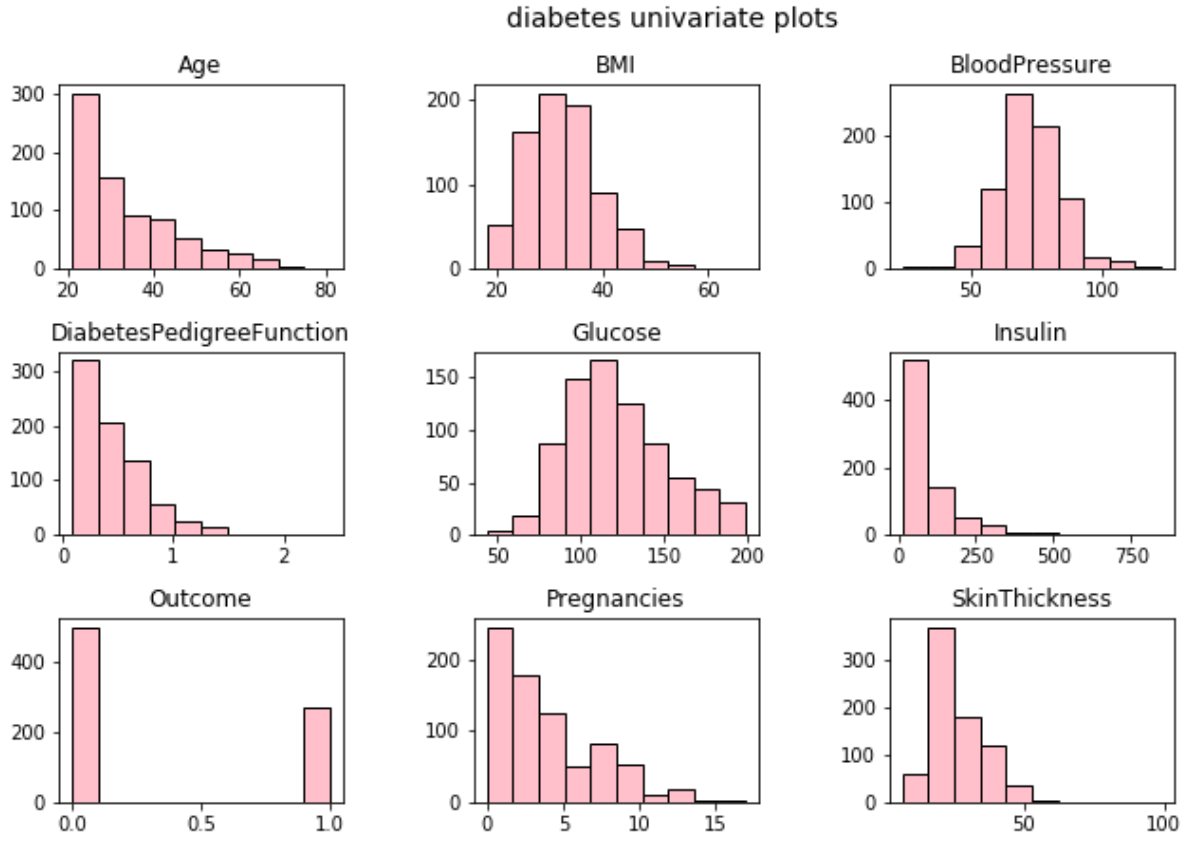
We have created 2 box subplots corresponding to the glucose and insulin values and the Results of the tests.

From the first subplot we can infer that the median values for positive result lies around 140 and the same for negative lies around 110. Also the positive values don't have any outliers but the negative box plot shows the presence of outliers with maximum and minimum values for glucose less the ones with a positve result.

From the second subplot we can infer that the insulin values certainly has a lot of outliers present making it difficult to analyse the data through box plots . What we know is that inter quartile range for negative test results is still a lot less than the positive test results.

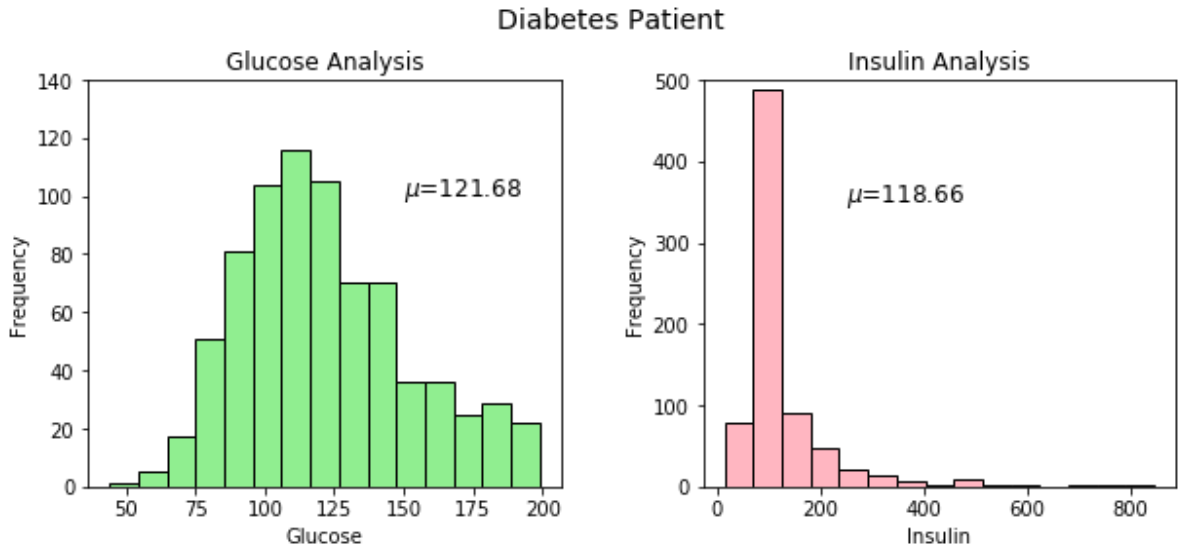
Univariate Analysis

In [25]:
```
df.hist(bins=10 , color='pink' , edgecolor = 'black' ,linewidth = 1.0,grid = F
alse)
plt.tight_layout(rect =(0,0,1.5,1.5))
d1 = plt.suptitle('diabetes univariate plots' , x=0.85 , y=1.55 ,fontsize=14)
```



The Univariate plot shows us the distribution of each variable and summarizes its distribution The analysis has allowed us to analyse each and every column with numerical data present in our dataset. We get to know the count for a particular range of values for the particular column irrespective of whether the results for that woman are positive or negative.

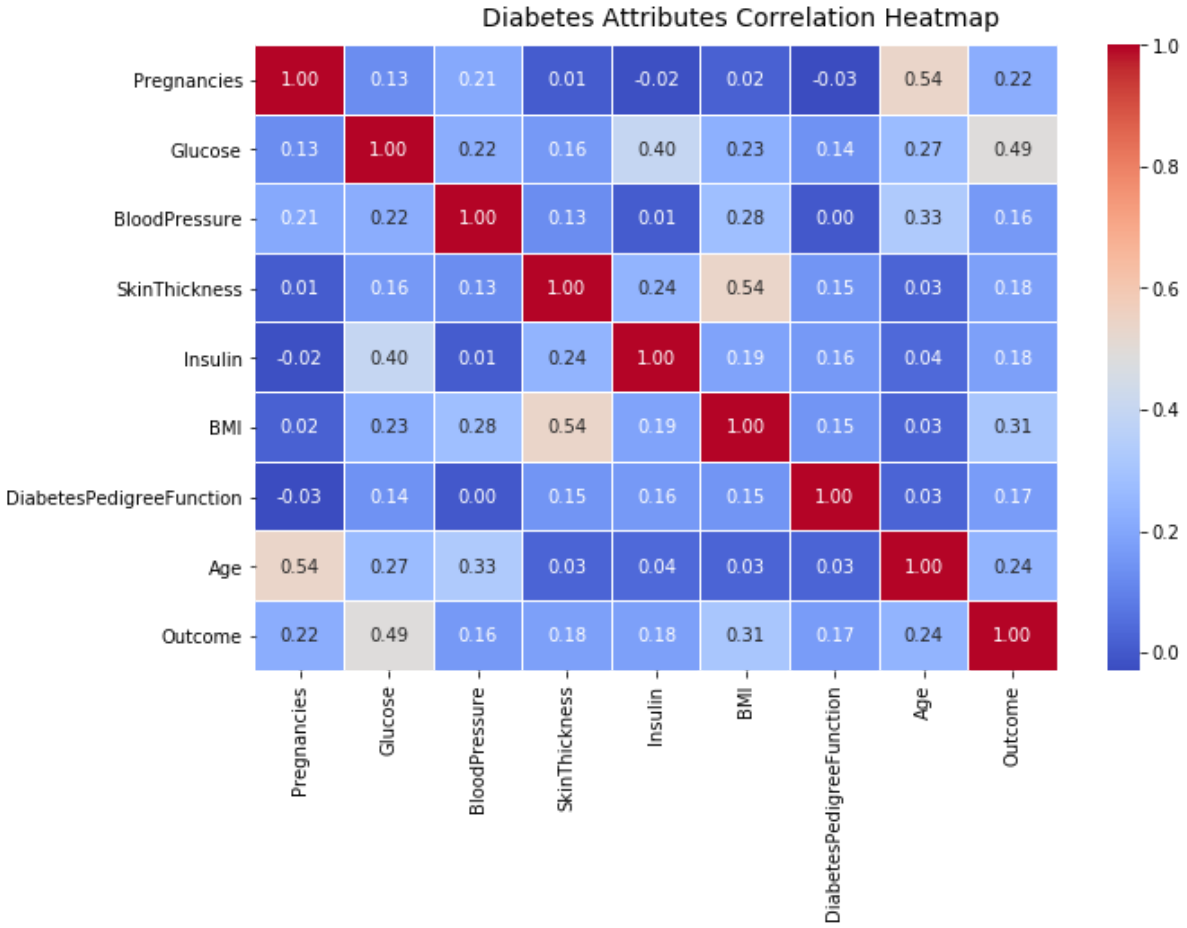
In [26]:
```
fig = plt.figure(figsize = (10,4))
title = fig.suptitle("Diabetes Patient ", fontsize=14)
fig.subplots_adjust(top=0.85, wspace=0.3)

ax1 = fig.add_subplot(1,2, 1)
ax1.set_title("Glucose Analysis")
ax1.set_xlabel("Glucose")
ax1.set_ylabel("Frequency")
ax1.set_ylim([0, 140])
ax1.text(150, 100, r'$\mu$='+str(round(df['Glucose'].mean(),2)), fontsize=12)
g_freq, g_bins, g_patches = ax1.hist(df['Glucose'], color='lightgreen', bins=1
5,edgecolor='black', linewidth=1)

ax2 = fig.add_subplot(1,2, 2)
ax2.set_title("Insulin Analysis")
ax2.set_xlabel("Insulin")
ax2.set_ylabel("Frequency")
ax2.set_ylim([0, 500])
ax2.text(250, 350, r'$\mu$='+str(round(df['Insulin'].mean(),2)), fontsize=12)
i_freq, i_bins, i_patches = ax2.hist(df['Insulin'], color='lightpink', bins=15
,edgecolor='black', linewidth=1)
```



These univariate graphs help us to analyse the glucose and insulin levels of our dataset patients wrt the frequency / count for the number of pateints with that value. We also see the respective means calculated and displayed inside the canvas of our graph.

Multivariate Analysis

In [27]:
```
f, ax = plt.subplots(figsize=(10, 6))
corr = df.corr()
hm = sns.heatmap(round(corr,2), annot=True, ax=ax,fmt='.2f', cmap="coolwarm",
            linewidths=.05)
f.subplots_adjust(top=0.93)
t= f.suptitle('Diabetes Attributes Correlation Heatmap', fontsize=14)
```

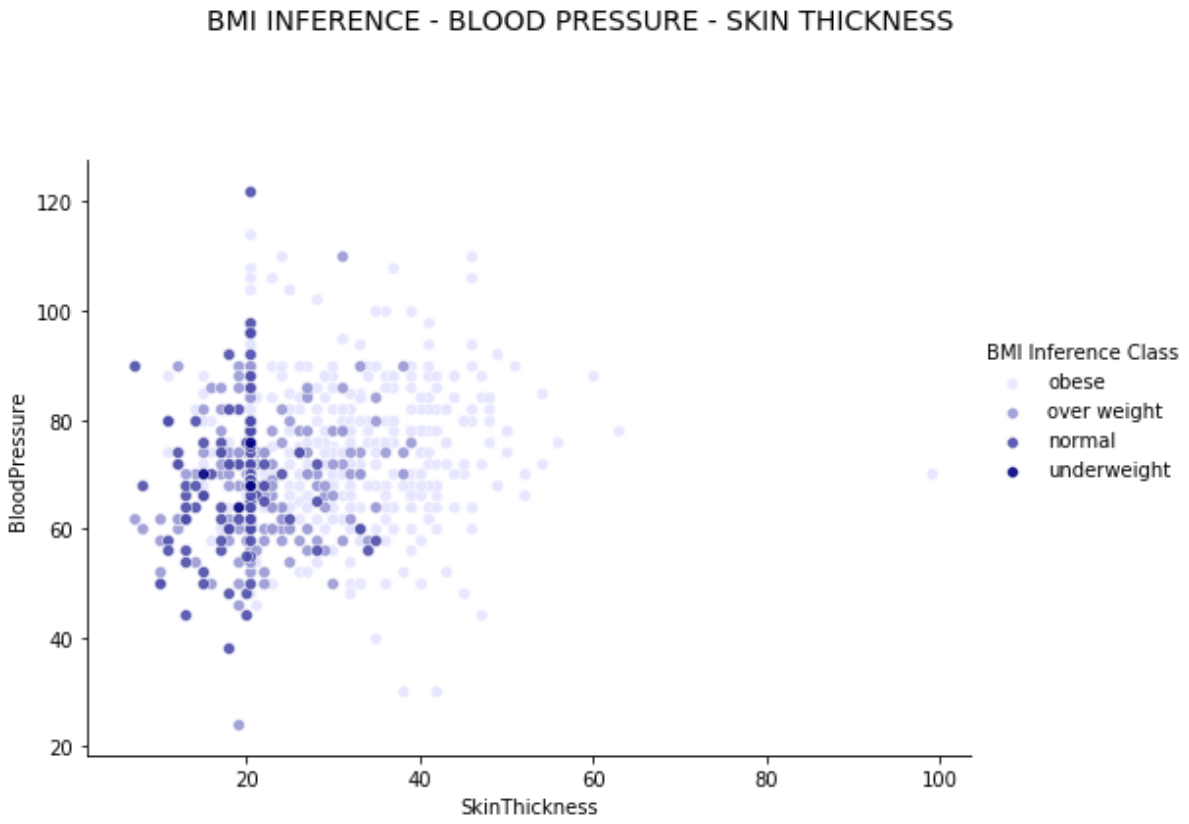


The Multivariate Analysis Heatmap helps us to know the correlation between each attribute of the dataset with every other attribute. It helps us in understanding the clear correlation values.

FacetGrid for Skin Thickness and BMI

In [28]:
```
g = sns.FacetGrid(df ,hue="BMI Inference",aspect=1.2, size=6, palette=sns.light_palette('navy', 4))
g.map(plt.scatter, "SkinThickness", "BloodPressure", alpha=0.9, edgecolor='white', linewidth=0.5)
fig = g.fig
fig.subplots_adjust(top=0.8, wspace=0.3)
fig.suptitle('BMI INFERENCE - BLOOD PRESSURE - SKIN THICKNESS', fontsize=14)
l = g.add_legend(title='BMI Inference Class')
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\seaborn\axisgrid.py:230: UserWarning: The `size` paramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



The scatter graph between skin thickness and blood pressure helps us in understanding these parameters better wrt the BMI inferences drawn. We see that the people in the category under normal and underweight have a skin thickness less than 20 but the ones obese and overweight have comparitively a thicker skin.

On the other hand in comparison to Blood pressure this value doesnot depend on BMI inference much as all category women have their vlood pressures lying in all the ranges.

It means a symptom of being diabtes positive is thicker skin.

Maximum and Minimum Glucose Value

In [29]:
```
print("Glucose values")
print("Max : ",max(df['Glucose'].unique()))
print('Min : ',min(df['Glucose'].unique()))
```

```
Glucose values
Max :  199.0
Min :  44.0
```

Maximum and Minimum Insulin Value

In [188]:
```
print("Insulin values")
print("Max : ",max(df['Insulin'].unique()))
print('Min : ',min(df['Insulin'].unique()))
```

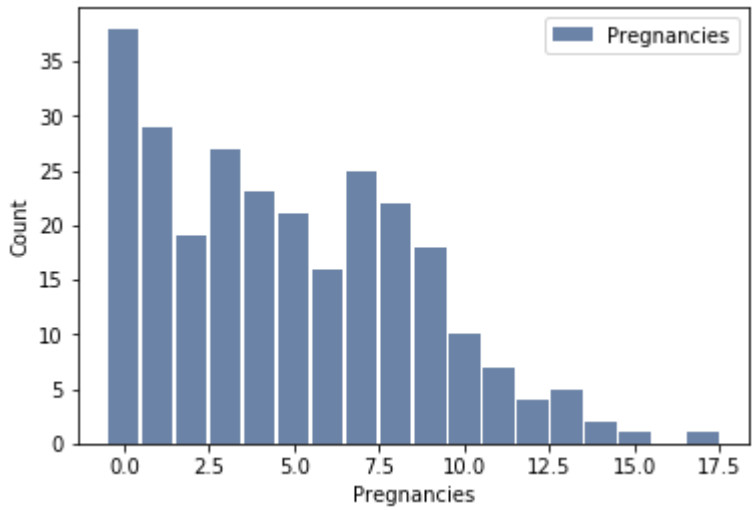
```
Insulin values
Max :  846.0
Min :  14.0
```

Next up we import some files to be used in futher commands in plots.These files are containig certain functions.

In [30]:
```
%matplotlib inline

import numpy as np

import nsfg
import first
import thinkstats2
import thinkplot
import analytic

import matplotlib.pyplot as plt

from IPython.core import page
page.page = print
```

In [31]:
```
positive = df[df.Outcome==1]
```

In [32]:
```
hist = thinkstats2.Hist(positive.Pregnancies, label='Pregnancies')
thinkplot.Hist(hist)
thinkplot.Config(xlabel='Pregnancies', ylabel='Count')
```

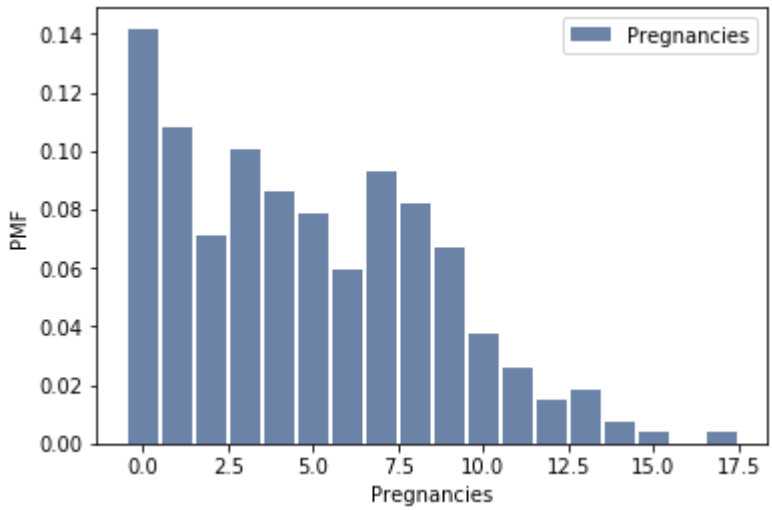


We create a histogram for the number of pregnancies wrt the count present in our dataset.
Histogram has integer value on y-axis.

Probability plot for Pregnancies

In [33]:
```
n = hist.Total()
pmf = hist.Copy()
for x, freq in hist.Items():
    pmf[x] = freq / n
```

In [34]:
```
thinkplot.Hist(pmf)
thinkplot.Config(xlabel='Pregnancies', ylabel='PMF')
```



This plot now has the plot of Number of Pregnancies wrt the PMF values for each pregnancy.
The shape of the graph remains same but the values on y-axis have changed to float value as Pmf maps random variable to its frequency.

In [35]:
```
d1 = df['Outcome']
```

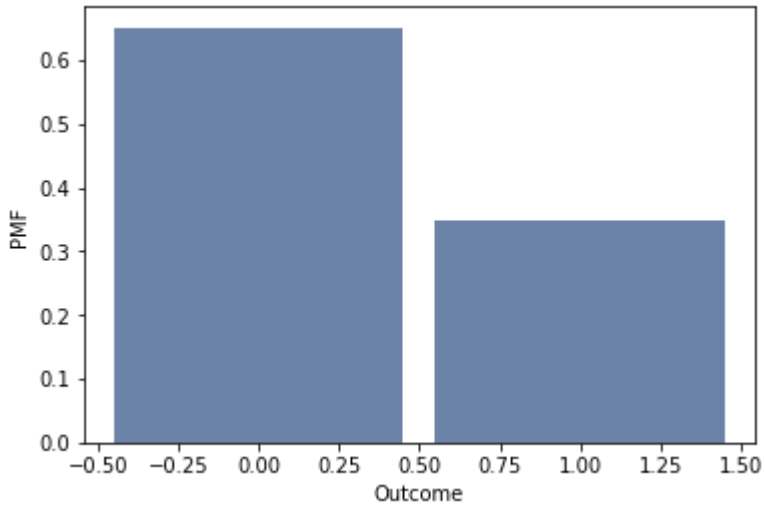
In [36]:
```
pmf = thinkstats2.Pmf(d1)
pmf
```

Out[36]:
```
Pmf({0: 0.6510416666666666, 1: 0.3489583333333333})
```

Calculatting the probability of occurence of 0 and 1 respectively using the pmf theory and plotting the same in the graph below.

Probability plot for outcome

In [37]:
```
thinkplot.Hist(pmf)
thinkplot.Config(xlabel='Outcome', ylabel='PMF')
```



Probability of positive outcome 0.35 and that of negative is 0.65 making it a total of 1

In [197]:
```
p = df[df["Outcome"] == 1][['Pregnancies','Outcome']]
p.head()
```

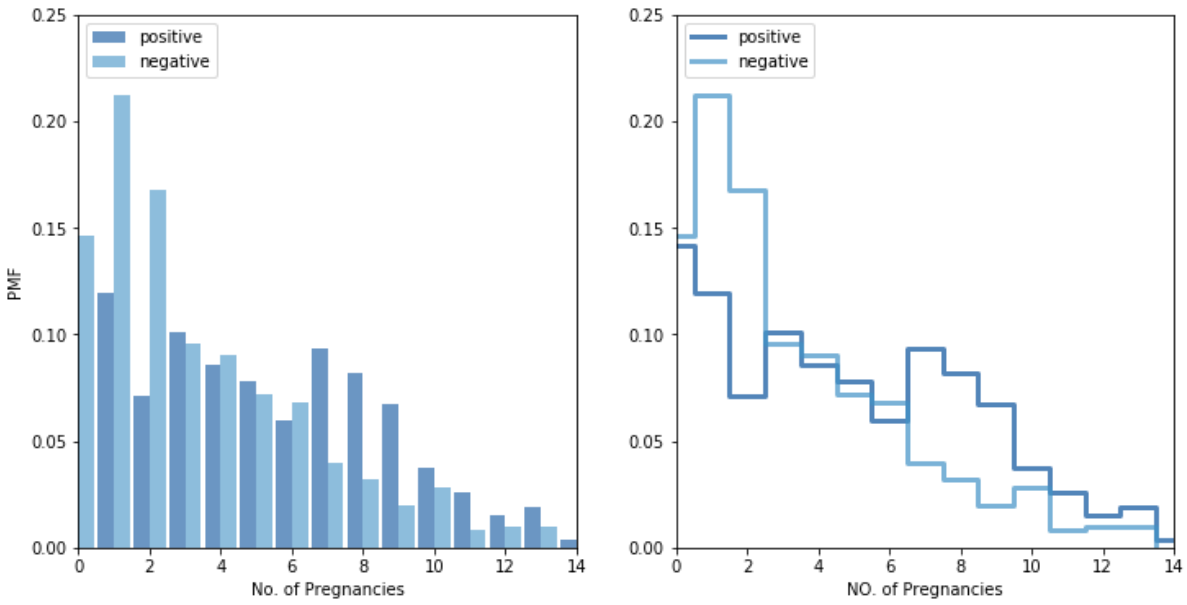
Out[197]:

	Pregnancies	Outcome
0	6	1
2	8	1
4	0	1
6	3	1
8	2	1

In [198]:
```
n = df[df["Outcome"] == 0][['Pregnancies','Outcome']]
```

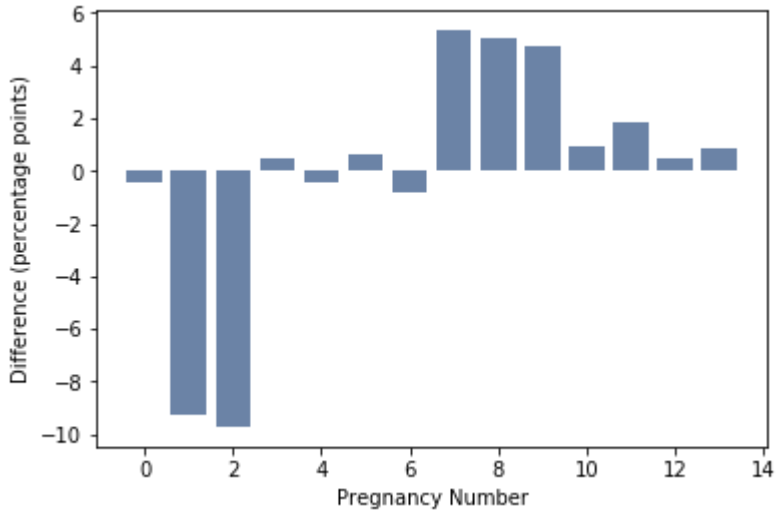
In [199]:
```
positive = thinkstats2.Pmf(p.Pregnancies, label='positive')
negative = thinkstats2.Pmf(n.Pregnancies, label='negative')
```

In [200]:
```
width=0.45
axis = [0, 14, 0, 0.25]
thinkplot.PrePlot(2, cols=2)
thinkplot.Hist(positive, align='right', width=width)
thinkplot.Hist(negative, align='left', width=width)
thinkplot.Config(xlabel='No. of Pregnancies', ylabel='PMF', axis=axis)

thinkplot.PrePlot(2) # resets the color generator
thinkplot.SubPlot(2)
thinkplot.Pmfs([positive, negative])
thinkplot.Config(xlabel='NO. of Pregnancies', axis=axis)
```



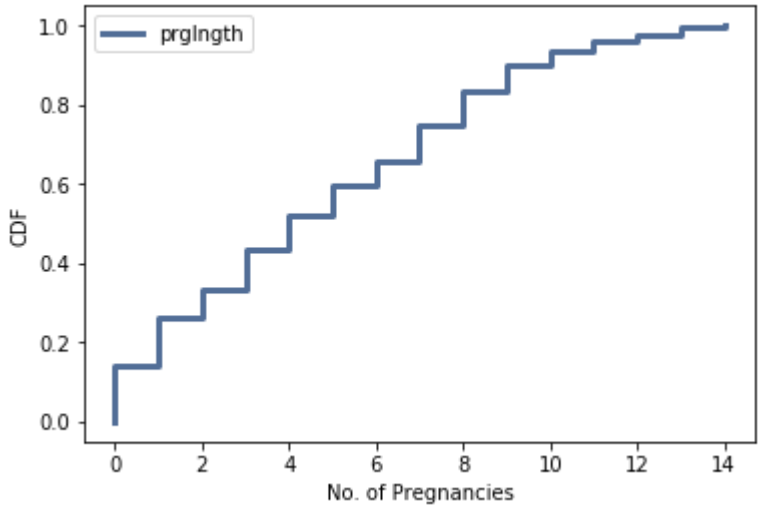
Next up in the variables p and n we store the Pregnancies with the test results as positive and negative. Then we calculated their pmfs seperately and plot these 2 different graphs which help us in understanding that the probability of result being positive in case of 0-8 pregnancies is almost around the same range while that being negative is greater for less number of pregnancies and less for the range 3-7.

In [201]:
```
weeks = range(0,14)
diffs = []
for week in weeks:
    p1 = positive.Prob(week)
    p2 = negative.Prob(week)
    diff = 100 * (p1 - p2)
    diffs.append(diff)

thinkplot.Bar(weeks, diffs)
thinkplot.Config(xlabel='Pregnancy Number', ylabel='Difference (percentage poi
nts)')
```

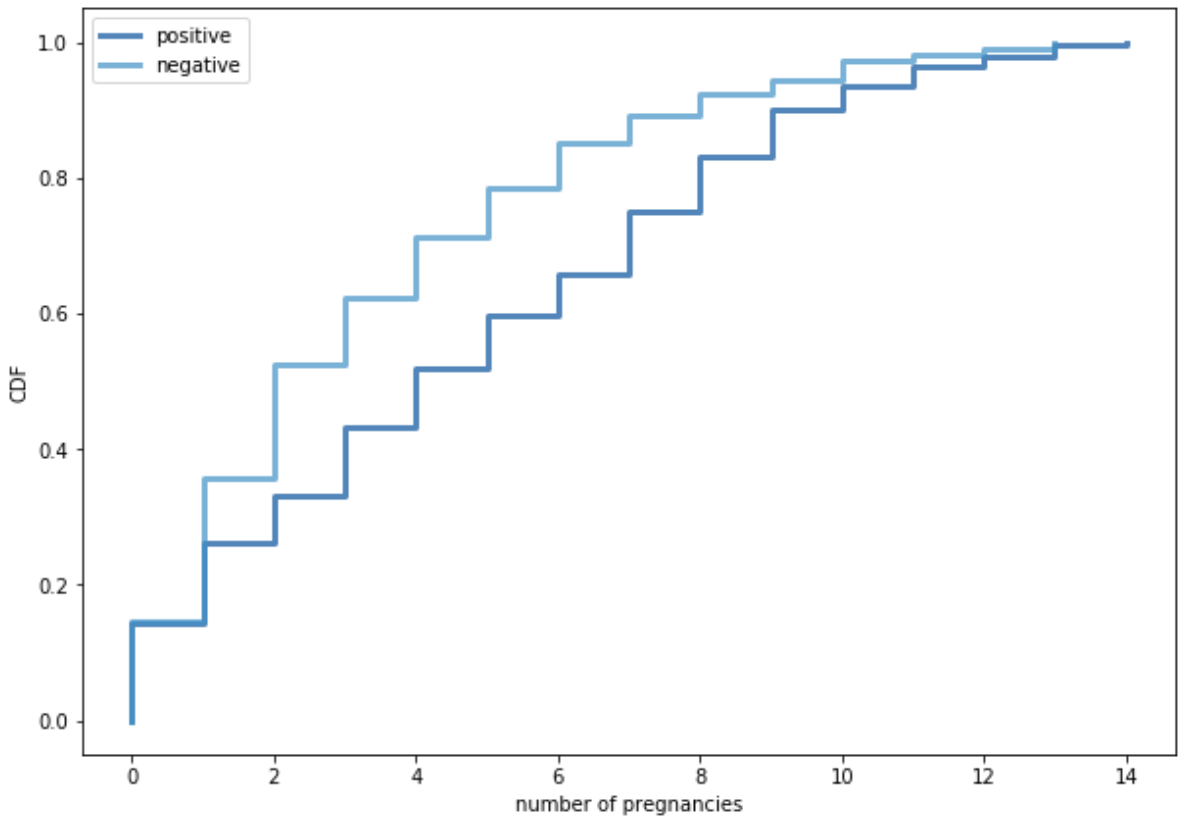


Cumulative probability plot for pregnancies

In [202]:
```
cdf = thinkstats2.Cdf(positive, label='prglngth')
thinkplot.Cdf(cdf)
thinkplot.Config(xlabel='No. of Pregnancies', ylabel='CDF', loc='upper left')
```



In [203]:
```
plt.figure(figsize=(10,7))

positive = thinkstats2.Cdf(positive, label='positive')
negative = thinkstats2.Cdf(negative, label='negative')

thinkplot.PrePlot(2)
thinkplot.Cdfs([positive,negative])
thinkplot.Config(xlabel='number of pregnancies', ylabel='CDF')
```



The first graph shows us the plot of positive result number of pregnancies wrt the CDF values while the above one shows both positive and negative plots wrt their CDF values. CDF values at each point tells us the probability of the number of pregnancies wrt that value and less than that values present in the dataset , i.e. if the number is 5 then CDF value tells us the probability for the number of pregnancy being less than and equal to 5.

In [204]:
```
p = df[df["Outcome"] == 1][['Glucose','Outcome']]

glucose = p.Glucose
glucose_cdf = thinkstats2.Cdf(glucose, label='glucose')
median = glucose_cdf.Percentile(50)
median
```

Out[204]:
```
140.0
```

50th percentile is also known as Median which has been calculated for the glucose levels.

In [205]:
```
iqr = (glucose_cdf.Percentile(25),glucose_cdf.Percentile(75))
iqr
```

Out[205]:
```
(119.0, 167.0)
```

The interquartile range lies between the 25th and 75th percentile which is the 50th percentile.

In [206]:
```
glucose_cdf.Prob(120)
```

Out[206]:
```
0.26492537313432835
```

Probability of getting a glucose value near 120 and 170.

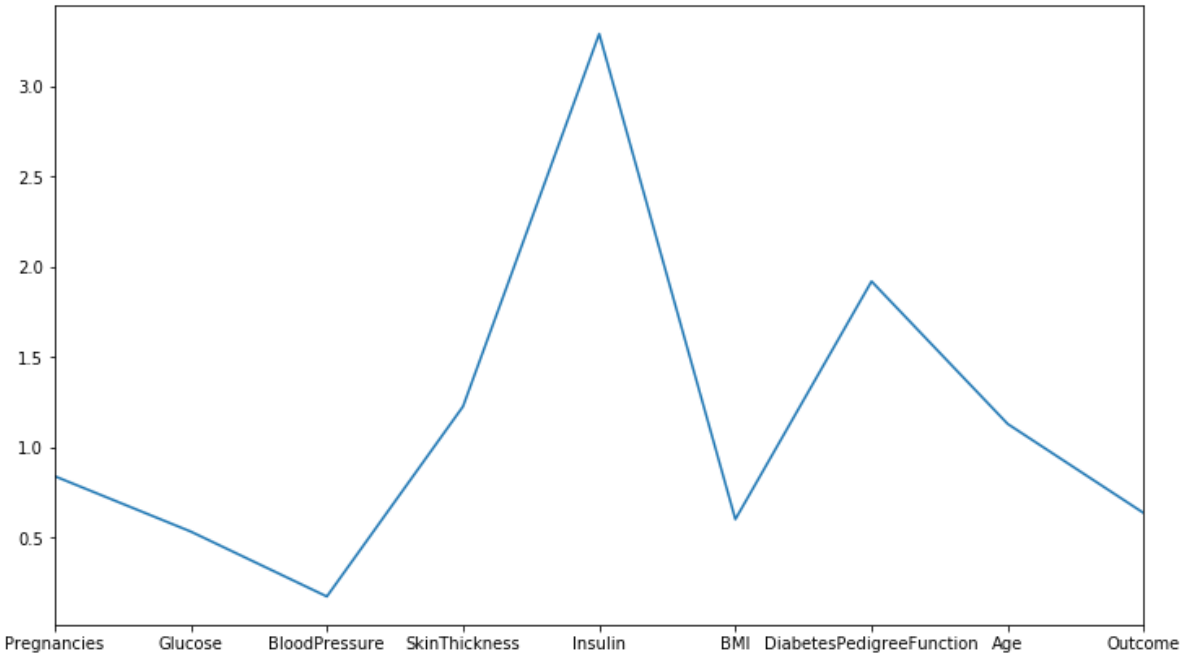
In [207]:
```
glucose_cdf.Prob(170)
```

Out[207]:
```
0.7835820895522388
```

Skewness

In [24]:
```
df1 = df.skew()
```

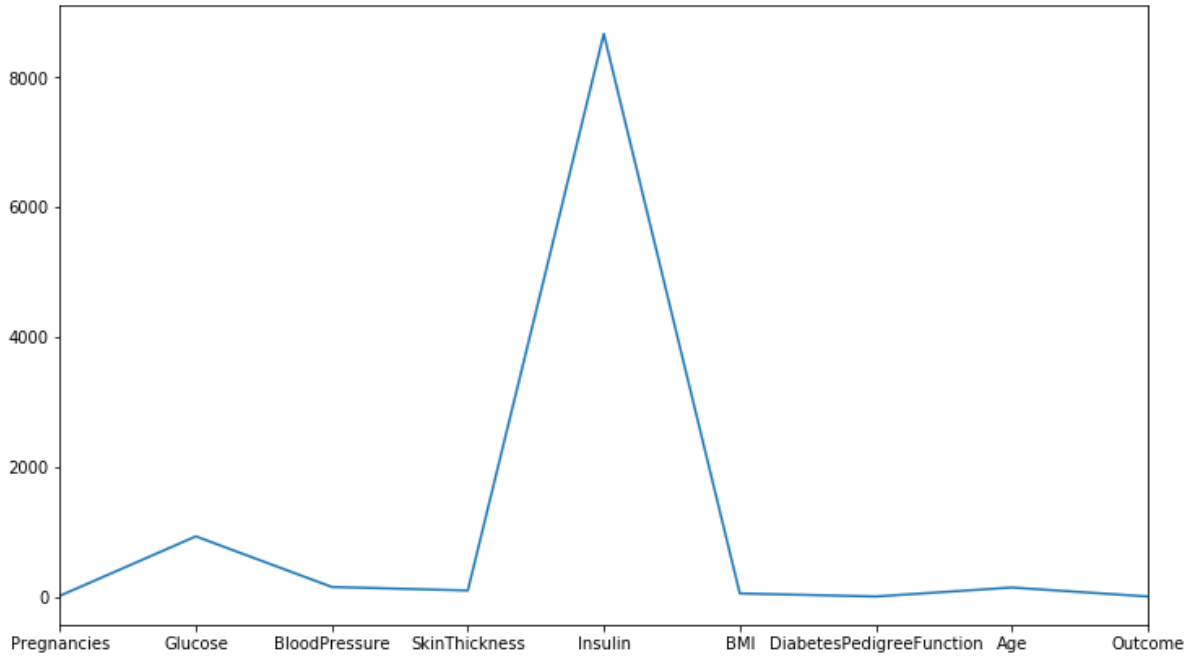
In [30]:
```
plt.figure(figsize=(12,7))
df1.plot();
```



As it is clearly evident from the graph that the value of skewness is greatest for Insulin which means that the highest number of outliers are present in Insulin Column of our data set with the value of Skewness even greater than 3 followed by DiabetesPedigreeFunction being the second highest.
Moreover the value of skewness id positive for all columns which means data in all cases are positively skewed.

Variance

In [31]:
```
df1 = df.var()
plt.figure(figsize=(12,7))
df1.plot();
```



The graph above shows that the value of variance is highest for Insulin which means that numbers are spread far from their mean value.Comparatively rest columns don't have as much high value of variance in comparison to Insulin column which means the numbers are not very far from their mean value they reside near mean only.

Relationship between Two Variables

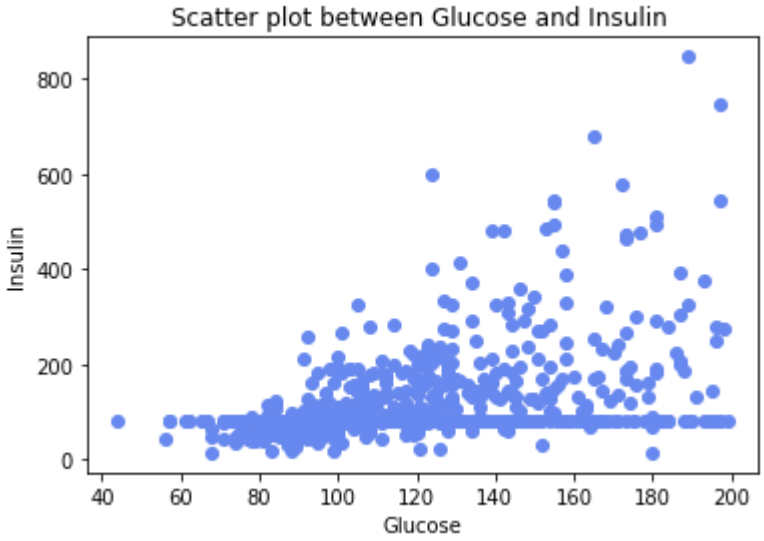
Scatter plots can be used to show correlation between two variable.

Scatter Plot between Glucose level and Insulin

As blood sugar level that is Glucose increases pancreas secretes more insulin.It shows positive correlation between the two variable as one increases so does the other.

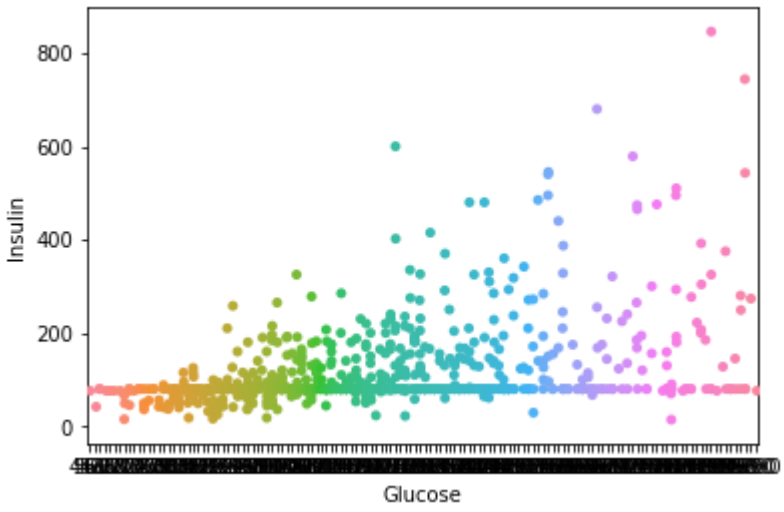
In [39]:
```
sns.set_palette("coolwarm")

plt.scatter(df.Glucose,df.Insulin);
plt.title('Scatter plot between Glucose and Insulin ')
plt.xlabel("Glucose")
plt.ylabel("Insulin")
```

Out[39]:
```
Text(0, 0.5, 'Insulin')
```



Since the graph shown above has many overlapping points we are using jitter to separate them so that they aren't plotted directly on top of each other

In [40]:
```
sns.stripplot(x='Glucose',y='Insulin',data=df,jitter=True);
```

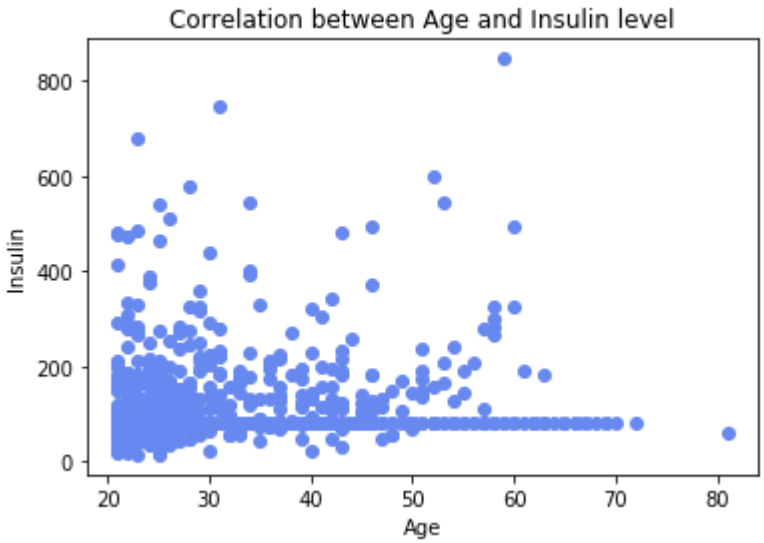


Scatter Plot between Age and Insulin level

There is no relationship between Age and Insulin level that is they are independent of one another.With increase in age decrease in pancreases secreting insulin has not been determined

In [41]:
```
plt.scatter(df.Age,df.Insulin)
plt.title('Correlation between Age and Insulin level')
plt.xlabel("Age")
plt.ylabel("Insulin")
```

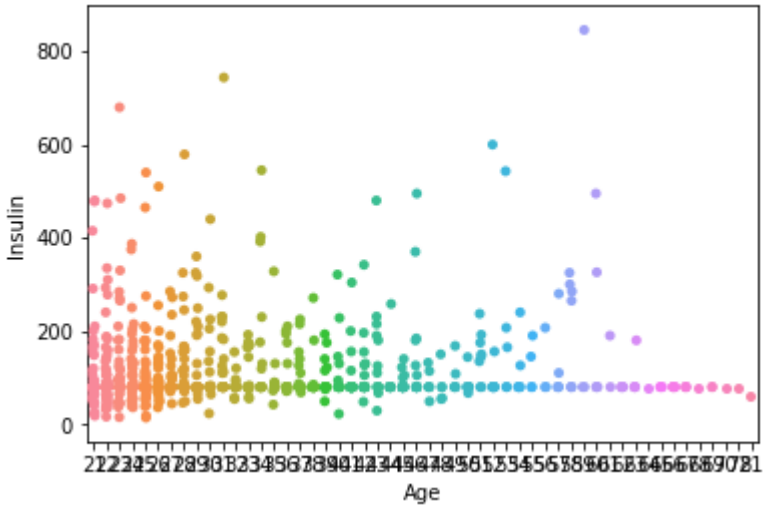
Out[41]:
```
Text(0, 0.5, 'Insulin')
```



Since the graph shown above has many overlapping points we are using jitter to separate them so that they aren't plotted directly on top of each other

In [42]:
```
sns.stripplot(x='Age',y='Insulin',data=df,jitter=True)
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22fd7def8d0>
```

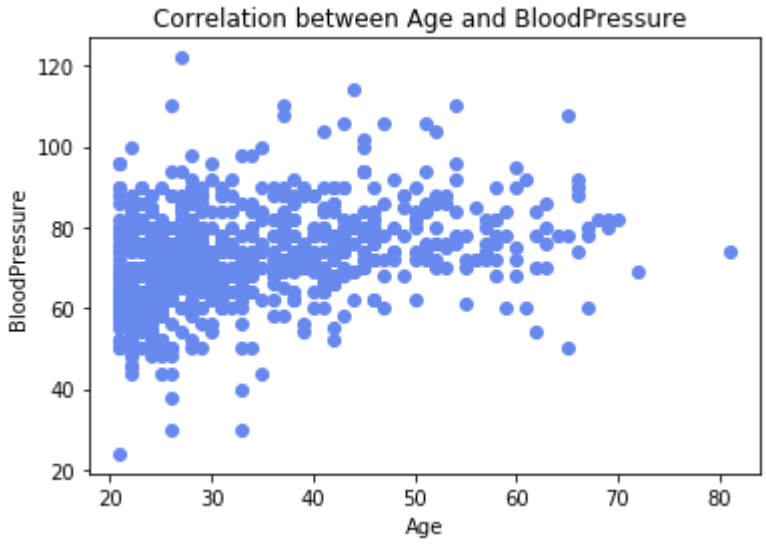


Scatter plot between Age and BloodPressure

In the youth population the blood pressure is spread over a larger range and as the age increases we notice that the range in which the blood pressure lies decreases a bit.

In [43]:
```
plt.scatter(df.Age,df.BloodPressure)
plt.title('Correlation between Age and BloodPressure ')
plt.xlabel("Age")
plt.ylabel("BloodPressure")
```

Out[43]:
```
Text(0, 0.5, 'BloodPressure')
```



Pearson's Correlation Matrix

In [315]:
```
df.corr(method='pearson')
```

Out[315]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.123305	0.212407	0.001973	-0.021267	0.00
Glucose	0.123305	1.000000	0.219666	0.160766	0.396597	0.23
BloodPressure	0.212407	0.219666	1.000000	0.134155	0.010926	0.28
SkinThickness	0.001973	0.160766	0.134155	1.000000	0.240361	0.53
Insulin	-0.021267	0.396597	0.010926	0.240361	1.000000	0.18
BMI	0.007402	0.231478	0.281231	0.535703	0.189856	1.00
DiabetesPedigreeFunction	-0.034610	0.137106	0.000371	0.154961	0.157806	0.15
Age	0.538996	0.266600	0.326740	0.026423	0.038652	0.02
Outcome	0.203665	0.492908	0.162986	0.175026	0.179185	0.31

The result is a matrix with self-correlations on the diagonal are always 1 and as most of the value of correlation coefficient are tending towards 0 so they have weak linear relationship.

In [2]:
```
Glucose = df['Glucose']
Insulin = df['Insulin']
```

In [314]:
```
np.corrcoef(Glucose,Insulin)
```

Out[314]:
```
array([[1.        , 0.39659667],
       [0.39659667, 1.        ]])
```

Spearman Correlation Matrix

In [316]:
```
df.corr(method='spearman')
```

Out[316]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.124661	0.192398	0.008441	-0.008065	-0.01
Glucose	0.124661	1.000000	0.245186	0.142468	0.414434	0.22
BloodPressure	0.192398	0.245186	1.000000	0.139616	0.007797	0.29
SkinThickness	0.008441	0.142468	0.139616	1.000000	0.277151	0.55
Insulin	-0.008065	0.414434	0.007797	0.277151	1.000000	0.22
BMI	-0.011001	0.225644	0.290916	0.557085	0.220655	1.00
DiabetesPedigreeFunction	-0.043544	0.090183	0.013213	0.131925	0.146543	0.13
Age	0.596729	0.281431	0.369219	0.075658	0.066463	0.12
Outcome	0.184803	0.481605	0.168555	0.177795	0.212215	0.30

Spearman corrleation is less sensitive than Pearsons correlation to strong outliers. As it is seen most of the values of correlation coefficient are tending to 0 which means they have weak linear relationship.

Hypothesis testing

We want to do hypothesis testing for checking whether the dataset has the probability of women with BMI value greater than 25 is 70%.

H0 : p=0.7
H1 : p>0.7

X takes number of people detected positive for the diabetes test,this means that we can use X as our test statistic. We take a sample of 200 data values. our test statistic is X ~ B(200 , 0.7)

In [210]:
```
# scikit-learn bootstrap
from sklearn.utils import resample
```

In [211]:
```
df1 = df['BMI']
```

In [212]:
```
df1 = pd.DataFrame(df1)
```

In [213]:
```
df1.head()
```

Out[213]:

	BMI
0	33.6
1	26.6
2	23.3
3	28.1
4	43.1

In [214]:
```
sample = df1.sample(n=200)
```

In [215]:
```
sample.head()
```

Out[215]:

	BMI
529	24.6
592	34.4
2	23.3
287	45.6
575	35.5

For the critical region we need a significance level and hence we decide upon a significance level of 5% which means α = 5%

We will be using the one tailed test with lower tail e.i. the critical region is at the lower end.

In [216]:
```
# if we chose 5th and 95th percentile, we are eliminating 5% data on left and
 5% data on right.
# Total we are eliminating 10% of the data. We need to eliminate 5% of the dat
a
# We need to use the range 0.025 to 0.975 to elimiate 2.5% data on left and 2.
5% data on right

confidence_interval = sample.quantile([0.05 - 0.025, 0.95 + 0.025])
```

In [217]:
```
confidence_interval
```

Out[217]:

	BMI
0.025	21.0975
0.975	45.6000

In [218]:
```
lower_interval = confidence_interval.iloc[0,0]
upper_interval = confidence_interval.iloc[1,0]

print(lower_interval, upper_interval)
```

```
21.0975 45.6
```

In [224]:
```
df1.mean().iloc[0]
```

Out[224]:
```
32.45080515543617
```

In [226]:
```
if df1.mean().iloc[0] >= lower_interval and df1.mean().iloc[0] <= upper_interv
al:
    print('The true mean {} is between the confidence interval of {} and {}'.f
ormat(df1.mean().iloc[0], confidence_interval.iloc[0,0], confidence_interval.i
loc[1,0]))
```

```
The true mean 32.45080515543617 is between the confidence interval of 21.0975
and 45.6
```

A p-value is the probability of getting a value up to and including the one in your sample in the direction of your critical region. It's a way of taking your sample and working out whether the result falls within the critical region for your hypothesis test. In other words, we use the p-value to say whether or not we can reject the null hypothesis.

For our hypothesis we will find the probability value for 140 people. so P(X < 140)

In [219]:
```
sample[sample['BMI'] > 25].count()
```

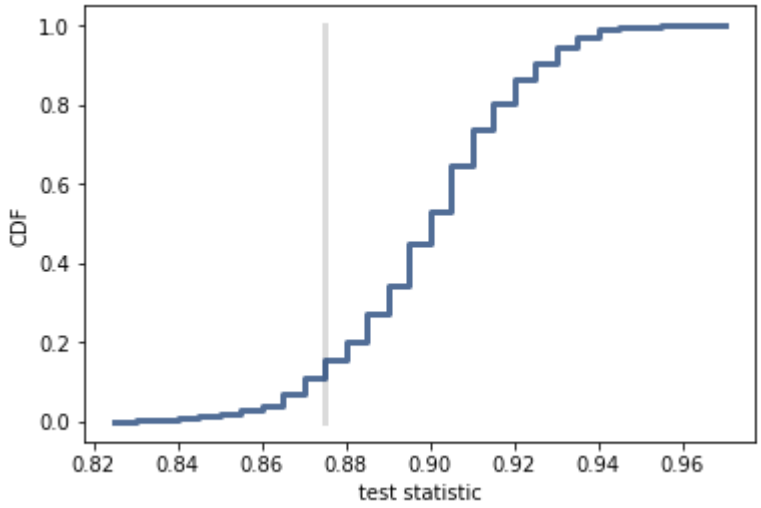
Out[219]:
```
BMI    175
dtype: int64
```

In [220]:
```
from scipy.stats import bernoulli


class BinomialTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        success, failure = data
        total = success + failure

        test_stat = success / total

        return test_stat

    # generate samples under null model
    def RunModel(self):
        success, failure = self.data
        total = success + failure

        sample_bernoulli = bernoulli.rvs(0.9, size=total)

        sample_success = sum(sample_bernoulli)
        sample_failure = total - sample_success

        data = sample_success, sample_failure

        return data

    def PValue(self, iters=1000):
        """Computes the distribution of the test statistic and p-value.

        iters: number of iterations

        returns: float p-value
        """
        self.test_stats = [self.TestStatistic(self.RunModel())
                           for _ in range(iters)]
        self.test_cdf = thinkstats2.Cdf(self.test_stats)


        count = sum(1 for x in self.test_stats if x <= self.actual)
        return count / iters
```

In [221]:
```
data = 175, 25
ct = BinomialTest(np.array(data))
pvalue = ct.PValue()
pvalue
```

Out[221]: 0.158

In [222]:
```
ct.PlotCdf()
thinkplot.Config(xlabel='test statistic',
                 ylabel='CDF')
```



The p value obtained for our sample is 0.158 . and this value lies outside the critical region.

As the p value is greater than the lower tail value which is 0.05 and hence we have sufficient evidence to accept our null hypothesis.

Hence we accept our null hypothesis whcih states that the probability of women having BMI greater than 25 is 70%.

Chi-Square Hypothesis Test

The Null-Hypothesis states:

H0:There is no statistically significant relationship between BMI of women and the result of diabetes test.

Ha:There is a statistically significant relationship between BMI of women and the result of diabetes test.

In [288]:
```
contingency_table = pd.crosstab(
    df['Result'],
    df['BMI Inference'],
    margins = True
)
contingency_table
```

Out[288]:

BMI Inference / Result	normal	obese	over weight	underweight	All
negative	95	262	139	4	500
positive	7	221	40	0	268
All	102	483	179	4	768

In [290]:
```
f_obs = np.append(contingency_table.iloc[0][0:4].values, contingency_table.ilo
c[1][0:4].values)
f_obs
```

Out[290]:
```
array([ 95, 262, 139,   4,   7, 221,  40,   0], dtype=int64)
```

In [292]:
```
row_sums = contingency_table.iloc[0:2,4].values
row_sums
```

Out[292]:
```
array([500, 268], dtype=int64)
```

In [293]:
```
col_sums = contingency_table.iloc[2,0:4].values
col_sums
```

Out[293]:
```
array([102, 483, 179,   4], dtype=int64)
```

In [294]:
```
total = contingency_table.loc['All', 'All']

f_expected = []
for j in range(2):
    for i in col_sums:
        f_expected.append(i*row_sums[j]/total)
f_expected
```

Out[294]:
```
[66.40625,
 314.453125,
 116.53645833333333,
 2.6041666666666665,
 35.59375,
 168.546875,
 62.463541666666664,
 1.3958333333333333]
```

In [298]:
```
chi_squared_statistic = ((f_obs - f_expected)**2/f_expected).sum()
print('Chi-squared Statistic: {}'.format(chi_squared_statistic))
```

```
Chi-squared Statistic: 74.9084804674606
```

In [299]:
```
dof = (len(row_sums)-1)*(len(col_sums)-1)
print("Degrees of Freedom: {}".format(dof))
```

```
Degrees of Freedom: 3
```

Now with df=3 and taking our signifivance level as α = 5% i.e. 0.05 from the chi-square test statistic table we get the statistic value as 7.82 .

As statistic value found for our data is 74.908 which is greater than 7.82 that is the critical value of test statistic we have sufficient evidence to reject the Null hypothesis and accept the fact that there exists a relationship between the BMI of women and the result of their Diabetes test conducted.

Odds

Odds tells us about probability of winning to the probability of loosing

In [63]:
```
d1 = thinkstats2.Cdf(df['Glucose'])
p1 = d1.Prob(120)
q1 = 1-p1
print("probability of getting Glucose value as 120 or less is :" , p1)
print("probability of getting Glucose value above 120 is :" , q1)
def Odds(p1):
    return p1 / (1-p1)
print()
print("the odds in favour of a glucose value as 120 are ", Odds(p1))
```

```
probability of getting Glucose value as 120 or less is : 0.5390625
probability of getting Glucose value above 120 is : 0.4609375

the odds in favour of a glucose value as 120 are  1.1694915254237288
```

In [66]:
```
d1 = thinkstats2.Pmf(df['Outcome'])
p1 = d1.Prob(1)
q1 = 1-p1
print("probability of getting Outcome as positive :" , p1)
print("probability of getting Outcome as negative :" , q1)
def Odds(p1):
    return p1 / (1-p1)
print()
print("the odds in favour of a positive outcome of the results are ", Odds(p1
))
```

```
probability of getting Outcome as positive : 0.3489583333333333
probability of getting Outcome as negative : 0.6510416666666667

the odds in favour of a positive outcome of the results are  0.53599999999999
99
```

Bayes Theorem

In [68]:
```
class BayesTable(pd.DataFrame):
    def __init__(self, hypothesis, prior=1):
        columns = ['hypothesis', 'prior', 'likelihood', 'unnormalized', 'poste
rior']
        super().__init__(columns=columns)
        self.hypothesis = hypothesis
        self.prior = prior

    def multiply(self):
        self.unnormalized = self.prior * self.likelihood # unnormalised is mul
tiplication of prior into likelihood

    def normalize(self):
        nc = np.sum(self.unnormalized)
        self.posterior = self.unnormalized / nc
        return nc

    def update(self):
        self.multiply()
        return self.normalize()

    def reset(self):
        return BayesTable(self.hypothesis, self.posterior)
```

In [69]:
```
table = BayesTable(['Positive', 'Negative'])
table
```

Out[69]:

	hypothesis	prior	likelihood	unnormalized	posterior
0	Positive	1	NaN	NaN	NaN
1	Negative	1	NaN	NaN	NaN

By using conditional probability theorem we can know the number of people in our data set who have a positive or a negative result given that they are lying in the obese category of BMI inference.

The probability of getting a Obese person with test result as Positive is 221/483 and the probability with test result as Negative is 262/483 .

In [32]:
```
table.likelihood = [221/768, 262/768]
table
```

Out[32]:

	hypothesis	prior	likelihood	unnormalized	posterior
0	Positive	1	0.287760	NaN	NaN
1	Negative	1	0.341146	NaN	NaN

The next step is to multiply the priors by the likelihoods, which yields the unnormalized posteriors

In [33]:
```
table.multiply()
table
```

Out[33]:

	hypothesis	prior	likelihood	unnormalized	posterior
0	Positive	1	0.287760	0.287760	NaN
1	Negative	1	0.341146	0.341146	NaN

We calculate the normalizing constant using the normalize function from our BayesTable class.

In [34]:
```
table.normalize()
```

Out[34]: `0.62890625`

In [35]:
```
table
```

Out[35]:

	hypothesis	prior	likelihood	unnormalized	posterior
0	Positive	1	0.287760	0.287760	0.457557
1	Negative	1	0.341146	0.341146	0.542443

We can read the posterior probabilities from the last column:

the probability that we get a women with positive result given the women is obese is 45.7% .