Data Preprocessing - Exploratory Data Analysis

In [69]:

```
import pandas as pd
import os
```

Load CSV Files with Pandas

In [70]:

```
os.chdir('/Users/tomisin/Dropbox/My Mac (Tomisins-MacBook-Pro.local)/Documents/Data
```

In [71]:

```
# Load CSV using Pandas
df = pd.read_csv("diabetes.csv")
```

In [72]:

```
df.head()
```

Out[72]:

	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

In [73]:

```
def myincom(x):
    y = x* 45
    return y



def dorsiResp():
    pass







a = 5


```

In [74]:

```python
j = myincom(a)
print(j)
```

225

In [75]:

```python
df1 = pd.read_csv("Housing_w_headers.csv")
```

#Step1: Peek at your data. Call for the top and bottom 'n' rows

In [103]:

```python
df1.head(20)
```

Out[103]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterl
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	
5	10850000	7500	3	3	1	yes	no	yes	
6	10150000	8580	4	3	4	yes	no	no	
7	10150000	16200	5	3	2	yes	no	no	
8	9870000	8100	4	1	2	yes	yes	yes	
9	9800000	5750	3	2	4	yes	yes	no	
10	9800000	13200	3	1	2	yes	no	yes	
11	9681000	6000	4	3	2	yes	yes	yes	
12	9310000	6550	4	2	2	yes	no	no	
13	9240000	3500	4	2	2	yes	no	no	
14	9240000	7800	3	2	2	yes	no	no	
15	9100000	6000	4	1	2	yes	no	yes	
16	9100000	6600	4	2	2	yes	yes	yes	
17	8960000	8500	3	2	4	yes	no	no	
18	8890000	4600	3	2	2	yes	yes	no	
19	8855000	6420	3	2	2	yes	no	no	

In [104]:

```
df1.tail(20)
```

Out[104]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterh
525	2345000	3640	2	1	1	yes	no	no	
526	2310000	3180	2	1	1	yes	no	no	
527	2275000	1836	2	1	1	no	no	yes	
528	2275000	3970	1	1	1	no	no	no	
529	2275000	3970	3	1	2	yes	no	yes	
530	2240000	1950	3	1	1	no	no	no	
531	2233000	5300	3	1	1	no	no	no	
532	2135000	3000	2	1	1	no	no	no	
533	2100000	2400	3	1	2	yes	no	no	
534	2100000	3000	4	1	2	yes	no	no	
535	2100000	3360	2	1	1	yes	no	no	
536	1960000	3420	5	1	2	no	no	no	
537	1890000	1700	3	1	2	yes	no	no	
538	1890000	3649	2	1	1	yes	no	no	
539	1855000	2990	2	1	1	no	no	no	
540	1820000	3000	2	1	1	yes	no	yes	
541	1767150	2400	3	1	1	no	no	no	
542	1750000	3620	2	1	1	yes	no	no	
543	1750000	2910	3	1	1	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	

In [77]:

```
df1.columns
```

Out[77]:

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroa
d',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

In [79]:

```
df1.info() # helps identify the structure of our data and see if any features is
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

In [80]:

```
# Conclusion from .info()

'''
summary of the size of the data,
the number of features and samples and the data type.
Also tells us if there any missing valuues

'''
```

Out[80]:

```
'\nsummary of the size of the data, \nthe number of features and sampl
es and the data type. \nAlso tells us if there any missing valuues\n
\n'
```

In [81]:

```
df.describe() #look out for data distribution and skewness
```

Out[81]:

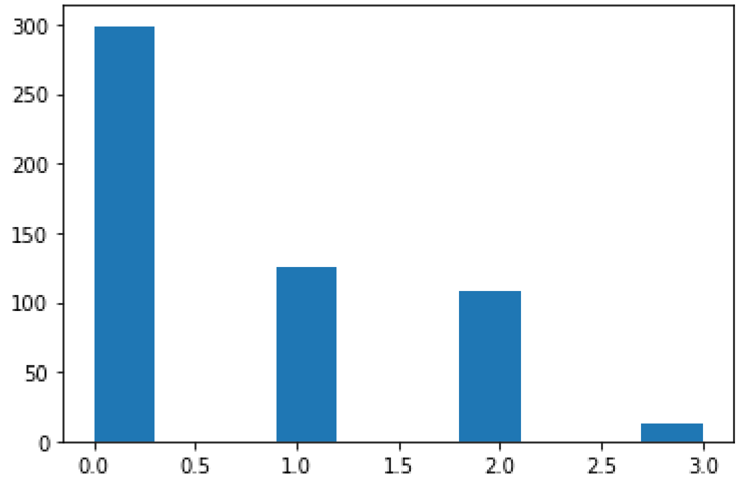
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	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	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [82]:

```
import matplotlib.pyplot as plt
plt.hist(df1['parking'].values)
```

Out[82]:

```
(array([299.,   0.,   0., 126.,   0.,   0., 108.,   0.,   0.,  12.]),
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),
 <BarContainer object of 10 artists>)
```

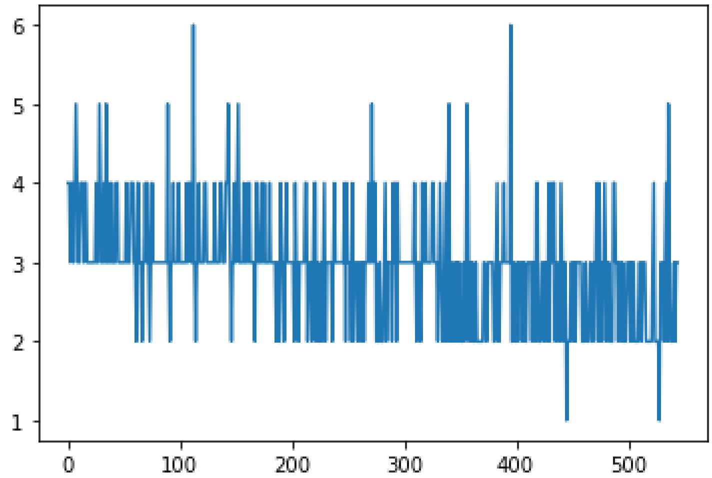


In [83]:

```
plt.plot(df1['bedrooms'].values)
```

Out[83]:

```
[<matplotlib.lines.Line2D at 0x7f83daefdd60>]
```



In [84]:

```
data = pd.read_csv("diamond.csv")
```

In [85]:

```
data.head()
```

Out[85]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	NaN	3.95	NaN	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	Good	NaN	VS1	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	Premium	I	NaN	62.4	58.0	334.0	4.20	NaN	2.63
4	0.31	Good	J	NaN	63.3	58.0	335.0	4.34	4.35	2.75

In [86]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53968 entries, 0 to 53967
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53968 non-null  float64
 1   cut      53968 non-null  object 
 2   color    48571 non-null  object 
 3   clarity  48568 non-null  object 
 4   depth    48573 non-null  float64
 5   table    48569 non-null  float64
 6   price    48571 non-null  float64
 7   x        48573 non-null  float64
 8   y        48571 non-null  float64
 9   z        48569 non-null  float64
dtypes: float64(7), object(3)
memory usage: 4.1+ MB
```

In [87]:

```
data.isna().sum() #shows number of missing values in each column
```

Out[87]:

```
carat         0
cut           0
color      5397
clarity    5400
depth      5395
table      5399
price      5397
x          5395
y          5397
z          5399
dtype: int64
```

In [88]:

```
#TODO:MISSING VALUES:  treat missing values for the following featurees

'''
color      5397
clarity    5400
depth      5395
table      5399
price      5397
x          5395
y          5397
z          5399
'''
```

Out[88]:

```
'\ncolor      5397\nclarity    5400\ndepth      5395\ntable      5399
\nprice      5397\nx          5395\ny          5397\nz          5399
\n'
```

Renaming Columns

In [89]:

```
df.columns
```

Out[89]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'In
sulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [94]:

```
from pandas import read_csv
filename = "diabetes.csv"
names = ['preg', 'Glu', 'BP', 'skinThickness', 'Insulin', 'BMI', 'pedi', 'age',
dfx = read_csv(filename, names=names)
```

In [95]:

```
dfx.columns
```

Out[95]:

```
Index(['preg', 'Glu', 'BP', 'skinThickness', 'Insulin', 'BMI', 'pedi',
'age',
       'class'],
      dtype='object')
```

Checking dataTypes

In [100]:

```
types = df.dtypes
print(types)
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [101]:

```
types = df1.dtypes
print(types)
```

```
price                  int64
area                   int64
bedrooms               int64
bathrooms              int64
stories                int64
mainroad              object
guestroom             object
basement              object
hotwaterheating       object
airconditioning       object
parking                int64
prefarea              object
furnishingstatus      object
dtype: object
```

Missing Data Treatment

```
import pandas as pd
data5 = data
for col in data5.columns:
    data5.loc[data5.sample(frac=0.1).index, col] = pd.np.nan
```

In [ ]:

```
# MISSING DATA TREATMENT

df.fillna(0)
df.fillna(method ='pad') #Filling null values with the previous ones
df.fillna(method ='bfill') #Filling null value with the next ones
data.replace(to_replace = np.nan, value = -99999999)
df.interpolate(method ='linear', limit_direction ='forward') #Using interpolate(
df.dropna() #Dropping rows with at least 1 null value
df.dropna(how = 'all') #Dropping rows if all values in that row are missing
```

In [112]:

```
for val in data['color'].isna().values:
    if val ==True:
        print('Yes')
```

```
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
Yes
--
```

In [106]:

```
df
```

Out[106]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

768 rows × 9 columns

In [107]:

```
df.isna()
```

Out[107]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
0	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	
763	False	False	False	False	False	False	Fa
764	False	False	False	False	False	False	Fa
765	False	False	False	False	False	False	Fa
766	False	False	False	False	False	False	Fa
767	False	False	False	False	False	False	Fa

768 rows × 9 columns

In [109]:

```
print(df.isna().sum())
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [111]:

```
x = data['color'].isna().values.tolist()
```

In [35]:

```
x
```

Out[35]:

```
[False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 True,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
```

In [23]:

```
#data.isnull().sum(axis = 0).sum()
print(data.isna().sum())
```

Out[23]:

```
30
```

Creating a dataFrame

In [114]:

```
temp = [37, 37.2,36.9,None , 37.8,39]
mtime = [1,2,None,4,None,6]

df_x = pd.DataFrame()
df_x['temp'] = temp
df_x['mtime']= mtime

df_x
```

Out[114]:

	temp	mtime
0	37.0	1.0
1	37.2	2.0
2	36.9	NaN
3	NaN	4.0
4	37.8	NaN
5	39.0	6.0

In [115]:

```
#df_x = df_x.interpolate(method ='linear', limit_direction ='forward')
```

Manipulating missing values

In [116]:

```
#Back fill method
df_x = df_x.fillna(method ='bfill')
```

In [117]:

```
df_x
```

Out[117]:

	temp	mtime
0	37.0	1.0
1	37.2	2.0
2	36.9	4.0
3	37.8	4.0
4	37.8	6.0
5	39.0	6.0

In [119]:

```
data.isna().sum()
```

Out[119]:

```
carat          0
cut            0
color       5397
clarity     5400
depth       5395
table       5399
price       5397
x           5395
y           5397
z           5399
dtype: int64
```

In [121]:

```
newData = data.fillna(method ='bfill')
newData
```

0	0.23	Ideal	E	SI2	61.5	55.0	326.0	3.95	3.84	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326.0	3.89	3.84	2.31
2	0.23	Good	I	VS1	56.9	65.0	327.0	4.05	4.07	2.31
3	0.29	Premium	I	VVS2	62.4	58.0	334.0	4.20	4.35	2.63
4	0.31	Good	J	VVS2	63.3	58.0	335.0	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53963	0.72	Ideal	D	SI1	60.8	57.0	2757.0	5.75	5.76	3.50
53964	0.72	Good	D	SI1	63.1	55.0	2757.0	5.69	5.75	3.61
53965	0.70	Very Good	H	SI1	62.8	60.0	2757.0	5.66	5.68	3.56
53966	0.86	Premium	H	SI2	61.0	58.0	2757.0	6.15	6.12	3.74
53967	0.75	Ideal	D	SI2	62.2	55.0	2757.0	5.83	5.87	3.64

53968 rows × 10 columns

In [122]:

```
newData.isna().sum()
```

Out[122]:

```
carat       0
cut         0
color       0
clarity     0
depth       0
table       0
price       0
x           0
y           0
z           0
dtype: int64
```

In [124]:

```
babyHgt = [2,4,7,9 ,11]
babyWt = [23,25,24.5,25,27]
babyDf = pd.DataFrame()

babyDf['wt'] = babyWt
babyDf['ht'] = babyHgt

babyDf
```

Out[124]:

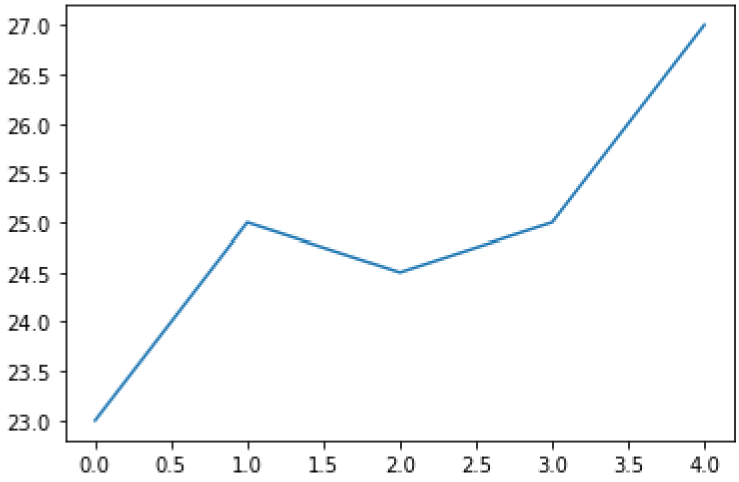
	wt	ht
0	23.0	2
1	25.0	4
2	24.5	7
3	25.0	9
4	27.0	11

In [125]:

```
plt.plot(babyDf["wt"])
```

Out[125]:

```
[<matplotlib.lines.Line2D at 0x7f83d86ca850>]
```



In [49]:

```
babyHgt = [2,4,None,9,11]
babyWt = [23,25,None,None,27]
babyDf = pd.DataFrame()

babyDf['wt'] = babyWt
babyDf['ht'] = babyHgt

babyDf
```

Out[49]:

	wt	ht
0	23.0	2.0
1	25.0	4.0
2	NaN	NaN
3	NaN	9.0
4	27.0	11.0

In [126]:

```
#Interpolate method
newBabyDf = babyDf.interpolate(method ='linear')
```

In [127]:

```
newBabyDf
```

Out[127]:

	wt	ht
0	23.0	2
1	25.0	4
2	24.5	7
3	25.0	9
4	27.0	11

In [131]:

```
df_x = df.fillna(method ='pad')
```

In [132]:

```python
cleanData = df_x.dropna()
cleanData
```

Out[132]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctic
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.17
764	2	122	70	27	0	36.8	0.34
765	5	121	72	23	112	26.2	0.24
766	1	126	60	0	0	30.1	0.34
767	1	93	70	31	0	30.4	0.31

768 rows × 9 columns

In [133]:

```python
cleanData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 60.0 KB
```

In [134]:

```
cleanData.isna().sum()
```

Out[134]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
x = data5.dropna()
print(x.isnull().sum())
```

Understand the Dimensions of Your Data

In [15]:

```
# Use the shape() function
shape = data.shape
print(shape)
```

```
(768, 9)
```

Data Type For Each Attribute

In [11]:

```
# Data Types for Each Attribute
'''from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)'''


types = data.dtypes
print(types)
```

```
preg       int64
plas       int64
pres       int64
skin       int64
test       int64
mass     float64
pedi     float64
age        int64
class      int64
dtype: object
```

Descriptive Statistics

In [ ]:

```

Descriptive statistics can give you great insight into the shape of each attribu

The describe() function on the Pandas
DataFrame lists 8 statistical properties of each attribute. They are:
• Count.
• Mean.
• Standard Deviation.
• Minimum Value.
• 25th Percentile.
• 50th Percentile (Median).
• 75th Percentile.
• Maximum Value.
```

In [ ]:

```
pandas.set option() changes the precision of the numbers and the preferred width

This display helps to quickly review data patterns like the presence
of NA values for missing data or surprising distributions for attributes
```

In [135]:

```
# Statistical Summary
from pandas import read_csv
from pandas import set_option
#filename = "c:/dataset/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
df = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)


description = df.describe()
print(description)
```

```
       preg plas pres skin test mass   pedi  age class
count   769  769  769  769  769  769    769  769   769
unique   18  137   48   52  187  249    518   53     3
top       1  100   70    0    0   32  0.254   22     0
freq    135   17   57  227  374   13      6   72   500
```

Data Imbalance

Classification problems require you know how balanced the class values are. Highly imbalanced problems need to be addressed before modeling.

In classification problems you need to inspect the Target Class for balance

In [85]:

```
# Class Distribution
'''from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)'''

# diabetes.csv
class_counts = data.groupby('Outcome').size()
print(class_counts)
```

```
Outcome
0.0    500
1.0    267
dtype: int64
```

In [ ]:

```
#You can see that there are more than double the number of observations with cla
#(no onset of diabetes) than there are with class 1 (onset of diabetes).
```

Correlations Between Attributes

In [24]:

```python
# Pairwise Pearson correlations
from pandas import read_csv
from pandas import set_option
#filename = 'c:/dataset/master/pima-indians-diabetes.data.csv'
#names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class
data = read_csv(filename)
set_option('display.width', 1000)
set_option('precision', 2)


correlations = data.corr(method='pearson')
print(correlations)
```

```
                          Pregnancies  Glucose  BloodPressure  SkinThi
ckness  Insulin   BMI  DiabetesPedigreeFunction   Age  Outcome
Pregnancies                      1.00     0.13           0.14
-0.08    -0.07  0.02                     -0.03  0.54     0.22
Glucose                          0.13     1.00           0.15
0.06     0.33  0.22                     0.14  0.26     0.47
BloodPressure                    0.14     0.15           1.00
0.21     0.09  0.28                     0.04  0.24     0.07
SkinThickness                   -0.08     0.06           0.21
1.00     0.44  0.39                     0.18 -0.11     0.07
Insulin                         -0.07     0.33           0.09
0.44     1.00  0.20                     0.19 -0.04     0.13
BMI                              0.02     0.22           0.28
0.39     0.20  1.00                     0.14  0.04     0.29
DiabetesPedigreeFunction        -0.03     0.14           0.04
0.18     0.19  0.14                     1.00  0.03     0.17
Age                              0.54     0.26           0.24
-0.11    -0.04  0.04                     0.03  1.00     0.24
Outcome                          0.22     0.47           0.07
0.07     0.13  0.29                     0.17  0.24     1.00
```

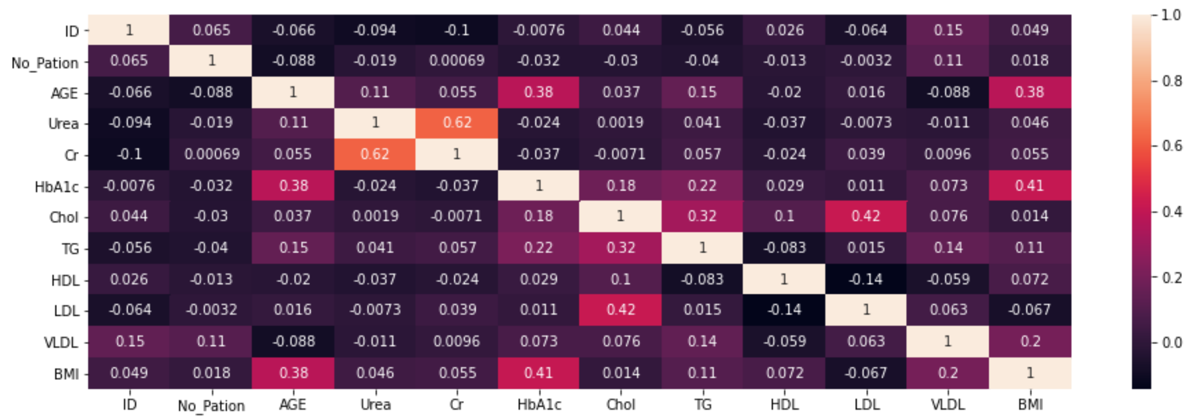
Seaborn Package Can be Used

In [37]:

```python

import seaborn as sb
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import set_option

#filename = 'c:/dataset/master/pima-indians-diabetes.data.csv'

#names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class
#data.columns = names
data = data #read_csv(filename)
set_option('display.width', 100)

plt.figure(figsize=(16,5))
#sns.heatmap(data.corr())

sb.heatmap(data.corr(), annot = True)
```

Out[37]:

```
<AxesSubplot:>
```



In [35]:

```
data.corr()
```

Out[35]:

	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	TG
ID	1.000000	0.064920	-0.065980	-0.094434	-0.102457	-0.007571	0.044390	-0.055908
No_Pation	0.064920	1.000000	-0.088006	-0.019160	0.000692	-0.032057	-0.030171	-0.039885
AGE	-0.065980	-0.088006	1.000000	0.105092	0.054941	0.379136	0.036649	0.148204
Urea	-0.094434	-0.019160	0.105092	1.000000	0.624134	-0.023603	0.001852	0.040980
Cr	-0.102457	0.000692	0.054941	0.624134	1.000000	-0.037412	-0.007097	0.056579
HbA1c	-0.007571	-0.032057	0.379136	-0.023603	-0.037412	1.000000	0.177489	0.218556
Chol	0.044390	-0.030171	0.036649	0.001852	-0.007097	0.177489	1.000000	0.321789
TG	-0.055908	-0.039885	0.148204	0.040980	0.056579	0.218556	0.321789	1.000000
HDL	0.026231	-0.013357	-0.020038	-0.036994	-0.023804	0.028933	0.103814	-0.083001
LDL	-0.064305	-0.003171	0.016105	-0.007301	0.039479	0.011057	0.416665	0.015378
VLDL	0.146142	0.113754	-0.087903	-0.011191	0.009615	0.073462	0.076294	0.144570
BMI	0.049409	0.017719	0.375956	0.045618	0.054746	0.413350	0.013678	0.110757

the Seaborn heatmap function can take in 18 arguments. This is what the function looks like with all the arguments:

sns.heatmap(data, vmin=None, vmax=None, cmap=None,center=None, robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs)

In [ ]:

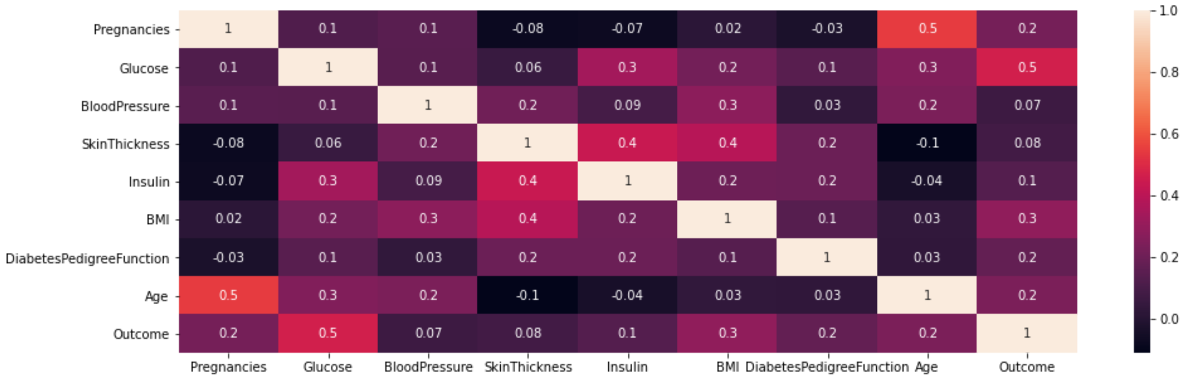
```
Sometimes your correlation coefficients mayhave too many floating digits.

Reduce the decimal places to improve readability using the argument
fmt = '.3g'or fmt = '.1g' because by default the function displays two digits af
decimal (greater than zero) i.e fmt='.2g'

MULTI-COLINEARITY
```

In [8]:

```
plt.figure(figsize=(16,5))
sns.heatmap(data.corr(), annot = True, fmt='.1g')
```

Out[8]:

<AxesSubplot:>

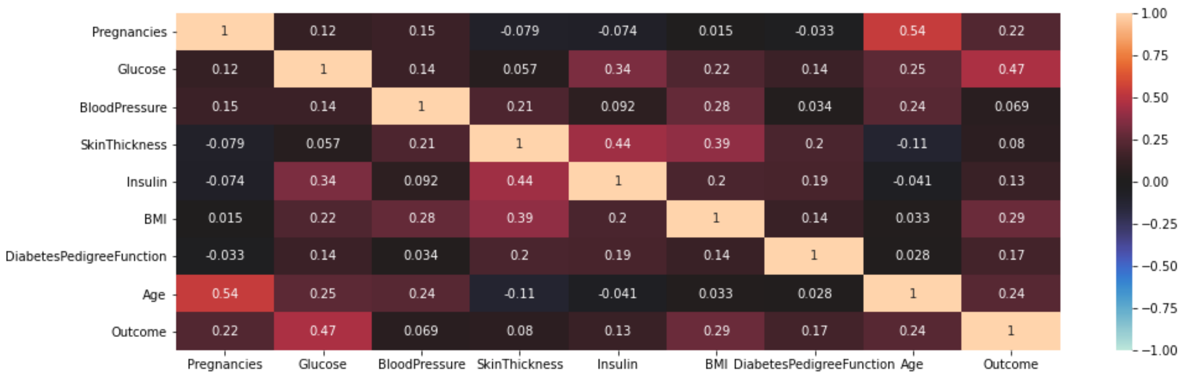


In [9]:

```
plt.figure(figsize=(16,5))
sns.heatmap(data.corr(), annot = True, vmin=-1, vmax=1, center= 0)
```

Out[9]:

<AxesSubplot:>



In [ ]:

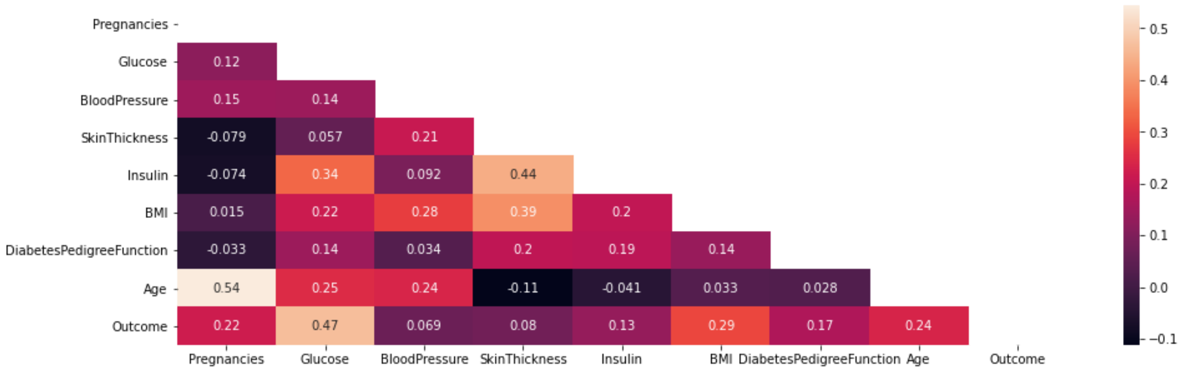
```
To change the shape use the NumPy methods; .triu() and .tril()
and then specify the Seaborn heatmap argument called mask=

.triu() is a method in NumPy that returns the lower triangle of any matrix giver
while .tril() returns the upper triangle of any matrix given to it.


```

In [10]:

```python

matrix = np.triu(data.corr())
plt.figure(figsize=(16,5))
sns.heatmap(data.corr(), annot=True, mask=matrix)
```

Out[10]:

```
<AxesSubplot:>
```

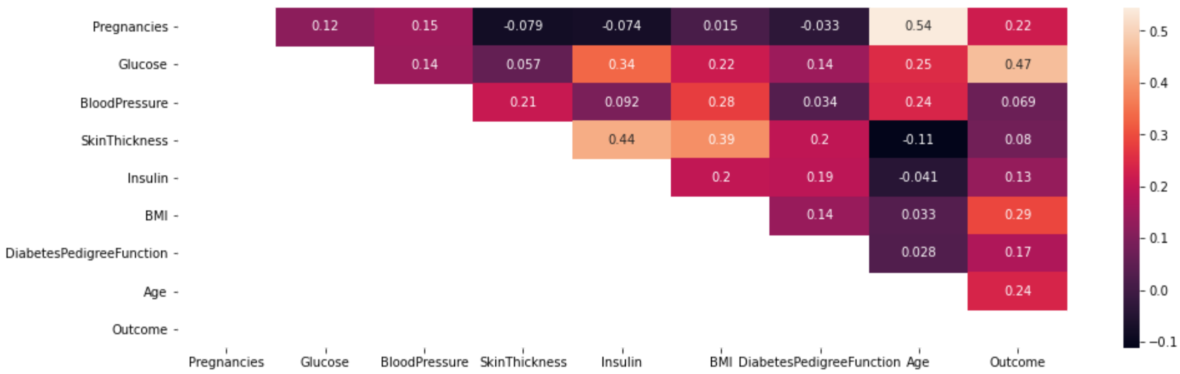


In [11]:

```python
mask = np.tril(data.corr())
plt.figure(figsize=(16,5))
sns.heatmap(data.corr(), annot=True, mask=mask)
```

Out[11]:

```
<AxesSubplot:>
```



Skew of Univariate Distributions

In [ ]:

```
Skew refers to a distribution that is assumed Gaussian (normal or bell curve) th
squashed in one direction or another.

Many machine learning algorithms assume a Gaussian distribution.
Knowing that an attribute has a skew may allow you to perform data preparation t
the skew and improve the accuracy of your models.

You can calculate the skew of each attribute using the skew() function on the Pa
```

In [ ]:

```
Skew can be positive (right) or negative (left) skew.
Values closer to zero show less skew.
```

In [88]:

```
# Skew for each attribute
'''from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''

skew = data.skew()
print(skew)
```

```
preg      0.901674
plas      0.170290
pres     -1.837988
skin      0.100862
test      2.273627
mass     -0.428455
pedi      1.927745
age       1.129597
class     0.638949
dtype: float64
```

In [ ]:

```

```

Understand Your Data With Visualization

Univariate Plots

In [ ]:

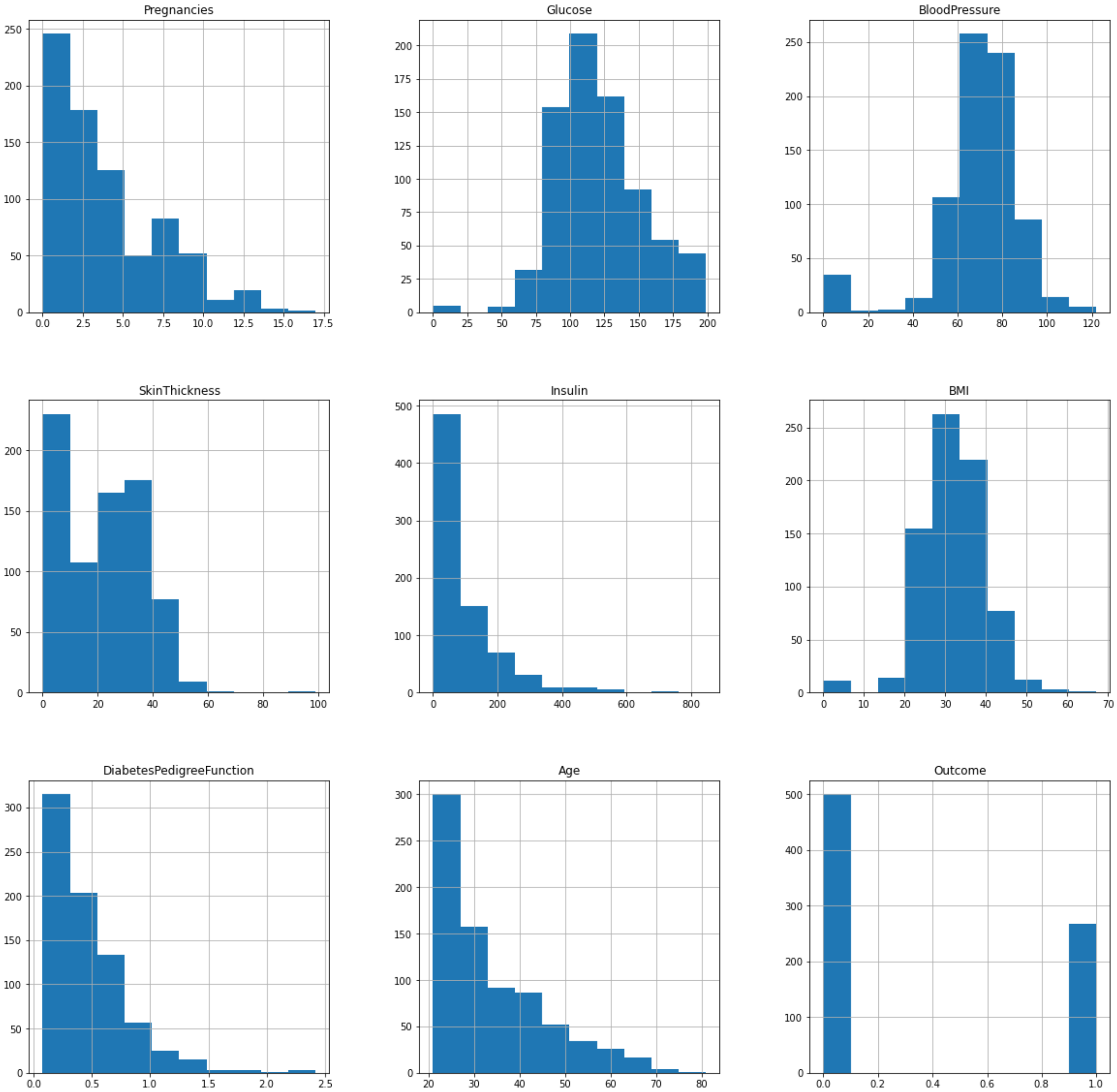
```
three techniques that you can use to understand each attribute of your dataset i
• Histograms.
• Density Plots.
• Box and Whisker Plots.
```

In [ ]:

```
A fast way to get an idea of the distribution of each attribute is to look at hi
group data into bins and provide you a count of the number of observations in ea

From the shape of the bins you can quickly get a feeling for whether an attribut
or even has an exponential distribution. It can also help you see possible outli
```

In [15]:

```python
#Univariate Histograms
from matplotlib import pyplot
'''
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''


data.hist()
plt.gcf().set_size_inches(20,20)
pyplot.show()
```



In [ ]:

```
the attributes age, pedi and test may have an exponential distribution.
mass and pres and plas attributes may have a Gaussian or nearly Gaussian distrib

many machine learning techniques assume a Gaussian univariate distribution on th
```

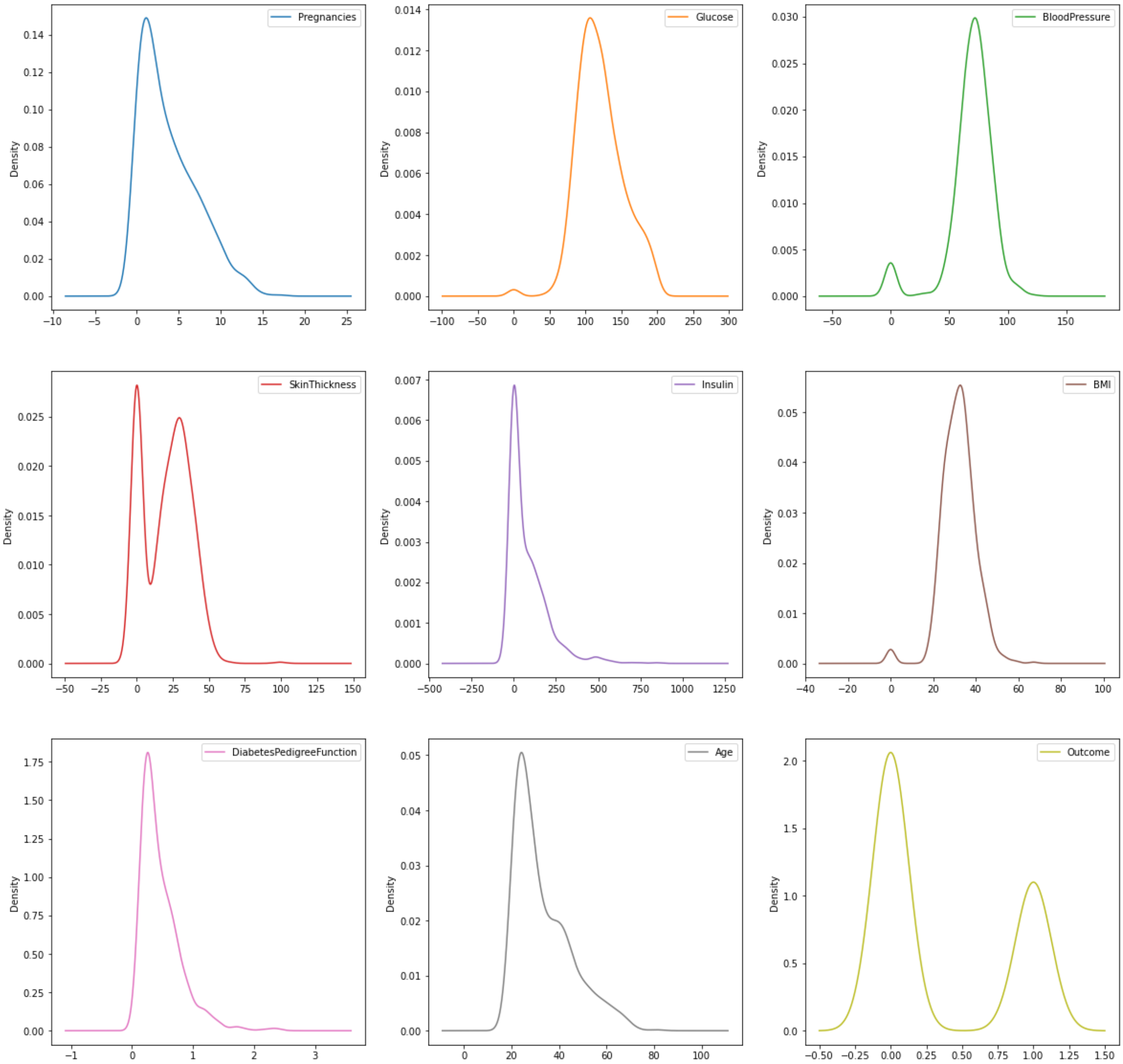
Density Plots

In [ ]:

```
Density plots are another way of getting a quick idea of the distribution of eac
The plots look like an abstracted histogram with a smooth curve drawn through th
```

In [16]:

```python
# Univariate Density Plots
from matplotlib import pyplot
from pandas import read_csv
'''filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''


data.plot(kind='density', subplots=True, layout=(3,3), sharex=False)
plt.gcf().set_size_inches(20,20)
pyplot.show()
```



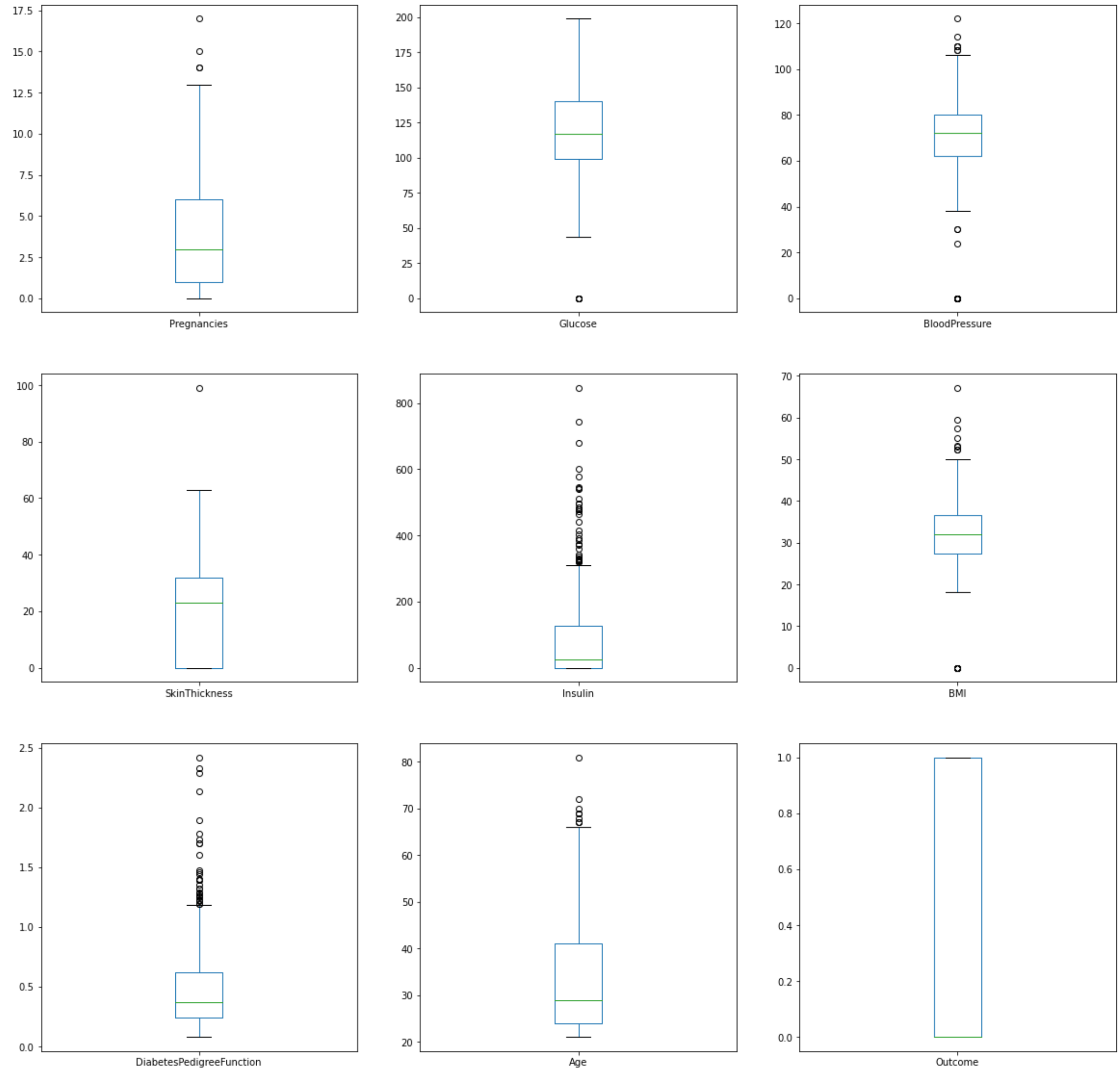
Box and Whisker Plots

In [ ]:

```

Boxplots summarize the distribution of each attribute,
drawing a line for the median (middle value) and a box around the 25th and 75th
percentiles (the middle 50% of the data).

The whiskers give an idea of the spread of the data
and dots outside of the whiskers show candidate outlier values
(values that are 1.5 times greater than the size of spread of the middle 50% of
```

In [17]:

```
# Box and Whisker Plots
from matplotlib import pyplot
from pandas import read_csv
'''filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''

data.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False)
plt.gcf().set_size_inches(20,20)
pyplot.show()
```



In [ ]:

```
Multivariate Plots
```

In [ ]:

```
Here we look at two plots that show the interactions between multiple variables
. Correlation Matrix Plot.
. Scatter Plot Matrix.
```

In [ ]:

```
Correlation gives an indication of how related the changes are between two varia
If two variables change in the same direction they are positively correlated.

If they change in opposite directions together (one goes up, one goes down),
then they are negatively correlated.

You can calculate the correlation between each pair of attributes using a correl

This is useful to know, because some machine learning algorithms like linear
and logistic regression can have poor performance if there are highly correlatec
in your data.
```

In [54]:

```
data.columns
```

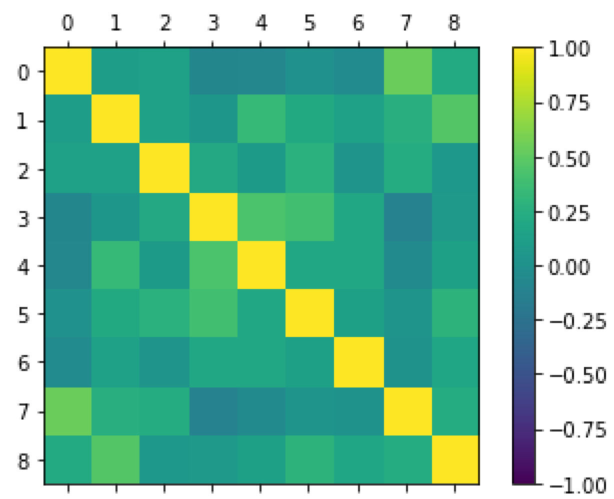
Out[54]:

```
Index(['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio', 'Oc
cupancy'], dtype='object')
```

In [18]:

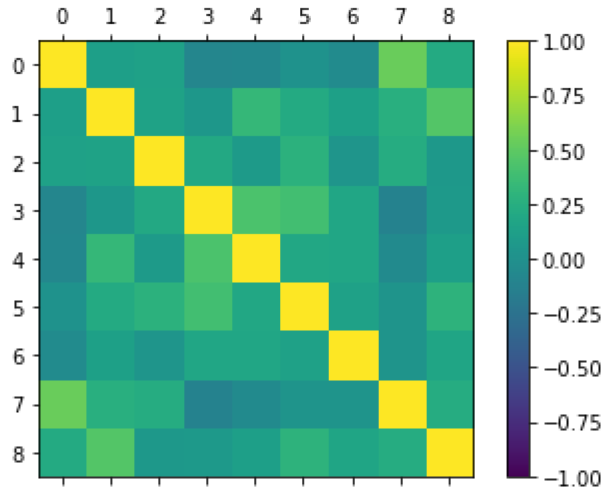
```python
# Correction Matrix Plot
from matplotlib import pyplot
from pandas import read_csv
import numpy
'''filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''
#names = list(data.columns)

correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-1d4b593c6ea8> in <module>
     17 ax.set_xticks(ticks)
     18 ax.set_yticks(ticks)
---> 19 ax.set_xticklabels(names)
     20 ax.set_yticklabels(names)
     21 pyplot.show()

NameError: name 'names' is not defined
```



In [35]:

```

# Correction Matrix Plot (generic)
from matplotlib import pyplot
from pandas import read_csv
import numpy
'''filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''

correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
pyplot.show()
```



In [ ]:

```
Scatter Plot Matrix
```

In [ ]:

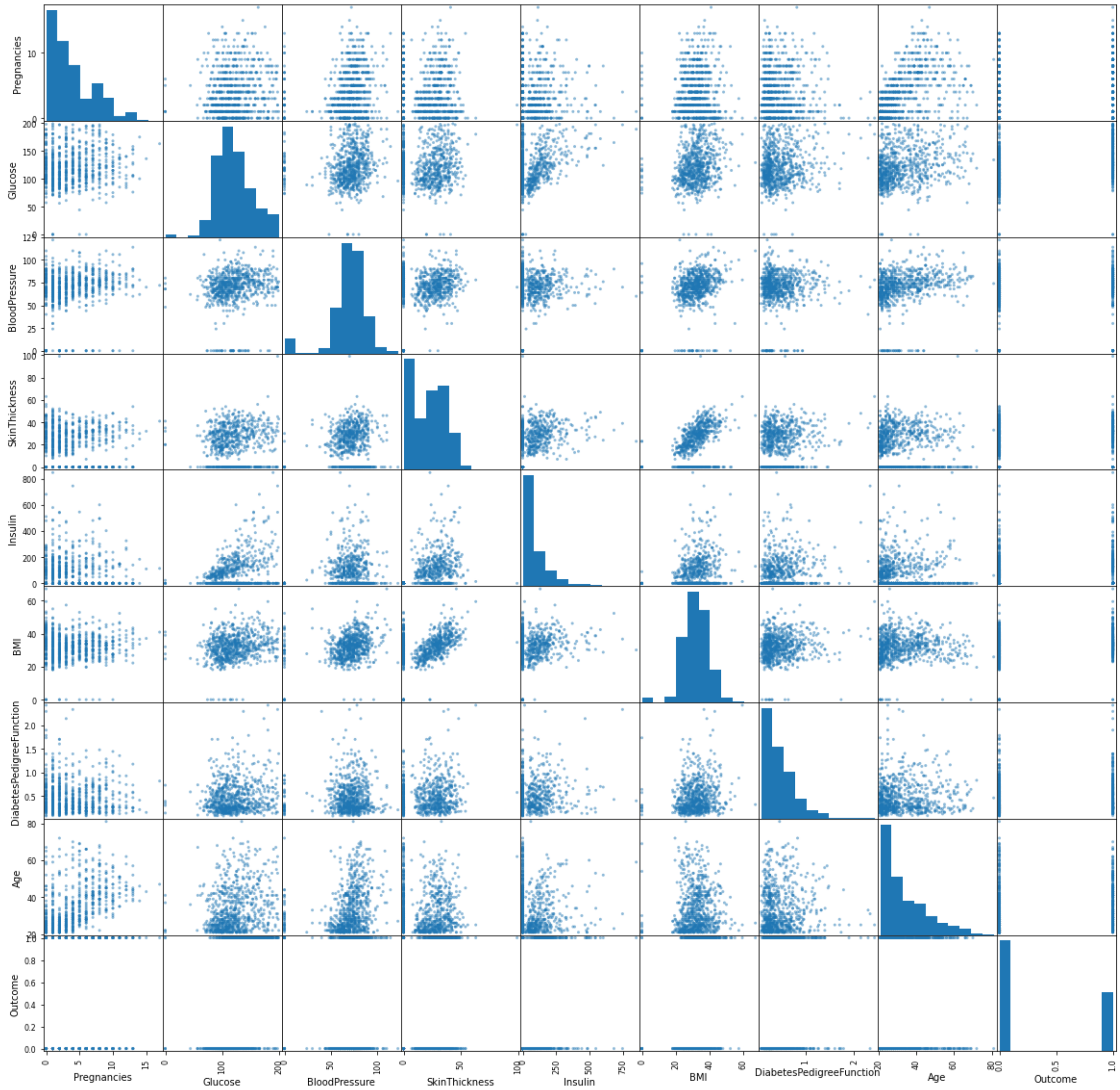
```

A scatter plot shows the relationship between two variables as dots in two dimer
axis for each attribute.

You can create a scatter plot for each pair of attributes in your data.
Drawing all these scatter plots together is called a scatter plot matrix.

Scatter plots are useful for spotting structured relationships between
variables, like whether you could summarize the
relationship between two variables with a line.

Attributes with structured relationships may also be correlated
and good candidates for removal from your dataset.
```

In [ ]:

```
the scatter plot matrix is symmetrical.
This is useful to look at the pairwise relationships from
different perspectives.
the diagonal shows histograms of each attribute.
```

In [19]:

```python
# Scatterplot Matrix
from matplotlib import pyplot
from pandas import read_csv
from pandas.plotting import scatter_matrix
'''filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)'''

scatter_matrix(data)
plt.gcf().set_size_inches(20,20)
pyplot.show()
```



In [68]:

```python
cleanData =0
```

In [69]:

```python
cleanData
```

Out[69]:

0