Data Exploration

In [26]:

```
import warnings
warnings.filterwarnings("ignore")
```

In [1]:

```
# Importing the Dataset to the Notebook
import numpy as np
import pandas as pd
arquivo = 'data/pima-data.csv'
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'classif']
df = pd.read_csv(arquivo, names=colunas)
```

In [2]:

```
# First contact with Dataset
df.head()
```

Out[2]:

	preg	plas	pres	skin	test	mass	pedi	age	classif
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

In [3]:

```
#Dataset shape (rows, columns)
df.shape
```

Out[3]:

```
(768, 9)
```

In [4]:

```
# Data types for each column
df.dtypes
```

Out[4]:

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

In [5]:

```
# Statistic description of the dataset
df.describe()
```

Out[5]:

	preg	plas	pres	skin	test	mass	pedi
count	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
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000

In [6]:

```
# Correlation between the variables
df.corr(method='pearson')
```

Out[6]:

	preg	plas	pres	skin	test	mass	pedi	age
preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341
plas	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514
pres	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970
test	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163
mass	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242
pedi	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000
classif	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356

In [7]:

```
#Checking the skew coefficient for each variable.
df.skew()
```

Out[7]:

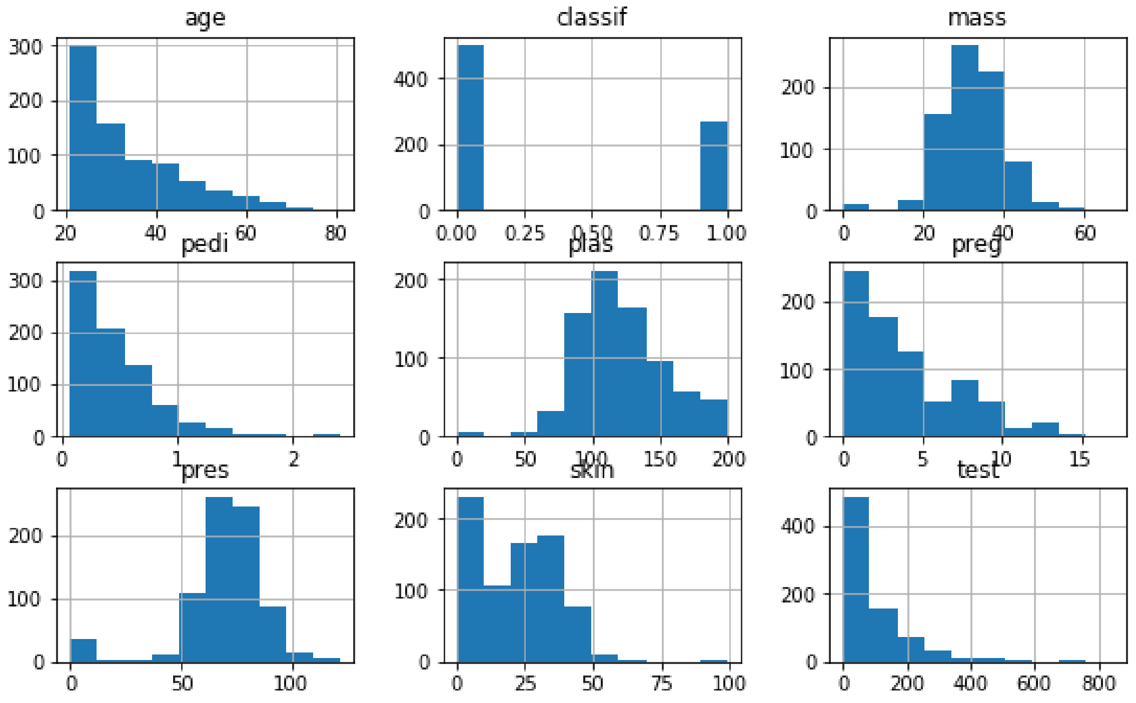
```
preg       0.901674
plas       0.173754
pres      -1.843608
skin       0.109372
test       2.272251
mass      -0.428982
pedi       1.919911
age        1.129597
classif    0.635017
dtype: float64
```

In [8]:

```
# Import MatplotLib for graphics
import matplotlib.pyplot as plt
%matplotlib inline
```

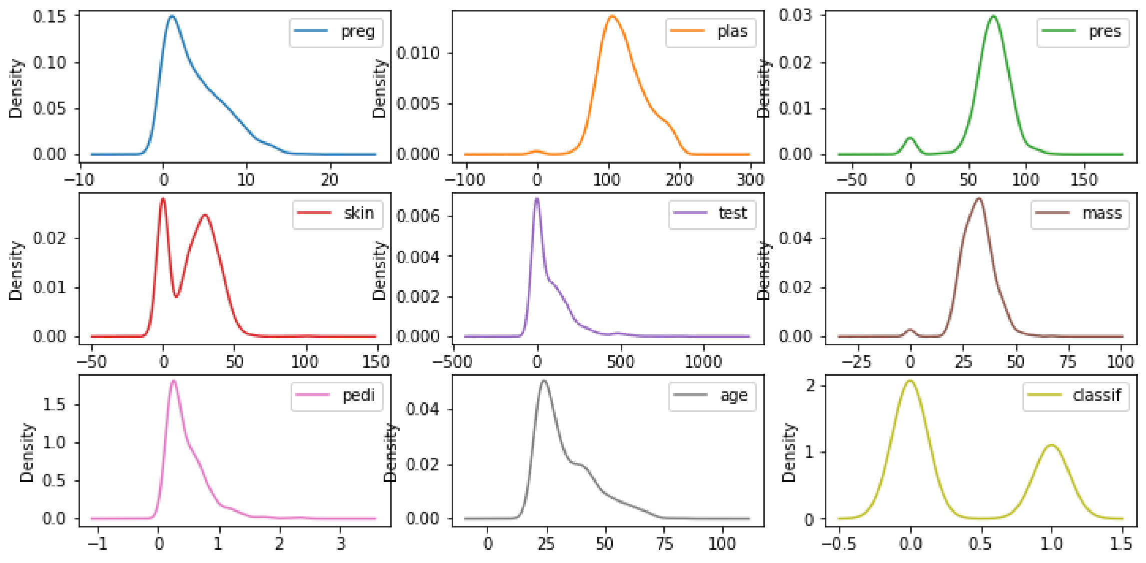
In [9]:

```
#Generating histograms for the dataset
df.hist(figsize=(10,6))
plt.show()
```



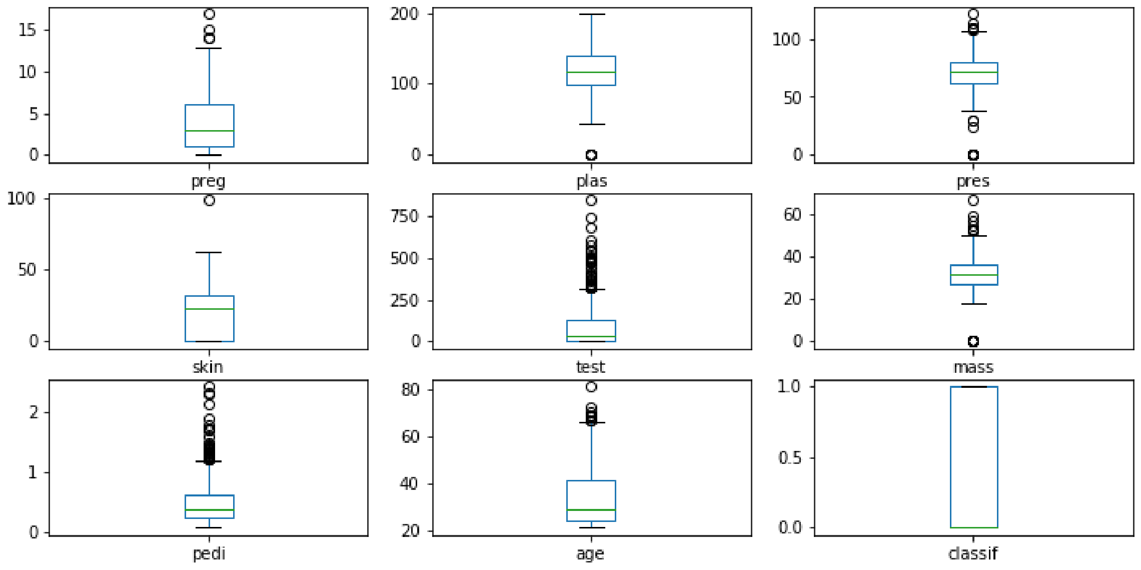
In [10]:

```
#Creating Density plots, another way to visualize the data distribution
#Sharex=False so each graphic has its own axis x values
df.plot(kind='density', subplots=True, layout=(3,3), sharex=False, figsize = (12,6))
plt.show()
```



In [11]:

```
# Creating Box Plots to see outliers.
# Here we can see that pedi, test, skin, test have the data concentrated in the lower v
alues.
df.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False, figsize =
(12,6))
plt.show()
```



In [12]:

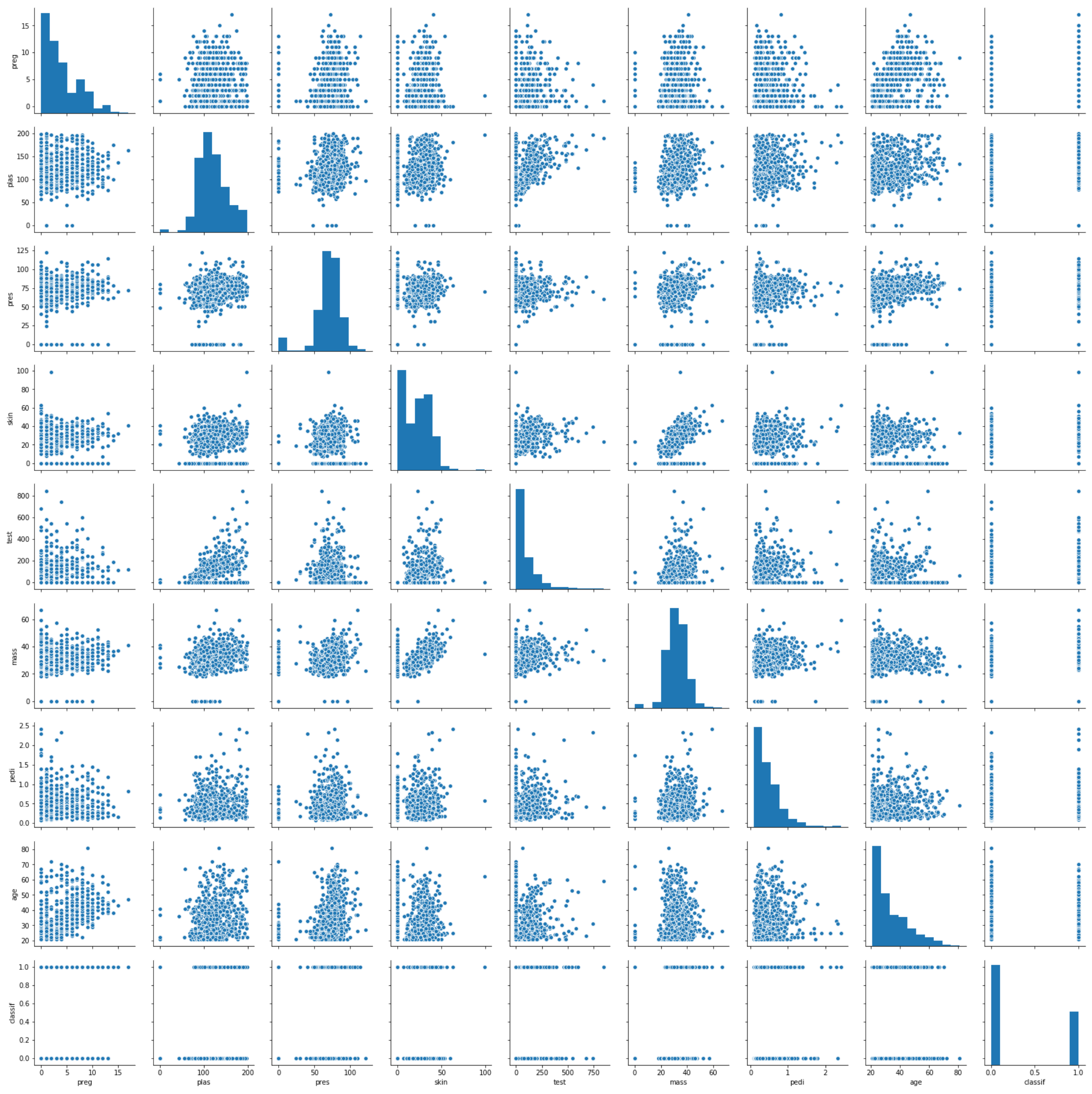
```
# Importing Seaborn for a few more data viz
import seaborn as sns
```

In [13]:

```
# With pairplot, we can see how the data is correlated. Mass vs Skin, Plas and test are
positively correlated, for example.
sns.pairplot(df)
```

```
Out[13]:

<seaborn.axisgrid.PairGrid at 0x18b65c855c0>
```

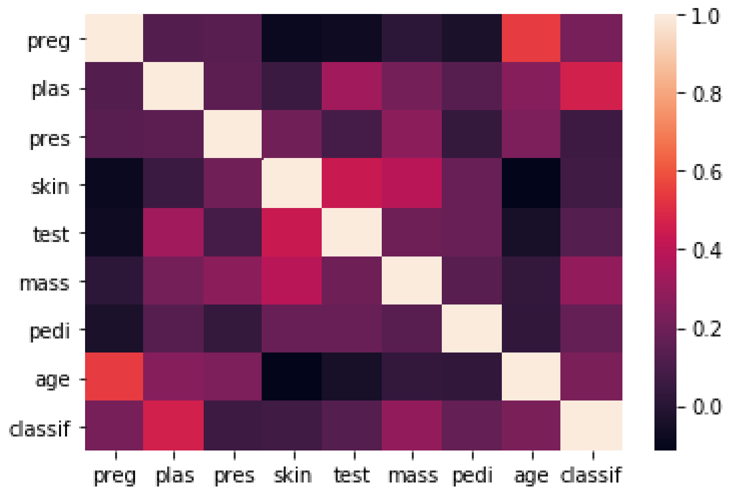


In [14]:

```
# The heatmap shows where the correlation is higher. Values closer to 1 indicate bigger
correlation.
# Those correlations can help us in the Feature seleaction phase, once the variables mo
re correlated might help with
# better results to the model.
sns.heatmap(df.corr())
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18b6ac0e860>
```



Data Standardization

In [27]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# Let's determine what is features (X) and what is target (y)
features = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age']
X = df[features]
y = df['classif']

#Normalization
#scaler = MinMaxScaler(feature_range = (0,1))
#rescaledX = scaler.fit_transform(X)

#Standardization
scaler = StandardScaler().fit(X)
standardX = scaler.transform(X)
```

Feature Selection

In [16]:

```
#Importing modules
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

In [18]:

```
# Checking the features and its importance in the model
x1 = standardX
# Create the Model for RFE and select only the 4 best variables
model =  LogisticRegression(solver='lbfgs')
rfe = RFE(model, 4)
fit = rfe.fit(x1,Y)
print('Variables: ', df.columns)
print('Variables with True are the ones selected:', fit.support_)
print('Ranking of variables by importance to the Logistic Regression Model: ', fit.rank
ing_)
print('Despite the fact we did this grading in the features, I am still using all of th
em, as it is a small dataset')
```

```
Variables:  Index(['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age',
       'classif'],
      dtype='object')
Variables with True are the ones selected: [ True  True False False False
True  True False]
Ranking of variables by importance to the Logistic Regression Model:  [1 1
2 5 4 1 1 3]
Despite the fact we did this grading in the features, I am still using all
of them, as it is a small dataset
```

Cross Validation and Model Selection

In [19]:

```
# Import modules
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

In [46]:

```
# Creating a list of models, results and names
models = []
models.append(('LR', LogisticRegression()))
models.append(('NB', GaussianNB()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('SVM', SVC(gamma='auto')))
results = []
names = []
```

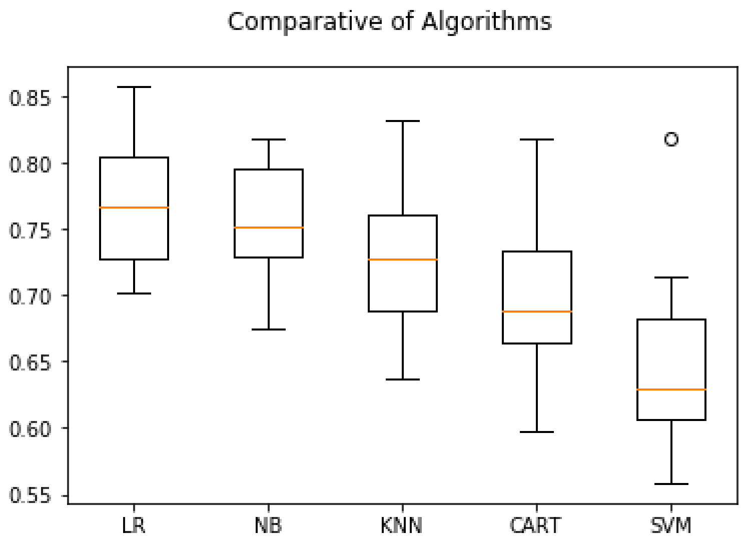
In [47]:

```
# Creating a Loop to run and compare the performance of each model
for name, model in models:
    Kfold = KFold(n_splits = 10, random_state = 7)
    cv_results = cross_val_score(model, X, y, cv = Kfold, scoring = 'accuracy')
    results.append(cv_results)
    names.append(name)
    print("%s: %f (%f)" % (name, cv_results.mean(), cv_results.std()))
```

```
LR: 0.769515 (0.048411)
NB: 0.755178 (0.042766)
KNN: 0.726555 (0.061821)
CART: 0.701709 (0.063541)
SVM: 0.651025 (0.072141)
```

In [68]:

```
# Comparing the results
fig = plt.figure()
fig.suptitle('Comparative of Algorithms')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Adjusting Parameters for Logistic Regression

In [61]:

```
# After choosing the best performer, Logistic Regression, Let's adjust the parameters.
from sklearn.model_selection import GridSearchCV
values_grid = {'penalty': ['l1','l2'], 'C': [0.001,0.01,0.1,1,10,100,1000]}
model = LogisticRegression()
grid = GridSearchCV(estimator = model, param_grid=values_grid)
grid.fit(X,y)
print('Accuracy: % .3f' %(grid.best_score_ *100))
print('Best Parameters: ', grid.best_estimator_)
```

```
Accuracy:  77.083
Best Parameters:  LogisticRegression(C=10, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

# Now that we have the best Algorithm, Let's create a train-test split, a new model and create a predictor

In [66]:

```
# Train-Test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
=7)

# Training
modelLR = LogisticRegression(C=10, max_iter=100, penalty = 'l1', solver='warn')
modelLR.fit(X_train, y_train)

# Accuracy
Acc = modelLR.score(X_test, y_test)

# Result
print("Accuracy of our Logistic Regression Model: %.3f" % (Acc.mean() * 100))
```

```
Accuracy of our Logistic Regression Model: 77.489
```