Data Wrangling: Clean, Transform, Merge, Reshape

Much of the programming work in data analysis and modeling is spent on data preparation loading, cleaning, transforming, and rearranging. Sometimes the way that data is stored in files or databases is not the way you need it for a data processing application.

pandas along with the Python standard library provide you with a high-level, flexible, and high-performance set of core manipulations and algorithms to enable you to wrangle data into the right form without much trouble.

```
import pandas as pd
import numpy as np
#Combining and Merging Data Sets

#1. pandas.merge :
    #connects rows in DataFrames based on one or more keys.


df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                 'data1': range(7)})

df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                 'data2': range(3)})

print("First Dataframe: \n",df1)

print("Second Dataframe: \n",df2)

#This is an example of a many-to-one merge situation; the data in df1 has
multiple rows labeled a and b,
#whereas df2 has only one row for each value in the key column. Calling
merge with these objects we obtain:

print("Many-to-one Merge: \n",pd.merge(df1, df2))
First Dataframe:
    data1 key
0      0   b
1      1   b
2      2   a
3      3   c
4      4   a
5      5   a
6      6   b
Second Dataframe:
    data2 key
0      0   a
1      1   b
2      2   d
Many-to-one Merge:
    data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
##Note that I didn't specify which column to join on. If not specified,
merge uses the
```

```
#overlapping column names as the keys. It’s a good practice to specify
explicitly, though:

print("Many-to-one Merge: \n",pd.merge(df1, df2, on = 'key'))
Many-to-one Merge:
    data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
#Many-to-many merges

df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                 'data1': range(6)})
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                           'data2': range(5)})

print("First Data Frame:\n",df1)
print("First Data Frame:\n",df2)


print("Left Outer Join: \n",pd.merge(df1, df2, on='key',
                                     how='left'))
print("Inner Join: \n",pd.merge(df1, df2, on='key', how='inner'))
First Data Frame:
    data1 key
0      0   b
1      1   b
2      2   a
3      3   c
4      4   a
5      5   b
First Data Frame:
    data2 key
0      0   a
1      1   b
2      2   a
3      3   b
4      4   d
Left Outer Join:
     data1 key  data2
0       0   b    1.0
1       0   b    3.0
2       1   b    1.0
3       1   b    3.0
4       2   a    0.0
5       2   a    2.0
6       3   c    NaN
7       4   a    0.0
8       4   a    2.0
9       5   b    1.0
10      5   b    3.0
Inner Join:
    data1 key  data2
0      0   b      1
1      0   b      3
2      1   b      1
3      1   b      3
4      5   b      1
```

```
5       5   b       3
6       2   a       0
7       2   a       2
8       4   a       0
9       4   a       2
#To merge with multiple keys, pass a list of column names:

left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                  'key2': ['one', 'two', 'one'],
                  'lval': [1, 2, 3]})

right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                   'key2': ['one', 'one','one', 'two'],
                   'rval': [4, 5, 6, 7]})

pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

	key1	key2	lval	rval
0	foo	one	1.0	4.0
1	foo	one	1.0	5.0
2	foo	two	2.0	NaN
3	bar	one	3.0	6.0
4	bar	two	NaN	7.0

```
#To merge with multiple keys, when column names are not same:

left = pd.DataFrame({'key11': ['foo', 'foo', 'bar'],
                  'lval1': [1, 2, 3]})

right = pd.DataFrame({'key22': ['foo', 'foo', 'bar', 'bar'],
                   'rva2l': [4, 5, 6, 7]})

pd.merge(left,right, left_on='key11',right_on='key22',
         how='outer')
```

	key11	lval1	key22	rva2l
0	foo	1	foo	4
1	foo	1	foo	5
2	foo	2	foo	4
3	foo	2	foo	5
4	bar	3	bar	6
5	bar	3	bar	7

```
#To merge with multiple keys, when column names are not same:

left = pd.DataFrame({'key11': ['foo', 'foo', 'bar'],
                  'key21': ['one', 'two', 'one'],
                  'lval1': [1, 2, 3]})

right = pd.DataFrame({'key21': ['foo', 'foo', 'bar', 'bar'],
                   'key22': ['one', 'one','one', 'two'],
                   'rva2l': [4, 5, 6, 7]})

pd.merge(left,right, left_on=['key11', 'key21'],
         right_on=['key21', 'key22']
         , how='outer')
```

	key11	key21_x	lval1	key21_y	key22	rva2l
0	foo	one	1.0	foo	one	4.0
1	foo	one	1.0	foo	one	5.0
2	foo	two	2.0	NaN	NaN	NaN
3	bar	one	3.0	bar	one	6.0
4	NaN	NaN	NaN	bar	two	7.0

```
#2. Concatenating Along an Axis

#Another kind of data combination operation is alternatively referred to as
concatenation, binding, or stacking.
#NumPy has a concatenate function for doing this with raw NumPy arrays:

#The concat function in pandas provides you hte same functionality

s1 = pd.Series([0, 1], index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
s3 = pd.Series([5, 6], index=['f', 'g'])

print("First Series:\n",s1)
print("Second Series:\n",s2)
print("Third Series:\n",s3)

print("Concatinated Series: \n",pd.concat([s1, s2, s3]))
First Series:
 a    0
b    1
dtype: int64
Second Series:
 c    2
d    3
e    4
dtype: int64
Third Series:
 f    5
g    6
dtype: int64
Concatinated Series:
 a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
#By default concat works along axis=0, producing another Series. If you
pass axis=1, the
#result will instead be a DataFrame (axis=1 is the columns):


print("Concatinated Series on axis = 1: \n",
      pd.concat([s1, s2, s3], axis=1))
Concatinated Series on axis = 1:
      0    1    2
a  0.0  NaN  NaN
b  1.0  NaN  NaN
c  NaN  2.0  NaN
```

```
d  NaN  3.0  NaN
e  NaN  4.0  NaN
f  NaN  NaN  5.0
g  NaN  NaN  6.0
s1 = pd.Series([0, 1], index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['a', 'd', 'e'])
s3 = pd.Series([5, 6], index=['a', 'g'])

print("First Series:\n",s1)
print("Second Series:\n",s2)
print("Third Series:\n",s3)

print("Concatinated Series on axis = 1: \n",
      pd.concat([s1, s2, s3], axis=1))
First Series:
 a    0
b    1
dtype: int64
Second Series:
 a    2
d    3
e    4
dtype: int64
Third Series:
 a    5
g    6
dtype: int64
Concatinated Series on axis = 1:
      0    1    2
a  0.0  2.0  5.0
b  1.0  NaN  NaN
d  NaN  3.0  NaN
e  NaN  4.0  NaN
g  NaN  NaN  6.0
#Duplicates

data = pd.DataFrame({'k1': ['one'] * 3 + ['two'] * 4,
                  'k2': [1, 1, 2, 3, 3, 4, 4]})

print("Data Frame with duplicate values: \n",data)
Data Frame with duplicate values:
     k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
#Identifying the duplicate values:

#The DataFrame method duplicated returns a boolean Series indicating
whether each
#row is a duplicate or not:

print("Identify duplicate values: \n",data.duplicated())
Identify duplicate values:
 0    False
1     True
2    False
3    False
```

```
4     True
5    False
6     True
dtype: bool
#drop_duplicates returns a DataFrame where the duplicated array is True:

print("Drop Duplicates: \n",data.drop_duplicates())
Drop Duplicates:
     k1  k2
0  one   1
2  one   2
3  two   3
5  two   4
#Both of these methods by default consider all of the columns;
alternatively you can specify
#any subset of them to detect duplicates

print("Drop K1 Duplicate:\n ",data.drop_duplicates(['k1']))
Drop K1 Duplicate:
      k1  k2
0  one   1
3  two   3
#duplicated and drop_duplicates by default keep the first observed value
combination.
#Passing take_last=True will return the last one:

data.drop_duplicates(['k1', 'k2'], keep='last')
```

	k1	**k2**
1	one	1
2	one	2
4	two	3
6	two	4

```
#Replacing Values

#Filling in missing data with the fillna method can be thought of as a
special case of more general value replacement.

data = pd.Series([1., -999., 2., -999., -1000., 3.])

print(data)
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
#The -999 values might be sentinel values for missing data. To replace
these with NA
#values that pandas understands, we can use replace, producing a new
Series:

print("Replace -999 with NA :\n",data.replace(-999, np.nan))

print("Replace -999 and -1000 with NA :\n",data.replace([-999,-1000],
                                                        np.nan))
Replace -999 with NA :
 0       1.0
```

```
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
Replace -999 and -1000 with NA :
 0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
#To use a different replacement for each value, pass a list of substitutes:

print("Replace -999 and -1000 with NA :\n",data.replace([-999,-1000],
                                                         [np.nan,0]))
Replace -999 and -1000 with NA :
 0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
#Detecting and Filtering Outliers

np.random.seed(12345)

data = pd.DataFrame(np.random.randn(1000, 4))

print("Data Frame describe: \n",data.describe())
Data Frame describe:
                  0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
mean     -0.067684     0.067924     0.025598    -0.002298
std       0.998035     0.992106     1.006835     0.996794
min      -3.428254    -3.548824    -3.184377    -3.745356
25%      -0.774890    -0.591841    -0.641675    -0.644144
50%      -0.116401     0.101143     0.002073    -0.013611
75%       0.616366     0.780282     0.680391     0.654328
max       3.366626     2.653656     3.260383     3.927528
#Computing Indicator/Dummy Variables

#Another type of transformation for statistical modeling or machine
learning applications is
#converting a categorical variable into a “dummy” or “indicator” matrix. If
a column in a DataFrame
#has k distinct values, you would derive a matrix or DataFrame containing k
columns containing
#all 1’s and 0’s. pandas has a get_dummies function for doing this.

df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                'data1': range(6)})


print("Dummy Encoding: \n", pd.get_dummies(df['key']))
Dummy Encoding:
    a  b  c
0  0  1  0
```

```
1   0   1   0
2   1   0   0
3   0   0   1
4   1   0   0
5   0   1   0
```

GroupBy: Split, Apply, Combine

Simple aggregations can give you a flavor of your dataset, but often we would prefer to aggregate conditionally on some label or index: this is implemented in the so-called groupby operation.

```
import pandas as pd
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'data': range(6)}, columns=['key', 'data'])
df
```

	key	data
0	A	0
1	B	1
2	C	2
3	A	3
4	B	4
5	C	5

```
#he most basic split-apply-combine operation can be computed with the
groupby() method of DataFrames,
#passing the name of the desired key column:

df.groupby('key')
<pandas.core.groupby.DataFrameGroupBy object at 0x000002101CCB30B8>
```

It returned a DataFrameGroupBy object. This object is where the magic is: you can think of it as a special view of the DataFrame, which is poised to dig into the groups but does no actual computation until the aggregation is applied. This "lazy evaluation" approach means that common aggregates can be implemented very efficiently in a way that is almost transparent to the user.

To produce a result, we can apply an aggregate to this DataFrameGroupBy object, which will perform the appropriate apply/combine steps to produce the desired result:

```
df.groupby('key').sum()
```

	data
key	
A	3
B	5
C	7

```
rng = np.random.RandomState(0)
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'data1': range(6),
                   'data2': rng.randint(0, 10, 6)},
                   columns = ['key', 'data1', 'data2'])
df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

```
df.groupby('key').aggregate(['min', np.median, max])
```

	data1			data2		
	min	**median**	**max**	**min**	**median**	**max**
key						
A	0	1.5	3	3	4.0	5
B	1	2.5	4	0	3.5	7
C	2	3.5	5	3	6.0	9

```
#Another useful pattern is to pass a dictionary mapping column names to
operations to be applied on that column:

df.groupby('key').aggregate({'data1': 'min',
                              'data2': 'max'})
```

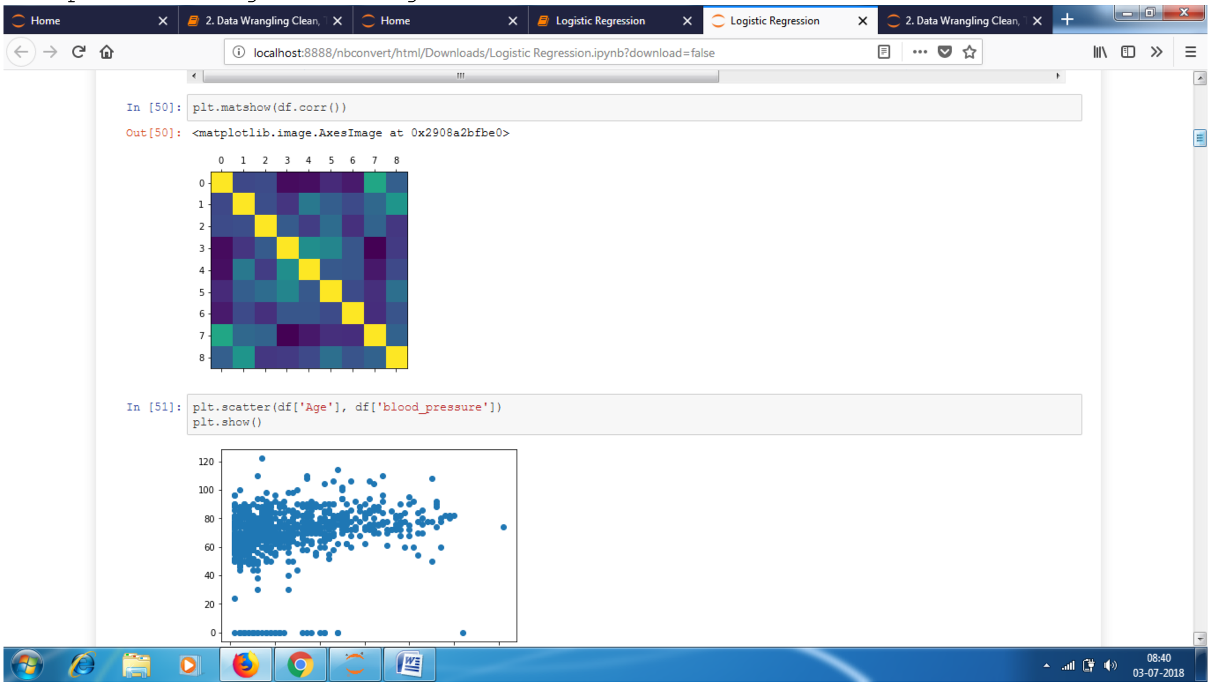
	data1	data2
key		
A	0	5
B	1	7
C	2	9

```
import pandas as pd
import pylab as plt
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

import os

os.chdir('C:\\Analytics\\Personal\\Machine Learning\\Training\\R\\Dataset')
# read the data in
df = pd.read_csv("diabetes.csv")
print(df.columns.values)
['times_pregnant' 'Plasma_glucose_concentration_2 hr' 'blood_pressure'
 ' Triceps_skin_fold_thickness ' ' Hr2_serum_insulin' 'BOI'
 ' Diabetes_pedigree_function' 'Age' 'Class']
print("Summary Statistics :\n",df.describe())
Summary Statistics :
        times_pregnant  Plasma_glucose_concentration_2 hr  blood_pressure  \
count      768.000000                         768.000000      768.000000
mean         3.845052                         120.894531       69.105469
std          3.369578                          31.972618       19.355807
min          0.000000                           0.000000        0.000000
25%          1.000000                          99.000000       62.000000
50%          3.000000                         117.000000       72.000000
75%          6.000000                         140.250000       80.000000
max         17.000000                         199.000000      122.000000

        Triceps_skin_fold_thickness    Hr2_serum_insulin         BOI  \
count                    768.000000           768.000000  768.000000
mean                      20.536458            79.799479   31.992578
std                       15.952218           115.244002    7.884160
min                        0.000000             0.000000    0.000000
25%                        0.000000             0.000000   27.300000
50%                       23.000000            30.500000   32.000000
75%                       32.000000           127.250000   36.600000
max                       99.000000           846.000000   67.100000

        Diabetes_pedigree_function         Age       Class
count                   768.000000  768.000000  768.000000
mean                      0.471876   33.240885    0.348958
std                       0.331329   11.760232    0.476951
min                       0.078000   21.000000    0.000000
25%                       0.243750   24.000000    0.000000
50%                       0.372500   29.000000    0.000000
75%                       0.626250   41.000000    1.000000
max                       2.420000   81.000000    1.000000
df.corr()
```

	times_pregnant	**Plasma_glucose_concentration_2 hr**	**blood_pressure**	**Triceps_skin_fold_thickness**	**Hr2_serum_insulin**	**BOI**	**Diabetes_pedigree_function**	**Age**	**Class**
times_pregnant	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Plasma_glucose_concentr	0.129459	1.000000	0.152590	0.057328	0.331357	0.2210	0.137337	0.2635	0.4665

ation_2 hr						71		14	81
blood_pressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
Triceps_skin_fold_thickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Hr2_serum_insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BOI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
Diabetes_pedigree_function	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

```
plt.matshow(df.corr())
<matplotlib.image.AxesImage at 0x2908a2bfbe0>
```



Home | 2. Data Wrangling Clean, | Home | Logistic Regression | Logistic Regression | 2. Data Wrangling Clean,

localhost:8888/nbconvert/html/Downloads/Logistic Regression.ipynb?download=false

```
In [50]: plt.matshow(df.corr())
Out[50]: <matplotlib.image.AxesImage at 0x2908a2bfbe0>
```

```
In [51]: plt.scatter(df['Age'], df['blood_pressure'])
         plt.show()
```

```
plt.scatter(df['Age'], df['blood_pressure'])
plt.show()
```

```
sum(df['Class'].isnull())
0
df.isnull()
```

	times_pregnant	Plasma_glucose_concentration_2 hr	blood_pressure	Triceps_skin_fold_thickness	Hr2_serum_insulin	BOI	Diabetes_pedigree_function	Age	Class
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	Fa	False	Fa	Fa

3						lse		lse	lse
14	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False	False
17	False	False	False	False	False	False	False	False	False
18	False	False	False	False	False	False	False	False	False
19	False	False	False	False	False	False	False	False	False
20	False	False	False	False	False	False	False	False	False
21	False	False	False	False	False	False	False	False	False
22	False	False	False	False	False	False	False	False	False
23	False	False	False	False	False	False	False	False	False
24	False	False	False	False	False	False	False	False	False
25	False	False	False	False	False	False	False	False	False
26	False	False	False	False	False	False	False	False	False
27	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	Fa	False	Fa	Fa

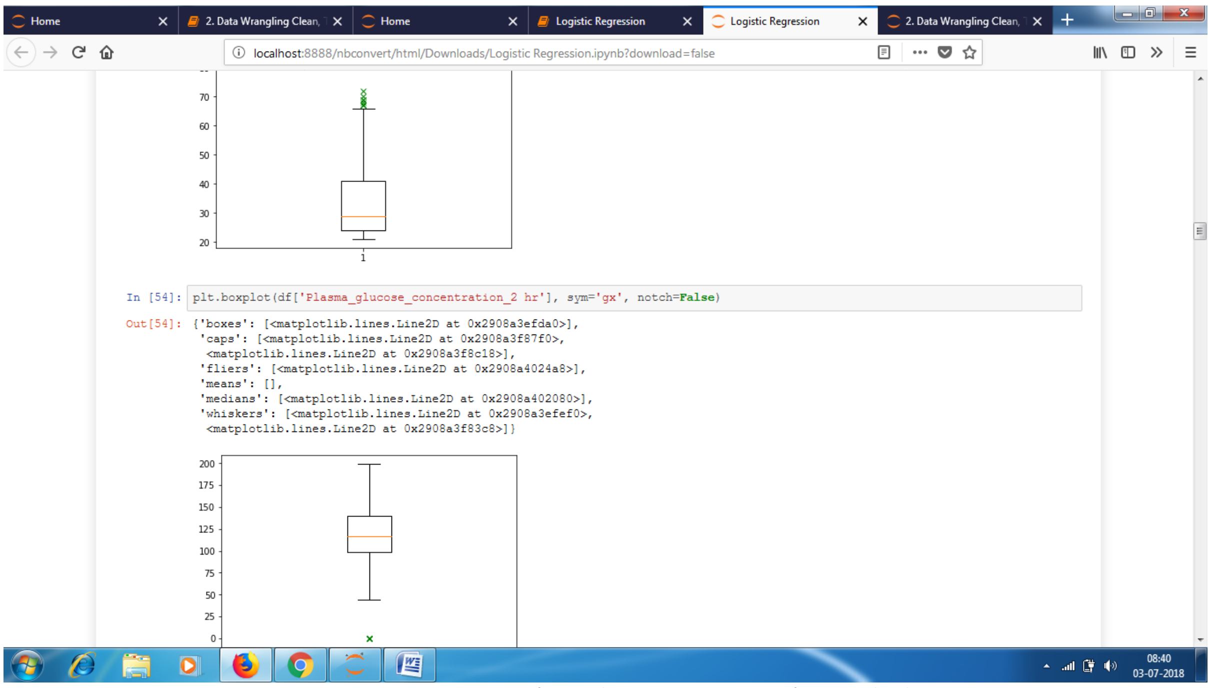
8						lse		lse	lse
29	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
738	False	False	False	False	False	False	False	False	False
739	False	False	False	False	False	False	False	False	False
740	False	False	False	False	False	False	False	False	False
741	False	False	False	False	False	False	False	False	False
742	False	False	False	False	False	False	False	False	False
743	False	False	False	False	False	False	False	False	False
744	False	False	False	False	False	False	False	False	False
745	False	False	False	False	False	False	False	False	False
746	False	False	False	False	False	False	False	False	False
747	False	False	False	False	False	False	False	False	False
748	False	False	False	False	False	False	False	False	False
749	False	False	False	False	False	False	False	False	False
750	False	False	False	False	False	False	False	False	False

751	False	False	False	False	False	False	False	False	False
752	False	False	False	False	False	False	False	False	False
753	False	False	False	False	False	False	False	False	False
754	False	False	False	False	False	False	False	False	False
755	False	False	False	False	False	False	False	False	False
756	False	False	False	False	False	False	False	False	False
757	False	False	False	False	False	False	False	False	False
758	False	False	False	False	False	False	False	False	False
759	False	False	False	False	False	False	False	False	False
760	False	False	False	False	False	False	False	False	False
761	False	False	False	False	False	False	False	False	False
762	False	False	False	False	False	False	False	False	False
763	False	False	False	False	False	False	False	False	False
764	False	False	False	False	False	False	False	False	False
765	False	False	False	False	False	False	False	False	False

766	False	False	False	False	False	False	False	False	False
767	False	False	False	False	False	False	False	False	False

768 rows × 9 columns

```
plt.boxplot(df['Age'], sym='gx', notch=False)
{'boxes': [<matplotlib.lines.Line2D at 0x2908a390a90>],
 'caps': [<matplotlib.lines.Line2D at 0x2908a3984e0>,
  <matplotlib.lines.Line2D at 0x2908a398908>],
 'fliers': [<matplotlib.lines.Line2D at 0x2908a3a1198>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x2908a398d30>],
 'whiskers': [<matplotlib.lines.Line2D at 0x2908a390be0>,
  <matplotlib.lines.Line2D at 0x2908a3980b8>]}

plt.boxplot(df['Plasma_glucose_concentration_2 hr'], sym='gx', notch=False)
{'boxes': [<matplotlib.lines.Line2D at 0x2908a3efda0>],
 'caps': [<matplotlib.lines.Line2D at 0x2908a3f87f0>,
  <matplotlib.lines.Line2D at 0x2908a3f8c18>],
 'fliers': [<matplotlib.lines.Line2D at 0x2908a4024a8>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x2908a402080>],
 'whiskers': [<matplotlib.lines.Line2D at 0x2908a3efef0>,
  <matplotlib.lines.Line2D at 0x2908a3f83c8>]}
```



```
In [54]: plt.boxplot(df['Plasma_glucose_concentration_2 hr'], sym='gx', notch=False)

Out[54]: {'boxes': [<matplotlib.lines.Line2D at 0x2908a3efda0>],
          'caps': [<matplotlib.lines.Line2D at 0x2908a3f87f0>,
           <matplotlib.lines.Line2D at 0x2908a3f8c18>],
          'fliers': [<matplotlib.lines.Line2D at 0x2908a4024a8>],
          'means': [],
          'medians': [<matplotlib.lines.Line2D at 0x2908a402080>],
          'whiskers': [<matplotlib.lines.Line2D at 0x2908a3efef0>,
           <matplotlib.lines.Line2D at 0x2908a3f83c8>]}
```

```
x = df['Plasma_glucose_concentration_2 hr'].quantile([0.25,
                                                      0.5,0.75])

print(x.values)
print(x.index)
print(x[.25])
print(x[.5])
print(x[.75])


[ 99.   117.   140.25]
Float64Index([0.25, 0.5, 0.75], dtype='float64')
99.0
117.0
140.25
IQR = x[.75] - x[.25]
print(IQR)

IQR15 = 1.5*IQR
IQR15
41.25
61.875
U_W = x[.75] + IQR15
L_W = x[.25] - IQR15

print("Upper Whisker :", U_W)
print("Lower Whisker :", L_W)
Upper Whisker : 202.125
Lower Whisker : 37.125
#Identify outliers
print("Upper Outliers")
df['Plasma_glucose_concentration_2 hr']
[df['Plasma_glucose_concentration_2 hr'] > U_W]

print("Lower Outliers")
df['Plasma_glucose_concentration_2 hr']
[df['Plasma_glucose_concentration_2 hr'] < L_W]
Upper Outliers
Lower Outliers
75     0
182    0
342    0
349    0
502    0
Name: Plasma_glucose_concentration_2 hr, dtype: int64
#Calculating 5 nad 95 percentile
cap = df['Plasma_glucose_concentration_2 hr'].quantile([0.05,0.95])
cap
0.05     79.0
0.95    181.0
Name: Plasma_glucose_concentration_2 hr, dtype: float64
#Treat outlier with capping and flooring

df['Plasma_glucose_concentration_2 hr']
[df['Plasma_glucose_concentration_2 hr'] < L_W] = cap[.05]
C:\Users\manish.khati\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
df['Plasma_glucose_concentration_2 hr']
[df['Plasma_glucose_concentration_2 hr'] == 0]
Series([], Name: Plasma_glucose_concentration_2 hr, dtype: int64)
df.iloc[75]
times_pregnant                        1.00
```

```
Plasma_glucose_concentration_2 hr    79.00
blood_pressure                       48.00
 Triceps_skin_fold_thickness         20.00
 Hr2_serum_insulin                    0.00
BOI                                  24.70
 Diabetes_pedigree_function           0.14
Age                                  22.00
Class                                 0.00
Name: 75, dtype: float64
plt.boxplot(df['Plasma_glucose_concentration_2 hr'], sym='gx', notch=False)
{'boxes': [<matplotlib.lines.Line2D at 0x2908a2ff160>],
 'caps': [<matplotlib.lines.Line2D at 0x2908a2ffeb8>,
  <matplotlib.lines.Line2D at 0x2908a296908>],
 'fliers': [<matplotlib.lines.Line2D at 0x2908a3573c8>],
 'means': [],
 'medians': [<matplotlib.lines.Line2D at 0x2908a29a438>],
 'whiskers': [<matplotlib.lines.Line2D at 0x2908a2ff358>,
  <matplotlib.lines.Line2D at 0x2908a2ffc88>]}

# target variable % distribution
print(df['Class'].value_counts(normalize=True))
0    0.651042
1    0.348958
Name: Class, dtype: float64
#build a quick logistic regression model and check the accuracy

X = df.iloc[:,:8] # independent variables
y = df['Class'] # dependent variables
# evaluate the model by splitting into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=0)
# instantiate a logistic regression model, and fit
model = LogisticRegression()
model = model.fit(X_train, y_train)
# predict class labels for the train set. The predict fuction converts
probability values > .5 to 1 else 0
y_pred = model.predict(X_test)
y_pred
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0], dtype=int64)
# generate class probabilities
# Notice that 2 elements will be returned in probs array,

probs = model.predict_proba(X_test)
# 1st element is probability for negative class,
# 2nd element gives probability for positive class
probs
array([[ 0.10927859,  0.89072141],
       [ 0.78972605,  0.21027395],
       [ 0.85929963,  0.14070037],
       [ 0.39128836,  0.60871164],
       [ 0.8255974 ,  0.1744026 ],
       [ 0.92353765,  0.07646235],
       [ 0.32291251,  0.67708749],
       [ 0.25669134,  0.74330866],
       [ 0.59214541,  0.40785459],
       [ 0.62429282,  0.37570718],
       [ 0.45786141,  0.54213859],
       [ 0.1044592 ,  0.8955408 ],
       [ 0.69141999,  0.30858001],
       [ 0.7763338 ,  0.2236662 ],
       [ 0.83010808,  0.16989192],
       [ 0.79492177,  0.20507823],
       [ 0.19582735,  0.80417265],
       [ 0.93029869,  0.06970131],
       [ 0.59223701,  0.40776299],
       [ 0.68591115,  0.31408885],
       [ 0.42864491,  0.57135509],
       [ 0.64675796,  0.35324204],
       [ 0.64309567,  0.35690433],
       [ 0.9036953 ,  0.0963047 ],
       [ 0.89736227,  0.10263773],
       [ 0.62023166,  0.37976834],
       [ 0.91298827,  0.08701173],
       [ 0.17810709,  0.82189291],
       [ 0.82052122,  0.17947878],
       [ 0.79428368,  0.20571632],
       [ 0.52622367,  0.47377633],
       [ 0.7213916 ,  0.2786084 ],
       [ 0.87219639,  0.12780361],
       [ 0.52874841,  0.47125159],
       [ 0.81705159,  0.18294841],
       [ 0.3377263 ,  0.6622737 ],
       [ 0.52367941,  0.47632059],
       [ 0.86912701,  0.13087299],
       [ 0.59807671,  0.40192329],
       [ 0.30428295,  0.69571705],
       [ 0.69586516,  0.30413484],
       [ 0.7858872 ,  0.2141128 ],
       [ 0.76514396,  0.23485604],
       [ 0.23706703,  0.76293297],
       [ 0.27601201,  0.72398799],
       [ 0.96562809,  0.03437191],
       [ 0.84117169,  0.15882831],
       [ 0.71594937,  0.28405063],
       [ 0.62322191,  0.37677809],
       [ 0.69504479,  0.30495521],
       [ 0.55749578,  0.44250422],
       [ 0.73760552,  0.26239448],
       [ 0.19011977,  0.80988023],
       [ 0.53559803,  0.46440197],
       [ 0.82079379,  0.17920621],
       [ 0.98741573,  0.01258427],
       [ 0.8875292 ,  0.1124708 ],
       [ 0.58893465,  0.41106535],
       [ 0.69324164,  0.30675836],
       [ 0.74394832,  0.25605168],
       [ 0.36730702,  0.63269298],
       [ 0.53267682,  0.46732318],
       [ 0.83234712,  0.16765288],
       [ 0.2815969 ,  0.7184031 ],
       [ 0.3726847 ,  0.6273153 ],
       [ 0.1623601 ,  0.8376399 ],
       [ 0.36029727,  0.63970273],
       [ 0.80412302,  0.19587698],
       [ 0.59169181,  0.40830819],
       [ 0.83918858,  0.16081142],
       [ 0.81967979,  0.18032021],
       [ 0.44720975,  0.55279025],
       [ 0.85648869,  0.14351131],
       [ 0.14312213,  0.85687787],
       [ 0.25419269,  0.74580731],
       [ 0.67701231,  0.32298769],
       [ 0.86568801,  0.13431199],
       [ 0.40564398,  0.59435602],
       [ 0.89073124,  0.10926876],
       [ 0.77335809,  0.22664191],
       [ 0.67077545,  0.32922455],
       [ 0.59961518,  0.40038482],
       [ 0.75562622,  0.24437378],
       [ 0.92028342,  0.07971658],
       [ 0.7401714 ,  0.2598286 ],
       [ 0.78267166,  0.21732834],
       [ 0.64410799,  0.35589201],
       [ 0.54073229,  0.45926771],
       [ 0.2090163 ,  0.7909837 ],
       [ 0.78782814,  0.21217186],
       [ 0.77239018,  0.22760982],
       [ 0.78718278,  0.21281722],
       [ 0.7010004 ,  0.2989996 ],
       [ 0.91892982,  0.08107018],
       [ 0.41596588,  0.58403412],
       [ 0.73851515,  0.26148485],
       [ 0.60895467,  0.39104533],
       [ 0.52933733,  0.47066267],
       [ 0.46773456,  0.53226544],
       [ 0.72152427,  0.27847573],
       [ 0.7352068 ,  0.2647932 ],
       [ 0.83548029,  0.16451971],
       [ 0.7807759 ,  0.2192241 ],
       [ 0.91557142,  0.08442858],
       [ 0.4390371 ,  0.5609629 ],
       [ 0.63970547,  0.36029453],
       [ 0.82152404,  0.17847596],
       [ 0.66152342,  0.33847658],
       [ 0.91349106,  0.08650894],
       [ 0.27980132,  0.72019868],
       [ 0.83402541,  0.16597459],
       [ 0.66092381,  0.33907619],
       [ 0.41726913,  0.58273087],
       [ 0.65812051,  0.34187949],
       [ 0.48187829,  0.51812171],
       [ 0.43938641,  0.56061359],
       [ 0.82481335,  0.17518665],
       [ 0.33213996,  0.66786004],
       [ 0.85898979,  0.14101021],
       [ 0.34326459,  0.65673541],
       [ 0.63145637,  0.36854363],
       [ 0.69477932,  0.30522068],
       [ 0.66796469,  0.33203531],
       [ 0.53097845,  0.46902155],
       [ 0.72619779,  0.27380221],
       [ 0.90341794,  0.09658206],
       [ 0.69050041,  0.30949959],
       [ 0.63384367,  0.36615633],
       [ 0.52694453,  0.47305547],
       [ 0.62094463,  0.37905537],
       [ 0.57428149,  0.42571851],
       [ 0.83644517,  0.16355483],
       [ 0.88883466,  0.11116534],
       [ 0.30409721,  0.69590279],
       [ 0.66746273,  0.33253727],
       [ 0.59750695,  0.40249305],
       [ 0.78503939,  0.21496061],
       [ 0.60488064,  0.39511936],
       [ 0.29693819,  0.70306181],
       [ 0.73331476,  0.26668524],
       [ 0.85254385,  0.14745615],
       [ 0.47188582,  0.52811418],
       [ 0.87047909,  0.12952091],
       [ 0.88678133,  0.11321867],
       [ 0.63146561,  0.36853439],
       [ 0.85382236,  0.14617764],
       [ 0.8650449 ,  0.1349551 ],
       [ 0.8439462 ,  0.1560538 ],
       [ 0.8237131 ,  0.1762869 ],
       [ 0.77246386,  0.22753614],
       [ 0.85320098,  0.14679902],
       [ 0.46713938,  0.53286062],
       [ 0.83453965,  0.16546035],
       [ 0.78265946,  0.21734054],
       [ 0.31600886,  0.68399114],
       [ 0.80678239,  0.19321761],
       [ 0.41330422,  0.58669578],
       [ 0.79454205,  0.20545795],
       [ 0.35967255,  0.64032745],
       [ 0.06695361,  0.93304639],
       [ 0.34618537,  0.65381463],
       [ 0.28133327,  0.71866673],
       [ 0.93770393,  0.06229607],
       [ 0.69421699,  0.30578301],
       [ 0.20394624,  0.79605376],
       [ 0.54616554,  0.45383446],
       [ 0.73014079,  0.26985921],
       [ 0.81645646,  0.18354354],
       [ 0.69130033,  0.30869967],
       [ 0.86964599,  0.13035401],
       [ 0.78377486,  0.21622514],
       [ 0.74550723,  0.25449277],
       [ 0.76520772,  0.23479228],
       [ 0.7502134 ,  0.2497866 ],
       [ 0.51310088,  0.48689912],
       [ 0.81912057,  0.18087943],
       [ 0.62723812,  0.37276188],
       [ 0.8507758 ,  0.1492242 ],
       [ 0.82318537,  0.17681463],
       [ 0.87873537,  0.12126463],
       [ 0.80753242,  0.19246758],
       [ 0.30564435,  0.69435565],
       [ 0.81512333,  0.18487667],
       [ 0.12839787,  0.87160213],
       [ 0.54130768,  0.45869232],
       [ 0.94243983,  0.05756017],
       [ 0.31014895,  0.68985105],
       [ 0.70286151,  0.29713849],
       [ 0.54124547,  0.45875453],
       [ 0.81514038,  0.18485962],
       [ 0.75843945,  0.24156055],
       [ 0.85466539,  0.14533461],
       [ 0.84488969,  0.15511031],
       [ 0.71051218,  0.28948782],
       [ 0.88075621,  0.11924379],
       [ 0.27690216,  0.72309784],
       [ 0.31386417,  0.68613583],
       [ 0.49730795,  0.50269205],
       [ 0.87822967,  0.12177033],
       [ 0.69840754,  0.30159246],
       [ 0.86961677,  0.13038323],
       [ 0.79262011,  0.20737989],
       [ 0.76343622,  0.23656378],
       [ 0.60836888,  0.39163112],
       [ 0.7176765 ,  0.2823235 ],
       [ 0.80565245,  0.19434755],
       [ 0.73316523,  0.26683477],
       [ 0.78141224,  0.21858776],
       [ 0.60816757,  0.39183243],
       [ 0.81227026,  0.18772974],
       [ 0.8614662 ,  0.1385338 ],
       [ 0.86718409,  0.13281591],
       [ 0.64292765,  0.35707235],
       [ 0.65264333,  0.34735667],
       [ 0.75141086,  0.24858914],
       [ 0.66434613,  0.33565387],
       [ 0.53062071,  0.46937929],
       [ 0.66292099,  0.33707901],
       [ 0.46941441,  0.53058559],
       [ 0.75534171,  0.24465829],
       [ 0.81583236,  0.18416764],
       [ 0.90430254,  0.09569746],
       [ 0.59232393,  0.40767607],
       [ 0.41446952,  0.58553048],
       [ 0.48318088,  0.51681912],
       [ 0.34831506,  0.65168494],
       [ 0.7254485 ,  0.2745515 ],
       [ 0.70155369,  0.29844631],
       [ 0.89273365,  0.10726635],
       [ 0.80289605,  0.19710395],
       [ 0.55771498,  0.44228502]])
#probability for positive class
y_pred_prob = probs[:, 1]
y_pred_prob
array([ 0.89072141,  0.21027395,  0.14070037,  0.60871164,  0.1744026 ,
        0.07646235,  0.67708749,  0.74330866,  0.40785459,  0.37570718,
        0.54213859,  0.8955408 ,  0.30858001,  0.2236662 ,  0.16989192,
        0.20507823,  0.80417265,  0.06970131,  0.40776299,  0.31408885,
        0.57135509,  0.35324204,  0.35690433,  0.0963047 ,  0.10263773,
        0.37976834,  0.08701173,  0.82189291,  0.17947878,  0.20571632,
        0.47377633,  0.2786084 ,  0.12780361,  0.47125159,  0.18294841,
        0.6622737 ,  0.47632059,  0.13087299,  0.40192329,  0.69571705,
        0.30413484,  0.2141128 ,  0.23485604,  0.76293297,  0.72398799,
        0.03437191,  0.15882831,  0.28405063,  0.37677809,  0.30495521,
        0.44250422,  0.26239448,  0.80988023,  0.46440197,  0.17920621,
        0.01258427,  0.1124708 ,  0.41106535,  0.30675836,  0.25605168,
        0.63269298,  0.46732318,  0.16765288,  0.7184031 ,  0.6273153 ,
        0.8376399 ,  0.63970273,  0.19587698,  0.40830819,  0.16081142,
        0.18032021,  0.55279025,  0.14351131,  0.85687787,  0.74580731,
        0.32298769,  0.13431199,  0.59435602,  0.10926876,  0.22664191,
        0.32922455,  0.40038482,  0.24437378,  0.07971658,  0.2598286 ,
        0.21732834,  0.35589201,  0.45926771,  0.7909837 ,  0.21217186,
        0.22760982,  0.21281722,  0.2989996 ,  0.08107018,  0.58403412,
        0.26148485,  0.39104533,  0.47066267,  0.53226544,  0.27847573,
        0.2647932 ,  0.16451971,  0.2192241 ,  0.08442858,  0.5609629 ,
        0.36029453,  0.17847596,  0.33847658,  0.08650894,  0.72019868,
        0.16597459,  0.33907619,  0.58273087,  0.34187949,  0.51812171,
        0.56061359,  0.17518665,  0.66786004,  0.14101021,  0.65673541,
        0.36854363,  0.30522068,  0.33203531,  0.46902155,  0.27380221,
        0.09658206,  0.30949959,  0.36615633,  0.47305547,  0.37905537,
        0.42571851,  0.16355483,  0.11116534,  0.69590279,  0.33253727,
        0.40249305,  0.21496061,  0.39511936,  0.70306181,  0.26668524,
        0.14745615,  0.52811418,  0.12952091,  0.11321867,  0.36853439,
        0.14617764,  0.1349551 ,  0.1560538 ,  0.1762869 ,  0.22753614,
        0.14679902,  0.53286062,  0.16546035,  0.21734054,  0.68399114,
        0.19321761,  0.58669578,  0.20545795,  0.64032745,  0.93304639,
        0.65381463,  0.71866673,  0.06229607,  0.30578301,  0.79605376,
        0.45383446,  0.26985921,  0.18354354,  0.30869967,  0.13035401,
        0.21622514,  0.25449277,  0.23479228,  0.2497866 ,  0.48689912,
        0.18087943,  0.37276188,  0.1492242 ,  0.17681463,  0.12126463,
        0.19246758,  0.69435565,  0.18487667,  0.87160213,  0.45869232,
        0.05756017,  0.68985105,  0.29713849,  0.45875453,  0.18485962,
        0.24156055,  0.14533461,  0.15511031,  0.28948782,  0.11924379,
        0.72309784,  0.68613583,  0.50269205,  0.12177033,  0.30159246,
        0.13038323,  0.20737989,  0.23656378,  0.39163112,  0.2823235 ,
        0.19434755,  0.26683477,  0.21858776,  0.39183243,  0.18772974,
        0.1385338 ,  0.13281591,  0.35707235,  0.34735667,  0.24858914,
        0.33565387,  0.46937929,  0.33707901,  0.53058559,  0.24465829,
        0.18416764,  0.09569746,  0.40767607,  0.58553048,  0.51681912,
        0.65168494,  0.2745515 ,  0.29844631,  0.10726635,  0.19710395,
        0.44228502])
# generate evaluation metrics
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
Accuracy:  0.78354978355
# extract false positive, true positive rate
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
roc_auc = metrics.auc(fpr, tpr)
print("Area under the ROC curve : %f" % roc_auc)
Area under the ROC curve : 0.838785
i = np.arange(len(tpr)) # index for df
roc = pd.DataFrame({'fpr' : pd.Series(fpr, index=i),'tpr' : pd.Series(tpr,
index = i),'1-fpr' : pd.Series(1-fpr, index = i)})

print(roc)
      1-fpr       fpr       tpr
0   1.000000  0.000000  0.013514
1   1.000000  0.000000  0.054054
2   0.993631  0.006369  0.054054
3   0.993631  0.006369  0.067568
4   0.987261  0.012739  0.067568
5   0.987261  0.012739  0.094595
6   0.980892  0.019108  0.094595
7   0.980892  0.019108  0.243243
8   0.974522  0.025478  0.243243
9   0.974522  0.025478  0.310811
10  0.968153  0.031847  0.310811
11  0.968153  0.031847  0.351351
12  0.961783  0.038217  0.351351
13  0.961783  0.038217  0.418919
```

```
14  0.949045  0.050955  0.418919
15  0.949045  0.050955  0.432432
16  0.942675  0.057325  0.432432
17  0.942675  0.057325  0.459459
18  0.936306  0.063694  0.459459
19  0.936306  0.063694  0.472973
20  0.929936  0.070064  0.472973
21  0.929936  0.070064  0.513514
22  0.917197  0.082803  0.513514
23  0.917197  0.082803  0.527027
24  0.898089  0.101911  0.527027
25  0.898089  0.101911  0.540541
26  0.872611  0.127389  0.540541
27  0.872611  0.127389  0.554054
28  0.847134  0.152866  0.554054
29  0.847134  0.152866  0.594595
..       ...       ...       ...
43  0.745223  0.254777  0.756757
44  0.738854  0.261146  0.756757
45  0.738854  0.261146  0.783784
46  0.732484  0.267516  0.783784
47  0.732484  0.267516  0.797297
48  0.675159  0.324841  0.797297
49  0.675159  0.324841  0.810811
50  0.662420  0.337580  0.810811
51  0.662420  0.337580  0.824324
52  0.643312  0.356688  0.824324
53  0.643312  0.356688  0.851351
54  0.636943  0.363057  0.851351
55  0.636943  0.363057  0.878378
56  0.630573  0.369427  0.878378
57  0.630573  0.369427  0.905405
58  0.579618  0.420382  0.905405
59  0.579618  0.420382  0.918919
60  0.573248  0.426752  0.918919
61  0.573248  0.426752  0.932432
62  0.535032  0.464968  0.932432
63  0.535032  0.464968  0.945946
64  0.528662  0.471338  0.945946
65  0.528662  0.471338  0.959459
66  0.401274  0.598726  0.959459
67  0.401274  0.598726  0.972973
68  0.382166  0.617834  0.972973
69  0.382166  0.617834  0.986486
70  0.197452  0.802548  0.986486
71  0.197452  0.802548  1.000000
72  0.000000  1.000000  1.000000

[73 rows x 3 columns]
#Which Error is Costly??
```

Rare Event or Imbalanced Dataset

Providing an equal samples of positive and negative instances to the classification algorithm will result in an optimal result. Datasets that are highly skewed toward one or more classes have proven to be a challenge.

Resampling is a common practice to address the imbalanced dataset issue.

Random under-sampling - Reduce majority class to match minority class count.

Random over-sampling - Increase minority class by randomly picking samples within minority class till counts of both class match.

Synthetic Minority Over-Sampling Technique (SMOTE) - Increase minority class by introducing synthetic examples through connecting all k (default = 5) minority class nearest neighbors using feature space similarity (Euclidean distance).

Bias and Variance

A fundamental problem with supervised learning is the bias variance trade-off. Ideally a model should have two key characteristics.

1. Sensitive enough to accurately capture the key patterns in the training dataset.
2. It should be generalized enough to work well on any unseen datasets. Unfortunately, while trying to achieve the above-mentioned first point, there is an ample chance of over-fitting to noisy or unrepresentative training data points leading to a failure of generalizing the model. On the other hand, trying to generalize a model may result in failing to capture important regularities.

Bias

If model accuracy is low on a training dataset as well as test dataset the model is said to be under-fitting or that the model has high bias. This means the model is not fitting the training dataset points well in regression or the decision boundary is not separating the classes well in classification; and two key reasons for bias are 1) not including the right features, and 2) not picking the correct order of polynomial degrees for model fitting.

To solve an under-fitting issue or to reduced bias, try including more meaningful features and try to increase the model complexity by trying higher-order polynomial fittings.

Variance

If a model is giving high accuracy on a training dataset, however on a test dataset the accuracy drops drastically, then the model is said to be over-fitting or a model that has high variance. The key reason for over-fitting is using higher-order polynomial degree (may not be required), which will fit decision boundary tools well to all data points including the noise of train dataset, instead of the underlying relationship. This will lead to a high accuracy (actual vs. predicted) in the train dataset and when applied to the test dataset, the prediction error will be high. To solve the over-fitting issue:

Try to reduce the number of features, that is, keep only the meaningful features.

Dimension reduction can eliminate noisy features, in turn, reducing the model variance.

Brining more data points to make training dataset large will also reduce variance.

Choosing right model parameters can help to reduce the bias and variance, for example.

Using right regularization parameters can decrease variance in regression-based models.

For a decision tree reducing the depth of the decision tree will reduce the variance.

K-Fold Cross-Validation

K-folds cross-validation splits the training dataset into k-folds without replacement, that is, any given data point will only be part of one of the subset, where k-1 folds are used for the model training and one fold is used for testing. The procedure is repeated k times so that we obtain k models and performance estimates.

```
from sklearn.cross_validation import cross_val_score

df = pd.read_csv("diabetes.csv")

X = df.iloc[:,:8].values # independent variables
y = df['Class'].values # dependent variables
# Normalize Data
from sklearn import preprocessing
sc = preprocessing.StandardScaler()
sc.fit(X)
X = sc.transform(X)
# evaluate the model by splitting into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=2017)
# build a decision tree classifier
from sklearn import tree
from sklearn import metrics
from sklearn.cross_validation import train_test_split
clf = tree.DecisionTreeClassifier(random_state=2017)
# evaluate the model using 10-fold cross-validation
train_scores = cross_val_score(clf, X_train, y_train,
                               scoring='accuracy', cv=5)
test_scores = cross_val_score(clf, X_test, y_test,
                              scoring='accuracy', cv=5)
print("Train Fold AUC Scores: ", train_scores)
print("Train CV AUC Score: ", train_scores.mean())
Train Fold AUC Scores:  [ 0.7037037   0.63888889  0.65420561  0.6635514
0.71028037]
Train CV AUC Score:  0.674125995154
print("\nTest Fold AUC Scores: ", test_scores)
print("Test CV AUC Score: ", test_scores.mean())
Test Fold AUC Scores:  [ 0.70212766  0.74468085  0.74468085  0.64444444
0.66666667]
Test CV AUC Score:  0.700520094563
```

Stratified K-Fold Cross-Validation

An extended cross-validation is the Stratified K-fold cross-validation, where the class proportions are preserved in each fold, leading to better bias and variance estimates

Ensemble Methods

Ensemble methods enable combining multiple model scores into a single score to create a robust generalized model. At a high level there are two types of ensemble methods.

1. Combine multiple models of similar type #### Bagging (Bootstrap aggregation)
2. Bootstrap aggregation (also known as bagging) was proposed by Leo Breiman in 1994, which is a model aggregation technique to reduce model variance. The training data is split into multiple samples with replacements called bootstrap samples. Bootstrap sample size will be the same as the original sample size, with 3/4th of the original values and replacement result in repetition of values
3.
4. Independent models on each of the bootstrap samples are built, and the average of the predictions for regression or majority vote for classification is used to create the final model.
5.
6. Random Forest
7.

Boosting

The core concept of boosting is that rather than an independent individual hypothesis, combining hypotheses in a sequential order increases the accuracy. Essentially, boosting algorithms convert the weak learners into strong learners. Boosting algorithms are well designed to address the bias problems.

At a high level the AdaBoosting (adaptive boosting) process can be divided into three steps.
1. Assign uniform weights for all data points W0(x) = 1 / N, where N is the total number of training data points.
2. At each iteration fit a classifier ym(xn) to the training data and update weights to minimize the weighted error function.
3. The final model.