Data-X Fall 2018: Homework 06

Machine Learning

Authors: Sana Iqbal (Part 1, 2, 3)

Student: Grant Pemberton

ID#: 3034347047

In this homework, you will do some exercises with prediction.

In [54]:
```
import numpy as np
import pandas as pd
```

In [55]:
```
 # machine learning libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing#need this for logreg apparently...

#import xgboost as xgb
```

Part 1

1. Read `diabetesdata.csv` file into a pandas dataframe. About the data:

1. **TimesPregnant**: Number of times pregnant
2. **glucoseLevel**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. **BP**: Diastolic blood pressure (mm Hg)
4. **insulin**: 2-Hour serum insulin (mu U/ml)
5. **BMI**: Body mass index (weight in kg/(height in m)^2)
6. **pedigree**: Diabetes pedigree function
7. **Age**: Age (years)
8. **IsDiabetic**: 0 if not diabetic or 1 if diabetic)

In [56]:
```
#Read data & print it
data = pd.read_csv('diabetesdata.csv')



data
```

Out[56]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
0	6	148.0	72	0	33.6	0.627	50.0	1
1	1	NaN	66	0	26.6	0.351	31.0	0
2	8	183.0	64	0	23.3	0.672	NaN	1
3	1	NaN	66	94	28.1	0.167	21.0	0
4	0	137.0	40	168	43.1	2.288	33.0	1
5	5	116.0	74	0	25.6	0.201	30.0	0
6	3	78.0	50	88	31.0	0.248	26.0	1
7	10	115.0	0	0	35.3	0.134	29.0	0
8	2	197.0	70	543	30.5	0.158	53.0	1
9	8	NaN	96	0	0.0	0.232	54.0	1
10	4	110.0	92	0	37.6	0.191	NaN	0
11	10	168.0	74	0	38.0	0.537	34.0	1
12	10	139.0	80	0	27.1	1.441	57.0	0
13	1	NaN	60	846	30.1	0.398	59.0	1
14	5	166.0	72	175	25.8	0.587	51.0	1
15	7	100.0	0	0	30.0	0.484	32.0	1
16	0	NaN	84	230	45.8	0.551	31.0	1
17	7	107.0	74	0	29.6	0.254	31.0	1
18	1	103.0	30	83	43.3	0.183	33.0	0
19	1	115.0	70	96	34.6	0.529	32.0	1
20	3	126.0	88	235	39.3	0.704	27.0	0
21	8	99.0	84	0	35.4	0.388	50.0	0
22	7	196.0	90	0	39.8	0.451	41.0	1
23	9	119.0	80	0	29.0	0.263	29.0	1
24	11	143.0	94	146	36.6	0.254	51.0	1
25	10	125.0	70	115	31.1	0.205	41.0	1
26	7	147.0	76	0	39.4	0.257	43.0	1
27	1	97.0	66	140	23.2	0.487	NaN	0
28	13	NaN	82	110	22.2	0.245	57.0	0
29	5	117.0	92	0	34.1	0.337	38.0	0
...	...	...	...	...	...	...	...	...
738	2	99.0	60	160	36.6	0.453	NaN	0

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic
739	1	102.0	74	0	39.5	0.293	42.0	1
740	11	120.0	80	150	42.3	0.785	48.0	1
741	3	102.0	44	94	30.8	0.400	26.0	0
742	1	109.0	58	116	28.5	0.219	22.0	0
743	9	140.0	94	0	32.7	0.734	45.0	1
744	13	153.0	88	140	40.6	1.174	39.0	0
745	12	100.0	84	105	30.0	0.488	46.0	0
746	1	147.0	94	0	49.3	0.358	27.0	1
747	1	81.0	74	57	46.3	1.096	32.0	0
748	3	187.0	70	200	36.4	0.408	36.0	1
749	6	162.0	62	0	24.3	0.178	50.0	1
750	4	136.0	70	0	31.2	1.182	22.0	1
751	1	121.0	78	74	39.0	0.261	28.0	0
752	3	108.0	62	0	26.0	0.223	25.0	0
753	0	181.0	88	510	43.3	0.222	26.0	1
754	8	154.0	78	0	32.4	0.443	45.0	1
755	1	128.0	88	110	36.5	1.057	37.0	1
756	7	137.0	90	0	32.0	0.391	39.0	0
757	0	123.0	72	0	36.3	0.258	52.0	1
758	1	106.0	76	0	37.5	0.197	26.0	0
759	6	190.0	92	0	35.5	0.278	66.0	1
760	2	88.0	58	16	28.4	0.766	22.0	0
761	9	170.0	74	0	44.0	0.403	43.0	1
762	9	89.0	62	0	22.5	0.142	33.0	0
763	10	101.0	76	180	32.9	0.171	63.0	0
764	2	122.0	70	0	36.8	0.340	27.0	0
765	5	121.0	72	112	26.2	0.245	30.0	0
766	1	126.0	60	0	30.1	0.349	47.0	1
767	1	93.0	70	0	30.4	0.315	23.0	0

768 rows × 8 columns

2. Calculate the percentage of NaN values in each column.

In [89]:
```
NullsPerColumn = data.isnull().sum()


#make NullsPerColumn into dataframe to make the below code work


AllsPerColumn = len(data.index)

ColPercentage = NullsPerColumn/AllsPerColumn



print (ColPercentage)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-89-d4c38c4b136c> in <module>()
      1 NullsPerColumn = data.isnull().sum()
      2 
----> 3 NullsPerColumn.dtype()
      4 
      5 #make NullsPerColumn into dataframe to make the below code work

TypeError: 'numpy.dtype' object is not callable
```

In [58]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert all(NullsPerColumn.columns == ['Percentage Null'])
assert NullsPerColumn['Percentage Null'][-2] ==  0.04296875
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-58-f427e4bb1300> in <module>()
      1 ###RUN THIS CELL BUT DO NOT ALTER IT
----> 2 assert all(NullsPerColumn.columns == ['Percentage Null'])
      3 assert NullsPerColumn['Percentage Null'][-2] ==  0.04296875

~/anaconda/envs/data-x/lib/python3.6/site-packages/pandas/core/generic.py in __getattr__(self, name)
   4366         if (name in self._internal_names_set or name in self._metadata or
   4367                 name in self._accessors):
-> 4368             return object.__getattribute__(self, name)
   4369         else:
   4370             if self._info_axis._can_hold_identifiers_and_holds_name(name):

AttributeError: 'Series' object has no attribute 'columns'
```

^This Assert Cell isn't working for me because my series has no attribute columns. I got the correct number though

...

3. Calculate the TOTAL percent of ROWS with NaN values in the dataframe (make sure values are floats).

In [59]:
```
NaN_rows = data.isnull().any(axis=1)

sum_NaN_rows = NaN_rows.sum()


PercentNull =  (sum_NaN_rows/len(data.index))


print (PercentNull*100, "% of rows contain at least 1 NaN value")
```

```
8.333333333333332 % of rows contain at least 1 NaN value
```

4. Split `data` into `train_df` and `test_df` with 15% test split.

In [60]:
```
#split values
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split( data, test_size=0.15)
```

In [61]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
np.testing.assert_almost_equal(float(len(train_df))/float(len(data)), 0.
8489583333333334, 1)
np.testing.assert_almost_equal(float(len(test_df))/float(len(data)), 0.1
5104166666666666, 1)
```

5. Replace the Nan values in `train_df` and `test_df` with the mean of EACH feature.

In [62]:
```
for column in data:
    data_mean = data[column].mean()
    print (column, data_mean)

for column in train_df:
    train_df.loc[train_df[column].isnull()] = data_mean
for column in test_df:
    test_df.loc[test_df[column].isnull()] = data_mean


#print ("OG data set is null status: \n", data.isnull().sum(), "\n")
#print ("train data set is null status: \n", train_df.isnull().sum(),
 "\n")
#print ("test data set is null status: \n", test_df.isnull().sum(),
 "\n")
```

```
TimesPregnant 3.8450520833333335
glucoseLevel 121.01634877384195
BP 69.10546875
insulin 79.79947916666667
BMI 31.992578124999977
Pedigree 0.4718763020833327
Age 33.35374149659864
IsDiabetic 0.3489583333333333
```

```
/Users/grant/anaconda/envs/data-x/lib/python3.6/site-packages/pandas/co
re/indexing.py:543: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
/Users/grant/anaconda/envs/data-x/lib/python3.6/site-packages/pandas/co
re/indexing.py:189: 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-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
/Users/grant/anaconda/envs/data-x/lib/python3.6/site-packages/ipykernel
_launcher.py:8: 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-d
ocs/stable/indexing.html#indexing-view-versus-copy

/Users/grant/anaconda/envs/data-x/lib/python3.6/site-packages/ipykernel
_launcher.py:10: 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-d
ocs/stable/indexing.html#indexing-view-versus-copy
  # Remove the CWD from sys.path while we load stuff.
```

In [14]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert sum(train_df.isnull().sum()) == 0
assert sum(test_df.isnull().sum()) == 0
```

6. Split `train_df` & `test_df` into `X_train`, `Y_train` and `X_test`, `Y_test`. `Y_train` and `Y_test` should only have the column we are trying to predict, `IsDiabetic`.

In [63]:
```
X_train = train_df.drop("IsDiabetic", axis = 1)
Y_train = train_df["IsDiabetic"].astype(bool)


X_test = test_df.drop("IsDiabetic", axis = 1)
Y_test = test_df["IsDiabetic"].astype(bool)


print (X_train.shape)
print (Y_train.shape)
print (X_test.shape)
print (Y_test.shape)
```

```
(652, 7)
(652,)
(116, 7)
(116,)
```

In [64]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert [X_train.shape, Y_train.shape, X_test.shape,Y_test.shape] == [(65
2, 7), (652,), (116, 7),(116,)]
```

7.Use this dataset to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies.

In [65]:
```
# Logistic Regression


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
logreg_train_acc = logreg.score(X_train, Y_train)
logreg_test_acc = logreg.score(X_test, Y_test)
print ('logreg training acuracy= ',logreg_train_acc)
print('logreg test accuracy= ',logreg_test_acc)
```

```
logreg training acuracy=  0.7101226993865031
logreg test accuracy=  0.7068965517241379
```

In [66]:
```
# Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
perceptron_train_acc = perceptron.score(X_train, Y_train)
perceptron_test_acc = perceptron.score(X_test, Y_test)
print ('perceptron training acuracy= ',perceptron_train_acc)
print('perceptron test accuracy= ',perceptron_test_acc)
```

```
perceptron training acuracy=  0.7070552147239264
perceptron test accuracy=  0.7327586206896551
```

```
/Users/grant/anaconda/envs/data-x/lib/python3.6/site-packages/sklearn/l
inear_model/stochastic_gradient.py:128: FutureWarning: max_iter and tol
parameters have been added in <class 'sklearn.linear_model.perceptron.P
erceptron'> in 0.19. If both are left unset, they default to max_iter=5
and tol=None. If tol is not None, max_iter defaults to max_iter=1000. F
rom 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [67]:
```
# Adaboost

adaboost = AdaBoostClassifier()
adaboost.fit(X_train, Y_train)
adaboost_train_acc = adaboost.score(X_train, Y_train)
adaboost_test_acc = adaboost.score(X_test, Y_test)
print ('adaboost training acuracy= ',adaboost_train_acc)
print('adaboost test accuracy= ',adaboost_test_acc)
```

```
adaboost training acuracy=  0.8251533742331288
adaboost test accuracy=  0.8275862068965517
```

In [68]:
```
# Random Forest

random_forest = RandomForestClassifier()
random_forest.fit(X_train, Y_train)
random_forest_train_acc = random_forest.score(X_train, Y_train)
random_forest_test_acc = random_forest.score(X_test, Y_test)
print('random_forest training acuracy= ',random_forest_train_acc)
print('random_forest test accuracy= ',random_forest_test_acc)
```

```
random_forest training acuracy=  0.9831288343558282
random_forest test accuracy=  0.7758620689655172
```

8. Is mean imputation is the best type of imputation to use? Why or why not? What are some other ways to impute the data?

In short, no.

Mean imputation does not capture the correlation between variables very well. For example: if we just put the mean blood pressure (69.10546875) for all missing values, then we could have a very old person with an inaccurate blood pressure (since the elderly (men especially) tend to have higher blood pressure). This could really mess with the entropy of our features and cause us to assume things about our dataset that aren't true. See question 2.4

Other ways to impute the data are: single regression imputation (where the guess is based on a single other feature (e.g. high age would correlate to higher blood pressure)) and multiple regression imputation (where the guess is based on multiple features (e.g. in the titanic example, Alex guessed age based on sex and socioeconomic status by using a matrix to choose the most probable age to assign to the NaN values)

...

Part 2

1.Add columns `BMI_band` & `Pedigree_band` to `Data` by cutting `BMI` & `Pedigree` into 3 intervals. PRINT the first 5 rows of`data`.

In [69]:
```
# YOUR CODE HERE


data['BMI_band'] = pd.cut(data['BMI'], bins = 3)
data['Pedigree_band'] = pd.cut(data['Pedigree'], bins = 3)

data.head()
```

Out[69]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band
0	6	148.0	72	0	33.6	0.627	50.0	1	(22.367, 44.733]
1	1	NaN	66	0	26.6	0.351	31.0	0	(22.367, 44.733]
2	8	183.0	64	0	23.3	0.672	NaN	1	(22.367, 44.733]
3	1	NaN	66	94	28.1	0.167	21.0	0	(22.367, 44.733]
4	0	137.0	40	168	43.1	2.288	33.0	1	(22.367, 44.733]

1a. Print the category intervals for `BMI_band` & `Pedigree_band`.

In [70]:
```
print('BMI_Band_Interval: ', (data['BMI_band'].unique()))
print ("\n")
print('Pedigree_Band_Interval: ', (data['Pedigree_band'].unique()))

#The .unique() function tells you a lot about the dataset
```

```
BMI_Band_Interval:  [(22.367, 44.733], (-0.0671, 22.367], (44.733, 67.
1]]
Categories (3, interval[float64]): [(-0.0671, 22.367] < (22.367, 44.73
3] < (44.733, 67.1]]


Pedigree_Band_Interval:  [(0.0757, 0.859], (1.639, 2.42], (0.859, 1.63
9]]
Categories (3, interval[float64]): [(0.0757, 0.859] < (0.859, 1.639] <
(1.639, 2.42]]
```

2. Group `data` by `Pedigree_band` & determine ratio of diabetic in each band.

In [71]:
```
# YOUR CODE HERE

pedigree_DiabeticRatio = data.groupby('Pedigree_band').mean()


#should be .54 in the mid range
```

2a. Group `data` by `BMI_band` & determine ratio of diabetic in each band.

In [72]:
```
# YOUR CODE HERE

BMI_DiabeticRatio = data.groupby('BMI_band', as_index= False).mean()

#BMI_DiabeticRatio['IsDiabetic'][1]

#should be .358 in the mid range
```

In [73]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert BMI_DiabeticRatio['IsDiabetic'][1] == 0.35829662261380324
assert pedigree_DiabeticRatio['IsDiabetic'][1] == 0.5405405405405406
```

3. Convert these features - 'BP','insulin','BMI' and 'Pedigree' into categorical values by mapping different bands of values of these features to integers 0,1,2.

HINT: USE pd.cut with bin=3 to create 3 bins

In [74]:
```
# YOUR CODE HERE



data['BP'] = pd.cut(data['BP'], bins = 3, labels = [0,1,2])
data['insulin'] = pd.cut(data['insulin'], bins = 3, labels = [0,1,2])
data['BMI'] = pd.cut(data['BMI'], bins = 3, labels = [0,1,2])
data['Pedigree'] = pd.cut(data['Pedigree'], bins = 3, labels = [0,1,2])

data
```

Out[74]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_banc
0	6	148.0	1	0	1	0	50.0	1	(22.367, 44.733]
1	1	NaN	1	0	1	0	31.0	0	(22.367, 44.733]
2	8	183.0	1	0	1	0	NaN	1	(22.367, 44.733]
3	1	NaN	1	0	1	0	21.0	0	(22.367, 44.733]
4	0	137.0	0	0	1	2	33.0	1	(22.367, 44.733]
5	5	116.0	1	0	1	0	30.0	0	(22.367, 44.733]
6	3	78.0	1	0	1	0	26.0	1	(22.367, 44.733]
7	10	115.0	0	0	1	0	29.0	0	(22.367, 44.733]
8	2	197.0	1	1	1	0	53.0	1	(22.367, 44.733]
9	8	NaN	2	0	0	0	54.0	1	(-0.0671, 22.367]
10	4	110.0	2	0	1	0	NaN	0	(22.367, 44.733]
11	10	168.0	1	0	1	0	34.0	1	(22.367, 44.733]
12	10	139.0	1	0	1	1	57.0	0	(22.367, 44.733]
13	1	NaN	1	2	1	0	59.0	1	(22.367, 44.733]
14	5	166.0	1	0	1	0	51.0	1	(22.367, 44.733]
15	7	100.0	0	0	1	0	32.0	1	(22.367, 44.733]
16	0	NaN	2	0	2	0	31.0	1	(44.733, 67.1]
17	7	107.0	1	0	1	0	31.0	1	(22.367, 44.733]
18	1	103.0	0	0	1	0	33.0	0	(22.367, 44.733]

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band
19	1	115.0	1	0	1	0	32.0	1	(22.367, 44.733]
20	3	126.0	2	0	1	0	27.0	0	(22.367, 44.733]
21	8	99.0	2	0	1	0	50.0	0	(22.367, 44.733]
22	7	196.0	2	0	1	0	41.0	1	(22.367, 44.733]
23	9	119.0	1	0	1	0	29.0	1	(22.367, 44.733]
24	11	143.0	2	0	1	0	51.0	1	(22.367, 44.733]
25	10	125.0	1	0	1	0	41.0	1	(22.367, 44.733]
26	7	147.0	1	0	1	0	43.0	1	(22.367, 44.733]
27	1	97.0	1	0	1	0	NaN	0	(22.367, 44.733]
28	13	NaN	2	0	0	0	57.0	0	(-0.0671, 22.367]
29	5	117.0	2	0	1	0	38.0	0	(22.367, 44.733]
...	...	...	...	...	...	...	...	...	...
738	2	99.0	1	0	1	0	NaN	0	(22.367, 44.733]
739	1	102.0	1	0	1	0	42.0	1	(22.367, 44.733]
740	11	120.0	1	0	1	0	48.0	1	(22.367, 44.733]
741	3	102.0	1	0	1	0	26.0	0	(22.367, 44.733]
742	1	109.0	1	0	1	0	22.0	0	(22.367, 44.733]
743	9	140.0	2	0	1	0	45.0	1	(22.367, 44.733]
744	13	153.0	2	0	1	1	39.0	0	(22.367, 44.733]

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_ban
745	12	100.0	2	0	1	0	46.0	0	(22.367, 44.733]
746	1	147.0	2	0	2	0	27.0	1	(44.733, 67.1]
747	1	81.0	1	0	2	1	32.0	0	(44.733, 67.1]
748	3	187.0	1	0	1	0	36.0	1	(22.367, 44.733]
749	6	162.0	1	0	1	0	50.0	1	(22.367, 44.733]
750	4	136.0	1	0	1	1	22.0	1	(22.367, 44.733]
751	1	121.0	1	0	1	0	28.0	0	(22.367, 44.733]
752	3	108.0	1	0	1	0	25.0	0	(22.367, 44.733]
753	0	181.0	2	1	1	0	26.0	1	(22.367, 44.733]
754	8	154.0	1	0	1	0	45.0	1	(22.367, 44.733]
755	1	128.0	2	0	1	1	37.0	1	(22.367, 44.733]
756	7	137.0	2	0	1	0	39.0	0	(22.367, 44.733]
757	0	123.0	1	0	1	0	52.0	1	(22.367, 44.733]
758	1	106.0	1	0	1	0	26.0	0	(22.367, 44.733]
759	6	190.0	2	0	1	0	66.0	1	(22.367, 44.733]
760	2	88.0	1	0	1	0	22.0	0	(22.367, 44.733]
761	9	170.0	1	0	1	0	43.0	1	(22.367, 44.733]
762	9	89.0	1	0	1	0	33.0	0	(22.367, 44.733]
763	10	101.0	1	0	1	0	63.0	0	(22.367, 44.733]

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_ban
764	2	122.0	1	0	1	0	27.0	0	(22.367, 44.733]
765	5	121.0	1	0	1	0	30.0	0	(22.367, 44.733]
766	1	126.0	1	0	1	0	47.0	1	(22.367, 44.733]
767	1	93.0	1	0	1	0	23.0	0	(22.367, 44.733]

768 rows × 10 columns

In [75]:
```
###RUN THIS CELL BUT DO NOT ALTER IT
assert sum(data['insulin'])==49
assert sum(data['BMI'])==753
assert sum(data['Pedigree'])==92
```

4. Now consider the original dataset again, instead of generalizing the NAN values with the mean of the feature we will try assigning values to NANs based on some hypothesis. For example for age we assume that the relation between BMI and BP of people is a reflection of the age group. We can have 9 types of BMI and BP relations and our aim is to find the median age of each of that group:

Your Age guess matrix will look like this:

BMI	**0**	**1**	**2**
BP			
0	a00	a01	a02
1	a10	a11	a12
2	a20	a21	a22

Create a guess_matrix for NaN values of '*Age*' (using 'BMI' and 'BP') and '*glucoseLevel*' (using 'BP' and 'Pedigree') for the given dataset and assign values accordingly to the NaNs in 'Age' or '*glucoseLevel*' .

Refer to how we guessed age in the titanic notebook in the class.

In [78]:
```
# YOUR CODE HERE


###################Age####################
guess_ages = np.zeros((3,3),dtype=int) #initialize for 3 types of BMI an
d 3 types of BP


print('Guess values of age based on BMI and BP of subject...')
for i in range(0, 3):
    for j in range(0,3):
        guess_df = data[(data['BMI'] == i)&(data['BP'] == j)]['Age'].dro
pna()

        # Extract the median age for this group
        # (less sensitive) to outliers
        age_guess = guess_df.median()

        # Convert random age float to int
        guess_ages[i,j] = int(age_guess)


print('Guess_Age table:\n',guess_ages)
print ('\nAssigning age values to NAN age values in the dataset...')

for i in range(0, 3):
    for j in range(0, 3):
        data.loc[ (data.Age.isnull()) & (data.BMI == i) \
                & (data.BP == j),'Age'] = guess_ages[i,j]


data['Age'] = data['Age'].astype(int)

##################Glucose Level#################
guess_gluc = np.zeros((3,3),dtype=int) #initialize for 3 types of BMI an
d 3 types of BP


print('Guess values of glucose based on Pedigree and BP of subject...')
for i in range(0, 3):
    for j in range(0,3):
        guess_df = data[(data['Pedigree'] == i)&(data['BP'] == j)]['gluc
oseLevel'].dropna()

        # Extract the median age for this group
        # (less sensitive) to outliers
        gluc_guess = guess_df.median()

        # Convert random age float to int
        guess_gluc[i,j] = int(gluc_guess)


print('Guess_Age table:\n',guess_gluc)
print ('\nAssigning age values to NAN age values in the dataset...')

for i in range(0, 3):
```

```
    for j in range(0, 3):
        data.loc[ (data.glucoseLevel.isnull()) & (data.Pedigree == i) \
                  & (data.BP == j),'glucoseLevel'] = guess_gluc[i,j]


data['glucoseLevel'] = data['glucoseLevel'].astype(int)


print()
print('Done!')

print (data.isnull().sum())
#check for NaNs

data.head()
```

```
Guess values of age based on BMI and BP of subject...
Guess_Age table:
 [[ 24  25 133]
 [ 29  29  37]
 [ 33  32  31]]

Assigning age values to NAN age values in the dataset...
Guess values of glucose based on Pedigree and BP of subject...
Guess_Age table:
 [[115 112 133]
 [127 115 129]
 [137 149 159]]

Assigning age values to NAN age values in the dataset...

Done!
TimesPregnant    0
glucoseLevel     0
BP               0
insulin          0
BMI              0
Pedigree         0
Age              0
IsDiabetic       0
BMI_band         0
Pedigree_band    0
dtype: int64
```

Out[78]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band
0	6	148	1	0	1	0	50	1	(22.367, 44.733]
1	1	112	1	0	1	0	112	0	(22.367, 44.733]
2	8	183	1	0	1	0	29	1	(22.367, 44.733]
3	1	112	1	0	1	0	112	0	(22.367, 44.733]
4	0	137	0	0	1	2	33	1	(22.367, 44.733]

5. Now, convert 'glucoseLevel' and 'Age' features also to categorical variables of 4 categories each. PRINT the head of `data`

In [79]:
```
# YOUR CODE HERE

data['glucoseLevel'] = pd.cut(data['glucoseLevel'], bins = 4, labels = [
0,1,2,3])

data['Age'] = pd.cut(data['Age'], bins = 4, labels = [0,1,2,3])

data.head()
```

Out[79]:

	TimesPregnant	glucoseLevel	BP	insulin	BMI	Pedigree	Age	IsDiabetic	BMI_band
0	6	2	1	0	1	0	1	1	(22.367, 44.733]
1	1	2	1	0	1	0	3	0	(22.367, 44.733]
2	8	3	1	0	1	0	0	1	(22.367, 44.733]
3	1	2	1	0	1	0	3	0	(22.367, 44.733]
4	0	2	0	0	1	2	0	1	(22.367, 44.733]

6.Use this dataset (with all features in categorical form) to train perceptron, logistic regression and random forest models using 15% test split. Report training and test accuracies.

In [83]:
```
train_df, test_df = train_test_split( data, test_size=0.15)



X_train = train_df.drop(["IsDiabetic", "BMI_band", "Pedigree_band"], axi
s = 1)
Y_train = train_df["IsDiabetic"].astype(bool)


X_test = test_df.drop(["IsDiabetic", "BMI_band", "Pedigree_band"], axis
= 1)
Y_test = test_df["IsDiabetic"].astype(bool)


X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
```

Out[83]: ((652, 7), (652,), (116, 7), (116,))

In [84]:
```
# Logistic Regression


logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
logreg_train_acc = logreg.score(X_train, Y_train)
logreg_test_acc = logreg.score(X_test, Y_test)
print ('logreg training acuracy= ',logreg_train_acc)
print('logreg test accuracy= ',logreg_test_acc)
```

```
logreg training acuracy=  0.7331288343558282
logreg test accuracy=  0.8017241379310345
```

In [85]:
```
# Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
perceptron_train_acc = perceptron.score(X_train, Y_train)
perceptron_test_acc = perceptron.score(X_test, Y_test)
print ('perceptron training acuracy= ',perceptron_train_acc)
print('perceptron test accuracy= ',perceptron_test_acc)
```

```
perceptron training acuracy=  0.6840490797546013
perceptron test accuracy=  0.6120689655172413
```

```
/Users/grant/anaconda/envs/data-x/lib/python3.6/site-packages/sklearn/l
inear_model/stochastic_gradient.py:128: FutureWarning: max_iter and tol
parameters have been added in <class 'sklearn.linear_model.perceptron.P
erceptron'> in 0.19. If both are left unset, they default to max_iter=5
and tol=None. If tol is not None, max_iter defaults to max_iter=1000. F
rom 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [86]:
```
# Random Forest
random_forest = RandomForestClassifier()
random_forest.fit(X_train, Y_train)
random_forest_train_acc = random_forest.score(X_train, Y_train)
random_forest_test_acc = random_forest.score(X_test, Y_test)
print('random_forest training acuracy= ',random_forest_train_acc)
print('random_forest test accuracy= ',random_forest_test_acc)
```

```
random_forest training acuracy=  0.8358895705521472
random_forest test accuracy=  0.7413793103448276
```

Categorizing the data didn't hurt our accuracy as much as I thought it would...interesting