EXPLORATORY ANALYSIS ON BREAST CANCER DATA SET

EXPLORATORY ANALYSIS

♥ **9** (/projects/3/like/)

PYTHON, JUPYTER NOTEBOOK, SKLEARN, PANDAS, MATPLOTLIB

MEDIUM

last hacked on **Apr 01, 2018**

(https://github.com/raviolli77/machineLearning_breastCancer_Python)

David Campos

(/david)

Raul Eulogio

(/ravi)

Origins

This data set originated in early 1990's, when Dr. William H. Wolberg was curious if he could find a way to accurately predict breast cancer diagnosis based on **FNA**'s.

The research was broken down into two parts; the extraction of the data (which we will go over) and the classficattion (if you want to read more find information on this section here (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.6745&rep=rep1&type=pdf)).

Steps Taken

- **FNA**'s were done on a total of 569 patients, once done the samples were then stained to help differentiate distinguished cell nuclei
- Samples were classified as cancer-based through biopsy and historical confirmation. Non-cancer samples were confirmed by biopsy or follow ups.
- Users then chose areas of the **FNA** with minimal overlap between nuclei; they then took scans utilizing a digital camera.
- Using a software called Xcyt , the team created approximate boundaries, which would then used a process called snakes (https://en.wikipedia.org/wiki/Active_contour_model) which converged to give the exact nuclei boundary.
- Finally, once the boundaries for the nuclei were set, calculations were made resulting in 29 features, creating this data set!

More information regarding the process can be found here (http://pages.cs.wisc.edu/~olvi/uwmp/cancer.html). I will source them at the end of the project as well, but I found these to be interesting reads especially since I've seen the data set used heavily without a lot of context as to how the data was actually extracted. More information here (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.6745&rep=rep1&type=pdf)

Table of Contents

Load Modules

We load our modules into our python environment. In my case I am employing a **Jupyter Notebook** while running inside a **virtualenv** environment.

```
%matplotlib inline

import numpy as np
import pandas as pd # Data frames
import matplotlib.pyplot as plt # Visuals
import seaborn as sns # Danker visuals
import helper_functions as hf
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve # ROC Curves
from sklearn.metrics import auc # Calculating AUC for ROC's!
from urllib.request import urlopen

pd.set_option('display.max_columns', 500)
# Included to show all the columns
# since it is a fairly large data set

plt.style.use('ggplot') # Using ggplot2 style visuals
# because that's how I learned my visuals
# and I'm sticking to it!
```

Loading Data

For this section, I'll load the data into a **Pandas** dataframe using `urlopen` from the `urllib.request` module.

Instead of downloading a **csv**, I started implementing this method(Inspired by Jason's Python Tutorials (https://github.com/JasonFreeberg/PythonTutorials)) where I grab the data straight from the UCI Machine Learning Database (https://archive.ics.uci.edu/ml/datasets.html) using an http request. This makes it easier to go about analysis from online sources and cuts out the need to download/upload a **csv** file when uploading on *GitHub*, since most files in the UCI database are easily accessible in the desired format. Finally, I created a list with the appropriate names and set them as the column names.

NOTE: The names were not documented to well so I used this analysis (https://www.kaggle.com/buddhiniw/d/uciml/breast-cancer-wisconsin-data/breast-cancer-prediction) (I will refer to it as *Buddhini W.* from now on) to grab the variable names and some other tricks that I didn't know that were available in *Python* (I will mention the use in the script!)

Finally I set the column `id_number` as the index for the dataframe.

```
UCI_data_URL = 'https://archive.ics.uci.edu/ml/machine-learning
-databases/breast-cancer-wisconsin/wdbc.data'

names = ['id_number', 'diagnosis', 'radius_mean',
         'texture_mean', 'perimeter_mean', 'area_mean',
         'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
         'concave_points_mean', 'symmetry_mean',
         'fractal_dimension_mean', 'radius_se', 'texture_se',
         'perimeter_se', 'area_se', 'smoothness_se',
         'compactness_se', 'concavity_se', 'concave_points_se',
         'symmetry_se', 'fractal_dimension_se',
         'radius_worst', 'texture_worst', 'perimeter_worst',
         'area_worst', 'smoothness_worst',
         'compactness_worst', 'concavity_worst',
         'concave_points_worst', 'symmetry_worst',
         'fractal_dimension_worst']

breast_cancer = pd.read_csv(urlopen(UCI_data_URL), names=names)

# Setting 'id_number' as our index
breast_cancer.set_index(['id_number'], inplace = True)
namesInd = names[2:] # FOR CART MODELS LATER
```

Exploratory Analysis

An important process in **Machine Learning** is doing **Exploratory Analysis** to get a *feel* for your data. Creating visuals can help people understand the data set and allow for digestable pieces of information that sometimes code and predicitive analytics wouldn't allow. Often times we will try to jump into the predictive modeling, but it helps us create narratives which will allow us to give context to people who are not as driven by data.

Its good to always output sections of your data so you can give context to the reader as to what each column looks like, as well as seeing examples of how the data is suppose to be formatted when loaded correctly. Many people run into the issue (especially if you run into a data set with poor documentation w.r.t. the column names), so its good habit to show your data during your analysis.

We use the function `head()` which is essentially the same as the `head` function in *R* if you come from an *R* background.

```
breast_cancer.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	c
id_number							
842302	1	17.99	10.38	122.80	1001.0	0.11840	0
842517	1	20.57	17.77	132.90	1326.0	0.08474	0
84300903	1	19.69	21.25	130.00	1203.0	0.10960	0
84348301	1	11.42	20.38	77.58	386.1	0.14250	0
84358402	1	20.29	14.34	135.10	1297.0	0.10030	0

More Preliminary Analysis

Much of these sections are given to give someone context to the dataset you are utilizing. Often looking at raw data will not give people the desired context, so it is important for us as data enthusiast to fill in the gaps for people who are interested in the analysis. But don't plan on running it anytime soon.

Data Frame Dimensions

Here we use the `.shape` function to give us the lengths of our data frame, where the first output is the row-length and the second output is the column-length.

Data Types

Another piece of information that is **important** is the data types of your variables in the data set.

It is often good practice to check the variable types with either the source or with your own knowledge of the data set. For this data set, we know that all variables are measurements, so they are all continous (Except **Dx**), so no further processing is needed for this step.

A common error that often happens is say a variable is *discrete* (or *categorical*), but has a numerical representation someone can easily forget the pre-processing step and do analysis on the data type as is. Since they are numeric they will be interpretted as either `int` or `float`, this isn't as big a problem in *python* as it is for *R* since most `sklearn` classifiers require numeric inputs, but still important to note.

```
print("Here's the dimensions of our data frame:\n",
     breast_cancer.shape)
print("Here's the data types of our columns:\n",
     breast_cancer.dtypes)
```

Terminal Output

```
Here's the dimensions of our data frame:
 (569, 31)
Here's the data types of our columns:
 diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave_points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave_points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave_points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

As you can see we'll be dealing with 30 independent variables that make up our feature space, all `float` types! Our next step is converting the Diagnoses into the appropriate binary representation.

Converting Diagnoses

Important when doing analysis, converting variable types to the appropriate representation. A tool is as useful as the person utilizing it, so if we enter our data incorrectly the algorithm will suffer not as a result from its capabilities, but from the human component (More on this later).

Here I converted the Dx to **binary** represenations using the `map` functionality in `pandas`. I borrowed this from *Buddhini W*. We are using a dictionary to map out this conversion:

```
{'M':1, 'B':0}
```

which then converts the previous string representations of the Dx to the **binary** representation, where 1 == **Malignant** and 0 == **Benign**.

```
# Converted to binary to help later on with models and plots
breast_cancer['diagnosis'] = breast_cancer['diagnosis']\
  .map({'M':1, 'B':0})

# Let's look at the count of the new representations of our D
x's
breast_cancer['diagnosis'].value_counts()
```

Terminal Output

```
0    357
1    212
Name: diagnosis, dtype: int64
```

Class Imbalance

The count for our Dx is important because it brings up the discussion of *Class Imbalance* within *Machine learning* and *data mining* applications.

Class Imbalance refers to when a class within a data set is outnumbered by the other class (or classes). Reading documentation online, *Class Imbalance* is present when a class populates 10-20% of the data set.

However for this data set, its pretty obvious that we don't suffer from this, but since I'm practicing my **Python**, I decided to experiment with *functions* to get better at **Python**!

NOTE: If your data set suffers from *class imbalance* I suggest reading documentation on *upsampling* and *downsampling*.

```
def print_dx_perc(data_frame, col):
    """Function used to print class distribution for our data s
et"""
    dx_vals = data_frame[col].value_counts()
    dx_vals = dx_vals.reset_index()
    # Create a function to output the percentage
    f = lambda x, y: 100 * (x / sum(y))
    for i in range(0, len(dx)):
        print('{0} accounts for {1:.2f}% of the diagnosis clas
s'\
              .format(dx[i], f(dx_vals[col].iloc[i],
                                 dx_vals[col])))
```

```
print_dx_perc(breast_cancer, 'diagnosis')
```

Terminal Output

```
Benign accounts for 62.74% of the diagnosis class
Malignant accounts for 37.26% of the diagnosis class
```

As we can see here our data set is not suffering from *class imbalance* so we can proceed with our analysis.

So I started by using the .describe() function to give some basic statistics relating to each variable. We can see there are 569 instances of each variable (which should make sense), but important to note that the distributions of the different variables have very high variance by looking at the **means** (Some can go as low as .0n while some as large as 800!)

```
breast_cancer.describe()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.1
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.0
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.0
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.0
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.0
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.1
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.3

We will discuss the high variance in the distribution of the variables later within context of the appropriate analysis. For now we move on to visual representations of our data. Still a continuation of our **Exploratory Analysis.**

Visual Exploratory Analysis

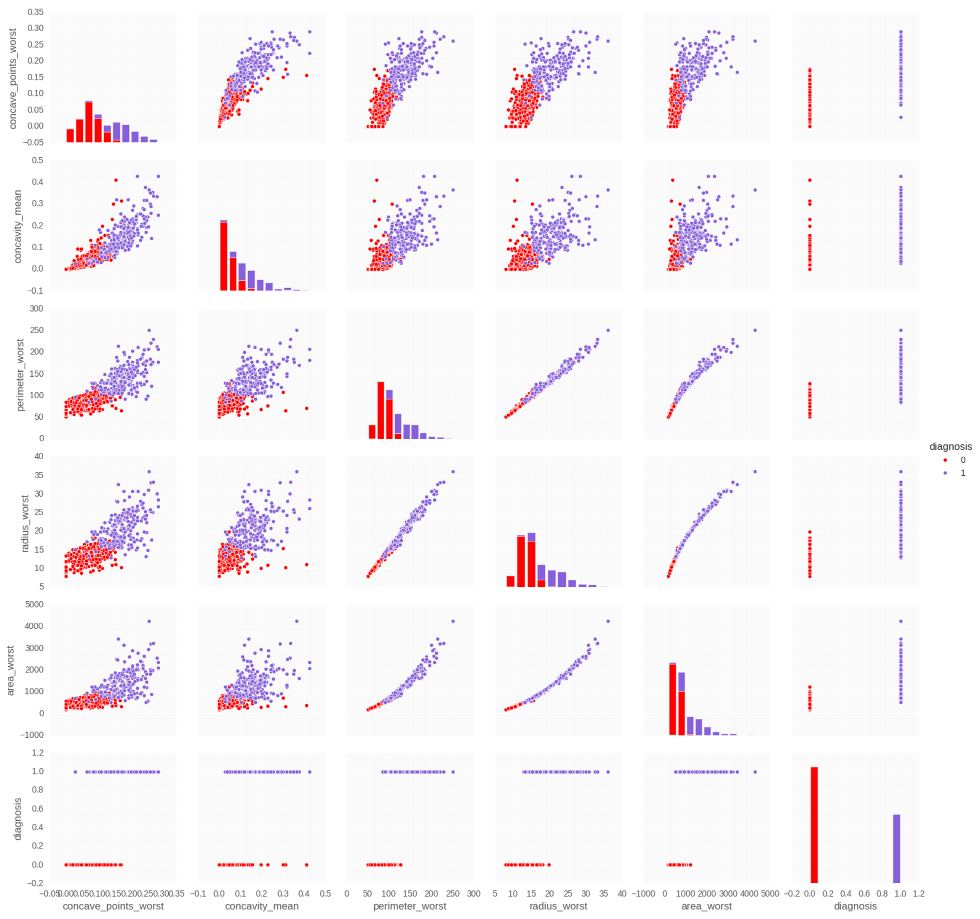
For this section we utilize the module `Seaborn` which contains many powerful statistical graphs that would have been hard to produce using `matplotlib` (My note: `matplotlib` is not the most intuitive visualizing tool in comparison to `ggplot2` in *R*, but *Python* seems like its well on its way to creating visually pleasing and intuitive plots!)

Scatterplot Matrix

For this visual I cheated by referencing some variables that were indicators of being influencial to the analysis (See **Random Forest** Section, more importantly the *variable importance* section).

```
# Scatterplot Matrix
# Variables chosen from Random Forest modeling.
cols = ['concave_points_worst', 'concavity_mean',
        'perimeter_worst', 'radius_worst',
        'area_worst', 'diagnosis']

sns.pairplot(breast_cancer,
             x_vars = cols,
             y_vars = cols,
             hue = 'diagnosis',
             palette = ('Red', '#875FDB'),
             markers=["o", "D"])
```



You see a matrix of the visual representation of the relationship between 6 variables:

- concave_points_worst
- concavity_mean
- perimeter_worst
- radius_worst
- area_worst
- diagnosis

Within each scatterplot we were able to color the two classes of **Dx**, which we can clearly see that we can easily distinguish the difference between **Malignant** and **Begnin**. As well as some variable interactions have an almost linear relationship.

Of course these are just 2-dimensional representations, but its still interesting to see how variables interact with each other in our data set.

Pearson Correlation Matrix

The next visual gives similar context that the last visual provided, and it is called the *Pearson Correlation Matrix*.

Variable correlation within a *Machine Learning* context doesn't play as an important role as say *linear regression*, but there can still be some dangers when a data set has too many correlated variables.

When two features (or more) are almost perfectly correlated in a *Machine Learning* setting then the inclusion of these features does not add addition information to your process. This then has the potential to hurt your algorithm's accuracy, since we are potentially utilizing a large feature space that can cause what is know as the Curse of Dimensionality (https://en.wikipedia.org/wiki/Curse_of_dimensionality).

Thus feature extraction would help reduce the amount of *noise* in your feature space, see *principal component analysis*, or *t-distributed stochastic neighbor embedding*.

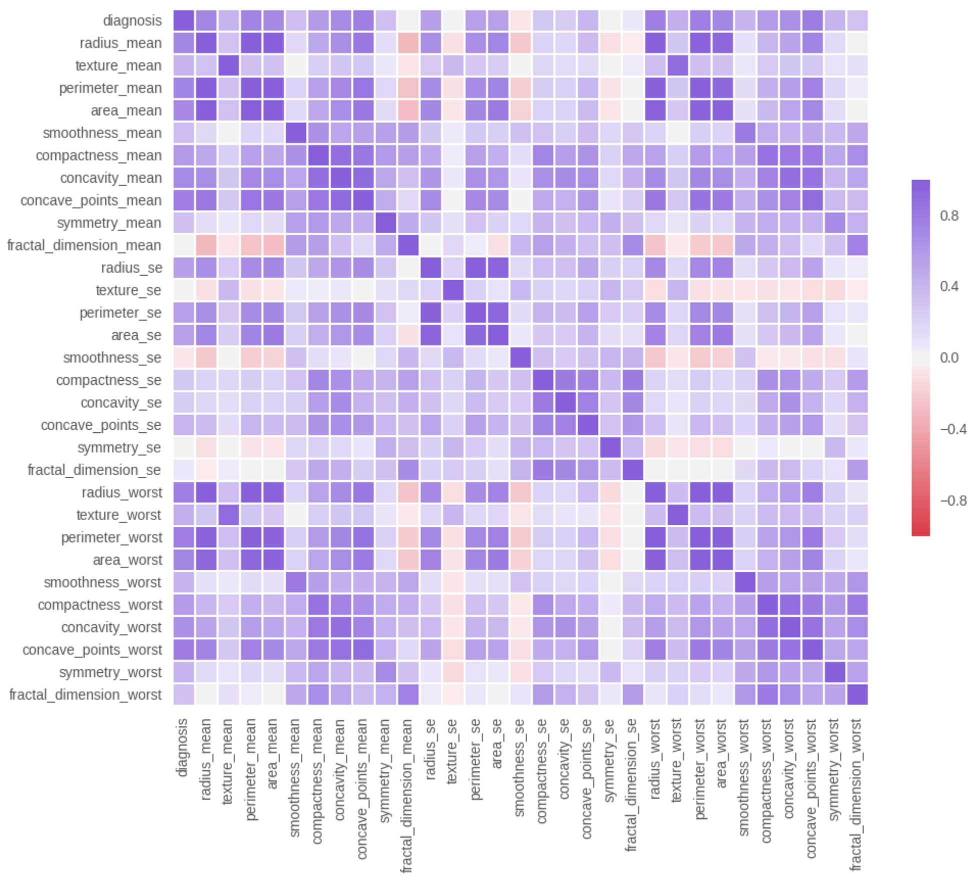
Many of our algorithms are also very computationally expensive, so utilizing a dimension reduction algorithm would also help performance and computational time.

```
corr = breast_cancer.corr(method = 'pearson') # Correlation Mat
rix

f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(10, 275, as_cmap=True)


# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr,  cmap=cmap,square=True,
            xticklabels=True, yticklabels=True,
            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)
```



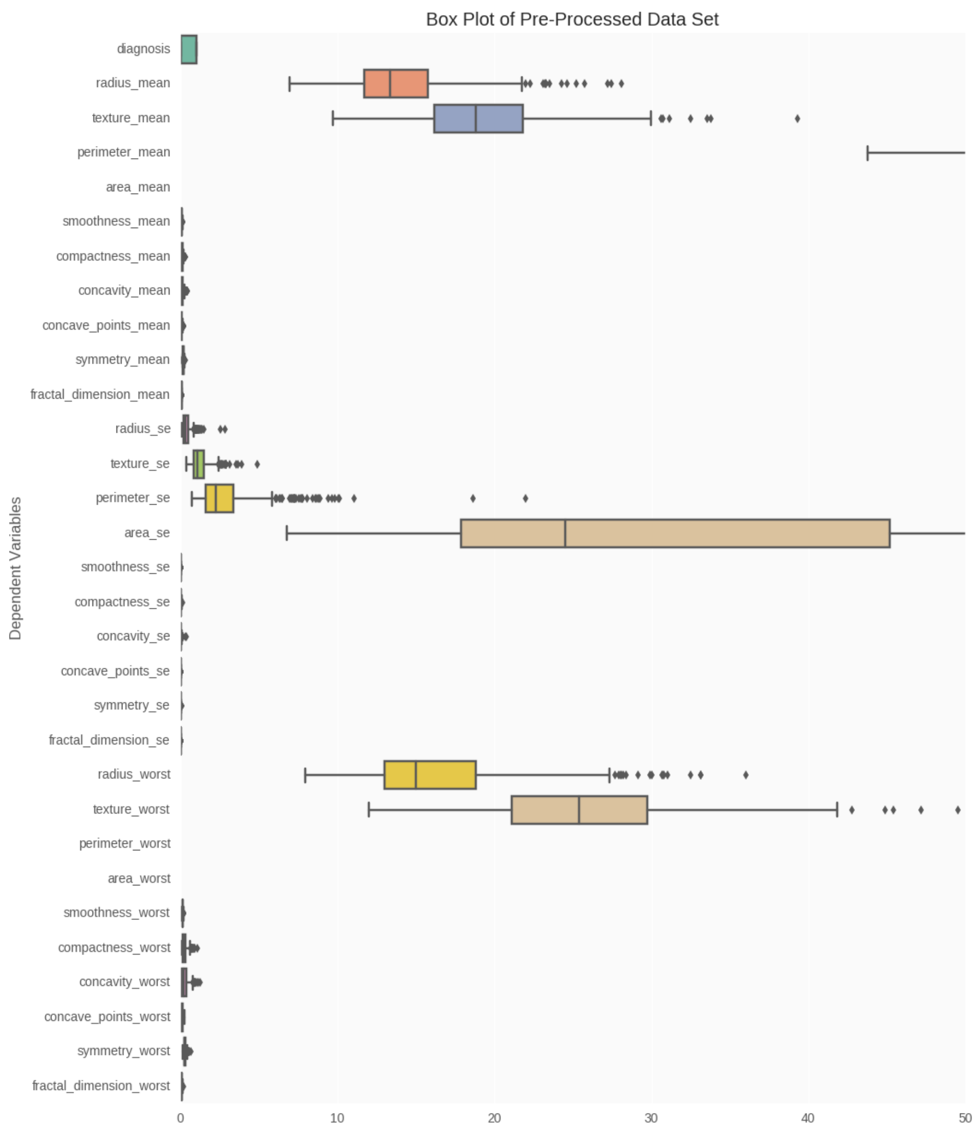
We can see that our data set contains mostly positive correlation, as well as re-iterating to us that the 5 dependent variables we featured in the *Scatterplot Matrix* have strong *correlation*. Our variables don't have too much correlation so I won't go about doing feature extraction processes like *Principal Component Analysis* (**PCA**), but you are more welcomed to do so (you will probably get better prediction estimates).

Boxplots

Next I decided to include boxplots of the data to show the high variance in the distribution of our variables. This will help drive home the point of the need to do some appropriate transformation for some models I will be employing. This is especially true for *Neural Networks*.

```
f, ax = plt.subplots(figsize=(11, 15))

ax.set_axis_bgcolor('#fafafa')
ax.set(xlim=(-.05, 50))
plt.ylabel('Dependent Variables')
plt.title("Box Plot of Pre-Processed Data Set")
ax = sns.boxplot(data = breast_cancer,
  orient = 'h',
  palette = 'Set2')
```



Not the best picture but this is a good segue into the next step in our *Machine learning* process.

Here I used a function I created in my python script. Refer to `helperFunction.py` to understand the process but I'm setting the minimum of 0 and maximum of 1 to help with some machine learning applications later on in this report. Notice that I will use this function only for the visualization of my data set. Important to note because if I were to use this transformation, during my machine learning process I would be guilty of the process called, *data leakage*, more on this later in the *neural networks* section.

```
# From helperFunction script
# From helperFunction script
def normalize_df(frame):
    '''
    Helper function to Normalize data set
    Intializes an empty data frame which
    will normalize all floats types
    and just append the non-float types
    so basically the class in our data frame
    '''
    breast_cancerNorm = pd.DataFrame()
    for item in frame:
        if item in frame.select_dtypes(include=[np.float]):
            breast_cancerNorm[item] = ((frame[item] - frame[ite
m].min()) /
            (frame[item].max() - frame[item].min()))
        else:
            breast_cancerNorm[item] = frame[item]
    return breast_cancerNorm
```

Next we utilize the function on our dataframe.

```
breast_cancerNorm = normalize_df(breast_cancer)
```

Note that we won't use this dataframe until we start fitting *Neural Networks*.

Let's try the `.describe()` function again and you'll see that all variables have a maximum of 1 which means we did our process correctly.

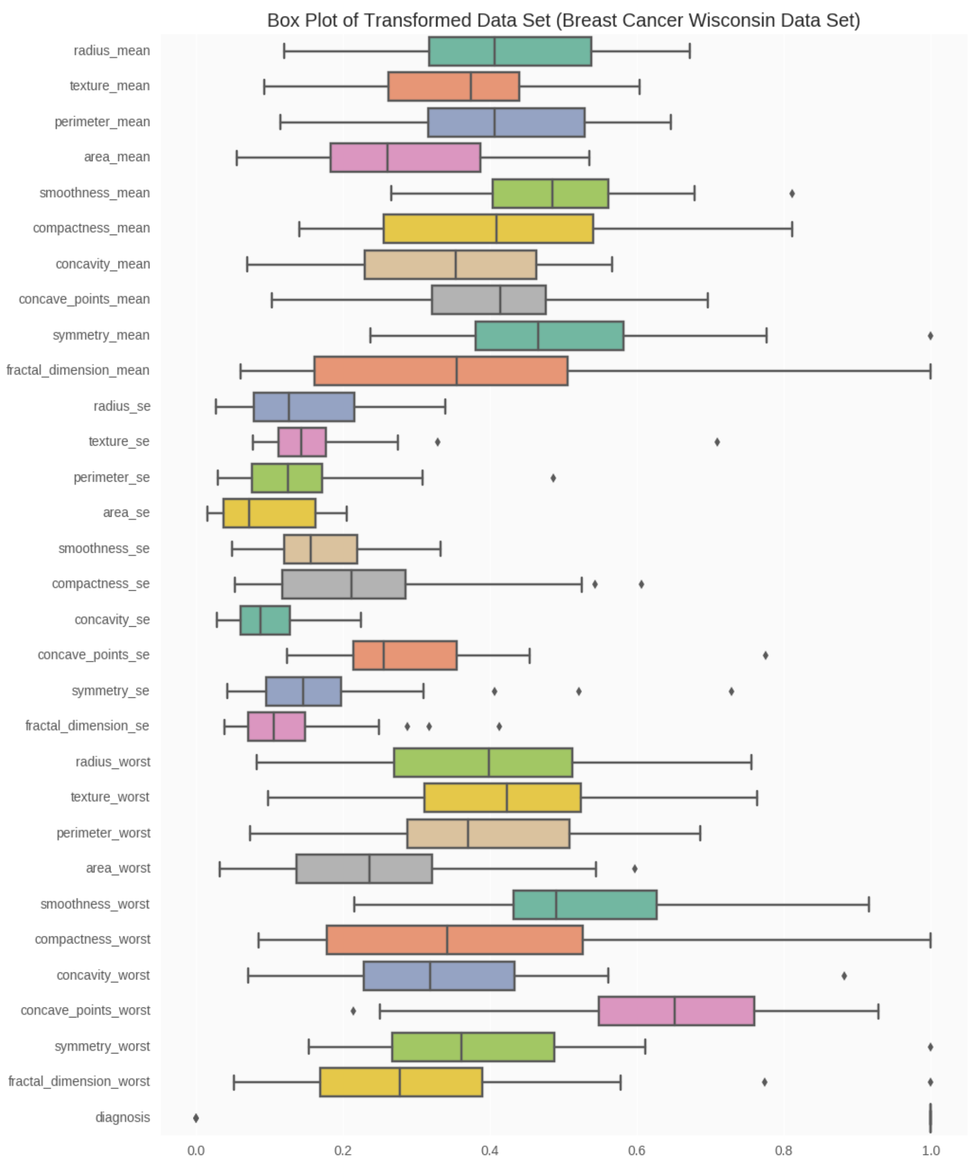
```
breast_cancerNorm.describe()
```

Box Plot of Transformed Data

Now to further illustrate the transformation let's create a *boxplot* of the scaled data set, and see the difference from our first *boxplot*.

```
f, ax = plt.subplots(figsize=(11, 15))

ax.set_axis_bgcolor('#fafafa')
plt.title("Box Plot of Transformed Data Set (Breast Cancer Wisc
onsin Data Set)")
ax.set(xlim=(-.05, 1.05))
ax = sns.boxplot(data = breast_cancerNorm[1:29],
  orient = 'h',
  palette = 'Set2')
```



There are different forms of transformations that are available for *machine learning* and I suggest you research them to gain a better understanding as to when to use a transformation. But for this project I will only employ the transformed dataframe on *Neural Networks*.

Part 2

To read about the machine learning techniques specifically random forest click here (https://www.inertia7.com/projects/95)

Sources Cited

- W. Street, Nick. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml (http://archive.ics.uci.edu/ml)]. Irvine, CA: University of California, School of Information and Computer Science.
- Waidyawansa, Buddhini. Kaggle [https://www.kaggle.com/ (https://www.kaggle.com/)]. *Using the Wisconsin breast cancer diagnostic data set for predictive analysis*. Kernel Source (https://www.kaggle.com/buddhiniw/breast-cancer-prediction)
- Zakka, Kevin. Kevin Zakka's Blog [https://kevinzakka.github.io/ (https://kevinzakka.github.io/)]. *A Complete Guide to K-Nearest-*

Neighbors with Applications in Python and R. Blog Source (https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/)

COMMENTS

Content*

Post comment

Good stuff here, I learn something new every time from your projects!

— by **njfritter** (/njfritter) | 1 year, 3 months ago

congrats Ravi, awesome project

— by **david** (/david) | 1 year, 8 months ago

keep exploring!

MYERS BRIGGS PERSONALITY TYPE NATURAL LANGUAGE PROCESSING (NLP) ANALYSIS (PART 2)

ANALZYING TWEETS LABELED WITH THEIR CORRESPONDING PERSONALITY TYPE

Python 3, Virtual Environments | hard | ♥ **2**

(/projects/110)

MACHINE LEARNING IV: K-NEAREST NEIGHBORS

A DIVE INTO THE BASICS OF THE ALGORITHM AS WELL AS VISUALIZING MULTI-DIMENSIONAL DATASETS

Python, Pandas | medium | ♥ **0**

(/projects/199)

U.S. ILLEGAL IMMIGRATION ARRESTS FROM 2000-2016

DATA VISUALIZATION USING R

R | medium | ♥ **1**

(/projects/32)

back to all projects (/projects/)

User Feedback (/projects/134)

Inertia7 is a social platform of open-source code & data science projects.

Hacked together since 2016 by David Campos (https://davidacampos.com) & Raul Eulogio (http://rauleulogio.com/) in Santa Barbara, CA inertia7

(/)