BANKRUPTCY DATA ANALYSIS

GROUP MEMBERS
NITESH KUMAR
VARUN SRIRAM
SAGAR KUMAR
ROOBIYA RAHAMATHULLA KHAN
ABHINAV TOMER
SUDEEP VENKATKRISHNA MADDELA

INTRODUCTION:

Bankruptcy is a legal proceeding involving a person or business that is unable to repay outstanding debts. Therefore, determining the risk of bankruptcy of a company plays a significant role for big institutional investors in making their financial decisions.

As part of this project we are trying to develop a model using analytical techniques which will classify the companies based on their chances of going bankrupt. For our analysis, we are using a dataset containing information about 12200 companies and their different financial ratios from the period of 1980 – 2001. These financial ratios act a proxy for determining financial health of these companies and our analytical model has tried to leverage this information.

In the following sections, we will provide step by step details of building our bankruptcy classification model.

- **DATASET DESCRIPTION:**

  Before describing our data preparations steps, we wanted a list the different columns and variables present in this bankruptcy dataset.

  As you can see, most of variables are for financial ratios that describe an individual company's financial health at that point in time. There are several types of ratios present in the dataset which involves liquidity ratio, profitability ratio, activity ratio, debt ratio, market ratio and capital budgeting ratios which can determine the current and potential performance of the company in near future. The table below contains all the ratios provided in the dataset:

label	Description	Measurement Level
cf_ta	Cash Flow/Total Assets	Continous/Numerical Value
c_s	Cash /Sales	Continous/Numerical Value
cf_td	Cash Flow/Total Debt	Continous/Numerical Value
ca_cl	Current Assets/Current Liabilities	Continous/Numerical Value
ca_ta	Current Assets/Total Assets	Continous/Numerical Value
ca_s	Current Assets/Sales	Continous/Numerical Value
ebit_ta	Earn.bef.tax.int/Total Assets	Continous/Numerical Value
re_ta	Retained Earnings/Total Assets	Continous/Numerical Value
ni_ta	Net Income/Total Assets	Continous/Numerical Value
td_ta	Total Debt/Total Assets	Continous/Numerical Value
s_ta	Sales/Total Assets	Continous/Numerical Value
wc_ta	Working Capital/Total Assets	Continous/Numerical Value
wc_s	Working Capital/Sales	Continous/Numerical Value
qa_ta	Quick Assets/Total Assets	Continous/Numerical Value
qa_cl	Quick Assets/Current Liabilities	Continous/Numerical Value
qa_s	Quick Assets/Sales	Continous/Numerical Value

mve_tc	Market Value of Equity/Total Capitalization	Continous/Numerical Value
c_cl	Cash/Current Liabilities	Continous/Numerical Value
cl_e	Current Liabilities/Equity	Continous/Numerical Value
in_s	Inventory/Sales	Continous/Numerical Value
e_s	Equity/Sales	Continous/Numerical Value
mve_td	Market Value of Equity/Total Debt	Continous/Numerical Value
ni_tc	Net Income/Total Capitalization	Continous/Numerical Value
Byear	year when the firm filed bankruptcy	Numerical Value
CONAME	firm name	Nominal Value
INAME	name of the industry to which the firm belongs	Nominal Value
year	data year (i.e., All following financial data are of this year)	Numerical Value
ID	A unique number for each company	Numerical Value
Bstatus	Bankruptcy Status	Binary Variable

But for our analysis we have used 12 significant ratios through which we can determine either the firm will bankrupt or not. So, we ended up with 14 variables in total for analysis, which contains 12 financial ratios, one target variable and one ID variable. Further we will describe each variable in detail in report below.

- **SCOPE OF PROJECT:**

We are building a classification model to predict which firms are likely to file bankruptcy in future. This model can be used by various industries to make decisions:

1. This model can be used by Financial Institutions in making decision about whether to grant a loan to a firm based on the model prediction.
2. This can also be used by Investment Firms to change their investment strategies.
3. This model can be helpful for the individual firms as they can change their future plan according to the predictions

- **DATA MANIPULATION:**

But before we could proceed with loading the input dataset to our big data environment, we had to tackle the issue of unbalanced instances of bankruptcy versus non-bankruptcy in our input dataset. As we know in real life, bankruptcy is used a last resort tool and most of the companies don't go into bankruptcy. The same pattern was shown in input dataset as well. we had around 11850 instances of non-bankruptcy versus 362 instances of companies going into bankruptcy. To handle this issue, we manually enriched the bankruptcy data and replicated bankrupt firms 9 times with an incremental ID, so have at least 1:3 ratio of bankrupt vs non-bankrupt companies in our input datasets. So now we have 11850 observations which are non-bankrupt firms and 3620 observations which are bankrupt firms. Once unbalanced data issue was handled, we had 3 major tasks in this data preparation step. These tasks were performed using pig latin scripts. Snippet of pig latin code for each step is

- **Load the input bankrupt and non-bankrupt text documents into our big data environment and merge it.**

```
data1 = LOAD '/user/admin/bdata1.txt' USING PigStorage ('\t')
    AS (ID: INT,
        cf_td: DOUBLE,
        ca_cl: DOUBLE,
        re_ta: DOUBLE,
        ni_ta: DOUBLE,
        td_ta: DOUBLE,
        s_ta: DOUBLE,
        wc_ta: DOUBLE,
        wc_s: DOUBLE,
        c_cl: DOUBLE,
        cl_e: DOUBLE,
        in_s: DOUBLE,
        mve_td: DOUBLE,
        bstatus: INT);
data2 = LOAD '/user/admin/bdata2.txt' USING PigStorage ('\t')
    AS (ID: INT,
        cf_td: DOUBLE,
        ca_cl: DOUBLE,
        re_ta: DOUBLE,
        ni_ta: DOUBLE,
        td_ta: DOUBLE,
        s_ta: DOUBLE,
        wc_ta: DOUBLE,
        wc_s: DOUBLE,
        c_cl: DOUBLE,
        cl_e: DOUBLE,
        in_s: DOUBLE,
        mve_td: DOUBLE,
        bstatus: INT);
new_data = UNION data1, data2;
a = FILTER new_data BY ID IS NOT NULL;
b = ORDER a BY ID ASC;
STORE b INTO 'user/admin/new_data1.csv' USING PigStorage(',');
```

DATA PREPROCESSING:

Once the input data was loaded and merged, the next step was to go ahead with our data preprocessing steps before partitioning it into training and test sets using PIG. The preprocessing tasks were performed in programming language Python. So, the below code shows the import of various libraries in pyhton and data import.

```
# Importing libraries used in analysis
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Importing Data file and printing first 5 observations
data = pd.read_csv('C:/MSBA/Big Data Analytics/Project/bankruptcy_data/merged_data.csv')
dataset = pd.DataFrame(data)
dataset.head()
```

	ID	cf_td	ca_cl	re_ta	ni_ta	td_ta	s_ta	wc_ta	wc_s	c_cl	cl_e	in_s	mve_td	bstatus
0	1	-0.272767	1.377092	-0.022256	-0.115008	0.408842	1.309895	0.266947	0.203793	0.009092	2.483959	0.355742	0.595134	1
1	2	-0.136488	0.106107	-0.556599	-0.219810	1.091245	0.818140	-1.159255	-1.416939	0.031919	-4.138730	0.003958	0.042403	1
2	3	-0.282272	1.071777	-0.562102	-0.282680	0.268303	1.401746	0.040862	0.029151	0.117942	3.702331	0.182852	0.111444	1
3	4	-0.303333	0.214162	-12.228137	0.106464	1.140684	2.239544	-1.730038	-0.772496	0.041451	-0.941463	0.033956	1.798000	1
4	5	-0.539683	0.827048	-3.563087	-0.198123	0.197080	1.443170	-0.138686	-0.096098	0.010403	4.470930	0.255780	6.621545	1

Once the data has been loaded in python, the next task is identifying what sort of data we have and what are the basic statistics for that data. Let's have a look at the number of observations and variables in total as below:

```
# Listing number of rows and columns parsed in whole data set
dataset.shape
```

```
(15470, 14)
```

The basic statistics of some of the variables are shown below:

```
#Summary Statistics
dataset.describe()
```

	ID	cf_td	ca_cl	re_ta	ni_ta	td_ta	s_ta	wc_ta	wc_s
count	15470.000000	15470.000000	15470.000000	15470.000000	15470.000000	15470.000000	15470.000000	15470.000000	15470.000000
mean	7735.500000	0.274949	2.183071	-0.776232	-0.218326	0.373715	1.331898	0.091327	0.883062
std	4465.948667	268.352240	3.099497	3.791542	0.861659	0.441404	1.083028	0.727624	25.135955
min	1.000000	-6010.162162	0.002231	-145.033708	-53.378049	0.000006	-0.045226	-31.035294	-1011.666667
25%	3868.250000	-0.400076	0.940752	-0.556599	-0.225469	0.132591	0.667312	-0.020148	-0.019325
50%	7735.500000	0.080942	1.534262	-0.024648	-0.015334	0.306375	1.154772	0.160445	0.126256
75%	11602.750000	0.423231	2.489409	0.188007	0.050096	0.491286	1.743567	0.360580	0.300396
max	15470.000000	22934.000000	117.058824	1.233487	10.317797	17.010989	39.911704	0.977173	1749.400000

As we can see the variables are on different scale and to have better predictor model we need to normalize the data.

- **Missing Values:**

But before normalizing, we need to further mine the data and we can see from the below output that there are no missing values in the dataset:

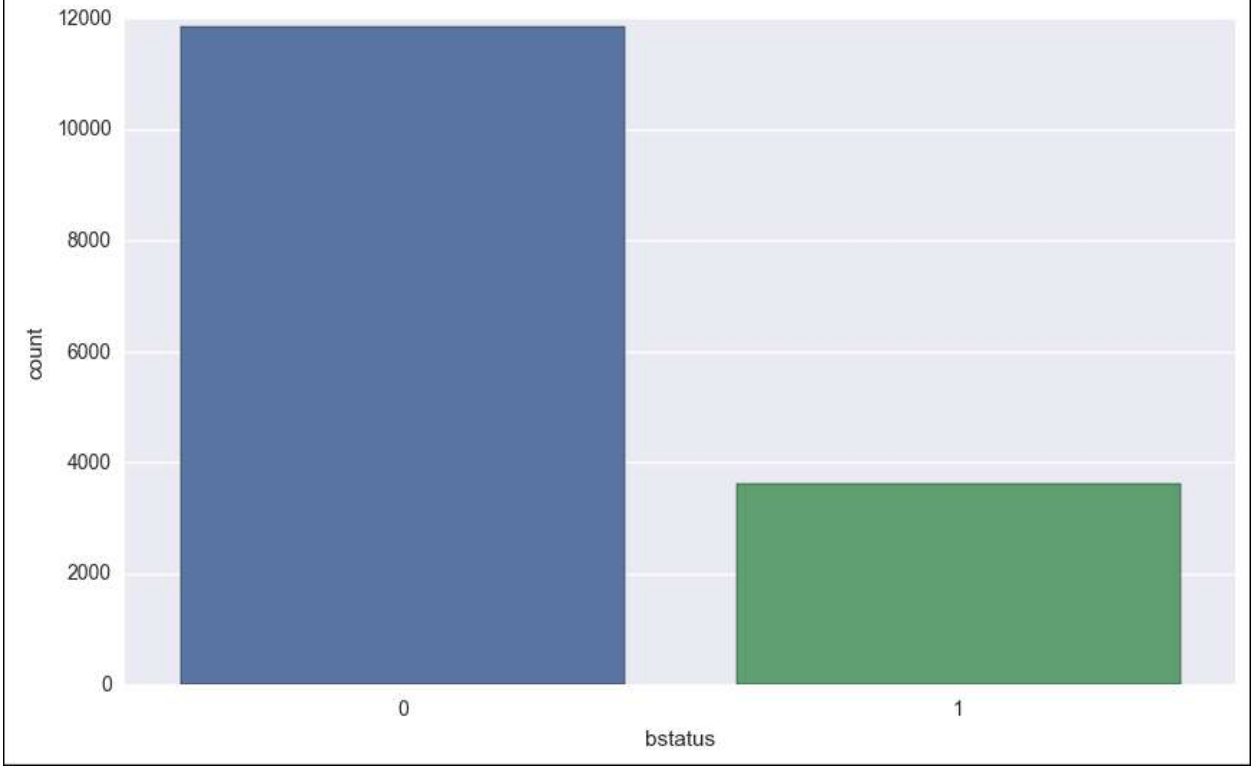
```
# Sum for number of observations with missing values
dataset.isnull().sum()
```

```
ID          0
cf_td       0
ca_cl       0
re_ta       0
ni_ta       0
td_ta       0
s_ta        0
wc_ta       0
wc_s        0
c_cl        0
cl_e        0
in_s        0
mve_td      0
bstatus     0
dtype: int64
```

To have a better understanding of the target variable we plotted it in bar graph as below:

```
# Displaying the frequency chart for output variable (bstatus = bankruptcy status)
plt.figure(figsize=(10,6))
sns.countplot(x='bstatus',data = dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ae7e3cb1d0>
```



So in order to further proceed with our analysis we have to do preprocessing on data. The major preprocessing steps are listed below:

- **Data normalization**: Before using input variables in our models, we wanted to standardize them using normalization. Below is the code snippet that handle the same:

```
data_preprocessing = dataset.copy()
data_preprocessing.drop(['ID', 'bstatus'], axis = 1, inplace = True)
```

```
normalized_data = (data_preprocessing - data_preprocessing.mean())/data_preprocessing.std()
normalized_data.head()
```

	cf_td	ca_cl	re_ta	ni_ta	td_ta	s_ta	wc_ta	wc_s	c_cl	cl_e	in_s	mve_td
0	-0.002041	-0.260035	0.198858	0.119907	0.079580	-0.020316	0.241361	-0.027024	-0.271253	0.038322	0.027705	-0.041468
1	-0.001533	-0.670097	0.057927	-0.001721	1.625561	-0.474371	-1.718719	-0.091502	-0.258073	-0.063219	-0.106014	-0.041579
2	-0.002076	-0.358540	0.056476	-0.074686	-0.238810	0.064493	-0.069356	-0.033972	-0.208404	0.057002	-0.038013	-0.041565
3	-0.002155	-0.635235	-3.020382	0.376936	1.737566	0.838063	-2.503167	-0.065864	-0.252570	-0.014198	-0.094611	-0.041226
4	-0.003036	-0.437498	-0.735019	0.023447	-0.400166	0.102742	-0.316115	-0.038955	-0.270496	0.068787	-0.010292	-0.040254

- **Correlation Analysis**: As part of our variable reduction strategy, we performed Correlation Analysis on the input dataset. We used 0.5 as our threshold value to define correlated variables. The code for the same is below:

```
# Check for Correlation among all variables
corr = normalized_data[normalized_data.columns].corr()
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr,annot = True, linewidths=0.5, ax=ax)
```

As seen from the heatmap, none of the variables are highly correlated with each other so we can use all the variables simultaneously in our analysis.

- **Handling Skewness**: To check whether the variables are normally distributed or not, we also checked for any positive and negative skewness in our dataset. So, the skewness for all the variable is shown below:

```
# Check for Skewness in the dataset
normalized_data.skew()
```

```
cf_td     66.346696
ca_cl     12.454274
re_ta    -15.962319
ni_ta    -21.545559
td_ta     11.204956
s_ta       6.614011
wc_ta    -18.431761
wc_s      33.786250
c_cl      19.156822
cl_e     -20.921551
in_s      67.215816
mve_td    65.372467
dtype: float64
```

The rule of thumb for skewness is that it should be between a range of -2 to 2 but all the variables have skewness value more than the threshhold. So, we need to transform these variables as below:

- **Positive Skewness:**
  We have transformed the variables with positive skewness by using log10 transformation. The variables for which the observation's minimum value is positive, we just take the log10 of observation's value plus one. The variables for which the observation's minimum value is negative, we take the log10 of observation's value plus absolute minimum value plus one.
- **Negative Skewness:**
  We have transformed the variables with negative skewness by using square root transformation. The variables for which the observation's minimum value is positive, we just take the square root of observation's value plus one. The variables for which the observation's minimum value is negative, we take the square root of observation's value plus absolute minimum value plus one.

The below code is used for the above-mentioned process and skewness after transformation is shows:

```
skness_positive = []
skness_negative = []
for ratio in normalized_data:
    sk = normalized_data[ratio].skew()
    if sk > 2:
        if np.min(normalized_data[ratio]) < 0:
            for value in normalized_data[ratio]:
                transformed_value = np.log10(value + np.abs(np.min(normalized_data[ratio])) + 1)
                skness_positive.append(transformed_value)
            a = pd.Series(skness_positive)
            print(ratio, a.skew())
            normalized_data.loc[:, ratio] = a
            del skness_positive[:]
        else:
            for value in normalized_data[ratio]:
                c = np.log10(value + 1)
                skness_positive.append(c)
            a = pd.Series(skness_positive)
            print(ratio, a.skew())
            normalized_data.loc[:, ratio] = a
            del skness_positive[:]
    else:
        if np.min(normalized_data[ratio]) < 0:
            for value in normalized_data[ratio]:
                transformed_value = np.sqrt(value + np.abs(np.min(normalized_data[ratio])) + 1)
                skness_negative.append(transformed_value)
            x = pd.Series(skness_negative)
            print(ratio, x.skew())
            normalized_data.loc[:, ratio] = x
            del skness_negative[:]
        else:
            for value in normalized_data[ratio]:
                y = np.sqrt(value + 1)
                skness_negative.append(y)
            x = pd.Series(skness_negative)
            print(ratio, x.skew())
            normalized_data.loc[:, ratio] = x
            del skness_negative[:]
```

```
cf_td -39.5880646944
ca_cl 2.09227220516
re_ta -23.0536305047
ni_ta -46.3436895862
td_ta 0.948979082049
s_ta 0.492737766626
wc_ta -29.9660656069
wc_s -79.4237612572
c_cl 3.85040347759
cl_e -31.480489566
in_s 17.5894382455
mve_td 17.3585261839
```

As we can see from the tranformed skewness that most of the variables have been transformed into the threshhold range and we tried to transform the remaining variables which are not in the range but this the best transformation we could get so then we used this transformed data for our further analysis. The transformed data with updated variable names looks as below:

```
normalized_data.head()
```

	trans_cf_td	trans_ca_cl	trans_re_ta	trans_ni_ta	trans_td_ta	trans_s_ta	trans_wc_ta	trans_wc_s	trans_c_cl	trans_cl_e	trans_in_s
0	1.369133	0.159440	6.264666	7.925562	0.284705	0.352420	6.634741	1.615486	0.043509	5.754275	0.492229
1	1.369142	0.014316	6.253407	7.917886	0.540605	0.254591	6.485345	1.614807	0.048657	5.745445	0.473119
2	1.369132	0.128745	6.253291	7.913277	0.206239	0.368481	6.611283	1.615413	0.067526	5.755898	0.482942
3	1.369131	0.028724	6.002233	7.941761	0.554393	0.492706	6.424582	1.615077	0.050789	5.749710	0.474782
4	1.369114	0.102472	6.189682	7.919475	0.160310	0.375534	6.592595	1.615360	0.043807	5.756921	0.486884

Then we checked for correlation between transformed variables to make sure that none of them is correlated as below:

```
# Check for Correlation among all variables
corr1 = data[data.columns].corr()
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr1,annot = True, linewidths=0.5, ax=ax)
```

Handling Outliers:

To check for the outliers in observations, we plotted box plots for all the variables which is attached in appendix. Most of the variables show that there are obvious outliers in the data but we did not remove those observations because our input variables are financial ratios which can be any continous. And for companies that are performing well, there financial ratios will be huge which might result in values being showed as outliers, so we included all the observations to get a better predictor model.

Data Partition:
After preprocessing steps were complete, we loaded our dataset in PIG to split into training and test set(70:30).

- **Partition the input dataset in individual training and test datasets.**

```
dataset = LOAD '/user/admin/data.csv' USING PigStorage (',') AS
          (ID: INT,
           bstatus: INT,
           trans_cf_td: DOUBLE,
        trans_ca_cl: DOUBLE,
        trans_re_ta: DOUBLE,
        trans_ni_ta: DOUBLE,
        trans_td_ta: DOUBLE,
        trans_s_ta: DOUBLE,
        trans_wc_ta: DOUBLE,
        trans_wc_s: DOUBLE,
        trans_c_cl: DOUBLE,
        trans_cl_e: DOUBLE,
        trans_in_s: DOUBLE,
        trans_mve_td: DOUBLE);
x = FILTER dataset BY ID IS NOT NULL;
y = FOREACH x GENERATE ID.., RANDOM() AS num;
SPLIT y INTO test IF num <= 0.30,
             train IF num > 0.30;
test_data = FOREACH test GENERATE ID, bstatus, trans_cf_td, trans_ca_cl, trans_re_ta, trans_ni_ta, trans_td_ta, trans_s_ta, trans_wc_ta, trans_wc_s, trans_c_cl, trans_cl_e, trans_in_s, trans_mve_td;
train_data = FOREACH train GENERATE ID, bstatus, trans_cf_td, trans_ca_cl, trans_re_ta, trans_ni_ta, trans_td_ta, trans_s_ta, trans_wc_ta, trans_wc_s, trans_c_cl, trans_cl_e, trans_in_s, trans_mve_td;
STORE train_data INTO '/user/admin/train_data_new.csv' USING PigStorage(',');
STORE test_data INTO '/user/admin/test_data_new.csv' USING PigStorage(',');
```

- **Splitting Training data into three subsets:**

Again, we repartitioned our training set into 3 different subsets by using PIGscript as below:

```
train = LOAD '/user/admin/train_data_new.csv' USING PigStorage (',') AS
        (ID: INT,
         bstatus: INT,
         trans_cf_td: DOUBLE,
      trans_ca_cl: DOUBLE,
      trans_re_ta: DOUBLE,
      trans_ni_ta: DOUBLE,
      trans_td_ta: DOUBLE,
      trans_s_ta: DOUBLE,
      trans_wc_ta: DOUBLE,
      trans_wc_s: DOUBLE,
      trans_c_cl: DOUBLE,
      trans_cl_e: DOUBLE,
      trans_in_s: DOUBLE,
      trans_mve_td: DOUBLE);
x = FILTER train BY ID IS NOT NULL;
y = FOREACH x GENERATE ID.., RANDOM() AS num;
SPLIT y INTO subset_1 IF num < 0.33,
                subset_2 IF num >= 0.33 AND num < 0.66,
        subset_3 IF num >= 0.66;
subset_one = FOREACH subset_1 GENERATE ID, bstatus, trans_cf_td, trans_ca_cl, trans_re_ta, trans_ni_ta, trans_td_ta, trans_s_ta, trans_wc_ta, trans_wc_s, trans_c_cl, trans_cl_e, trans_in_s, trans_mve_td;
subset_two = FOREACH subset_2 GENERATE ID, bstatus, trans_cf_td, trans_ca_cl, trans_re_ta, trans_ni_ta, trans_td_ta, trans_s_ta, trans_wc_ta, trans_wc_s, trans_c_cl, trans_cl_e, trans_in_s, trans_mve_td;
subset_three = FOREACH subset_3 GENERATE ID, bstatus, trans_cf_td, trans_ca_cl, trans_re_ta, trans_ni_ta, trans_td_ta, trans_s_ta, trans_wc_ta, trans_wc_s, trans_c_cl, trans_cl_e, trans_in_s, trans_mve_td;
STORE subset_one INTO '/user/admin/subset_1.csv' USING PigStorage(',');
STORE subset_two INTO '/user/admin/subset_2.csv' USING PigStorage(',');
STORE subset_three INTO '/user/admin/subset_3.csv' USING PigStorage(',');
```

Now, our next step was to run our analytical models on these datasets and analyze the result from different models.

ANALYTICAL MODELS :

The idea was to build the models based on training datasets and run them on test dataset and analyze the accuracy of output values. The description of different algorithms and their results are described below:

- **Logistic Regression**:

We ran our first training subset with Logistic regression classifier. The coefficients of logit model are selected using Maximum Likelihood Estimation. Here the dependent variable is binary (bstatus = 0 if non-bankrupt, bstatus = 1 if bankrupt). This binary logistic model is used to estimate the probability of a binary response based on the independent variables.

```
from patsy import dmatrices
from sklearn.linear_model import LogisticRegression
# create dataframes with an intercept column and dummy variables for
# occupation and occupation_husb
y, X = dmatrices('bstatus ~ trans_cf_td + trans_ca_cl + trans_re_ta + trans_ni_ta + trans_td_ta + trans_s_ta + trans_wc_ta + tran
                 train_one, return_type="dataframe")
print (X.columns)
```

```
Index(['Intercept', 'trans_cf_td', 'trans_ca_cl', 'trans_re_ta', 'trans_ni_ta',
       'trans_td_ta', 'trans_s_ta', 'trans_wc_ta', 'trans_wc_s', 'trans_c_cl',
       'trans_cl_e', 'trans_in_s', 'trans_mve_td'],
      dtype='object')
```

```
# flatten y into a 1-D array
y = np.ravel(y)
```

```
# instantiate a logistic regression model, and fit with X and y
model = LogisticRegression()
model = model.fit(X, y)
```

To make a good predictor for our problem, we have also checked the multicollinearity among independent variables by using variance inflation factor. If the VIF value is greater than 10, then the variables are collinear. And in our case none of the variable is collinear as below:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
[44783.524662750766, 1.0235603992335043, 2.5486668259268721, 2.7720869343761314, 1.4004309702855104, 1.6565389061363449, 1.1611
782734880625, 3.6334598529492119, 1.2975109245973755, 2.2988784978914438, 1.0044812334793976, 1.3244185251713112, 1.07735674501
42203]
```

In the above image, we have constructed the model now we should evaluate the performance of model by applying it on test data.

```
test_data = pd.read_csv("C:/MSBA/Big Data Analytics/Project/bankruptcy_data/test_data_new.csv")
test_data.head()
```

```
y_test, X_test = dmatrices('bstatus ~ trans_cf_td + trans_ca_cl + trans_re_ta + trans_ni_ta + trans_td_ta + trans_s_ta + trans_wc
                 test_data, return_type="dataframe")
print (X_test.columns)
# flatten y into a 1-D array
y_test = np.ravel(y_test)
```

```
Index(['Intercept', 'trans_cf_td', 'trans_ca_cl', 'trans_re_ta', 'trans_ni_ta',
       'trans_td_ta', 'trans_s_ta', 'trans_wc_ta', 'trans_wc_s', 'trans_c_cl',
       'trans_cl_e', 'trans_in_s', 'trans_mve_td'],
      dtype='object')
```

```
predicted_logit = model.predict(X_test)
print (predicted_logit)
```

```
[ 1.  1.  0. ...,  0.  0.  0.]
```

```
model.score(X_test, y_test)
```

```
0.78148774302620461
```

```
p_lm = pd.Series(predicted_logit)
p_lm.columns = ['p_lm']
print(p_lm)
```

```
0      1.0
1      1.0
2      0.0
3      1.0
4      0.0
5      0.0
6      0.0
7      0.0
8      0.0
9      0.0
10     0.0
11     0.0
12     1.0
13     0.0
```

We evaluated this model's performance by evaluating the **Accuracy rate** which was **78.15%**. We built the confusion matrix to describe its performance.

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
print (metrics.confusion_matrix(y_test, predicted_logit))
print (metrics.classification_report(y_test, predicted_logit))
cm_logit_test = confusion_matrix(y_test, predicted_logit)
sns.heatmap(cm_logit_test,
            xticklabels=['Non Bankrupt', 'Bankrupt'],
            yticklabels=['Non Bankrupt', 'Bankrupt'])
```

```
[[3507  124]
 [ 910  191]]
             precision    recall  f1-score   support

        0.0       0.79      0.97      0.87      3631
        1.0       0.61      0.17      0.27      1101

avg / total       0.75      0.78      0.73      4732

<matplotlib.axes._subplots.AxesSubplot at 0x1543cfd35f8>
```

We could see the **Misclassification rate** to be **0.277** ((910+124/3732)). In diagram above, precision means, when it predicts yes, how often is it correct? Which is 75% of the times. Similarly recall means, when it's yes, how often does it predict yes? Which is 78% of the times.

- **Neural Networks:**

We trained our second subset with neural networks. We used Multilayer Perceptron(MLP) which is a feedforward artificial neural network. The errors from the initial classification of the first record is fed back into the network, and used to modify the networks algorithm for further iterations.

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(12,12,12))
mlp.fit(X2,y2)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(12, 12, 12), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

```
predict_MLP = mlp.predict(X2)
```

For second subset also, none of variables were collinear as shown below:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X2.values, i) for i in range(X2.shape[1])]
print(vif)
```

```
[171670.56827126013, 1.1304371847457608, 2.530276376320487, 2.3752150831588508, 1.6327308159932641, 1.5502559010827919, 1.19148
58475064884, 2.021421326485016, 1.0971081056047536, 2.192586227291724, 1.0028828144048902, 1.1910681306449795, 1.20452535586323
01]
```

In the above image, we have constructed our second model now we should evaluate the performance of model by applying it on test data.

```
predicted_MLP = mlp.predict(X_test)
print (predicted_MLP)
mlp.score(X_test, y_test)
```

```
[ 1.  1.  0. ...,  0.  0.  0.]
0.78212172442941674
```

```
p_MLP = pd.Series(predicted_MLP)
p_MLP.columns = ['p_MLP']
print(p_MLP)
```

```
0     1.0
1     1.0
2     0.0
3     1.0
4     0.0
5     0.0
6     0.0
7     0.0
8     0.0
9     0.0
10    1.0
11    0.0
12    1.0
13    0.0
```

Our **Accuracy** was **78.21%** and **Misclassification rate** found to be **0.2178**. The Confusion matrix below measures its performance.

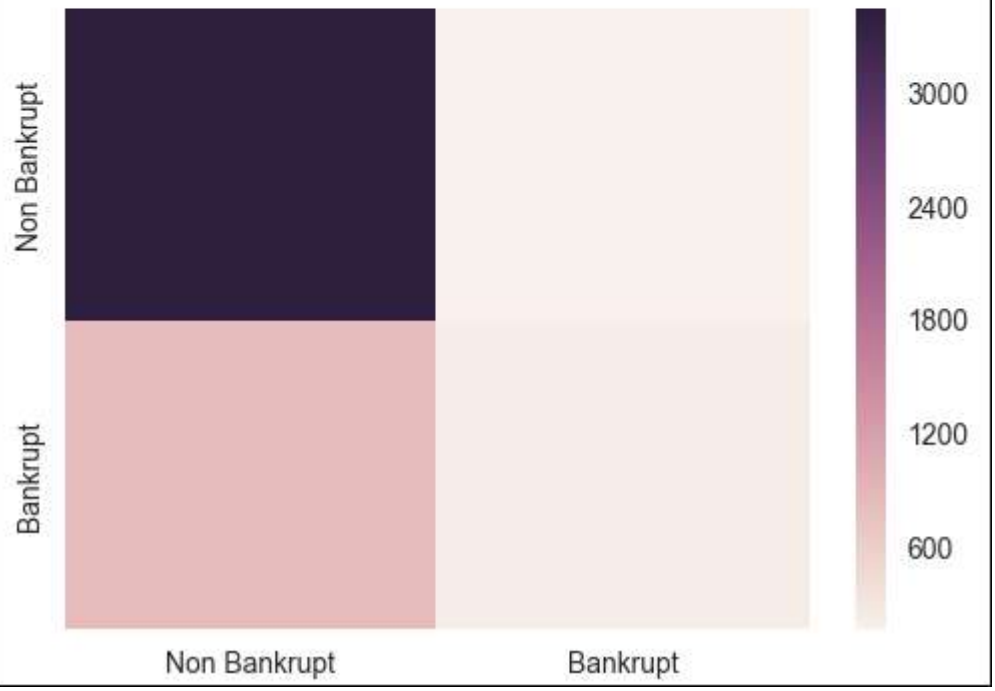
```
print (metrics.confusion_matrix(y_test, predicted_MLP))
print (metrics.classification_report(y_test, predicted_MLP))
cm_mlp_test = confusion_matrix(y_test, predicted_MLP)
sns.heatmap(cm_mlp_test,
            xticklabels=['Non Bankrupt', 'Bankrupt'],
            yticklabels=['Non Bankrupt', 'Bankrupt'])
```

```
[[3448  183]
 [ 848  253]]
             precision    recall  f1-score   support

        0.0       0.80      0.95      0.87      3631
        1.0       0.58      0.23      0.33      1101

avg / total       0.75      0.78      0.74      4732

<matplotlib.axes._subplots.AxesSubplot at 0x1543d04bd30>
```



- **Support Vector Machine:**

A Support Vector Machine(SVM) is a discriminative classifier defined by a separating hyperplane. We trained this classifier on our third subset as below:

```
y3, X3 = dmatrices('bstatus ~ trans_cf_td + trans_ca_cl + trans_re_ta + trans_ni_ta + trans_td_ta + trans_s_ta + trans_wc_ta + tr
                    train_three, return_type="dataframe")
print (X3.columns)
# flatten y into a 1-D array
y3 = np.ravel(y3)
```

```
Index(['Intercept', 'trans_cf_td', 'trans_ca_cl', 'trans_re_ta', 'trans_ni_ta',
       'trans_td_ta', 'trans_s_ta', 'trans_wc_ta', 'trans_wc_s', 'trans_c_cl',
       'trans_cl_e', 'trans_in_s', 'trans_mve_td'],
      dtype='object')
```

```
# Building a Support Vector Machine on train data
SVC = SVC()
SVC = SVC.fit(X3, y3)
```

For third subset also, none of variables were collinear as shown below:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X3.values, i) for i in range(X3.shape[1])]
print(vif)
```

```
[204323.32646983254, 1.0453744647957153, 2.1955172955373223, 2.2808182819503986, 1.4877720942620778, 1.6296849647369069, 1.1772
101366410281, 2.2854201474870184, 1.1501567988240149, 2.0058382922468239, 1.0030579144380696, 1.1773681638404501, 1.12478433192
79074]
```

In the above image, we have constructed our third model now we should evaluate the performance of model by applying it on test data.

```
predicted_SVC = SVC.predict(X_test)
print (predicted_SVC)
SVC.score(X_test, y_test)
```

```
[ 0.  0.  0. ...,  0.  0.  0.]

0.76754015215553673
```

```
p_SVC = pd.Series(predicted_SVC)
p_SVC.columns = ['p_SVC']
print(p_SVC)
```

```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
5        0.0
6        0.0
7        0.0
8        0.0
9        0.0
10       0.0
11       0.0
12       0.0
13       0.0
```

```
print (metrics.confusion_matrix(y_test, predicted_SVC))
print (metrics.classification_report(y_test, predicted_SVC))
cm_SVC_test = confusion_matrix(y_test, predicted_SVC)
sns.heatmap(cm_SVC_test,
            xticklabels=['Non Bankrupt', 'Bankrupt'],
            yticklabels=['Non Bankrupt', 'Bankrupt'])
```

```
[[3631    0]
 [1100    1]]
             precision    recall  f1-score   support

        0.0       0.77      1.00      0.87      3631
        1.0       1.00      0.00      0.00      1101

avg / total       0.82      0.77      0.67      4732


<matplotlib.axes._subplots.AxesSubplot at 0x1543cedf780>
```

We had **Accuracy** of **76.75%** and **Misclassification rate** of **0.2324**.
Based on the models above we created a new predictor which will take the predicted value for each observation as suggested by most of the models out of the three models. So we calculated predicted value based on mode of predicted value of all three models as below:

```
final_pred = pd.concat([p_lm, p_MLP, p_SVC], axis =1)
final_pred.columns = ['p_lm', 'p_MLP', 'p_SVC']
```

```
predicted_bankrupt = final_pred.mode(axis=1)
print(predicted_bankrupt)
```

But when we compared the results of three models with the fourth predicted value model its showed below results:

```
# generate evaluation metrics
print (metrics.accuracy_score(y_test, predicted_logit))
print (metrics.accuracy_score(y_test, predicted_MLP))
print (metrics.accuracy_score(y_test, predicted_SVC))
print (metrics.accuracy_score(y_test, predicted_bankrupt))
```

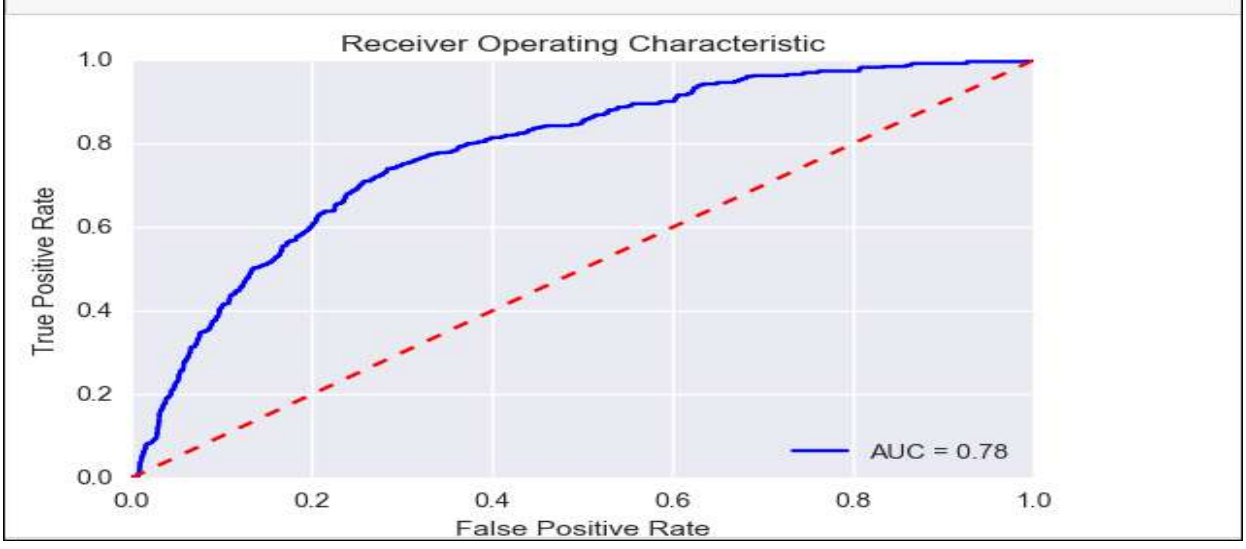
```
0.781487743026
0.782121724429
0.767540152156
0.781699070161
```

As we can see that neural network gives the best results in terms of accuracy which is 78.21% and even if we compare the misclassification rate, neural network gives the best results as compared to all other models and fourth predicted model which gives a misclassification rate of 21.8%.

So we select Neural network as the best predictor for our model as proceed further. The Receiver Operating Characteristic curve also known as ROC curve, plots the performance of classifier in terms of its true positive rate (in y-axis) and false positive rates (in x-axis). We found that our **ROC score** to be **78%** which indicated Neural Network as a good classifier.

```
probs = mlp.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



CONCLUSION :

We have used three machine learning algorithms to built a bankruptcy classification model. These algorithms are Logistic Regression, Neural Networks, and Support Vector Machine (SVM). Logistic Regression and Neural Networks model performed better than Support Vector Machine. Since the accuracy and misclassification rate of Neural Network model is slightly better than Logistic Regression, we have decided to use Neural Network model as our final model for bankruptcy classification.

Recommendations:
Since we have built a Neural Network model for classification problem we can use this model in various applications such as:

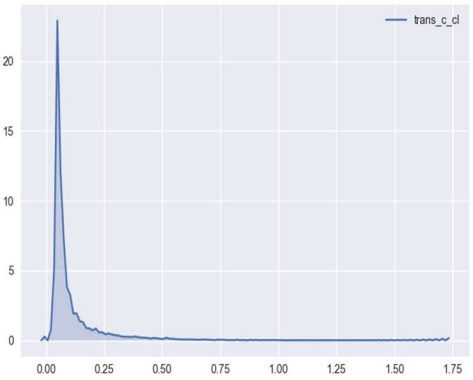
1. **Fraud Detection** - detecting whether the transaction made is fraudulent
2. **Target Marketing** - finding the customers which will more likely to respond to the marketing campaign.
3. **Medical Diagnosis** - detecting whether the patient is likely to have medical issues in future.
4. **Churn Forecasting** - predicting whether the customer is going to leave in future

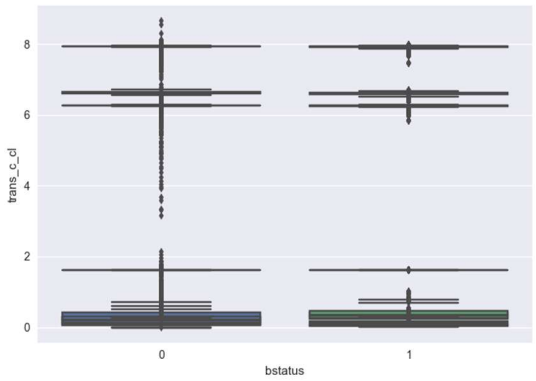
We can use neural network models in variety of other applications such as:

1. **Character Recognition** – Neural Networks can be used to identify handwritten characters.
2. **Image Compression** – As websites are using more images Neural Network can be used to for image compression because of its capability of handling vats amount of information at once.
3. **Stock Market Prediction** – Since Neural Networks are really fast with large amount of information we can use it to predict stock prices.

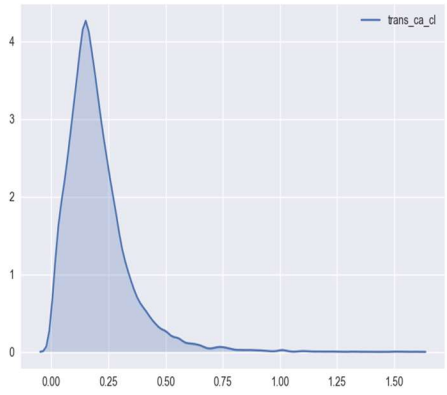
APPENDIX:

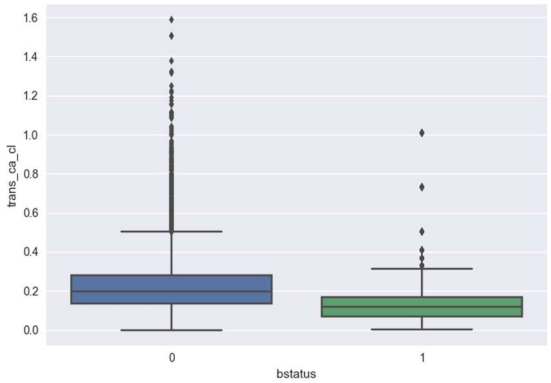
a) Trans_c_cl



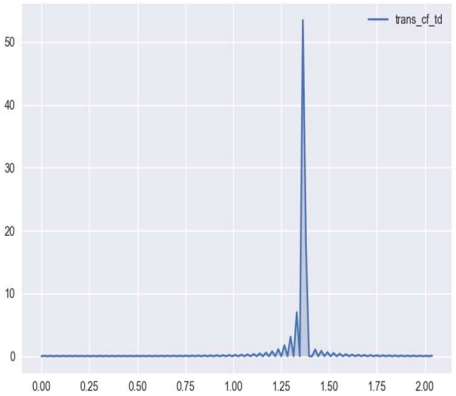


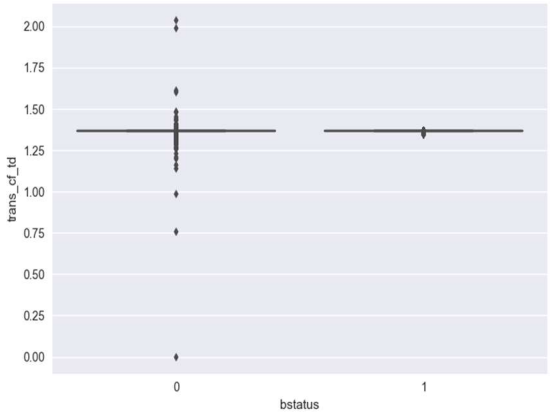
b) trans_ca_cl



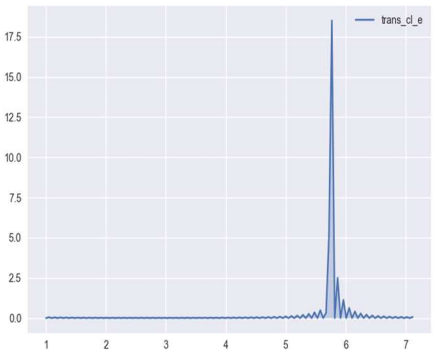


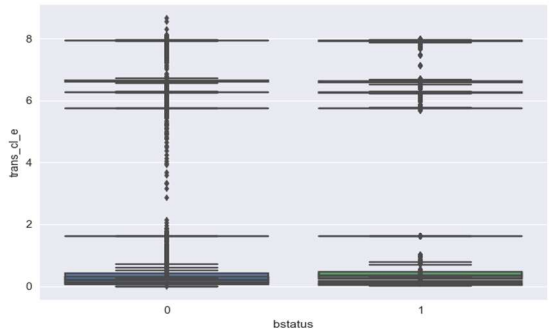
c) trans_cf_td



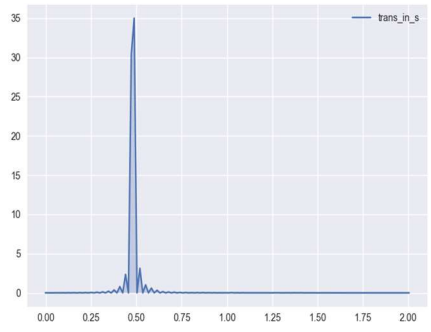


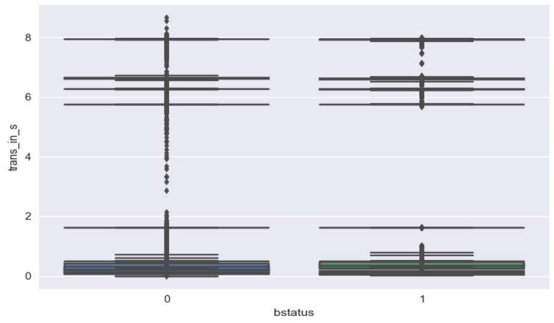
d) trans_cl_e



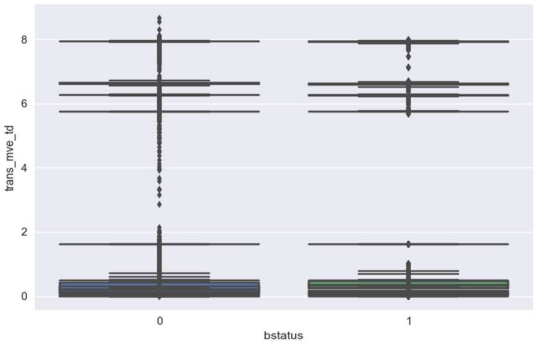


e) trans_in_s

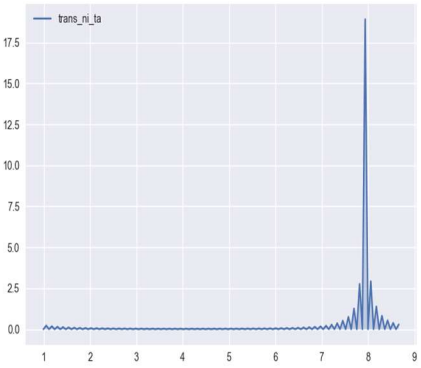


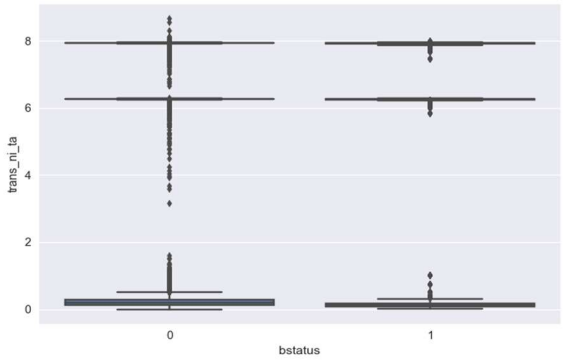


f) trans_mve_td

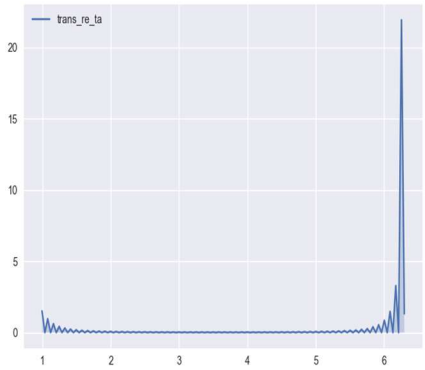


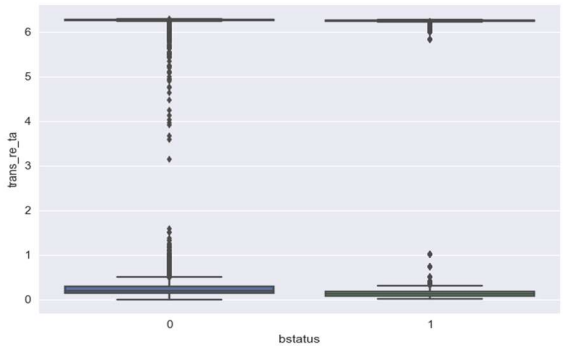
g) trans_ni_ta



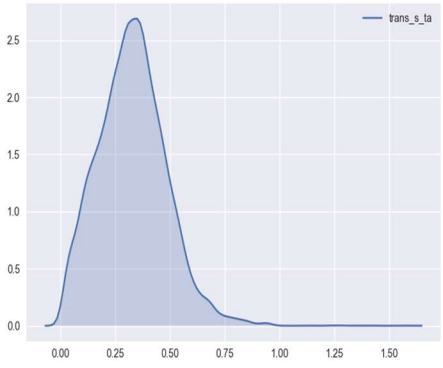


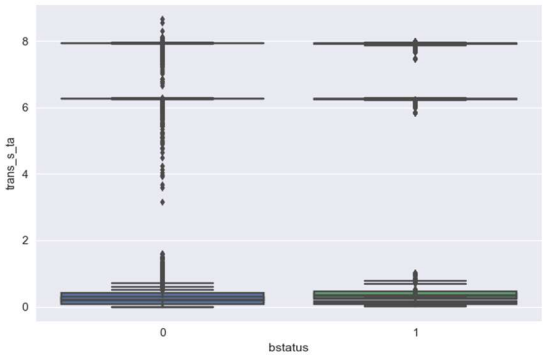
h) trans_re_ta



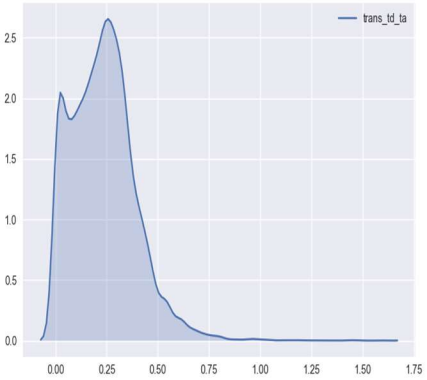


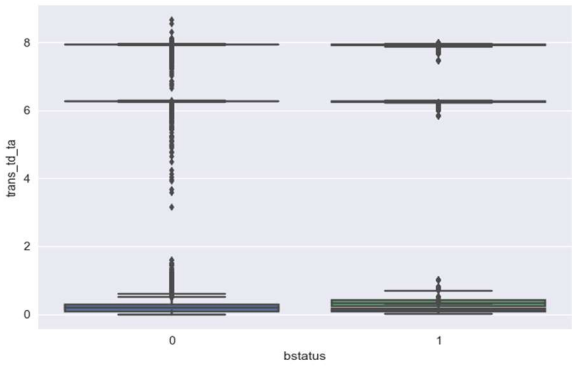
i) trans_s_ta



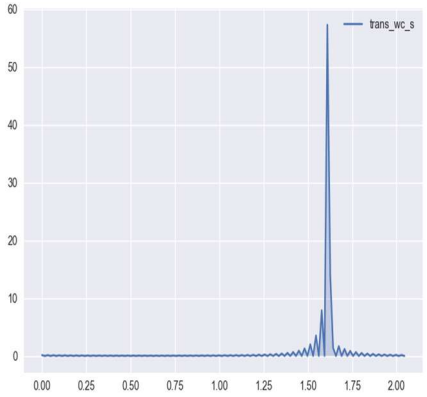


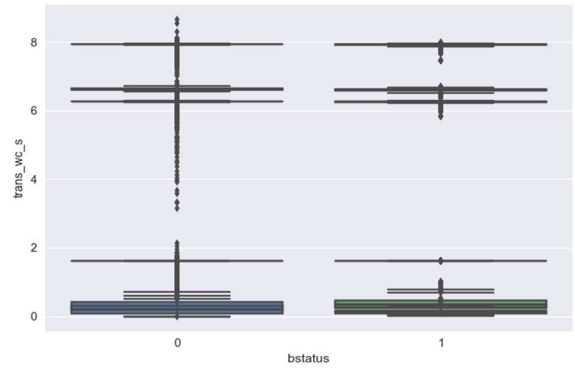
j) trans_td_ta





k) trans_wc_s





l) trans_wc_ta