device_failure_prediction

September 20, 2019

Creator: Changhee Kang

Note that data manupulation and exploration techniques illustrated in this demonstraton does not mean that the audience have to follow the same ways as shown here.

1 Device Failure and Maintenance Prediction Model

It is to build a predictive model or models with diagnoses of telemetry attributes to classify whether maintenance should be performed on devices or not. The column to be use to predict is called "failure" with binary value 0 for "non-failure" and 1 for "failure". ***The goal is to minimize false positives and false negatives.***

2 Assumption

As there is no meta data for the description of the current dataset, assumptions can be applied to the current dataset. The dataset consists of diagnoses of telemetry attributes, so it might be rational that some variables are assumed to consist of *categorical nominal type values* while other variables would consist of *continuous type values*.

3 Roadmap

This demonstration is to show how to handle datasets which are imbalanced. The provided dataset has no description for variables. The dataset will be analyzed by using various statistical approaches. The main issue with the current dataset is that it is highly imblanaced. Various sampling techniques usually applied to the imbalanced dataset. There will be detailed explanations with each sampling techniques as the analyses progresses further. In this demonstration, data exploration will only be extensively explained. In the following successive sections, first, no sampling technique will be applied in this demonstration to understand how sampling techniques will affect performances of the predictive model on an imblanaces dataset. Although data preprocessing is done for most datasets but there will be no data preprocessing for this demonstration.

Additive variable generations such as day, day of the week, month, device operation days, or season from the *'date'* variable could be an option to enhance the model performance. However, additivie variable generations does not always have to be applied to introduce new variables to the original dataset, unless there is any acceptable model performance with the orginal dataset. One could introduce additive variables before any model development but there is no guarantee that those additive variables are going to improve the model performances.

However, it should be noted that normalizations for continuous values shoud not be confused with additive variable generations. One might be also tempted to apply binning to continuous values but it sould be applied when there is some group of values that are significantly different from other groups of values in a variable. Otherwise, binning would not be much help and it will only cause lose of information of continuity in the values. Distribution transformations, for example, log-normal transformation, could be used but it is not neccessary to be applied for Randome Forest learning algorithm. It all depends on what the analyzer intend to do with the dataset.

Logsitic Regression and Neural Network based algorithms could also have been used but assuming that values of some variables are categorical nominal type values, to properly use those algorithms, creating dummy or one-hot-encoded variables for those algorithms simply overwhelms contraints on the given task. Therefore, for the predictive model development, instead of using Losigitic Regression or Neural Network based algorithms, Random Forest predictive model learning algorhtm will be adopted because Random Forest can handle very well for both categorical nominal type values and continuous type values at the same time. Random Forest does not require continuous type values to be normalized at all.

4 Data Loading

import necessary python modules.

[25]:
```
import pandas as pd
import numpy as np
```

Load the dataset into memory.

[26]:
```
datafile = r'/home/thomas/Downloads/device_failure.csv'
dataset = pd.read_csv(datafile, sep=',', engine='python')
```

5 Data Exploration

See if the dataset has been loaded correctly. There should be 12 columns meaning 12 variables.

[3]:
```
dataset.head()
```

[3]:
```
         date    device  failure  attribute1  attribute2  attribute3  \
0  2015-01-01  S1F01085        0   215630672          56           0
1  2015-01-01  S1F0166B        0    61370680           0           3
2  2015-01-01  S1F01E6Y        0   173295968           0           0
3  2015-01-01  S1F01JE0        0    79694024           0           0
4  2015-01-01  S1F01R2B        0   135970480           0           0

   attribute4  attribute5  attribute6  attribute7  attribute8  attribute9
0          52           6      407438           0           0           7
1           0           6      403174           0           0           0
2           0          12      237394           0           0           0
3           0           6      410186           0           0           0
4           0          15      313173           0           0           3
```

Take a look at the number of total records.

[4]:
```
dataset.failure.count()
```

[4]:
```
124494
```

Take a look at the data structure.

[5]:
```
dataset.dtypes
```

[5]:
```
date           object
device         object
failure         int64
attribute1      int64
attribute2      int64
attribute3      int64
attribute4      int64
attribute5      int64
attribute6      int64
attribute7      int64
attribute8      int64
attribute9      int64
dtype: object
```

There are 12 variables in the dataset. All of the variables in the dataset are all integer values except date and device variables. Let's take a look at the distribution of the dataset in terms of failure and non-failure.

[6]:
```
value_distribution = dataset.failure.value_counts()
```

[7]:
```
print("Non-failure: {} [{:.2f}%]".format(value_distribution[0],␣
 ↪(value_distribution[0]/dataset.failure.count())*100))
print("Failure: {} [{:.2f}%]".format(value_distribution[1],␣
 ↪(value_distribution[1]/dataset.failure.count())*100))
```

```
Non-failure: 124388 [99.91%]
Failure: 106 [0.09%]
```

[8]:
```
value_distribution.plot(kind='bar')
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50c88322b0>
```

The number of device failures is less than 0.1% and the dataset appears to be highly imbalanced. As shown above, there are 12 varaibles. Failure is the target variable and 11 variables are considered to be independent variables. All the independent variables seems to be integers. The next things to do is to see if there are any null values or missing values in columns of the dataset. The following two lines will figure out if there any null and missing values in the dataset.

[30]:
```
dataset.isnull().any()
```

[30]:
```
date          False
device        False
failure       False
attribute1    False
attribute2    False
attribute3    False
```

```
attribute4    False
attribute5    False
attribute6    False
attribute7    False
attribute8    False
attribute9    False
dtype: bool
```

[31]:
```python
dataset.isna().any()
```

[31]:
```
date          False
device        False
failure       False
attribute1    False
attribute2    False
attribute3    False
attribute4    False
attribute5    False
attribute6    False
attribute7    False
attribute8    False
attribute9    False
dtype: bool
```

In order to confirm that there are no missing and null values in every variable, the following can be executed for a list of all the summations of each column:

[32]:
```python
dataset.isnull().sum()
```

[32]:
```
date          0
device        0
failure       0
attribute1    0
attribute2    0
attribute3    0
attribute4    0
attribute5    0
attribute6    0
attribute7    0
attribute8    0
attribute9    0
dtype: int64
```

It is clear to say that there are no null values in the dataset. Currently the meaning of each variable is unknown and it is definitely required to explore each variable to find out what each variable would mean. There are 106 samples, approaximately 0.0852%, indicating failure. The above distribution shows that the dataset is perfectly imbalanced. In order to grasp the overall representation of the dataset, it is favorable to look into the characteristics of each variable, first. The following lines will first reveal the number of unique values in each variable and will give an overall idea of how each variable contributes to failure and non-failure.

[169]:
```python
# Show distinct values of each variable.
columns = dataset.columns
no_unique_values = []
for column in columns:
    unique_values = np.unique(dataset[column])
    print('{}: {} distinct values -  {}'.format(column, len(unique_values),␣
 ↪unique_values))
    no_unique_values.append(len(unique_values))
```

```
date: 304 distinct values -  ['2015-01-01' '2015-01-02' '2015-01-03'
'2015-01-04' '2015-01-05'
 '2015-01-06' '2015-01-07' '2015-01-08' '2015-01-09' '2015-01-10'
 '2015-01-11' '2015-01-12' '2015-01-13' '2015-01-14' '2015-01-15'
 '2015-01-16' '2015-01-17' '2015-01-18' '2015-01-19' '2015-01-20'
 '2015-01-21' '2015-01-22' '2015-01-23' '2015-01-24' '2015-01-25'
 '2015-01-26' '2015-01-27' '2015-01-28' '2015-01-29' '2015-01-30'
 '2015-01-31' '2015-02-01' '2015-02-02' '2015-02-03' '2015-02-04'
 '2015-02-05' '2015-02-06' '2015-02-07' '2015-02-08' '2015-02-09'
 '2015-02-10' '2015-02-11' '2015-02-12' '2015-02-13' '2015-02-14'
 '2015-02-15' '2015-02-16' '2015-02-17' '2015-02-18' '2015-02-19'
 '2015-02-20' '2015-02-21' '2015-02-22' '2015-02-23' '2015-02-24'
 '2015-02-25' '2015-02-26' '2015-02-27' '2015-02-28' '2015-03-01'
 '2015-03-02' '2015-03-03' '2015-03-04' '2015-03-05' '2015-03-06'
 '2015-03-07' '2015-03-08' '2015-03-09' '2015-03-10' '2015-03-11'
 '2015-03-12' '2015-03-13' '2015-03-14' '2015-03-15' '2015-03-16'
 '2015-03-17' '2015-03-18' '2015-03-19' '2015-03-20' '2015-03-21'
 '2015-03-22' '2015-03-23' '2015-03-24' '2015-03-25' '2015-03-26'
 '2015-03-27' '2015-03-28' '2015-03-29' '2015-03-30' '2015-03-31'
 '2015-04-01' '2015-04-02' '2015-04-03' '2015-04-04' '2015-04-05'
 '2015-04-06' '2015-04-07' '2015-04-08' '2015-04-09' '2015-04-10'
 '2015-04-11' '2015-04-12' '2015-04-13' '2015-04-14' '2015-04-15'
 '2015-04-16' '2015-04-17' '2015-04-18' '2015-04-19' '2015-04-20'
 '2015-04-21' '2015-04-22' '2015-04-23' '2015-04-24' '2015-04-25'
 '2015-04-26' '2015-04-27' '2015-04-28' '2015-04-29' '2015-04-30'
 '2015-05-01' '2015-05-02' '2015-05-03' '2015-05-04' '2015-05-05'
 '2015-05-06' '2015-05-07' '2015-05-08' '2015-05-09' '2015-05-10'
 '2015-05-11' '2015-05-12' '2015-05-13' '2015-05-14' '2015-05-15'
 '2015-05-16' '2015-05-17' '2015-05-18' '2015-05-19' '2015-05-20'
 '2015-05-21' '2015-05-22' '2015-05-23' '2015-05-24' '2015-05-25'
 '2015-05-26' '2015-05-27' '2015-05-28' '2015-05-29' '2015-05-30'
 '2015-05-31' '2015-06-01' '2015-06-02' '2015-06-03' '2015-06-04'
 '2015-06-05' '2015-06-06' '2015-06-07' '2015-06-08' '2015-06-09'
 '2015-06-10' '2015-06-11' '2015-06-12' '2015-06-13' '2015-06-14'
 '2015-06-15' '2015-06-16' '2015-06-17' '2015-06-18' '2015-06-19'
 '2015-06-20' '2015-06-21' '2015-06-22' '2015-06-23' '2015-06-24'
 '2015-06-25' '2015-06-26' '2015-06-27' '2015-06-28' '2015-06-29'
 '2015-06-30' '2015-07-01' '2015-07-02' '2015-07-03' '2015-07-04'
```

```
 '2015-07-05' '2015-07-06' '2015-07-07' '2015-07-08' '2015-07-09'
 '2015-07-10' '2015-07-11' '2015-07-12' '2015-07-13' '2015-07-14'
 '2015-07-15' '2015-07-16' '2015-07-17' '2015-07-18' '2015-07-19'
 '2015-07-20' '2015-07-21' '2015-07-22' '2015-07-23' '2015-07-24'
 '2015-07-25' '2015-07-26' '2015-07-27' '2015-07-28' '2015-07-29'
 '2015-07-30' '2015-07-31' '2015-08-01' '2015-08-02' '2015-08-03'
 '2015-08-04' '2015-08-05' '2015-08-06' '2015-08-07' '2015-08-08'
 '2015-08-09' '2015-08-10' '2015-08-11' '2015-08-12' '2015-08-13'
 '2015-08-14' '2015-08-15' '2015-08-16' '2015-08-17' '2015-08-18'
 '2015-08-19' '2015-08-20' '2015-08-21' '2015-08-22' '2015-08-23'
 '2015-08-24' '2015-08-25' '2015-08-26' '2015-08-27' '2015-08-28'
 '2015-08-29' '2015-08-30' '2015-08-31' '2015-09-01' '2015-09-02'
 '2015-09-03' '2015-09-04' '2015-09-05' '2015-09-06' '2015-09-07'
 '2015-09-08' '2015-09-09' '2015-09-10' '2015-09-11' '2015-09-12'
 '2015-09-13' '2015-09-14' '2015-09-15' '2015-09-16' '2015-09-17'
 '2015-09-18' '2015-09-19' '2015-09-20' '2015-09-21' '2015-09-22'
 '2015-09-23' '2015-09-24' '2015-09-25' '2015-09-26' '2015-09-27'
 '2015-09-28' '2015-09-29' '2015-09-30' '2015-10-01' '2015-10-02'
 '2015-10-03' '2015-10-04' '2015-10-05' '2015-10-06' '2015-10-07'
 '2015-10-08' '2015-10-09' '2015-10-10' '2015-10-11' '2015-10-12'
 '2015-10-13' '2015-10-14' '2015-10-15' '2015-10-16' '2015-10-17'
 '2015-10-18' '2015-10-19' '2015-10-20' '2015-10-21' '2015-10-22'
 '2015-10-23' '2015-10-24' '2015-10-25' '2015-10-26' '2015-10-27'
 '2015-10-29' '2015-10-30' '2015-10-31' '2015-11-02']
device: 1169 distinct values -  ['S1F01085' 'S1F013BB' 'S1F0166B' ... 'Z1F26YZB'
'Z1F282ZV' 'Z1F2PBHX']
failure: 2 distinct values -  [0 1]
attribute1: 123877 distinct values -  [        0      2048      2056 ...
244136552 244138600 244140480]
attribute2: 558 distinct values -  [    0     8    16    24    32    40    48
56    64    72    80    88
    96   104   112   120   128   136   144   152   160   168   176   184
   192   200   208   216   224   232   240   248   256   264   272   280
   288   296   304   320   328   336   344   352   360   368   376   384
   392   400   408   416   424   432   440   448   456   464   472   480
   488   496   504   512   520   528   536   544   552   560   568   576
   584   592   600   608   616   624   632   640   648   656   664   672
   680   704   712   728   736   744   752   760   776   792   800   808
   816   824   832   840   848   864   872   888   896   912   920   928
   936   944   952   960   968   976   984   992  1000  1024  1032  1040
  1048  1056  1064  1072  1080  1088  1096  1104  1112  1120  1128  1136
  1144  1152  1160  1176  1184  1192  1200  1208  1232  1240  1248  1256
  1264  1280  1288  1296  1312  1320  1336  1360  1376  1392  1400  1416
  1424  1440  1464  1480  1488  1504  1536  1552  1560  1568  1576  1584
  1592  1600  1616  1624  1632  1648  1656  1664  1672  1720  1744  1752
  1768  1776  1800  1808  1816  1832  1840  1848  1864  1880  1888  1912
  1920  1928  1936  1944  1952  1960  1968  1976  1984  1992  2000  2008
  2016  2024  2032  2040  2048  2056  2064  2072  2096  2104  2144  2192
```

```
  2208  2232  2248  2256  2272  2280  2288  2296  2304  2320  2328  2336
  2344  2352  2360  2368  2376  2384  2392  2400  2416  2424  2432  2440
  2448  2456  2464  2472  2480  2488  2496  2520  2528  2536  2544  2552
  2568  2576  2600  2608  2616  2640  2648  2656  2664  2704  2712  2720
  2744  2768  2784  2792  2808  2816  2824  2832  2848  2856  2880  2904
  2912  2920  2936  2944  2984  2992  3000  3016  3032  3040  3048  3064
  3072  3080  3096  3104  3112  3128  3136  3144  3152  3160  3200  3216
  3248  3272  3288  3352  3376  3432  3440  3456  3472  3488  3512  3528
  3536  3552  3576  3584  3592  3600  3608  3616  3624  3696  3800  3840
  3848  4072  4080  4160  4240  4248  4264  4280  4304  4312  4360  4440
  4448  4456  4464  4472  4488  4496  4512  4536  4768  4792  4808  4816
  4832  4920  4960  5160  5184  5192  5560  5624  6048  6080  6096  6128
  6176  6216  6248  6264  6272  6280  6288  6304  6328  6336  6344  6352
  6360  6368  6376  6416  6464  6472  6480  6504  6544  6656  6680  6720
  6744  6792  6800  6808  6856  6912  6992  7024  7096  7128  7160  7216
  7240  7344  7384  7400  7440  7448  7456  7504  7520  7544  7552  7624
  7640  7648  7672  7680  7712  7752  7768  7800  7808  7888  7904  7928
  7944  7960  7968  7976  8064  8104  8120  8136  8160  8392  8480  8488
  8520  8528  8536  8544  8688  8760  9008  9264  9536  9592  9848  9920
  9952  9960  9984 10024 10064 10096 10168 10184 10200 10232 10248 10288
 10304 10328 10336 10344 10352 10360 10368 10376 10384 10392 10440 10448
 10592 10648 11216 11432 11856 11872 11880 11888 12360 12928 13088 13560
 14856 14904 14920 15304 15328 15336 16408 16864 17408 17704 18072 19200
 21200 21528 21544 21816 21928 21944 22816 23208 23400 23936 24656 24680
 24792 24904 24920 24976 24984 27856 28344 32936 33000 33016 33024 33528
 34912 35616 37936 41232 43712 44008 44024 44152 44176 44224 44232 46264
 46280 46296 46304 46328 46400 46408 46496 46520 46584 46848 46872 47944
 49768 49840 51976 54752 54896 56584 56736 61592 62296 64464 64472 64584
 64728 64736 64776 64784 64792 64968]
attribute3: 47 distinct values -  [    0     1     2     3     4     5     7
8     9    10    11    12
    14    15    16    18    21    24    25    34    35    36    38    53
    56    61    62    70    72   100   107   208   220   263   266   279
   318   323   377   382   406  1162  1326  1331  2112  2693 24929]
attribute4: 115 distinct values -  [   0    1    2    3    4    5    6    7    8
9   10   11   12   13
   14   15   16   17   18   19   20   21   22   23   24   25   26   27
   28   29   30   31   32   34   35   36   37   38   39   40   41   43
   44   45   46   48   49   50   51   52   53   55   56   57   58   60
   62   65   67   69   73   74   76   79   80   86   90   91   94   95
   97  100  108  112  118  121  122  128  129  135  147  160  164  173
  175  186  204  214  215  235  236  256  288  297  299  300  305  322
  331  399  400  401  405  406  431  486  487  521  529  533  763  841
 1033 1074 1666]
attribute5: 60 distinct values -  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24
 25 29 30 31 32 33 34 35 36 37 38 39 40 41 42 57 58 59 60 61 62 63 64 65
 66 67 68 70 78 89 90 91 92 94 95 98]
```

```
attribute6: 44838 distinct values -  [     8      9     12 ... 689035 689062
689161]
attribute7: 28 distinct values -  [  0   6   8  16  22  24  32  40  48  56  72
80  96 104 112 128 136 152
 176 216 240 272 312 424 496 736 744 832]
attribute8: 28 distinct values -  [  0   6   8  16  22  24  32  40  48  56  72
80  96 104 112 128 136 152
 176 216 240 272 312 424 496 736 744 832]
attribute9: 65 distinct values -  [    0     1     2     3     4     5     6
7     8     9    10    11
    12    13    14    15    18    19    20    21    22    23    24    25
    26    30    33    34    38    41    42    51    52    57    65    70
    98   104   120   145   155   164   177   205   222   233   241   248
   255   263   269   400   898  1080  1150  1165  1864  2269  2270  2522
  2637  2794  7226 10137 18701]
```

Summary of unique values

- date : 304 unique values.
- devce: 1,169 unique values.
- attribute1: 123,877 unique values.
- attribute2: 558 unique values.
- attribute3: 47 unique values.
- attribute4: 115 unique values.
- attribute5: 60 unique values.
- attribute6: 44,838 unique values.
- attribute7: 28 unique values.
- attribute9: 65 unique values.

For a dataset with a large number of variables, it would be extremly hard to manually look at all the values of each variable in the dataset. Nonetheless, data exploration for understanding characteristics of individual variables is mandatory to undergo when a story telling about the current dataset is required. However, the current dataset only has 9 variables excluding *'date'*, *'device'*, and *'failure'* variables, so it is reasonble to look at individual characteristics of unique values of each variable.

- *'attribute1'* and *'attribute6'* have relatively high numbers of unique values while numbers of unique values for other values are small. Those two variables must have a lot of ways for the predictive model to learn non-failures of devices. The predictive model must not only know how to tell devices with failure but it also has to learn ways to failure, too. A brief unique value summary will help grasp a further picture of the dataset.

- The ranges of unique values of *'attribute1'* and *'attribute6'* are much broader than the ranges of unique values of other variables.

- *'attribute7'* and *'attribute8'* seem to be idential to one another. Either of the variables, *'attribute7'* or *'attribute8'*, should be removed from analysis if those variables are used by Logistic Regression or Ordinal Regression since those two analyses strongly assume little or no multicollinearity.

- *'attribute1'* appears to be multiples of *'attribute2'*, either of those two variables should also be remove from the dataset unless only one of them shoudl be used. Otherwise, they will cause a multicollinearity problem if Logistic Regression or Ordinal Regression analyses are implemented for the predictive model. *'attribute1'* seems to express the amount of daily data transmission handled by each device.
- Although Logistics Regression and Ordinary Regression are not applied here, but those multi-collinearity issues should be noted. When Logsitics Regression or Ordinary Regression is undertaken, multi-collinearity tests must be performed for a reliable multi-collinearity asiciation to ensure the predictive model performance.

Let's first check if *'attribute7'* and *'attribute8'* are identical to one another.

[23]:
```
# Create two different dataframes for attribute7 and attribute8.
attribute_7 = pd.DataFrame(dataset, columns = ['attribute7'])
attribute_8 = pd.DataFrame(dataset, columns = ['attribute8'])
attribute_test = pd.DataFrame(dataset, columns = ['matched'])

# Compare attribute7 and attribute8 by their values for each record.
attribute_test['matched'] = pd.DataFrame(np.where(attribute_7.attribute7 ==␣
↪attribute_8.attribute8, 'True', 'False'))
```

[24]:
```
# See the number of matched values.
attribute_test['matched'].value_counts()
```

[24]:
```
True     124494
Name: matched, dtype: int64
```

It is safe to say *'attribute7'* is a duplicate of *'attribute8'* or the other way around. As mentioned earlier above, if a variable is a copy of another variable, it could be problematic when the predictive model is developed using Logistic Regression Classifier since Logistic Regression assumes little or no multi-collinearity among independent variables, so either of them should be removed from the dataset or they both can stay in the dataset as long as only one of them is applied to the model development.

If the model is developed by using a ensemble classfier such as Random Forest, the effect of multi-collinearity would not have much impact on the preditive model since Random Forest will create the predictive model with random selection of features at each node creation, but in general the effect is not removed completely. It could be experimented during the model development.

It would be useful if the overall number of unique values in each variable could be presented with graphical representations. A pie chart and a bar chart would do the trick to quickly understand the overall characteristics of distributions of unique values in the dataset.

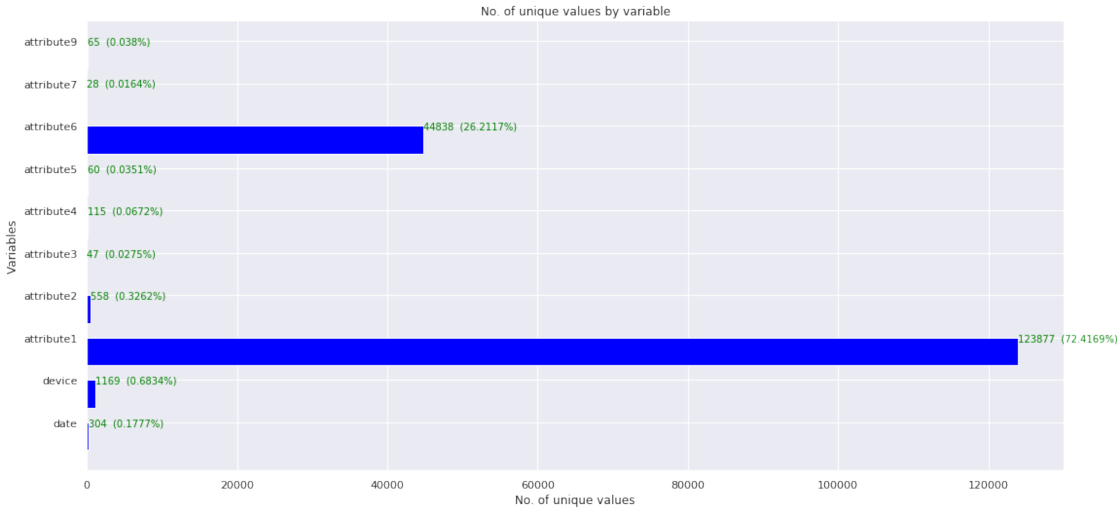
[11]:
```
# import matplotlib to plot graphical statistics.
import matplotlib.pyplot as plt
```

[27]:
```
'''
    It returns an array of numbers of unique values in each variable.
'''
def get_no_unique_values(columns):
    no_unique_values = []
    for column in columns:
        no_unique_value = dataset[column].nunique()
```

```
        no_unique_values.append(no_unique_value)
    return no_unique_values

'''
    Plot a bar chart for the number of unique values in each variable.
    Numbers of unique values of each variable will also be displayed on top of␣
 ↪each bar,
    otherwise those small values would have no way to display themselves.
'''
def plot_no_unique_values_bar_chart(columns):
    no_unique_values = get_no_unique_values(columns)
    total_no_unique_value = sum(no_unique_values)
    percentages = []

    for value in no_unique_values:
        percentages.append(round((value/total_no_unique_value)*100, 4))

    fig, ax = plt.subplots()
    percent_idx = 0
    for i, v in enumerate(no_unique_values):
        ax.text(v + 3, i + .25, str(v)+str(" ␣
 ↪(")+str(percentages[percent_idx])+str("%)"), color='green')
        percent_idx += 1

    width = 0.65 # the width of the bars
    ind = np.arange(len(no_unique_values))
    ax.barh(ind, no_unique_values, width, color="blue")
    ax.set_yticks(ind+width/2)
    ax.set_yticklabels(columns, minor=False)
    plt.title('No. of unique values by variable')
    plt.xlabel('No. of unique values')
    plt.ylabel('Variables')
    plt.show()
```

[31]:
```
'''
    attribute8 is removed from the column list since it is a duplicate of␣
 ↪attribute7.
'''
columns = ['date',       'device',     'attribute1',
           'attribute2', 'attribute3', 'attribute4',
           'attribute5', 'attribute6', 'attribute7',
           'attribute9']

'''
    Generate a bar chart for more details with counts of unique values in each␣
 ↪variable.
'''
```

```
plt.rcParams['figure.figsize'] = (18.0, 8.5)
plot_no_unique_values_bar_chart(columns)
```

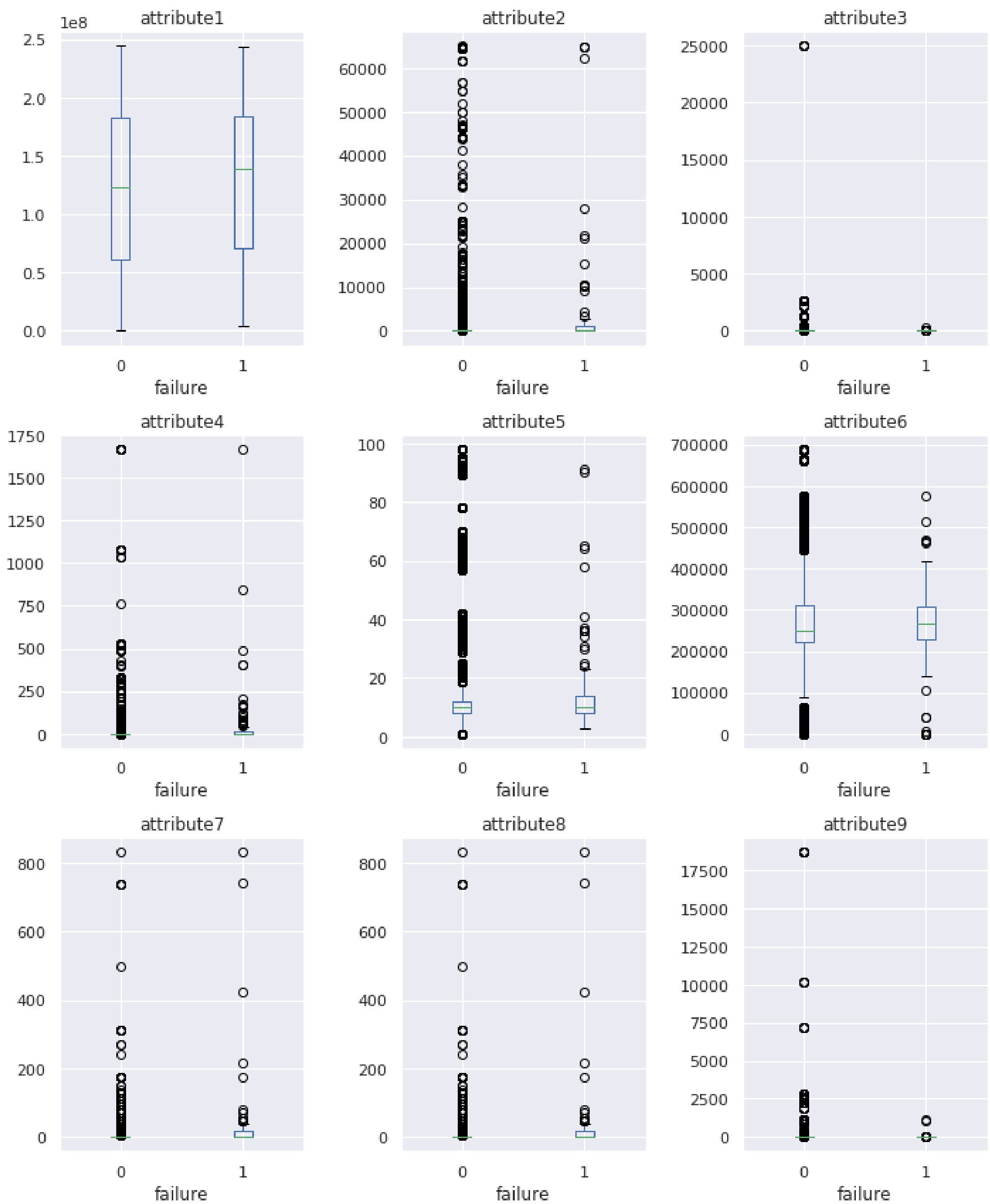


- *'attribute1'* and *'attribute6'* have large numbers of unique values while other variables have relatively small numbers of unique values.
- Unique values of *'attribute1'* are assumed to be the amount of daily data exchange. *'attribute6'* also a large number of unique values but it is not really conclude what *'attribute6'* means.
- It seems reasonable to consider unique values of *'attribute1'* and *'attribute6'* as continuous values.
- *'attribute2'* has 558 unique values and *'attribute4'* has 115 unique values.
- Treating unique values of *'attribute2'* as a continuous values seems to be rational.
- *'attribute3'*, *'attribute4'*, *'attribute5'*, 'attribute7', *and 'attribute9'* are assumed to be categorical nominal type vaues. discrete values of those variables could be treated as continuous type values but in this demonstration, they are forced to be categorical nominal type values to apply a variety of modeling approaches.

'attribute1', *'attribute2'*, and *'attribute6'* are considered to be continuous type values while *'attribute3'*, *'attribute4'*, *'attribute5'*, 'attribute7', *and 'attribute9' are assumed to be categorical nominal value types. Recall that 'attribute8'* is not included because *'attribute8'* is identical to *'attribute7'*. One can use both *'attribute7'* and *'attribute8'* if Random Forest is the learning algorithm but for this demonstration, *'attribute8'* will not be used. Before precedeing further, let's take a look at ranges of values in each variable for overall characteristics using boxplots. Although boxplots are for continuous values, they can be utilized to picture overall characteristics of variables even for categorical values, too.

[230]:
```
fig = plt.figure(figsize=(10,12))

ax1 = plt.subplot(331)
ax2 = plt.subplot(332)
```

```
ax3 = plt.subplot(333)
ax4 = plt.subplot(334)
ax5 = plt.subplot(335)
ax6 = plt.subplot(336)
ax7 = plt.subplot(337)
ax8 = plt.subplot(338)
ax9 = plt.subplot(339)

dataset.boxplot(column='attribute1',by='failure',ax=ax1)
dataset.boxplot(column='attribute2',by='failure',ax=ax2)
dataset.boxplot(column='attribute3',by='failure',ax=ax3)
dataset.boxplot(column='attribute4',by='failure',ax=ax4)
dataset.boxplot(column='attribute5',by='failure',ax=ax5)
dataset.boxplot(column='attribute6',by='failure',ax=ax6)
dataset.boxplot(column='attribute7',by='failure',ax=ax7)
dataset.boxplot(column='attribute8',by='failure',ax=ax8)
dataset.boxplot(column='attribute9',by='failure',ax=ax9)

plt.suptitle('')
plt.tight_layout()
```



5.1 Categorical Variable Exploration

Unique values, contributing to failure, of each variable will be studied. The following uitility functions will help plot the distribution of contributions of unique values in each variable.

[247]:

```python
total_record = dataset['failure'].count() # Get a count for total records.
failure_summary_table = [["Variable", "Total", "Failure", "Non-failure",␣
↪"Intersected"]] # a summary table.

'''
    This will plot statistics on unique values with failure contribution.
'''
def plot_unique_value_contribution_device_failure(attr):

    '''
        Prepare dataframe from the original dataset for required statistics.
    '''
    plt.rcParams['figure.figsize'] = (18.0, 3.5)
    attr_failure_info = dataset['failure'] == 1
    df1 = pd.DataFrame(dataset, columns=[attr])
    df2 = pd.DataFrame(dataset[attr_failure_info], columns=[attr])
    attr_failure_info = dataset[attr_failure_info].groupby(attr).failure.sum()

    '''
        Compute necessary information to generate summary statistics on unique␣
↪values with
        contributions to both failure and non-failure.
    '''
    total_no_uniq_values = dataset[attr].nunique()
    unique_value_ratio = round((total_no_uniq_values/total_record)*100, 4)
    failures = attr_failure_info.count()
    failure_ratio = round((failures/total_no_uniq_values)*100, 4)
    non_failures = total_no_uniq_values - failures
    non_failure_ratio = round((non_failures/total_no_uniq_values)*100, 4)
    intersection_df = pd.merge(df1, df2, on=[attr], how='inner')
    intersection = intersection_df[attr].nunique()
    intersection_ratio = round((intersection/failures)*100, 4)

    '''
        Store summaries on unique values with failure contribution.
    '''
    tr = ['{}'.format(attr),
          '{:,}({:.2f}%)'.format(total_no_uniq_values, unique_value_ratio),
          '{:,}({:.2f}%)'.format(failures, failure_ratio),
          '{:,}({:.2f}%)'.format(non_failures, non_failure_ratio),
          '{:,}({:.2f}%)'.format(intersection, intersection_ratio)
         ]
    failure_summary_table.append(tr)
```

[248]:
```python
# See how unique values in attribute3 contributes to failure.
plot_unique_value_contribution_device_failure('attribute3')
plot_unique_value_contribution_device_failure('attribute4')
```

```
plot_unique_value_contribution_device_failure('attribute5')
plot_unique_value_contribution_device_failure('attribute7')
plot_unique_value_contribution_device_failure('attribute9')
```

Let's take a look at the summary of category variable analyses.

[249]:
```
from IPython.display import HTML, display
import tabulate
```

[250]:
```
display(HTML(tabulate.tabulate(failure_summary_table, tablefmt='html')))
```

```
<IPython.core.display.HTML object>
```

Those variables in the summary are all assumed to have categorical nominal type values. All the category variables have high cardinarities, meaning that those category variables have a large number of unique categorical values.

The predictive model needs to classify devices into either failure or non-failure. Looking at the summary above, *'attribute3'* will be the most importance variable for the predictive model to learn that a device is either in failure or non-failure since it has the highest number of intersected unique values. Having a higher number of intersected unique values for a categorical variable means that the predictive model knows many ways to tell if a device is in failure or non-failure. The rank of variable importances will follow high numbers of intersected unique values in each variable.

In addition to the summary above, it might a good idea to look at numerical categorical variable importance measures by using *'ExtraTreesClassifer'* in *'sklearn'* and compare the numerical feature importances to the summary above.

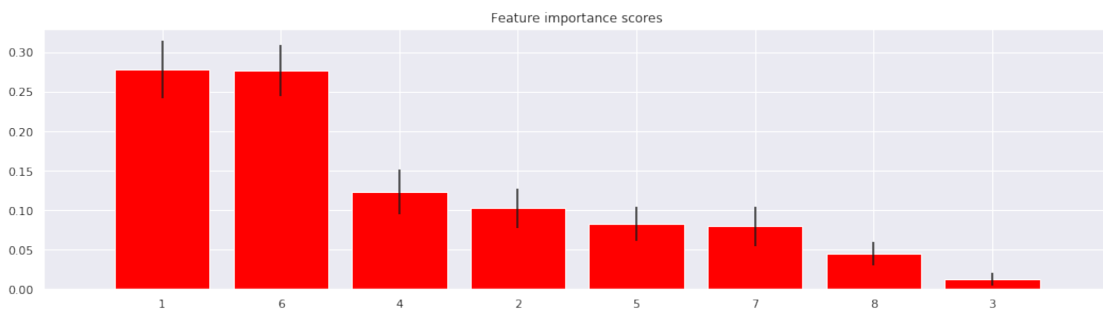
[217]:
```
# import ExtraTreesClassifier.
from sklearn.ensemble import ExtraTreesClassifier
```

[223]:
```
# Test features for importances.
def examine_features(x_train, y_train):
    # Build a forest and compute the feature importances
    forest = ExtraTreesClassifier(n_estimators=250, random_state=0)
    forest.fit(x_train, y_train)
    importances = forest.feature_importances_
    std = np.std([tree.feature_importances_ for tree in forest.estimators_],
                 axis=0)
    indices = np.argsort(importances)[::-1]

    # Print the feature ranking
    print("Feature ranking:")
    for f in range(X.shape[1]):
        print("%d. attribute %d (%f)" % (f + 1, indices[f] + 1,␣
 ↪importances[indices[f]]))

    # Plot the feature importances of the forest
    plt.rcParams['figure.figsize'] = (18.0, 4.5)
    plt.figure()
    plt.title("Feature importance scores")
```

```
    plt.bar(range(X.shape[1]), importances[indices], color="red",␣
 ↪yerr=std[indices], align="center")
    plt.xticks(range(X.shape[1]), indices+1)
    plt.xlim([-1, X.shape[1]])
    plt.show()
```

[224]:
```
# Execute variable importance test with all the variables in the current␣
 ↪dataset.
X = dataset[['attribute1', 'attribute2', 'attribute3', 'attribute4',␣
 ↪'attribute5', 'attribute6', 'attribute7', 'attribute9']]
Y = dataset.failure
examine_features(X, Y)
```

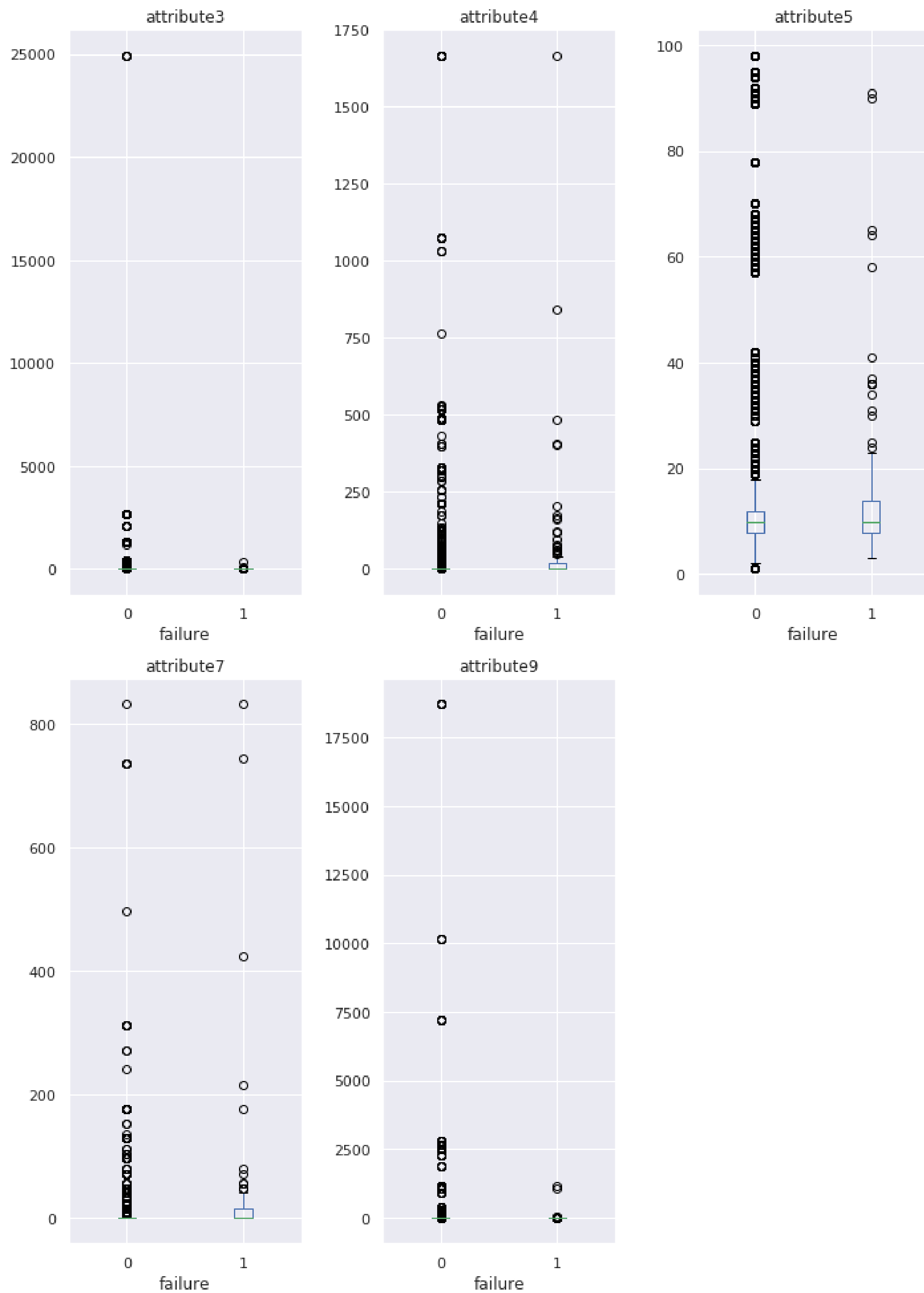
```
Feature ranking:
1. attribute 1 (0.278196)
2. attribute 6 (0.276792)
3. attribute 4 (0.123269)
4. attribute 2 (0.102534)
5. attribute 5 (0.082676)
6. attribute 7 (0.079533)
7. attribute 8 (0.044559)
8. attribute 3 (0.012441)
```



The numeric variable importance test with ExtraTreesClassifier shows the order of important variables according to the variable importance score of each variable. As mentioned above, it is conclusive that variables with a higher number of intersected unique values take higher variable importance scores. Knowing numerical importances of variables gives more confidence in choosing variables to include or exclude when developing the predictive model. With these understandings about variables in the current dataset, diving deeper into the spreads of unique values on failure and non-failure would give more insights to further understand about the variables. Although boxplots are for continuous values, boxplots can be improvised to look at the spreads of unique values of categorical variables, too. Note that one can use *'Logistic Regression'" to display the spread of values separately.

[253]:
```
fig = plt.figure(figsize=(10,20))

ax1 = plt.subplot(331)
```

```
ax2 = plt.subplot(332)
ax3 = plt.subplot(333)
ax4 = plt.subplot(334)
ax5 = plt.subplot(335)

dataset.boxplot(column='attribute3',by='failure',ax=ax1)
dataset.boxplot(column='attribute4',by='failure',ax=ax2)
dataset.boxplot(column='attribute5',by='failure',ax=ax3)
dataset.boxplot(column='attribute7',by='failure',ax=ax4)
dataset.boxplot(column='attribute9',by='failure',ax=ax5)

plt.suptitle('')
plt.tight_layout()
```



By looking at the boxplots above, *'attribute4'*, *'attribute5'*, and *'attribute7'* seem to have more options to contribute to failure than other variables: *'attribute3'* and *'attribute9'*. Above boxplots reveal that the unique value, *'0'*, or close to *'0'* in *'attribute3'*, and *'attribute9'* show that has most frequencies for both failure and non-failure while *'attribute4'*, *'attribute5'*, *'attribute7'* show similar spread patterns of their unique values except ranges of spreads of their unique values. From the study of variables, *'attribute4'*, *'attribute5'*, and *'attribute7'* might be good to use for the predictive model developments since they have more options to failure and have different spreads of unique values other than unique values for non-failure.

[ ]:

```

```