ML PROJECT

Authors

- Andrea Scarpellini
- Martina Scarpellini

Supervised Binary classification project.

DATASET EXPLORATION

In [219…

```
import pandas as pd
import numpy as np

%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
```

At first, we imported the dataset and printed it. It has 3000 rows and 16 columns.

In [220…

```
df = pd.read_csv('tyres_train.csv')
print(df.shape)
```

```
(3000, 16)
```

In [221…

```
df.head()
```

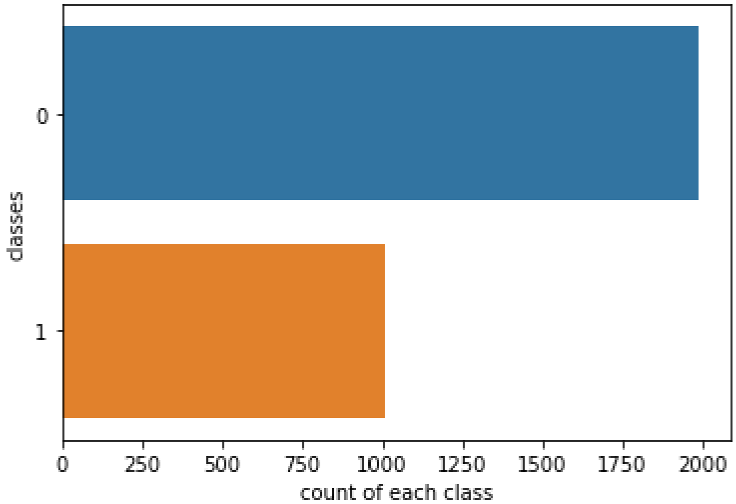
Out[221…

	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature	tread_type	tyre_season	e
0	17.990	26	1	0.16	0.01	-8.12	0	1	
1	20.704	36	1	0.30	0.01	-4.52	2	0	
2	19.156	34	1	0.30	0.01	-1.08	0	0	
3	16.802	35	1	0.19	0.02	7.44	1	0	
4	17.140	23	2	0.39	0.01	30.52	0	1	

CLASS COUNTS

In [222…

```
#Visualize Class Counts
sns.countplot(y=df.failure ,data=df)
plt.xlabel("count of each class")
plt.ylabel("classes")
plt.show()
```



Since the upper barchart shows us an imbalanced dataset, our analysis will be characterized as follow:

- Classification with original training data set
- Classification with oversampled training data set

MISSING VALUES

We wanted to look for missing values in our dataset. We noticed that the coloumn 'diameter' was the only one with at least one missing value (because it was the only one that gave 'true' as output).

In [223…

```
#in which column of our dataset is there at least a missing values?
df.isna().any()
```

Out[223…

```
vulc                False
perc_nat_rubber     False
wiring_strength     False
weather             False
perc_imp            False
temperature         False
tread_type          False
tyre_season         False
elevation           False
month               False
tread_depth         False
tyre_quality        False
perc_exp_comp       False
diameter             True
add_layers          False
failure             False
dtype: bool
```

In [224…

```
#percentage of NaN values in "diameter"
a=df['diameter'].isna().sum();
print("NaN value in diameter:",a)
print("Perc of NaN:", a/len(df)*100,"%")
print("\nDivided in:")
print(df[df['failure']==0]['diameter'].isna().sum(),"failure 0")
print(df[df['failure']==1]['diameter'].isna().sum(),"failure 1")
```

```
NaN value in diameter: 2110
Perc of NaN: 70.33333333333334 %

Divided in:
1407 failure 0
703 failure 1
```

We searched the percentage of Nan values in the diameter. We found that it was more than 70%. We decided to omit the diameter attribute from our analysis.

We listed all the attributes: Attributes:

- vulc Numerical
- perc_nat_rubber Numerical
- weather Numerical
- perc_imp Numerical
- temperature Numerical
- elevation Numerical
- perc_exp_comp Numerical
- diameter Numerical
- tread_type Categorical
- tyre_season Categorical
- month Categorical
- tread_depth Categorical
- wiring_strenght Categorical
- tyre_quality Categorical
- add_layers Categorical

Categorical attributes assume a finite number of distinct values, in most cases limited to less than a hundred, representing a qualitative property of an entity to which they refer. Numerical attributes assume a finite or infinite number of values and lend themselves to subtraction or division operations.

Numerical attributes are one that may take on any value within a finite or infinite interval.

We identified the datatype of our data. Some of the data are integers and some others are float.

In [225…

```
#come trattiamo le variabili categoriche
print(df.dtypes)
```

```
vulc               float64
perc_nat_rubber      int64
wiring_strength      int64
weather            float64
perc_imp           float64
temperature        float64
tread_type           int64
tyre_season          int64
elevation          float64
month                int64
tread_depth          int64
tyre_quality         int64
perc_exp_comp      float64
```

```
diameter           float64
add_layers           int64
failure              int64
dtype: object
```

In [226…

```
cat=df[["tyre_season","month", "tread_depth","wiring_strength","tyre_quality","tread_ty
num=df[["vulc","perc_nat_rubber","weather","perc_imp","temperature", "elevation", "perc

print(cat.shape)
print(num.shape)
```

```
(3000, 7)
(3000, 7)
```

CATEGORICAL DATA

In [227…

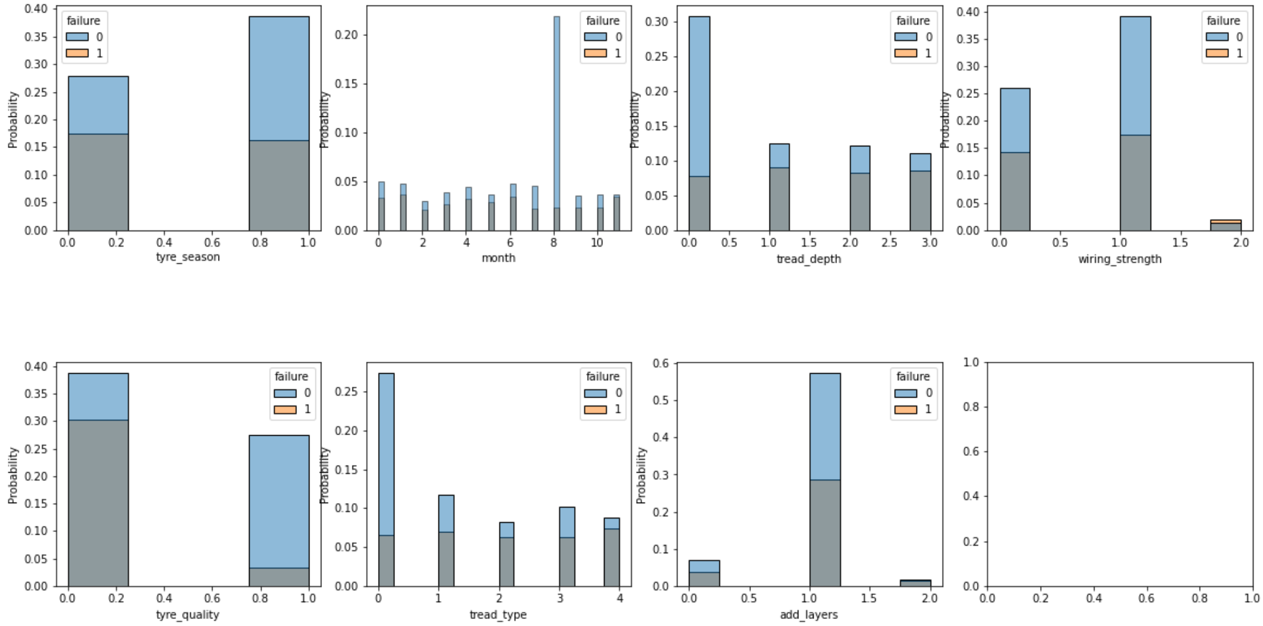
```
cat.head()
```

Out[227…

	tyre_season	month	tread_depth	wiring_strength	tyre_quality	tread_type	add_layers
0	1	8	0	1	1	0	1
1	0	11	1	1	0	2	1
2	0	0	1	1	0	0	1
3	0	7	3	1	1	1	1
4	1	2	2	2	0	0	1

In [228…

```
cat_plot=df[["tyre_season","month","tread_depth","wiring_strength","tyre_quality","trea
fig, axes = plt.subplots(2, 4,figsize=[16,8])
axes = axes.flatten()
fig.tight_layout(h_pad=10)
i=0
for x in cat.columns:
    sns.histplot(data=cat_plot, x=x, hue="failure",stat ='probability' , ax=axes[i], bi
    i=i+1
plt.show()
```



We can observe a peak of 0 labeled observations in correspondence of "month" = 8.This could be a noise-affected attribute.

So let's say how many 0 observation we have in september.

In [229…

```
#try to correct month 8
print(((df['month']==8) & (df['failure']==0)).sum())
print(((df['month']==8) & (df['failure']==1)).sum())

#too many items to delete o change the labels --> so i wont consider the month column
```

```
655
69
```

ENTROPY e GINI

We can measure homogeneity and heterogeneity of the data. In this way we can better understand how the failure distribution behave for each categorical attribute.

In [230…

```
import collections
def CountFrequency(arr):
    return collections.Counter(arr)

def entropy(x):
    e=0
    for i in range(0,len(set(x))):
        e=e+CountFrequency(x)[i]/len(x) * np.log2( CountFrequency(x)[i]/len(x) )
    return -e

#gini function
def gini(x):
    tot=0
    for i in range(0,len(set(x))):
        tot=tot+ (CountFrequency(x)[i]/len(x))**2
    return 1-tot
```

In [231…

```
categories=['tyre_season','month','tread_depth','wiring_strength','tyre_quality','tread

print("GINI --> 0.5 MAX ETEROGENETY\nGINI --> 0.0 MAX HOMOGENEITY**")
print("EI   --> 1   MAX ETEROGENETY\nEI   --> 0.0 MAX HOMOGENEITY**\n\n")
#gini index for categories:
for j in categories:
    print("-->",j,":\n")
    for i in set(df[j]):
        lis=(np.array([df[df[j]==i]['failure']])).astype(int)
        print(j ,"=",i)
        print('Gini index:', round(gini(lis[0]),3), " || Entropy index:", round(entropy

    print("##")
    print("\n")
```

```
GINI --> 0.5 MAX ETEROGENETY
GINI --> 0.0 MAX HOMOGENEITY**
EI   --> 1   MAX ETEROGENETY
EI   --> 0.0 MAX HOMOGENEITY**


--> tyre_season :

tyre_season = 0
Gini index: 0.474  || Entropy index: 0.962
tyre_season = 1
Gini index: 0.416  || Entropy index: 0.875
##


--> month :

month = 0
Gini index: 0.48  || Entropy index: 0.97
month = 1
Gini index: 0.491  || Entropy index: 0.987
month = 2
Gini index: 0.486  || Entropy index: 0.98
month = 3
Gini index: 0.482  || Entropy index: 0.973
month = 4
Gini index: 0.487  || Entropy index: 0.981
month = 5
Gini index: 0.492  || Entropy index: 0.988
month = 6
Gini index: 0.489  || Entropy index: 0.983
month = 7
Gini index: 0.442  || Entropy index: 0.915
month = 8
Gini index: 0.172  || Entropy index: 0.454
month = 9
Gini index: 0.476  || Entropy index: 0.965
month = 10
Gini index: 0.474  || Entropy index: 0.962
month = 11
Gini index: 0.499  || Entropy index: 0.999
##


--> tread_depth :

tread_depth = 0
```

```
Gini index: 0.321  || Entropy index: 0.724
tread_depth = 1
Gini index: 0.487  || Entropy index: 0.982
tread_depth = 2
Gini index: 0.482  || Entropy index: 0.973
tread_depth = 3
Gini index: 0.492  || Entropy index: 0.989
##


--> wiring_strength :

wiring_strength = 0
Gini index: 0.457  || Entropy index: 0.937
wiring_strength = 1
Gini index: 0.427  || Entropy index: 0.892
wiring_strength = 2
Gini index: 0.485  || Entropy index: 0.979
##


--> tyre_quality :

tyre_quality = 0
Gini index: 0.492  || Entropy index: 0.989
tyre_quality = 1
Gini index: 0.197  || Entropy index: 0.503
##


--> tread_type :

tread_type = 0
Gini index: 0.311  || Entropy index: 0.708
tread_type = 1
Gini index: 0.468  || Entropy index: 0.954
tread_type = 2
Gini index: 0.491  || Entropy index: 0.986
tread_type = 3
Gini index: 0.473  || Entropy index: 0.961
tread_type = 4
Gini index: 0.496  || Entropy index: 0.995
##


--> add_layers :

add_layers = 0
Gini index: 0.45  || Entropy index: 0.926
add_layers = 1
Gini index: 0.444  || Entropy index: 0.917
add_layers = 2
Gini index: 0.49  || Entropy index: 0.985
##
```

The high difference between the target distrbution in september is highlighted also by the gini and entropy indexes

DUMMIES

To properly treat the categorical variables, we first need to define N-1 binary variables Dj1,Dj2....DjN-1, called dummies varibles.

In [232…

```
#categorical variables have the "object" type
#Categorical variables must be changed in the pre-processing section
#since machine learning models require numeric input variables
cat = cat.astype(str)
cat.dtypes
```

Out[232…

```
tyre_season        object
month              object
tread_depth        object
wiring_strength    object
tyre_quality       object
tread_type         object
add_layers         object
dtype: object
```

In [233…

```
selection_categorical=['tyre_season','tread_depth','wiring_strength','tyre_quality','tr
dummies = pd.get_dummies(cat[selection_categorical])
dummies.tail()
```

Out[233…

	tyre_season_0	tyre_season_1	tread_depth_0	tread_depth_1	tread_depth_2	tread_depth_3	wiring_
2995	0	1	0	1	0	0	
2996	0	1	0	1	0	0	
2997	1	0	0	0	0	1	
2998	1	0	1	0	0	0	
2999	1	0	0	1	0	0	

5 rows × 31 columns

NUMERICAL DATA

In [234…

```
num.head()
```

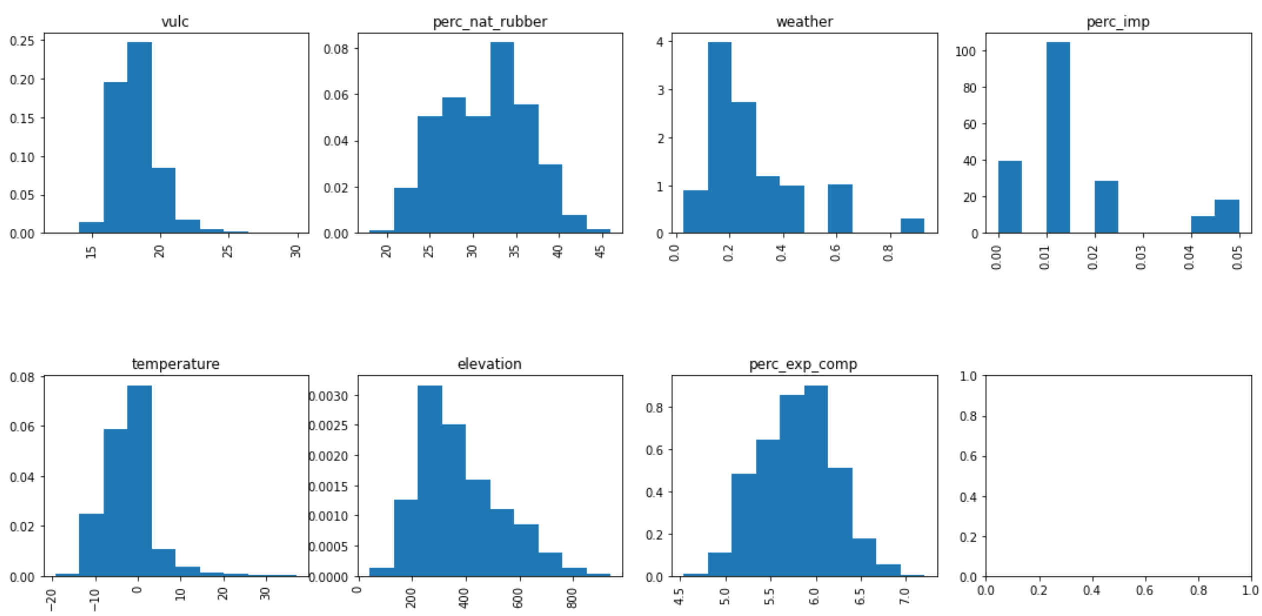
Out[234…

	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp
0	17.990	26	0.16	0.01	-8.12	332.5	5.13
1	20.704	36	0.30	0.01	-4.52	328.0	6.15
2	19.156	34	0.30	0.01	-1.08	247.0	6.36
3	16.802	35	0.19	0.02	7.44	408.0	6.62
4	17.140	23	0.39	0.01	30.52	308.0	6.15

In [235…

```
fig, axes = plt.subplots(2, 4,figsize=[15,7])
axes = axes.flatten()
```

```python
fig.tight_layout(h_pad=10)

i=0
for x in num.columns:
    plt.sca(axes[i]) # set the current Axes
    plt.hist(num[x],density=True)
    plt.xticks(rotation = 90) # Rotates X-Axis Ticks by 45-degrees
    plt.title(x)
    i+=1
plt.show()
```

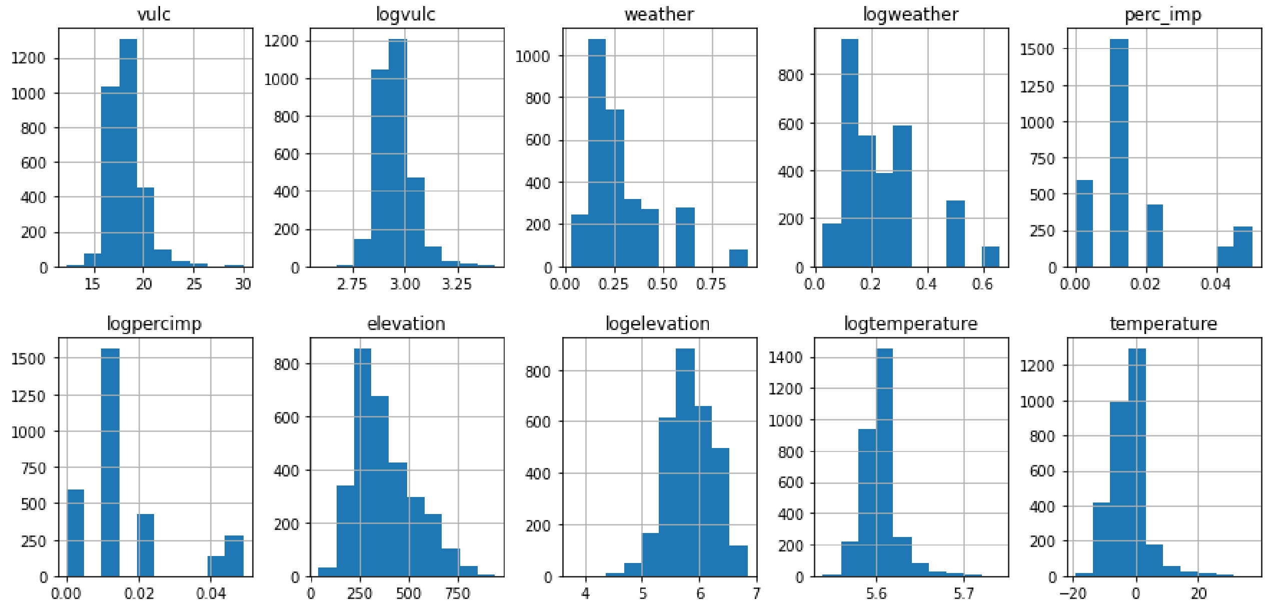


In [236…

```python
import math
pd.options.mode.chained_assignment = None # no warning on creating a new column

num['logvulc']=num['vulc'].apply(lambda x: math.log(x+1))
num['logweather']=num['weather'].apply(lambda x: math.log(x+1))
num['logpercimp']=num['perc_imp'].apply(lambda x: math.log(x+1))
num['logelevation']=num['elevation'].apply(lambda x: math.log(x+1))
num['logtemperature']=(num['temperature']+273.15).apply(lambda x: math.log(x+1))
num[['vulc','logvulc',
     'weather','logweather',
     'perc_imp','logpercimp',
     'elevation','logelevation',
     'logtemperature','temperature']].hist(layout=(2,5), figsize=(15,7))
```

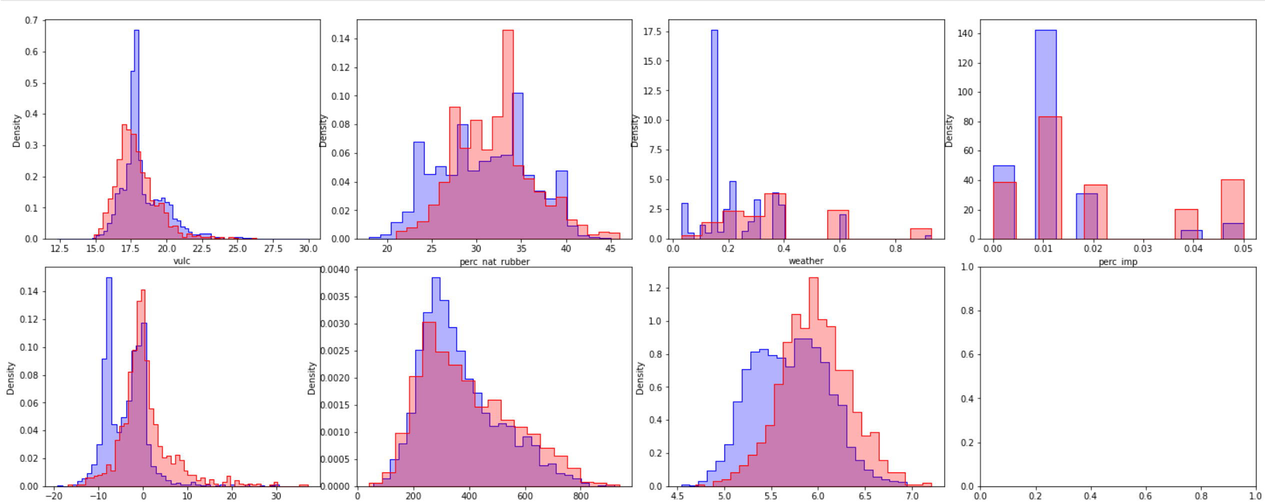
Out[236…

```
array([[<AxesSubplot:title={'center':'vulc'}>,
        <AxesSubplot:title={'center':'logvulc'}>,
        <AxesSubplot:title={'center':'weather'}>,
        <AxesSubplot:title={'center':'logweather'}>,
        <AxesSubplot:title={'center':'perc_imp'}>],
       [<AxesSubplot:title={'center':'logpercimp'}>,
        <AxesSubplot:title={'center':'elevation'}>,
        <AxesSubplot:title={'center':'logelevation'}>,
        <AxesSubplot:title={'center':'logtemperature'}>,
        <AxesSubplot:title={'center':'temperature'}>]], dtype=object)
```



In [237…

```
X0 = num[df['failure']==0]
X1 = num[df['failure']==1]

fig, axes = plt.subplots(ncols=4, nrows=2, figsize=(20,8))
fig.tight_layout()

for i, ax in zip(range(cat.columns.size), axes.flat):
    sns.histplot(X0.iloc[:,i], color="blue", ax=ax, stat='density', element="step",  al
    sns.histplot(X1.iloc[:,i], color="red", ax=ax,stat='density', element="step",  alph
plt.show()

##provare ad applicare quache trasf
```



The log transformation distributions are quite similar to the orginal one so we kept the originals one. Since we do not exclude any numerical features except diameter and we do not apply any numerical transformation let's consider the columns of interest as follows:

In [238…

```
#at the end let's take the column of interest
num=df[["vulc", "perc_nat_rubber","weather","perc_imp","temperature","elevation","perc_
```

ANALISI PCA

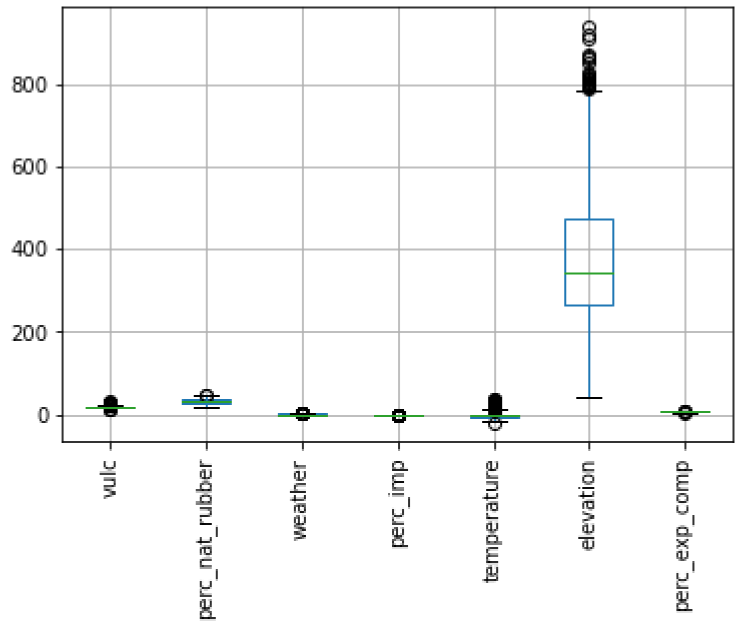
The purpose of this method is to obtain a projective transformation that replaces a subset of the original numerical attributes with a lower number of new attributes obtained as their linear combination, without this change causing a loss of information.

In [239…

```
num.boxplot(rot=90)

#elevation has totally different numbers --> standardization
```

Out[239…

```
<AxesSubplot:>
```



to properly apply PCA analysis standardization is required

In [240…

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(num) # No target
scaled_num = pd.DataFrame(scaler.transform(num))
scaled_num.columns = num.columns
scaled_num.tail()
```

Out[240…

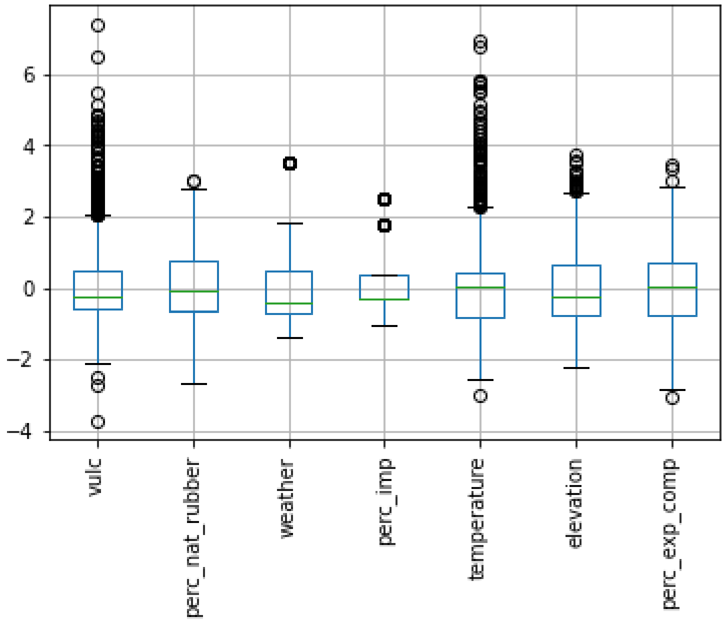
	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp
2995	-0.231083	-0.456093	0.584065	-0.319087	1.702514	-0.586827	-0.271809
2996	-0.698653	-0.253355	-0.343773	-1.020379	0.164930	-1.480129	0.045070
2997	-1.269567	0.354859	0.584065	-0.319087	-0.187726	-0.934222	-0.539937
2998	0.433093	1.165810	-1.380768	-1.020379	0.284834	-0.570284	0.240072
2999	1.315301	0.354859	-1.217032	-1.020379	0.912563	0.190677	0.508200

In [241…

```
scaled_num.boxplot(rot=90)
#ok!
```

Out[241…

<AxesSubplot:>



In [242…

```
#PCA fit
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(scaled_num)
```

Out[242…

PCA()

In [243…

```
#let's use the pca to transform the dataset
num_pca = pd.DataFrame(pca.transform(scaled_num))
num_pca.columns =['PC1','PC2','PC3','PC4','PC5','PC6','PC7']
num_pca
```

Out[243…

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
0	-2.039807	-1.005052	0.143276	0.280682	-0.055187	0.362670	0.070427
1	0.230444	1.339307	1.267905	-0.763265	0.159000	-0.653739	-0.277058
2	0.623754	0.939546	0.072528	-1.191816	0.466955	-0.734820	-0.208645
3	1.967707	1.162369	-1.581563	-0.294469	0.319924	-0.303452	0.923830
4	2.771142	0.328019	-2.985996	-0.799060	3.063998	3.395502	-0.162010
...	...	...	...	...	...	...	...
2995	0.625458	-0.250981	-0.783154	-0.636191	0.673647	1.360141	-0.495100
2996	-0.779224	-0.093864	-1.159095	-1.344233	0.107791	-0.019557	-0.391418
2997	-0.118266	-0.862350	-0.778459	-0.789547	-1.028185	0.076134	-0.574664
2998	-0.695504	1.737254	-0.289416	-0.992616	-0.390970	0.111279	0.471484
2999	-0.383800	2.064986	0.069821	-0.266022	0.816098	0.511576	0.301391

3000 rows × 7 columns

In [244…

```
pd.DataFrame(pca.components_,index=['PC1','PC2','PC3','PC4','PC5','PC6','PC7'],columns=
```

Out[244…

	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp
PC1	-0.080281	0.292608	0.509765	0.442437	0.469686	0.157051	0.455033
PC2	0.388180	0.472226	-0.373902	-0.496069	0.214217	0.262835	0.354203
PC3	0.796293	0.176996	0.232871	0.334312	-0.341043	0.047571	-0.223659
PC4	-0.104280	-0.256250	-0.006530	0.057527	-0.132234	0.945590	-0.092107
PC5	0.425614	-0.764839	-0.071712	0.003050	0.302535	-0.083554	0.360851
PC6	0.124018	0.013110	0.024990	-0.070775	0.705190	0.052928	-0.691896
PC7	-0.037241	0.096131	-0.735050	0.661871	0.104354	-0.008160	0.006628

In [245…

```
pd.DataFrame(pca.explained_variance_).transpose()
```

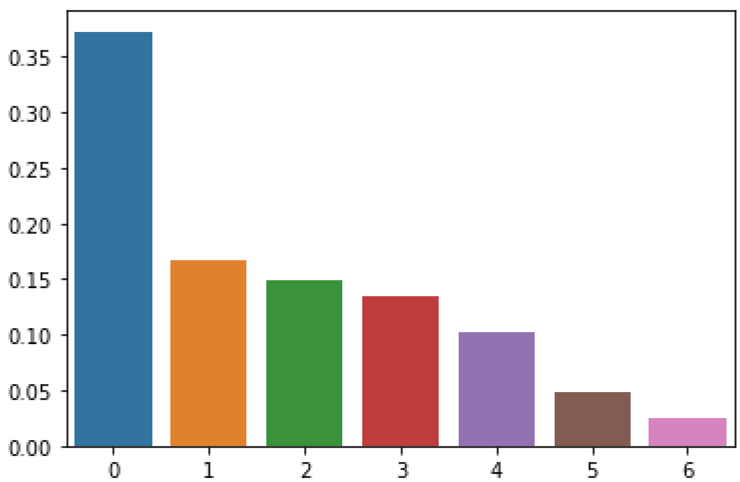
Out[245…

	0	1	2	3	4	5	6
0	2.611119	1.17498	1.042926	0.946682	0.7149	0.332083	0.179644

In [246…

```
#VISUALIZE The percentage of variance explained by each of the selected components.
explained_var=pd.DataFrame(pca.explained_variance_ratio_).transpose()
print(explained_var)
ax = sns.barplot( data=explained_var)
```

```
          0         1        2         3         4         5         6
0  0.372893  0.167798  0.14894  0.135195  0.102095  0.047425  0.025655
```



In [247…

```
cum_explained_var=np.cumsum(pca.explained_variance_ratio_)
cum_explained_var= pd.DataFrame(cum_explained_var).transpose()
cum_explained_var
```

Out[247…

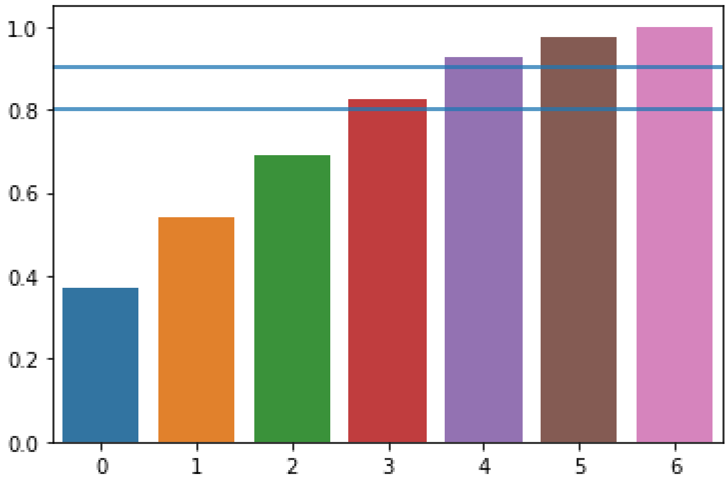
	0	1	2	3	4	5	6
0	0.372893	0.540691	0.689631	0.824826	0.92692	0.974345	1.0

In [248…

```
ax = sns.barplot(data=cum_explained_var)
ax.axhline(0.9)
ax.axhline(0.8)
```

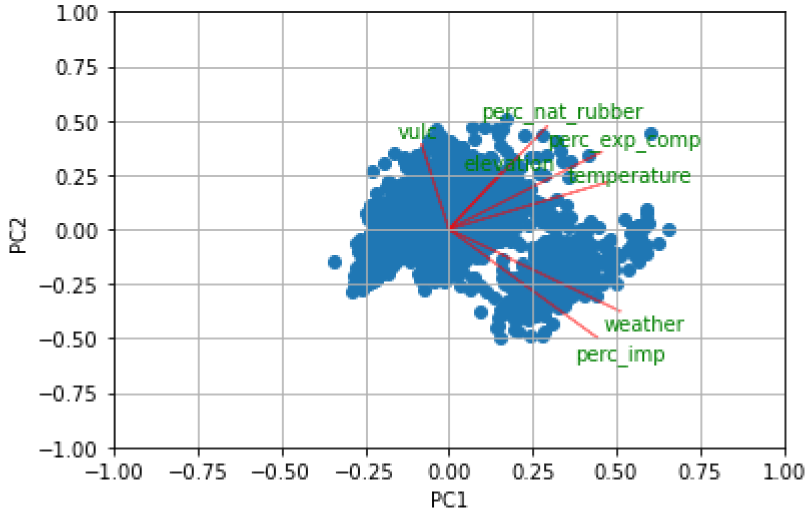
Out[248…

```
<matplotlib.lines.Line2D at 0x2b200af9438>
```



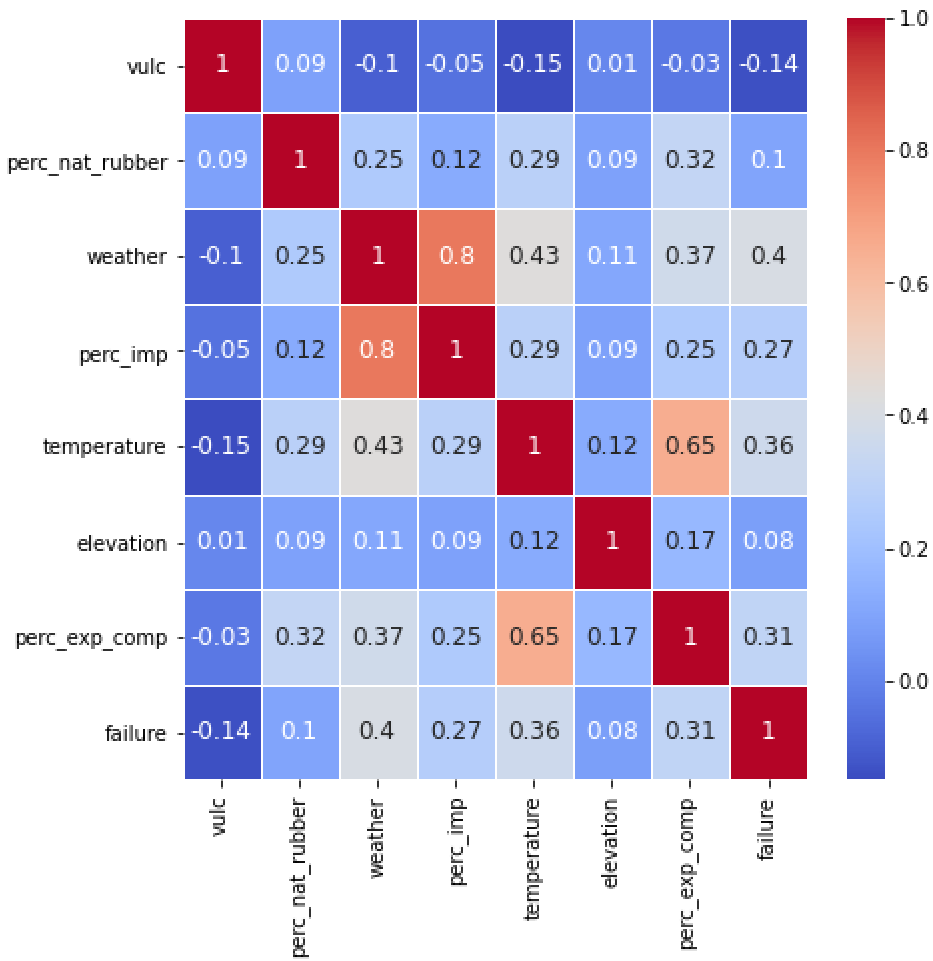
In [249…

```
def myplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    plt.scatter(xs * scalex,ys * scaley)
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'g',
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha =
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()

#Call the function. Use only the 2 PCs.
myplot(pca.transform(scaled_num)[:,0:2],np.transpose(pca.components_[0:2, :]), num.colu
plt.show()



#perc_imp e wheater sono correlate?
```



In [250…

```
scaled_num['failure']=df['failure']
plt.figure(figsize = (7,7))
sns.heatmap(data=scaled_num.corr().round(2), cmap='coolwarm', linewidths=.5, annot=True
plt.show()

#We see that there is a numerical correlation between "weather" and "perc_imp"
#but we can not rationally explain it.
```



	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp	failure
vulc	1	0.09	-0.1	-0.05	-0.15	0.01	-0.03	-0.14
perc_nat_rubber	0.09	1	0.25	0.12	0.29	0.09	0.32	0.1
weather	-0.1	0.25	1	0.8	0.43	0.11	0.37	0.4
perc_imp	-0.05	0.12	0.8	1	0.29	0.09	0.25	0.27
temperature	-0.15	0.29	0.43	0.29	1	0.12	0.65	0.36
elevation	0.01	0.09	0.11	0.09	0.12	1	0.17	0.08
perc_exp_comp	-0.03	0.32	0.37	0.25	0.65	0.17	1	0.31
failure	-0.14	0.1	0.4	0.27	0.36	0.08	0.31	1

At the end of the PCA analysis we did not found any interest information to describe the data distribution in a convinient way. PCA was not useful to obtain a better interpretation of the data.

CLASSIFICATION

In [251…

```
num.head()
```

Out[251…

	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp
0	17.990	26	0.16	0.01	-8.12	332.5	5.13
1	20.704	36	0.30	0.01	-4.52	328.0	6.15
2	19.156	34	0.30	0.01	-1.08	247.0	6.36
3	16.802	35	0.19	0.02	7.44	408.0	6.62
4	17.140	23	0.39	0.01	30.52	308.0	6.15

In [252…

```
dummies.tail()
```

Out[252…

	tyre_season_0	tyre_season_1	tread_depth_0	tread_depth_1	tread_depth_2	tread_depth_3	wiring_
2995	0	1	0	1	0	0	
2996	0	1	0	1	0	0	
2997	1	0	0	0	0	1	
2998	1	0	1	0	0	0	
2999	1	0	0	1	0	0	

5 rows × 31 columns

CREATION OF X AND Y

In [253…

```
X=pd.concat([num,dummies], axis = 1)
print(X.shape)
X.tail()
```

```
(3000, 38)
```

Out[253…

	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp	tyre_season_
2995	17.818	29	0.39	0.01	7.28	287.5	5.68	
2996	17.076	30	0.22	0.00	-1.44	152.5	5.81	
2997	16.170	33	0.39	0.01	-3.44	235.0	5.57	
2998	18.872	37	0.03	0.00	-0.76	290.0	5.89	
2999	20.272	33	0.06	0.00	2.80	405.0	6.00	

5 rows × 38 columns

In [254…

```
y=df['failure']
print(len(y))
```

```
3000
```

SPLIT DATA

- TRAIN SET 70%
- TEST SET 30%

In [255…

```
from sklearn.model_selection import train_test_split

#SPLIT DATA INTO TRAIN AND TEST SET
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size =0.30,
                                                    stratify=y,        #preserve target
                                                    random_state= 321) #fix random seed

print( X_train.shape, X_test.shape)
```

```
(2100, 38) (900, 38)
```

SCALER

Only for the training set.

In [256…

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train[num.columns])
X_train[num.columns] = scaler.transform(X_train[num.columns])
#scaled_train_num.columns =num.columns

X_train.head()
```

Out[256…

	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp	tyre_sea
1478	-1.272006	0.538925	3.570635	1.802901	0.219643	-0.463013	1.196003	
1575	0.580725	0.338475	-0.664065	-0.309088	-1.260420	0.804318	-1.124672	
333	-0.755024	-1.064678	-1.269022	-1.013084	0.240787	3.226475	0.389873	
1440	1.241596	-0.262876	0.545849	1.802901	1.904097	1.247718	1.391428	
1023	-0.367924	-1.064678	-0.664065	-0.309088	-1.133558	0.483349	-1.149101	

5 rows × 38 columns

In [257…

```
#Now lets's apply the scaler to the test set
X_test[num.columns] = scaler.transform(X_test[num.columns])
X_test.head()
```

Out[257…

	vulc	perc_nat_rubber	weather	perc_imp	temperature	elevation	perc_exp_comp	tyre_sea
364	-1.459189	-0.062426	-0.939045	-1.013084	-0.372382	-1.128113	0.927293	
2163	-0.244409	-1.465579	-0.664065	-0.309088	-1.056030	-0.099028	-1.149101	
12	-0.278789	-1.465579	-0.664065	-0.309088	-0.851641	-0.565591	-0.562825	
606	0.333694	-1.866480	0.490853	0.394908	3.856371	1.611703	1.122718	
2939	-0.552561	0.138024	-0.609069	-0.309088	-0.111609	-0.486176	-0.367399	

5 rows × 38 columns

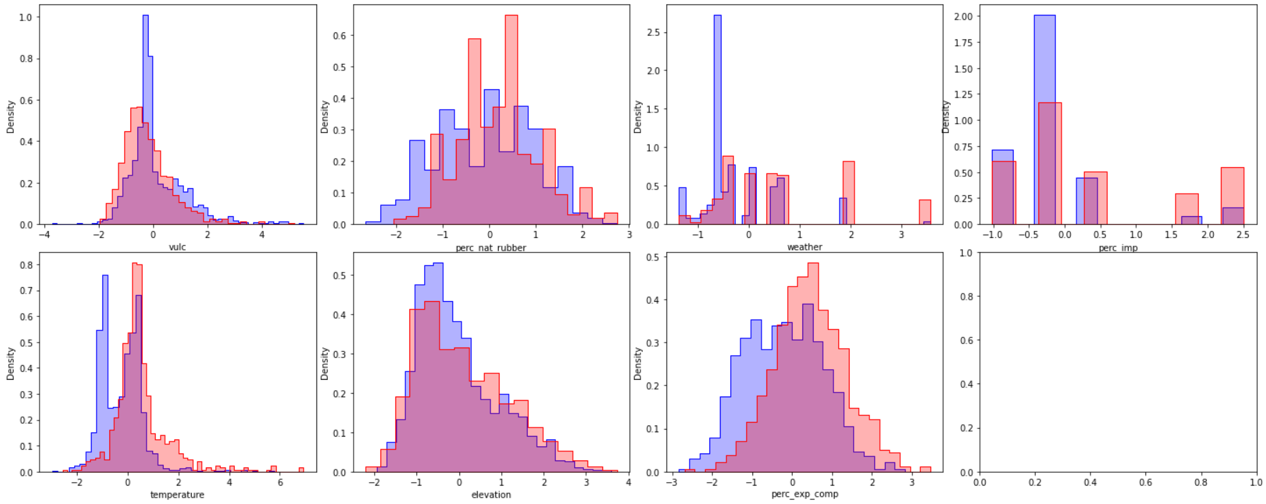
In [258…

```
#Save the scaler
import pickle
pickle.dump(scaler, open('scaler_NOT_OVERSAMPLED.pkl', 'wb'))
```

In [259…

```
X0 = X_train[df['failure']==0]
X1 = X_train[df['failure']==1]

fig, axes = plt.subplots(ncols=4, nrows=2, figsize=(20,8))
fig.tight_layout()

for i, ax in zip(range(num.columns.size), axes.flat):
    sns.histplot(X0.iloc[:,i], color="blue", ax=ax, stat='density', element="step",  al
    sns.histplot(X1.iloc[:,i], color="red", ax=ax,stat='density', element="step",  alph
plt.show()
```

```
C:\Users\scrpa\miniconda3\envs\myenv\lib\site-packages\ipykernel_launcher.py:1: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
C:\Users\scrpa\miniconda3\envs\myenv\lib\site-packages\ipykernel_launcher.py:2: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
```



DATA DISCRETIZATION

In order to make learning algorithms more efficient we perfrom also a data discretization of some numerical attributes.

The general purpose of data reduction methods is to obtain a decrease in the number of distinct values assumed by one or more attributes. Data discretization is the primary reduction method. On the one hand, it reduces continuous attributes to categorical attributes characterized by a limited number of distinct values. On the other hand, its aim is to significantly reduce the number of distinct values assumed by the categorical attributes

(do not run the following cells to the optimal analysis--> skip to the "model" section)

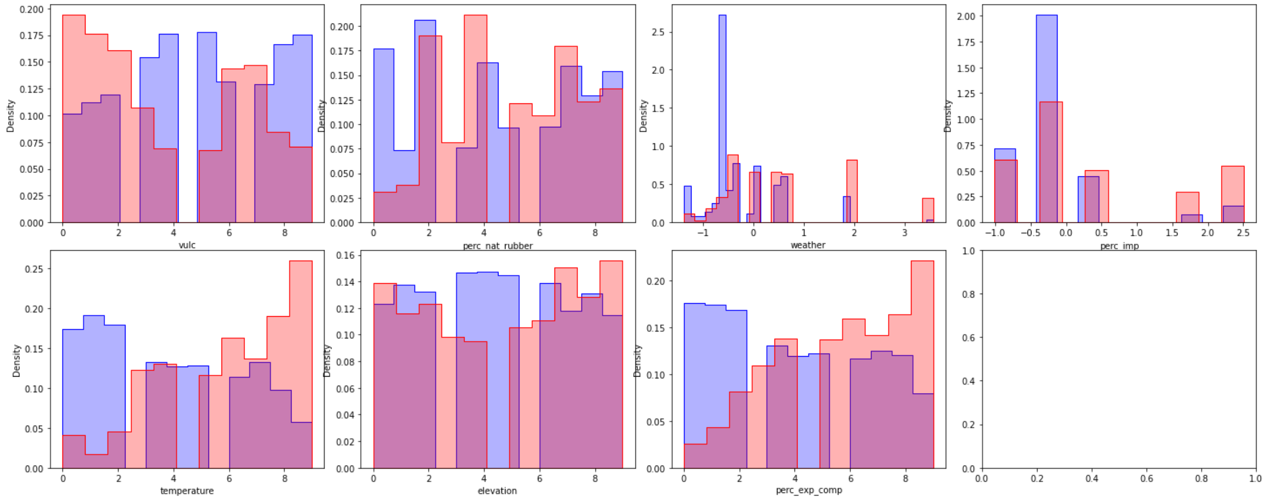
In [43]:

```
from sklearn.preprocessing import KBinsDiscretizer

sel=['vulc','temperature','elevation','perc_nat_rubber','perc_exp_comp']
# transform the dataset with KBinsDiscretizer
enc = KBinsDiscretizer(n_bins=10, encode="ordinal")
for  i in sel:
    X_train[i] = enc.fit_transform((np.array(X_train[i])).reshape(-1, 1))
    X_test[i]=   enc.fit_transform((np.array(X_test[i])).reshape(-1, 1))
```

In [44]:

```
X0 = X_train[df['failure']==0]
X1 = X_train[df['failure']==1]

fig, axes = plt.subplots(ncols=4, nrows=2, figsize=(20,8))
fig.tight_layout()

for i, ax in zip(range(num.columns.size), axes.flat):
    sns.histplot(X0.iloc[:,i], color="blue", ax=ax, stat='density', element="step",  al
    sns.histplot(X1.iloc[:,i], color="red", ax=ax,stat='density', element="step",  alph
plt.show()

print("Result of the data discretization")
```

```
C:\Users\scrpa\miniconda3\envs\myenv\lib\site-packages\ipykernel_launcher.py:1: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
C:\Users\scrpa\miniconda3\envs\myenv\lib\site-packages\ipykernel_launcher.py:2: UserWarn
ing: Boolean Series key will be reindexed to match DataFrame index.
```




```
Result of the data discretization
```

The results afeter the discretization does not change so much.

MODELS

In [260…

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
```

In [261…

```python
#funzioni


def hyperp_search(classifier, parameters):
    gs = GridSearchCV(classifier, parameters, cv=3, scoring = 'f1', verbose=10, n_jobs=
    gs = gs.fit(X_train, y_train)
    print("f1_train: %f using %s" % (gs.best_score_, gs.best_params_))

    best_model = gs.best_estimator_
    y_pred = best_model.predict(X_test)
    y_pred_train = best_model.predict(X_train)
    print("\n")
    print("f1         train %.3f   test %.3f" % (f1_score(y_train, y_pred_train), f1_sc
    print("\n")
    print(confusion_matrix(y_test, y_pred))
    return ( f1_score(y_train, y_pred_train),f1_score(y_test, y_pred) )

def roc(model,X_train,y_train,X_test,y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    y_probs = model.predict_proba(X_test) #predict_proba gives the probabilities for th
```

```
    fpr, tpr, thresholds1=metrics.roc_curve(y_test,  y_probs[:,1])

    import matplotlib.pyplot as plt
    plt.plot(fpr, tpr, label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

    auc = metrics.roc_auc_score(y_test, y_probs[:,1])
    print('AUC: %.3f' % auc)
    return (fpr, tpr ,auc)
```

KNN

In [262…

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier()
parameters = {'n_neighbors':np.arange(1,500,1)}
knn_f1_train,knn_f1_test = hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 499 candidates, totalling 1497 fits
f1_train: 0.550428 using {'n_neighbors': 21}


f1         train 0.585   test 0.527


[[517  81]
 [165 137]]
```

TREE

In [263…

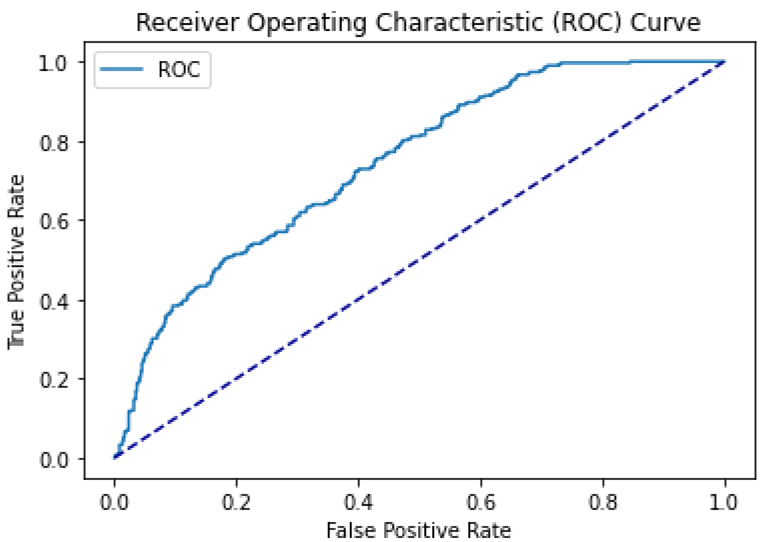
```
#Tree
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
parameters = {'criterion': ['entropy','gini'],
              'max_depth':          np.arange(5,100,5),
              'min_samples_split':  np.arange(5,100,1),
              'min_samples_leaf':   [2,4,6,7]}

tree_f1_train,tree_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 14440 candidates, totalling 43320 fits
f1_train: 0.555800 using {'criterion': 'gini', 'max_depth': 40, 'min_samples_leaf': 7,
'min_samples_split': 18}


f1         train 0.786   test 0.569


[[465 133]
 [129 173]]
```

Naive Bayes

In [264…

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB #or alternative NB implementations

model = GaussianNB()

model.fit(X_train, y_train)
y_pred=model.predict(X_test)



y_probs = model.predict_proba(X_test)
fpr3,tpr3,AUC3=roc(model,X_train,y_train,X_test,y_test)
```




```
AUC: 0.744
```

Logistic

In [265…

```
# Logistic

from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()
parameters = {"C":np.arange(1,10,1), "max_iter":[2000] }


logi_f1_train,logi_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
f1_train: 0.563489 using {'C': 5, 'max_iter': 2000}


f1          train 0.577    test 0.623


[[513  85]
 [127 175]]
```

SUPPORT VECTOR MACHINE

In [266…

```
from sklearn.svm import SVC

classifier = SVC()
parameters = {"kernel":['linear','sigmoid','rbf'],
              "C":[0.001,0.3,1],
              "degree":[2,3],
              "gamma":[1]}

SV_f1_train,SV_f1_test=hyperp_search(classifier,parameters)


#OVER-FITTING: using {'C': 50, 'kernel': 'rbf'}
# so we omit the 'rbf' among the possible parameters.
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
f1_train: 0.540987 using {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'linear'}


f1          train 0.548   test 0.593


[[535  63]
 [148 154]]
```

NEURAL NETWORK

In [267…

```
# Multi-layer Perceptron classifier

from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
parameters = {"hidden_layer_sizes":[(9,6),(10, 5)],
              "alpha": [0.01,1,10],
              "activation":['logistic', 'relu'],
              "learning_rate":['constant','invscaling'], #'adaptive'],
              "max_iter": [3000]}



NN_f1_train,NN_f1_test=hyperp_search(classifier,parameters)

#over fitting with: 'alpha': 0.1, 'hidden_layer_sizes': (100, 20, 5), 'max_iter': 2000
#over fitting
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
f1_train: 0.571338 using {'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes':
(10, 5), 'learning_rate': 'invscaling', 'max_iter': 3000}


f1          train 0.740   test 0.574


[[483 115]
 [134 168]]
```

RANDOM FOREST

In [268…

```
from sklearn.ensemble import RandomForestClassifier


classifier= RandomForestClassifier()
parameters = {'n_estimators' :     [8,10,100],
              'criterion' :        ['entropy', 'gini'],
              'max_depth' :        np.arange(5,10,1),
              'min_samples_split': np.arange(5,10,1),
              'min_samples_leaf' : [2,4,6,8,]
             };


RF_f1_train,RF_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 600 candidates, totalling 1800 fits
f1_train: 0.555948 using {'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 2,
'min_samples_split': 6, 'n_estimators': 10}


f1         train 0.720   test 0.587


[[532  66]
 [149 153]]
```

ADA BOOST

In [269…

```
from sklearn.ensemble import AdaBoostClassifier


classifier= AdaBoostClassifier()
parameters = {'n_estimators'  : [7000],
              'learning_rate' : [0.01,0.1]}

ADAB_f1_train,ADAB_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
f1_train: 0.546213 using {'learning_rate': 0.01, 'n_estimators': 7000}


f1         train 0.620   test 0.588


[[497 101]
 [134 168]]
```

In [270…

```
F1_TRAIN=[knn_f1_train,tree_f1_train,logi_f1_train,SV_f1_train,NN_f1_train,RF_f1_train,
F1_TEST= [knn_f1_test, tree_f1_test, logi_f1_test, SV_f1_test, NN_f1_test, RF_f1_test,A
data = {'Name': ['knn','tree','Logistic','SVM','Neural Network','Random Forest','ADABOO
data['f1_train']=F1_TRAIN
data['f1_test']=F1_TEST

data_no_oversampling=pd.DataFrame(data)
```

In [271…

```
data_no_oversampling
```

Out[271…

	Name	f1_train	f1_test
0	knn	0.585204	0.526923
1	tree	0.785714	0.569079
2	Logistic	0.576985	0.622776
3	SVM	0.547969	0.593449
4	Neural Network	0.740192	0.574359
5	Random Forest	0.720131	0.587332
6	ADABOOST	0.619718	0.588441

OVER SAMPLING

We oversample the train data and not the test data since if train data is unbalanced, our validation data will most likely show the same trait and be unbalanced.

In [272…

```
print(len(y_train[y_train==0]))
print(len(y_train[y_train==1]))
```

```
1394
706
```

In [273…

```
 from sklearn.utils import resample

df_train=pd.concat([X_train,y_train], axis = 1)

#Down-sample Majority Class
#1) Separate majority and minority classes

df_majority = df_train[df_train.failure==0]
df_minority = df_train[df_train.failure==1]
#2) Oversample minority class
df_minority_oversampled = resample(df_minority,
                                   replace=True,
                                   n_samples=len(y_train[y_train==0]),    # number of s
                                   random_state=123)                      # reproducibl

#3) Combine oversampled minority class with majority class
df_train_oversampled = pd.concat([df_minority_oversampled, df_majority])

#4) Display new class counts
df_train_oversampled.failure.value_counts()
```

Out[273…

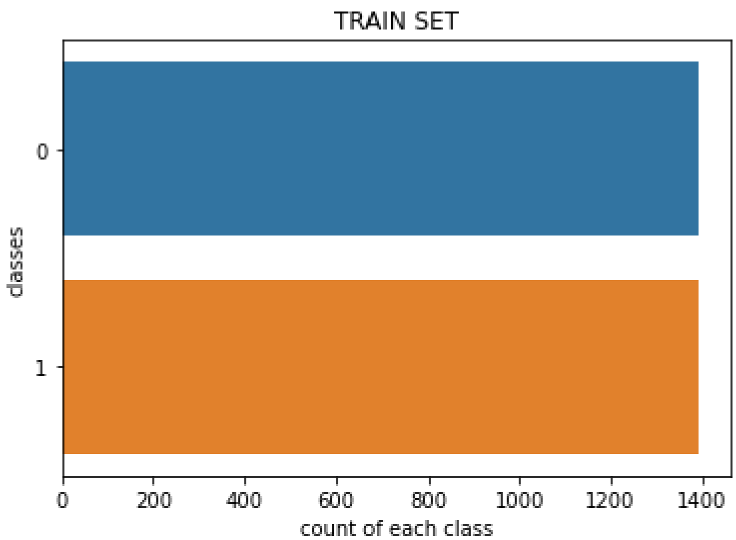
```
0    1394
1    1394
Name: failure, dtype: int64
```

In [274…

```
y_train=df_train_oversampled['failure']
X_train=df_train_oversampled.loc[:, df_train_oversampled.columns!='failure']
```

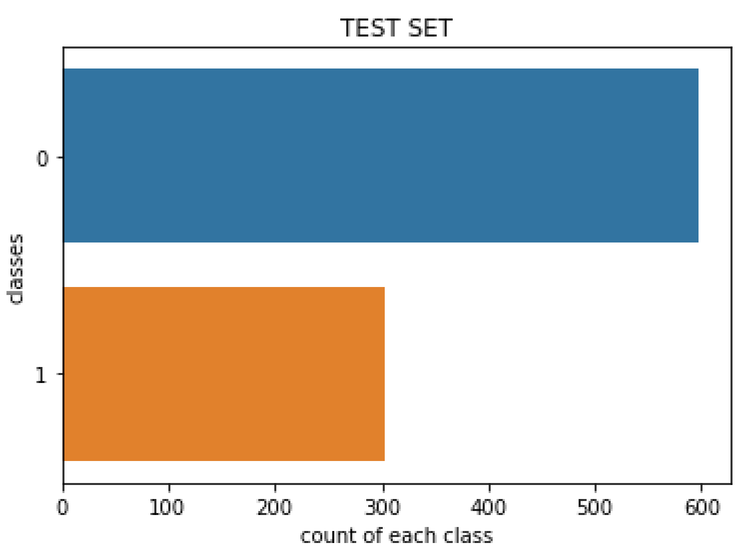
In [275…

```
#Visualize Class Counts
sns.countplot(y=y_train[:])
plt.xlabel("count of each class")
plt.ylabel("classes")
plt.title("TRAIN SET")
plt.show()
```



In [276…

```
#Visualize Class Counts
sns.countplot(y=y_test[:])
plt.xlabel("count of each class")
plt.ylabel("classes")
plt.title("TEST SET")
plt.show()
```



KNN - OS

In [277…

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier()
parameters = {'n_neighbors':np.arange(2,10,1)}
knn_f1_train,knn_f1_test = hyperp_search(classifier,parameters)

print("DIfference between train-test")
print(knn_f1_train-knn_f1_test)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
f1_train: 0.753678 using {'n_neighbors': 3}


f1         train 0.892   test 0.566


[[406 192]
 [107 195]]
DIfference between train-test
0.3261978732540225
```

TREE - OS

In [278…

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
parameters = {'criterion': ['entropy','gini'],
              'max_depth':         [3,4,5,6],
              'min_samples_split':  np.arange(4,24,4),
              'min_samples_leaf':   [4]}

tree_f1_train,tree_f1_test=hyperp_search(classifier,parameters)
print(tree_f1_train-tree_f1_test)
```

```
Fitting 3 folds for each of 40 candidates, totalling 120 fits
f1_train: 0.736549 using {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 4,
'min_samples_split': 16}


f1         train 0.765   test 0.609


[[336 262]
 [ 55 247]]
0.15552604214767096
```

LOGISTIC - OS

In [279…

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
parameters = {"C":[0.15,0.01,1], "max_iter":[1000] }

logi_f1_train,logi_f1_test=hyperp_search(classifier,parameters)
print(logi_f1_train-logi_f1_test)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
f1_train: 0.723685 using {'C': 0.15, 'max_iter': 1000}
```

```
f1         train 0.732   test 0.642


[[407 191]
 [ 69 233]]
0.09005654632448878
```

SUPPORT VECTOR MACHINE OS

In [280…

```
from sklearn.svm import SVC
classifier = SVC()
parameters = {"kernel":['linear','sigmoid','rbf'],
              "C":[0.001,0.01,0.3],
              "degree":[2,3,4],
              "gamma": [1]}

SV_f1_train,SV_f1_test=hyperp_search(classifier,parameters)
print(SV_f1_train-SV_f1_test)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
f1_train: 0.732469 using {'C': 0.3, 'degree': 2, 'gamma': 1, 'kernel': 'linear'}


f1         train 0.746   test 0.643


[[400 198]
 [ 65 237]]
0.10296609275806534
```

NEURAL NETWORKS - OS

In [281…

```
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
parameters = {"hidden_layer_sizes":[(9,6),(10, 5)],
              "alpha": [0.01,1,10],
              "activation":['logistic', 'relu'],
              "learning_rate":['constant','invscaling'], #'adaptive'],
              "max_iter": [3000]}

NN_f1_train,NN_f1_test=hyperp_search(classifier,parameters)
print(NN_f1_train-NN_f1_test)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
f1_train: 0.783095 using {'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (1
0, 5), 'learning_rate': 'invscaling', 'max_iter': 3000}


f1         train 0.853   test 0.579


[[444 154]
 [116 186]]
0.2732519657938629
```

RANDOM FOREST CLASSIFIER - OS

In [282…

```
from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier()
parameters = {'criterion' :        ['entropy', 'gini'],
              'max_depth' :        np.arange(4,5,1),
              'min_samples_leaf' : np.arange(5,30,5),
              'min_samples_split': np.arange(10,500,10),
              'n_estimators' :     np.arange(2,10,1),
             };

RF_f1_train,RF_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 3920 candidates, totalling 11760 fits
f1_train: 0.762042 using {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 5,
'min_samples_split': 10, 'n_estimators': 9}


f1         train 0.755   test 0.643


[[383 215]
 [ 57 245]]
```

ADABOOST OS

In [283…

```
from sklearn.ensemble import AdaBoostClassifier


classifier= AdaBoostClassifier()
parameters = {'n_estimators'   : [1000,2000,3000,4000,5000,6000],
              'learning_rate' : [0.001, 0.01]}

ADAB_f1_train,ADAB_f1_test=hyperp_search(classifier,parameters)
print(ADAB_f1_train - ADAB_f1_test)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
f1_train: 0.734872 using {'learning_rate': 0.01, 'n_estimators': 6000}


f1         train 0.773   test 0.645


[[415 183]
 [ 71 231]]
0.12808193668528856
```

In [284…

```
F1_TRAIN=[knn_f1_train,tree_f1_train,logi_f1_train,SV_f1_train,NN_f1_train,RF_f1_train,
F1_TEST= [knn_f1_test, tree_f1_test, logi_f1_test, SV_f1_test, NN_f1_test, RF_f1_test,A
data = {'Name': ['knn','tree','Logistic','SVM','Neural Network','Random Forest','ADABOO
data['f1_train']=F1_TRAIN
data['f1_test']=F1_TEST

data_oversampling=pd.DataFrame(data)
```

In [285…

```
data_oversampling
```

Out[285...

	Name	f1_train	f1_test
0	knn	0.892236	0.566038
1	tree	0.764651	0.609125
2	Logistic	0.731930	0.641873
3	SVM	0.746114	0.643148
4	Neural Network	0.852691	0.579439
5	Random Forest	0.755203	0.643045
6	ADABOOST	0.773333	0.645251

OVERSAMPLING WITH ANOTHER METHOD

Scikit-learn's imblearn has a class named SMOTETomek that combines the concepts of under-sampling and oversampling techniques, theoretically providing a good compromise between the pros and cons of each.

In [286...

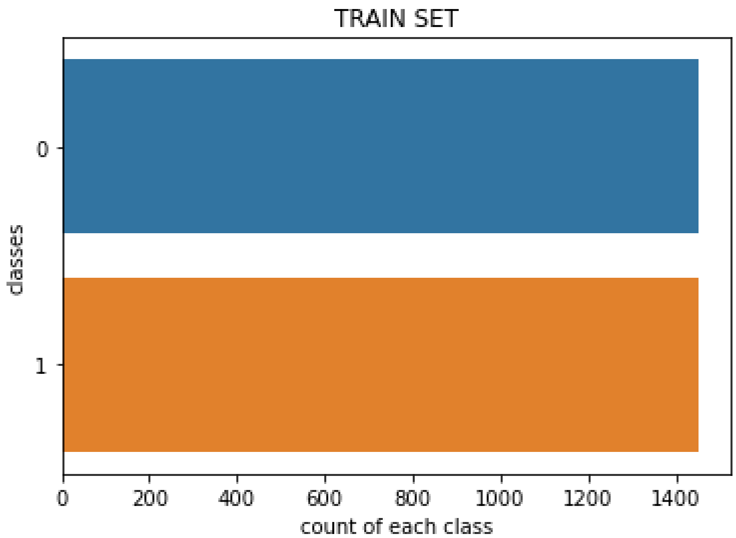
```
#same process
X=pd.concat([num,dummies], axis = 1)
y=df['failure']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size =0.25,
                                                    stratify=y,        #preserve target
                                                    random_state= 321) #fix random seed
# same scaler
X_train[num.columns] = scaler.transform(X_train[num.columns])
X_test[num.columns] = scaler.transform(X_test[num.columns])
```

In [287...

```
from imblearn.combine import SMOTETomek
smt = SMOTETomek()

X_train, y_train= smt.fit_resample(X_train, y_train)
```

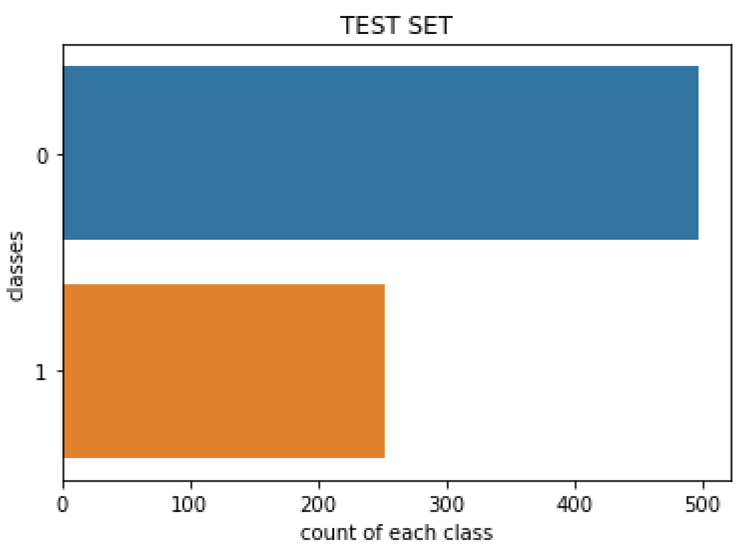
In [288...

```
#Visualize Class Counts
sns.countplot(y=y_train[:])
plt.xlabel("count of each class")
plt.ylabel("classes")
plt.title("TRAIN SET")
plt.show()
```



In [289…

```
#Visualize Class Counts
sns.countplot(y=y_test[:])
plt.xlabel("count of each class")
plt.ylabel("classes")
plt.title("TEST SET")
plt.show()
```



SUPPORT VECTOR MACHINE OS 2

In [290…

```
from sklearn.svm import SVC
classifier = SVC()
parameters = {"kernel":['linear','sigmoid','rbf'],
              "C":[0.001,0.01,0.3],
              "degree":[2,3,4],
              "gamma": [1]}

SV_f1_train,SV_f1_test=hyperp_search(classifier,parameters)
print(SV_f1_train-SV_f1_test)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
f1_train: 0.774358 using {'C': 0.01, 'degree': 2, 'gamma': 1, 'kernel': 'linear'}
```

```
f1         train 0.798   test 0.629


[[342 156]
 [ 65 187]]
0.16985789080029923
```

NEURAL NETWORKS - OS 2

In [291…

```
from sklearn.neural_network import MLPClassifier

classifier = MLPClassifier()
parameters = {"hidden_layer_sizes":[(12,6)],
              "max_iter": [2000],
              "alpha":    [0.1],
              "activation":['logistic', 'relu'],
              "learning_rate":['invscaling']}

NN_f1_train,NN_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
f1_train: 0.726281 using {'activation': 'logistic', 'alpha': 0.1, 'hidden_layer_sizes':
(12, 6), 'learning_rate': 'invscaling', 'max_iter': 2000}


f1         train 0.828   test 0.605


[[409  89]
 [104 148]]
```

RANDOM FOREST CLASSIFIER - OS 2

In [292…

```
from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier()
parameters = {'criterion' :        ['entropy', 'gini'],
              'max_depth' :        np.arange(3,5,1),
              'min_samples_leaf' : np.arange(5,30,5),
              'min_samples_split': np.arange(10,50,10),
              'n_estimators' :     np.arange(2,10,1),
             };

RF_f1_train,RF_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 640 candidates, totalling 1920 fits
f1_train: 0.769472 using {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 10,
'min_samples_split': 20, 'n_estimators': 9}


f1         train 0.782   test 0.632


[[289 209]
 [ 39 213]]
```

ADABOOST OS 2

In [293…

```
from sklearn.ensemble import AdaBoostClassifier


classifier= AdaBoostClassifier()
parameters = {'n_estimators'  : [5000],
              'learning_rate' : [0.01]}

ADAB_f1_train,ADAB_f1_test=hyperp_search(classifier,parameters)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
f1_train: 0.755811 using {'learning_rate': 0.01, 'n_estimators': 5000}


f1         train 0.802   test 0.618


[[361 137]
 [ 78 174]]
```

Since that the reults does not change so much we take in to consideration the oversampling method showed during the course.

Considerations

The following tabs show the F1 scores of all the algorithms we used. T he first one refers to the analysis with no oversampled data, while the second one refers to the analysis conducted whit an oversampled train set.

In [294…

```
data_no_oversampling
```

Out[294…

	Name	f1_train	f1_test
0	knn	0.585204	0.526923
1	tree	0.785714	0.569079
2	Logistic	0.576985	0.622776
3	SVM	0.547969	0.593449
4	Neural Network	0.740192	0.574359
5	Random Forest	0.720131	0.587332
6	ADABOOST	0.619718	0.588441

In [295…

```
data_oversampling
```

Out[295…

	Name	f1_train	f1_test
0	knn	0.892236	0.566038
1	tree	0.764651	0.609125
2	Logistic	0.731930	0.641873

	Name	f1_train	f1_test
3	SVM	0.746114	0.643148
4	Neural Network	0.852691	0.579439
5	Random Forest	0.755203	0.643045
6	ADABOOST	0.773333	0.645251

SAVING THE MODEL

Finally, we chose the Multi-layer Perceptron classifier. This model shows the highest F1 score after the analysis with no oversampled train test. We decided to take in to consideration the non oversampled train test because we saw that the F1 score performances of the models after the oversampling showed an overfitting tendency.

So we trained the model with the original features ande we saved it.

In [296…

```
import pickle

#The choosen model
from sklearn.neural_network import MLPClassifier
model_mlp=MLPClassifier(activation ='relu',
                    alpha= 1,
                    hidden_layer_sizes= (9, 6),
                    learning_rate= 'invscaling',
                    max_iter= 3000);
#save the model
pickle.dump(model_mlp, open('MLP_model.pkl', 'wb'))

#save the scaler
loaded_scaler = pickle.load(open('scaler_NOT_OVERSAMPLED.pkl', 'rb'))
```

Train the model

In [297…

```
X_tot_train=pd.concat([num,dummies], axis = 1)
Y_tot_train=df['failure']

selection_categorical=['tyre_season','tread_depth','wiring_strength','tyre_quality','tr
selection_numerical=['vulc', 'perc_nat_rubber','weather','perc_imp','temperature','elev

X_tot_train[selection_numerical] = pd.DataFrame(loaded_scaler.transform(X_tot_train[sel

loaded_model = pickle.load(open('MLP_model.pkl', 'rb'))
loaded_model.fit(X_tot_train,Y_tot_train)
```

Out[297…

```
MLPClassifier(alpha=1, hidden_layer_sizes=(9, 6), learning_rate='invscaling',
              max_iter=3000)
```

TEST THE DATA !

Now let's apply the same transformation we arleady applied to the train and test set.

In [298…

```
TEST = pd.read_csv('tyres_test.csv')

#drop diameter
TEST.drop(columns=['diameter'])

#Categorical
TEST[selection_categorical] = TEST[selection_categorical].astype(str)
#dummies
dummies2 = pd.get_dummies(TEST[selection_categorical])

#Numerical
TEST[selection_numerical] = pd.DataFrame(loaded_scaler.transform(TEST[selection_numeric

#Features
X2=pd.concat([TEST[selection_numerical],dummies2], axis = 1)
```

In [299…

```
y_SVM_predictions = loaded_model.predict(X2)
```

In [300…

```
import json

with open('predictions.txt', 'w') as filehandle:
    json.dump(y_SVM_predictions.tolist(), filehandle)
```

Feature importance

One of the drawbacks to chose the MLP model is the interpretability.

Trying to bypass this problem, to describe the features impertances we used:

- bar plot of the distributions of the numerical and categorical variables
- Gini and Entropy indexes

The principle we followed was that the more the target distributions were different the more the respective attributes would be important

- Permutation importance

Permutation feature importance is a model inspection technique that can be used for any fitted estimator when the data is tabular. This is especially useful for non-linear or opaque estimators. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled

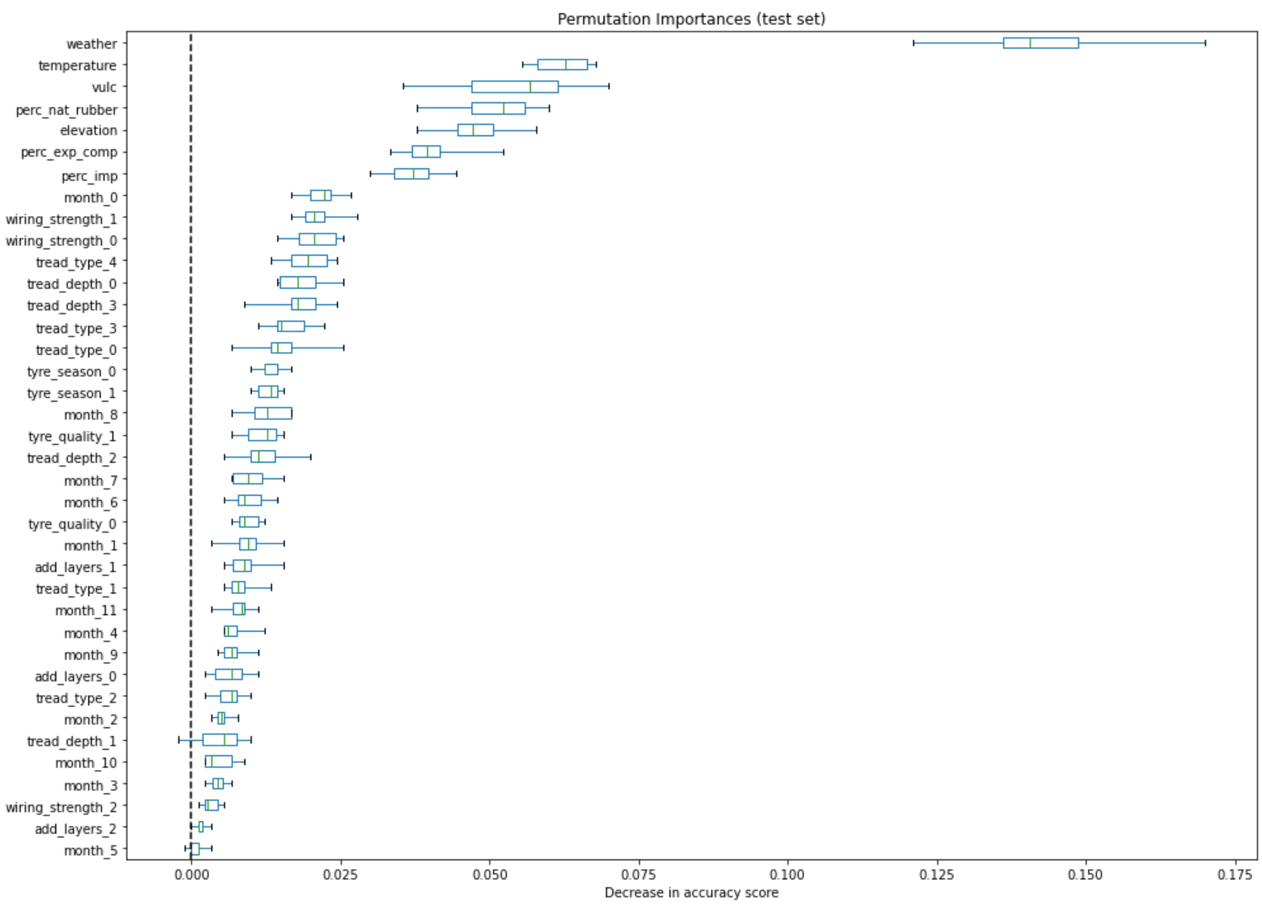
In [310…

```
#let's re divide the same train and test data to inspect the features importances

X=pd.concat([num,dummies], axis = 1)
y=df['failure']
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size =0.30,
                                                    stratify=y,        #preserve target
                                                    random_state= 321) #fix random seed
```

```
X_train[num.columns] = scaler.transform(X_train[num.columns])
X_test[num.columns] = scaler.transform(X_test[num.columns])


from sklearn.inspection import permutation_importance

loaded_model.fit(X_test,y_test)
result = permutation_importance(loaded_model,
                                X_test,
                                y_test,
                                n_repeats=10,
                                random_state=42,
                                n_jobs=2)
```

In [311…

```
sorted_importances_idx = result.importances_mean.argsort()
importances = pd.DataFrame(result.importances[sorted_importances_idx].T,
                           columns=X.columns[sorted_importances_idx],
)



ax = importances.plot.box(vert=False, whis=20,figsize=(14,10))
ax.set_title("Permutation Importances (test set)")
ax.axvline(x=0, color="k", linestyle="--")
ax.set_xlabel("Decrease in accuracy score")
ax.figure.tight_layout()
```



According with the visual inspection of the bar plots and with thu upper box plots we concludes that the most important features for this classification are:

- Weather
- Temperature

So the atmosferic conditions. And:

- per_nat_rubber
- vulc
- perc_exp_comp

So some components of the structure of the tyre