Machine Learning Project 22-23

Reza Gonabadi
Zeinab Mohammadpour

- First of all, We should import most needed library and package :

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
# During the project we will add some necessary library
```

- Our dataset is in the CSV format and we use pandas function like read_csv to import data to our environment

In [2]:

```
Data = pd.read_csv(r'C:\Reza Gonabadi\Python Code\Polimi Project\assignment_part1\tyres_tra
data = pd.DataFrame(Data)
```

In [3]:

```
#Importing The Test Set
tyres_test = pd.read_csv(r'C:\Reza Gonabadi\Python Code\Polimi Project\tyres_test.csv')
tyres_test = pd.DataFrame(tyres_test)
print("the test set size is : ", tyres_test.shape)
```

```
the test set size is :  (7984, 15)
```

- The basic things we should do are knowing about size of our data

In [4]:

```
data.shape
```

Out[4]:

```
(3000, 16)
```

- As you can see we have 3000 rows which shows number of sample and 16 columns which shows features and labels of classes; the next stepshould describe our data, what are our features and labels names and how they present in the dataset*

Exploratory Data Analysis

In [5]:

```
#gives information about the data types,columns, null value counts, memory usage etc
data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   vulc             3000 non-null   float64
 1   perc_nat_rubber  3000 non-null   int64  
 2   wiring_strength  3000 non-null   int64  
 3   weather          3000 non-null   float64
 4   perc_imp         3000 non-null   float64
 5   temperature      3000 non-null   float64
 6   tread_type       3000 non-null   int64  
 7   tyre_season      3000 non-null   int64  
 8   elevation        3000 non-null   float64
 9   month            3000 non-null   int64  
 10  tread_depth      3000 non-null   int64  
 11  tyre_quality     3000 non-null   int64  
 12  perc_exp_comp    3000 non-null   float64
 13  diameter         890 non-null    float64
 14  add_layers       3000 non-null   int64  
 15  failure          3000 non-null   int64  
dtypes: float64(7), int64(9)
memory usage: 375.1 KB
```

In [6]:

```
data.isna().sum()
```

Out[6]:

```
vulc                  0
perc_nat_rubber       0
wiring_strength       0
weather               0
perc_imp              0
temperature           0
tread_type            0
tyre_season           0
elevation             0
month                 0
tread_depth           0
tyre_quality          0
perc_exp_comp         0
diameter           2110
add_layers            0
failure               0
dtype: int64
```

In [7]:

```
#basic statistic details about the data
data.describe(include="all")
```

Out[7]:

	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000
mean	18.184712	31.249667	0.631333	0.282987	0.014550	-2.375360
std	1.587193	4.933300	0.546673	0.183252	0.014262	5.672184
min	12.312000	18.000000	0.000000	0.030000	0.000000	-19.280000
25%	17.241500	28.000000	0.000000	0.160000	0.010000	-6.960000
50%	17.834000	31.000000	1.000000	0.210000	0.010000	-2.080000
75%	18.934000	35.000000	1.000000	0.370000	0.020000	0.080000
max	29.932000	46.000000	2.000000	0.930000	0.050000	37.000000

- those two function (isna().sum() , info) show us number of null features in the dataset and the last function present dataset in rows and columns, so because the number of diameter feature has too much null number it would be better to remove this feature from our dataset and work with others*

- In the section below we remove column 'diameter' from our dataset*

In [8]:

```
data.drop(['diameter'], axis = 1, inplace=True)
tyres_test.drop(['diameter'], axis = 1, inplace=True)
```

In [9]:

```
# As you can see, the diameter feature was removed
data.head(10)
```

Out[9]:

	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature	tread_type	tyre_s
0	17.990	26	1	0.16	0.01	-8.12	0	
1	20.704	36	1	0.30	0.01	-4.52	2	
2	19.156	34	1	0.30	0.01	-1.08	0	
3	16.802	35	1	0.19	0.02	7.44	1	
4	17.140	23	2	0.39	0.01	30.52	0	
5	20.042	38	0	0.04	0.01	-0.20	2	
6	21.172	33	1	0.39	0.01	-2.28	0	
7	16.706	32	0	0.62	0.05	-3.96	3	
8	17.616	25	1	0.16	0.01	-6.88	0	
9	17.370	34	0	0.27	0.01	-1.28	2	

In [10]:

```
tyres_test.head(10)
```

Out[10]:

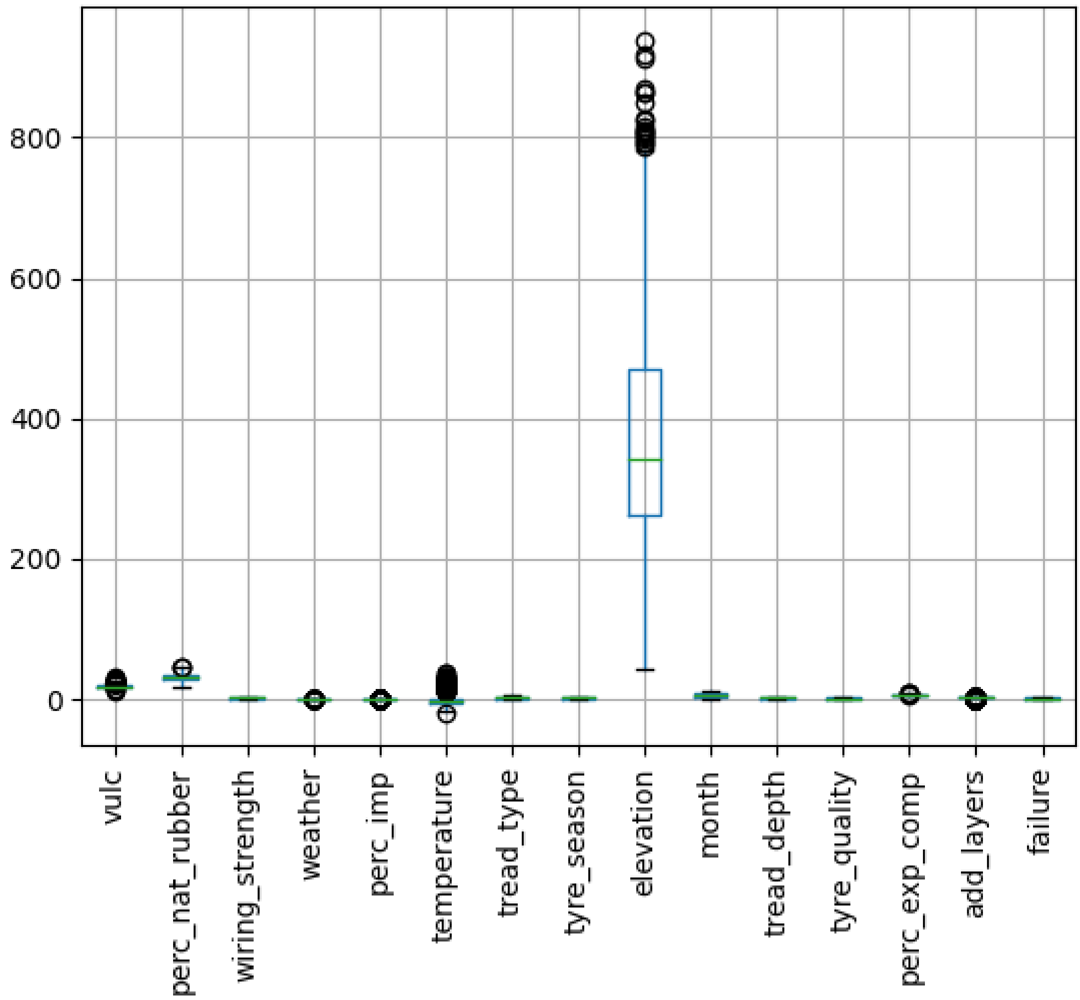
	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature	tread_type	tyre_s
0	17.180	30	1	0.21	0.00	-9.24	0	
1	17.744	24	1	0.16	0.01	-9.12	0	
2	16.930	34	0	0.27	0.01	3.64	2	
3	22.428	34	1	0.03	0.00	0.56	3	
4	16.818	29	1	0.06	0.00	-0.96	3	
5	17.284	27	1	0.16	0.01	-11.76	4	
6	20.050	32	1	0.30	0.01	-4.24	0	
7	17.932	24	1	0.16	0.01	-7.48	0	
8	16.486	33	1	0.62	0.05	-1.84	2	
9	17.690	25	1	0.16	0.01	-8.56	0	

In [11]:

```
%matplotlib inline
data.boxplot(rot=90)
```

Out[11]:

<AxesSubplot: >

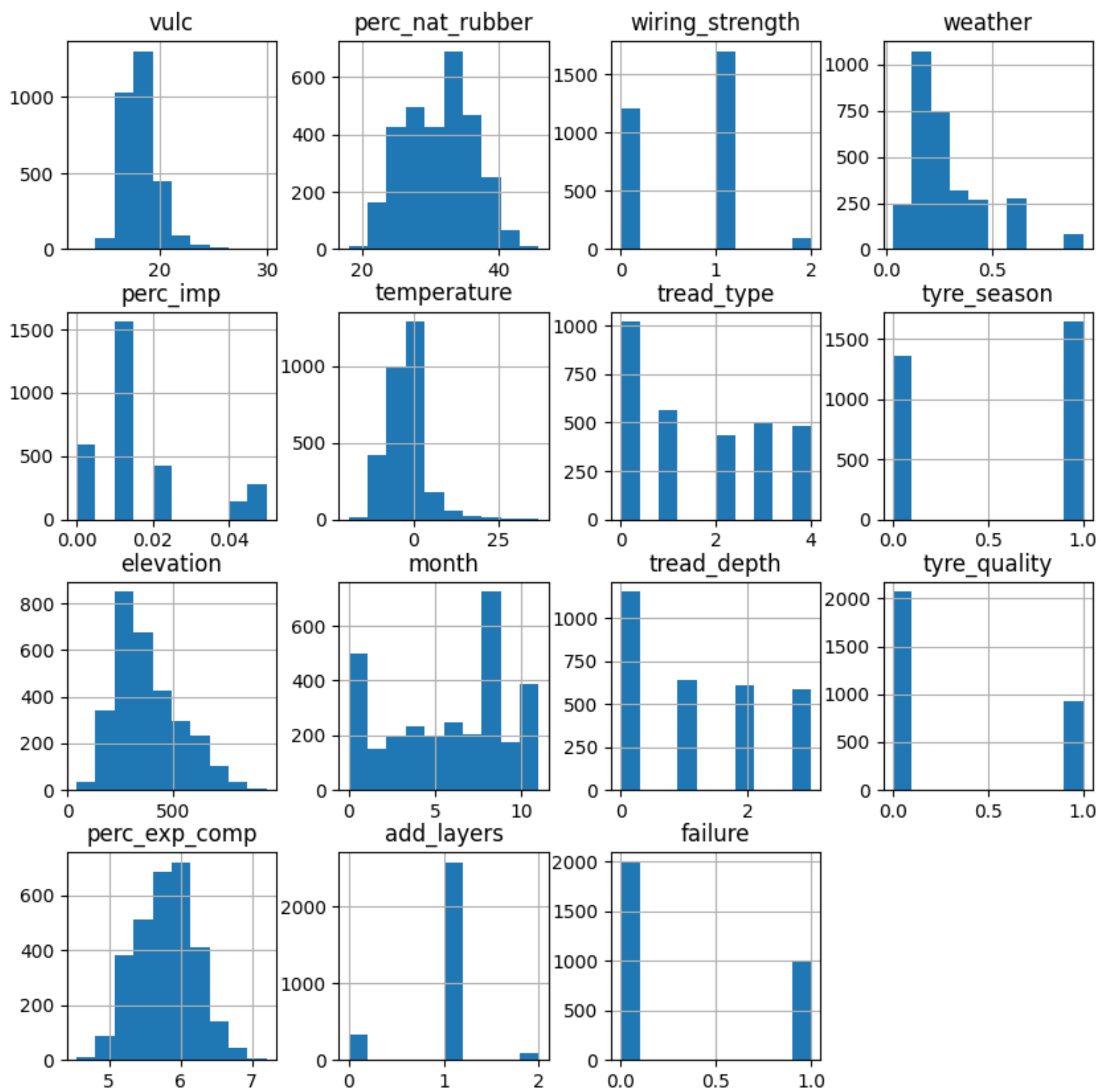


In [12]:

```
data.hist(figsize=(10,10))
```

Out[12]:

```
array([[<AxesSubplot: title={'center': 'vulc'}>,
        <AxesSubplot: title={'center': 'perc_nat_rubber'}>,
        <AxesSubplot: title={'center': 'wiring_strength'}>,
        <AxesSubplot: title={'center': 'weather'}>],
       [<AxesSubplot: title={'center': 'perc_imp'}>,
        <AxesSubplot: title={'center': 'temperature'}>,
        <AxesSubplot: title={'center': 'tread_type'}>,
        <AxesSubplot: title={'center': 'tyre_season'}>],
       [<AxesSubplot: title={'center': 'elevation'}>,
        <AxesSubplot: title={'center': 'month'}>,
        <AxesSubplot: title={'center': 'tread_depth'}>,
        <AxesSubplot: title={'center': 'tyre_quality'}>],
       [<AxesSubplot: title={'center': 'perc_exp_comp'}>,
        <AxesSubplot: title={'center': 'add_layers'}>,
        <AxesSubplot: title={'center': 'failure'}>, <AxesSubplot: >]],
      dtype=object)
```



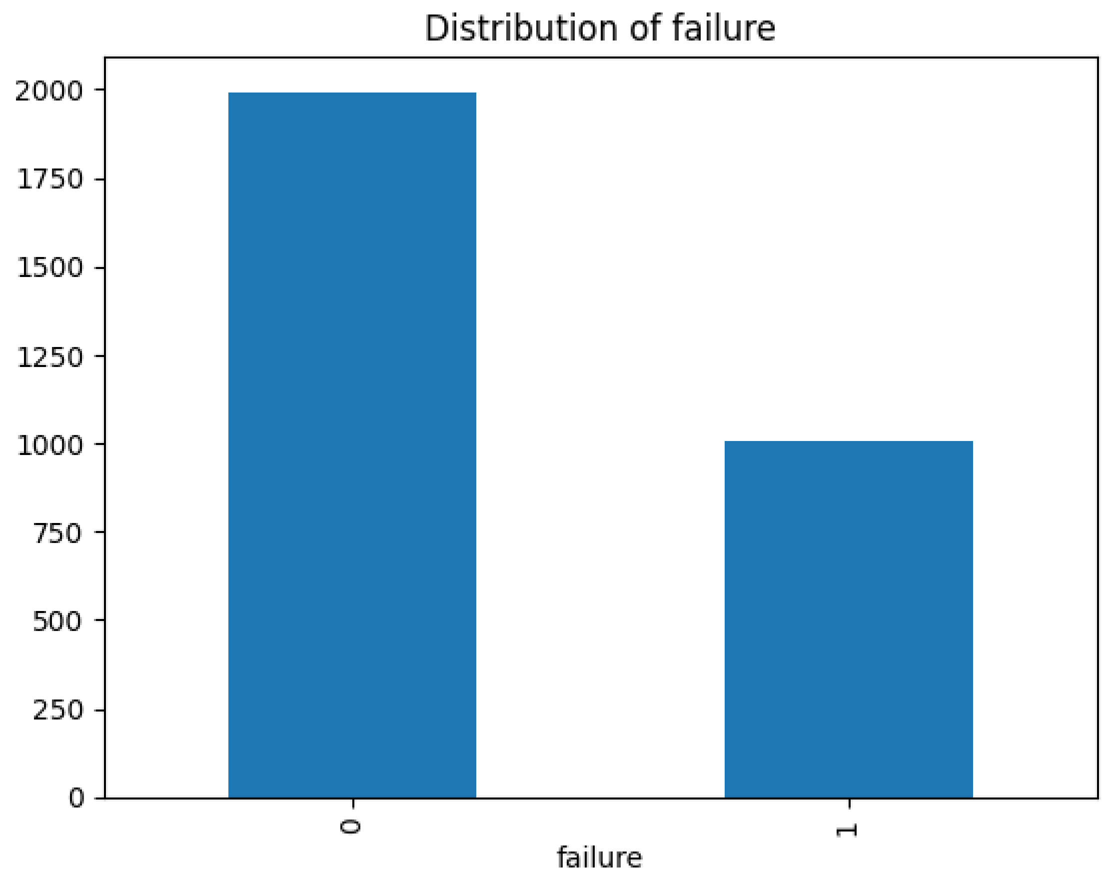
In [13]:

```
#Print class freq. through pandas: we group the data by the column target and we count the
target_dist=data.groupby('failure').size()
print(target_dist)

%matplotlib inline

#Visualize Class Counts
target_dist.plot.bar(x='',y='',title='Distribution of failure')
```

```
failure
0    1992
1    1008
dtype: int64
```

Out[13]:

```
<AxesSubplot: title={'center': 'Distribution of failure'}, xlabel='failure'>
```



- the Distribution of failure above shows us that our cataset classes are noe well-balance so we will oversample the data in the next steps

Data Preparation

- We want to remove outliers in dataset using standard deviations technique in this section we remove every sample which its feature greater than 3 in STD

In [14]:

```
def remove_outliers(df,columns,n_std):
    for col in columns:
        print('Working on column: {}'.format(col))

        mean = df[col].mean()
        sd = df[col].std()

        df = df[(df[col] <= mean+(n_std*sd))]

    return df

data = remove_outliers(data,['vulc','perc_nat_rubber','weather','temperature','perc_imp','w
# data.failure.value_counts()
```

```
Working on column: vulc
Working on column: perc_nat_rubber
Working on column: weather
Working on column: temperature
Working on column: perc_imp
Working on column: wiring_strength
Working on column: elevation
Working on column: perc_exp_comp
```

- As you see before our dataset was not well balanced in classes, so it would be better to oversample it first and then using it in our machines

In [15]:

```
from sklearn.utils import resample

#Over-sample Minority Class
#1) Separate majority and minority classes
df_majority = data[data.failure==0] #"target" is the name of the target column, change it a
df_minority = data[data.failure==1] #"target" is the name of the target column, change it a

#2) Oversample minority class
df_minority_oversampled = resample(df_minority,
                                   replace=True,
                                   n_samples=1900,     # number of samples into the minorit
                                   random_state=123)  # reproducible results

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

#4) Display new class counts
df_oversampled.failure.value_counts() #"target" is the name of the target column, change it

data = df_oversampled.copy()
data
```

Out[15]:

	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature	tread_type	tyı
1677	16.384	33	0	0.37	0.02	-0.48	0	
1192	17.628	26	0	0.62	0.05	9.12	1	
1248	19.524	24	1	0.38	0.04	6.88	2	
1045	16.968	33	0	0.37	0.02	-0.12	3	
291	19.884	34	1	0.38	0.04	-1.16	3	
...	...	...	...	...	...	...	...	
2993	17.860	21	1	0.16	0.01	-6.48	0	
2994	19.298	29	1	0.03	0.00	-1.00	2	
2997	16.170	33	1	0.39	0.01	-3.44	1	
2998	18.872	37	0	0.03	0.00	-0.76	4	
2999	20.272	33	2	0.06	0.00	2.80	1	

3834 rows × 15 columns

- We seperte X , y from dataset X means our features and y means our labels

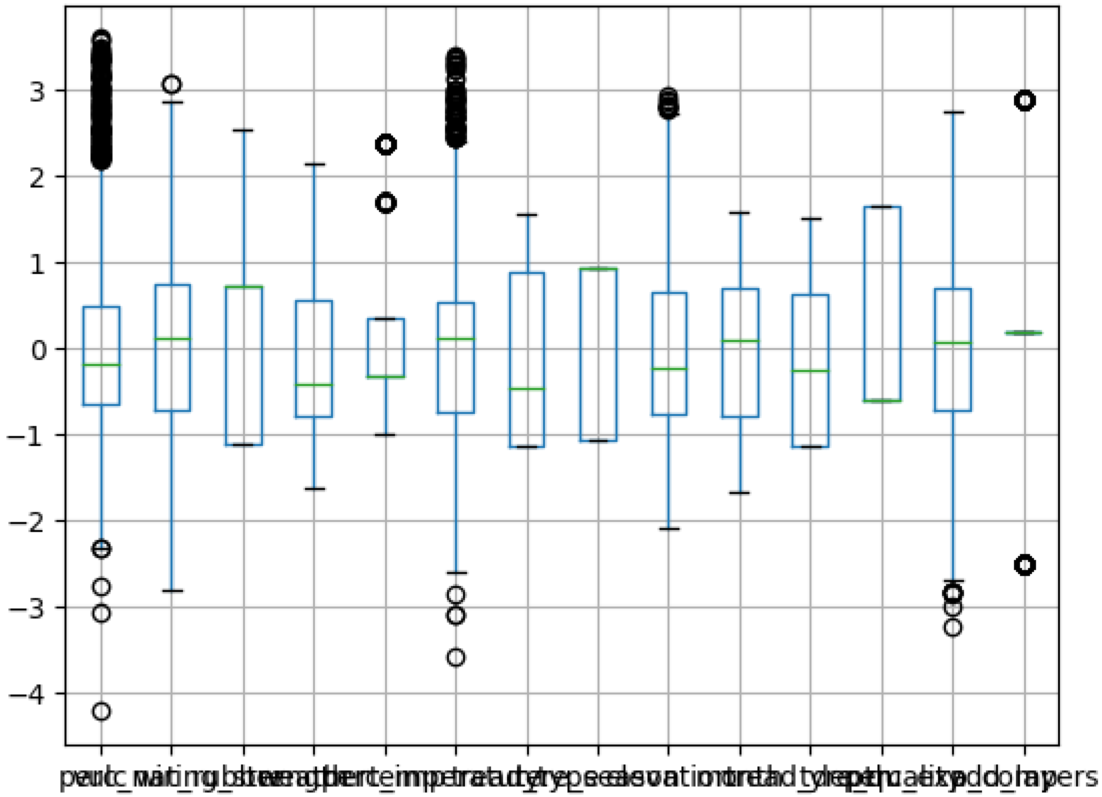
In [16]:

```
X = data.drop(['failure'],axis=1)

y = data.failure.values
```

- One on the important thing before classification and every machine learning problem is Normalizing data, so we normalize the data in the next step

In [17]:

```
# Scale data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)

scaler_tyres = StandardScaler().fit(tyres_test)

# We compute the scaler
scaled_data = scaler.transform(X.astype(float))
scaled_X = pd.DataFrame(scaled_data.astype(float))
scaled_X.columns = X.columns

scaled_tyres_test_data = scaler_tyres.transform(tyres_test.astype(float))
scaled_tyres_test = pd.DataFrame(scaled_tyres_test_data.astype(float))
scaled_tyres_test.columns = X.columns

scaled_X.boxplot()
# scaled_tyres_test.boxplot()

X = scaled_X.copy()
tyres_test = scaled_tyres_test.copy()
```



In [18]:

```
tyres_test.shape
```

Out[18]:

```
(7984, 14)
```

In [19]:

```
X
```

Out[19]:

	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature	tread_type
0	-1.220108	0.338219	-1.114479	0.549167	0.349774	0.375608	-1.144667
1	-0.306383	-1.129147	-1.114479	2.140485	2.374484	2.398950	-0.470941
2	1.086240	-1.548395	0.715462	0.612820	1.699581	1.926837	0.202786
3	-0.791156	0.338219	-1.114479	0.549167	0.349774	0.451483	0.876512
4	1.350662	0.547843	0.715462	0.612820	1.699581	0.232288	0.876512
...	...	...	...	...	...	...	...
3829	-0.135977	-2.177266	0.715462	-0.787541	-0.325130	-0.888981	-1.144667
3830	0.920242	-0.500276	0.715462	-1.615026	-1.000033	0.266010	0.202786
3831	-1.377292	0.338219	0.715462	0.676472	-0.325130	-0.248256	-0.470941
3832	0.607342	1.176714	-1.114479	-1.615026	-1.000033	0.316594	1.550239
3833	1.635650	0.338219	2.545403	-1.424068	-1.000033	1.066916	-0.470941

3834 rows × 14 columns

- As you can see above our dataset is normalize now and we can use it in our machines

- Now we seperate The data into Train and Test which we could use in our machine

In [20]:

```
#SPLIT DATA INTO TRAIN AND TEST SET
X_train, X_test, y_train, y_test = train_test_split(X, y,  #X_scaled
                                                    test_size =0.30, #by default is 75%-25%
                                                    #shuffle is set True by default,
                                                    stratify=y,
                                                    random_state= 123) #fix random seed for

print(X_train.shape)
```

```
(2683, 14)
```

SVM

- We tried lots of machines and after all we decided to use SVM because it is more reliable and has more score from other ones

- We use Grid Sreach for finding best parameters in SVM inputs and after finding best parameters we use them in our machines

In [21]:

```
#DEFINE YOUR CLASSIFIER and THE PARAMETERS GRID
from sklearn.svm import SVC

classifier = SVC()
parameters = {"kernel":['linear','rbf','polinomial'], "C":[0.1,1,100],"gamma":[1], "degree"
```

In [22]:

```
#DEFINE YOUR GRIDSEARCH
'''
GS perfoms an exhaustive search over specified parameter values for an estimator.
GS uses a Stratified K-Folds cross-validator
(The folds are made by preserving the percentage of samples for each class.)
If refit=True the model is retrained on the whole training set with the best found params
'''
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(classifier, parameters, cv=3, scoring = 'f1', verbose=50, n_jobs=-1, refi
```

In [23]:

```
#TRAIN YOUR CLASSIFIER
gs = gs.fit(X_train, y_train)
```

```
sklearn.utils._param_validation.InvalidParameterError: The 'kernel' parame
ter of SVC must be a str among {'sigmoid', 'linear', 'precomputed', 'rbf',
'poly'} or a callable. Got 'polinomial' instead.

--------------------------------------------------------------------------
------
1 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Reza.Gonabadi\AppData\Local\Programs\Python\Python311\Lib
\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_
and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Reza.Gonabadi\AppData\Local\Programs\Python\Python311\Lib
\site-packages\sklearn\svm\_base.py", line 180, in fit
    self._validate_params()
  File "C:\Users\Reza.Gonabadi\AppData\Local\Programs\Python\Python311\Lib
\site-packages\sklearn\base.py", line 570, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\Reza.Gonabadi\AppData\Local\Programs\Python\Python311\Lib
```

In [24]:

```
#summarize the results of your GRIDSEARCH
print('***GRIDSEARCH RESULTS***')

print("Best score: %f using %s" % (gs.best_score_, gs.best_params_))
means = gs.cv_results_['mean_test_score']
stds = gs.cv_results_['std_test_score']
params = gs.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
***GRIDSEARCH RESULTS***
Best score: 0.777330 using {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'rb
f'}
0.723492 (0.006207) with: {'C': 0.1, 'degree': 2, 'gamma': 1, 'kernel': 'lin
ear'}
0.723221 (0.000870) with: {'C': 0.1, 'degree': 2, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 0.1, 'degree': 2, 'gamma': 1, 'kernel': 'polinomial'}
0.723492 (0.006207) with: {'C': 0.1, 'degree': 3, 'gamma': 1, 'kernel': 'lin
ear'}
0.723221 (0.000870) with: {'C': 0.1, 'degree': 3, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 0.1, 'degree': 3, 'gamma': 1, 'kernel': 'polinomial'}
0.723492 (0.006207) with: {'C': 0.1, 'degree': 4, 'gamma': 1, 'kernel': 'lin
ear'}
0.723221 (0.000870) with: {'C': 0.1, 'degree': 4, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 0.1, 'degree': 4, 'gamma': 1, 'kernel': 'polinomial'}
0.725227 (0.004027) with: {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'linea
r'}
0.774696 (0.015731) with: {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'rbf'}
nan (nan) with: {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'polinomial'}
0.725227 (0.004027) with: {'C': 1, 'degree': 3, 'gamma': 1, 'kernel': 'linea
r'}
0.774696 (0.015731) with: {'C': 1, 'degree': 3, 'gamma': 1, 'kernel': 'rbf'}
nan (nan) with: {'C': 1, 'degree': 3, 'gamma': 1, 'kernel': 'polinomial'}
0.725227 (0.004027) with: {'C': 1, 'degree': 4, 'gamma': 1, 'kernel': 'linea
r'}
0.774696 (0.015731) with: {'C': 1, 'degree': 4, 'gamma': 1, 'kernel': 'rbf'}
nan (nan) with: {'C': 1, 'degree': 4, 'gamma': 1, 'kernel': 'polinomial'}
0.726991 (0.004164) with: {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'lin
ear'}
0.777330 (0.014885) with: {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'polinomial'}
0.726991 (0.004164) with: {'C': 100, 'degree': 3, 'gamma': 1, 'kernel': 'lin
ear'}
0.777330 (0.014885) with: {'C': 100, 'degree': 3, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 100, 'degree': 3, 'gamma': 1, 'kernel': 'polinomial'}
0.726991 (0.004164) with: {'C': 100, 'degree': 4, 'gamma': 1, 'kernel': 'lin
ear'}
0.777330 (0.014885) with: {'C': 100, 'degree': 4, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 100, 'degree': 4, 'gamma': 1, 'kernel': 'polinomial'}
```

In [25]:

```
#TEST ON YOUR TEST SET
best_model = gs.best_estimator_
y_pred = best_model.predict(X_test)

y_pred_train = best_model.predict(X_train)

y_tyres_test = best_model.predict(tyres_test)
```

In [26]:

```
#EVALUATE YOUR PREDICTION (on the y_test that you left aside)
from sklearn.metrics import f1_score


print('***RESULTS ON TRAIN SET***')
print("f1_score: ", f1_score(y_train, y_pred_train))
print("--")
print('***RESULTS ON TEST SET***')
print("f1_score: ", f1_score(y_test, y_pred))
```

```
***RESULTS ON TRAIN SET***
f1_score:  1.0
--
***RESULTS ON TEST SET***
f1_score:  0.8576923076923076
```

As you can see the f1 score in test set data is 85% and in train set is 100%

In [27]:

```
#PRINT SOME FURTHER METRICS
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.96      0.88       581
           1       0.95      0.78      0.86       570

    accuracy                           0.87      1151
   macro avg       0.88      0.87      0.87      1151
weighted avg       0.88      0.87      0.87      1151
```

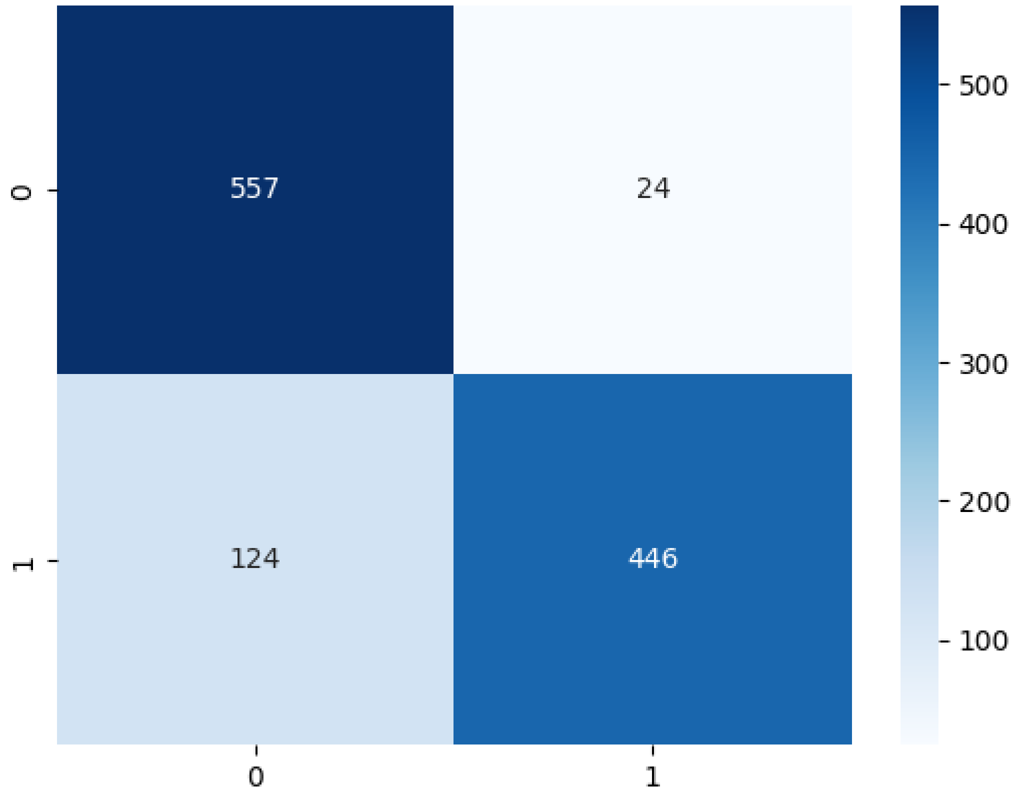
In [28]:

```
#CONFUSION MATRIX
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[557  24]
 [124 446]]
```

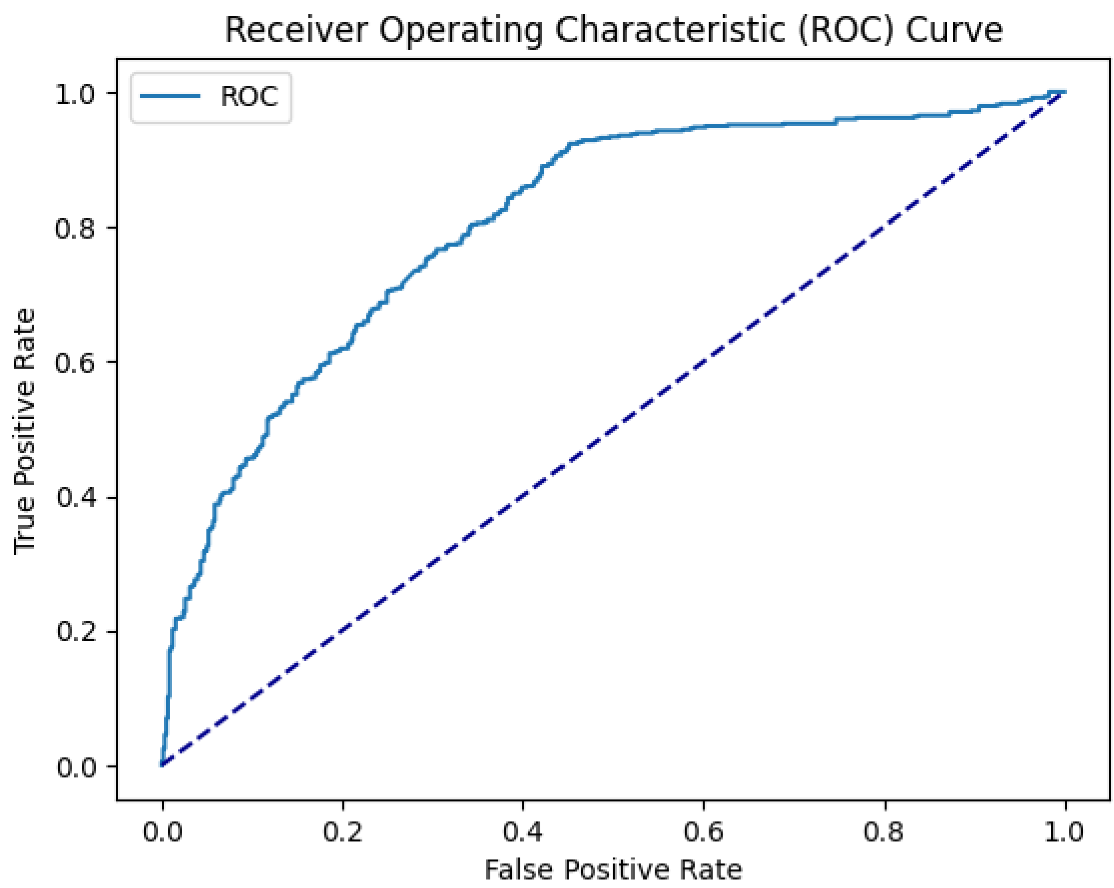
In [29]:

```
# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap="Blues"); #annot=Tr
```



In [30]:

```
from sklearn import metrics

model = SVC(C=0.1, gamma=0.0001, kernel='linear',probability=True)

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 the target

fpr, tpr, thresholds=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: %.2f' % auc)
```



```
AUC: 0.81
```

- We use all 14 features in our machine. It would be better if we apply feauter selection technique to reduce size of the features. In that way our machine would learn faster and better than using all 14 features.

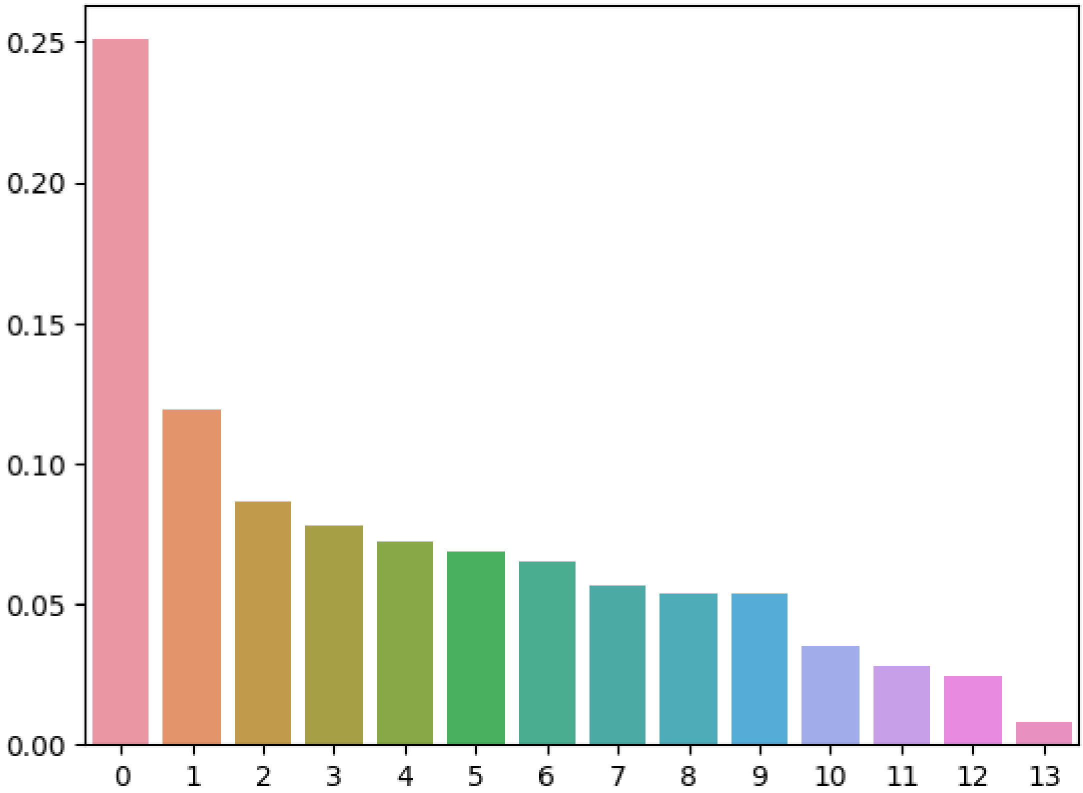
Apply PCA

In [31]:

```
#PCA fit
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(scaled_X)
df_pca = pd.DataFrame(pca.transform(scaled_X))

pca_tyres_test = PCA()
pca_tyres_test.fit(tyres_test)
df_pca_tyres_test = pd.DataFrame(pca_tyres_test.transform(tyres_test))
```

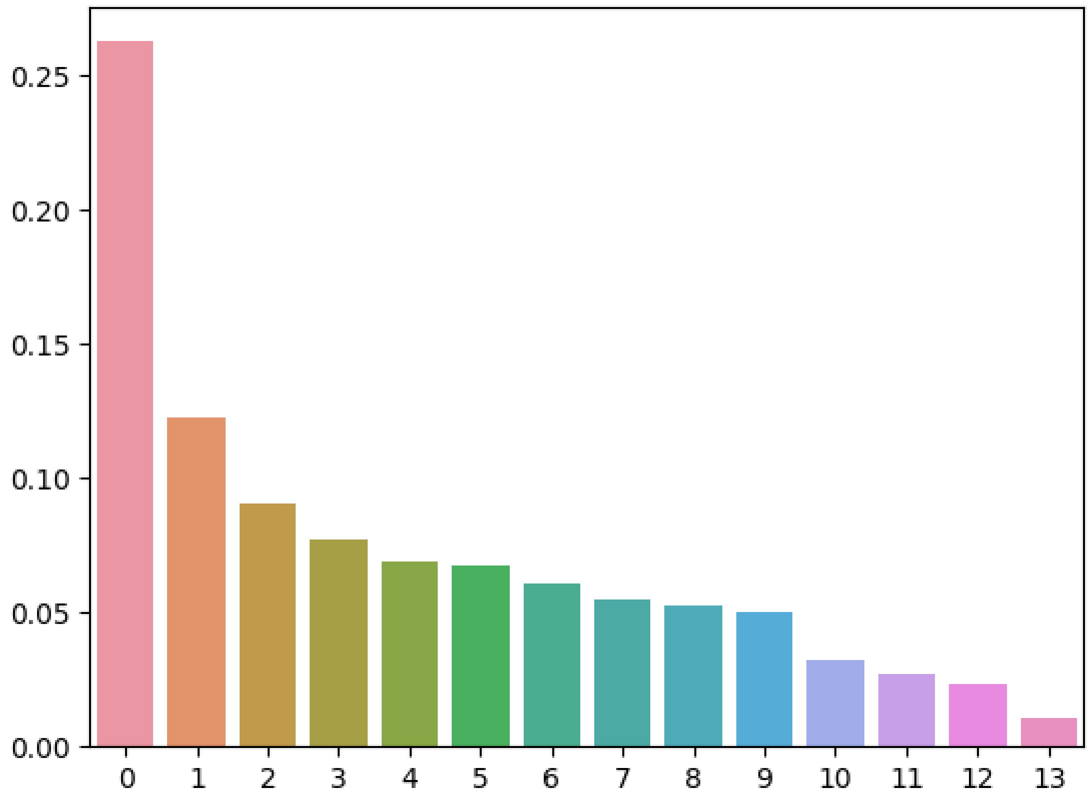
In [32]:

```
explained_variance=pd.DataFrame(pca.explained_variance_ratio_)
%matplotlib inline
import seaborn as sns
ax = sns.barplot( data=explained_variance.transpose())
```



In [33]:

```
explained_variance_tyres=pd.DataFrame(pca_tyres_test.explained_variance_ratio_)
%matplotlib inline
import seaborn as sns
ax = sns.barplot( data=explained_variance_tyres.transpose())
```



- Based on this plot I believe that the first 6 pca's are enough for choosing in our machine

In [34]:

```
pd.DataFrame(pca.components_,columns=X.columns)
```

Out[34]:

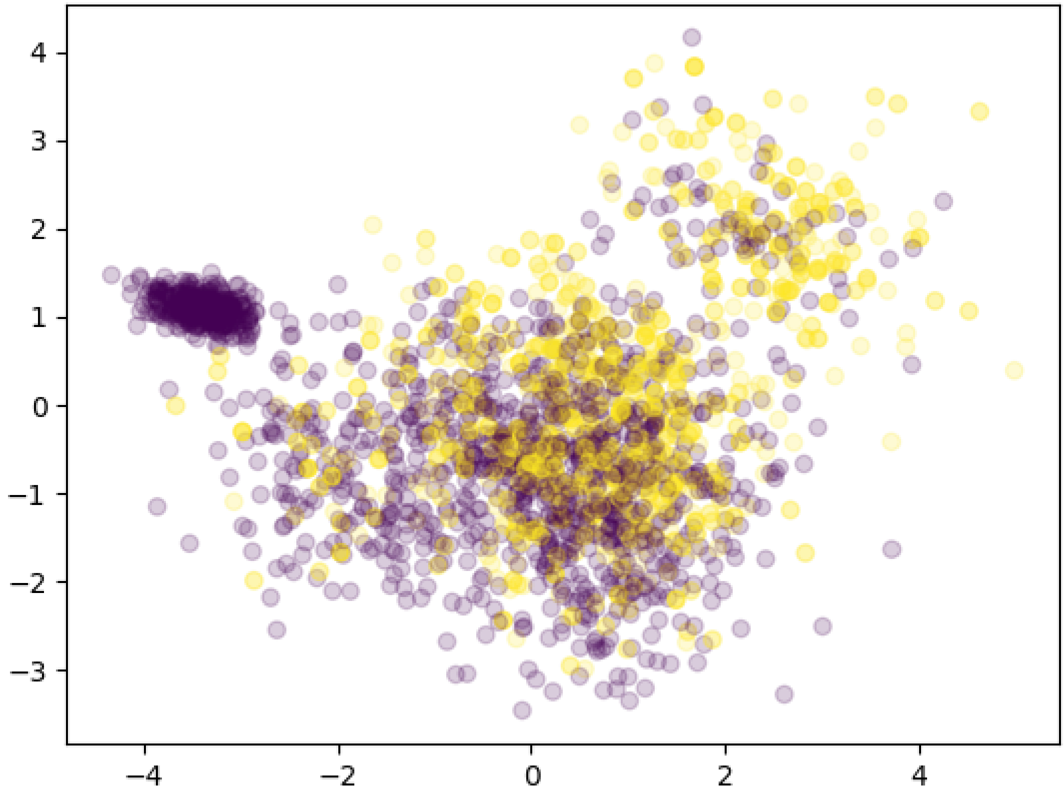
	vulc	perc_nat_rubber	wiring_strength	weather	perc_imp	temperature	tread_type
0	-0.068478	0.314061	-0.186414	0.355673	0.255935	0.400677	0.243082
1	-0.318820	-0.314747	0.212791	0.444360	0.515069	0.080814	-0.169973
2	0.603934	-0.074155	0.342172	0.259703	0.370231	-0.141793	0.054388
3	-0.031871	-0.316511	0.603494	-0.228791	-0.295907	0.374409	-0.080074
4	-0.196049	-0.026206	-0.209736	-0.055748	-0.012506	-0.055157	0.094289
5	-0.115457	0.142002	-0.221883	0.061045	0.021604	0.140607	-0.508125
6	0.067273	0.026721	0.063559	0.020762	0.016635	-0.019508	0.041356
7	0.227963	0.043501	0.076921	-0.032865	-0.038475	0.088803	-0.202644
8	0.198979	0.086795	0.021021	0.001516	-0.060888	0.026925	0.651920
9	0.497395	0.345737	-0.054672	0.023918	-0.056856	-0.018146	-0.405003
10	-0.294691	0.645104	0.550864	0.021962	-0.003508	0.040392	-0.003914
11	-0.050479	0.336125	0.008142	-0.235416	0.166322	0.178617	0.057171
12	0.207874	-0.127866	-0.168554	-0.039514	-0.003174	0.778998	0.018468
13	0.007559	-0.024653	-0.012077	-0.699904	0.638407	-0.011526	-0.011938

In [35]:

```
X_pca = pd.DataFrame(data = pca.transform(X)
             ,columns = ['pc1', 'pc2','pc3','pc4','pc5', 'pc6','pc7','pc8', 'pc9','pc10','p

X_train_pca = pd.DataFrame(data = pca.transform(X_train)
             ,columns = ['pc1', 'pc2','pc3','pc4','pc5', 'pc6','pc7','pc8', 'pc9','pc10','p

X_test_pca = pd.DataFrame(data = pca.transform(X_test)
             ,columns = ['pc1', 'pc2','pc3','pc4','pc5', 'pc6','pc7','pc8', 'pc9','pc10','p

tyres_test_pca = pd.DataFrame(data = pca_tyres_test.transform(tyres_test)
             ,columns = ['pc1', 'pc2','pc3','pc4','pc5', 'pc6','pc7','pc8', 'pc9','pc10','p
```

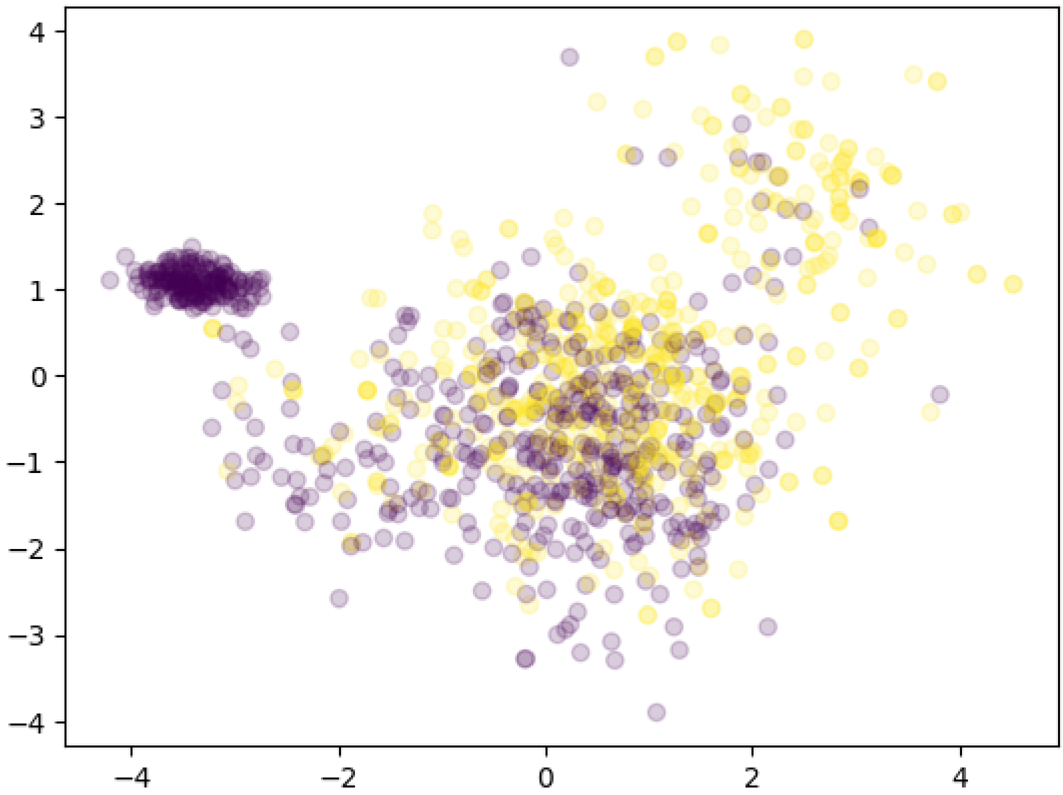
In [36]:

```
import matplotlib.pyplot as plt

x = X_train_pca.iloc[:,0]
y = X_train_pca.iloc[:,1]

plt.scatter(x, y,alpha=0.2,c=y_train )
plt.show()
```



In [37]:

```
import matplotlib.pyplot as plt

x = X_test_pca.iloc[:,0]
y = X_test_pca.iloc[:,1]

plt.scatter(x, y,alpha=0.2,c=y_test )
plt.show()
```



Using 6 PC's for the Machines

In [38]:

```
#DEFINE YOUR CLASSIFIER and THE PARAMETERS GRID
from sklearn.svm import SVC

classifier = SVC()
parameters = {"kernel":['linear','rbf','polinomial'], "C":[0.1,1,100],"gamma":[1], "degree"

#DEFINE YOUR GRIDSEARCH
'''
GS perfoms an exhaustive search over specified parameter values for an estimator.
GS uses a Stratified K-Folds cross-validator
(The folds are made by preserving the percentage of samples for each class.)
If refit=True the model is retrained on the whole training set with the best found params
'''
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(classifier, parameters, cv=3, scoring = 'f1', verbose=50, n_jobs=-1, refi
```

In [39]:

```
#TRAIN YOUR CLASSIFIER
gs = gs.fit(X_train_pca.iloc[:,:6], y_train)
```

In [40]:

```
#summarize the results of your GRIDSEARCH
print('***GRIDSEARCH RESULTS***')

print("Best score: %f using %s" % (gs.best_score_, gs.best_params_))
means = gs.cv_results_['mean_test_score']
stds = gs.cv_results_['std_test_score']
params = gs.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
***GRIDSEARCH RESULTS***
Best score: 0.778840 using {'C': 100, 'degree': 2, 'gamma': 1, 'kernel':
'rbf'}
0.683551 (0.005481) with: {'C': 0.1, 'degree': 2, 'gamma': 1, 'kernel': 'l
inear'}
0.723810 (0.002364) with: {'C': 0.1, 'degree': 2, 'gamma': 1, 'kernel': 'r
bf'}
nan (nan) with: {'C': 0.1, 'degree': 2, 'gamma': 1, 'kernel': 'polinomia
l'}
0.683551 (0.005481) with: {'C': 0.1, 'degree': 3, 'gamma': 1, 'kernel': 'l
inear'}
0.723810 (0.002364) with: {'C': 0.1, 'degree': 3, 'gamma': 1, 'kernel': 'r
bf'}
nan (nan) with: {'C': 0.1, 'degree': 3, 'gamma': 1, 'kernel': 'polinomia
l'}
0.683551 (0.005481) with: {'C': 0.1, 'degree': 4, 'gamma': 1, 'kernel': 'l
inear'}
0.723810 (0.002364) with: {'C': 0.1, 'degree': 4, 'gamma': 1, 'kernel': 'r
bf'}
nan (nan) with: {'C': 0.1, 'degree': 4, 'gamma': 1, 'kernel': 'polinomia
l'}
0.685512 (0.006622) with: {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'lin
ear'}
0.778506 (0.017438) with: {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 1, 'degree': 2, 'gamma': 1, 'kernel': 'polinomial'}
0.685512 (0.006622) with: {'C': 1, 'degree': 3, 'gamma': 1, 'kernel': 'lin
ear'}
0.778506 (0.017438) with: {'C': 1, 'degree': 3, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 1, 'degree': 3, 'gamma': 1, 'kernel': 'polinomial'}
0.685512 (0.006622) with: {'C': 1, 'degree': 4, 'gamma': 1, 'kernel': 'lin
ear'}
0.778506 (0.017438) with: {'C': 1, 'degree': 4, 'gamma': 1, 'kernel': 'rb
f'}
nan (nan) with: {'C': 1, 'degree': 4, 'gamma': 1, 'kernel': 'polinomial'}
0.685024 (0.007205) with: {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'l
inear'}
0.778840 (0.012534) with: {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'r
bf'}
nan (nan) with: {'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'polinomia
l'}
0.685024 (0.007205) with: {'C': 100, 'degree': 3, 'gamma': 1, 'kernel': 'l
inear'}
0.778840 (0.012534) with: {'C': 100, 'degree': 3, 'gamma': 1, 'kernel': 'r
bf'}
nan (nan) with: {'C': 100, 'degree': 3, 'gamma': 1, 'kernel': 'polinomia
```

```
l'}
0.685024 (0.007205) with: {'C': 100, 'degree': 4, 'gamma': 1, 'kernel': 'l
inear'}
0.778840 (0.012534) with: {'C': 100, 'degree': 4, 'gamma': 1, 'kernel': 'r
bf'}
nan (nan) with: {'C': 100, 'degree': 4, 'gamma': 1, 'kernel': 'polinomia
l'}
```

In [48]:

```
#TEST ON YOUR TEST SET
best_model = gs.best_estimator_
y_pred = best_model.predict(X_test_pca.iloc[:,:6])

y_pred_train = best_model.predict(X_train_pca.iloc[:,:6])

y_tyres_test_result = best_model.predict(tyres_test_pca.iloc[:,:6])
```

In [49]:

```
#EVALUATE YOUR PREDICTION (on the y_test that you left aside)
from sklearn.metrics import f1_score


print('***RESULTS ON TRAIN SET***')
print("f1_score: ", f1_score(y_train, y_pred_train))
print("--")
print('***RESULTS ON TEST SET***')
print("f1_score: ", f1_score(y_test, y_pred))
```

```
***RESULTS ON TRAIN SET***
f1_score:  1.0
--
***RESULTS ON TEST SET***
f1_score:  0.8266666666666668
```

As you can see the f1 score in the test set data is 82.7% and in the train set is 100%, and I consider this as a final score of this project.

In [50]:

```
#PRINT SOME FURTHER METRICS
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.77      0.81       581
           1       0.79      0.87      0.83       570

    accuracy                           0.82      1151
   macro avg       0.82      0.82      0.82      1151
weighted avg       0.82      0.82      0.82      1151
```

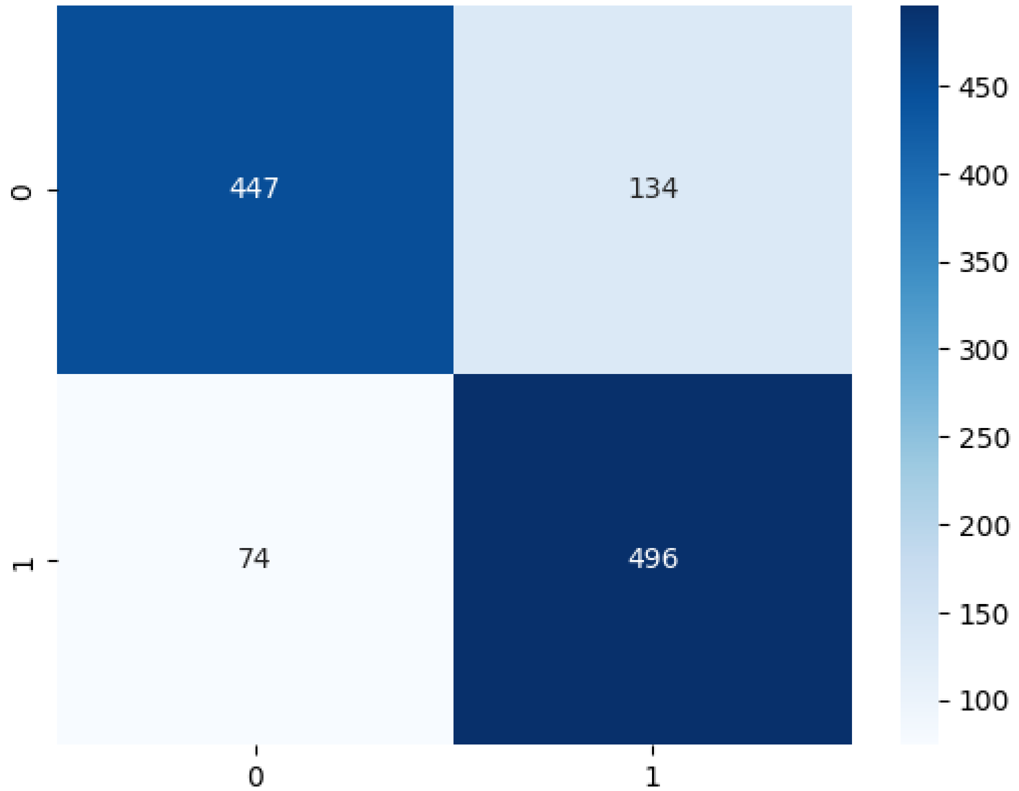
In [51]:

```
#CONFUSION MATRIX
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[447 134]
 [ 74 496]]
```

In [52]:

```
# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap="Blues"); #annot=Tr
```



In [57]:

```
count = (y_tyres_test_result == 1).sum()
count
```

Out[57]:

```
2529
```

In [59]:

```
arr = y_tyres_test_result
int_array = arr.astype('int')

np.savetxt("result.txt", int_array)
```

In [ ]: