..... By Mr Jecky Master

=> Importing all the libraries.

In [140]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost
import warnings
warnings.filterwarnings("ignore")ada_model = AdaBoostClassifier()
```

=> Introduction of APS Failure Scania Trucks dataset

=>The dataset consists of data collected from heavy Scania trucks in everyday usage. The system in focus is the **Air Pressure system (APS)** which generates pressurised air that are utilized in various functions in a truck, such as braking and gear changes. The datasets' positive class consists of component failures for a specific component of the APS system. The negative class consists of trucks with failures for components not related to the APS.

In [58]:
```
data = pd.read_csv('aps_failure_training_set.csv', header=None)
```

In [59]:
```
# data.head(20)
```

In [60]:
```
data = data.drop(data.index[0:20])
```

In [61]:
```
data.head(20)
```

29	neg	78696	na	0	na	0	0	0	0	0	...	0
30	pos	153204	0	182	na	0	0	0	0	0	...	
31	neg	39196	na	204	170	0	0	0	0	0	...	
32	neg	45912	na	0	454	0	0	0	0	0	...	4
33	neg	2104	na	36	26	0	0	0	0	9744	...	
34	neg	118950	na	1390	1298	0	0	0	0	0	...	1(
35	neg	24416	na	0	na	0	0	0	0	0	...	
36	neg	14	0	62	34	0	0	0	0	0	...	
37	neg	31300	0	784	740	0	0	0	0	0	...	
38	neg	736	2	24	22	16	20	0	0	0	...	
39	neg	332	na	2130706432	20	0	0	0	0	0	...	

20 rows × 171 columns

In [62]:
```
data.columns=data.iloc[0]
```

In [63]:
```
data.head()
```

Out[63]:

20	**class**	**aa_000**	**ab_000**	**ac_000**	**ad_000**	**ae_000**	**af_000**	**ag_000**	**ag_001**	**ag_002**	**...**	**ee**
20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee
21	neg	76698	na	2130706438	280	0	0	0	0	0	...	124(
22	neg	33058	na	0	na	0	0	0	0	0	...	42
23	neg	41040	na	228	100	0	0	0	0	0	...	27
24	neg	12	0	70	66	0	10	0	0	0	...	

5 rows × 171 columns

In [64]:
```
data = data.drop(data.index[0])
```

In [65]:
```
data.head()
```

Out[65]:

20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee
21	neg	76698	na	2130706438	280	0	0	0	0	0	...	124
22	neg	33058	na	0	na	0	0	0	0	0	...	42
23	neg	41040	na	228	100	0	0	0	0	0	...	27
24	neg	12	0	70	66	0	10	0	0	0	...	
25	neg	60874	na	1368	458	0	0	0	0	0	...	62

5 rows × 171 columns

In [66]:
```
data.drop_duplicates()
```

Out[66]:

20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...
21	neg	76698	na	2130706438	280	0	0	0	0	0	...
22	neg	33058	na	0	na	0	0	0	0	0	...
23	neg	41040	na	228	100	0	0	0	0	0	...
24	neg	12	0	70	66	0	10	0	0	0	...
25	neg	60874	na	1368	458	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
60016	neg	153002	na	664	186	0	0	0	0	0	...
60017	neg	2286	na	2130706538	224	0	0	0	0	0	...
60018	neg	112	0	2130706432	18	0	0	0	0	0	...
60019	neg	80292	na	2130706432	494	0	0	0	0	0	...
60020	neg	40222	na	698	628	0	0	0	0	0	...

60000 rows × 171 columns

In [67]:
```
data.head()
```

Out[67]:

20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee
21	neg	76698	na	2130706438	280	0	0	0	0	0	...	124
22	neg	33058	na	0	na	0	0	0	0	0	...	42
23	neg	41040	na	228	100	0	0	0	0	0	...	27
24	neg	12	0	70	66	0	10	0	0	0	...	
25	neg	60874	na	1368	458	0	0	0	0	0	...	62

5 rows × 171 columns

=>There are no duplicates in the dataset

In [68]:
```
data1=data
```

In [69]:
```
data1 = data1.reset_index(drop=True)
```

In [70]:
```
data = data1
```

=> Data Cleaning

In [71]:
```
data.head()
```

Out[71]:

20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee
0	neg	76698	na	2130706438	280	0	0	0	0	0	...	124
1	neg	33058	na	0	na	0	0	0	0	0	...	42
2	neg	41040	na	228	100	0	0	0	0	0	...	27
3	neg	12	0	70	66	0	10	0	0	0	...	
4	neg	60874	na	1368	458	0	0	0	0	0	...	62

5 rows × 171 columns

In [72]:
```
data.columns
```

Out[72]:
```
Index(['class', 'aa_000', 'ab_000', 'ac_000', 'ad_000', 'ae_000', 'af_000',
       'ag_000', 'ag_001', 'ag_002',
       ...
       'ee_002', 'ee_003', 'ee_004', 'ee_005', 'ee_006', 'ee_007', 'ee_008',
       'ee_009', 'ef_000', 'eg_000'],
      dtype='object', name=20, length=171)
```

In [73]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 171 entries, class to eg_000
dtypes: object(171)
memory usage: 78.3+ MB
```

In [74]:
```
data['class'].unique()
```

Out[74]:
```
array(['neg', 'pos'], dtype=object)
```

- **There are two outputs only therefore it is the case of classification problem**

In [75]:
```
data.isnull().sum()
```

Out[75]:
```
20
class     0
aa_000    0
ab_000    0
ac_000    0
ad_000    0
         ..
ee_007    0
ee_008    0
ee_009    0
ef_000    0
eg_000    0
Length: 171, dtype: int64
```

In [76]:
```
data.dtypes
```

Out[76]:
```
20
class     object
aa_000    object
ab_000    object
ac_000    object
ad_000    object
           ...
ee_007    object
ee_008    object
ee_009    object
ef_000    object
eg_000    object
Length: 171, dtype: object
```

In [77]:
```
data.select_dtypes(exclude = 'object')
```

Out[77]:

	20
0	
1	
2	
3	
4	
...	
59995	
59996	
59997	
59998	
59999	

60000 rows × 0 columns

=>It shows there is not such column which having data type of integers. Need to change data type to integers

In [78]:
```
#data['ab_000']=data['ab_000'].astype(int)
```

In [79]:
```
data['ab_000']=='na'
```

Out[79]:
```
0         True
1         True
2         True
3        False
4         True
         ...
59995     True
59996     True
59997    False
59998     True
59999     True
Name: ab_000, Length: 60000, dtype: bool
```

In [80]:
```
data = data.replace('na', np.nan)
```

In [81]:
```
data.head(20)
```

Out[81]:

20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee
0	neg	76698	NaN	2130706438	280	0	0	0	0	0	...	124
1	neg	33058	NaN	0	NaN	0	0	0	0	0	...	42
2	neg	41040	NaN	228	100	0	0	0	0	0	...	27
3	neg	12	0	70	66	0	10	0	0	0	...	
4	neg	60874	NaN	1368	458	0	0	0	0	0	...	62
5	neg	38312	NaN	2130706432	218	0	0	0	0	0	...	38
6	neg	14	0	6	NaN	0	0	0	0	0	...	
7	neg	102960	NaN	2130706432	116	0	0	0	0	0	...	71
8	neg	78696	NaN	0	NaN	0	0	0	0	0	...	69
9	pos	153204	0	182	NaN	0	0	0	0	0	...	12
10	neg	39196	NaN	204	170	0	0	0	0	0	...	19
11	neg	45912	NaN	0	454	0	0	0	0	0	...	49
12	neg	2104	NaN	36	26	0	0	0	0	9744	...	
13	neg	118950	NaN	1390	1298	0	0	0	0	0	...	167
14	neg	24416	NaN	0	NaN	0	0	0	0	0	...	19
15	neg	14	0	62	34	0	0	0	0	0	...	
16	neg	31300	0	784	740	0	0	0	0	0	...	79
17	neg	736	2	24	22	16	20	0	0	0	...	
18	neg	332	NaN	2130706432	20	0	0	0	0	0	...	
19	neg	1432	NaN	2130706440	82	0	0	0	0	0	...	

20 rows × 171 columns

In [82]:
```
data.isnull().sum()
```

Out[82]:
```
20
class         0
aa_000        0
ab_000    46329
ac_000     3335
ad_000    14861
          ...
ee_007      671
ee_008      671
ee_009      671
ef_000     2724
eg_000     2723
Length: 171, dtype: int64
```

- **So now it can be seen that there are numbers of Null values in the dataframe.**

In [83]:
```
data_column = data.columns
data_column
```

Out[83]:
```
Index(['class', 'aa_000', 'ab_000', 'ac_000', 'ad_000', 'ae_000', 'af_000',
       'ag_000', 'ag_001', 'ag_002',
       ...
       'ee_002', 'ee_003', 'ee_004', 'ee_005', 'ee_006', 'ee_007', 'ee_008',
       'ee_009', 'ef_000', 'eg_000'],
      dtype='object', name=20, length=171)
```

- **Handling the null values with the help of forward fill and back ward fill methods**

In [84]:
```
for col in data_column[1:]:
    data[col]=data[col].replace(np.nan,0)
```

In [85]:
```
data.head()
```

Out[85]:

20	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee_
0	neg	76698	0	2130706438	280	0	0	0	0	0	...	124
1	neg	33058	0	0	0	0	0	0	0	0	...	42
2	neg	41040	0	228	100	0	0	0	0	0	...	27
3	neg	12	0	70	66	0	10	0	0	0	...	
4	neg	60874	0	1368	458	0	0	0	0	0	...	62

5 rows × 171 columns

In [86]:
```
data.isnull().sum()
```

Out[86]:
```
20
class     0
aa_000    0
ab_000    0
ac_000    0
ad_000    0
          ..
ee_007    0
ee_008    0
ee_009    0
ef_000    0
eg_000    0
Length: 171, dtype: int64
```

- **So there is no null values in the dataframe**

- **lets convert the data types from 'Object' to 'integer' and 'float'.**

In [87]:
```
for col in data_column:
    if col == 'class':
        pass
    else:
        data[col]=data[col].astype(float)
```

In [88]:
```
for col in data_column:
    if col == 'class':
        pass
    else:
        data[col]=data[col].astype('int64')
```

In [89]:
```
data.dtypes
```

Out[89]:
```
20
class     object
aa_000     int64
ab_000     int64
ac_000     int64
ad_000     int64
           ...
ee_007     int64
ee_008     int64
ee_009     int64
ef_000     int64
eg_000     int64
Length: 171, dtype: object
```

In [90]:
```
data.describe()
```

Out[90]:

20	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	
count	6.000000e+04	60000.000000	6.000000e+04	6.000000e+04	60000.000000	60000.000000	6.000
mean	5.933650e+04	0.162500	3.362258e+08	1.434071e+05	6.535000	10.548200	2.191
std	1.454301e+05	1.687318	7.767625e+08	3.504525e+07	158.147893	205.387115	2.036
min	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000
25%	8.340000e+02	0.000000	8.000000e+00	0.000000e+00	0.000000	0.000000	0.000
50%	3.077600e+04	0.000000	1.200000e+02	4.200000e+01	0.000000	0.000000	0.000
75%	4.866800e+04	0.000000	8.480000e+02	2.920000e+02	0.000000	0.000000	0.000
max	2.746564e+06	204.000000	2.130707e+09	8.584298e+09	21050.000000	20070.000000	3.376

8 rows × 170 columns

In [91]:
```
data.to_csv("APS_failure_Scania_updated.csv", index=False)
```

In [92]:
```
data = pd.read_csv("APS_failure_Scania_updated.csv")
```

In [93]:
```
data
```

Out[93]:

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...
0	neg	76698	0	2130706438	280	0	0	0	0	0	...
1	neg	33058	0	0	0	0	0	0	0	0	...
2	neg	41040	0	228	100	0	0	0	0	0	...
3	neg	12	0	70	66	0	10	0	0	0	...
4	neg	60874	0	1368	458	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
59995	neg	153002	0	664	186	0	0	0	0	0	...
59996	neg	2286	0	2130706538	224	0	0	0	0	0	...
59997	neg	112	0	2130706432	18	0	0	0	0	0	...
59998	neg	80292	0	2130706432	494	0	0	0	0	0	...
59999	neg	40222	0	698	628	0	0	0	0	0	...

60000 rows × 171 columns

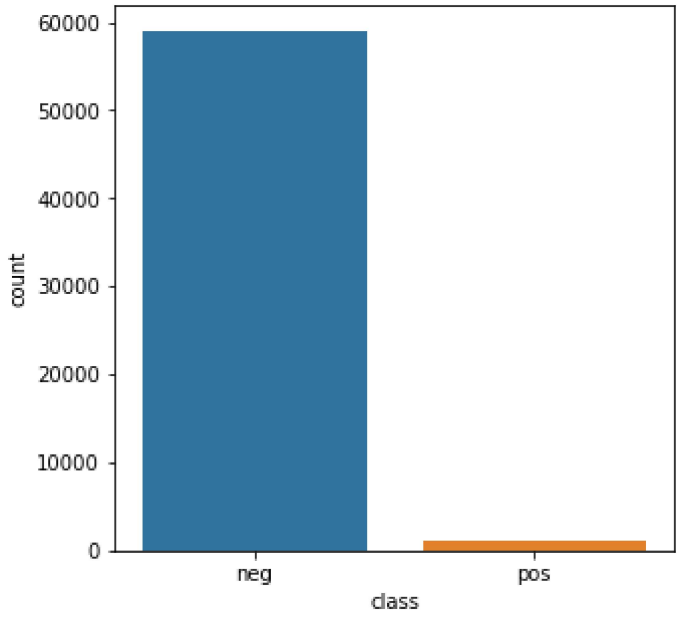
=> Memory usage of each columns

In [94]:
```
data.memory_usage()
```

Out[94]:
```
Index        128
class     480000
aa_000    480000
ab_000    480000
ac_000    480000
             ...
ee_007    480000
ee_008    480000
ee_009    480000
ef_000    480000
eg_000    480000
Length: 172, dtype: int64
```

In [95]:
```
plt.figure(figsize=(5, 5))
sns.countplot(x=data['class'])
```

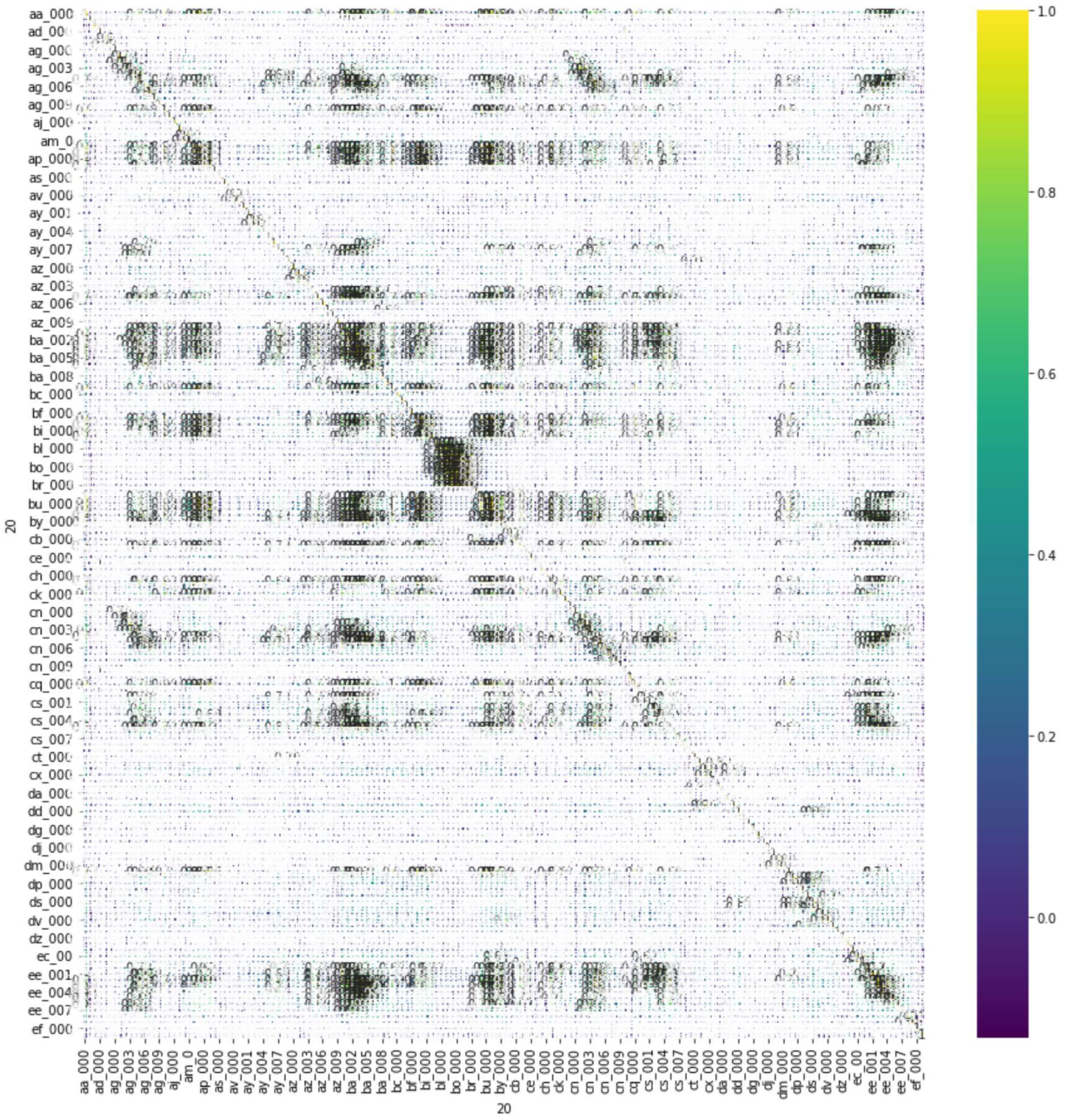
Out[95]:
```
<AxesSubplot:xlabel='class', ylabel='count'>
```



- **As it can be seen that very few positives are there**

In [81]:
```
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(),cmap = 'viridis',linewidths=0.5, annot= True)
```

Out[81]: <AxesSubplot:xlabel='20', ylabel='20'>



In [96]:
```
from sklearn.preprocessing import StandardScaler  # to standardize the features
from sklearn.decomposition import PCA  # to apply PCA
```

In [97]:
```
y = data['class']
x = data.drop('class', axis=1)
```

In [98]:
```
pca = PCA(n_components = 5)
pca.fit(x)
data_pca = pca.transform(x)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3','PC4','PC5'])
data_pca
```

Out[98]:

	PC1	PC2	PC3	PC4	PC5
0	1.794496e+09	3.211523e+05	-13914.100932	3.221798e+06	1.246086e+07
1	-3.361937e+08	-5.375804e+06	-284084.983415	-6.810371e+06	-1.193847e+05
2	-3.361927e+08	-5.608250e+06	-277150.003652	-1.020018e+07	-1.375904e+06
3	-3.361710e+08	-6.252036e+06	-263685.285572	-1.471480e+07	-8.833286e+06
4	-3.362444e+08	-4.891118e+06	-310272.149088	1.014545e+07	-5.099864e+06
...	...	...	...	...	...
59995	-3.364199e+08	-2.457138e+06	-415694.435132	7.192584e+07	-1.089942e+07
59996	1.794511e+09	-1.373746e+06	20155.983755	-7.800156e+06	-6.727554e+06
59997	1.794512e+09	-1.429219e+06	22479.383013	-9.193486e+06	-6.593671e+06
59998	1.794448e+09	9.781449e+05	-63584.040441	3.726327e+07	-3.386379e+06
59999	-3.362756e+08	-4.894499e+06	-328713.944386	2.487751e+07	-1.621831e+07

60000 rows × 5 columns

=>Getting the traning and test data

In [99]:
```
x_train, x_test, y_train, y_test = train_test_split(data_pca, y, test_size=0.33,
```

In [100]:
```
x_train
```

Out[100]:

	PC1	PC2	PC3	PC4	PC5
9567	-3.361711e+08	-6.252874e+06	-263701.145573	-1.471407e+07	-8.829998e+06
58473	-3.361864e+08	-5.318960e+06	-283051.181310	-8.439531e+06	2.151669e+06
10175	-3.361715e+08	-6.254555e+06	-263805.989289	-1.471714e+07	-8.838974e+06
48088	-3.361858e+08	-5.837176e+06	-271311.007405	-1.242323e+07	-3.645228e+06
8847	1.794512e+09	-1.435025e+06	22521.699410	-9.199432e+06	-6.607794e+06
...	...	...	...	...	...
33268	-3.361711e+08	-6.248183e+06	-263779.949974	-1.469553e+07	-8.792502e+06
44845	-3.362451e+08	-5.230466e+06	-310485.796531	1.354037e+07	-1.324324e+07
48045	-3.362017e+08	-5.356053e+06	-281359.381774	-8.580454e+06	1.646221e+06
4517	-3.361749e+08	-6.200549e+06	-266116.384370	-1.327598e+07	-9.049021e+06
38693	-3.361886e+08	-5.691133e+06	-274520.105260	-1.124985e+07	-1.927497e+06

40200 rows × 5 columns

In [101]:
```
y_train
```

Out[101]:
```
9567     neg
58473    neg
10175    neg
48088    neg
8847     neg
        ...
33268    neg
44845    neg
48045    neg
4517     neg
38693    neg
Name: class, Length: 40200, dtype: object
```

=>Training the model by SVM (SVC)

In [103]:
```
scaler = StandardScaler()
scaler.fit(x_train)
x_train_tf = scaler.transform(x_train)
x_test_tf = scaler.transform(x_test)
model=SVC()
model.fit(x_train_tf,y_train)
model.score(x_train_tf,y_train)
y_predict = model.predict(x_test_tf)
accuracy_SVC_PCM = accuracy_score(y_test, y_predict)
print('Accuracy score =', round(accuracy_SVC_PCM*100, 2), '%')
```

```
Accuracy score = 98.46 %
```

- **The above accuracy is getting after doing PCM**

In [104]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_
```

In [105]:
```
x_train
```

Out[105]:

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	...
9567	4	0	44	38	0	0	0	0	0	0	...
58473	40944	0	0	0	0	0	0	0	0	0	...
10175	4	0	0	0	0	0	0	0	0	0	...
48088	24138	0	0	0	0	0	0	0	1214	164512	...
8847	32	0	2130706446	52	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
33268	96	0	14	12	0	0	0	0	0	0	...
44845	28622	0	888	754	0	0	0	0	0	0	...
48045	40372	0	1358	1244	0	0	0	0	0	8042	...
4517	2404	0	26	26	0	0	0	0	0	0	...
38693	30032	0	762	634	0	0	0	0	0	0	...

40200 rows × 170 columns

In [106]:
```
y_train
```

Out[106]:
```
9567     neg
58473    neg
10175    neg
48088    neg
8847     neg
        ...
33268    neg
44845    neg
48045    neg
4517     neg
38693    neg
Name: class, Length: 40200, dtype: object
```

In [107]:
```
scaler = StandardScaler()
scaler.fit(x_train)
x_train_tf = scaler.transform(x_train)
x_test_tf = scaler.transform(x_test)
model=SVC()
model.fit(x_train_tf,y_train)
model.score(x_train_tf,y_train)
y_predict = model.predict(x_test_tf)
accuracy = accuracy_score(y_test, y_predict)
```

In [108]:
```
print(accuracy)
print('Accuracy score =', round(accuracy*100, 2), '%')
```

```
0.9872222222222222
Accuracy score = 98.72 %
```

- **There is not much differece in the both accuracies**

=> 1. Training the model using Logistic Regression

In [110]:
```
model_log = LogisticRegression()
model_log.fit(x_train,y_train)
y_predict = model.predict(x_test)
accuracy_log = accuracy_score(y_test, y_predict)
print(accuracy_log)
print('Accuracy score =', round(accuracy_log*100, 2), '%')
```

```
0.9833333333333333
Accuracy score = 98.33 %
```

=>Confusion Matrix

In [111]:
```
conf_mat = confusion_matrix(y_test,y_predict)
conf_mat
```

Out[111]:
```
array([[19470,     0],
       [  330,     0]], dtype=int64)
```

In [113]:
```
TP = conf_mat[0][0]
FP = conf_mat[0][1]
FN = conf_mat[1][0]
TN = conf_mat[1][1]
```

In [114]:
```
Accuracy_cof_mat = (TP + TN) / (TP + FP + FN + TN)
Accuracy_cof_mat
```

Out[114]:
```
0.9833333333333333
```

In [115]:
```
print(classification_report(y_predict,y_test))
```

```
              precision    recall  f1-score   support

         neg       1.00      0.98      0.99     19800
         pos       0.00      0.00      0.00         0

    accuracy                           0.98     19800
   macro avg       0.50      0.49      0.50     19800
weighted avg       1.00      0.98      0.99     19800
```

```
<span style="color:#800080 ;font-size:20px"> =><font face="Verdana"><b> ROC AUC Curve
```

In [ ]:
```
#auc = roc_auc_score(y_test, y_predict)
#auc
```

=> 2. Decision tree

In [116]:
```
model_dec_tree = DecisionTreeClassifier()
model_dec_tree.fit(x_train,y_train)
y_predict_dec_tree = model_dec_tree.predict(x_test)
accuracy_dec_tree = accuracy_score(y_test, y_predict_dec_tree)
print(accuracy_dec_tree)
print('Accuracy score =', round(accuracy_dec_tree*100, 2), '%')
```

```
0.9901515151515151
Accuracy score = 99.02 %
```

In [117]:
```
from sklearn import tree
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model_dec_tree,filled=True)
```

```
Text(0.9261943986820428, 0.7794117647058824, 'X[115] <= 8387833.0\ngini = 0.
365\nsamples = 25\nvalue = [6, 19]'),
 Text(0.9209225700164745, 0.75, 'X[10] <= 532378.0\ngini = 0.236\nsamples = 2
2\nvalue = [3, 19]'),
 Text(0.9156507413509061, 0.7205882352941176, 'X[46] <= 47313036.0\ngini = 0.
095\nsamples = 20\nvalue = [1, 19]'),
 Text(0.9103789126853378, 0.6911764705882353, 'gini = 0.0\nsamples = 19\nvalu
e = [0, 19]'),
 Text(0.9209225700164745, 0.6911764705882353, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.9261943986820428, 0.7205882352941176, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.9314662273476112, 0.75, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.9472817133443163, 0.8088235294117647, 'X[56] <= 816961.0\ngini = 0.16
7\nsamples = 98\nvalue = [9, 89]'),
 Text(0.9420098846787479, 0.7794117647058824, 'gini = 0.0\nsamples = 3\nvalue
= [3, 0]'),
 Text(0.9525535420098846, 0.7794117647058824, 'X[15] <= 30841.0\ngini = 0.118
\nsamples = 95\nvalue = [6, 89]'),
 Text(0.9472817133443163, 0.75, 'X[72] <= 175230.0\ngini = 0.082\nsamples = 9
```

In [118]:
```
fig.savefig("decision Tree Scania Truck.png")
```

=> Decision tree with best hyperparameters

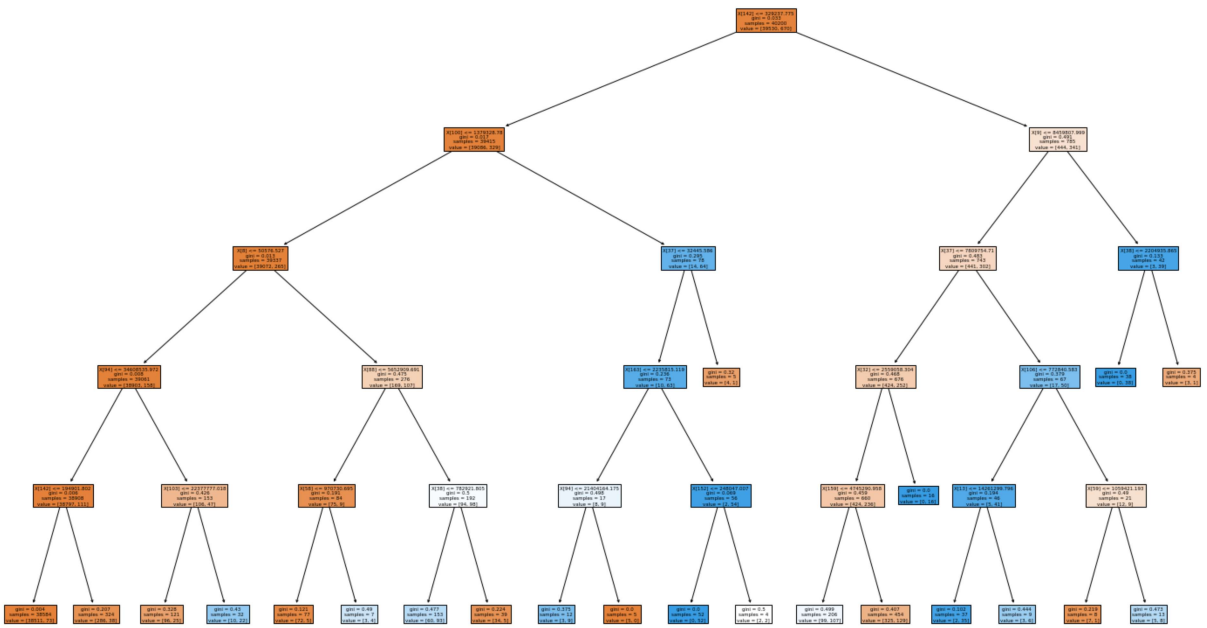
In [127]:
```
model_dec_tree_with_best_fit = DecisionTreeClassifier(criterion = 'gini',  max_de
model_dec_tree_with_best_fit.fit(x_train, y_train)
y_pred_dec_tree_with_best_fit = model_dec_tree_with_best_fit.predict(x_test)
accuracy_dec_tree_with_best_fit = accuracy_score(y_test, y_pred_dec_tree_with_bes
print(accuracy_dec_tree_with_best_fit)
print('Accuracy score =', round(accuracy_dec_tree_with_best_fit*100, 2), '%')
```

```
0.987979797979798
Accuracy score = 98.8 %
```

In [128]:
```
from sklearn import tree
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model_dec_tree_with_best_fit,filled=True)
```

Out[128]:
```
[Text(0.6317567567567568, 0.9166666666666666, 'X[142] <= 329237.775\ngini =
0.033\nsamples = 40200\nvalue = [39530, 670]'),
 Text(0.3918918918918919, 0.75, 'X[100] <= 1379328.78\ngini = 0.017\nsamples
= 39415\nvalue = [39086, 329]'),
 Text(0.21621621621621623, 0.5833333333333334, 'X[8] <= 50576.527\ngini = 0.0
13\nsamples = 39337\nvalue = [39072, 265]'),
 Text(0.10810810810810811, 0.4166666666666667, 'X[94] <= 34608535.972\ngini =
0.008\nsamples = 39061\nvalue = [38903, 158]'),
 Text(0.05405405405405406, 0.25, 'X[142] <= 194901.802\ngini = 0.006\nsamples
= 38908\nvalue = [38797, 111]'),
 Text(0.02702702702702703, 0.08333333333333333, 'gini = 0.004\nsamples = 3858
4\nvalue = [38511, 73]'),
 Text(0.08108108108108109, 0.08333333333333333, 'gini = 0.207\nsamples = 324
\nvalue = [286, 38]'),
 Text(0.16216216216216217, 0.25, 'X[103] <= 22377777.018\ngini = 0.426\nsampl
es = 153\nvalue = [106, 47]'),
 Text(0.13513513513513514, 0.08333333333333333, 'gini = 0.328\nsamples = 121
\nvalue = [96, 25]'),
 Text(0.1891891891891892, 0.08333333333333333, 'gini = 0.43\nsamples = 32\nva
lue = [10, 22]'),
 Text(0.32432432432432434, 0.4166666666666667, 'X[88] <= 5652909.691\ngini =
0.475\nsamples = 276\nvalue = [169, 107]'),
 Text(0.2702702702702703, 0.25, 'X[58] <= 970730.695\ngini = 0.191\nsamples =
84\nvalue = [75, 9]'),
 Text(0.24324324324324326, 0.08333333333333333, 'gini = 0.121\nsamples = 77\n
value = [72, 5]'),
 Text(0.2972972972972973, 0.08333333333333333, 'gini = 0.49\nsamples = 7\nval
ue = [3, 4]'),
 Text(0.3783783783783784, 0.25, 'X[38] <= 782921.805\ngini = 0.5\nsamples = 1
92\nvalue = [94, 98]'),
 Text(0.35135135135135137, 0.08333333333333333, 'gini = 0.477\nsamples = 153
\nvalue = [60, 93]'),
 Text(0.40540540540540543, 0.08333333333333333, 'gini = 0.224\nsamples = 39\n
value = [34, 5]'),
 Text(0.5675675675675675, 0.5833333333333334, 'X[37] <= 32445.586\ngini = 0.2
95\nsamples = 78\nvalue = [14, 64]'),
 Text(0.5405405405405406, 0.4166666666666667, 'X[163] <= 2235815.119\ngini =
0.236\nsamples = 73\nvalue = [10, 63]'),
 Text(0.4864864864864865, 0.25, 'X[94] <= 21404164.175\ngini = 0.498\nsamples
= 17\nvalue = [8, 9]'),
 Text(0.4594594594594595, 0.08333333333333333, 'gini = 0.375\nsamples = 12\nv
alue = [3, 9]'),
 Text(0.5135135135135135, 0.08333333333333333, 'gini = 0.0\nsamples = 5\nvalu
e = [5, 0]'),
 Text(0.5945945945945946, 0.25, 'X[152] <= 248047.007\ngini = 0.069\nsamples
= 56\nvalue = [2, 54]'),
 Text(0.5675675675675675, 0.08333333333333333, 'gini = 0.0\nsamples = 52\nval
ue = [0, 52]'),
 Text(0.6216216216216216, 0.08333333333333333, 'gini = 0.5\nsamples = 4\nvalu
e = [2, 2]'),
 Text(0.5945945945945946, 0.4166666666666667, 'gini = 0.32\nsamples = 5\nvalu
e = [4, 1]'),
```

```
 Text(0.8716216216216216, 0.75, 'X[9] <= 8459807.999\ngini = 0.491\nsamples =
785\nvalue = [444, 341]'),
 Text(0.7972972972972973, 0.5833333333333334, 'X[37] <= 7809754.71\ngini = 0.
483\nsamples = 743\nvalue = [441, 302]'),
 Text(0.7297297297297297, 0.4166666666666667, 'X[32] <= 2559058.304\ngini =
0.468\nsamples = 676\nvalue = [424, 252]'),
 Text(0.7027027027027027, 0.25, 'X[159] <= 4745290.958\ngini = 0.459\nsamples
= 660\nvalue = [424, 236]'),
 Text(0.6756756756756757, 0.08333333333333333, 'gini = 0.499\nsamples = 206\n
value = [99, 107]'),
 Text(0.7297297297297297, 0.08333333333333333, 'gini = 0.407\nsamples = 454\n
value = [325, 129]'),
 Text(0.7567567567567568, 0.25, 'gini = 0.0\nsamples = 16\nvalue = [0, 16]'),
 Text(0.8648648648648649, 0.4166666666666667, 'X[106] <= 772840.583\ngini =
0.379\nsamples = 67\nvalue = [17, 50]'),
 Text(0.8108108108108109, 0.25, 'X[13] <= 14261299.796\ngini = 0.194\nsamples
= 46\nvalue = [5, 41]'),
 Text(0.7837837837837838, 0.08333333333333333, 'gini = 0.102\nsamples = 37\nv
alue = [2, 35]'),
 Text(0.8378378378378378, 0.08333333333333333, 'gini = 0.444\nsamples = 9\nva
lue = [3, 6]'),
 Text(0.918918918918919, 0.25, 'X[59] <= 1059421.193\ngini = 0.49\nsamples =
21\nvalue = [12, 9]'),
 Text(0.8918918918918919, 0.08333333333333333, 'gini = 0.219\nsamples = 8\nva
lue = [7, 1]'),
 Text(0.9459459459459459, 0.08333333333333333, 'gini = 0.473\nsamples = 13\nv
alue = [5, 8]'),
 Text(0.9459459459459459, 0.5833333333333334, 'X[38] <= 2204935.865\ngini =
0.133\nsamples = 42\nvalue = [3, 39]'),
 Text(0.918918918918919, 0.4166666666666667, 'gini = 0.0\nsamples = 38\nvalue
= [0, 38]'),
 Text(0.972972972972973, 0.4166666666666667, 'gini = 0.375\nsamples = 4\nvalu
e = [3, 1]')]
```



In [129]:
```
fig.savefig("decision Tree Scania Truck with best fit.png")
```

=> 3. Random Forest

In [132]:
```
Rf_model=RandomForestClassifier()
Rf_model.fit(x_train,y_train)
y_pred_rf=Rf_model.predict(x_test)
accuracy_score_rf=accuracy_score(y_test,y_pred_rf)
print(accuracy_score_rf)
print('Accuracy score =', round(accuracy_score_rf*100, 2), '%')
```

```
0.9938888888888889
Accuracy score = 99.39 %
```

=> Random Forest with best hyper parameters

In [134]:
```
Rf_model_WBP=RandomForestClassifier(criterion='gini',max_depth= 14,max_features=
Rf_model_WBP.fit(x_train,y_train)
y_pred_rf_WBP=Rf_model_WBP.predict(x_test)
accuracy_score_rf_WBP=accuracy_score(y_test,y_pred_rf_WBP)
print(accuracy_score_rf_WBP)
print('Accuracy score =', round(accuracy_score_rf_WBP*100, 2), '%')
```

```
0.9926767676767677
Accuracy score = 99.27 %
```

=> 4. Bagging Classifier

In [138]:
```
model_bagging_svc = BaggingClassifier(base_estimator=SVC(),n_estimators=50, randc
model_bagging_svc.fit(x_train,y_train)
y_predict_bagging = model_bagging_svc.predict(x_test)
accuracy_score_bagging = accuracy_score(y_test,y_predict_bagging)
print(accuracy_score_bagging)
print('Accuracy score =', round(accuracy_score_bagging*100, 2), '%')
```

```
0.9836363636363636
Accuracy score = 98.36 %
```

=> 5. Adaboost

In [141]:
```
ada_model = AdaBoostClassifier()
ada_model.fit(x_train,y_train)
y_pred_ada=ada_model.predict(x_test)
accuracy_score_ada = accuracy_score(y_test,y_pred_ada)
print(accuracy_score_ada)
print('Accuracy score =', round(accuracy_score_ada*100, 2), '%')
```

```
0.9910606060606061
Accuracy score = 99.11 %
```

```
<span style="color:#ff1493;font-size:20px"> =><font face="Verdana"><b> Adaboost
with best hyperparameters
```

In [142]:
```
ada_model_with_best_params=AdaBoostClassifier(learning_rate= 0.003, n_estimators=
ada_model_with_best_params.fit(x_train,y_train)
y_predict_ada_bp = ada_model_with_best_params.predict(x_test)
accuracy_score_ada_with_best_params = accuracy_score(y_test,y_predict_ada_bp)
print(accuracy_score_ada_with_best_params)
print('Accuracy score =', round(accuracy_score_ada_with_best_params*100, 2), '%')
```

```
0.9833333333333333
Accuracy score = 98.33 %
```

=> 6. Gradian Boost

In [143]:
```
gradBoost_model = GradientBoostingClassifier()
gradBoost_model.fit(x_train,y_train)
y_predict_gradBoost = gradBoost_model.predict(x_test)
accuracy_score_gradboost = accuracy_score(y_test,y_predict_gradBoost)
print(accuracy_score_gradboost)
print('Accuracy score =', round(accuracy_score_gradboost*100, 2), '%')
```

```
0.9922727272727273
Accuracy score = 99.23 %
```

=> 7. XG Boost

In [144]:
```
model_xgb=xgboost.XGBClassifier()
model_xgb.fit(x_train,y_train)
y_pred_xgb=model_xgb.predict(x_test)
accuracy_score_xgb = accuracy_score(y_test,y_pred_xgb)
print(accuracy_score_xgb)
print('Accuracy score =', round(accuracy_score_xgb*100, 2), '%')
```

```
0.9936363636363637
Accuracy score = 99.36 %
```

In [150]:
```
print('1. Logistic Regression'+'=>'+ 'Accuracy score =', round(accuracy_log*100,
print('2. Decision tree'+'=>'+ 'Accuracy score       =', round(accuracy_dec_tree*
print('3. Random Forest'+'=>'+ 'Accuracy score       =', round(accuracy_score_rf*
print('4. Bagging Classifier'+'=>'+ 'Accuracy score  =', round(accuracy_score_bag
print('5. Adaboost'+'=>'+ 'Accuracy score            =', round(accuracy_score_ada
print('6. Gradian Boost'+'=>'+ 'Accuracy score       =', round(accuracy_score_gra
print('7. XG Boost'+'=>'+ 'Accuracy score            =', round(accuracy_score_xgb
```

```
1. Logistic Regression=>Accuracy score = 98.33 %
2. Decision tree=>Accuracy score       = 99.02 %
3. Random Forest=>Accuracy score       = 99.39 %
4. Bagging Classifier=>Accuracy score  = 98.36 %
5. Adaboost=>Accuracy score            = 99.11 %
6. Gradian Boost=>Accuracy score       = 99.23 %
7. XG Boost=>Accuracy score            = 99.36 %
```

=> Merci Beacoup

In [ ]: