Importing Necessary Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


import warnings
warnings.filterwarnings(action='ignore')
```

Data Ingestion

In [2]:
```
train_df = pd.read_csv('dataset/aps_failure_training_set_processed_8bit.csv')
train_labels = pd.read_csv('dataset/aps_failure_training_set.csv')['class']

test_df = pd.read_csv('dataset/aps_failure_test_set_processed_8bit.csv')
test_labels = pd.read_csv('dataset/aps_failure_test_set.csv')['class']
```

Exploring the Data

In [3]:
```
train_df['class'] = train_labels
test_df['class'] = test_labels
```

In [4]:
```
train_df.head()
```

Out[4]:

	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	
0	neg	0.117188	-0.289062	0.992188	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	0.687500	0.515625	0.234375	0.
1	neg	-0.179688	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	-0.023438	-0.062500	-0.132812	-0.
2	neg	-0.125000	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	-0.140625	-0.093750	-0.015625	0.
3	neg	-0.406250	-0.289062	-0.468750	-0.007812	-0.046875	-0.007812	-0.007812	-0.03125	-0.054688	...	-0.382812	-0.382812	-0.375000	-0.
4	neg	0.007812	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	0.156250	0.031250	-0.031250	-0.

In [5]:
```
test_df.head()
```

Out[5]:

	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	e
0	neg	-0.406250	-0.289062	-0.46875	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	-0.382812	-0.382812	-0.375000	-0.3
1	neg	-0.406250	-0.289062	-0.46875	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	-0.382812	-0.382812	-0.375000	-0.3
2	neg	0.046875	0.554688	-0.46875	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	0.046875	0.312500	-0.000000	-0.1
3	neg	0.000000	-0.289062	-0.46875	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	0.085938	0.062500	0.031250	0.0
4	neg	-0.390625	-0.289062	-0.46875	-0.007812	-0.046875	-0.054688	-0.007812	-0.03125	-0.054688	...	-0.375000	-0.375000	-0.359375	-0.3

In [6]:
```
train_df.describe()
```

Out[6]:

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag
count	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.00
mean	-0.124611	-0.071121	-0.198529	-0.007737	-0.033483	-0.040633	-0.006584	-0.026241	-0.040699	-0.07
std	0.367680	0.356812	0.564872	0.004138	0.107086	0.111752	0.032016	0.065200	0.105864	0.18
min	-0.406250	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688	-0.11
25%	-0.398438	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688	-0.11
50%	-0.195312	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688	-0.11
75%	-0.070312	0.000000	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688	-0.11
max	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.99

In [7]:
```
train_df.describe(include='all')
```

Out[7]:

	class	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002
count	60000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000
unique	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	neg	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	59000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	-0.124611	-0.071121	-0.198529	-0.007737	-0.033483	-0.040633	-0.006584	-0.026241	-0.040699
...	...	...	...	...	...	...	...	...	...	...
min	NaN	-0.406250	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688
25%	NaN	-0.398438	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688
50%	NaN	-0.195312	-0.289062	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688
75%	NaN	-0.070312	0.000000	-0.468750	-0.007812	-0.046875	-0.054688	-0.007812	-0.031250	-0.054688
max	NaN	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188	0.992188

Checking the Null Values

In [8]:
```
train_df.isnull().sum()
```

Out[8]:
```
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 [63]:
```
train_df.corr()
```

Out[63]:

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	...	ee_002	ee_003	ee
aa_000	1.000000	0.121999	0.091271	0.025188	0.110415	0.115131	0.067289	0.185408	0.327367	0.490721	...	0.867362	0.861476	0.83
ab_000	0.121999	1.000000	0.015372	0.011020	-0.017268	-0.016700	0.016740	0.042054	0.064040	0.076730	...	0.139755	0.124066	0.11
ac_000	0.091271	0.015372	1.000000	0.005124	0.007997	0.007327	0.021513	0.022585	0.022782	0.020065	...	0.102385	0.104665	0.09
ad_000	0.025188	0.011020	0.005124	1.000000	-0.001366	-0.001411	0.006440	0.016721	0.017864	0.014560	...	0.028775	0.023424	0.02
ae_000	0.110415	-0.017268	0.007997	-0.001366	1.000000	0.966834	-0.004659	0.000273	0.037658	0.138005	...	0.037365	0.046288	0.02
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
ee_007	0.714641	0.103732	0.066857	0.015753	0.124427	0.128893	0.043951	0.160518	0.354358	0.601018	...	0.595695	0.580086	0.55
ee_008	0.488417	0.034100	0.072585	0.004279	-0.058660	-0.057943	0.017027	0.021267	0.024793	0.018597	...	0.480937	0.482618	0.44
ee_009	0.302424	0.021730	0.047044	0.003246	-0.049432	-0.049671	-0.000373	-0.020576	-0.045709	-0.081193	...	0.295910	0.292490	0.26
ef_000	0.048662	0.011490	0.011207	0.000651	0.221157	0.218371	0.004524	0.019810	0.047570	0.080130	...	0.019257	0.017986	0.01
eg_000	0.025319	0.004143	0.017086	0.000069	0.204498	0.202610	0.002599	0.007825	0.030623	0.055827	...	-0.002936	-0.002632	-0.01

In [9]:
```
X = train_df.drop('class', axis=1)
Y = train_df['class']
```

Splitting into Train & Test Data

In [10]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3,random_state=0)
```

In [11]:
```
print("Data Shape: {}".format(train_df.shape))
print("Train Shape: {}".format(X_train.shape))
print("Test Shape: {}".format(X_test.shape))
```

```
Data Shape: (60000, 171)
Train Shape: (42000, 170)
Test Shape: (18000, 170)
```

Logistic Regression

In [12]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

Out[12]:
```
▾ LogisticRegression
LogisticRegression()
```

In [13]:
```
print("Test Accuracy: {:.2f}%".format(log_reg.score(X_test, y_test) * 100))
```

```
Test Accuracy: 99.12%
```

In [14]:
```
test_score = log_reg.score(X_test, y_test)
test_score_per_truck = test_score/X_test.shape[0]

print("Best model on test set (Cost = $ %0.2f):" % test_score)
print("Best model cost per truck on test set (Cost = $ %0.2f)" % test_score_per_truck)
```

```
Best model on test set (Cost = $ 0.99):
Best model cost per truck on test set (Cost = $ 0.00)
```

In [15]:
```
y_pred_logreg = log_reg.predict(X_test)
```

In [16]:
```
from sklearn.metrics import accuracy_score,classification_report
accuracy_score(y_test, y_pred_logreg)
```

Out[16]:
```
0.9911666666666666
```

In [17]:
```
print(classification_report(y_test, y_pred_logreg))
```

```
              precision    recall  f1-score   support

         neg       0.99      1.00      1.00     17702
         pos       0.82      0.60      0.69       298

    accuracy                           0.99     18000
   macro avg       0.91      0.80      0.84     18000
weighted avg       0.99      0.99      0.99     18000
```

Decision Tree Classifier

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

Out[18]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [19]:
```
y_pred_dt = dt.predict(X_test)
accuracy_score(y_test, y_pred_dt)
```

Out[19]:
```
0.9887777777777778
```

In [20]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

         neg       0.99      0.99      0.99     17702
         pos       0.66      0.65      0.66       298

    accuracy                           0.99     18000
   macro avg       0.83      0.82      0.83     18000
weighted avg       0.99      0.99      0.99     18000
```

Support Vector Classifier

In [21]:
```
from sklearn.svm import SVC
from sklearn.svm import LinearSVC,SVC
from sklearn.pipeline import make_pipeline
clf = SVC(C = 1.2, gamma =  0.9, kernel= 'rbf')
```

In [22]:
```
clf = clf.fit(X_train,y_train)
y_pred_svc = clf.predict(X_test)
```

In [23]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

         neg       0.98      1.00      0.99     17702
         pos       1.00      0.04      0.07       298

    accuracy                           0.98     18000
   macro avg       0.99      0.52      0.53     18000
weighted avg       0.98      0.98      0.98     18000
```

In [24]:
```
accuracy_score(y_test, y_pred_svc)
```

Out[24]:
```
0.9840555555555556
```

Ensemble Methods

In [25]:
```
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
```

In [26]:
```
extra_model = ExtraTreeClassifier()
extra_model.fit(X,Y)
feature_imp = extra_model.feature_importances_
```

In [27]:
```
for index, val in enumerate(feature_imp):
    print(index, round((val * 100), 2))
```

```
0 0.99
1 0.27
2 0.7
3 0.02
4 0.04
5 0.03
6 0.06
7 5.36
8 1.15
9 13.38
10 0.12
11 0.12
12 0.0
13 0.4
14 0.17
15 0.28
16 5.71
17 1.01
18 0.48
19 0.23
20 0.55
21 0.77
22 0.24
23 0.04
24 3.06
25 0.37
26 0.19
27 0.0
28 0.23
29 0.09
30 0.33
31 0.12
32 0.01
33 0.51
34 2.78
35 0.43
36 0.4
37 3.8
38 0.8
39 1.13
40 2.3
41 0.28
42 0.71
43 0.22
44 0.59
45 0.56
46 0.54
47 0.37
48 0.2
49 0.08
50 0.15
51 0.39
52 0.31
53 0.56
54 0.15
55 0.11
56 0.19
57 0.44
58 0.27
59 0.34
60 0.57
61 0.54
62 0.32
63 0.52
64 0.66
65 0.59
66 0.27
67 0.26
68 0.14
69 0.36
70 1.82
71 0.34
72 0.85
73 0.23
74 0.41
75 0.72
76 0.45
77 0.72
78 0.78
79 0.72
80 0.11
81 0.05
82 0.06
83 0.11
84 0.17
85 0.64
86 0.45
87 0.27
88 0.15
89 0.17
90 0.3
91 0.17
92 0.09
93 0.09
94 0.06
95 0.29
96 0.09
97 5.29
98 0.71
99 1.72
100 0.4
101 1.16
102 0.73
103 0.26
104 0.07
105 0.52
106 0.38
107 0.52
108 0.31
109 0.07
110 0.36
111 0.29
112 0.23
113 0.14
114 0.74
115 1.02
116 0.24
117 0.33
118 0.17
119 0.87
120 0.3
121 0.24
122 0.0
123 0.87
124 0.29
125 0.24
126 0.27
127 0.27
128 0.03
129 0.23
130 0.3
131 0.41
132 0.32
133 0.32
134 0.18
135 0.14
136 0.12
137 0.21
138 0.0
139 0.08
140 0.0
141 0.03
142 0.22
143 0.18
144 0.33
145 0.07
146 0.13
147 0.2
148 0.29
149 0.19
150 0.61
151 0.33
152 1.05
153 0.23
154 0.0
155 0.51
156 0.32
157 0.37
158 0.48
159 0.23
160 0.03
161 0.15
162 0.21
163 0.21
164 0.25
165 1.77
166 0.18
167 0.45
168 0.0
169 0.05
```

Random Forest Classifier

In [28]:
```
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

Out[28]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [29]:
```
y_pred_rf = rf.predict(X_test)
accuracy_score(y_test,y_pred_rf)
```

Out[29]:
```
0.9923333333333333
```

In [30]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

         neg       0.99      1.00      1.00     17702
         pos       0.93      0.58      0.71       298

    accuracy                           0.99     18000
   macro avg       0.96      0.79      0.85     18000
weighted avg       0.99      0.99      0.99     18000
```

Bagging Classifier

In [31]:
```
cls = BaggingClassifier(rf, random_state=0).fit(X_train, y_train)
cls.score(X_test, y_test)
```

Out[31]:
```
0.9914444444444445
```

In [32]:
```
y_pred_clf = cls.predict(X_test)
```

In [33]:
```
accuracy_score(y_test,y_pred_clf)
```

Out[33]:
```
0.9914444444444445
```

In [34]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_clf))
```

```
              precision    recall  f1-score   support

         neg       0.99      1.00      1.00     17702
         pos       0.93      0.52      0.67       298

    accuracy                           0.99     18000
   macro avg       0.96      0.76      0.83     18000
weighted avg       0.99      0.99      0.99     18000
```

Extra Tree Classifier

In [35]:
```
extra_tree = ExtraTreeClassifier(random_state=0)
cls_extra = BaggingClassifier(extra_tree, random_state=0).fit(X_train, y_train)
cls_extra.score(X_test, y_test)
```

Out[35]:
```
0.9908888888888889
```

Voting Classifier

In [36]:
```
clf1 = ExtraTreeClassifier(random_state=0)
clf2 = RandomForestClassifier()
clf3 = DecisionTreeClassifier()
clf4 = LogisticRegression()
clf5 = SVC(C = 1.2, gamma =  0.9, kernel= 'rbf')
```

In [37]:
```
eclf1 = VotingClassifier(estimators=[
    ('ETC', clf1), ('RF', clf2), ('DT', clf3),('Log_Reg', clf4),('SVC', clf5),('Bagging',cls),], voting='hard')
```

In [38]:
```
eclf1 = eclf1.fit(X, Y)
print(eclf1.predict(X))
```

```
['neg' 'neg' 'neg' ... 'neg' 'neg' 'neg']
```

Boosting Algorithms

Adaboost Classifier

In [39]:
```
from sklearn.ensemble import AdaBoostClassifier
ada_model = AdaBoostClassifier()
```

In [41]:
```
ada_model.fit(X_train,y_train)
```

Out[41]:
```
▾ AdaBoostClassifier
AdaBoostClassifier()
```

In [42]:
```
y_pred_ada = ada_model.predict(X_test)
```

In [43]:
```
accuracy_score(y_test,y_pred_ada)
```

Out[43]:
```
0.9904444444444445
```

In [44]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_ada))
```

```
              precision    recall  f1-score   support

         neg       0.99      1.00      1.00     17702
         pos       0.76      0.61      0.68       298

    accuracy                           0.99     18000
   macro avg       0.88      0.81      0.84     18000
weighted avg       0.99      0.99      0.99     18000
```

Gradient Boosting Algorithm

In [45]:
```
from sklearn.ensemble import GradientBoostingClassifier
GB_model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0)
```

In [46]:
```
GB_model.fit(X_train,y_train)
```

Out[46]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=0)
```

In [47]:
```
y_pred_gb = ada_model.predict(X_test)
accuracy_score(y_test,y_pred_gb)
```

Out[47]:
```
0.9904444444444445
```

In [51]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_gb))
```

```
              precision    recall  f1-score   support

         neg       0.99      1.00      1.00     17702
         pos       0.76      0.61      0.68       298

    accuracy                           0.99     18000
   macro avg       0.88      0.81      0.84     18000
weighted avg       0.99      0.99      0.99     18000
```

In [ ]:
```

```