Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Loading data

```
df = pd.read_csv('aps_failure_training_set.csv')
```

```
df.head()
```

```
  class  aa_000 ab_000      ac_000 ad_000 ae_000 af_000 ag_000 ag_001
ag_002  \
0   neg   76698     na  2130706438    280      0      0      0      0
0
1   neg   33058     na           0     na      0      0      0      0
0
2   neg   41040     na         228    100      0      0      0      0
0
3   neg      12      0          70     66      0     10      0      0
0
4   neg   60874     na        1368    458      0      0      0      0
0

   ...   ee_002  ee_003  ee_004  ee_005  ee_006  ee_007  ee_008 ee_009
ef_000  \
0  ...  1240520  493384  721044  469792  339156  157956   73224      0
0
1  ...   421400  178064  293306  245416  133654   81140   97576   1500
0
2  ...   277378  159812  423992  409564  320746  158022   95128    514
0
3  ...      240      46      58      44      10       0       0      0
4
4  ...   622012  229790  405298  347188  286954  311560  433954   1218
0

  eg_000
0      0
1      0
2      0
3     32
4      0

[5 rows x 171 columns]
```

Preprocessing (Filling missing values with mean) and mapping positive and negative to 1 and 0 in target variable

```
# Replace na by np.nan, so that we can use fillna() method of pandas
dataframe
df.replace('na', np.nan, inplace=True)

for col in df.columns:
    if col == 'class':
        continue

    null_percentage = (df[col].isnull().sum() / len(df[col])) * 100

    # If the amount of null values exceed 25% of total number of rows,
just ignore that column, as filling it with mean may not be that
useful.
    if null_percentage > 25:
        df.drop(col, axis=1, inplace=True)
        continue

    # Converting the object datatype into numeric (float) datatype.
    if df[col].dtype != np.float64:
        df[col] = pd.to_numeric(df[col],
errors='coerce').astype('Float64')

    df[col].fillna(df[col].mean(), inplace=True)

# Replacing the neg and pos by 0 and 1
df['class'] = df['class'].replace({'neg':0, 'pos':1})
```

```
/tmp/ipykernel_17830/731454417.py:16: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df[col].fillna(df[col].mean(), inplace=True)
/tmp/ipykernel_17830/731454417.py:16: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
```

```
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df[col].fillna(df[col].mean(), inplace=True)
/tmp/ipykernel_17830/731454417.py:16: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df[col].fillna(df[col].mean(), inplace=True)
/tmp/ipykernel_17830/731454417.py:16: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df[col].fillna(df[col].mean(), inplace=True)
/tmp/ipykernel_17830/731454417.py:18: FutureWarning: Downcasting
behavior in `replace` is deprecated and will be removed in a future
version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`
  df['class'] = df['class'].replace({'neg':0, 'pos':1})
```

```python
df['class'].value_counts()
```

```
class
0    59000
```

```
1      1000
Name: count, dtype: int64
```

There are 59000 negative cases and 1000 cases in positive. We will perform operations to handle the imbalance later in task 2.

Standardising data

```
from sklearn.preprocessing import StandardScaler

X = df.drop('class',axis=1)
y = df['class']

# Using standard scaler to scale the values of every column except the
target col between 0 and 1.
for col in X.columns:
    X[col] = StandardScaler().fit_transform(df[[col]])
```

Dimensionality reduction using PCA

```
from sklearn.decomposition import PCA

# Using PCA to reduce the dimension of the data while retaining
maximum information, so that our models could learn efficiently.
pca = PCA(n_components=0.95)

X_pca = pca.fit_transform(X)

X_pca.shape

(60000, 78)
```

We have reduced the number of features from 160 to 78.

Select K best features

```
from sklearn.feature_selection import SelectKBest,f_classif

# Algorithms like SVC still struggles to perform in the 68 column
data, thus we use select k best feature to select 5 most contributing
features.
X_kbest = SelectKBest(f_classif, k=5).fit_transform(X, y)

/home/ubuntu/miniconda3/envs/dsai/lib/python3.13/site-packages/
sklearn/feature_selection/_univariate_selection.py:112: UserWarning:
```

```
Features [80] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
UserWarning)
/home/ubuntu/miniconda3/envs/dsai/lib/python3.13/site-packages/sklearn
/feature_selection/_univariate_selection.py:113: RuntimeWarning:
invalid value encountered in divide
  f = msb / msw

X_kbest.shape

(60000, 5)
```

Splitting into test and train sets

```
from sklearn.model_selection import train_test_split

# All models excpet SVC will use this usual train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,
random_state=42)

# Only SVC will use this 5 col data for hyperparameter tuning, as it
is computationally expensive
X_train_k, X_test_k, y_train_k, y_test_k =
train_test_split(X_kbest,y,test_size=0.3,random_state=42)

X_train.shape , X_test.shape

((42000, 160), (18000, 160))

y_train.shape, y_test.shape

((42000,), (18000,))
```

Building Models on the imbalanced data to get basedline scores

Hyperparameter tuning on SVC

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

param_grid={
    'kernel':['rbf','linear'],
    'C': [0.01,0.1,1]
}
```

```
svc_grid = GridSearchCV(estimator=SVC(),
param_grid=param_grid,n_jobs=-1,verbose=1)

svc_grid.fit(X_train_k,y_train_k)

Fitting 5 folds for each of 6 candidates, totalling 30 fits

GridSearchCV(estimator=SVC(), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1], 'kernel': ['rbf',
'linear']},
             verbose=1)

print(f"Best parameters on svc are {svc_grid.best_params_}")

Best parameters on svc are {'C': 1, 'kernel': 'rbf'}

from sklearn.metrics import classification_report

print('Classification report on SVC')
classification_report(y_test_k, svc_grid.predict(X_test_k))

Classification report on SVC

'              precision    recall  f1-score   support\n\n           0
0.99      1.00      0.99     17698\n           1       0.65      0.26
0.38       302\n\n    accuracy                           0.99
18000\n   macro avg       0.82      0.63      0.68     18000\nweighted
avg       0.98      0.99      0.98     18000\n'
```

The F1 score for class 1 is 0.38, which is low. This could be due to the fact that we only chose K best features. Later,we will also try training the SVC model with above hyperparameters with original data to check its performance

Hyperparameter tuning on Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

param_grid={
    'solver':['liblinear'],
    'penalty': ['l1', 'l2'],
    'C': [0.01,  1, 10]
}

log_grid = GridSearchCV(estimator=LogisticRegression(),
param_grid=param_grid,verbose=1,scoring='f1_weighted',n_jobs=-1)

log_grid.fit(X_train,y_train)

Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
GridSearchCV(estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [0.01, 1, 10], 'penalty': ['l1', 'l2'],
                         'solver': ['liblinear']},
             scoring='f1_weighted', verbose=1)
```

```
print(f"Best parameters on logistic regression are
{log_grid.best_params_}")
```

```
Best parameters on logistic regression are {'C': 1, 'penalty': 'l1',
'solver': 'liblinear'}
```

```
from sklearn.metrics import classification_report
```

```
print('Classification report on logistic regression')
classification_report(y_test, log_grid.predict(X_test))
```

```
Classification report on logistic regression
```

```
'              precision    recall  f1-score   support\n\n           0
0.99      1.00      1.00     17698\n           1       0.79      0.68
0.73       302\n\n    accuracy                           0.99
18000\n   macro avg       0.89      0.84      0.86     18000\nweighted
avg       0.99      0.99      0.99     18000\n'
```

The F1 score for class 1 is 0.73, which is good compared to SVC with K best features.

Hyperparameter tuning on Descision tree

```
from sklearn.tree import DecisionTreeClassifier

param_grid={
    'max_depth': [3, 5,10,None],
    'min_samples_leaf': [1, 5, 10]
}

dt_grid = GridSearchCV(estimator=DecisionTreeClassifier(),
param_grid=param_grid,n_jobs=-1)

dt_grid.fit(X_train,y_train)
```

```
GridSearchCV(estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 10, None],
                         'min_samples_leaf': [1, 5, 10]})
```

```
print(f"Best parameters on Decision Trees are {dt_grid.best_params_}")
```

```
Best parameters on Decision Trees are {'max_depth': 5,
'min_samples_leaf': 10}
```

```
print('Classification report on Decision Trees')
classification_report(y_test, dt_grid.predict(X_test))
```

```
Classification report on Decision Trees

'              precision    recall  f1-score   support\n\n           0
0.99      1.00      0.99     17698\n           1       0.77      0.56
0.64       302\n\n    accuracy                           0.99
18000\n   macro avg       0.88      0.78      0.82     18000\nweighted
avg       0.99      0.99      0.99     18000\n'
```

Here, we get the F1 score for class 1 as 0.64, which is poor than logistic regression model.

Task 2 - Handling class imbalance

a) Consider undersampling the majority class and/or oversampling the minority class. -- SMOTE ANALYSIS

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)

X_smote, y_smote = smote.fit_resample(X_train, y_train)

y_smote.value_counts()

class
0    41302
1    41302
Name: count, dtype: int64

# fitting X_smote and y_smote on the above models with best parameter.
from sklearn.svm import SVC
svc = SVC(max_iter=10000,C=1, kernel='rbf')
svc.fit(X_smote,y_smote)
print('Classification report on SVC using smote')
print(classification_report(y_test,svc.predict(X_test)))
print(' ')

log =
LogisticRegression(max_iter=10000,C=1,penalty='l1',solver='liblinear')
log.fit(X_smote,y_smote)
print('Classification report on Logistic Regression using smote')
print(classification_report(y_test,log.predict(X_test)))
print(' ')

dt = DecisionTreeClassifier(max_depth=5, min_samples_leaf=10)
dt.fit(X_smote,y_smote)
print('Classification report on Decision Tree using smote')
print(classification_report(y_test,dt.predict(X_test)))
print(' ')
```

```
/home/ubuntu/miniconda3/envs/dsai/lib/python3.13/site-packages/
sklearn/svm/_base.py:297: ConvergenceWarning: Solver terminated early
(max_iter=10000).  Consider pre-processing your data with
StandardScaler or MinMaxScaler.
  warnings.warn(

Classification report on SVC using smote
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     17698
           1       0.44      0.83      0.57       302

    accuracy                           0.98     18000
   macro avg       0.72      0.91      0.78     18000
weighted avg       0.99      0.98      0.98     18000


Classification report on Logistic Regression using smote
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     17698
           1       0.37      0.88      0.52       302

    accuracy                           0.97     18000
   macro avg       0.68      0.93      0.75     18000
weighted avg       0.99      0.97      0.98     18000


Classification report on Decision Tree using smote
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     17698
           1       0.29      0.89      0.43       302

    accuracy                           0.96     18000
   macro avg       0.64      0.93      0.71     18000
weighted avg       0.99      0.96      0.97     18000
```

The F1 score for 1 and the macro average is high for SVC compared to logistic regression and decision trees.

b) Consider using class_weight which is inversely proportional to the class population.

```
# Add class weight = balanced in the training process

svc = SVC(class_weight='balanced', max_iter=10000,C=1, kernel='rbf')
svc.fit(X_train,y_train)
```

```
print('Classification report on SVC using Class weight = Balanced')
print(classification_report(y_test,svc.predict(X_test)))
print(' ')

log =
LogisticRegression(class_weight='balanced',max_iter=10000,C=1,penalty=
'l1',solver='liblinear')
log.fit(X_train,y_train)
print('Classification report on Logistic Regression using Class weight
= Balanced')
print(classification_report(y_test,log.predict(X_test)))
print(' ')

dt = DecisionTreeClassifier(class_weight='balanced',max_depth=5,
min_samples_leaf=10)
dt.fit(X_train,y_train)
print('Classification report on Decision Tree using Class weight =
Balanced')
print(classification_report(y_test,dt.predict(X_test)))
print(' ')
```

```
Classification report on SVC using Class weight = Balanced
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     17698
           1       0.53      0.87      0.66       302

    accuracy                           0.98     18000
   macro avg       0.77      0.93      0.83     18000
weighted avg       0.99      0.98      0.99     18000


Classification report on Logistic Regression using Class weight =
Balanced
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     17698
           1       0.38      0.90      0.53       302

    accuracy                           0.97     18000
   macro avg       0.69      0.94      0.76     18000
weighted avg       0.99      0.97      0.98     18000


Classification report on Decision Tree using Class weight = Balanced
              precision    recall  f1-score   support

           0       1.00      0.95      0.98     17698
           1       0.25      0.93      0.40       302
```

```
    accuracy                           0.95     18000
   macro avg       0.63      0.94      0.69     18000
weighted avg       0.99      0.95      0.97     18000
```

When using class weight as balanced, still SVC managed to outperform Logistic Regression and Decision Trees. Also it has better performance than SMOTE technique.

c) Consider using sample_weights, where you may assign a penalty for misclassifying every data point depending on the class it falls in.

```
# usinf sample weights
from sklearn.utils.class_weight import compute_sample_weight

sample_weights = compute_sample_weight(class_weight='balanced',
y=y_train)

svc = SVC(max_iter=10000,C=1, kernel='rbf')
svc.fit(X_train,y_train,sample_weight=sample_weights)
print('Classification report on SVC using sample weight')
print(classification_report(y_test,svc.predict(X_test)))
print(' ')

log =
LogisticRegression(max_iter=10000,C=1,penalty='l1',solver='liblinear')
log.fit(X_train,y_train,sample_weight=sample_weights)
print('Classification report on Logistic Regression using sample
weight ')
print(classification_report(y_test,log.predict(X_test)))
print(' ')

dt = DecisionTreeClassifier(max_depth=5, min_samples_leaf=10)
dt.fit(X_train,y_train,sample_weight=sample_weights)
print('Classification report on Decision Tree using sample weight')
print(classification_report(y_test,dt.predict(X_test)))
print(' ')
```

```
Classification report on SVC using sample weight
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     17698
           1       0.53      0.87      0.66       302

    accuracy                           0.98     18000
   macro avg       0.77      0.93      0.83     18000
weighted avg       0.99      0.98      0.99     18000


Classification report on Logistic Regression using sample weight
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     17698
           1       0.38      0.90      0.53       302

    accuracy                           0.97     18000
   macro avg       0.69      0.94      0.76     18000
weighted avg       0.99      0.97      0.98     18000


Classification report on Decision Tree using sample weight
              precision    recall  f1-score   support

           0       1.00      0.95      0.98     17698
           1       0.25      0.94      0.40       302

    accuracy                           0.95     18000
   macro avg       0.63      0.95      0.69     18000
weighted avg       0.99      0.95      0.97     18000
```

We get similar result when using sample weight as balanced.

My creative way:

```
- To use bagging of various models that are trained on data that has
equal number of classes (max no of data points in each iteration will
the no of data points in minority class)
```

```
y_train.value_counts()
```

```
class
0    41302
1      698
Name: count, dtype: int64
```

```
n = 41302 // 698
one_cnt = 698
```

```
X_0=[]
X_1=[]
```

```
for i in range(X_train.shape[0]):
    if y_train.iloc[i] == 0:
        X_0.append(X_train.iloc[i,:].to_numpy())
    else:
        X_1.append(X_train.iloc[i,:].to_numpy())
```

```
len(X_0), len(X_1)
```

```
(41302, 698)
```

```python
models=[SVC(C=1, kernel='rbf') for _ in range(n)]

for i in range(n):
    X1 = np.vstack((np.array(X_0[i*one_cnt : (i+1)*one_cnt]) ,
np.array(X_1)))
    y1 = np.append(np.zeros(one_cnt) , np.ones(one_cnt))

    models[i].fit(X1,y1)
    print(f"Model No: {i} trained successfully")
```

```
Model No: 0 trained successfully
Model No: 1 trained successfully
Model No: 2 trained successfully
Model No: 3 trained successfully
Model No: 4 trained successfully
Model No: 5 trained successfully
Model No: 6 trained successfully
Model No: 7 trained successfully
Model No: 8 trained successfully
Model No: 9 trained successfully
Model No: 10 trained successfully
Model No: 11 trained successfully
Model No: 12 trained successfully
Model No: 13 trained successfully
Model No: 14 trained successfully
Model No: 15 trained successfully
Model No: 16 trained successfully
Model No: 17 trained successfully
Model No: 18 trained successfully
Model No: 19 trained successfully
Model No: 20 trained successfully
Model No: 21 trained successfully
Model No: 22 trained successfully
Model No: 23 trained successfully
Model No: 24 trained successfully
Model No: 25 trained successfully
Model No: 26 trained successfully
Model No: 27 trained successfully
Model No: 28 trained successfully
Model No: 29 trained successfully
Model No: 30 trained successfully
Model No: 31 trained successfully
Model No: 32 trained successfully
Model No: 33 trained successfully
Model No: 34 trained successfully
Model No: 35 trained successfully
Model No: 36 trained successfully
Model No: 37 trained successfully
Model No: 38 trained successfully
Model No: 39 trained successfully
```

```
Model No: 40 trained successfully
Model No: 41 trained successfully
Model No: 42 trained successfully
Model No: 43 trained successfully
Model No: 44 trained successfully
Model No: 45 trained successfully
Model No: 46 trained successfully
Model No: 47 trained successfully
Model No: 48 trained successfully
Model No: 49 trained successfully
Model No: 50 trained successfully
Model No: 51 trained successfully
Model No: 52 trained successfully
Model No: 53 trained successfully
Model No: 54 trained successfully
Model No: 55 trained successfully
Model No: 56 trained successfully
Model No: 57 trained successfully
Model No: 58 trained successfully
```

```python
# prediction

predictions = np.zeros(X_test.shape[0])

for i in range(59):
    predictions += models[i].predict(X_test.to_numpy())
    print(f"model NO: {i} predicted successfully")

print(predictions)
```

```
model NO: 0 predicted successfully
model NO: 1 predicted successfully
model NO: 2 predicted successfully
model NO: 3 predicted successfully
model NO: 4 predicted successfully
model NO: 5 predicted successfully
model NO: 6 predicted successfully
model NO: 7 predicted successfully
model NO: 8 predicted successfully
model NO: 9 predicted successfully
model NO: 10 predicted successfully
model NO: 11 predicted successfully
model NO: 12 predicted successfully
model NO: 13 predicted successfully
model NO: 14 predicted successfully
model NO: 15 predicted successfully
model NO: 16 predicted successfully
model NO: 17 predicted successfully
model NO: 18 predicted successfully
model NO: 19 predicted successfully
```

```
model NO: 20 predicted successfully
model NO: 21 predicted successfully
model NO: 22 predicted successfully
model NO: 23 predicted successfully
model NO: 24 predicted successfully
model NO: 25 predicted successfully
model NO: 26 predicted successfully
model NO: 27 predicted successfully
model NO: 28 predicted successfully
model NO: 29 predicted successfully
model NO: 30 predicted successfully
model NO: 31 predicted successfully
model NO: 32 predicted successfully
model NO: 33 predicted successfully
model NO: 34 predicted successfully
model NO: 35 predicted successfully
model NO: 36 predicted successfully
model NO: 37 predicted successfully
model NO: 38 predicted successfully
model NO: 39 predicted successfully
model NO: 40 predicted successfully
model NO: 41 predicted successfully
model NO: 42 predicted successfully
model NO: 43 predicted successfully
model NO: 44 predicted successfully
model NO: 45 predicted successfully
model NO: 46 predicted successfully
model NO: 47 predicted successfully
model NO: 48 predicted successfully
model NO: 49 predicted successfully
model NO: 50 predicted successfully
model NO: 51 predicted successfully
model NO: 52 predicted successfully
model NO: 53 predicted successfully
model NO: 54 predicted successfully
model NO: 55 predicted successfully
model NO: 56 predicted successfully
model NO: 57 predicted successfully
model NO: 58 predicted successfully
[0. 0. 0. ... 0. 0. 0.]
```

```
predictions_scaled = predictions / 59
y_pred = np.where(predictions_scaled < 0.5, 0, 1)

print(pd.Series(predictions).value_counts())
pd.Series(y_pred).value_counts()
```

```
0.0     16812
59.0      836
58.0       49
```

57.0	23
1.0	23
2.0	18
56.0	17
3.0	14
54.0	11
55.0	11
4.0	10
9.0	8
52.0	7
6.0	7
8.0	7
40.0	7
53.0	7
17.0	6
26.0	6
51.0	6
16.0	5
42.0	5
27.0	5
38.0	5
21.0	5
7.0	5
34.0	5
18.0	5
50.0	4
25.0	4
14.0	4
10.0	4
23.0	4
46.0	3
43.0	3
49.0	3
15.0	3
41.0	3
28.0	3
32.0	3
5.0	3
45.0	2
44.0	2
13.0	2
19.0	2
33.0	2
48.0	2
20.0	2
11.0	2
31.0	2
12.0	2
47.0	2

```
30.0        2
39.0        2
35.0        2
36.0        2
22.0        1
Name: count, dtype: int64
```

```
0    16972
1     1028
Name: count, dtype: int64
```

```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     17698
           1       0.27      0.92      0.42       302

    accuracy                           0.96     18000
   macro avg       0.64      0.94      0.70     18000
weighted avg       0.99      0.96      0.97     18000
```

Here, we can see that performace is poor than the `smote` technique and `class_weight = balanced` technique. This approach is not good.