Ishan Prabhune

A20538828

In [1]:
```
#Importing the required Python Libraries
import numpy as npy
import scipy as spy
import pandas as pds
import matplotlib as mpl
import seaborn as sbn
from IPython.display import display, HTML

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

In [2]:
```
#Importing the Malware_Multiclass Dataset
DataFrame = pds.read_csv("C:/Users/ishan/Downloads/Malware Data/malware_MultiCla

# Printing and Display the Dataframe in HTML
display(HTML(DataFrame.head(10).to_html()))
```

	hash	millisecond	classi
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	0	r
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	1	r
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	2	r
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	3	r
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	4	r
5	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	5	r
6	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	6	r
7	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	7	r
8	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	8	r
9	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914e223349672eca79ad0	9	r

In [3]:
```
# Printing the Shape of the dataframe
print(DataFrame.shape)

# Selected required columns
SelectedColumns = ['classification','os','usage_counter','prio','static_prio','n

# Extracting the required selected columns
DataFrame = DataFrame[SelectedColumns]

# Displaying the selected data
DataFrame.head(5)
```

```
(100000, 36)
```

Out[3]:

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgo
0	malware	CentOS	0	3069378560	14274	0	
1	malware	Windows	0	3069378560	14274	0	
2	malware	Mac	0	3069378560	14274	0	
3	malware	Ubuntu	0	3069378560	14274	0	
4	malware	Mac	0	3069378560	14274	0	

In [4]:
```
# Striping the column names
DataFrame=DataFrame.rename(columns=lambda x: x.strip())
cols=DataFrame.columns

# Displaying rows of the DataFrame
DataFrame.head(5)
```

Out[4]:

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgo
0	malware	CentOS	0	3069378560	14274	0	
1	malware	Windows	0	3069378560	14274	0	
2	malware	Mac	0	3069378560	14274	0	
3	malware	Ubuntu	0	3069378560	14274	0	
4	malware	Mac	0	3069378560	14274	0	

In [5]:
```
# Checking for missing values in DataFrame
print('ColumnName, DataType, MissingValues')
for i in cols:
    print(i, ',', DataFrame[i].dtype,',', DataFrame[i].isnull().any())
```

```
ColumnName, DataType, MissingValues
classification , object , False
os , object , False
usage_counter , int64 , False
prio , int64 , False
static_prio , int64 , False
normal_prio , int64 , False
vm_pgoff , int64 , False
vm_truncate_count , int64 , False
task_size , int64 , False
map_count , int64 , False
hiwater_rss , int64 , False
total_vm , int64 , False
shared_vm , int64 , False
exec_vm , int64 , False
reserved_vm , int64 , False
nr_ptes , int64 , False
nvcsw , int64 , False
nivcsw , int64 , False
signal_nvcsw , int64 , False
```

In [6]:
```
# Encoding the labels

from sklearn import preprocessing
from IPython.display import display, HTML

# defining label as nominal values
y = DataFrame['classification']
le = preprocessing.LabelEncoder()
le.fit(y)

# Encoding the nominal labels to integers
y_encoded = le.transform(y)
DataFrame['classification'] = y_encoded

# Printing and displaying dataframe as tables in HTML
display(HTML(DataFrame.head(10).to_html()))
```

	classification	os	usage_counter	prio	static_prio	normal_prio	vm_pgoff
0	1	CentOS	0	3069378560	14274	0	0
1	1	Windows	0	3069378560	14274	0	0
2	1	Mac	0	3069378560	14274	0	0
3	1	Ubuntu	0	3069378560	14274	0	0
4	1	Mac	0	3069378560	14274	0	0
5	1	Windows	0	3069378560	14274	0	0
6	1	Ubuntu	0	3069378560	14274	0	0
7	1	Mac	0	3069378560	14274	0	0
8	1	CentOS	0	3069378560	14274	0	0
9	1	Mac	0	3069378560	14274	0	0

In [7]:
```
# Data preprocessing

print('Column Datatypes:\n',DataFrame.dtypes)

# Converting all the nominal variables to binary variables
DataFrame_num=DataFrame.copy(deep=True)

# Creating the new binary columns
DataFrame_dummies=pds.get_dummies(DataFrame_num[['os']])

# Adding them to the dataframe
DataFrame_num=DataFrame_num.join(DataFrame_dummies)

# Droping the original columns
DataFrame_num=DataFrame_num.drop('os',axis=1)

display('DataFrame_num:',HTML(DataFrame_num.head(5).to_html()))
```

```
Column Datatypes:
 classification        int32
os                    object
usage_counter          int64
prio                   int64
static_prio            int64
normal_prio            int64
vm_pgoff               int64
vm_truncate_count      int64
task_size              int64
map_count              int64
hiwater_rss            int64
total_vm               int64
shared_vm              int64
exec_vm                int64
reserved_vm            int64
nr_ptes                int64
nvcsw                  int64
nivcsw                 int64
signal_nvcsw           int64
dtype: object
'DataFrame_num:'
```

	classification	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_trun
0	1	0	3069378560	14274	0	0	
1	1	0	3069378560	14274	0	0	
2	1	0	3069378560	14274	0	0	
3	1	0	3069378560	14274	0	0	
4	1	0	3069378560	14274	0	0	

In [8]:
```
# N-1 dummy variable is encoded to avoid the multicollinearity issues
DataFrame_num=DataFrame_num.drop('os_Windows', axis=1)
DataFrame_num.head(10)
```

Out[8]:

	classification	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_tru
0	1	0	3069378560	14274	0	0	
1	1	0	3069378560	14274	0	0	
2	1	0	3069378560	14274	0	0	
3	1	0	3069378560	14274	0	0	
4	1	0	3069378560	14274	0	0	
5	1	0	3069378560	14274	0	0	
6	1	0	3069378560	14274	0	0	
7	1	0	3069378560	14274	0	0	
8	1	0	3069378560	14274	0	0	
9	1	0	3069378560	14274	0	0	

10 rows × 22 columns

In [10]:
```
# Standarizing the data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
DataFrame_std = DataFrame_num.copy(deep=True)
x_features = DataFrame_num.loc[:, DataFrame_std.columns != 'classification']
cols = x_features.columns
DataFrame_std = pds.DataFrame(scaler.fit_transform(x_features), columns = cols)

DataFrame_std['classification'] = y_encoded
display('data_frame_num_standardized:',HTML(DataFrame_std.head(10).to_html()))
```

'data_frame_num_standardized:'

	usage_counter	prio	static_prio	normal_prio	vm_pgoff	vm_truncate_count	task
0	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
1	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
2	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
3	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
4	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
5	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
6	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
7	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
8	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	
9	0.0	-1.105059	-0.848177	0.0	0.0	-0.657076	

In [17]:
```
# Solutions for Imbalances

# Checking for Imbalances in the DataSet
print(DataFrame_std.groupby(['classification']).size())

# Note that, imbalance solutions are only applied on training set.
# In terms of N-folds, you have to split data into train-test splits, and apply
from sklearn.model_selection import KFold

# Assume last column is your label, other columns are features
from sklearn import neighbors
from sklearn.metrics import accuracy_score
from collections import Counter
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

x=DataFrame_std.drop('classification',axis=1)
y=DataFrame_std['classification']

rosam = RandomOverSampler(random_state=10)
rosam.fit(x, y)
print('\nOriginal dataset shape {}'.format(Counter(y)))
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After Oversampling the shape of the dataset {}'.format(Counter(y_resample

print('\nOriginal dataset shape {}'.format(Counter(y)))
rosam = RandomUnderSampler(random_state=30)
rosam.fit(x, y)
x_resampled, y_resampled = ros.fit_resample(x, y)
print('After Undersampling the shape of the dataset {}'.format(Counter(y_resampl
```

```
classification
0    49858
1    49871
2      271
dtype: int64

Original dataset shape Counter({1: 49871, 0: 49858, 2: 271})
After Oversampling the shape of the dataset Counter({1: 49871, 2: 49871, 0: 4987
1})

Original dataset shape Counter({1: 49871, 0: 49858, 2: 271})
After Undersampling the shape of the dataset Counter({1: 49871, 2: 49871, 0: 4987
1})
```

In [18]:
```
# SMOTE is applied on the numerical features to get the features and labels

rosam = SMOTE(k_neighbors=2)
rosam.fit(x, y)
print('\n The Original shape of the dataset {}'.format(Counter(y)))
x_resampled, y_resampled = rosam.fit_resample(x, y)
print('After the Oversampling procedure by using SMOTE shape of the Dataset {}'.
```

```
 The Original shape of the dataset Counter({1: 49871, 0: 49858, 2: 271})
After the Oversampling procedure by using SMOTE shape of the Dataset Counter({1:
49871, 2: 49871, 0: 49871})
```

In [37]:
```
# Logistic Regression - All Features - Random Over Sampling
```

```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from imblearn.over_sampling import RandomOverSampler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from sklearn import neighbors
import numpy as np

kfld = KFold(n_splits=5, shuffle=True)
data_5_folds = []

for train_index, test_index in kfld.split(x, y):
    print("\nTRAIN:", train_index, "TEST:", test_index)
    # Getting the actual data by index
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Saving the data into folds
    folds = [x_train, x_test, y_train, y_test]
    # Adding each folds data into the 5_folds
    data_5_folds.append(folds)

for k in range(5, 6, 2):
    f1_5_folds = []
    auc_5_folds = []

    # Iterating through each of the folds
    for x_train, x_test, y_train, y_test in data_5_folds:

        # Applying random over-sampling
        rosam = RandomOverSampler(random_state=10)
        x_resampled, y_resampled = rosam.fit_resample(x_train, y_train)

        # logistic regression
        clfn_logistic = LogisticRegression(multi_class='ovr')
        clfn_logistic.fit(x_resampled, y_resampled)
        y_pred_proba_logistic = clfn_logistic.predict_proba(x_test)

        # Calculating F1 score
        f1_sc = f1_score(y_test, np.argmax(y_pred_proba_logistic, axis=1), avera
        f1_5_folds.append(f1_sc)

        # Calculating AUC
        auc = roc_auc_score(y_test, y_pred_proba_logistic, average='micro', mult
        auc_5_folds.append(auc)

print('\nThe F1 Score for all features with Logistic Regression Random Oversampl
print('\n The AUC for all features with Logistic Regression Random Oversampling
```

```
TRAIN: [    0     1     2 ... 99996 99997 99998] TEST: [   12    20    26 ... 999
93 99994 99999]

TRAIN: [    0     1     2 ... 99996 99998 99999] TEST: [    9    11    16 ... 999
85 99992 99997]

TRAIN: [    0     5     6 ... 99997 99998 99999] TEST: [    1     2     3 ... 999
89 99995 99996]

TRAIN: [    1     2     3 ... 99997 99998 99999] TEST: [    0     7    13 ... 999
80 99981 99991]

TRAIN: [    0     1     2 ... 99996 99997 99999] TEST: [    5     6    14 ... 999
86 99987 99998]

The F1 Score for all features with Logistic Regression Random Oversampling on the
5 folds :  0.72541

 The AUC for all features with Logistic Regression Random Oversampling on the 5 f
olds :  0.86925525925
```

In [38]:
```
# Logistic Regression - All Features - Random Under Sampling

from imblearn.under_sampling import RandomUnderSampler

kfld = KFold(n_splits=5, shuffle=True)
data_5_folds = []

for train_index, test_index in kfld.split(x, y):
    print("\nTRAIN:", train_index, "TEST:", test_index)
    # Geting the actual data from the index
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Saving the data into folds
    folds = [x_train, x_test, y_train, y_test]
    # Adding each of the fold data into 5 folds
    data_5_folds.append(folds)

for k in range(5, 6, 2):
    f1_5_folds = []
    auc_5_folds = []

    # Iterating through each of the folds
    for x_train, x_test, y_train, y_test in data_5_folds:
        # Applying random under-sampling
        rusum = RandomUnderSampler(random_state=10)
        x_resampled, y_resampled = rusum.fit_resample(x_train, y_train)

        # Logistic regression
        clfn_logistic = LogisticRegression(multi_class='ovr')
        clfn_logistic.fit(x_resampled, y_resampled)
        y_pred_proba_logistic = clfn_logistic.predict_proba(x_test)

        # Calculating F1 score
        f1_sc = f1_score(y_test, np.argmax(y_pred_proba_logistic, axis=1), avera
        f1_5_folds.append(f1_sc)

        # Calculating AUC
        auc = roc_auc_score(y_test, y_pred_proba_logistic, average='micro', mult
        auc_5_folds.append(auc)
```

```
print('\n The F1 Score for all features with Logistic Regression Random Undersam
print('\nThe AUC for all features with Logistic Regression Random Undersampling
```

```
TRAIN: [    1     2     3 ... 99997 99998 99999] TEST: [    0    10    13 ... 999
93 99995 99996]

TRAIN: [    0     1     2 ... 99997 99998 99999] TEST: [    5     9    17 ... 999
76 99983 99991]

TRAIN: [    0     1     4 ... 99995 99996 99998] TEST: [    2     3    11 ... 999
94 99997 99999]

TRAIN: [    0     2     3 ... 99997 99998 99999] TEST: [    1    12    15 ... 999
73 99989 99990]

TRAIN: [    0     1     2 ... 99996 99997 99999] TEST: [    4     6     7 ... 999
85 99988 99998]

 The F1 Score for all features with Logistic Regression Random Undersampling on t
he 5 folds:  0.6755699999999999

The AUC for all features with Logistic Regression Random Undersampling on the 5 f
olds:  0.85216967025
```

In [39]:
```
# Logistic Regression Using KBest for OverSampling (One feature selection method
from sklearn.feature_selection import SelectKBest, mutual_info_classif

KBest = 1
selector = SelectKBest(score_func=mutual_info_classif, k=KBest)

# Fit selector on the data
selector.fit(x, y)

# Get selected feature indices
selected_features_indices = selector.get_support(indices=True)

# Select only one feature
x_one_feature = x.iloc[:, selected_features_indices]

kfld = KFold(n_splits=5, shuffle=True)
f1_5_folds = []
auc_5_folds = []

for train_index, test_index in kfld.split(x_one_feature, y):
    x_train, x_test = x_one_feature.iloc[train_index], x_one_feature.iloc[test_i
    y_train, y_test = y[train_index], y[test_index]

    # Applying random over-sampling
    rosam = RandomOverSampler(random_state=10)
    x_resampled, y_resampled = rosam.fit_resample(x_train, y_train)

    # Logistic regression
    clfn_logistic = LogisticRegression(multi_class='ovr')
    clfn_logistic.fit(x_resampled, y_resampled)
    y_pred_proba_logistic = clfn_logistic.predict_proba(x_test)

    # Calculating F1 score
    f1_sc = f1_score(y_test, clfn_logistic.predict(x_test), average='micro')
    f1_5_folds.append(f1_sc)
```

```
    # Calculating AUC
    auc = roc_auc_score(y_test, y_pred_proba_logistic, average='micro', multi_cl
    auc_5_folds.append(auc)

print('\nThe F1 Score for One feature with Logistic Regression using SelectKBest
print('\nThe AUC for One feature with Logistic Regression using SelectKBest and
```

```
The F1 Score for One feature with Logistic Regression using SelectKBest and Overs
ampling on the 5 folds:  0.5721

The AUC for One feature with Logistic Regression using SelectKBest and Oversampli
ng on the 5 folds:  0.6422362040000001
```

In [40]:
```
# Logistic Regression Using KBest for UnderSampling (One feature selection metho

kfld = KFold(n_splits=5, shuffle=True)
f1_5_folds = []
auc_5_folds = []

for train_index, test_index in kfld.split(x_one_feature, y):
    x_train, x_test = x_one_feature.iloc[train_index], x_one_feature.iloc[test_i
    y_train, y_test = y[train_index], y[test_index]

    # Applying random under-sampling
    rusum = RandomUnderSampler(random_state=10)
    x_resampled, y_resampled = rusum.fit_resample(x_train, y_train)

    # Logistic regression
    clfn_logistic = LogisticRegression(multi_class='ovr')
    clfn_logistic.fit(x_resampled, y_resampled)
    y_pred_proba_logistic = clfn_logistic.predict_proba(x_test)

    # Calculating F1 score
    f1_sc = f1_score(y_test, clfn_logistic.predict(x_test), average='micro')
    f1_5_folds.append(f1_sc)

    # Calculating AUC
    auc = roc_auc_score(y_test, y_pred_proba_logistic, average='micro', multi_cl
    auc_5_folds.append(auc)

print('\nThe F1 Score for One feature with Logistic Regression using SelectKBest
print('\nThe AUC for One feature with Logistic Regression using SelectKBest and
```

```
The F1 Score for One feature with Logistic Regression using SelectKBest and Under
sampling on the 5 folds:  0.57248

The AUC for One feature with Logistic Regression using SelectKBest and Undersampl
ing on the 5 folds:  0.6373370575
```

In [41]:
```
# Logistic regression - Random Over Sampling (Using PCA)

from sklearn.decomposition import PCA

# Performing the PCA
pcan = PCA(n_components=10)
fit_pcan = pcan.fit_transform(x)

kfld = KFold(n_splits=5, shuffle=True)
data_5_folds = []
```

```
for train_index, test_index in kfld.split(fit_pcan, y):
    print("\nTRAIN:", train_index, "TEST:", test_index)
    # Getting the actual data from index
    x_train, x_test = fit_pcan[train_index], fit_pcan[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Saving the data into folds
    folds = [x_train, x_test, y_train, y_test]
    # Adding each of the folds data into 5 folds
    data_5_folds.append(folds)

for k in range(5, 6, 2):
    f1_5_folds = []
    auc_5_folds = []

    # Iterating through each of the folds
    for x_train, x_test, y_train, y_test in data_5_folds:

        # Applying the random over-sampling
        rosam = RandomOverSampler(random_state=10)
        x_resampled, y_resampled = rosam.fit_resample(x_train, y_train)

        # Logistic regression
        clfn_logistic = LogisticRegression(multi_class='ovr')
        clfn_logistic.fit(x_resampled, y_resampled)
        y_pred_proba_logistic = clfn_logistic.predict_proba(x_test)

        # Calculating F1 score
        f1_ovr_pcan = f1_score(y_test, np.argmax(y_pred_proba_logistic, axis=1),
        f1_5_folds.append(f1_sc)

        # Calculating AUC
        auc_ovr_pcan = roc_auc_score(y_test, y_pred_proba_logistic, average='mic
        auc_5_folds.append(auc)

print('\nThe F1 Score for PCA features with Logistic Regression Random Oversampl
print('\nThe AUC for PCA features with Logistic Regression Random Oversampling u
```

```
TRAIN: [    2     3     4 ... 99994 99995 99998] TEST: [    0     1    16 ... 999
96 99997 99999]

TRAIN: [    0     1     2 ... 99996 99997 99999] TEST: [   25    32    40 ... 999
85 99989 99998]

TRAIN: [    0     1     2 ... 99997 99998 99999] TEST: [    5     7     8 ... 999
91 99992 99994]

TRAIN: [    0     1     2 ... 99997 99998 99999] TEST: [    3     6    13 ... 999
83 99986 99995]

TRAIN: [    0     1     3 ... 99997 99998 99999] TEST: [    2     4     9 ... 999
75 99982 99984]

The F1 Score for PCA features with Logistic Regression Random Oversampling using
PCA on the 5 folds :  0.5729

The AUC for PCA features with Logistic Regression Random Oversampling using PCA o
n the 5 folds :  0.6431578275000001
```

In [42]:
```
# Logistic regression - Random Under Sampling (Using PCA)
```

```
from imblearn.under_sampling import RandomUnderSampler

# Performing the PCA
pcan = PCA(n_components=10)
fit_pcan = pcan.fit_transform(x)

kfld = KFold(n_splits=5, shuffle=True)
data_5_folds = []

for train_index, test_index in kfld.split(fit_pcan, y):
    print("\nTRAIN:", train_index, "TEST:", test_index)
    # Getting the actual data from index
    x_train, x_test = fit_pcan[train_index], fit_pcan[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Saving the data into the folds
    folds = [x_train, x_test, y_train, y_test]
    # Add each fold data into 5folds
    data_5_folds.append(folds)

for k in range(5, 6, 2):
    f1_5_folds = []
    auc_5_folds = []

    # Iterating through each of the folds
    for x_train, x_test, y_train, y_test in data_5_folds:

        # Applying the random under-sampling
        rusum = RandomUnderSampler(random_state=10)
        x_resampled, y_resampled = rusum.fit_resample(x_train, y_train)

        # Logistic regression
        clfn_logistic = LogisticRegression(multi_class='ovr')
        clfn_logistic.fit(x_resampled, y_resampled)
        y_pred_proba_logistic = clfn_logistic.predict_proba(x_test)

        # Calculating F1 score
        f1_sc = f1_score(y_test, np.argmax(y_pred_proba_logistic, axis=1), avera
        f1_5_folds.append(f1_sc)

        # Calculating AUC
        auc = roc_auc_score(y_test, y_pred_proba_logistic, average='micro', mult
        auc_5_folds.append(auc)

print('\nThe F1 Score for PCA features with Logistic Regression Random Undersamp
print('\nThe AUC for PCA features with Logistic Regression Random Undersampling
```

```
TRAIN: [    1     2     3 ... 99997 99998 99999] TEST: [    0     6    20 ... 999
75 99977 99983]

TRAIN: [    0     1     2 ... 99996 99997 99999] TEST: [    7     9    11 ... 999
94 99995 99998]

TRAIN: [    0     1     4 ... 99997 99998 99999] TEST: [    2     3    14 ... 999
88 99993 99996]

TRAIN: [    0     2     3 ... 99996 99998 99999] TEST: [    1     8    10 ... 999
86 99992 99997]

TRAIN: [    0     1     2 ... 99996 99997 99998] TEST: [    4     5    17 ... 999
89 99990 99999]

The F1 Score for PCA features with Logistic Regression Random Undersampling using
PCA on the 5 folds:  0.64587

The AUC for PCA features with Logistic Regression Random Undersampling using PCA
on the 5 folds:  0.8422585037499999
```

Conclusion:

1. **Random Oversampling vs. Random Undersampling:**
   - Random Oversampling:
     - F1 Score: 0.72541
     - AUC: 0.8693
   - Random Undersampling:
     - F1 Score: 0.67557
     - AUC: 0.8522
   - Random Oversampling shows a higher F1 Score and AUC compared to Random Undersampling, indicating better performance in this case.
2. **Using One Feature vs. All Features with SelectKBest:**
   - One Feature with Oversampling:
     - F1 Score: 0.5721
     - AUC: 0.6422
   - One Feature with Undersampling:
     - F1 Score: 0.5725
     - AUC: 0.6373
   - Selecting one feature with SelectKBest and Logistic Regression yields lower F1 Score and AUC compared to using all features.
3. **PCA Features vs. All Features:**
   - PCA Features with Random Oversampling:
     - F1 Score: 0.5729
     - AUC: 0.6432
   - PCA Features with Random Undersampling:
     - F1 Score: 0.6459
     - AUC: 0.8423
   - Using PCA features shows mixed results:

- Random Oversampling with PCA has lower F1 Score and AUC compared to using all features.
- Random Undersampling with PCA has a higher F1 Score and significantly higher AUC compared to using all features.

Overall Summary:

- Random Oversampling generally performs better than Random Undersampling in terms of F1 Score and AUC.
- Using all features with Logistic Regression yields the best performance in terms of F1 Score and AUC.
- Selecting one feature using SelectKBest and Logistic Regression shows the lowest performance.
- PCA with Random Undersampling shows a notable improvement in AUC compared to using all features.

In [ ]: