1. Loading the Dataset

Now, lets get started by importing important packages and the dataset.

1.1 Import the necessary Python modules

```
# Load python modules

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import model_selection
```

```
from IPython.display import HTML
def pretty_print_df(value_counts_):
  "Quick function to display value counts more nicely"
  display(HTML(pd.DataFrame(value_counts_).to_html()))
```

**1.2 Load Dataset **

Using pandas to load the data and explore the data both with descriptive statistics and data visualization.

```
# Load dataset from local drive (for colab notebook)
from google.colab import files
import io

uploaded = files.upload()    # Will prompt you to select file: remember to choose the right one!
data = pd.read_csv(io.BytesIO(uploaded['processed_reviews_split_surnamesABCD_minimal.csv']))
```

**1.2.1 Inspect Dataset **

**1.2.1.1 Dimensions of Dataset **

2. DATA EXPLORATION

```
# list of column titles
print(data.columns)
```

```
Index(['review_id', 'text', 'confidence_score', 'review_score',
       'acceptance_status'],
      dtype='object')
```

```
# list of column (field) data types
print(data.dtypes)
```

```
# Note: object is
```

```
review_id             object
text                  object
confidence_score     float64
review_score         float64
acceptance_status     object
dtype: object
```

*2.1 Taking a peek at the Dataset *

Python replaces empty/missing fields in the data with "NaN". **bold text**

```
# showing the first N rows in a dataframe with the function "head"
data.head(10)
# 2nd, 3rd,7thto 9th row has a missing value
```

```
# To show a random subset of the data with the function "sample":
data.sample(10)
```

2.2 The Summary statistics for numerical features

```
# The Summary statistics for numerical features
data.describe()
```

2.3 To find out how many missing values (or NaN values) there are in each feature, using Pandas `isna()` function.

```
# Number of missing values per column
data.isna().sum()
```

```
review_id              0
text                  84
confidence_score    2801
review_score           0
acceptance_status    390
dtype: int64
```

Double-click (or enter) to edit

2.3.1 Remove all rows that contain missing data

```
# remove all rows with missing data
# dropna removes all rows that contain at least one missing value
print(f'Original dataset length: {len(data)}')
data = data.dropna()
print(f'Dataset length after removing missing rows: {len(data)}')
print()
print(data[['review_id']].head(5))
data.head(5)
```

2.3.2 Remove specific rows*

We can drop specific rows by passing index labels to the drop method.

```
# remove selected column

print(data.drop("text", axis=1))
```

```
             review_id  confidence_score  review_score acceptance_status
0     iclr_review_0000               3.0           6.0            Accept
3     iclr_review_0003               3.0           5.0            Reject
4     iclr_review_0004               3.0           8.0            Accept
5     iclr_review_0005               4.0           7.0            Accept
6     iclr_review_0006               4.0           6.0            Accept
...                ...               ...           ...               ...
6112  iclr_review_6112               4.0           5.0            Reject
6113  iclr_review_6113               4.0           7.0            Accept
6114  iclr_review_6114               3.0           7.0            Accept
6116  iclr_review_6116               4.0           6.0            Reject
6117  iclr_review_6117               4.0           3.0            Reject

[3275 rows x 4 columns]
```

```
data.head(3)
```

2.4 Further exploratory using the bar chat

```
score = data.review_score.value_counts().plot(kind='bar')

fig = score.get_figure()

fig.savefig("score.png");
```

```
data.shape
```

```
(3275, 5)
```

3.1 #NEXT IS TFID VECTORISZER

Using tfidf conversion function. We discard tokens that appear in more than half the documents (max_df)

We discard tokens that appear in less than 10 documents (min_df) We only use unigrams (ngram_range)

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_df=0.5, min_df= 10, stop_words="english",
                             ngram_range= (1,1))#, sublinear_tf=True)
# fit on and apply to training data
X = vectorizer.fit_transform(data['text'])
y=data['acceptance_status']
```

```
print(X)
```

```
  (0, 4673)	0.05796956868217538
  (0, 1104)	0.046901458806378665
  (0, 3544)	0.04490611716736125
  (0, 2353)	0.05228048059329899
  (0, 1069)	0.08634685968762751
  (0, 3120)	0.025592580038584146
  (0, 2984)	0.03951896411315237
  (0, 400)	0.042025478642245734
  (0, 569)	0.0842297055364024
  (0, 3987)	0.07212293485398621
  (0, 2971)	0.04911630932884475
  (0, 1912)	0.060418328393127135
  (0, 2236)	0.055092145378295834
  (0, 170)	0.0727904844954055
  (0, 264)	0.05188742566166973
  (0, 1613)	0.09593495229082306
  (0, 698)	0.0869271238814221
  (0, 399)	0.0432714716860649
  (0, 2628)	0.05530003882884267
  (0, 2663)	0.06311574673814682
  (0, 964)	0.04967820624472587
  (0, 3616)	0.06409427456419738
  (0, 5046)	0.09400776443582887
  (0, 1612)	0.08691671273251463
  (0, 2495)	0.06356941868608881
```

```
  :	:
  (3274, 1864)	0.027013405510731777
  (3274, 3712)	0.03387379538010176
  (3274, 3324)	0.022394384472451387
  (3274, 1440)	0.05008840713602383
  (3274, 597)	0.05086621401430216
  (3274, 3544)	0.03823349970938939
  (3274, 3120)	0.021789768592572117
  (3274, 2971)	0.04181809777612115
  (3274, 5046)	0.10671881875520176
  (3274, 4063)	0.03152524320154025
  (3274, 3380)	0.03453635784863075
  (3274, 1608)	0.022129360219777867
  (3274, 2429)	0.037971367188402036
  (3274, 240)	0.03676293655320006
  (3274, 1496)	0.03697420304584078
  (3274, 773)	0.029326932782561446
  (3274, 2138)	0.04275159657572659
  (3274, 972)	0.028471418069670046
  (3274, 2967)	0.030754474093171134
  (3274, 628)	0.03408374139583719
  (3274, 1024)	0.03883631661322463
  (3274, 2275)	0.042215495697278504
  (3274, 409)	0.046964706922297976
  (3274, 1828)	0.04465829250469773
  (3274, 4531)	0.05390632806254136
```

3.2 Binary Classificaion

```
data['binary_category'] = data['acceptance_status'].factorize()[0]
y= data['binary_category']
print(y.shape)
```

```
(3275,)
```

4.1 Training the data dataset

```
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)
print(y_train.shape)
```

```
(2292,)
```

5.1. Modelling with Logistics Regression with Hyper-parameter tunning. (Using GridSearchCV)

GridSearchCV is a convenient function in scikit-learn that helps us fine-tune the hyper-parameters of our ML mod

```
#Logistic analysis Modeling
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score, f1_score
from sklearn.model_selection import GridSearchCV
```

```
# hyper-parameter tuning using in-built GridSearchCV
# pipeline is used to standardisation and also creating the normal instance of a LogisticRegression
# This may take a while: good to set the max_iter parameter as well...
param_grid=[{'C': np.logspace(-4,4,15)}, #inverse of regularization strength
           {'penalty': ['l1', 'l2']},
           {'solver': ['lbfgs', 'liblinear', 'adam']},
          {'max_iter': [10000]}] #we set this low to speed things up

lr = LogisticRegression(class_weight='balanced')



grid = GridSearchCV(estimator=lr, param_grid=param_grid, cv=10, scoring='recall', refit=True) # you can change scorin
grid = grid.fit(X_train, y_train)
print('Best estimator: {}\nWeights: {}, Intercept: {}\nBest params: {}\nScorer: {}'.format(grid.best_estimator_,
                                                                                           grid.best_estimator_.coef_
                                                                                           grid.best_estimator_.inter
                                                                                           grid.best_params_,
                                                                                           grid.scorer_))
print('Available parameters for the estimator (fine-tuning): ',lr.get_params().keys())
```

```
Best estimator: LogisticRegression(C=719.6856730011514, class_weight='balanced')
Weights: [[ 0.00835233  0.11992895  1.06972178 ...  1.28839292 -1.10583722
  -0.13483108]], Intercept: [-0.61036768]
Best params: {'C': 719.6856730011514}
Scorer: make_scorer(recall_score, average=binary)
Available parameters for the estimator (fine-tuning):  dict_keys(['C', 'class_weight', 'dual', 'fit_intercept',
```

```
lr = LogisticRegression(C=719.7, class_weight='balanced')
model=lr.fit(X_train,y_train)
```

```
print(x_test)
```

```
  (0, 2747)     0.039702727380389
  (0, 138)      0.079405454760778
  (0, 3439)     0.03883259100141582
  (0, 4925)     0.03115742707423479
  (0, 78)       0.03883259100141582
  (0, 2852)     0.035487989280714574
  (0, 1857)     0.039252107797007614
  (0, 5361)     0.037395209710420825
  (0, 601)      0.03650377530002335
  (0, 3677)     0.0350373696973332
  (0, 5438)     0.039252107797007614
  (0, 5423)     0.040718513399697766
  (0, 552)      0.079405454760778
  (0, 788)      0.033680527090455825
  (0, 1642)     0.03678664405833733
  (0, 4352)     0.035726474033534315
  (0, 1450)     0.03883259100141582
  (0, 5348)     0.03385678817829105
  (0, 1169)     0.29666653312329344
  (0, 4944)     0.033680527090455825
  (0, 3937)     0.03844015924728905
  (0, 1548)     0.03844015924728905
  (0, 1294)     0.035974695508618766
  (0, 1173)     0.03151173593385989
  (0, 3377)     0.03708331664041168
  :     :
  (982, 5040)   0.04389053339003981
  (982, 4248)   0.08153338562359493
  (982, 1367)   0.04586596779263486
  (982, 3969)   0.03526167041478821
  (982, 2991)   0.028114860033312258
  (982, 2619)   0.04344737482823121
  (982, 3510)   0.04921518339940008
```

```
  (982, 3345)	0.06226639014859081
  (982, 2462)	0.031497504206407276
  (982, 597)	0.05343697035778295
  (982, 4359)	0.09734996459025497
  (982, 1104)	0.04195051650504036
  (982, 3544)	0.04016580416365343
  (982, 264)	0.0464101621147847
  (982, 2628)	0.19785015226888694
  (982, 964)	0.044434149044565344
  (982, 5298)	0.02250287062818809
  (982, 1608)	0.02324776846510491
  (982, 3988)	0.07369402081190059
  (982, 773)	0.03080910321624866
  (982, 3000)	0.040042704769622954
  (982, 4837)	0.035134223945604025
  (982, 3386)	0.02730675752535856
  (982, 188)	0.032654774673950004
  (982, 1985)	0.03802909724956896
```

```
y_predict=lr.predict(x_test)
```

```
from sklearn.metrics import classification_report, accuracy_score

print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.70      0.68      0.69       422
           1       0.76      0.79      0.77       561

    accuracy                           0.74       983
   macro avg       0.73      0.73      0.73       983
weighted avg       0.74      0.74      0.74       983
```

```
data['acceptance_status'].value_counts()

data.columns
```

```
Index(['review_id', 'text', 'confidence_score', 'review_score',
       'acceptance_status', 'binary_category'],
      dtype='object')
```

```
#2nd Analysis
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_df=0.5, min_df= 10, stop_words="english",
                             ngram_range= (1,1))#, sublinear_tf=True)
# fit on and apply to training data
X = vectorizer.fit_transform(data['text'])
y_2=data['review_score']
```

```
x_train, x_test, y_2_train, y_2_test = train_test_split(X, y_2, test_size=0.3)
print(y_train.shape)
```

```
(2292,)
```

```
#Second model algorthm model - support vector machine
from sklearn import svm
clf = svm.SVC()
```

```
clf.fit(x_train, y_2_train)
```

```
SVC()
```

```
y_2_predict = clf.predict(x_test)
print(classification_report(y_2_test,y_2_predict))
```

```
              precision    recall  f1-score   support

        -1.0       0.00      0.00      0.00        24
         1.0       0.00      0.00      0.00         2
         2.0       0.00      0.00      0.00        23
         3.0       0.33      0.01      0.02        92
         4.0       0.31      0.24      0.27       184
         5.0       0.17      0.30      0.22       155
         6.0       0.27      0.45      0.34       233
         7.0       0.30      0.26      0.28       184
         8.0       0.25      0.02      0.03        65
         9.0       0.00      0.00      0.00        20
        10.0       0.00      0.00      0.00         1

    accuracy                           0.25       983
   macro avg       0.15      0.12      0.11       983
weighted avg       0.25      0.25      0.22       983
```

EVALUATION

```
#To wrap common procedures into functions for ease of re-usability

def evaluate_classifier(grid, X_train, y_train, X_test, y_test):
    # model evaluation for training set
    y_train_predict = grid.predict(X_train)
    print("Training SET")
    print("--------------------------------------")
    print('Accuracy: {:.3f}, Precision: {:.3f}, Recall: {:.3f}, F1 Score: {:.3f}'.format(accuracy_score(y_train, y_tr
                                                                                         precision_score(y_train, y_tr
                                                                                         recall_score(y_train, y_train
                                                                                         f1_score(y_train, y_train_pre
    print("Confusion Matrix:\n {}".format(confusion_matrix(y_train, y_train_predict)))

    # model evaluation for testing set
    y_test_predict = grid.predict(X_test)

    print("\nTesting SET")
    print("--------------------------------------")
    print('Accuracy: {:.3f}, Precision: {:.3f}, Recall: {:.3f}, F1 Score: {:.3f}'.format(accuracy_score(y_test, y_tes
                                                                                         precision_score(y_test, y_tes
                                                                                         recall_score(y_test, y_test_p
                                                                                         f1_score(y_test, y_test_predi
    print("Confusion Matrix:\n {}".format(confusion_matrix(y_test, y_test_predict)))
    return y_train_predict, y_test_predict

y_train_predict, y_test_predict = evaluate_classifier(grid, X_train, y_train, x_test, y_test)
```

```
Training SET
--------------------------------------
Accuracy: 0.993, Precision: 1.000, Recall: 0.988, F1 Score: 0.994
Confusion Matrix:
 [[ 901    0]
 [  16 1375]]
```

```
Testing SET
-------------------------------------
Accuracy: 0.487, Precision: 0.550, Recall: 0.563, F1 Score: 0.556
Confusion Matrix:
 [[163 259]
 [245 316]]
```

More Evaluation using the Roc Curve

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

def roc_classifier(grid, X_train, y_train, X_test, y_test):
    # predict probabilities
    lr_probs_train = grid.predict_proba(X_train)
    lr_probs_test = grid.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    lr_probs_train = lr_probs_train[:, 1]
    lr_probs_test = lr_probs_test[:, 1]

    print('ROC AUC (Training)={:.3f}'.format(roc_auc_score(y_train, lr_probs_train)))
    print('ROC AUC (Testing)={:.3f}'.format(roc_auc_score(y_test, lr_probs_test)))

    # compute false positive and true positive rates
    lr_fpr_train, lr_tpr_train, _ = roc_curve(y_train, lr_probs_train)
    lr_fpr_test, lr_tpr_test, _ = roc_curve(y_test, lr_probs_test)

    # plot the roc curve for the training set
    _ = plt.figure(figsize=(15, 5))
    ax1 =plt.subplot(121)
    _ = ax1.plot(lr_fpr_train, lr_tpr_train, marker='x')
    _ = ax1.plot([0,1], [0, 1], 'gray', linestyle=':', marker='')
    _ = ax1.set_title('Receiver Operating Characteristics (ROC) - Training')
    _ = ax1.set_xlabel('False Positive Rate')
    _ = ax1.set_ylabel('True Positive Rate')

    # plot the roc curve for the testing set
    ax2 = plt.subplot(122)
    _ = ax2.plot(lr_fpr_test, lr_tpr_test, marker='x')
    _ = ax2.plot([0,1], [0, 1], 'gray', linestyle=':', marker='')
    _ = ax2.set_title('Receiver Operating Characteristics (ROC) - Testing')
    _ = ax2.set_xlabel('False Positive Rate')
    _ = ax2.set_ylabel('True Positive Rate')

    return (lr_probs_train, lr_fpr_train, lr_tpr_train,
            lr_probs_test,  lr_fpr_test, lr_tpr_test)

lr_probs_train, lr_fpr_train, lr_tpr_train, lr_probs_test, lr_fpr_test, lr_tpr_test = roc_classifier(
                                                                    grid, X_train, y_train, x_test, y_test)
```

```
def exclusion(text, review_score, acceptance_status):
  value = 1
  if text == ' ' or review_score == ' ' or acceptance_status == ' ':
    value = 1
  else:
      value = 0
  return value
```

```
data['excluded'] =data[['text', 'review_score','acceptance_status']].apply(lambda X : exclusion(*X), axis = 1)
```

```
def reason(row):
  if row['excluded'] == 1:
    return 'missing_value'
  else:
    return 'N/A'
```

```
data['reason_for_exclusion'] = data.apply(lambda row: reason(row), axis=1)
```

```
data.to_csv('reason_for_exclusion.csv')
```