Import the required packages and set up the Spark session.

In [ ]:
```
# install required packages
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
%pip install -q pyspark
%pip install -q findspark
%pip install -q joblibspark
%pip install -q spacy
!python -m spacy download en_core_web_sm
```

In [ ]:
```
import findspark
findspark.init()
findspark.find()
```

Out[ ]:
```
'/usr/local/lib/python3.8/dist-packages/pyspark'
```

In [ ]:
```
from pyspark.sql import SparkSession
import pyspark;
spark = SparkSession.builder.appName('612Project').getOrCreate();
```

Note: We will be using Spark for GridSearchCV only. This appears to be the only sklearn method currently supported by Spark. The remaining code will be executed using Pandas dataframes.

Load the CSV file (n=1000 samples) containing our manual labels as the target vector

In [ ]:
```
# This notebook was originally run in Google colab due to better hardware perf
# This code should not be run locally (it will not work)

# from google.colab import files
# uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving GH-React.csv to GH-React.csv
```

In [ ]:
```
import pandas as pd

df = pd.read_csv("./GH-React.csv")

#keep only the columns we need
df = df[['title','author_association','body','Target']]
```

In [ ]:
```
print("Shape:", df.shape)
df.columns
```

```
Shape: (1000, 4)
```

Out[ ]:
```
Index(['title', 'author_association', 'body', 'Target'], dtype='object')
```

Machine Learning Pipeline

Stage 1

The preprocess() function is defined below. It takes in a String formatted as Markdown from GitHub and pre-processes it to return a new string ready for the next stages in our ML Pipeline.

In [ ]:
```
import re

def preprocess(text):
    stripped = text.lower()

    # remove all headings, bold text, and HTML comments from the Markdown text
    # These items have all been used by the React team in their issue template
    headings_pattern = r'(<=\s|^)#{1,6}(.*?)$'
    bold_pattern = r'\*\*(.+?)\*\*(?!\*)'
    comments_pattern = r'<!--((.|\n)*?)-->'
    combined_pattern = r'|'.join((headings_pattern, bold_pattern, comments_pat

    stripped = re.sub(combined_pattern, '', stripped)

    # find all URLs in the string, and then remove the final directory from ea
    # there may be useful patterns based on what URLs issues are commonly link
    url_pattern = re.compile(r'(https?://[^\s]+)')
    for url in re.findall(url_pattern, stripped):
        new_url = url.rsplit("/", 1)[0]
        stripped = stripped.replace(url, new_url)

    non_alpha_pattern = r'[^A-Za-z ]+'
    stripped = re.sub(non_alpha_pattern, '', stripped)

    return ' '.join(stripped.split())
```

In [ ]:
```
#convert body and title column to unicode, there were some issues with process
df['body'] = df['body'].astype('U')
df['title'] = df['title'].astype('U')
```

Test the preprocess function on a sample post to ensure that it works as expected:

In [ ]:
```
test = df['body'][4]
preprocess(test)
```

Out[ ]:
```
'bug or undefined behaviourdoingreactchildrentoarray reactdomcreateportalfails
withobjects are not valid as a react child found object with keys typeof key c
hildren containerinfo implementation if you meant to render a collection of ch
ildren use an array insteadnamely the following complete snippet failsjsximpor
t react from reactimport render createportal from reactdomconst renderchildren
children children reactchildrentoarraychildren return hrenders children with t
oarray childrenhconst app renderchildren namecodesandbox createportaldivrender
ed in portaldiv documentgetelementbyidportal renderchildrenrenderapp documentg
etelementbyidrootwhile the following one which wraps the portal in another ele
ment works just finejsximport react from reactimport render createportal from
reactdomconst renderchildren children children reactchildrentoarraychildren re
turn hrenders children with toarray childrenhconst app renderchildren namecode
sandbox div createportaldivrendered in portaldiv documentgetelementbyidportal
div renderchildrenrenderapp documentgetelementbyidrooti am aware that createpo
rtal is a new feature but in the best case scenario it should be possible to u
se it everywhere other valid nodes are acceptedthe same thing is happening for
reactcloneelementreactdomcreateportal its probably weird to try and clone a po
rtal but maybe we should specify in the createportal documentation that it can
not be cloned at least for now should i open a pr for thatlet me know your tho
ughtsim using react'
```

Stage 2

Split the data into training (80%) and validation(20%) sets. We will stratify based on the label since our dataset is imbalanced.

In [ ]:
```
y = df['Target']
X = df.drop(['Target'], axis=1)
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, train_size=0.8, strati

print(X_train.shape)
print(X_val.shape)
```

```
(800, 3)
(200, 3)
```

Stage 3

Create a TF-IDF features matrix using TfidfVectorizer from sklearn applied to the title and body of each issue.

We will additionally add in the feature 'author_association' from the GitHub issue, as there may be a correlation between Members/Collaborators/Contributors submitting more valid bugs/feature requests than "None" users.

While lemmatization could have been done earlier in the pre-processsing stage, it is more efficient to lemmatize at this point in a custom_tokenizer() function passed to TfidfVectorizer since tokenization is part of both processses.

First, define the tokenizer and vectorizer:

In [ ]:
```
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer

nlp = spacy.load("en_core_web_sm")

# create a custom tokenizer using the spacy document processing pipeline
def custom_tokenizer(document):
    ppd = preprocess(document)
    doc = nlp(ppd)
    return [token.lemma_ for token in doc]

tfidfvect = TfidfVectorizer(tokenizer=custom_tokenizer, ngram_range=(1, 2), mi
```

We will also use one-hot-encoding on the author-association feature

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

# use one hot encoder to transform the author_association to a feature set
ohe = OneHotEncoder()
```

Create the features matrix using a ColumnTransformer to create a pipeline with the different feature generation methods. We will use a separate vectorizer on the body and title to produce a different set of features for each. The tokens in the title may hold different importance than the same token in the body.

In [ ]:
```
from sklearn.compose import make_column_transformer
from sklearn.compose import ColumnTransformer

ct = ColumnTransformer(
    [("title", tfidfvect, "title"),
     ("body", tfidfvect, "body"),
     ("ohe", ohe, ['author_association'])]
)

ct.fit(X_train)
X_train_trans = ct.transform(X_train)
X_train_trans.shape
```

Out[ ]:
```
(800, 3544)
```

Perform an initial analysis on our model to see how our train and validation scores look (spoiler: not great)

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_trans, y_train)

X_val_trans = ct.transform(X_val)
print("Train score: {:.2f}".format(logreg.score(X_train_trans, y_train)))
print("Validation score: {:.2f}".format(logreg.score(X_val_trans, y_val)))
```

```
Train score: 0.89
Validation score: 0.57
```

Look at the features with the lowest and highest idf from the 'body' column just to see if things look reasonable. The features with the lowest idf are what we would think of as 'stop words', so this seems intuitive.

In [ ]:
```
import numpy as np
sorted_by_idf = np.argsort(ct.named_transformers_.body.idf_)
feature_names = np.array(ct.named_transformers_.body.get_feature_names_out())

print("Features with lowest idf:\n{}".format(feature_names[sorted_by_idf[:20]]
print("Features with highest idf:\n{}".format(feature_names[sorted_by_idf[-20:
```

```
Features with lowest idf:
['the' 'be' 'to' 'a' 'not' 'in' 'and' 'this' 'I' 'react' 'it' 'of' 'do'
 'use' 'version' 'that' 'for' 'with' 'component' 'have']
Features with highest idf:
['standalone' 'eject' 'start build' 'either the' 'start with' 'state I'
 'devdependencie' 'state dispatch' 'during the' 'down the' 'still be'
 'still have' 'do in' 'dispatch type' 'discussion' 'disabled' 'diff'
 'subcomponent' 'state for' 'your time']
```

TODO: See if we can visualize our text features to make sure it seems logical. Take an example from Chapter 7 in ML book.

Stage 4

Grid Search for optimizing model

Use grid search to find potentially better model parameters:

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
import numpy as np
from sklearn.model_selection import GridSearchCV

####### import joblibspark for gridsearch
from joblibspark import register_spark
from sklearn.utils import parallel_backend
register_spark() # register spark backend

pipe = make_pipeline(ct, LogisticRegression(max_iter=1000))

param_grid = {"logisticregression__C": [0.1, 0.5, 1, 2, 5],
              "columntransformer__body__ngram_range": [(1, 1), (1, 2), (1, 3
              "columntransformer__body__min_df": [1,5]
             }
```

In [ ]:
```
# WARNING: Running this cell will run GridSearch. Skip to loading the pickle f

with parallel_backend('spark',n_jobs=-1):
  grid = GridSearchCV(pipe, param_grid, cv=5)
  grid.fit(X_train, y_train)
```

In [ ]:
```
# write the grid object to a file so that it can be loaded in a different sess
import dill as pickle
pickle.dump(grid, open("grid.pkl", "wb"))
```

In [ ]:
```
# this was originally run on Google colab as their hardware is better than min
# download the results from colab to my local drive
from google.colab import files
files.download('grid.pkl')
```

In [ ]:
```
# load the grid variable back from file to continue using it's
# contents for analysis in future sessions
grid = pickle.load(open("grid.pkl", "rb"))
```

In [ ]:
```
print("Best parameters: {}".format(grid.best_params_))
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

```
Best parameters: {'columntransformer__body__min_df': 1, 'columntransformer__bo
dy__ngram_range': (1, 3), 'logisticregression__C': 0.5}
Best cross-validation score: 0.56
```

Stage 5

Use the best parameters generated by grid transform to re-train and validate our model with the X_val and y_val data we saved in stage 3:

In [ ]:
```
tfidfvect = TfidfVectorizer(tokenizer=custom_tokenizer, ngram_range=(1, 3), mi

ct = ColumnTransformer(
    [("title", tfidfvect, "title"),
     ("body", tfidfvect, "body"),
     ("ohe", ohe, ['author_association'])]
)

X_train_trans = ct.fit_transform(X_train)

logreg = LogisticRegression(C=0.5, max_iter=1000)
logreg.fit(X_train_trans, y_train)

X_val_trans = ct.transform(X_val)
print("Train score: {:.2f}".format(logreg.score(X_train_trans, y_train)))
print("Validation score: {:.2f}".format(logreg.score(X_val_trans, y_val)))
```

```
Train score: 0.87
Validation score: 0.55
```

Our model doesn't perform great, but at least it beats random chance by about 20%!

In [ ]:
```
from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier(strategy="stratified", random_state=0)
dummy_clf.fit(X, y)

print("Score based on chance: {:.2f}".format(dummy_clf.score(X, y)))
```

```
Score based on chance: 0.35
```