

(https://colab.research.google.com/github/BrianMutisyo/Lets-Go-Sixers/blob/main/Kaggle_Classification_AnkitV2.ipynb).

Hello world!

76ers TEAM MEMBERS:

1. SAM FISHER
2. ANKIT SRIVASTAVA
3. NZUKI MUTISYO

Sam's Section

In [322]: `import numpy as np
import pandas as pd`

In [323]: `train_dt = pd.read_csv("../train_data.csv")`

Bright Idea 1: We should make a small subset of the data to play with.

In [324]: `sample_train = train_dt.sample(n=500)
temp = pd.DataFrame(train_dt.drop(sample_train.index, axis = 0))
sample_validate = temp.sample(n=100)`

Bright Idea 2: We should look for missing data and patterns therein

In [325]: `train_dt.isnull().sum().sum()`

Out[325]: 0

There is no missing data! Wo Hoo!

Bright Idea 3: We should look for sparse data

```
In [326]: temp = pd.DataFrame(train_dt)
```

```
In [327]: np.count_nonzero(temp)/temp.size
```

```
Out[327]: 0.878275181683207
```

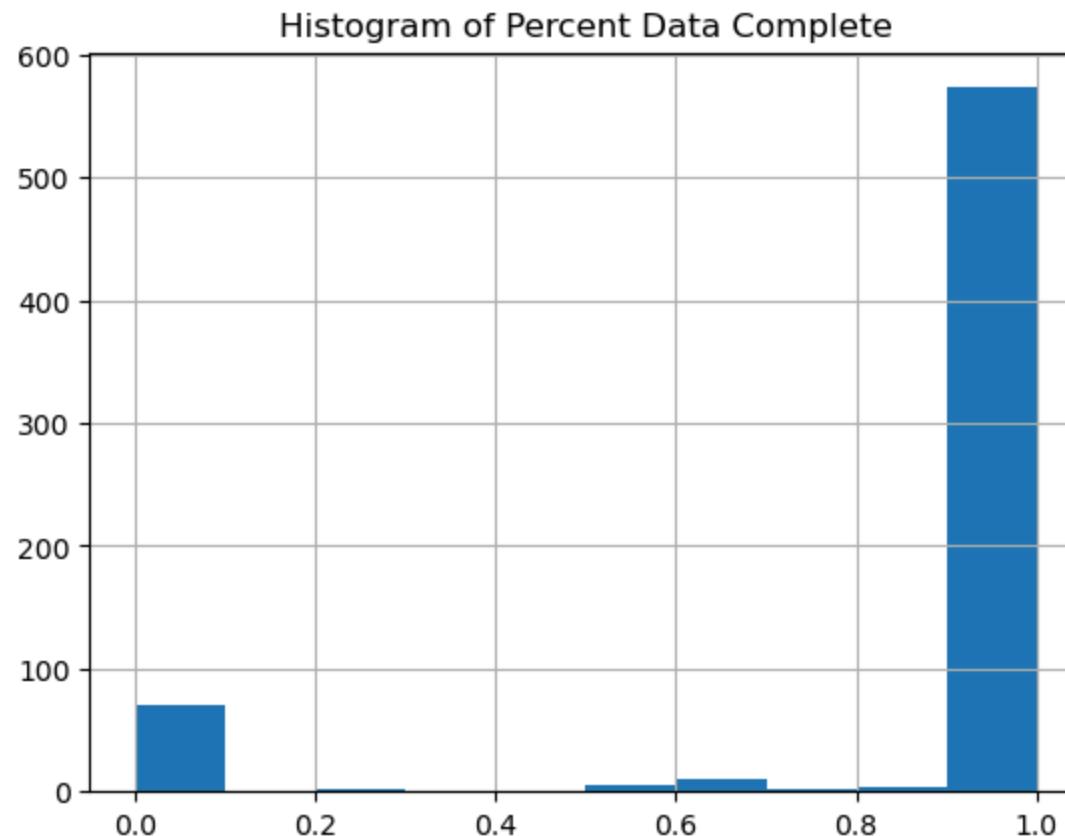
Overall it seems mostly non-zero

Let's visualize it

```
In [328]: import matplotlib.pyplot as plt
```

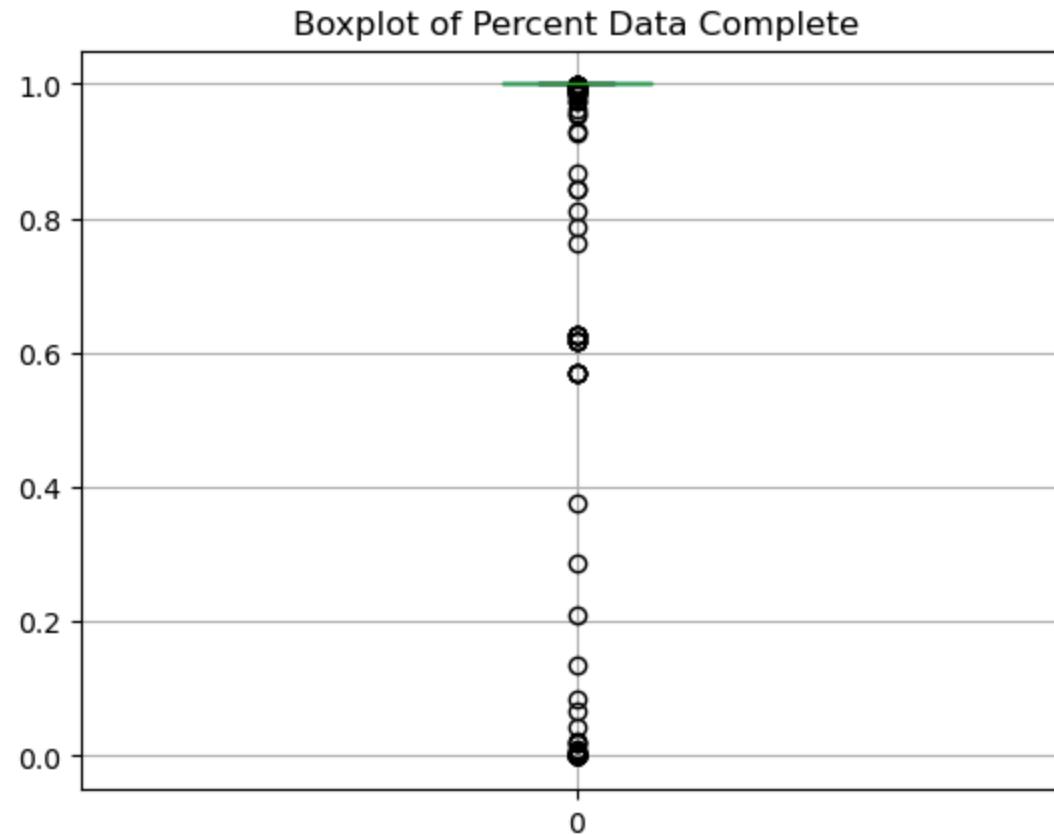
```
In [329]: pd.DataFrame([np.count_nonzero(temp[i])/temp.iloc[:,1].size for i in temp.columns]).hist(bins=10)  
plt.title("Histogram of Percent Data Complete")
```

```
Out[329]: Text(0.5, 1.0, 'Histogram of Percent Data Complete')
```



It looks like there are some sparse columns...

```
In [330]: pd.DataFrame([np.count_nonzero(temp[i])/temp.iloc[:,1].size for i in temp.columns]).boxplot();  
plt.title("Boxplot of Percent Data Complete");
```



Definitely less than 25% of them...

What features are they? What are their values?

```
In [331]: pd.DataFrame([np.count_nonzero(temp[i])/temp.iloc[:,1].size for i in temp.columns])
```

Out[331]:

	0
0	0.000000
1	0.000218
2	0.000218
3	0.625436
4	1.000000
...	...
665	1.000000
666	1.000000
667	1.000000
668	1.000000
669	0.844895

670 rows × 1 columns

```
In [332]: sparse_col_list = []
one_hot_col_list = []
all_zero_list = []
for i in temp.columns:
    if np.count_nonzero(temp[i])/temp.iloc[:,1].size < .1:
        sparse_col_list.append(i)
    if temp[i].value_counts().size == 2:
        one_hot_col_list.append(i)
    if temp[i].value_counts().size == 1:
        all_zero_list.append(i)

print(len(sparse_col_list))
print(len(one_hot_col_list))
print(len(all_zero_list))

#print("here are sparse col names: ", sparse_col_list)
#print("numer of sparse cols: ", len(sparse_col_list))

#print("here are sparse col names: ", sparse_col_list)
#print("numer of sparse cols: ", len(sparse_col_list))
```

```
71
19
48
```

I'm a little surprised there are 48 features with nothing in them... maybe let's get rid of them? I suppose they could be in the test data, but even so, the model couldn't stand a chance to learn something about them... Let's double check they are in fact all zero

```
In [333]: (temp[all_zero_list[:]] == 0).value_counts().value_counts()
```

```
Out[333]: count
4584      1
Name: count, dtype: int64
```

OK great! We've identified 48 features we can eliminate right off the bat.

```
In [334]: feat_engineered_train = pd.DataFrame(train_dt.drop(columns=all_zero_list))
```

```
In [335]: feat_engineered_train.columns.size == train_dt.columns.size - len(all_zero_list)
```

```
Out[335]: True
```

```
In [336]: sample_train = sample_train.drop(columns=all_zero_list)
sample_validate = sample_validate.drop(columns=all_zero_list)
```

"Bright" Idea 3.5: Check Var Types

Whoops! Sometimes you forget the obvious. We should check the variable types!

```
In [337]: print(sample_train.dtypes)
```

```
x2          int64
x3          int64
x4          int64
x5      float64
x6      float64
...
z222      float64
subject    object
phase      int64
state      object
output     int64
Length: 622, dtype: object
```

```
In [338]: sample_train.dtypes.value_counts()
```

```
Out[338]: float64    474
int64      146
object      2
Name: count, dtype: int64
```

```
In [339]: object_cols = []
for i in sample_train.columns:
    if sample_train[i].dtype == 'object':
        object_cols.append(i)

print(object_cols)
```

```
['subject', 'state']
```

```
In [340]: sample_train[object_cols]
```

```
Out[340]:
```

	subject	state
2988	I	C
3679	C	B
1807	I	B
379	D	A
272	L	B
...
1779	D	D
753	B	D
861	A	C
1346	D	B
3771	G	C

500 rows × 2 columns

Ah interesting--we can't necessarily assume that these columns are ordinal. That is it's not necessarily true that just because C comes after B in the alphabet that there is some ordered relationship between observations with state C and state B.

If we wanted to play it safe, then, to create a numeric encoding that doesn't assume ordinality, we could create "one hot" encodings.

These one hot encodings basically make it such that each value of the state and subject columns would have its own column which would be a binary indicator variable. For example, all observations would have a column for "B" state and for "B" subject.

But this could create a lot of columns! Let's see how many

```
In [341]: values = []
for i in object_cols:
    values.append(sample_train[i].value_counts().size)

print(values)
```

[11, 4]

That's fifteen columns! Not terrible, but not great. We can actually take a column away from each set to create a reference category and avoid multi-collinearity. This would give us a total of 13 columns.

It's possible that these columns are perfectly correlated so that, for example, subject B always has state A. Let's see if there are redundancies in the combinations of columns

```
In [342]: sample_train[object_cols].value_counts().shape
```

```
Out[342]: (43,)
```

Nope! I'd rather have 13 one hot columns than 41!

```
In [343]: from sklearn.preprocessing import OneHotEncoder
```

```
In [344]: drop_enc = OneHotEncoder().fit(sample_train[object_cols])
drop_enc.categories_
```

```
Out[344]: [array(['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'K', 'L', 'M'],
      dtype=object),
 array(['A', 'B', 'C', 'D'], dtype=object)]
```

```
In [345]: data = drop_enc.transform(sample_train[object_cols]).toarray()
```

```
In [346]: labels = drop_enc.get_feature_names_out()
```

```
In [347]: one_hot_object_cols = pd.DataFrame(data, columns=labels)
```

```
In [348]: print(one_hot_object_cols.index)
print(sample_train.index)
```

```
RangeIndex(start=0, stop=500, step=1)
Index([2988, 3679, 1807, 379, 272, 3243, 857, 341, 163, 3902,
       ...
       2611, 3802, 1852, 3036, 2333, 1779, 753, 861, 1346, 3771],
      dtype='int64', length=500)
```

```
In [349]: one_hot_object_cols.index = sample_train.index
```

```
In [350]: print(one_hot_object_cols.index)
print(sample_train.index)
```

```
Index([2988, 3679, 1807, 379, 272, 3243, 857, 341, 163, 3902,
       ...
       2611, 3802, 1852, 3036, 2333, 1779, 753, 861, 1346, 3771],
      dtype='int64', length=500)
Index([2988, 3679, 1807, 379, 272, 3243, 857, 341, 163, 3902,
       ...
       2611, 3802, 1852, 3036, 2333, 1779, 753, 861, 1346, 3771],
      dtype='int64', length=500)
```

```
In [351]: temp = pd.concat([sample_train, one_hot_object_cols], axis=1, join="inner")
```

```
In [352]: temp.head(3)
```

Out[352]:

	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	...	subject_G	subject_H	subject_I	subject_K	sul
2988	0	0	1	-237.763054	4.048771	-1.618950	214.828424	0.012966	0.000680	-0.000057	...	0.0	0.0	1.0	0.0	0.0
3679	0	0	0	-352.397998	2.346890	-2.382778	310.954298	0.018412	-0.000352	-0.000015	...	0.0	0.0	0.0	0.0	0.0
1807	0	0	1	-198.609350	4.174770	-1.242541	100.472941	0.007323	0.000106	0.000007	...	0.0	0.0	1.0	0.0	0.0

3 rows × 637 columns

```
In [353]: print(temp.shape)
print(sample_train.shape)
print(one_hot_object_cols.shape)
```

```
(500, 637)
(500, 622)
(500, 15)
```

```
In [354]: sample_train = temp.drop(columns=object_cols)
```

```
In [355]: print(sample_train.shape)
```

```
(500, 635)
```

Great! We now have our one hot variables. Now let's do the same with the validation data as well.

First let's double check we're dropping the same columns

```
In [356]: object_cols = []
for i in sample_validate.columns:
    if sample_validate[i].dtype == 'object':
        object_cols.append(i)

print(object_cols)
['subject', 'state']
```

Good!

```
In [357]: drop_enc = OneHotEncoder().fit(sample_validate[object_cols])
data = drop_enc.transform(sample_validate[object_cols]).toarray()
labels = drop_enc.get_feature_names_out()
one_hot_object_cols = pd.DataFrame(data, columns=labels)
one_hot_object_cols.index = sample_validate.index
temp = pd.concat([sample_validate, one_hot_object_cols], axis=1, join="inner")
sample_validate = temp.drop(columns=object_cols)

print(temp.shape)
print(sample_validate.shape)
print(one_hot_object_cols.shape)

(100, 637)
(100, 635)
(100, 15)
```

Neat!

Bright Idea 4: Reduce Dimensionality

The big idea here is that we want to reduce the dimensionality of the data. We could use PCA to get a list of principal components, and then use a certain subset of principal components to regress the data onto the output class. This would (1) represent the data with fewer dimensions and (2) give us a baseline accuracy.

However, the issue with this is that PCA reduces the dimensionality of the data with respect to the variation rather than with respect to the data's relevance for predicting a certain outcome. That means that using PCR (principal component regression) can sacrifice accuracy.

We can overcome this by keeping the feature-engineered data we have (we deleted some entirely zero columns) and using a model that has a L1 regularization method. L1 regularization, also known as Lasso, will use ML to find optimal models that reduce some variables' contribution to the model to zero. It's advantageous for us because we can set the amount of regularization to be heavy or light--removing more or less variables. It also is advantageous because this dimension reduction is happening with respect to the outcome, rather than with respect to the variance of the predictor data alone.

For our particular problem, we are going to use multinomial logistic regression because we are trying to classify into multiple discrete bins.

```
In [358]: import sklearn  
from sklearn.linear_model import LogisticRegression
```

```
In [359]: sample_train_y = np.array(sample_train['output'])  
sample_train_x = pd.DataFrame(sample_train.drop(columns='output'))
```

```
In [360]: print(sample_train_y.shape)  
print(sample_train_x.shape)  
print(sample_train.shape)  
  
(500,)  
(500, 634)  
(500, 635)
```

```
In [361]: clf = LogisticRegression(penalty = 'l1',  
                               C=0.5,  
                               random_state=1,  
                               solver = 'saga',  
                               max_iter = 25).fit(sample_train_x, sample_train_y)
```

```
/Users/bmutisyo/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning:  
  The max_iter was reached which means the coef_ did not converge  
  warnings.warn(
```

```
In [362]: clf.score(sample_train_x, sample_train_y)
```

```
Out[362]: 0.84
```

Mean accuracy 84% for training seems good--possibly too good e.g. overfitting. How many columns did it need to use? How many did we start with?

```
In [363]: col_l1_drop = clf.coef_ != 0
```

```
In [364]: sample_train_x.columns[col_l1_drop[0]].size
```

```
Out[364]: 310
```

```
In [365]: train_dt.columns.size
```

```
Out[365]: 670
```

```
In [366]: sample_train_x.columns[col_l1_drop[0]].size/train_dt.columns.size
```

```
Out[366]: 0.4626865671641791
```

It returned a model with about 60% of the variables. Neat.

Well, let's see how it does with the validation dataset eh?

```
In [367]: sample_validate_y = np.array(sample_validate['output'])
sample_validate_x = pd.DataFrame(sample_validate.drop(columns='output'))
```

```
In [368]: pred = clf.predict(sample_validate_x)
```

```
In [369]: sum(pred == sample_validate_y)/len(pred)
```

```
Out[369]: 0.87
```

Well that seems very promising. It got a validated probability of 87% on a sample with only 100 observations!

Bright idea 5: Run promising model on the full test set

First, let's train the model on the full training dataset.

```
In [370]: train_dt = pd.DataFrame(train_dt.drop(columns=all_zero_list))
drop_enc = OneHotEncoder().fit(train_dt[object_cols])
data = drop_enc.transform(train_dt[object_cols]).toarray()
labels = drop_enc.get_feature_names_out()
one_hot_object_cols = pd.DataFrame(data, columns=labels)
one_hot_object_cols.index = train_dt.index
temp = pd.concat([train_dt, one_hot_object_cols], axis=1, join="inner")
train_dt = temp.drop(columns=object_cols)
```

Now that we've feature engineered the training data as we did on the sample--let's train the full model.

```
In [371]: train_y = np.array(train_dt['output'])
train_x = pd.DataFrame(train_dt.drop(columns='output'))
```

```
In [372]: clf = LogisticRegression(penalty = 'l1',
                                 C=0.5,
                                 random_state=1,
                                 solver = 'saga',
                                 max_iter = 25).fit(train_x, train_y)
```

```
/Users/bmutisyo/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

```
In [373]: clf.score(train_x, train_y)
```

```
Out[373]: 0.8444589877835951
```

84% accuracy feels good to me! Let's hope it's not too good to be true...

Let's load the full test dataset and get ready to run it on that.

```
In [374]: test_dt = pd.read_csv("../test_data.csv")
```

Let's feature engineer the test dataset as above

```
In [375]: test_dt = pd.DataFrame(test_dt.drop(columns=all_zero_list))
test_dt[object_cols]
drop_enc = OneHotEncoder().fit(test_dt[object_cols])
data = drop_enc.transform(test_dt[object_cols]).toarray()
labels = drop_enc.get_feature_names_out()
one_hot_object_cols = pd.DataFrame(data, columns=labels)
one_hot_object_cols.index = test_dt.index
temp = pd.concat([test_dt, one_hot_object_cols], axis=1, join="inner")
test_dt = temp.drop(columns=object_cols)
```

```
In [376]: test_set = set(test_dt.columns)
```

```
In [377]: train_set = set(train_dt.columns)
```

```
In [378]: test_train_diff = test_set - train_set
```

```
In [379]: test_train_diff
```

```
Out[379]: {'subject_E', 'subject_J'}
```

We note that the test set has two subject values that the train set does not. Let's examine the values really quickly.

```
In [380]: print(test_dt['subject_E'].value_counts()/len(test_dt))
print(test_dt['subject_J'].value_counts()/len(test_dt))
```

```
subject_E
0.0    0.819861
1.0    0.180139
Name: count, dtype: float64
subject_J
0.0    0.636836
1.0    0.363164
Name: count, dtype: float64
```

We note that a lot of the data in the test set are not from the subjects in the train set--in fact over 50%.

Anticipating this difficulty, we set our one hot encoder to not create a reference group. We note that usually it's good practice to do so--dropping the first group can often mitigate multi-collinearity that comes from one hot encoding.

However, in this case, it was advantageous to keep all possible subject and states as binary variables. This means, in the training set, that an observation will never have all 0s for its subject or state binary variables. It will always have one feature with a one in it.

This lets us handle this new data in the test set by dropping it. Doing so will make subject E and subject J (when we drop them) a reference category. In the test set, it is fully possible that all subjects could be 0, unlike in the training set. This will let us run the trained algorithm on the test set, without indicating to the trained algorithm that these subject E or subject J rows are part of the training's reference group (the training set has no reference group).

```
In [381]: test_dt = pd.DataFrame(test_dt.drop(list(test_train_diff),axis=1))
```

Now with our data all set up we run the model.

```
In [382]: test_dt.columns
```

```
Out[382]: Index(['x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11',
                 ...
                 'subject_H', 'subject_I', 'subject_K', 'subject_L', 'subject_M',
                 'state_A', 'state_B', 'state_C', 'state_D', 'state_E'],
                 dtype='object', length=635)
```

```
In [383]: pred = clf.predict(test_dt)
```

Does this pass the eye test?

```
In [384]: pd.DataFrame(pred).value_counts()
```

```
Out[384]: 0
           1    1717
           0      15
Name: count, dtype: int64
```

While I was hopeful, I am cringing a bit. It seems unlikely to me that it would be so heavily in one class and not another... Well... Let's upload it to kaggle and find out!

```
In [385]: output = pd.DataFrame(pred)
```

```
In [386]: output.head()
```

```
Out[386]: 0
          0  1
          1  1
          2  1
          3  1
          4  1
```

```
In [387]: output.to_csv("output.csv", index=True)
```

Big Idea 6: the truly naive approach

After seeing this model crash and burn on the public boards (50% accuracy) it made me wonder--what's the most naive approach I could take?

```
In [388]: train_dt['output'].value_counts()/len(train_dt)
```

```
Out[388]: output
1    0.844895
0    0.155105
Name: count, dtype: float64
```

Well--if I had guessed the training set was all 1s I would be about 84% correct...is that what happened with my predictions?

```
In [389]: train_dt_x = pd.DataFrame(train_dt.drop(columns='output'))
```

```
In [390]: pred_train = clf.predict(train_dt_x)
```

```
In [391]: pd.DataFrame(pred_train).value_counts()
```

```
Out[391]: 0
1    4576
0      8
Name: count, dtype: int64
```

Haha! Well would you look at that!

Well, let's use those class notes and try to build a better model! Specifically, by oversampling or undersampling some data.

Big Idea 7: Overtrain on small data

Let's fix this by trying to train on a dataset that more evenly represents both classes

```
In [392]: train_dt['output'].value_counts()
```

```
Out[392]: output
1    3873
0    711
Name: count, dtype: int64
```

```
In [393]: train_dt_0 = pd.DataFrame(train_dt[train_dt['output']==0])
```

```
In [394]: train_dt_1 = pd.DataFrame(train_dt[train_dt['output']==1].sample(n=700))
```

```
In [395]: train_dt_even = pd.concat([train_dt_0,train_dt_1])
```

```
In [396]: train_y = np.array(train_dt_even['output'])
train_x = pd.DataFrame(train_dt_even.drop(columns='output'))
```

```
In [397]: clf = LogisticRegression(penalty = 'l1',
                                C=0.5,
                                random_state=1,
                                solver = 'saga',
                                max_iter = 25).fit(train_x, train_y)
```

```
/Users/bmutisyo/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

```
In [398]: clf.score(train_x,train_y)
```

```
Out[398]: 0.6017009213323884
```

Still not too great... I wonder if it's doing a better job with one class than the other...

```
In [399]: pred = clf.predict(train_x)
```

```
In [400]: pd.crosstab(train_y,pred)/len(train_y)
```

```
Out[400]: col_0      0      1
row_0
  0  0.335932  0.167966
  1  0.230333  0.265769
```

It's not doing as well for 1 as it is for 0. Interesting. I think we had the reverse last time. Let's boost the C...

```
In [401]: clf = LogisticRegression(penalty = 'l1',
                                 C=0.75,
                                 random_state=1,
                                 solver = 'saga',
                                 max_iter = 25).fit(train_x, train_y)
```

```
/Users/bmutisyo/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

```
In [402]: clf.score(train_x,train_y)
```

```
Out[402]: 0.6017009213323884
```

Not any different at all. Maybe more iterations...

```
In [403]: clf = LogisticRegression(penalty = 'l1',
                                 C=0.75,
                                 random_state=1,
                                 solver = 'saga',
                                 max_iter = 50).fit(train_x, train_y)
```

```
/Users/bmutisyo/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

```
In [404]: clf.score(train_x,train_y)
```

```
Out[404]: 0.6059532246633593
```

Little bit better, not too much better though...

```
In [405]: pd.crosstab(train_y,pred)/len(train_y)
```

```
Out[405]: col_0      0      1
row_0
0  0.335932  0.167966
1  0.230333  0.265769
```

Still better for 0 than for 1... Well, let's run it and submit some predictions.

```
In [406]: output = pd.DataFrame(clf.predict(test_dt))
```

```
In [407]: output.to_csv("output.csv", index=True)
```

Ankit's Section

Loading the training and test data

```
In [ ]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')
```

Remove output column in training data and assign it as the label to predict

```
In [ ]: # Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']
```

```
In [ ]: # Define categorical columns to be encoded
categorical_columns = ['subject', 'phase', 'state']
```

```
In [ ]: # One-hot encoding for categorical columns
onehot_encoder = ColumnTransformer([
    ('onehot', OneHotEncoder(handle_unknown='ignore'), categorical_columns)
], remainder='passthrough')
```

```
In [ ]: # Transform the train and test data
X_encoded = onehot_encoder.fit_transform(X)
test_data_encoded = onehot_encoder.transform(test_data)
```

```
In [ ]: # Split the data for validation  
X_train, X_val, y_train, y_val = train_test_split(X_encoded, y, test_size=0.2, random_state=42)
```

Training a random forest classifier

```
In [ ]: # Initialize and train the Random Forest model  
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)  
rf_model.fit(X_train, y_train)
```

```
Out[7]: RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [ ]: # Validate the model  
y_val_pred = rf_model.predict(X_val)  
validation_accuracy = accuracy_score(y_val, y_val_pred)  
print(f"Validation Accuracy: {validation_accuracy:.4f}")
```

Validation Accuracy: 0.8571

```
In [ ]: # Predict on the test data  
test_predictions = rf_model.predict(test_data_encoded)
```

```
In [ ]: # Format the predictions for submission  
submission_df = pd.DataFrame({  
    'id': test_data.index,  
    'output': test_predictions  
})
```

```
In [ ]: # Save the predictions to a CSV file  
submission_df.to_csv('submission.csv', index=False)
```

```
In [ ]: print("Predictions saved to submission.csv")
```

Predictions saved to submission.csv

This results in test accuracy in Kaggle of 54%

Trying below Enhancements

- Scaling: The numeric columns are scaled using StandardScaler to help the model handle numeric values more efficiently.

- Hyperparameter Tuning: GridSearchCV is used to find the best combination of Random Forest hyperparameters.
- Cross-Validation: Instead of using a single validation split, k-fold cross-validation (with cv=5) gives a more robust score.


```
In [ ]: import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import pandas as pd

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns

# Preprocessing: One-hot encoding for categorical columns, scaling for numeric columns
preprocessor = ColumnTransformer([
    ('onehot', OneHotEncoder(handle_unknown='ignore'), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Define XGBoost model with GPU support
xgb_model = xgb.XGBClassifier(tree_method='gpu_hist', use_label_encoder=False, eval_metric='logloss')

# Define a pipeline: preprocessing followed by XGBoost
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', xgb_model)
])

# Randomized Search Parameters
param_distributions = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [3, 6, 9],
    'classifier__learning_rate': [0.01, 0.1, 0.2]
}

# RandomizedSearchCV with parallelization and GPU-accelerated XGBoost
random_search = RandomizedSearchCV(
    pipeline, param_distributions, n_iter=10, scoring='accuracy', cv=5, random_state=42, n_jobs=-1
)

# Fit the model
```

```

random_search.fit(X, y)

# Best parameters and score
print(f"Best Parameters: {random_search.best_params_}")
print(f"Best Cross-validation Score: {random_search.best_score_.4f}")

# Predict on the test data
test_data_encoded = random_search.best_estimator_.named_steps['preprocessor'].transform(test_data)
test_predictions = random_search.best_estimator_.named_steps['classifier'].predict(test_data_encoded)

# Format the predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})

# Save the predictions to a CSV file
submission_df.to_csv('submission_gpu.csv', index=False)

print("Predictions saved to submission_gpu.csv")

```

```

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [18:04:55] WARNING: /workspace/src/common/error_msg.cc:27: The tree method `gpu_hist` is deprecated since 2.0.0. To use GPU training, set the `device` parameter to CUDA instead.

```

E.g. `tree_method = "hist", device = "cuda"`

```

warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [18:04:55] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

```

```

warnings.warn(smsg, UserWarning)

Best Parameters: {'classifier__n_estimators': 200, 'classifier__max_depth': 9, 'classifier__learning_rate': 0.1}
Best Cross-validation Score: 0.8656
Predictions saved to submission_gpu.csv

```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [18:04:58] WARNING: /workspace/src/common/error_msg.cc:27: The tree method `gpu_hist` is deprecated since 2.0.0. To use GPU training, set the `device` parameter to CUDA instead.
```

E.g. `tree_method = "hist", device = "cuda"`

```
warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [18:04:58] WARNING: /workspace/src/common/error_msg.cc:58: Falling back to prediction using DMatrix due to mismatched devices. This might lead to higher memory usage and slower performance. XGBoost is running on: cuda:0, while the input data is on: cpu.
```

Potential solutions:

- Use a data structure that matches the device ordinal in the booster.
- Set the device for booster before call to `inplace_predict`.

This warning will only be shown once.

```
warnings.warn(smsg, UserWarning)
```

```
In [ ]: pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.0.0-py3-none-any.whl.metadata (16 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.3-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.4.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.0.36)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna) (4.66.5)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna) (6.0.2)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.5-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0->optuna) (3.1.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0->optuna) (3.0.1)
Downloading optuna-4.0.0-py3-none-any.whl (362 kB)
  362.8/362.8 kB 7.6 MB/s eta 0:00:00
Downloading alembic-1.13.3-py3-none-any.whl (233 kB)
  233.2/233.2 kB 20.9 MB/s eta 0:00:00
Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Downloading Mako-1.3.5-py3-none-any.whl (78 kB)
  78.6/78.6 kB 7.4 MB/s eta 0:00:00
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.5 alembic-1.13.3 colorlog-6.8.2 optuna-4.0.0
```



```
In [ ]: import optuna
import xgboost as xgb
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
import pandas as pd

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns

# Preprocessing: One-hot encoding for categorical columns, scaling for numeric columns
preprocessor = ColumnTransformer([
    ('onehot', OneHotEncoder(handle_unknown='ignore'), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

X_train = preprocessor.fit_transform(X_train)
X_valid = preprocessor.transform(X_valid)

# Define the objective function for Optuna
def objective(trial):
    param = {
        'tree_method': 'hist', # Faster on GPU
        'device': 'cuda', # GPU support
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'max_depth': trial.suggest_int('max_depth', 3, 15),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.3),
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'lambda': trial.suggest_loguniform('lambda', 1e-3, 10.0),
        'alpha': trial.suggest_loguniform('alpha', 1e-3, 10.0)
    }

# Train XGBoost model
```

```
model = xgb.XGBClassifier(**param)
model.fit(X_train, y_train)

# Validate the model
score = model.score(X_valid, y_valid)
return score

# Create a study and optimize
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

# Best hyperparameters
print("Best Hyperparameters:", study.best_params)

# Train on full data using best hyperparameters
best_model = xgb.XGBClassifier(**study.best_params)
best_model.fit(X_train, y_train)

# Predict on the test data
test_data_encoded = preprocessor.transform(test_data)
test_predictions = best_model.predict(test_data_encoded)

# Format the predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})

# Save the predictions to a CSV file
submission_df.to_csv('submission_optuna.csv', index=False)

print("Predictions saved to submission_optuna.csv")
```

```
[I 2024-10-19 18:09:19,371] A new study created in memory with name: no-name-0cd94eea-f786-44ac-b931-ca99  
8f26d2ef  
<ipython-input-3-aa0f503babcb>:40: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This  
feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. (https://github.com/optuna/optuna/releases/tag/v3.0.0) Use suggest_float(..., log=True) instead.  
    'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.3),  
<ipython-input-3-aa0f503babcb>:43: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This  
feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. (https://github.com/optuna/optuna/releases/tag/v3.0.0) Use suggest_float(..., log=True) instead.  
    'lambda': trial.suggest_loguniform('lambda', 1e-3, 10.0),  
<ipython-input-3-aa0f503babcb>:44: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This  
feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. (https://github.com/optuna/optuna/releases/tag/v3.0.0) Use suggest_float(..., log=True) instead.  
    'alpha': trial.suggest_loguniform('alpha', 1e-3, 10.0)  
[I 2024-10-19 18:09:26,699] Trial 0 finished with value: 0.8538713195201745 and parameters: {'n_estimators': 545, 'max_depth': 10, 'learning_rate': 0.05999025028123221, 'subsample': 0.6527704946107059, 'colsample_bytree': 0.5053480249649472, 'lambda': 0.03958551551372708, 'alpha': 0.7714255394145872}. Best is tria  
l 0 with value: 0.8538713195201745.  
<ipython-input-3-aa0f503babcb>:40: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This  
feature will be removed in v6.0.0. See https://github.com/optuna/optuna/releases/tag/v3.0.0. (https://github.com/optuna/optuna/releases/tag/v3.0.0) Use suggest_float(..., log=True) instead.
```

```
In [ ]: pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlibb->catboost) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlibb->catboost) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (9.0.0)
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
   98.7/98.7 MB 22.0 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

```
In [ ]: pip install category-encoders
```

```
Collecting category-encoders
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category-encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category-encoders) (1.5.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category-encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category-encoders) (0.14.4)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category-encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category-encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category-encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category-encoders) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category-encoders) (2024.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category-encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category-encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category-encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category-encoders) (24.1)
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl (82 kB)
```

82.0

Installing collected packages: category-encoder
Successfully installed category-encoders-2.6.4

- Memory Management: The code optimizes the data types and uses a subset of the training data to manage memory consumption effectively.
 - Preprocessing: It includes target encoding for categorical features and scales numeric features.
 - Polynomial Features: Polynomial features are applied with interactions only, helping to reduce dimensionality.
 - Optuna Integration: It includes a hyperparameter tuning setup using Optuna for the XGBoost model.
 - Stacking Classifier: It combines predictions from XGBoost, LightGBM, and CatBoost for better performance.

```
In [ ]: pip install --upgrade xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
```

```
In [ ]: gc.collect()
```

```
Out[18]: 1096
```

```
In [ ]: gc.collect()
```

```
Out[22]: 111346
```

```
In [ ]: import warnings
```

```
# Suppress warnings
warnings.filterwarnings("ignore")
```



```
In [ ]: import optuna
import xgboost as xgb
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingClassifier
from catboost import CatBoostClassifier
import pandas as pd
import matplotlib.pyplot as plt
import category_encoders as ce
import time
import gc

# Function to monitor time
class MonitorTime:
    def __init__(self, step):
        self.step = step

    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        end_time = time.time()
        print(f"{self.step} took {end_time - self.start_time:.2f} seconds")

# Load the data
with MonitorTime("Loading data"):
    train_data = pd.read_csv('train_data.csv') # Use full data for training
    test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
with MonitorTime("Splitting data"):
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
def create_preprocessor():
    with MonitorTime("Creating preprocessor"):
        preprocessors = ColumnTransformer([
            ('cat', OneHotEncoder(), categorical_columns),
            ('num', StandardScaler(), numeric_columns)
        ])
        return preprocessors
```

```
'target_encode', ce.TargetEncoder(cols=categorical_columns), categorical_columns),
('scaler', StandardScaler(), numeric_columns)
])
return preprocessor

preprocessor = create_preprocessor()

# Fit and transform the training data with target encoding
with MonitorTime("Applying target encoding"):
    X_train_transformed = preprocessor.fit_transform(X_train, y_train)
    X_valid_transformed = preprocessor.transform(X_valid)

def objective(trial):
    param = {
        'tree_method': 'gpu_hist',
        'device': 'gpu',
        'n_estimators': trial.suggest_int('n_estimators', 100, 1500), # Wider range
        'max_depth': trial.suggest_int('max_depth', 3, 15), # Increased max depth
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.3, log=True), # Broader range
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'lambda': trial.suggest_float('lambda', 1e-5, 10.0, log=True), # Broader range
        'alpha': trial.suggest_float('alpha', 1e-5, 10.0, log=True) # Broader range
    }

    model = xgb.XGBClassifier(**param)
    model.fit(X_train_transformed, y_train)
    return model.score(X_valid_transformed, y_valid)

# Create Optuna study for hyperparameter tuning
with MonitorTime("Running Optuna study"):
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=20) # Limit trials to manage memory

# Display best hyperparameters
print("Best Hyperparameters:", study.best_params)

# Train best XGBoost model
xgb_model = xgb.XGBClassifier(**study.best_params)
with MonitorTime("Training final XGBoost model"):
    xgb_model.fit(X_train_transformed, y_train)

# Feature Importance Plot
with MonitorTime("Plotting feature importance"):
    xgb.plot_importance(xgb_model)
    plt.show()

# Prepare test data for prediction
```

```

with MonitorTime("Preparing test data"):
    test_data_encoded = preprocessor.transform(test_data)

# CatBoost Classifier
catboost_model = CatBoostClassifier(task_type='GPU', verbose=0) # CatBoost with GPU

# Stacking model
stacked_model = StackingClassifier(
    estimators=[('xgb', xgb_model), ('catboost', catboost_model)],
    final_estimator=xgb.XGBClassifier() # Using XGBoost as the final estimator
)

# Train and predict using stacking
with MonitorTime("Training stacking model"):
    stacked_model.fit(X_train_transformed, y_train)

with MonitorTime("Making predictions"):
    test_predictions = stacked_model.predict(test_data_encoded)

# Save predictions for submission
with MonitorTime("Saving predictions"):
    submission_df = pd.DataFrame({
        'id': test_data.index,
        'output': test_predictions
    })
    submission_df.to_csv('submission_stacked.csv', index=False)

print("Predictions saved to submission_stacked2.csv")

# Clean up memory
gc.collect()

```

[I 2024-10-19 19:18:08,831] A new study created in memory with name: no-name-34f5de8d-b518-4abc-b92f-f7a5886c8a75

Loading data took 0.54 seconds
Splitting data took 0.01 seconds
Creating preprocessor took 0.00 seconds
Applying target encoding took 0.10 seconds

[I 2024-10-19 19:18:11,160] Trial 0 finished with value: 0.8538713195201745 and parameters: {'n_estimators': 776, 'max_depth': 3, 'learning_rate': 0.04353396078675053, 'subsample': 0.5039884802499481, 'colsample_bytree': 0.8838428159709926, 'lambda': 0.023868234768160555, 'alpha': 0.087621776680293}. Best is trial 0 with value: 0.8538713195201745.

[I 2024-10-19 19:18:17,406] Trial 1 finished with value: 0.8407851690294439 and parameters: {'n_estimators': 319, 'max_depth': 9, 'learning_rate': 0.002831956461430996, 'subsample': 0.7380920520451759, 'colsample_bytree': 0.6645893938266938, 'lambda': 0.5319219126984857, 'alpha': 5.382835728697154}. Best is trial 0 with value: 0.8538713195201745.

[I 2024-10-19 19:18:18,717] Trial 2 finished with value: 0.8386041439476554 and parameters: {'n_estimators': 498, 'max_depth': 3, 'learning_rate': 0.0015340798339090504, 'subsample': 0.626835598416845, 'colsample_bytree': 0.7391119959318266, 'lambda': 0.0003108046901624853, 'alpha': 0.6936008876221961}. Best is trial 0 with value: 0.8538713195201745.

[I 2024-10-19 19:18:27,285] Trial 3 finished with value: 0.8571428571428571 and parameters: {'n_estimators': 1130, 'max_depth': 14, 'learning_rate': 0.16946453772074938, 'subsample': 0.706944220171168, 'colsample_bytree': 0.6314207710790593, 'lambda': 6.635022018091383e-05, 'alpha': 0.15831752853519643}. Best is trial 3 with value: 0.8571428571428571.

[I 2024-10-19 19:18:30,783] Trial 4 finished with value: 0.8593238822246456 and parameters: {'n_estimators': 916, 'max_depth': 7, 'learning_rate': 0.2642041580451899, 'subsample': 0.7702423266813991, 'colsample_bytree': 0.8320456718157465, 'lambda': 0.006101016097933144, 'alpha': 0.006069020160319749}. Best is trial 4 with value: 0.8593238822246456.

[I 2024-10-19 19:18:35,739] Trial 5 finished with value: 0.8582333696837514 and parameters: {'n_estimators': 452, 'max_depth': 15, 'learning_rate': 0.11196547585722057, 'subsample': 0.6773065113269046, 'colsample_bytree': 0.6236083180229823, 'lambda': 0.25718040049102564, 'alpha': 0.01011232839940664}. Best is trial 4 with value: 0.8593238822246456.

[I 2024-10-19 19:18:40,116] Trial 6 finished with value: 0.8582333696837514 and parameters: {'n_estimators': 319, 'max_depth': 8, 'learning_rate': 0.037925150485235086, 'subsample': 0.6328760500811155, 'colsample_bytree': 0.7457712023630055, 'lambda': 0.41409839157986095, 'alpha': 0.008770402709084316}. Best is trial 4 with value: 0.8593238822246456.

[I 2024-10-19 19:19:11,897] Trial 7 finished with value: 0.8560523446019629 and parameters: {'n_estimators': 791, 'max_depth': 14, 'learning_rate': 0.004141452202459492, 'subsample': 0.9753281630681238, 'colsample_bytree': 0.8212144272687903, 'lambda': 8.658803044229726, 'alpha': 1.3308855961322703}. Best is trial 4 with value: 0.8593238822246456.

[I 2024-10-19 19:19:13,602] Trial 8 finished with value: 0.8538713195201745 and parameters: {'n_estimators': 518, 'max_depth': 4, 'learning_rate': 0.031487803287417536, 'subsample': 0.5608290074949515, 'colsample_bytree': 0.5104353776114773, 'lambda': 0.00013942953116310398, 'alpha': 0.008697060278327864}. Best is trial 4 with value: 0.8593238822246456.

[I 2024-10-19 19:19:23,717] Trial 9 finished with value: 0.8538713195201745 and parameters: {'n_estimators': 903, 'max_depth': 12, 'learning_rate': 0.029298984979500894, 'subsample': 0.6664395175700621, 'colsample_bytree': 0.6993049184478559, 'lambda': 1.1258842560053722e-05, 'alpha': 7.804785634854904}. Best is trial 4 with value: 0.8593238822246456.

[I 2024-10-19 19:19:36,853] Trial 10 finished with value: 0.8680479825517994 and parameters: {'n_estimators': 1409, 'max_depth': 6, 'learning_rate': 0.010271183666581949, 'subsample': 0.8583749136186822, 'colsample_bytree': 0.9988309346665843, 'lambda': 0.003062102714269216, 'alpha': 5.338599128652177e-05}. Best is trial 10 with value: 0.8680479825517994.

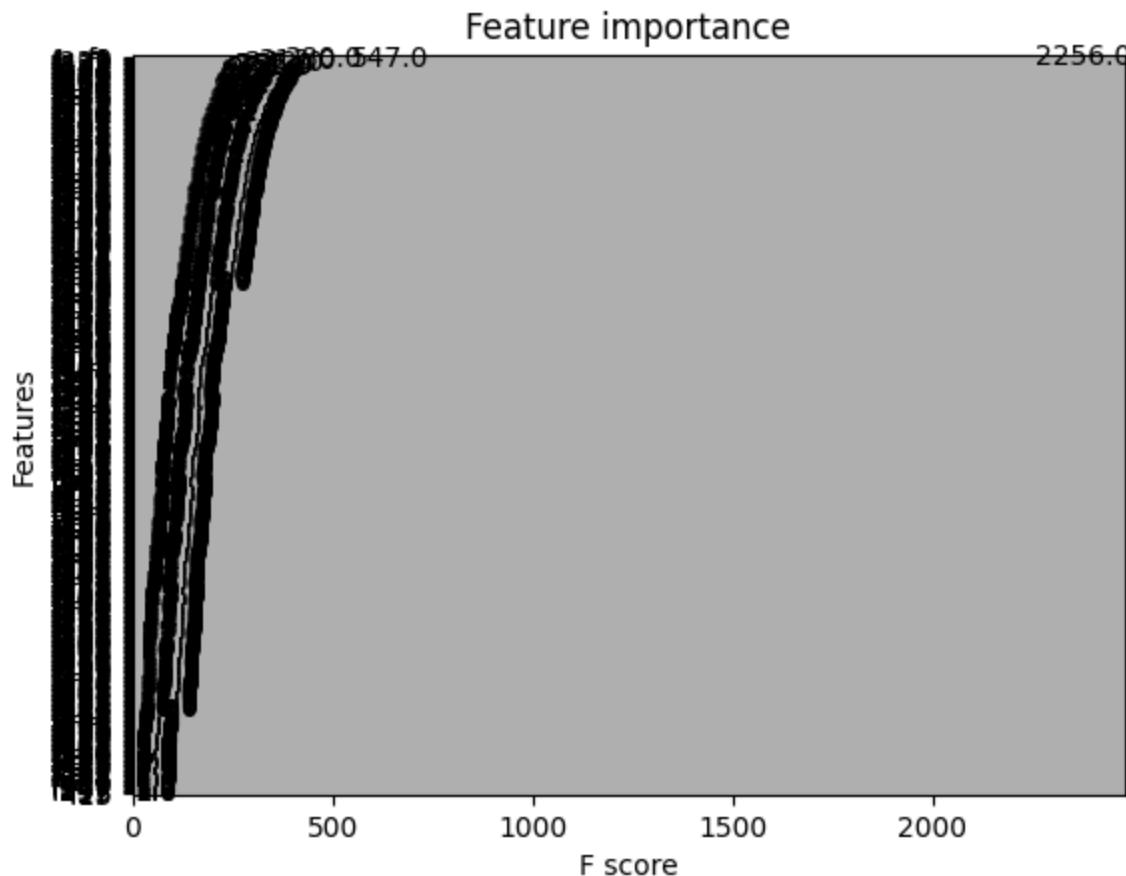
[I 2024-10-19 19:19:49,858] Trial 11 finished with value: 0.8658669574700109 and parameters: {'n_estimators': 1489, 'max_depth': 6, 'learning_rate': 0.010212339375801117, 'subsample': 0.8605171970399402, 'colsample_bytree': 0.9983318414930633, 'lambda': 0.004509005144692937, 'alpha': 1.7480368513390988e-05}. Best is tr

ial 10 with value: 0.8680479825517994.
[I 2024-10-19 19:20:03,781] Trial 12 finished with value: 0.8702290076335878 and parameters: {'n_estimators': 1462, 'max_depth': 6, 'learning_rate': 0.008252647239585268, 'subsample': 0.8798353150886109, 'colsample_bytree': 0.9915078582609945, 'lambda': 0.0021412922470020242, 'alpha': 1.1851271361097405e-05}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:20:13,386] Trial 13 finished with value: 0.8636859323882224 and parameters: {'n_estimators': 1492, 'max_depth': 5, 'learning_rate': 0.008962531395197029, 'subsample': 0.8942799428503623, 'colsample_bytree': 0.9983120312430457, 'lambda': 0.0012101036094909787, 'alpha': 1.0458550460983804e-05}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:20:42,617] Trial 14 finished with value: 0.8658669574700109 and parameters: {'n_estimators': 1240, 'max_depth': 11, 'learning_rate': 0.007379923151464704, 'subsample': 0.8474279167639371, 'colsample_bytree': 0.9273789015114395, 'lambda': 0.028545419931855838, 'alpha': 0.00022186306770208153}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:20:53,584] Trial 15 finished with value: 0.8691384950926936 and parameters: {'n_estimators': 1286, 'max_depth': 6, 'learning_rate': 0.016826913215230756, 'subsample': 0.9919875593456059, 'colsample_bytree': 0.932584269319184, 'lambda': 0.0008440235900511421, 'alpha': 0.00013271710692551938}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:21:04,754] Trial 16 finished with value: 0.8636859323882224 and parameters: {'n_estimators': 1213, 'max_depth': 10, 'learning_rate': 0.07664684828016506, 'subsample': 0.9816721447705694, 'colsample_bytree': 0.9282701194894479, 'lambda': 0.0009976617073579348, 'alpha': 0.0003531850026053337}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:21:20,897] Trial 17 finished with value: 0.8658669574700109 and parameters: {'n_estimators': 1322, 'max_depth': 8, 'learning_rate': 0.015829346886564143, 'subsample': 0.9335041348584505, 'colsample_bytree': 0.9179194178132231, 'lambda': 3.173352657302952e-05, 'alpha': 0.0005989378631690782}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:21:27,859] Trial 18 finished with value: 0.8702290076335878 and parameters: {'n_estimators': 1062, 'max_depth': 5, 'learning_rate': 0.0037558713574506503, 'subsample': 0.9991764984580919, 'colsample_bytree': 0.8342562597640455, 'lambda': 0.0003643975492789433, 'alpha': 7.678984949143699e-05}. Best is trial 12 with value: 0.8702290076335878.
[I 2024-10-19 19:21:34,799] Trial 19 finished with value: 0.8538713195201745 and parameters: {'n_estimators': 1053, 'max_depth': 5, 'learning_rate': 0.0011197845750576193, 'subsample': 0.8043641621660212, 'colsample_bytree': 0.8016958452051532, 'lambda': 0.04064499454695009, 'alpha': 0.001780606708364923}. Best is trial 12 with value: 0.8702290076335878.

Running Optuna study took 205.97 seconds

Best Hyperparameters: {'n_estimators': 1462, 'max_depth': 6, 'learning_rate': 0.008252647239585268, 'subsample': 0.8798353150886109, 'colsample_bytree': 0.9915078582609945, 'lambda': 0.0021412922470020242, 'alpha': 1.1851271361097405e-05}

Training final XGBoost model took 55.63 seconds



Plotting feature importance took 4.16 seconds

Preparing test data took 0.03 seconds

Warning: less than 75% GPU memory available for training. Free: 5649.0625 Total: 15102.0625

Warning: less than 75% GPU memory available for training. Free: 5649.0625 Total: 15102.0625

Warning: less than 75% GPU memory available for training. Free: 5649.0625 Total: 15102.0625

Warning: less than 75% GPU memory available for training. Free: 5649.0625 Total: 15102.0625

Warning: less than 75% GPU memory available for training. Free: 5649.0625 Total: 15102.0625

Warning: less than 75% GPU memory available for training. Free: 5649.0625 Total: 15102.0625

Training stacking model took 577.64 seconds

Making predictions took 0.05 seconds

Saving predictions took 0.00 seconds

Predictions saved to submission_stacked2.csv

Out[27]: 119706

The results of xgboost and other enhancement do not very different test results than in past.

As training data has imbalance with label 0 being minority class, trying SMOTE to mitigate data imbalance

```
In [ ]: pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.12.4)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.5.0)
```



```
In [ ]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from category_encoders import TargetEncoder

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
preprocessor = ColumnTransformer([
    ('target_encode', TargetEncoder(cols=categorical_columns), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Apply preprocessing to training data
X_train_transformed = preprocessor.fit_transform(X_train, y_train) # Pass y_train here
X_valid_transformed = preprocessor.transform(X_valid)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# Train XGBoost model with balanced data and specified hyperparameters
model = xgb.XGBClassifier(
    n_estimators=1462,
    max_depth=6,
```

```

learning_rate=0.008252647239585268,
subsample=0.8798353150886109,
colsample_bytree=0.9915078582609945,
reg_lambda=0.0021412922470020242, # Fixed from lambda to reg_lambda
alpha=1.1851271361097405e-05,
scale_pos_weight=1,
use_label_encoder=False,
eval_metric='mlogloss' # Set an appropriate evaluation metric
)

model.fit(X_train_balanced, y_train_balanced)

# Validate the model
y_pred = model.predict(X_valid_transformed)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred))
print("Classification Report:\n", classification_report(y_valid, y_pred))

# Prepare test data for prediction
test_data_transformed = preprocessor.transform(test_data)
test_predictions = model.predict(test_data_transformed)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})
submission_df.to_csv('submission_xgb_smote.csv', index=False)

print("Predictions saved to submission_xgb_smote.csv")

```

Validation Accuracy: 0.8669574700109052

Classification Report:

	precision	recall	f1-score	support
0	0.65	0.37	0.47	148
1	0.89	0.96	0.92	769
accuracy			0.87	917
macro avg	0.77	0.67	0.70	917
weighted avg	0.85	0.87	0.85	917

Predictions saved to submission_xgb_smote.csv

```
In [ ]: import gc  
gc.collect()
```

```
Out[1]: 54
```

This improves test accuracy from 54% to 61%. So this is a good direction.

Trying neural networks to see if that helps improve test accuracy

```
In [ ]: pip install torch torchvision torchaudio
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.4.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.19.1+cu121)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.4.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.6.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (10.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (3.0.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```



```
In [ ]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Load your data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Check for non-numeric columns
print(train_data.dtypes)

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Define categorical and numeric columns
categorical_columns = X.select_dtypes(include=['object', 'category']).columns.tolist()
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Create preprocessing pipeline
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_columns),
    ('cat', OneHotEncoder(), categorical_columns)
])

# Apply the preprocessing to your dataset
X_transformed = preprocessor.fit_transform(X)

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X_transformed, y, test_size=0.2, random_state=42)

# Convert the data into tensors for PyTorch
X_train_tensor = torch.tensor(X_train).float()
y_train_tensor = torch.tensor(y_train.values).long()

X_valid_tensor = torch.tensor(X_valid).float()
y_valid_tensor = torch.tensor(y_valid.values).long()

# Create DataLoader for PyTorch
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
valid_dataset = TensorDataset(X_valid_tensor, y_valid_tensor)
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=32)

# Define the neural network
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Initialize the model, loss function, and optimizer
input_size = X_train.shape[1]
hidden_size = 64
num_classes = len(y.unique()) # Number of output classes
model = SimpleNN(input_size, hidden_size, num_classes)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 20
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader)}')

# Evaluation on validation data
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in valid_loader:
```

```
outputs = model(inputs)
_, predicted = torch.max(outputs.data, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()

print(f'Validation Accuracy: {100 * correct / total:.2f}%')
```

```
x1          int64
x2          int64
x3          int64
x4          int64
x5          float64
...
z222        float64
subject     object
phase        int64
state        object
output       int64
Length: 670, dtype: object
Epoch [1/20], Loss: 0.4246436735858088
Epoch [2/20], Loss: 0.3701604314472364
Epoch [3/20], Loss: 0.33908496771169744
Epoch [4/20], Loss: 0.31109429792217586
Epoch [5/20], Loss: 0.29127054719821266
Epoch [6/20], Loss: 0.2687688674615777
Epoch [7/20], Loss: 0.24375930078651595
Epoch [8/20], Loss: 0.22506670653820038
Epoch [9/20], Loss: 0.20976838012752325
Epoch [10/20], Loss: 0.2032266502795012
Epoch [11/20], Loss: 0.17374949678778648
Epoch [12/20], Loss: 0.15724096431032472
Epoch [13/20], Loss: 0.15463693608408388
Epoch [14/20], Loss: 0.14880875860867293
Epoch [15/20], Loss: 0.1248208956549997
Epoch [16/20], Loss: 0.11637893782361694
Epoch [17/20], Loss: 0.1089136201078477
Epoch [18/20], Loss: 0.09965925351109194
Epoch [19/20], Loss: 0.0850598821659451
Epoch [20/20], Loss: 0.08715752006062995
Validation Accuracy: 84.84%
```



```
In [ ]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Load your data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Define categorical and numeric columns
categorical_columns = X.select_dtypes(include=['object', 'category']).columns.tolist()
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Create preprocessing pipeline
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_columns),
    ('cat', OneHotEncoder(), categorical_columns)
])

# Apply the preprocessing to your dataset
X_transformed = preprocessor.fit_transform(X)

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X_transformed, y, test_size=0.2, random_state=42)

# Convert the data into tensors for PyTorch
X_train_tensor = torch.tensor(X_train).float()
y_train_tensor = torch.tensor(y_train.values).long()

X_valid_tensor = torch.tensor(X_valid).float()
y_valid_tensor = torch.tensor(y_valid.values).long()

# Create DataLoader for PyTorch
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
valid_dataset = TensorDataset(X_valid_tensor, y_valid_tensor)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=64)
```

```
# Define the enhanced neural network
class EnhancedNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(EnhancedNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.bn1 = nn.BatchNorm1d(hidden_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

        self.fc2 = nn.Linear(hidden_size, hidden_size // 2)
        self.bn2 = nn.BatchNorm1d(hidden_size // 2)

        self.fc3 = nn.Linear(hidden_size // 2, num_classes)

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.fc2(x)
        x = self.bn2(x)
        x = self.relu(x)

        x = self.fc3(x)
        return x

# Initialize the model, loss function, and optimizer
input_size = X_train.shape[1]
hidden_size = 128 # Increased size of the hidden layer
num_classes = len(y.unique()) # Number of output classes
model = EnhancedNN(input_size, hidden_size, num_classes)

# Use Adam optimizer with weight decay (L2 regularization)
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)
criterion = nn.CrossEntropyLoss()

# Learning rate scheduler
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)

# Training loop with improvements
num_epochs = 20
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
```

```

outputs = model(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item()

scheduler.step() # Update the learning rate

print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader)}')

# Evaluation on validation data
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in valid_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Validation Accuracy: {100 * correct / total:.2f}%')

```

Epoch [1/20], Loss: 0.46917218946177386
 Epoch [2/20], Loss: 0.4021368383847434
 Epoch [3/20], Loss: 0.3878478865171301
 Epoch [4/20], Loss: 0.3710547539180723
 Epoch [5/20], Loss: 0.3606165914699949
 Epoch [6/20], Loss: 0.3358851619835558
 Epoch [7/20], Loss: 0.32835103674181576
 Epoch [8/20], Loss: 0.3169963801729268
 Epoch [9/20], Loss: 0.3129342134142744
 Epoch [10/20], Loss: 0.29698213189840317
 Epoch [11/20], Loss: 0.28886267619914024
 Epoch [12/20], Loss: 0.2806277359867918
 Epoch [13/20], Loss: 0.2657181251922558
 Epoch [14/20], Loss: 0.2689751563914891
 Epoch [15/20], Loss: 0.2612591352442215
 Epoch [16/20], Loss: 0.25263171720093697
 Epoch [17/20], Loss: 0.24809935020989385
 Epoch [18/20], Loss: 0.2503225600411152
 Epoch [19/20], Loss: 0.24254371123067264
 Epoch [20/20], Loss: 0.24577735724120303
 Validation Accuracy: 84.19%

This did not lead to any improvement in test accuracy


```
In [ ]: # Install necessary packages
!pip install torch torchvision

import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline with handle_unknown parameter
preprocessor = ColumnTransformer([
    ('onehot', OneHotEncoder(handle_unknown='ignore'), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Transform the data
X_train_transformed = preprocessor.fit_transform(X_train)
X_valid_transformed = preprocessor.transform(X_valid)
X_test_transformed = preprocessor.transform(test_data)

# Convert the data into PyTorch tensors
X_train_tensor = torch.FloatTensor(X_train_transformed) # No need for .toarray() here
y_train_tensor = torch.LongTensor(y_train.values)
X_valid_tensor = torch.FloatTensor(X_valid_transformed)
y_valid_tensor = torch.LongTensor(y_valid.values)
X_test_tensor = torch.FloatTensor(X_test_transformed)
```

```

# Create a Dataset class
class CustomDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

# Create DataLoaders
train_dataset = CustomDataset(X_train_tensor, y_train_tensor)
valid_dataset = CustomDataset(X_valid_tensor, y_valid_tensor)
test_dataset = CustomDataset(X_test_tensor, None) # No labels for test data
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=32, shuffle=False)

# Define the Neural Network
class SimpleNN(nn.Module):
    def __init__(self, input_size):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 2) # Adjust the output layer for binary classification

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Instantiate the model, define the loss function and optimizer
model = SimpleNN(X_train_tensor.shape[1])
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 20
for epoch in range(num_epochs):
    model.train()
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()

```

```
optimizer.step()

# Validation phase
model.eval()
with torch.no_grad():
    valid_outputs = model(X_valid_tensor)
    _, predicted = torch.max(valid_outputs, 1)
    acc = (predicted == y_valid_tensor).float().mean()

print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}, Validation Accuracy: {acc.item()*100:.2f}')

# Prepare test data predictions
model.eval()
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    _, test_predictions = torch.max(test_outputs, 1)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index, # Assuming the test data has an index or unique ID
    'output': test_predictions.numpy() # Convert tensor to numpy array
})
submission_df.to_csv('submission.csv', index=False)

print("Predictions saved to submission.csv")
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.4.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.19.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.6.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (10.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (3.0.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Epoch [1/20], Loss: 0.3076, Validation Accuracy: 83.86%
Epoch [2/20], Loss: 0.3692, Validation Accuracy: 83.75%
Epoch [3/20], Loss: 0.3993, Validation Accuracy: 84.62%
Epoch [4/20], Loss: 0.1882, Validation Accuracy: 83.97%
Epoch [5/20], Loss: 0.5604, Validation Accuracy: 84.51%
Epoch [6/20], Loss: 0.4467, Validation Accuracy: 84.30%
Epoch [7/20], Loss: 0.3913, Validation Accuracy: 83.97%
Epoch [8/20], Loss: 0.2076, Validation Accuracy: 83.97%
Epoch [9/20], Loss: 0.1128, Validation Accuracy: 83.97%
Epoch [10/20], Loss: 0.3575, Validation Accuracy: 83.42%
Epoch [11/20], Loss: 0.0947, Validation Accuracy: 83.42%
Epoch [12/20], Loss: 0.1783, Validation Accuracy: 82.01%
Epoch [13/20], Loss: 0.0425, Validation Accuracy: 82.99%
Epoch [14/20], Loss: 0.3540, Validation Accuracy: 82.33%
Epoch [15/20], Loss: 0.0652, Validation Accuracy: 82.55%
Epoch [16/20], Loss: 0.0604, Validation Accuracy: 82.77%
Epoch [17/20], Loss: 0.0310, Validation Accuracy: 83.42%
Epoch [18/20], Loss: 0.1451, Validation Accuracy: 83.10%
Epoch [19/20], Loss: 0.0464, Validation Accuracy: 83.42%
Epoch [20/20], Loss: 0.0838, Validation Accuracy: 83.86%
Predictions saved to submission.csv
```

```
In [ ]: pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.5.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.4)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoders) (24.1)
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl (82 kB)
```

Total **82.0**

```
Installing collected packages: category_encoders  
Successfully installed category_encoders-2.6.4
```

Trying out feature selection to see if that helps


```
In [ ]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from category_encoders import TargetEncoder
from sklearn.feature_selection import SelectFromModel

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
preprocessor = ColumnTransformer([
    ('target_encode', TargetEncoder(cols=categorical_columns), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Apply preprocessing to training data
X_train_transformed = preprocessor.fit_transform(X_train, y_train) # Pass y_train here
X_valid_transformed = preprocessor.transform(X_valid)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# Feature selection using XGBoost
xgb_fs = xgb.XGBClassifier(
    n_estimators=1462,
```

```
max_depth=6,
learning_rate=0.008252647239585268,
subsample=0.8798353150886109,
colsample_bytree=0.9915078582609945,
reg_lambda=0.0021412922470020242,
alpha=1.1851271361097405e-05,
scale_pos_weight=3, # Adjusted for class imbalance
use_label_encoder=False,
eval_metric='mlogloss'
)

xgb_fs.fit(X_train_balanced, y_train_balanced)
selector = SelectFromModel(xgb_fs, prefit=True)

# Select important features
X_train_selected = selector.transform(X_train_balanced)
X_valid_selected = selector.transform(X_valid_transformed)

# Train final XGBoost model with selected features
model = xgb.XGBClassifier(
    n_estimators=1462,
    max_depth=6,
    learning_rate=0.008252647239585268,
    subsample=0.8798353150886109,
    colsample_bytree=0.9915078582609945,
    reg_lambda=0.0021412922470020242,
    alpha=1.1851271361097405e-05,
    scale_pos_weight=3, # Adjusted for class imbalance
    use_label_encoder=False,
    eval_metric='mlogloss'
)

model.fit(X_train_selected, y_train_balanced)

# Validate the model
y_pred = model.predict(X_valid_selected)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred))
print("Classification Report:\n", classification_report(y_valid, y_pred))

# Prepare test data for prediction
test_data_transformed = preprocessor.transform(test_data)
test_data_selected = selector.transform(test_data_transformed)
test_predictions = model.predict(test_data_selected)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})
```

```
})
submission_df.to_csv('submission_xgb_fs_smote.csv', index=False)

print("Predictions saved to submission_xgb_fs_smote.csv")
```

Validation Accuracy: 0.8604143947655398

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.23	0.35	148
1	0.87	0.98	0.92	769
accuracy			0.86	917
macro avg	0.79	0.61	0.63	917
weighted avg	0.84	0.86	0.83	917

Predictions saved to submission_xgb_fs_smote.csv

This leads to reduction in recall of label 0

Trying out lasso to reduce features


```
In [ ]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from category_encoders import TargetEncoder
from sklearn.linear_model import LassoCV
from sklearn.feature_selection import SelectFromModel

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
preprocessor = ColumnTransformer([
    ('target_encode', TargetEncoder(cols=categorical_columns), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Apply preprocessing to training data
X_train_transformed = preprocessor.fit_transform(X_train, y_train) # Pass y_train here
X_valid_transformed = preprocessor.transform(X_valid)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# Feature selection using LassoCV (L1 regularization)
lasso = LassoCV(cv=5, random_state=42).fit(X_train_balanced, y_train_balanced)
```

```

# Select important features
lasso_selector = SelectFromModel(lasso, prefit=True)
X_train_lasso_selected = lasso_selector.transform(X_train_balanced)
X_valid_lasso_selected = lasso_selector.transform(X_valid_transformed)

# Train final XGBoost model with selected features
model = xgb.XGBClassifier(
    n_estimators=1462,
    max_depth=6,
    learning_rate=0.008252647239585268,
    subsample=0.8798353150886109,
    colsample_bytree=0.9915078582609945,
    reg_lambda=0.0021412922470020242,
    alpha=1.1851271361097405e-05,
    scale_pos_weight=4, # Focus more on minority class
    use_label_encoder=False,
    eval_metric='mlogloss'
)

model.fit(X_train_lasso_selected, y_train_balanced)

# Validate the model
y_pred = model.predict(X_valid_lasso_selected)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred))
print("Classification Report:\n", classification_report(y_valid, y_pred))

# Prepare test data for prediction
test_data_transformed = preprocessor.transform(test_data)
test_data_lasso_selected = lasso_selector.transform(test_data_transformed)
test_predictions = model.predict(test_data_lasso_selected)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})
submission_df.to_csv('submission_xgb_lasso_smote.csv', index=False)

print("Predictions saved to submission_xgb_lasso_smote.csv")

```

Validation Accuracy: 0.8538713195201745

Classification Report:

	precision	recall	f1-score	support
0	0.67	0.19	0.29	148
1	0.86	0.98	0.92	769
accuracy			0.85	917
macro avg	0.76	0.59	0.61	917
weighted avg	0.83	0.85	0.82	917

Predictions saved to submission_xgb_lasso_smote.csv

This reduces label 0 recall further to 19%

Trying out focal loss to add higher importance to minority class to get right


```
In [ ]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from category_encoders import TargetEncoder
from sklearn.feature_selection import SelectFromModel

# Custom Focal Loss Objective for XGBoost
def focal_loss(alpha, gamma):
    def focal_loss_obj(preds, dtrain):
        labels = dtrain.get_label()
        preds = 1.0 / (1.0 + np.exp(-preds)) # Sigmoid
        loss = -alpha * labels * np.power(1 - preds, gamma) * np.log(preds) - \
              (1 - alpha) * (1 - labels) * np.power(preds, gamma) * np.log(1 - preds)
        grad = alpha * (preds - labels) * np.power(1 - preds, gamma - 1) * gamma
        hess = alpha * np.power(1 - preds, gamma - 1) * (1 - 2 * preds) * gamma + grad * (1 - grad)
        return grad, hess
    return focal_loss_obj

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
preprocessor = ColumnTransformer([
    ('target_encode', TargetEncoder(cols=categorical_columns), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])
```

```
# Apply preprocessing to training data
X_train_transformed = preprocess.fit_transform(X_train, y_train) # Pass y_train here
X_valid_transformed = preprocess.transform(X_valid)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# Feature selection using SelectFromModel
xgb_fs = xgb.XGBClassifier(
    n_estimators=1462,
    max_depth=6,
    learning_rate=0.008252647239585268,
    subsample=0.8798353150886109,
    colsample_bytree=0.9915078582609945,
    reg_lambda=0.0021412922470020242,
    alpha=1.1851271361097405e-05,
    scale_pos_weight=10, # Significantly increased to focus on label 0
    use_label_encoder=False,
    eval_metric='mlogloss'
)

xgb_fs.fit(X_train_balanced, y_train_balanced)
selector = SelectFromModel(xgb_fs, prefit=True)

# Select important features
X_train_selected = selector.transform(X_train_balanced)
X_valid_selected = selector.transform(X_valid_transformed)

# Convert data to DMatrix format for XGBoost
dtrain = xgb.DMatrix(X_train_selected, label=y_train_balanced)
dvalid = xgb.DMatrix(X_valid_selected, label=y_valid)

# Parameters (without the custom objective here)
params = {
    'eval_metric': 'logloss',
    'max_depth': 6,
    'learning_rate': 0.008252647239585268,
    'subsample': 0.8798353150886109,
    'colsample_bytree': 0.9915078582609945,
    'reg_lambda': 0.0021412922470020242,
    'alpha': 1.1851271361097405e-05,
    'scale_pos_weight': 10,
    'seed': 42,
    'silent': 1
}
```

```
# Custom training loop with Focal Loss objective
evallist = [(dvalid, 'eval')]
bst = xgb.train(params, dtrain, num_boost_round=1462, evals=evallist, obj=focal_loss(alpha=0.25, gamma=2))

# Validate the model
y_pred = bst.predict(dvalid)
y_pred_labels = np.where(y_pred > 0.5, 1, 0) # Convert probabilities to binary labels

print("Validation Accuracy:", accuracy_score(y_valid, y_pred_labels))
print("Classification Report:\n", classification_report(y_valid, y_pred_labels))

# Prepare test data for prediction
test_data_transformed = preprocessor.transform(test_data)
test_data_selected = selector.transform(test_data_transformed)
dtest = xgb.DMatrix(test_data_selected)
test_predictions = bst.predict(dtest)
test_predictions_labels = np.where(test_predictions > 0.5, 1, 0)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions_labels
})
submission_df.to_csv('submission_xgb_focal_smote_dmatrix.csv', index=False)

print("Predictions saved to submission_xgb_focal_smote_dmatrix.csv")
```

```
[0] eval-logloss:0.69315
[1] eval-logloss:0.69315
[2] eval-logloss:0.69315
[3] eval-logloss:0.69315
[4] eval-logloss:0.69315
[5] eval-logloss:0.69315
[6] eval-logloss:0.69315
[7] eval-logloss:0.69315
[8] eval-logloss:0.69315
[9] eval-logloss:0.69315
[10] eval-logloss:0.69315
[11] eval-logloss:0.69315
[12] eval-logloss:0.69315
[13] eval-logloss:0.69315
[14] eval-logloss:0.69315
[15] eval-logloss:0.69315
[16] eval-logloss:0.69315
[17] eval-logloss:0.69315
[18] eval-logloss:0.69315
[19] eval-logloss:0.69315
[20] eval-logloss:0.69315
[21] eval-logloss:0.69315
[22] eval-logloss:0.69315
[23] eval-logloss:0.69315
[24] eval-logloss:0.69315
[25] eval-logloss:0.69315
[26] eval-logloss:0.69315
[27] eval-logloss:0.69315
[28] eval-logloss:0.69315
[29] eval-logloss:0.69315
[30] eval-logloss:0.69315
[31] eval-logloss:0.69315
[32] eval-logloss:0.69315
[33] eval-logloss:0.69315
[34] eval-logloss:0.69315
[35] eval-logloss:0.69315
[36] eval-logloss:0.69315
[37] eval-logloss:0.69315
[38] eval-logloss:0.69315
[39] eval-logloss:0.69315
[40] eval-logloss:0.69315
[41] eval-logloss:0.69315
[42] eval-logloss:0.69315
[43] eval-logloss:0.69315
[44] eval-logloss:0.69315
[45] eval-logloss:0.69315
[46] eval-logloss:0.69315
```

```
[47] eval-logloss:0.69315
[48] eval-logloss:0.69315
[49] eval-logloss:0.69315
[50] eval-logloss:0.69315
[51] eval-logloss:0.69315
[52] eval-logloss:0.69315
[53] eval-logloss:0.69315
[54] eval-logloss:0.69315
[55] eval-logloss:0.69315
[56] eval-logloss:0.69315
[57] eval-logloss:0.69315
[58] eval-logloss:0.69315
[59] eval-logloss:0.69315
[60] eval-logloss:0.69315
[61] eval-logloss:0.69315
[62] eval-logloss:0.69315
[63] eval-logloss:0.69315
[64] eval-logloss:0.69315
[65] eval-logloss:0.69315
[66] eval-logloss:0.69315
[67] eval-logloss:0.69315
[68] eval-logloss:0.69315
[69] eval-logloss:0.69315
[70] eval-logloss:0.69315
[71] eval-logloss:0.69315
[72] eval-logloss:0.69315
[73] eval-logloss:0.69315
[74] eval-logloss:0.69315
[75] eval-logloss:0.69315
[76] eval-logloss:0.69315
[77] eval-logloss:0.69315
[78] eval-logloss:0.69315
[79] eval-logloss:0.69315
[80] eval-logloss:0.69315
[81] eval-logloss:0.69315
[82] eval-logloss:0.69315
[83] eval-logloss:0.69315
[84] eval-logloss:0.69315
[85] eval-logloss:0.69315
[86] eval-logloss:0.69315
[87] eval-logloss:0.69315
[88] eval-logloss:0.69315
[89] eval-logloss:0.69315
[90] eval-logloss:0.69315
[91] eval-logloss:0.69315
[92] eval-logloss:0.69315
[93] eval-logloss:0.69315
[94] eval-logloss:0.69315
```

```
[95] eval-logloss:0.69315
[96] eval-logloss:0.69315
[97] eval-logloss:0.69315
[98] eval-logloss:0.69315
[99] eval-logloss:0.69315
[100] eval-logloss:0.69315
[101] eval-logloss:0.69315
[102] eval-logloss:0.69315
[103] eval-logloss:0.69315
[104] eval-logloss:0.69315
[105] eval-logloss:0.69315
[106] eval-logloss:0.69315
[107] eval-logloss:0.69315
[108] eval-logloss:0.69315
[109] eval-logloss:0.69315
[110] eval-logloss:0.69315
[111] eval-logloss:0.69315
[112] eval-logloss:0.69315
[113] eval-logloss:0.69315
[114] eval-logloss:0.69315
[115] eval-logloss:0.69315
[116] eval-logloss:0.69315
[117] eval-logloss:0.69315
[118] eval-logloss:0.69315
[119] eval-logloss:0.69315
[120] eval-logloss:0.69315
[121] eval-logloss:0.69315
[122] eval-logloss:0.69315
[123] eval-logloss:0.69315
[124] eval-logloss:0.69315
[125] eval-logloss:0.69315
[126] eval-logloss:0.69315
[127] eval-logloss:0.69315
[128] eval-logloss:0.69315
[129] eval-logloss:0.69315
[130] eval-logloss:0.69315
[131] eval-logloss:0.69315
[132] eval-logloss:0.69315
[133] eval-logloss:0.69315
[134] eval-logloss:0.69315
[135] eval-logloss:0.69315
[136] eval-logloss:0.69315
[137] eval-logloss:0.69315
[138] eval-logloss:0.69315
[139] eval-logloss:0.69315
[140] eval-logloss:0.69315
[141] eval-logloss:0.69315
[142] eval-logloss:0.69315
```

```
[143] eval-logloss:0.69315
[144] eval-logloss:0.69315
[145] eval-logloss:0.69315
[146] eval-logloss:0.69315
[147] eval-logloss:0.69315
[148] eval-logloss:0.69315
[149] eval-logloss:0.69315
[150] eval-logloss:0.69315
[151] eval-logloss:0.69315
[152] eval-logloss:0.69315
[153] eval-logloss:0.69315
[154] eval-logloss:0.69315
[155] eval-logloss:0.69315
[156] eval-logloss:0.69315
[157] eval-logloss:0.69315
[158] eval-logloss:0.69315
[159] eval-logloss:0.69315
[160] eval-logloss:0.69315
[161] eval-logloss:0.69315
[162] eval-logloss:0.69315
[163] eval-logloss:0.69315
[164] eval-logloss:0.69315
[165] eval-logloss:0.69315
[166] eval-logloss:0.69315
[167] eval-logloss:0.69315
[168] eval-logloss:0.69315
[169] eval-logloss:0.69315
[170] eval-logloss:0.69315
[171] eval-logloss:0.69315
[172] eval-logloss:0.69315
[173] eval-logloss:0.69315
[174] eval-logloss:0.69315
[175] eval-logloss:0.69315
[176] eval-logloss:0.69315
[177] eval-logloss:0.69315
[178] eval-logloss:0.69315
[179] eval-logloss:0.69315
[180] eval-logloss:0.69315
[181] eval-logloss:0.69315
[182] eval-logloss:0.69315
[183] eval-logloss:0.69315
[184] eval-logloss:0.69315
[185] eval-logloss:0.69315
[186] eval-logloss:0.69315
[187] eval-logloss:0.69315
[188] eval-logloss:0.69315
[189] eval-logloss:0.69315
[190] eval-logloss:0.69315
```

```
[191] eval-logloss:0.69315
[192] eval-logloss:0.69315
[193] eval-logloss:0.69315
[194] eval-logloss:0.69315
[195] eval-logloss:0.69315
[196] eval-logloss:0.69315
[197] eval-logloss:0.69315
[198] eval-logloss:0.69315
[199] eval-logloss:0.69315
[200] eval-logloss:0.69315
[201] eval-logloss:0.69315
[202] eval-logloss:0.69315
[203] eval-logloss:0.69315
[204] eval-logloss:0.69315
[205] eval-logloss:0.69315
[206] eval-logloss:0.69315
[207] eval-logloss:0.69315
[208] eval-logloss:0.69315
[209] eval-logloss:0.69315
[210] eval-logloss:0.69315
[211] eval-logloss:0.69315
[212] eval-logloss:0.69315
[213] eval-logloss:0.69315
[214] eval-logloss:0.69315
[215] eval-logloss:0.69315
[216] eval-logloss:0.69315
[217] eval-logloss:0.69315
[218] eval-logloss:0.69315
[219] eval-logloss:0.69315
[220] eval-logloss:0.69315
[221] eval-logloss:0.69315
[222] eval-logloss:0.69315
[223] eval-logloss:0.69315
[224] eval-logloss:0.69315
[225] eval-logloss:0.69315
[226] eval-logloss:0.69315
[227] eval-logloss:0.69315
[228] eval-logloss:0.69315
[229] eval-logloss:0.69315
[230] eval-logloss:0.69315
[231] eval-logloss:0.69315
[232] eval-logloss:0.69315
[233] eval-logloss:0.69315
[234] eval-logloss:0.69315
[235] eval-logloss:0.69315
[236] eval-logloss:0.69315
[237] eval-logloss:0.69315
[238] eval-logloss:0.69315
```

```
[239] eval-logloss:0.69315
[240] eval-logloss:0.69315
[241] eval-logloss:0.69315
[242] eval-logloss:0.69315
[243] eval-logloss:0.69315
[244] eval-logloss:0.69315
[245] eval-logloss:0.69315
[246] eval-logloss:0.69315
[247] eval-logloss:0.69315
[248] eval-logloss:0.69315
[249] eval-logloss:0.69315
[250] eval-logloss:0.69315
[251] eval-logloss:0.69315
[252] eval-logloss:0.69315
[253] eval-logloss:0.69315
[254] eval-logloss:0.69315
[255] eval-logloss:0.69315
[256] eval-logloss:0.69315
[257] eval-logloss:0.69315
[258] eval-logloss:0.69315
[259] eval-logloss:0.69315
[260] eval-logloss:0.69315
[261] eval-logloss:0.69315
[262] eval-logloss:0.69315
[263] eval-logloss:0.69315
[264] eval-logloss:0.69315
[265] eval-logloss:0.69315
[266] eval-logloss:0.69315
[267] eval-logloss:0.69315
[268] eval-logloss:0.69315
[269] eval-logloss:0.69315
[270] eval-logloss:0.69315
[271] eval-logloss:0.69315
[272] eval-logloss:0.69315
[273] eval-logloss:0.69315
[274] eval-logloss:0.69315
[275] eval-logloss:0.69315
[276] eval-logloss:0.69315
[277] eval-logloss:0.69315
[278] eval-logloss:0.69315
[279] eval-logloss:0.69315
[280] eval-logloss:0.69315
[281] eval-logloss:0.69315
[282] eval-logloss:0.69315
[283] eval-logloss:0.69315
[284] eval-logloss:0.69315
[285] eval-logloss:0.69315
[286] eval-logloss:0.69315
```

```
[287] eval-logloss:0.69315
[288] eval-logloss:0.69315
[289] eval-logloss:0.69315
[290] eval-logloss:0.69315
[291] eval-logloss:0.69315
[292] eval-logloss:0.69315
[293] eval-logloss:0.69315
[294] eval-logloss:0.69315
[295] eval-logloss:0.69315
[296] eval-logloss:0.69315
[297] eval-logloss:0.69315
[298] eval-logloss:0.69315
[299] eval-logloss:0.69315
[300] eval-logloss:0.69315
[301] eval-logloss:0.69315
[302] eval-logloss:0.69315
[303] eval-logloss:0.69315
[304] eval-logloss:0.69315
[305] eval-logloss:0.69315
[306] eval-logloss:0.69315
[307] eval-logloss:0.69315
[308] eval-logloss:0.69315
[309] eval-logloss:0.69315
[310] eval-logloss:0.69315
[311] eval-logloss:0.69315
[312] eval-logloss:0.69315
[313] eval-logloss:0.69315
[314] eval-logloss:0.69315
[315] eval-logloss:0.69315
[316] eval-logloss:0.69315
[317] eval-logloss:0.69315
[318] eval-logloss:0.69315
[319] eval-logloss:0.69315
[320] eval-logloss:0.69315
[321] eval-logloss:0.69315
[322] eval-logloss:0.69315
[323] eval-logloss:0.69315
[324] eval-logloss:0.69315
[325] eval-logloss:0.69315
[326] eval-logloss:0.69315
[327] eval-logloss:0.69315
[328] eval-logloss:0.69315
[329] eval-logloss:0.69315
[330] eval-logloss:0.69315
[331] eval-logloss:0.69315
[332] eval-logloss:0.69315
[333] eval-logloss:0.69315
[334] eval-logloss:0.69315
```

```
[335] eval-logloss:0.69315
[336] eval-logloss:0.69315
[337] eval-logloss:0.69315
[338] eval-logloss:0.69315
[339] eval-logloss:0.69315
[340] eval-logloss:0.69315
[341] eval-logloss:0.69315
[342] eval-logloss:0.69315
[343] eval-logloss:0.69315
[344] eval-logloss:0.69315
[345] eval-logloss:0.69315
[346] eval-logloss:0.69315
[347] eval-logloss:0.69315
[348] eval-logloss:0.69315
[349] eval-logloss:0.69315
[350] eval-logloss:0.69315
[351] eval-logloss:0.69315
[352] eval-logloss:0.69315
[353] eval-logloss:0.69315
[354] eval-logloss:0.69315
[355] eval-logloss:0.69315
[356] eval-logloss:0.69315
[357] eval-logloss:0.69315
[358] eval-logloss:0.69315
[359] eval-logloss:0.69315
[360] eval-logloss:0.69315
[361] eval-logloss:0.69315
[362] eval-logloss:0.69315
[363] eval-logloss:0.69315
[364] eval-logloss:0.69315
[365] eval-logloss:0.69315
[366] eval-logloss:0.69315
[367] eval-logloss:0.69315
[368] eval-logloss:0.69315
[369] eval-logloss:0.69315
[370] eval-logloss:0.69315
[371] eval-logloss:0.69315
[372] eval-logloss:0.69315
[373] eval-logloss:0.69315
[374] eval-logloss:0.69315
[375] eval-logloss:0.69315
[376] eval-logloss:0.69315
[377] eval-logloss:0.69315
[378] eval-logloss:0.69315
[379] eval-logloss:0.69315
[380] eval-logloss:0.69315
[381] eval-logloss:0.69315
[382] eval-logloss:0.69315
```

```
[383] eval-logloss:0.69315
[384] eval-logloss:0.69315
[385] eval-logloss:0.69315
[386] eval-logloss:0.69315
[387] eval-logloss:0.69315
[388] eval-logloss:0.69315
[389] eval-logloss:0.69315
[390] eval-logloss:0.69315
[391] eval-logloss:0.69315
[392] eval-logloss:0.69315
[393] eval-logloss:0.69315
[394] eval-logloss:0.69315
[395] eval-logloss:0.69315
[396] eval-logloss:0.69315
[397] eval-logloss:0.69315
[398] eval-logloss:0.69315
[399] eval-logloss:0.69315
[400] eval-logloss:0.69315
[401] eval-logloss:0.69315
[402] eval-logloss:0.69315
[403] eval-logloss:0.69315
[404] eval-logloss:0.69315
[405] eval-logloss:0.69315
[406] eval-logloss:0.69315
[407] eval-logloss:0.69315
[408] eval-logloss:0.69315
[409] eval-logloss:0.69315
[410] eval-logloss:0.69315
[411] eval-logloss:0.69315
[412] eval-logloss:0.69315
[413] eval-logloss:0.69315
[414] eval-logloss:0.69315
[415] eval-logloss:0.69315
[416] eval-logloss:0.69315
[417] eval-logloss:0.69315
[418] eval-logloss:0.69315
[419] eval-logloss:0.69315
[420] eval-logloss:0.69315
[421] eval-logloss:0.69315
[422] eval-logloss:0.69315
[423] eval-logloss:0.69315
[424] eval-logloss:0.69315
[425] eval-logloss:0.69315
[426] eval-logloss:0.69315
[427] eval-logloss:0.69315
[428] eval-logloss:0.69315
[429] eval-logloss:0.69315
[430] eval-logloss:0.69315
```

```
[431] eval-logloss:0.69315
[432] eval-logloss:0.69315
[433] eval-logloss:0.69315
[434] eval-logloss:0.69315
[435] eval-logloss:0.69315
[436] eval-logloss:0.69315
[437] eval-logloss:0.69315
[438] eval-logloss:0.69315
[439] eval-logloss:0.69315
[440] eval-logloss:0.69315
[441] eval-logloss:0.69315
[442] eval-logloss:0.69315
[443] eval-logloss:0.69315
[444] eval-logloss:0.69315
[445] eval-logloss:0.69315
[446] eval-logloss:0.69315
[447] eval-logloss:0.69315
[448] eval-logloss:0.69315
[449] eval-logloss:0.69315
[450] eval-logloss:0.69315
[451] eval-logloss:0.69315
[452] eval-logloss:0.69315
[453] eval-logloss:0.69315
[454] eval-logloss:0.69315
[455] eval-logloss:0.69315
[456] eval-logloss:0.69315
[457] eval-logloss:0.69315
[458] eval-logloss:0.69315
[459] eval-logloss:0.69315
[460] eval-logloss:0.69315
[461] eval-logloss:0.69315
[462] eval-logloss:0.69315
[463] eval-logloss:0.69315
[464] eval-logloss:0.69315
[465] eval-logloss:0.69315
[466] eval-logloss:0.69315
[467] eval-logloss:0.69315
[468] eval-logloss:0.69315
[469] eval-logloss:0.69315
[470] eval-logloss:0.69315
[471] eval-logloss:0.69315
[472] eval-logloss:0.69315
[473] eval-logloss:0.69315
[474] eval-logloss:0.69315
[475] eval-logloss:0.69315
[476] eval-logloss:0.69315
[477] eval-logloss:0.69315
[478] eval-logloss:0.69315
```

```
[479] eval-logloss:0.69315
[480] eval-logloss:0.69315
[481] eval-logloss:0.69315
[482] eval-logloss:0.69315
[483] eval-logloss:0.69315
[484] eval-logloss:0.69315
[485] eval-logloss:0.69315
[486] eval-logloss:0.69315
[487] eval-logloss:0.69315
[488] eval-logloss:0.69315
[489] eval-logloss:0.69315
[490] eval-logloss:0.69315
[491] eval-logloss:0.69315
[492] eval-logloss:0.69315
[493] eval-logloss:0.69315
[494] eval-logloss:0.69315
[495] eval-logloss:0.69315
[496] eval-logloss:0.69315
[497] eval-logloss:0.69315
[498] eval-logloss:0.69315
[499] eval-logloss:0.69315
[500] eval-logloss:0.69315
[501] eval-logloss:0.69315
[502] eval-logloss:0.69315
[503] eval-logloss:0.69315
[504] eval-logloss:0.69315
[505] eval-logloss:0.69315
[506] eval-logloss:0.69315
[507] eval-logloss:0.69315
[508] eval-logloss:0.69315
[509] eval-logloss:0.69315
[510] eval-logloss:0.69315
[511] eval-logloss:0.69315
[512] eval-logloss:0.69315
[513] eval-logloss:0.69315
[514] eval-logloss:0.69315
[515] eval-logloss:0.69315
[516] eval-logloss:0.69315
[517] eval-logloss:0.69315
[518] eval-logloss:0.69315
[519] eval-logloss:0.69315
[520] eval-logloss:0.69315
[521] eval-logloss:0.69315
[522] eval-logloss:0.69315
[523] eval-logloss:0.69315
[524] eval-logloss:0.69315
[525] eval-logloss:0.69315
[526] eval-logloss:0.69315
```

[527] eval-logloss:0.69315
[528] eval-logloss:0.69315
[529] eval-logloss:0.69315
[530] eval-logloss:0.69315
[531] eval-logloss:0.69315
[532] eval-logloss:0.69315
[533] eval-logloss:0.69315
[534] eval-logloss:0.69315
[535] eval-logloss:0.69315
[536] eval-logloss:0.69315
[537] eval-logloss:0.69315
[538] eval-logloss:0.69315
[539] eval-logloss:0.69315
[540] eval-logloss:0.69315
[541] eval-logloss:0.69315
[542] eval-logloss:0.69315
[543] eval-logloss:0.69315
[544] eval-logloss:0.69315
[545] eval-logloss:0.69315
[546] eval-logloss:0.69315
[547] eval-logloss:0.69315
[548] eval-logloss:0.69315
[549] eval-logloss:0.69315
[550] eval-logloss:0.69315
[551] eval-logloss:0.69315
[552] eval-logloss:0.69315
[553] eval-logloss:0.69315
[554] eval-logloss:0.69315
[555] eval-logloss:0.69315
[556] eval-logloss:0.69315
[557] eval-logloss:0.69315
[558] eval-logloss:0.69315
[559] eval-logloss:0.69315
[560] eval-logloss:0.69315
[561] eval-logloss:0.69315
[562] eval-logloss:0.69315
[563] eval-logloss:0.69315
[564] eval-logloss:0.69315
[565] eval-logloss:0.69315
[566] eval-logloss:0.69315
[567] eval-logloss:0.69315
[568] eval-logloss:0.69315
[569] eval-logloss:0.69315
[570] eval-logloss:0.69315
[571] eval-logloss:0.69315
[572] eval-logloss:0.69315
[573] eval-logloss:0.69315
[574] eval-logloss:0.69315

[575] eval-logloss:0.69315
[576] eval-logloss:0.69315
[577] eval-logloss:0.69315
[578] eval-logloss:0.69315
[579] eval-logloss:0.69315
[580] eval-logloss:0.69315
[581] eval-logloss:0.69315
[582] eval-logloss:0.69315
[583] eval-logloss:0.69315
[584] eval-logloss:0.69315
[585] eval-logloss:0.69315
[586] eval-logloss:0.69315
[587] eval-logloss:0.69315
[588] eval-logloss:0.69315
[589] eval-logloss:0.69315
[590] eval-logloss:0.69315
[591] eval-logloss:0.69315
[592] eval-logloss:0.69315
[593] eval-logloss:0.69315
[594] eval-logloss:0.69315
[595] eval-logloss:0.69315
[596] eval-logloss:0.69315
[597] eval-logloss:0.69315
[598] eval-logloss:0.69315
[599] eval-logloss:0.69315
[600] eval-logloss:0.69315
[601] eval-logloss:0.69315
[602] eval-logloss:0.69315
[603] eval-logloss:0.69315
[604] eval-logloss:0.69315
[605] eval-logloss:0.69315
[606] eval-logloss:0.69315
[607] eval-logloss:0.69315
[608] eval-logloss:0.69315
[609] eval-logloss:0.69315
[610] eval-logloss:0.69315
[611] eval-logloss:0.69315
[612] eval-logloss:0.69315
[613] eval-logloss:0.69315
[614] eval-logloss:0.69315
[615] eval-logloss:0.69315
[616] eval-logloss:0.69315
[617] eval-logloss:0.69315
[618] eval-logloss:0.69315
[619] eval-logloss:0.69315
[620] eval-logloss:0.69315
[621] eval-logloss:0.69315
[622] eval-logloss:0.69315

[623] eval-logloss:0.69315
[624] eval-logloss:0.69315
[625] eval-logloss:0.69315
[626] eval-logloss:0.69315
[627] eval-logloss:0.69315
[628] eval-logloss:0.69315
[629] eval-logloss:0.69315
[630] eval-logloss:0.69315
[631] eval-logloss:0.69315
[632] eval-logloss:0.69315
[633] eval-logloss:0.69315
[634] eval-logloss:0.69315
[635] eval-logloss:0.69315
[636] eval-logloss:0.69315
[637] eval-logloss:0.69315
[638] eval-logloss:0.69315
[639] eval-logloss:0.69315
[640] eval-logloss:0.69315
[641] eval-logloss:0.69315
[642] eval-logloss:0.69315
[643] eval-logloss:0.69315
[644] eval-logloss:0.69315
[645] eval-logloss:0.69315
[646] eval-logloss:0.69315
[647] eval-logloss:0.69315
[648] eval-logloss:0.69315
[649] eval-logloss:0.69315
[650] eval-logloss:0.69315
[651] eval-logloss:0.69315
[652] eval-logloss:0.69315
[653] eval-logloss:0.69315
[654] eval-logloss:0.69315
[655] eval-logloss:0.69315
[656] eval-logloss:0.69315
[657] eval-logloss:0.69315
[658] eval-logloss:0.69315
[659] eval-logloss:0.69315
[660] eval-logloss:0.69315
[661] eval-logloss:0.69315
[662] eval-logloss:0.69315
[663] eval-logloss:0.69315
[664] eval-logloss:0.69315
[665] eval-logloss:0.69315
[666] eval-logloss:0.69315
[667] eval-logloss:0.69315
[668] eval-logloss:0.69315
[669] eval-logloss:0.69315
[670] eval-logloss:0.69315

[671] eval-logloss:0.69315
[672] eval-logloss:0.69315
[673] eval-logloss:0.69315
[674] eval-logloss:0.69315
[675] eval-logloss:0.69315
[676] eval-logloss:0.69315
[677] eval-logloss:0.69315
[678] eval-logloss:0.69315
[679] eval-logloss:0.69315
[680] eval-logloss:0.69315
[681] eval-logloss:0.69315
[682] eval-logloss:0.69315
[683] eval-logloss:0.69315
[684] eval-logloss:0.69315
[685] eval-logloss:0.69315
[686] eval-logloss:0.69315
[687] eval-logloss:0.69315
[688] eval-logloss:0.69315
[689] eval-logloss:0.69315
[690] eval-logloss:0.69315
[691] eval-logloss:0.69315
[692] eval-logloss:0.69315
[693] eval-logloss:0.69315
[694] eval-logloss:0.69315
[695] eval-logloss:0.69315
[696] eval-logloss:0.69315
[697] eval-logloss:0.69315
[698] eval-logloss:0.69315
[699] eval-logloss:0.69315
[700] eval-logloss:0.69315
[701] eval-logloss:0.69315
[702] eval-logloss:0.69315
[703] eval-logloss:0.69315
[704] eval-logloss:0.69315
[705] eval-logloss:0.69315
[706] eval-logloss:0.69315
[707] eval-logloss:0.69315
[708] eval-logloss:0.69315
[709] eval-logloss:0.69315
[710] eval-logloss:0.69315
[711] eval-logloss:0.69315
[712] eval-logloss:0.69315
[713] eval-logloss:0.69315
[714] eval-logloss:0.69315
[715] eval-logloss:0.69315
[716] eval-logloss:0.69315
[717] eval-logloss:0.69315
[718] eval-logloss:0.69315

```
[719] eval-logloss:0.69315
[720] eval-logloss:0.69315
[721] eval-logloss:0.69315
[722] eval-logloss:0.69315
[723] eval-logloss:0.69315
[724] eval-logloss:0.69315
[725] eval-logloss:0.69315
[726] eval-logloss:0.69315
[727] eval-logloss:0.69315
[728] eval-logloss:0.69315
[729] eval-logloss:0.69315
[730] eval-logloss:0.69315
[731] eval-logloss:0.69315
[732] eval-logloss:0.69315
[733] eval-logloss:0.69315
[734] eval-logloss:0.69315
[735] eval-logloss:0.69315
[736] eval-logloss:0.69315
[737] eval-logloss:0.69315
[738] eval-logloss:0.69315
[739] eval-logloss:0.69315
[740] eval-logloss:0.69315
[741] eval-logloss:0.69315
[742] eval-logloss:0.69315
[743] eval-logloss:0.69315
[744] eval-logloss:0.69315
[745] eval-logloss:0.69315
[746] eval-logloss:0.69315
[747] eval-logloss:0.69315
[748] eval-logloss:0.69315
[749] eval-logloss:0.69315
[750] eval-logloss:0.69315
[751] eval-logloss:0.69315
[752] eval-logloss:0.69315
[753] eval-logloss:0.69315
[754] eval-logloss:0.69315
[755] eval-logloss:0.69315
[756] eval-logloss:0.69315
[757] eval-logloss:0.69315
[758] eval-logloss:0.69315
[759] eval-logloss:0.69315
[760] eval-logloss:0.69315
[761] eval-logloss:0.69315
[762] eval-logloss:0.69315
[763] eval-logloss:0.69315
[764] eval-logloss:0.69315
[765] eval-logloss:0.69315
[766] eval-logloss:0.69315
```

[767] eval-logloss:0.69315
[768] eval-logloss:0.69315
[769] eval-logloss:0.69315
[770] eval-logloss:0.69315
[771] eval-logloss:0.69315
[772] eval-logloss:0.69315
[773] eval-logloss:0.69315
[774] eval-logloss:0.69315
[775] eval-logloss:0.69315
[776] eval-logloss:0.69315
[777] eval-logloss:0.69315
[778] eval-logloss:0.69315
[779] eval-logloss:0.69315
[780] eval-logloss:0.69315
[781] eval-logloss:0.69315
[782] eval-logloss:0.69315
[783] eval-logloss:0.69315
[784] eval-logloss:0.69315
[785] eval-logloss:0.69315
[786] eval-logloss:0.69315
[787] eval-logloss:0.69315
[788] eval-logloss:0.69315
[789] eval-logloss:0.69315
[790] eval-logloss:0.69315
[791] eval-logloss:0.69315
[792] eval-logloss:0.69315
[793] eval-logloss:0.69315
[794] eval-logloss:0.69315
[795] eval-logloss:0.69315
[796] eval-logloss:0.69315
[797] eval-logloss:0.69315
[798] eval-logloss:0.69315
[799] eval-logloss:0.69315
[800] eval-logloss:0.69315
[801] eval-logloss:0.69315
[802] eval-logloss:0.69315
[803] eval-logloss:0.69315
[804] eval-logloss:0.69315
[805] eval-logloss:0.69315
[806] eval-logloss:0.69315
[807] eval-logloss:0.69315
[808] eval-logloss:0.69315
[809] eval-logloss:0.69315
[810] eval-logloss:0.69315
[811] eval-logloss:0.69315
[812] eval-logloss:0.69315
[813] eval-logloss:0.69315
[814] eval-logloss:0.69315

```
[815] eval-logloss:0.69315
[816] eval-logloss:0.69315
[817] eval-logloss:0.69315
[818] eval-logloss:0.69315
[819] eval-logloss:0.69315
[820] eval-logloss:0.69315
[821] eval-logloss:0.69315
[822] eval-logloss:0.69315
[823] eval-logloss:0.69315
[824] eval-logloss:0.69315
[825] eval-logloss:0.69315
[826] eval-logloss:0.69315
[827] eval-logloss:0.69315
[828] eval-logloss:0.69315
[829] eval-logloss:0.69315
[830] eval-logloss:0.69315
[831] eval-logloss:0.69315
[832] eval-logloss:0.69315
[833] eval-logloss:0.69315
[834] eval-logloss:0.69315
[835] eval-logloss:0.69315
[836] eval-logloss:0.69315
[837] eval-logloss:0.69315
[838] eval-logloss:0.69315
[839] eval-logloss:0.69315
[840] eval-logloss:0.69315
[841] eval-logloss:0.69315
[842] eval-logloss:0.69315
[843] eval-logloss:0.69315
[844] eval-logloss:0.69315
[845] eval-logloss:0.69315
[846] eval-logloss:0.69315
[847] eval-logloss:0.69315
[848] eval-logloss:0.69315
[849] eval-logloss:0.69315
[850] eval-logloss:0.69315
[851] eval-logloss:0.69315
[852] eval-logloss:0.69315
[853] eval-logloss:0.69315
[854] eval-logloss:0.69315
[855] eval-logloss:0.69315
[856] eval-logloss:0.69315
[857] eval-logloss:0.69315
[858] eval-logloss:0.69315
[859] eval-logloss:0.69315
[860] eval-logloss:0.69315
[861] eval-logloss:0.69315
[862] eval-logloss:0.69315
```

[863] eval-logloss:0.69315
[864] eval-logloss:0.69315
[865] eval-logloss:0.69315
[866] eval-logloss:0.69315
[867] eval-logloss:0.69315
[868] eval-logloss:0.69315
[869] eval-logloss:0.69315
[870] eval-logloss:0.69315
[871] eval-logloss:0.69315
[872] eval-logloss:0.69315
[873] eval-logloss:0.69315
[874] eval-logloss:0.69315
[875] eval-logloss:0.69315
[876] eval-logloss:0.69315
[877] eval-logloss:0.69315
[878] eval-logloss:0.69315
[879] eval-logloss:0.69315
[880] eval-logloss:0.69315
[881] eval-logloss:0.69315
[882] eval-logloss:0.69315
[883] eval-logloss:0.69315
[884] eval-logloss:0.69315
[885] eval-logloss:0.69315
[886] eval-logloss:0.69315
[887] eval-logloss:0.69315
[888] eval-logloss:0.69315
[889] eval-logloss:0.69315
[890] eval-logloss:0.69315
[891] eval-logloss:0.69315
[892] eval-logloss:0.69315
[893] eval-logloss:0.69315
[894] eval-logloss:0.69315
[895] eval-logloss:0.69315
[896] eval-logloss:0.69315
[897] eval-logloss:0.69315
[898] eval-logloss:0.69315
[899] eval-logloss:0.69315
[900] eval-logloss:0.69315
[901] eval-logloss:0.69315
[902] eval-logloss:0.69315
[903] eval-logloss:0.69315
[904] eval-logloss:0.69315
[905] eval-logloss:0.69315
[906] eval-logloss:0.69315
[907] eval-logloss:0.69315
[908] eval-logloss:0.69315
[909] eval-logloss:0.69315
[910] eval-logloss:0.69315

[911] eval-logloss:0.69315
[912] eval-logloss:0.69315
[913] eval-logloss:0.69315
[914] eval-logloss:0.69315
[915] eval-logloss:0.69315
[916] eval-logloss:0.69315
[917] eval-logloss:0.69315
[918] eval-logloss:0.69315
[919] eval-logloss:0.69315
[920] eval-logloss:0.69315
[921] eval-logloss:0.69315
[922] eval-logloss:0.69315
[923] eval-logloss:0.69315
[924] eval-logloss:0.69315
[925] eval-logloss:0.69315
[926] eval-logloss:0.69315
[927] eval-logloss:0.69315
[928] eval-logloss:0.69315
[929] eval-logloss:0.69315
[930] eval-logloss:0.69315
[931] eval-logloss:0.69315
[932] eval-logloss:0.69315
[933] eval-logloss:0.69315
[934] eval-logloss:0.69315
[935] eval-logloss:0.69315
[936] eval-logloss:0.69315
[937] eval-logloss:0.69315
[938] eval-logloss:0.69315
[939] eval-logloss:0.69315
[940] eval-logloss:0.69315
[941] eval-logloss:0.69315
[942] eval-logloss:0.69315
[943] eval-logloss:0.69315
[944] eval-logloss:0.69315
[945] eval-logloss:0.69315
[946] eval-logloss:0.69315
[947] eval-logloss:0.69315
[948] eval-logloss:0.69315
[949] eval-logloss:0.69315
[950] eval-logloss:0.69315
[951] eval-logloss:0.69315
[952] eval-logloss:0.69315
[953] eval-logloss:0.69315
[954] eval-logloss:0.69315
[955] eval-logloss:0.69315
[956] eval-logloss:0.69315
[957] eval-logloss:0.69315
[958] eval-logloss:0.69315

```
[959] eval-logloss:0.69315
[960] eval-logloss:0.69315
[961] eval-logloss:0.69315
[962] eval-logloss:0.69315
[963] eval-logloss:0.69315
[964] eval-logloss:0.69315
[965] eval-logloss:0.69315
[966] eval-logloss:0.69315
[967] eval-logloss:0.69315
[968] eval-logloss:0.69315
[969] eval-logloss:0.69315
[970] eval-logloss:0.69315
[971] eval-logloss:0.69315
[972] eval-logloss:0.69315
[973] eval-logloss:0.69315
[974] eval-logloss:0.69315
[975] eval-logloss:0.69315
[976] eval-logloss:0.69315
[977] eval-logloss:0.69315
[978] eval-logloss:0.69315
[979] eval-logloss:0.69315
[980] eval-logloss:0.69315
[981] eval-logloss:0.69315
[982] eval-logloss:0.69315
[983] eval-logloss:0.69315
[984] eval-logloss:0.69315
[985] eval-logloss:0.69315
[986] eval-logloss:0.69315
[987] eval-logloss:0.69315
[988] eval-logloss:0.69315
[989] eval-logloss:0.69315
[990] eval-logloss:0.69315
[991] eval-logloss:0.69315
[992] eval-logloss:0.69315
[993] eval-logloss:0.69315
[994] eval-logloss:0.69315
[995] eval-logloss:0.69315
[996] eval-logloss:0.69315
[997] eval-logloss:0.69315
[998] eval-logloss:0.69315
[999] eval-logloss:0.69315
[1000] eval-logloss:0.69315
[1001] eval-logloss:0.69315
[1002] eval-logloss:0.69315
[1003] eval-logloss:0.69315
[1004] eval-logloss:0.69315
[1005] eval-logloss:0.69315
[1006] eval-logloss:0.69315
```

```
[1007] eval-logloss:0.69315
[1008] eval-logloss:0.69315
[1009] eval-logloss:0.69315
[1010] eval-logloss:0.69315
[1011] eval-logloss:0.69315
[1012] eval-logloss:0.69315
[1013] eval-logloss:0.69315
[1014] eval-logloss:0.69315
[1015] eval-logloss:0.69315
[1016] eval-logloss:0.69315
[1017] eval-logloss:0.69315
[1018] eval-logloss:0.69315
[1019] eval-logloss:0.69315
[1020] eval-logloss:0.69315
[1021] eval-logloss:0.69315
[1022] eval-logloss:0.69315
[1023] eval-logloss:0.69315
[1024] eval-logloss:0.69315
[1025] eval-logloss:0.69315
[1026] eval-logloss:0.69315
[1027] eval-logloss:0.69315
[1028] eval-logloss:0.69315
[1029] eval-logloss:0.69315
[1030] eval-logloss:0.69315
[1031] eval-logloss:0.69315
[1032] eval-logloss:0.69315
[1033] eval-logloss:0.69315
[1034] eval-logloss:0.69315
[1035] eval-logloss:0.69315
[1036] eval-logloss:0.69315
[1037] eval-logloss:0.69315
[1038] eval-logloss:0.69315
[1039] eval-logloss:0.69315
[1040] eval-logloss:0.69315
[1041] eval-logloss:0.69315
[1042] eval-logloss:0.69315
[1043] eval-logloss:0.69315
[1044] eval-logloss:0.69315
[1045] eval-logloss:0.69315
[1046] eval-logloss:0.69315
[1047] eval-logloss:0.69315
[1048] eval-logloss:0.69315
[1049] eval-logloss:0.69315
[1050] eval-logloss:0.69315
[1051] eval-logloss:0.69315
[1052] eval-logloss:0.69315
[1053] eval-logloss:0.69315
[1054] eval-logloss:0.69315
```

[1055] eval-logloss:0.69315
[1056] eval-logloss:0.69315
[1057] eval-logloss:0.69315
[1058] eval-logloss:0.69315
[1059] eval-logloss:0.69315
[1060] eval-logloss:0.69315
[1061] eval-logloss:0.69315
[1062] eval-logloss:0.69315
[1063] eval-logloss:0.69315
[1064] eval-logloss:0.69315
[1065] eval-logloss:0.69315
[1066] eval-logloss:0.69315
[1067] eval-logloss:0.69315
[1068] eval-logloss:0.69315
[1069] eval-logloss:0.69315
[1070] eval-logloss:0.69315
[1071] eval-logloss:0.69315
[1072] eval-logloss:0.69315
[1073] eval-logloss:0.69315
[1074] eval-logloss:0.69315
[1075] eval-logloss:0.69315
[1076] eval-logloss:0.69315
[1077] eval-logloss:0.69315
[1078] eval-logloss:0.69315
[1079] eval-logloss:0.69315
[1080] eval-logloss:0.69315
[1081] eval-logloss:0.69315
[1082] eval-logloss:0.69315
[1083] eval-logloss:0.69315
[1084] eval-logloss:0.69315
[1085] eval-logloss:0.69315
[1086] eval-logloss:0.69315
[1087] eval-logloss:0.69315
[1088] eval-logloss:0.69315
[1089] eval-logloss:0.69315
[1090] eval-logloss:0.69315
[1091] eval-logloss:0.69315
[1092] eval-logloss:0.69315
[1093] eval-logloss:0.69315
[1094] eval-logloss:0.69315
[1095] eval-logloss:0.69315
[1096] eval-logloss:0.69315
[1097] eval-logloss:0.69315
[1098] eval-logloss:0.69315
[1099] eval-logloss:0.69315
[1100] eval-logloss:0.69315
[1101] eval-logloss:0.69315
[1102] eval-logloss:0.69315

[1103] eval-logloss:0.69315
[1104] eval-logloss:0.69315
[1105] eval-logloss:0.69315
[1106] eval-logloss:0.69315
[1107] eval-logloss:0.69315
[1108] eval-logloss:0.69315
[1109] eval-logloss:0.69315
[1110] eval-logloss:0.69315
[1111] eval-logloss:0.69315
[1112] eval-logloss:0.69315
[1113] eval-logloss:0.69315
[1114] eval-logloss:0.69315
[1115] eval-logloss:0.69315
[1116] eval-logloss:0.69315
[1117] eval-logloss:0.69315
[1118] eval-logloss:0.69315
[1119] eval-logloss:0.69315
[1120] eval-logloss:0.69315
[1121] eval-logloss:0.69315
[1122] eval-logloss:0.69315
[1123] eval-logloss:0.69315
[1124] eval-logloss:0.69315
[1125] eval-logloss:0.69315
[1126] eval-logloss:0.69315
[1127] eval-logloss:0.69315
[1128] eval-logloss:0.69315
[1129] eval-logloss:0.69315
[1130] eval-logloss:0.69315
[1131] eval-logloss:0.69315
[1132] eval-logloss:0.69315
[1133] eval-logloss:0.69315
[1134] eval-logloss:0.69315
[1135] eval-logloss:0.69315
[1136] eval-logloss:0.69315
[1137] eval-logloss:0.69315
[1138] eval-logloss:0.69315
[1139] eval-logloss:0.69315
[1140] eval-logloss:0.69315
[1141] eval-logloss:0.69315
[1142] eval-logloss:0.69315
[1143] eval-logloss:0.69315
[1144] eval-logloss:0.69315
[1145] eval-logloss:0.69315
[1146] eval-logloss:0.69315
[1147] eval-logloss:0.69315
[1148] eval-logloss:0.69315
[1149] eval-logloss:0.69315
[1150] eval-logloss:0.69315

[1151] eval-logloss:0.69315
[1152] eval-logloss:0.69315
[1153] eval-logloss:0.69315
[1154] eval-logloss:0.69315
[1155] eval-logloss:0.69315
[1156] eval-logloss:0.69315
[1157] eval-logloss:0.69315
[1158] eval-logloss:0.69315
[1159] eval-logloss:0.69315
[1160] eval-logloss:0.69315
[1161] eval-logloss:0.69315
[1162] eval-logloss:0.69315
[1163] eval-logloss:0.69315
[1164] eval-logloss:0.69315
[1165] eval-logloss:0.69315
[1166] eval-logloss:0.69315
[1167] eval-logloss:0.69315
[1168] eval-logloss:0.69315
[1169] eval-logloss:0.69315
[1170] eval-logloss:0.69315
[1171] eval-logloss:0.69315
[1172] eval-logloss:0.69315
[1173] eval-logloss:0.69315
[1174] eval-logloss:0.69315
[1175] eval-logloss:0.69315
[1176] eval-logloss:0.69315
[1177] eval-logloss:0.69315
[1178] eval-logloss:0.69315
[1179] eval-logloss:0.69315
[1180] eval-logloss:0.69315
[1181] eval-logloss:0.69315
[1182] eval-logloss:0.69315
[1183] eval-logloss:0.69315
[1184] eval-logloss:0.69315
[1185] eval-logloss:0.69315
[1186] eval-logloss:0.69315
[1187] eval-logloss:0.69315
[1188] eval-logloss:0.69315
[1189] eval-logloss:0.69315
[1190] eval-logloss:0.69315
[1191] eval-logloss:0.69315
[1192] eval-logloss:0.69315
[1193] eval-logloss:0.69315
[1194] eval-logloss:0.69315
[1195] eval-logloss:0.69315
[1196] eval-logloss:0.69315
[1197] eval-logloss:0.69315
[1198] eval-logloss:0.69315

[1199] eval-logloss:0.69315
[1200] eval-logloss:0.69315
[1201] eval-logloss:0.69315
[1202] eval-logloss:0.69315
[1203] eval-logloss:0.69315
[1204] eval-logloss:0.69315
[1205] eval-logloss:0.69315
[1206] eval-logloss:0.69315
[1207] eval-logloss:0.69315
[1208] eval-logloss:0.69315
[1209] eval-logloss:0.69315
[1210] eval-logloss:0.69315
[1211] eval-logloss:0.69315
[1212] eval-logloss:0.69315
[1213] eval-logloss:0.69315
[1214] eval-logloss:0.69315
[1215] eval-logloss:0.69315
[1216] eval-logloss:0.69315
[1217] eval-logloss:0.69315
[1218] eval-logloss:0.69315
[1219] eval-logloss:0.69315
[1220] eval-logloss:0.69315
[1221] eval-logloss:0.69315
[1222] eval-logloss:0.69315
[1223] eval-logloss:0.69315
[1224] eval-logloss:0.69315
[1225] eval-logloss:0.69315
[1226] eval-logloss:0.69315
[1227] eval-logloss:0.69315
[1228] eval-logloss:0.69315
[1229] eval-logloss:0.69315
[1230] eval-logloss:0.69315
[1231] eval-logloss:0.69315
[1232] eval-logloss:0.69315
[1233] eval-logloss:0.69315
[1234] eval-logloss:0.69315
[1235] eval-logloss:0.69315
[1236] eval-logloss:0.69315
[1237] eval-logloss:0.69315
[1238] eval-logloss:0.69315
[1239] eval-logloss:0.69315
[1240] eval-logloss:0.69315
[1241] eval-logloss:0.69315
[1242] eval-logloss:0.69315
[1243] eval-logloss:0.69315
[1244] eval-logloss:0.69315
[1245] eval-logloss:0.69315
[1246] eval-logloss:0.69315

```
[1247] eval-logloss:0.69315
[1248] eval-logloss:0.69315
[1249] eval-logloss:0.69315
[1250] eval-logloss:0.69315
[1251] eval-logloss:0.69315
[1252] eval-logloss:0.69315
[1253] eval-logloss:0.69315
[1254] eval-logloss:0.69315
[1255] eval-logloss:0.69315
[1256] eval-logloss:0.69315
[1257] eval-logloss:0.69315
[1258] eval-logloss:0.69315
[1259] eval-logloss:0.69315
[1260] eval-logloss:0.69315
[1261] eval-logloss:0.69315
[1262] eval-logloss:0.69315
[1263] eval-logloss:0.69315
[1264] eval-logloss:0.69315
[1265] eval-logloss:0.69315
[1266] eval-logloss:0.69315
[1267] eval-logloss:0.69315
[1268] eval-logloss:0.69315
[1269] eval-logloss:0.69315
[1270] eval-logloss:0.69315
[1271] eval-logloss:0.69315
[1272] eval-logloss:0.69315
[1273] eval-logloss:0.69315
[1274] eval-logloss:0.69315
[1275] eval-logloss:0.69315
[1276] eval-logloss:0.69315
[1277] eval-logloss:0.69315
[1278] eval-logloss:0.69315
[1279] eval-logloss:0.69315
[1280] eval-logloss:0.69315
[1281] eval-logloss:0.69315
[1282] eval-logloss:0.69315
[1283] eval-logloss:0.69315
[1284] eval-logloss:0.69315
[1285] eval-logloss:0.69315
[1286] eval-logloss:0.69315
[1287] eval-logloss:0.69315
[1288] eval-logloss:0.69315
[1289] eval-logloss:0.69315
[1290] eval-logloss:0.69315
[1291] eval-logloss:0.69315
[1292] eval-logloss:0.69315
[1293] eval-logloss:0.69315
[1294] eval-logloss:0.69315
```

[1295] eval-logloss:0.69315
[1296] eval-logloss:0.69315
[1297] eval-logloss:0.69315
[1298] eval-logloss:0.69315
[1299] eval-logloss:0.69315
[1300] eval-logloss:0.69315
[1301] eval-logloss:0.69315
[1302] eval-logloss:0.69315
[1303] eval-logloss:0.69315
[1304] eval-logloss:0.69315
[1305] eval-logloss:0.69315
[1306] eval-logloss:0.69315
[1307] eval-logloss:0.69315
[1308] eval-logloss:0.69315
[1309] eval-logloss:0.69315
[1310] eval-logloss:0.69315
[1311] eval-logloss:0.69315
[1312] eval-logloss:0.69315
[1313] eval-logloss:0.69315
[1314] eval-logloss:0.69315
[1315] eval-logloss:0.69315
[1316] eval-logloss:0.69315
[1317] eval-logloss:0.69315
[1318] eval-logloss:0.69315
[1319] eval-logloss:0.69315
[1320] eval-logloss:0.69315
[1321] eval-logloss:0.69315
[1322] eval-logloss:0.69315
[1323] eval-logloss:0.69315
[1324] eval-logloss:0.69315
[1325] eval-logloss:0.69315
[1326] eval-logloss:0.69315
[1327] eval-logloss:0.69315
[1328] eval-logloss:0.69315
[1329] eval-logloss:0.69315
[1330] eval-logloss:0.69315
[1331] eval-logloss:0.69315
[1332] eval-logloss:0.69315
[1333] eval-logloss:0.69315
[1334] eval-logloss:0.69315
[1335] eval-logloss:0.69315
[1336] eval-logloss:0.69315
[1337] eval-logloss:0.69315
[1338] eval-logloss:0.69315
[1339] eval-logloss:0.69315
[1340] eval-logloss:0.69315
[1341] eval-logloss:0.69315
[1342] eval-logloss:0.69315

```
[1343] eval-logloss:0.69315
[1344] eval-logloss:0.69315
[1345] eval-logloss:0.69315
[1346] eval-logloss:0.69315
[1347] eval-logloss:0.69315
[1348] eval-logloss:0.69315
[1349] eval-logloss:0.69315
[1350] eval-logloss:0.69315
[1351] eval-logloss:0.69315
[1352] eval-logloss:0.69315
[1353] eval-logloss:0.69315
[1354] eval-logloss:0.69315
[1355] eval-logloss:0.69315
[1356] eval-logloss:0.69315
[1357] eval-logloss:0.69315
[1358] eval-logloss:0.69315
[1359] eval-logloss:0.69315
[1360] eval-logloss:0.69315
[1361] eval-logloss:0.69315
[1362] eval-logloss:0.69315
[1363] eval-logloss:0.69315
[1364] eval-logloss:0.69315
[1365] eval-logloss:0.69315
[1366] eval-logloss:0.69315
[1367] eval-logloss:0.69315
[1368] eval-logloss:0.69315
[1369] eval-logloss:0.69315
[1370] eval-logloss:0.69315
[1371] eval-logloss:0.69315
[1372] eval-logloss:0.69315
[1373] eval-logloss:0.69315
[1374] eval-logloss:0.69315
[1375] eval-logloss:0.69315
[1376] eval-logloss:0.69315
[1377] eval-logloss:0.69315
[1378] eval-logloss:0.69315
[1379] eval-logloss:0.69315
[1380] eval-logloss:0.69315
[1381] eval-logloss:0.69315
[1382] eval-logloss:0.69315
[1383] eval-logloss:0.69315
[1384] eval-logloss:0.69315
[1385] eval-logloss:0.69315
[1386] eval-logloss:0.69315
[1387] eval-logloss:0.69315
[1388] eval-logloss:0.69315
[1389] eval-logloss:0.69315
[1390] eval-logloss:0.69315
```

[1391] eval-logloss:0.69315
[1392] eval-logloss:0.69315
[1393] eval-logloss:0.69315
[1394] eval-logloss:0.69315
[1395] eval-logloss:0.69315
[1396] eval-logloss:0.69315
[1397] eval-logloss:0.69315
[1398] eval-logloss:0.69315
[1399] eval-logloss:0.69315
[1400] eval-logloss:0.69315
[1401] eval-logloss:0.69315
[1402] eval-logloss:0.69315
[1403] eval-logloss:0.69315
[1404] eval-logloss:0.69315
[1405] eval-logloss:0.69315
[1406] eval-logloss:0.69315
[1407] eval-logloss:0.69315
[1408] eval-logloss:0.69315
[1409] eval-logloss:0.69315
[1410] eval-logloss:0.69315
[1411] eval-logloss:0.69315
[1412] eval-logloss:0.69315
[1413] eval-logloss:0.69315
[1414] eval-logloss:0.69315
[1415] eval-logloss:0.69315
[1416] eval-logloss:0.69315
[1417] eval-logloss:0.69315
[1418] eval-logloss:0.69315
[1419] eval-logloss:0.69315
[1420] eval-logloss:0.69315
[1421] eval-logloss:0.69315
[1422] eval-logloss:0.69315
[1423] eval-logloss:0.69315
[1424] eval-logloss:0.69315
[1425] eval-logloss:0.69315
[1426] eval-logloss:0.69315
[1427] eval-logloss:0.69315
[1428] eval-logloss:0.69315
[1429] eval-logloss:0.69315
[1430] eval-logloss:0.69315
[1431] eval-logloss:0.69315
[1432] eval-logloss:0.69315
[1433] eval-logloss:0.69315
[1434] eval-logloss:0.69315
[1435] eval-logloss:0.69315
[1436] eval-logloss:0.69315
[1437] eval-logloss:0.69315
[1438] eval-logloss:0.69315

```
[1439] eval-logloss:0.69315
[1440] eval-logloss:0.69315
[1441] eval-logloss:0.69315
[1442] eval-logloss:0.69315
[1443] eval-logloss:0.69315
[1444] eval-logloss:0.69315
[1445] eval-logloss:0.69315
[1446] eval-logloss:0.69315
[1447] eval-logloss:0.69315
[1448] eval-logloss:0.69315
[1449] eval-logloss:0.69315
[1450] eval-logloss:0.69315
[1451] eval-logloss:0.69315
[1452] eval-logloss:0.69315
[1453] eval-logloss:0.69315
[1454] eval-logloss:0.69315
[1455] eval-logloss:0.69315
[1456] eval-logloss:0.69315
[1457] eval-logloss:0.69315
[1458] eval-logloss:0.69315
[1459] eval-logloss:0.69315
[1460] eval-logloss:0.69315
[1461] eval-logloss:0.69315
```

Validation Accuracy: 0.1613958560523446

Classification Report:

	precision	recall	f1-score	support
0	0.16	1.00	0.28	148
1	0.00	0.00	0.00	769
accuracy			0.16	917
macro avg	0.08	0.50	0.14	917
weighted avg	0.03	0.16	0.04	917

Predictions saved to submission_xgb_focal_smote_dmatrix.csv

This leads to all predictions swaying towards label 0.


```
In [ ]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from category_encoders import TargetEncoder

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define categorical and numeric columns
categorical_columns = ['subject', 'phase', 'state']
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
preprocessor = ColumnTransformer([
    ('target_encode', TargetEncoder(cols=categorical_columns), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Apply preprocessing to training data
X_train_transformed = preprocessor.fit_transform(X_train, y_train)
X_valid_transformed = preprocessor.transform(X_valid)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)

# Calculate class weights
class_0_weight = 1 # Adjust as needed
class_1_weight = (y_train_balanced.value_counts()[0] / y_train_balanced.value_counts()[1])
```

```
# Define the model parameters
params = {
    'n_estimators': 1462,
    'max_depth': 6,
    'learning_rate': 0.008252647239585268,
    'subsample': 0.8798353150886109,
    'colsample_bytree': 0.9915078582609945,
    'reg_lambda': 0.0021412922470020242,
    'alpha': 1.1851271361097405e-05,
    'scale_pos_weight': class_1_weight, # Adjusted for class imbalance
    'use_label_encoder': False,
    'eval_metric': 'mlogloss'
}

# Train the XGBoost model
model = xgb.XGBClassifier(**params)
model.fit(X_train_balanced, y_train_balanced)

# Validate the model
y_pred = model.predict(X_valid_transformed)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred))
print("Classification Report:\n", classification_report(y_valid, y_pred))

# Prepare test data for prediction
test_data_transformed = preprocessor.transform(test_data)
test_predictions = model.predict(test_data_transformed)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})
submission_df.to_csv('submission_xgb_smote.csv', index=False)

print("Predictions saved to submission_xgb_smote.csv")
```

Validation Accuracy: 0.8669574700109052

Classification Report:

	precision	recall	f1-score	support
0	0.65	0.37	0.47	148
1	0.89	0.96	0.92	769
accuracy			0.87	917
macro avg	0.77	0.67	0.70	917
weighted avg	0.85	0.87	0.85	917

Predictions saved to submission_xgb_smote.csv

Validation Accuracy (Adjusted): 0.8549618320610687

Classification Report (Adjusted):

	precision	recall	f1-score	support
0	0.68	0.19	0.30	148
1	0.86	0.98	0.92	769
accuracy			0.85	917
macro avg	0.77	0.59	0.61	917
weighted avg	0.83	0.85	0.82	917


```
In [ ]: import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import xgboost as xgb
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score
from category_encoders import TargetEncoder

# Load the data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Separate features and target
X = train_data.drop(columns=['output'])
y = train_data['output']

# Ensure categorical columns are of type 'category'
categorical_columns = ['subject', 'phase', 'state']
for col in categorical_columns:
    X[col] = X[col].astype('category')

# Split data for validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Define numeric columns
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Preprocessing Pipeline
preprocessor = ColumnTransformer([
    ('target_encode', TargetEncoder(cols=categorical_columns), categorical_columns),
    ('scaler', StandardScaler(), numeric_columns)
])

# Apply preprocessing to training data
X_train_transformed = preprocessor.fit_transform(X_train, y_train)
X_valid_transformed = preprocessor.transform(X_valid)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_transformed, y_train)
```

```
# Calculate class weights
class_0_weight = 1 # Adjust as needed
class_1_weight = (y_train_balanced.value_counts()[0] / y_train_balanced.value_counts()[1])

# Define the model parameters
params = {
    'n_estimators': 1462,
    'max_depth': 6,
    'learning_rate': 0.008252647239585268,
    'subsample': 0.8798353150886109,
    'colsample_bytree': 0.9915078582609945,
    'reg_lambda': 0.0021412922470020242,
    'alpha': 1.1851271361097405e-05,
    'scale_pos_weight': class_1_weight, # Adjusted for class imbalance
    'use_label_encoder': False,
    'eval_metric': 'mlogloss'
}

# Train the XGBoost model
model = xgb.XGBClassifier(**params)
model.fit(X_train_balanced, y_train_balanced)

# Validate the model
y_pred = model.predict(X_valid_transformed)
print("Validation Accuracy:", accuracy_score(y_valid, y_pred))
print("Classification Report:\n", classification_report(y_valid, y_pred))

# Prepare test data for prediction
test_data[categorical_columns] = test_data[categorical_columns].astype('category') # Ensure correct types
test_data_transformed = preprocessor.transform(test_data)
test_predictions = model.predict(test_data_transformed)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})
submission_df.to_csv('submission_xgb_smote.csv', index=False)

print("Predictions saved to submission_xgb_smote.csv")
```

Validation Accuracy: 0.8680479825517994

Classification Report:

	precision	recall	f1-score	support
0	0.67	0.36	0.47	148
1	0.89	0.96	0.92	769
accuracy			0.87	917
macro avg	0.78	0.66	0.70	917
weighted avg	0.85	0.87	0.85	917

Predictions saved to submission_xgb_smote.csv


```
In [ ]: import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the training and test datasets
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Display the first few rows of the training data to verify
print("Training Data:")
print(train_data.head())

print("\nTest Data:")
print(test_data.head())

# Identify feature columns and target column
feature_columns = [f'x{i}' for i in range(1, 221)] # Feature columns x1 to x220
target_column = 'output' # The target column

# Split the training data into features and target
X = train_data[feature_columns].values
y = train_data[target_column].values

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
X_val_tensor = torch.tensor(X_val, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val, dtype=torch.long)

# Define a simple neural network
class SimpleNN(nn.Module):
    def __init__(self, input_size, num_classes):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, 64) # First hidden layer
        self.fc2 = nn.Linear(64, 32) # Second hidden layer
```

```
    self.fc3 = nn.Linear(32, num_classes) # Output layer

def forward(self, x):
    x = torch.relu(self.fc1(x))           # Activation function
    x = torch.relu(self.fc2(x))
    x = self.fc3(x)
    return x

# Create the model
input_size = X_train.shape[1]
num_classes = len(np.unique(y)) # Assuming y contains class labels
model = SimpleNN(input_size, num_classes)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    model.train()

    # Forward pass
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')

# Validation
model.eval()
with torch.no_grad():
    val_outputs = model(X_val_tensor)
    _, val_predictions = torch.max(val_outputs, 1)
    val_accuracy = accuracy_score(y_val, val_predictions.numpy())
    print(f'Validation Accuracy: {val_accuracy:.4f}')

# Test set evaluation
X_test = test_data[feature_columns].values
X_test_scaled = scaler.transform(X_test) # Standardize test data
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)

model.eval()
```

```
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    _, test_predictions = torch.max(test_outputs, 1)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index, # Using the index of the test data for the 'id' column
    'output': test_predictions.numpy() # Predictions should be converted to a numpy array
})

submission_df.to_csv('submission.csv', index=False)
print("Predictions saved to 'submission.csv'")
```

Training Data:

	x1	x2	x3	x4	x5	x6	x7	x8	x9	\
0	0	0	0	1	-323.106605	2.694366	-1.987520	261.092784	0.013704	
1	0	0	0	1	-376.084691	0.969696	-6.933765	355.311648	0.030292	
2	0	0	0	0	91.955425	2.621643	-2.581162	51.357206	0.036668	
3	0	0	0	1	-391.814586	1.866914	-2.510799	382.900317	0.007947	
4	0	0	0	0	-363.823732	2.951346	-3.726368	330.527539	0.010074	

	x10	...	z217	z218	z219	z220	z221	\
0	0.000100	...	-0.004930	-0.005554	5.246375	-7.534092	3.530736	
1	-0.000153	...	0.022757	0.052506	-3.727741	-2.854443	-0.699268	
2	-0.000104	...	-0.086813	-0.101497	-7.510594	19.564182	-17.008130	
3	-0.000028	...	0.030856	-0.161398	-6.435819	2.174453	-0.153956	
4	-0.000004	...	-0.017226	-0.016454	-2.581403	3.011932	-1.281361	

	z222	subject	phase	state	output
0	-0.539045	K	3	C	1
1	-0.054074	A	4	C	1
2	4.945392	D	3	C	1
3	-0.003958	G	2	C	0
4	0.192647	C	2	C	1

[5 rows x 670 columns]

Test Data:

	x1	x2	x3	x4	x5	x6	x7	x8	x9	\
0	0	0	0	1	4.665949	0.083412	-17.397081	0.067865	0.005895	
1	0	0	0	0	-398.299487	0.784032	-10.785768	403.784446	0.070405	
2	0	0	0	1	-380.902140	0.671324	-9.265829	361.841929	0.006937	
3	0	0	0	1	-342.784124	0.469414	-8.044287	293.043601	0.007217	
4	0	0	0	1	6.248502	0.252336	-5.398056	0.110097	0.005573	

	x10	...	z216	z217	z218	z219	z220	\
0	0.000012	...	-0.017574	-0.004402	0.016508	-540.645279	-1511.001159	
1	0.000180	...	0.029533	0.113692	0.088024	-3.140961	-0.658456	
2	0.000025	...	-0.006361	0.005852	-0.003745	-14.340790	4.182214	
3	-0.000021	...	0.001950	-0.003426	-0.011191	251.659680	-441.129598	
4	0.000010	...	-0.001916	-0.015666	-0.015960	-137.847658	-377.758402	

	z221	z222	subject	phase	state
0	-1408.385976	-437.807290	E	4	D
1	-0.140010	-0.011017	H	4	D
2	-0.549953	0.019553	C	4	B
3	256.986592	-49.745359	H	1	C
4	-345.673251	-105.633150	E	4	B

[5 rows x 669 columns]

```
Epoch [10/100], Loss: 0.5108
Epoch [20/100], Loss: 0.4439
Epoch [30/100], Loss: 0.4312
Epoch [40/100], Loss: 0.4181
Epoch [50/100], Loss: 0.4096
Epoch [60/100], Loss: 0.4025
Epoch [70/100], Loss: 0.3957
Epoch [80/100], Loss: 0.3895
Epoch [90/100], Loss: 0.3833
Epoch [100/100], Loss: 0.3773
Validation Accuracy: 0.8386
Predictions saved to 'submission.csv'
```



```
In [ ]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the datasets
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Define feature and target columns
feature_columns = train_data.columns[:-4] # Assuming the last 4 columns are non-feature columns
target_column = 'output' # Replace with your actual target column name

# Split the training data into features and target
X = train_data[feature_columns].values
y = train_data[target_column].values

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train/test split
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Initialize and fit the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42) # You can tweak the number of estimators
model.fit(X_train, y_train)

# Validate the model
val_predictions = model.predict(X_val)
val_accuracy = accuracy_score(y_val, val_predictions)
print(f'Validation Accuracy: {val_accuracy:.2f}')

# Prepare the test data
X_test = test_data[feature_columns].values
X_test_scaled = scaler.transform(X_test) # Scale test data using the same scaler

# Make predictions on the test set
test_predictions = model.predict(X_test_scaled)

# Save predictions for submission
submission_df = pd.DataFrame({
    'id': test_data.index, # Use the appropriate ID column if available
    'output': test_predictions
})
submission_df.to_csv('submission.csv', index=False)
```

Validation Accuracy: 0.85

Trying out Tpot as automl to find best pipeline

In []: pip install tpot

```
Collecting tpot
  Downloading TPOT-0.12.2-py3-none-any.whl.metadata (2.0 kB)
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.13.1)
Requirement already satisfied: scikit-learn>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.5.2)
Collecting deap>=1.2 (from tpot)
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Collecting update-checker>=0.16 (from tpot)
  Downloading update_checker-0.18.0-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-packages (from tpot) (4.66.5)
Collecting stopit>=1.1.1 (from tpot)
  Downloading stopit-1.1.2.tar.gz (18 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from tpot) (2.2.2)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from tpot) (1.4.2)
Requirement already satisfied: xgboost>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tpot) (2.1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->tpot) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->tpot) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->tpot) (2024.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.1->tpot) (3.5.0)
Requirement already satisfied: requests>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from update-checker>=0.16->tpot) (2.32.3)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost>=1.1.0->tpot) (2.23.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=0.24.2->tpot) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=0.16->tpot) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=0.16->tpot) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=0.16->tpot) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.3.0->update-checker>=0.16->tpot) (2024.8.30)
  Downloading TPOT-0.12.2-py3-none-any.whl (87 kB)
```

87.4/87.4 kB 7.4 MB/s eta 0:00:00

Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13 kB)

14_x86_64.whl (135 kB) 135.4/135.4 kB 9.9 MB/s eta 0:00:00

Downloading update_checker-0.18.0-py3-none-any.whl (7.0 kB)

Building wheels for collected packages: stopit

 Building wheel for stopit (setup.py) ... done

 Created wheel for stopit: filename=stopit-1.1.2-py3-none-any.whl size=11939 sha256=9e22fa339d4cc95dfa6bc334f6d018508be5631080a48fc71dd1307921300971

 Stored in directory: /root/.cache/pip/wheels/af/f9/87/bf5b3d565c2a007b4dae9d8142dccc85a9f164e517062dd519

Successfully built stopit

Installing collected packages: stopit, deap, update-checker, tpot

Successfully installed deap-1.4.1 stopit-1.1.2 tpot-0.12.2 update-checker-0.18.0

```
In [ ]: import pandas as pd
from tpot import TPOTClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load your training data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Check if there are any non-numeric columns
def encode_categorical(df):
    label_encoders = {}
    for column in df.columns:
        if df[column].dtype == 'object':
            le = LabelEncoder()
            df[column] = le.fit_transform(df[column])
            label_encoders[column] = le # Store the encoder for later use if needed
    return df, label_encoders

# Assuming the 'output' column is your target column and the rest are features
X_train = train_data.drop(columns=['output']) # Drop target column
y_train = train_data['output']

# Encode non-numeric columns in X_train and test_data
X_train, train_encoders = encode_categorical(X_train)
test_data, _ = encode_categorical(test_data)

# Split the training data into train/validation sets for TPOT optimization
X_train_part, X_val_part, y_train_part, y_val_part = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Set up TPOT for AutoML (you can adjust the parameters as needed)
tpot = TPOTClassifier(generations=5, population_size=50, verbosity=2, random_state=42)

# Train AutoML to find the best classifier
tpot.fit(X_train_part, y_train_part)

# Evaluate the model on validation data
print("Validation Score: ", tpot.score(X_val_part, y_val_part))

# After optimization, predict on the test set
# X_test = test_data.drop(columns=['id']) # Assuming 'id' is not used for prediction
```

Optimization Progress: 0%

| 0/300 [00:00<?, ?pipeline/s]

Generation 1 – Current best internal CV score: 0.8658303935526799

Generation 2 – Current best internal CV score: 0.8658303935526799

Generation 3 – Current best internal CV score: 0.8677399808929745

Generation 4 – Current best internal CV score: 0.869922419529313

Generation 5 – Current best internal CV score: 0.869922419529313

Best pipeline: ExtraTreesClassifier(CombineDFs(input_matrix, input_matrix), bootstrap=False, criterion=entropy, max_features=0.6500000000000001, min_samples_leaf=11, min_samples_split=2, n_estimators=100)

Validation Score: 0.8767720828789531

```
KeyError Traceback (most recent call last)
<ipython-input-34-eec1a81d66fe> in <cell line: 41>()
    39
    40 # After optimization, predict on the test set
--> 41 X_test = test_data.drop(columns=['id']) # Assuming 'id' is not used for prediction
    42 test_predictions = tpot.predict(X_test)
    43

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in drop(self, labels, axis, index, columns, level, inplace, errors)
    579         weight 1.0      0.8
    580         ....
-> 581     return super().drop(
    582         labels=labels,
    583         axis=axis,

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in drop(self, labels, axis, index, columns, level, inplace, errors)
    4786     for axis, labels in axes.items():
    4787         if labels is not None:
-> 4788             obj = obj._drop_axis(labels, axis, level=level, errors=errors)
    4789
    4790     if inplace:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in _drop_axis(self, labels, axis, level, errors, only_slice)
    4828         new_axis = axis.drop(labels, level=level, errors=errors)
    4829     else:
-> 4830         new_axis = axis.drop(labels, errors=errors)
    4831     indexer = axis.get_indexer(new_axis)
    4832

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
    7068     if mask.any():
    7069         if errors != "ignore":
-> 7070             raise KeyError(f"{labels[mask].tolist()} not found in axis")
    7071         indexer = indexer[~mask]
    7072     return self.delete(indexer)

KeyError: "['id'] not found in axis"
```

```
In [ ]: X_test = test_data  
test_predictions = tpot.predict(X_test)
```

```
In [ ]: # Save the predictions to 'submission.csv' file  
submission = pd.DataFrame({  
    'id': test_data.index, # Assuming there's an 'id' column in test_data  
    'output': test_predictions  
})  
submission.to_csv('submission.csv', index=False)  
  
# Export the best model pipeline  
tpot.export('best_pipeline.py')  
print("Best model pipeline exported to 'best_pipeline.py'")
```

Best model pipeline exported to 'best_pipeline.py'


```
In [ ]: import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, PolynomialFeatures, StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from tpot.export_utils import set_param_recursive

# Load your training and test data
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')

# Encode categorical variables
def encode_categorical(df):
    label_encoders = {}
    for column in df.columns:
        if df[column].dtype == 'object':
            le = LabelEncoder()
            df[column] = le.fit_transform(df[column])
            label_encoders[column] = le
    return df, label_encoders

X_train = train_data.drop(columns=['output'])
y_train = train_data['output']

X_train, train_encoders = encode_categorical(X_train)
test_data, _ = encode_categorical(test_data)

# Check the actual columns in the dataset
print("Training feature columns:", X_train.columns.tolist())

# Example of creating interaction features
# Let's say we want to create interaction features for the first two numerical columns
numerical_features = X_train.select_dtypes(include=[np.number]).columns.tolist()
if len(numerical_features) >= 2:
    X_train['interaction'] = X_train[numerical_features[0]] * X_train[numerical_features[1]]
    test_data['interaction'] = test_data[numerical_features[0]] * test_data[numerical_features[1]]

# Example of polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
test_data_poly = poly.transform(test_data)

# Splitting the data
X_train_part, X_val_part, y_train_part, y_val_part = train_test_split(X_train_poly, y_train, test_size=0.2,
```

```
# Define the pipeline
best_pipeline = make_pipeline(
    StandardScaler(),
    PCA(n_components=50),
    ExtraTreesClassifier(
        bootstrap=False,
        criterion="entropy",
        max_features=0.65,
        min_samples_leaf=11,
        min_samples_split=2,
        n_estimators=100,
        random_state=42
    )
)

# Fit the model
best_pipeline.fit(X_train_part, y_train_part)

# Evaluate the model
val_score = best_pipeline.score(X_val_part, y_val_part)
print("Validation Score: ", val_score)

# Predict on the test set
test_predictions = best_pipeline.predict(test_data_poly)

# Prepare the submission
submission = pd.DataFrame({
    'id': test_data.index,
    'output': test_predictions
})

submission.to_csv('submission.csv', index=False)
```

Training feature columns: ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19', 'x20', 'x21', 'x22', 'x23', 'x24', 'x25', 'x26', 'x27', 'x28', 'x29', 'x30', 'x31', 'x32', 'x33', 'x34', 'x35', 'x36', 'x37', 'x38', 'x39', 'x40', 'x41', 'x42', 'x43', 'x44', 'x45', 'x46', 'x47', 'x48', 'x49', 'x50', 'x51', 'x52', 'x53', 'x54', 'x55', 'x56', 'x57', 'x58', 'x59', 'x60', 'x61', 'x62', 'x63', 'x64', 'x65', 'x66', 'x67', 'x68', 'x69', 'x70', 'x71', 'x72', 'x73', 'x74', 'x75', 'x76', 'x77', 'x78', 'x79', 'x80', 'x81', 'x82', 'x83', 'x84', 'x85', 'x86', 'x87', 'x88', 'x89', 'x90', 'x91', 'x92', 'x93', 'x94', 'x95', 'x96', 'x97', 'x98', 'x99', 'x100', 'x101', 'x102', 'x103', 'x104', 'x105', 'x106', 'x107', 'x108', 'x109', 'x110', 'x111', 'x112', 'x113', 'x114', 'x115', 'x116', 'x117', 'x118', 'x119', 'x120', 'x121', 'x122', 'x123', 'x124', 'x125', 'x126', 'x127', 'x128', 'x129', 'x130', 'x131', 'x132', 'x133', 'x134', 'x135', 'x136', 'x137', 'x138', 'x139', 'x140', 'x141', 'x142', 'x143', 'x144', 'x145', 'x146', 'x147', 'x148', 'x149', 'x150', 'x151', 'x152', 'x153', 'x154', 'x155', 'x156', 'x157', 'x158', 'x159', 'x160', 'x161', 'x162', 'x163', 'x164', 'x165', 'x166', 'x167', 'x168', 'x169', 'x170', 'x171', 'x172', 'x173', 'x174', 'x175', 'x176', 'x177', 'x178', 'x179', 'x180', 'x181', 'x182', 'x183', 'x184', 'x185', 'x186', 'x187', 'x188', 'x189', 'x190', 'x191', 'x192', 'x193', 'x194', 'x195', 'x196', 'x197', 'x198', 'x199', 'x200', 'x201', 'x202', 'x203', 'x204', 'x205', 'x206', 'x207', 'x208', 'x209', 'x210', 'x211', 'x212', 'x213', 'x214', 'x215', 'x216', 'x217', 'x218', 'x219', 'x220', 'x221', 'x222', 'y1', 'y2', 'y3', 'y4', 'y5', 'y6', 'y7', 'y8', 'y9', 'y10', 'y11', 'y12', 'y13', 'y14', 'y15', 'y16', 'y17', 'y18', 'y19', 'y20', 'y21', 'y22', 'y23', 'y24', 'y25', 'y26', 'y27', 'y28', 'y29', 'y30', 'y31', 'y32', 'y33', 'y34', 'y35', 'y36', 'y37', 'y38', 'y39', 'y40', 'y41', 'y42', 'y43', 'y44', 'y45', 'y46', 'y47', 'y48', 'y49', 'y50', 'y51', 'y52', 'y53', 'y54', 'y55', 'y56', 'y57', 'y58', 'y59', 'y60', 'y61', 'y62', 'y63', 'y64', 'y65', 'y66', 'y67', 'y68', 'y69', 'y70', 'y71', 'y72', 'y73', 'y74', 'y75', 'y76', 'y77', 'y78', 'y79', 'y80', 'y81', 'y82', 'y83', 'y84', 'y85', 'y86', 'y87', 'y88', 'y89', 'y90', 'y91', 'y92', 'y93', 'y94', 'y95', 'y96', 'y97', 'y98', 'y99', 'y100', 'y101', 'y102', 'y103', 'y104', 'y105', 'y106', 'y107', 'y108', 'y109', 'y110', 'y111', 'y112', 'y113', 'y114', 'y115', 'y116', 'y117', 'y118', 'y119', 'y120', 'y121', 'y122', 'y123', 'y124', 'y125', 'y126', 'y127', 'y128', 'y129', 'y130', 'y131', 'y132', 'y133', 'y134', 'y135', 'y136', 'y137', 'y138', 'y139', 'y140', 'y141', 'y142', 'y143', 'y144', 'y145', 'y146', 'y147', 'y148', 'y149', 'y150', 'y151', 'y152', 'y153', 'y154', 'y155', 'y156', 'y157', 'y158', 'y159', 'y160', 'y161', 'y162', 'y163', 'y164', 'y165', 'y166', 'y167', 'y168', 'y169', 'y170', 'y171', 'y172', 'y173', 'y174', 'y175', 'y176', 'y177', 'y178', 'y179', 'y180', 'y181', 'y182', 'y183', 'y184', 'y185', 'y186', 'y187', 'y188', 'y189', 'y190', 'y191', 'y192', 'y193', 'y194', 'y195', 'y196', 'y197', 'y198', 'y199', 'y200', 'y201', 'y202', 'y203', 'y204', 'y205', 'y206', 'y207', 'y208', 'y209', 'y210', 'y211', 'y212', 'y213', 'y214', 'y215', 'y216', 'y217', 'y218', 'y219', 'y220', 'y221', 'y222', 'z1', 'z2', 'z3', 'z4', 'z5', 'z6', 'z7', 'z8', 'z9', 'z10', 'z11', 'z12', 'z13', 'z14', 'z15', 'z16', 'z17', 'z18', 'z19', 'z20', 'z21', 'z22', 'z23', 'z24', 'z25', 'z26', 'z27', 'z28', 'z29', 'z30', 'z31', 'z32', 'z33', 'z34', 'z35', 'z36', 'z37', 'z38', 'z39', 'z40', 'z41', 'z42', 'z43', 'z44', 'z45', 'z46', 'z47', 'z48', 'z49', 'z50', 'z51', 'z52', 'z53', 'z54', 'z55', 'z56', 'z57', 'z58', 'z59', 'z60', 'z61', 'z62', 'z63', 'z64', 'z65', 'z66', 'z67', 'z68', 'z69', 'z70', 'z71', 'z72', 'z73', 'z74', 'z75', 'z76', 'z77', 'z78', 'z79', 'z80', 'z81', 'z82', 'z83', 'z84', 'z85', 'z86', 'z87', 'z88', 'z89', 'z90', 'z91', 'z92', 'z93', 'z94', 'z95', 'z96', 'z97', 'z98', 'z99', 'z100', 'z101', 'z102', 'z103', 'z104', 'z105', 'z106', 'z107', 'z108', 'z109', 'z110', 'z111', 'z112', 'z113', 'z114', 'z115', 'z116', 'z117', 'z118', 'z119', 'z120', 'z121', 'z122', 'z123', 'z124', 'z125', 'z126', 'z127', 'z128', 'z129', 'z130', 'z131', 'z132', 'z133', 'z134', 'z135', 'z136', 'z137', 'z138', 'z139', 'z140', 'z141', 'z142', 'z143', 'z144', 'z145', 'z146', 'z147', 'z148', 'z149', 'z150', 'z151', 'z152', 'z153', 'z154', 'z155', 'z156', 'z157', 'z158', 'z159', 'z160', 'z161', 'z162', 'z163', 'z164', 'z165', 'z166', 'z167', 'z168', 'z169', 'z170', 'z171', 'z172', 'z173', 'z174', 'z175', 'z176', 'z177', 'z178', 'z179', 'z180', 'z181', 'z182', 'z183', 'z184', 'z185', 'z186', 'z187', 'z188', 'z189', 'z190', 'z191', 'z192', 'z193', 'z194', 'z195', 'z196', 'z197', 'z198', 'z199', 'z200', 'z201', 'z202', 'z203', 'z204', 'z205', 'z206', 'z207', 'z208', 'z209', 'z210', 'z211', 'z212', 'z213', 'z214', 'z215', 'z216', 'z217', 'z218']

```
8', 'z219', 'z220', 'z221', 'z222', 'subject', 'phase', 'state']
Validation Score: 0.8440567066521265
```

This does not improve test accuracy even though TPOT runs for over 300 iterations and for 3 hours

Final Summary

Tried various techniques. The best performance with test accuracy of 61% I got was via xgboost and SMOTE. Many other techniques seem to overfit on training data and thereby we need to do more with undersampling majority, oversampling minority to make model better.

Brian's Section

Data Pre-Processing

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

```
In [3]: df = pd.read_csv('../train_data.csv')
df_test = pd.read_csv('../test_data.csv')
df.head()
```

```
Out[3]:
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	...	z217	z218	z219	z220	z221	
0	0	0	0	1	-323.106605	2.694366	-1.987520	261.092784	0.013704	0.000100	...	-0.004930	-0.005554	5.246375	-7.534092	3.530736	-1
1	0	0	0	1	-376.084691	0.969696	-6.933765	355.311648	0.030292	-0.000153	...	0.022757	0.052506	-3.727741	-2.854443	-0.699268	-1
2	0	0	0	0	91.955425	2.621643	-2.581162	51.357206	0.036668	-0.000104	...	-0.086813	-0.101497	-7.510594	19.564182	-17.008130	-1
3	0	0	0	1	-391.814586	1.866914	-2.510799	382.900317	0.007947	-0.000028	...	0.030856	-0.161398	-6.435819	2.174453	-0.153956	-1
4	0	0	0	0	-363.823732	2.951346	-3.726368	330.527539	0.010074	-0.000004	...	-0.017226	-0.016454	-2.581403	3.011932	-1.281361	1

5 rows × 670 columns

```
In [4]: # print the shapes of the data
df_test.shape, df.shape
```

```
Out[4]: ((1732, 669), (4584, 670))
```

```
In [5]: # check for missing data and NaN values
for col in df.columns:
    if df[col].isna().any():
        print(col)

print('No Missing Data in Training Set')
```

No Missing Data in Training Set

```
In [6]: # find the value counts for the target variable
print(df['output'].value_counts(normalize=True) * 100)
```

```
output
1    84.489529
0    15.510471
Name: proportion, dtype: float64
```

```
In [7]: # Create DataFrames for subject and state counts
train_subject = df['subject'].value_counts().reset_index()
train_state = df['state'].value_counts().reset_index()
test_subject = df_test['subject'].value_counts().reset_index()
test_state = df_test['state'].value_counts().reset_index()

# Rename columns
train_subject.columns = ['Subject', 'Train Count']
train_state.columns = ['State', 'Train Count']
test_subject.columns = ['Subject', 'Test Count']
test_state.columns = ['State', 'Test Count']

# Merge DataFrames
subject_table = pd.merge(train_subject, test_subject, on='Subject', how='outer').fillna(0)
state_table = pd.merge(train_state, test_state, on='State', how='outer').fillna(0)

# Convert counts to integers
subject_table[['Train Count', 'Test Count']] = subject_table[['Train Count', 'Test Count']].astype(int)
state_table[['Train Count', 'Test Count']] = state_table[['Train Count', 'Test Count']].astype(int)

print("Subject Counts:")
print(subject_table.to_string(index=False))
print("\nState Counts:")
print(state_table.to_string(index=False))
```

Subject Counts:

Subject	Train Count	Test Count
A	380	59
B	513	78
C	361	71
D	420	83
E	0	312
F	330	60
G	518	88
H	181	33
I	663	124
J	0	629
K	468	69
L	444	71
M	306	55

State Counts:

State	Train Count	Test Count
A	398	112
B	1502	525
C	2106	841
D	576	253
E	2	1

Quick Observations

1. The dataset has 4584 rows and 670 columns.
2. The target variable is 'output' which has two classes: 0 and 1.
3. The dataset is imbalanced with respect to the target variable.
4. The dataset has no missing values.
5. The state variable seems a somewhat imbalanced, with state E having the fewest observations.
6. Disproportionate state E.

```
In [8]: print(f"In the train set: {sorted(df['subject'].unique())}")
print(f"In the test set: {sorted(df_test['subject'].unique())}")
```

```
In the train set: ['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'K', 'L', 'M']
In the test set: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M']
```

There are some subjects in the test set that are not in the train set. We will be mapping unseen categories (e.g., subjects or states) to an 'unknown' category in both your training and test sets. This allows the model to handle unseen categories as a special case rather than failing outright.

```
In [10]: print(f"States In the train set: {sorted(df['state'].unique())}")
print(f"States In the test set: {sorted(df_test['state'].unique())}")

States In the train set: ['A', 'B', 'C', 'D', 'E']
States In the test set: ['A', 'B', 'C', 'D', 'E']
```

```
In [11]: # Convert subjects and states to string type
df['subject'] = df['subject'].astype(str)
df_test['subject'] = df_test['subject'].astype(str)

# Handle unseen subjects in the test set by assigning them to 'unknown'
df_test['subject'] = df_test['subject'].apply(lambda x: x if x in df['subject'].unique() else 'unknown')

# Similarly, for states (if necessary)
df_test['state'] = df_test['state'].apply(lambda x: x if x in df['state'].unique() else 'unknown')
```

```
In [12]: from sklearn.preprocessing import OneHotEncoder

# Define the categorical columns
categorical_features = ['state', 'subject']
ordinal_features = ['phase']
target_feature = ['output']

# numerical features
numerical_features = [col for col in df.columns if col not in categorical_features + ordinal_features + target_feature]

# Set up the encoder. OneHotEncoder using ha
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)

# Fit on the training data
X_train_encoded = encoder.fit_transform(df[categorical_features])

# Apply the same transformation to the test set
X_test_encoded = encoder.transform(df_test[categorical_features])
```

```
In [13]: X_train_encoded.shape, X_test_encoded.shape
```

```
Out[13]: ((4584, 16), (1732, 16))
```

```
In [14]: # Get the new column names after encoding
encoded_columns = encoder.get_feature_names_out(categorical_features)

# Convert the encoded data back into a DataFrame to preserve column names
X_train_encoded_df = pd.DataFrame(X_train_encoded, columns=encoded_columns, index=df.index)
X_test_encoded_df = pd.DataFrame(X_test_encoded, columns=encoded_columns, index=df_test.index)

# concatenate these encoded features back with the original data (excluding the original categorical columns)
X_train_final = pd.concat([df.drop(columns=categorical_features), X_train_encoded_df], axis=1)
X_test_final = pd.concat([df_test.drop(columns=categorical_features), X_test_encoded_df], axis=1)

X_train_final.shape, X_test_final.shape
```

```
Out[14]: ((4584, 684), (1732, 683))
```

Question: Which Data is Sparse?

```
In [15]: def compare_zero_proportions(train_df, test_df, threshold):
    """
    Compares the proportion of zeros between the train and test datasets
    and returns a list of columns where both datasets have sparse values
    above the given threshold.
    """
    train_zero_proportion = train_df.eq(0).sum() / train_df.shape[0]
    test_zero_proportion = test_df.eq(0).sum() / test_df.shape[0]

    # Find columns where both the train and test sets have proportions of zeros above the threshold
    sparse_columns = train_zero_proportion[train_zero_proportion > threshold].index
    sparse_in_test = test_zero_proportion[sparse_columns][test_zero_proportion[sparse_columns] > threshold].index

    return list(sparse_in_test)
```

```
In [16]: # using a threshold of 0.99
sparse_columns_to_drop = compare_zero_proportions(X_train_final[numerical_features], df_test[numerical_features])

print("Columns to drop:", sparse_columns_to_drop)
print(f" We are dropping {len(sparse_columns_to_drop)} columns")

df_train_dropped = X_train_final.drop(columns=sparse_columns_to_drop, axis=1)
df_test_dropped = X_test_final.drop(columns=sparse_columns_to_drop, axis=1)

# print shapes before and after dropping
print(f"Shape of the training dataframe before dropping: {df.shape}")
print(f"Shape of the training dataframe after dropping: {df_train_dropped.shape}")
print(f"Shape of the testing dataframe before dropping: {df_test.shape}")
print(f"Shape of the testing dataframe after dropping: {df_test_dropped.shape}")

# check that the remaining columns are the same, except for the target variable
train_columns = set(df_train_dropped.columns) - {'output'}
test_columns = set(df_test_dropped.columns)

# Ensure that both training and test sets have the same columns
assert train_columns == test_columns, "Training and test columns do not match"
```

```
Columns to drop: ['x1', 'x2', 'x3', 'x46', 'x47', 'x48', 'x81', 'x86', 'x87', 'x91', 'x92', 'x93', 'x96', 'x97', 'x98', 'x99', 'x101', 'x102', 'x103', 'x104', 'x105', 'x113', 'y1', 'y2', 'y3', 'y46', 'y47', 'y48', 'y81', 'y86', 'y87', 'y91', 'y92', 'y93', 'y96', 'y97', 'y98', 'y99', 'y102', 'y103', 'y104', 'y105', 'y113', 'z1', 'z2', 'z3', 'z46', 'z47', 'z48', 'z81', 'z86', 'z87', 'z91', 'z92', 'z93', 'z96', 'z97', 'z98', 'z99', 'z101', 'z102', 'z103', 'z104', 'z105', 'z113']
We are dropping 65 columns
Shape of the training dataframe before dropping: (4584, 670)
Shape of the training dataframe after dropping: (4584, 619)
Shape of the testing dataframe before dropping: (1732, 669)
Shape of the testing dataframe after dropping: (1732, 618)
```

```
In [17]: df_train_dropped.to_csv('train_data_processed.csv', index=False)
df_test_dropped.to_csv('test_data_processed.csv', index=False)
```

Quick Note: The saved dataframes are processed dataframes with the sparse columns removed. We have not done any feature scaling or normalization at this point. We also handled the subjects in the test set that are not in the train set.

```
In [18]: df_train_dropped.shape, df_test_dropped.shape
```

```
Out[18]: ((4584, 619), (1732, 618))
```

Model Building

```
In [19]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

```
In [20]: # load the pre-processed data
df = pd.read_csv('train_data_processed.csv')
df_test = pd.read_csv('test_data_processed.csv')

df.head(1)
```

```
Out[20]:
```

	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	...	subject_B	subject_C	subject_D	subject
0	1	-323.106605	2.694366	-1.98752	261.092784	0.013704	0.0001	-0.000009	-0.806753	-0.805752	...	0.0	0.0	0.0	(

1 rows × 619 columns

```
In [21]: numerical_columns = [col for col in df.columns if not any(keyword in col for keyword in ['subject', 'state',  
print(f"Numerical columns identified by name: {numerical_columns}")
```

Numerical columns identified by name: ['x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x19', 'x20', 'x21', 'x22', 'x23', 'x24', 'x25', 'x26', 'x27', 'x28', 'x29', 'x30', 'x31', 'x32', 'x33', 'x34', 'x35', 'x36', 'x37', 'x38', 'x39', 'x40', 'x41', 'x42', 'x43', 'x44', 'x45', 'x46', 'x47', 'x48', 'x49', 'x50', 'x51', 'x52', 'x53', 'x54', 'x55', 'x56', 'x57', 'x58', 'x59', 'x60', 'x61', 'x62', 'x63', 'x64', 'x65', 'x66', 'x67', 'x68', 'x69', 'x70', 'x71', 'x72', 'x73', 'x74', 'x75', 'x76', 'x77', 'x78', 'x79', 'x80', 'x81', 'x82', 'x83', 'x84', 'x85', 'x86', 'x87', 'x88', 'x89', 'x90', 'x91', 'x92', 'x93', 'x94', 'x95', 'x96', 'x97', 'x98', 'x99', 'x100', 'x101', 'x102', 'x103', 'x104', 'x105', 'x106', 'x107', 'x108', 'x109', 'x110', 'x111', 'x112', 'x113', 'x114', 'x115', 'x116', 'x117', 'x118', 'x119', 'x120', 'x121', 'x122', 'x123', 'x124', 'x125', 'x126', 'x127', 'x128', 'x129', 'x130', 'x131', 'x132', 'x133', 'x134', 'x135', 'x136', 'x137', 'x138', 'x139', 'x140', 'x141', 'x142', 'x143', 'x144', 'x145', 'x146', 'x147', 'x148', 'x149', 'x150', 'x151', 'x152', 'x153', 'x154', 'x155', 'x156', 'x157', 'x158', 'x159', 'x160', 'x161', 'x162', 'x163', 'x164', 'x165', 'x166', 'x167', 'x168', 'x169', 'x170', 'x171', 'x172', 'x173', 'x174', 'x175', 'x176', 'x177', 'x178', 'x179', 'x180', 'x181', 'x182', 'x183', 'x184', 'x185', 'x186', 'x187', 'x188', 'x189', 'x190', 'x191', 'x192', 'x193', 'x194', 'x195', 'x196', 'x197', 'x198', 'x199', 'x200', 'x201', 'x202', 'x203', 'x204', 'x205', 'x206', 'x207', 'x208', 'x209', 'x210', 'x211', 'x212', 'x213', 'x214', 'x215', 'x216', 'x217', 'x218', 'x219', 'x220', 'x221', 'x222', 'y4', 'y5', 'y6', 'y7', 'y8', 'y9', 'y10', 'y11', 'y12', 'y13', 'y14', 'y15', 'y16', 'y17', 'y18', 'y19', 'y20', 'y21', 'y22', 'y23', 'y24', 'y25', 'y26', 'y27', 'y28', 'y29', 'y30', 'y31', 'y32', 'y33', 'y34', 'y35', 'y36', 'y37', 'y38', 'y39', 'y40', 'y41', 'y42', 'y43', 'y44', 'y45', 'y46', 'y47', 'y48', 'y49', 'y50', 'y51', 'y52', 'y53', 'y54', 'y55', 'y56', 'y57', 'y58', 'y59', 'y60', 'y61', 'y62', 'y63', 'y64', 'y65', 'y66', 'y67', 'y68', 'y69', 'y70', 'y71', 'y72', 'y73', 'y74', 'y75', 'y76', 'y77', 'y78', 'y79', 'y80', 'y81', 'y82', 'y83', 'y84', 'y85', 'y86', 'y87', 'y88', 'y89', 'y90', 'y91', 'y92', 'y93', 'y94', 'y95', 'y96', 'y97', 'y98', 'y99', 'y100', 'y101', 'y102', 'y103', 'y104', 'y105', 'y106', 'y107', 'y108', 'y109', 'y110', 'y111', 'y112', 'y113', 'y114', 'y115', 'y116', 'y117', 'y118', 'y119', 'y120', 'y121', 'y122', 'y123', 'y124', 'y125', 'y126', 'y127', 'y128', 'y129', 'y130', 'y131', 'y132', 'y133', 'y134', 'y135', 'y136', 'y137', 'y138', 'y139', 'y140', 'y141', 'y142', 'y143', 'y144', 'y145', 'y146', 'y147', 'y148', 'y149', 'y150', 'y151', 'y152', 'y153', 'y154', 'y155', 'y156', 'y157', 'y158', 'y159', 'y160', 'y161', 'y162', 'y163', 'y164', 'y165', 'y166', 'y167', 'y168', 'y169', 'y170', 'y171', 'y172', 'y173', 'y174', 'y175', 'y176', 'y177', 'y178', 'y179', 'y180', 'y181', 'y182', 'y183', 'y184', 'y185', 'y186', 'y187', 'y188', 'y189', 'y190', 'y191', 'y192', 'y193', 'y194', 'y195', 'y196', 'y197', 'y198', 'y199', 'y200', 'y201', 'y202', 'y203', 'y204', 'y205', 'y206', 'y207', 'y208', 'y209', 'y210', 'y211', 'y212', 'y213', 'y214', 'y215', 'y216', 'y217', 'y218', 'y219', 'y220', 'y221', 'y222', 'z4', 'z5', 'z6', 'z7', 'z8', 'z9', 'z10', 'z11', 'z12', 'z13', 'z14', 'z15', 'z16', 'z17', 'z18', 'z19', 'z20', 'z21', 'z22', 'z23', 'z24', 'z25', 'z26', 'z27', 'z28', 'z29', 'z30', 'z31', 'z32', 'z33', 'z34', 'z35', 'z36', 'z37', 'z38', 'z39', 'z40', 'z41', 'z42', 'z43', 'z44', 'z45', 'z46', 'z47', 'z48', 'z49', 'z50', 'z51', 'z52', 'z53', 'z54', 'z55', 'z56', 'z57', 'z58', 'z59', 'z60', 'z61', 'z62', 'z63', 'z64', 'z65', 'z66', 'z67', 'z68', 'z69', 'z70', 'z71', 'z72', 'z73', 'z74', 'z75', 'z76', 'z77', 'z78', 'z79', 'z80', 'z81', 'z82', 'z83', 'z84', 'z85', 'z86', 'z87', 'z88', 'z89', 'z90', 'z91', 'z92', 'z93', 'z94', 'z95', 'z96', 'z97', 'z98', 'z99', 'z100', 'z101', 'z102', 'z103', 'z104', 'z105', 'z106', 'z107', 'z108', 'z109', 'z110', 'z111', 'z112', 'z113', 'z114', 'z115', 'z116', 'z117', 'z118', 'z119', 'z120', 'z121', 'z122', 'z123', 'z124', 'z125', 'z126', 'z127', 'z128', 'z129', 'z130', 'z131', 'z132', 'z133', 'z134', 'z135', 'z136', 'z137', 'z138', 'z139', 'z140', 'z141', 'z142', 'z143', 'z144', 'z145', 'z146', 'z147', 'z148', 'z149', 'z150', 'z151', 'z152', 'z153', 'z154', 'z155', 'z156', 'z157', 'z158', 'z159', 'z160', 'z161', 'z162', 'z163', 'z164', 'z165', 'z166', 'z167', 'z168', 'z169', 'z170', 'z171', 'z172', 'z173', 'z174', 'z175', 'z176', 'z177', 'z178', 'z179', 'z180', 'z181', 'z182', 'z183', 'z184', 'z185', 'z186', 'z187', 'z188', 'z189', 'z190', 'z191', 'z192', 'z193', 'z194', 'z195', 'z196', 'z197', 'z198', 'z199', 'z200', 'z201', 'z202', 'z203', 'z204', 'z205', 'z206', 'z207', 'z208', 'z209', 'z210', 'z211', 'z212', 'z213', 'z214', 'z215', 'z216', 'z217', 'z218', 'z219', 'z220', 'z221', 'z222']

Model 0: Logistic Regression

Step 1: Pre-process the data

This steps includes:

1. Standardizing the features
2. Handling the ordinal feature phase

In [22]: `def preprocess_dataframe(df, is_train=True):`

```
    """
    This function preprocesses the dataframe
    by performing feature normalization,
    and handling ordinal features.
    """
    df = df.copy()

    ordinal_features = ['phase']
    target_feature = ['output']

    # numerical features
    numerical_features = [col for col in df.columns if not any(keyword in col for keyword in ['subject', 'st'])

    # Perform feature Normalization for numerical features
    scaler = StandardScaler()
    df[numerical_features] = scaler.fit_transform(df[numerical_features])

    # handle ordinal features
    df['phase'] = df['phase'].astype(int)

    # print the shape of the dataframe after preprocessing
    if is_train:
        print(f"Shape of the training dataframe after preprocessing: {df.shape}")
    else:
        print(f"Shape of the testing dataframe after preprocessing: {df.shape}")

    return df

# apply to the dataframes
df_train = preprocess_dataframe(df, is_train=True)
df_test_final = preprocess_dataframe(df_test, is_train=False)
```

Shape of the training dataframe after preprocessing: (4584, 619)
Shape of the testing dataframe after preprocessing: (1732, 618)

Step 2: Handling class imbalance

The dataset has 84.5% of observations belonging to `class=1` vs 15.5% for `class=0`.

Techniques to use:

Class Weights: Assign higher weights to the minority class.

```
In [23]: from sklearn.utils import class_weight
import numpy as np

# Calculate class weights
classes = np.unique(df_train['output'])
class_weights = class_weight.compute_class_weight('balanced',
                                                    classes=classes,
                                                    y=df_train['output'])
class_weights_dict = dict(zip(classes, class_weights))
print("Class Weights:", class_weights_dict)
```

Class Weights: {0: 3.2236286919831225, 1: 0.5917893106119287}

Splitting the data into training and validation sets

```
In [24]: from sklearn.model_selection import train_test_split

X = df_train.drop('output', axis=1)
y = df_train['output']

X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Train shape: {X_train.shape}, Validation shape: {X_val.shape}")
```

Train shape: (3208, 618), Validation shape: (1376, 618)

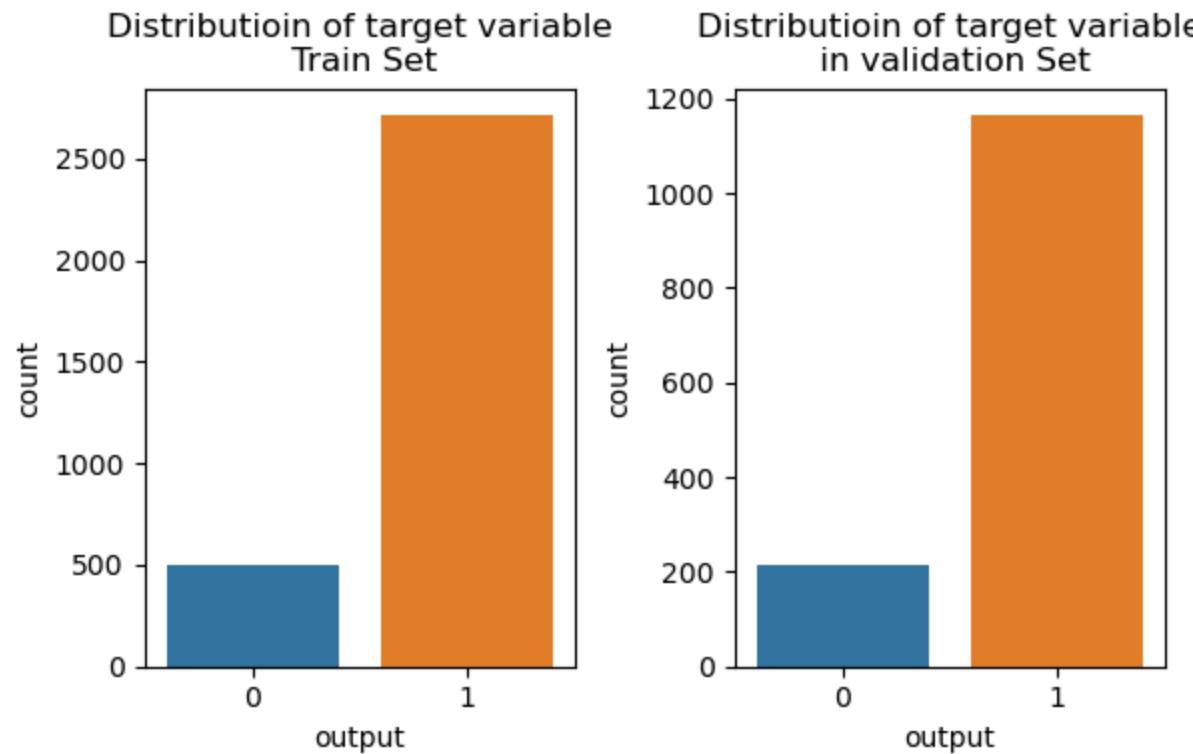
plot the distribution of the target variable in the training, validation, and test sets

```
In [26]: # plot the distribution of the target variable in the training, validation, and test sets
plt.figure(figsize=(9, 4))

plt.subplot(1, 3, 1)
sns.countplot(x='output', data=pd.DataFrame({'output': y_train}))
plt.title('Distribution of target variable\n Train Set')

plt.subplot(1, 3, 2)
sns.countplot(x='output', data=pd.DataFrame({'output': y_val}))
plt.title('Distribution of target variable\n in validation Set')

plt.tight_layout()
plt.show()
```



Logistic Regression with Regularization

```
In [27]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Define the base model
logreg = LogisticRegression(
    class_weight=class_weights_dict,
    random_state=42,
    max_iter=10000
)

# Define hyperparameters for regularization
param_grid = {
    'penalty': ['l1'],
    'C': [0.01, 0.1, 0.5, 1, 10, 100],
    'solver': ['saga']
}

# Setup GridSearch with cross-validation
grid_search_logreg = GridSearchCV(
    logreg,
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=2
)

# Fit the model
grid_search_logreg.fit(X_train, y_train)

print(f"Best parameters: {grid_search_logreg.best_params_}")
print(f"Best ROC AUC: {grid_search_logreg.best_score_}")
```

Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best parameters: {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
Best ROC AUC: 0.7462638692463587

```
In [28]: # Extract results from grid search
results = grid_search_logreg.cv_results_ # returns a dictionary
```

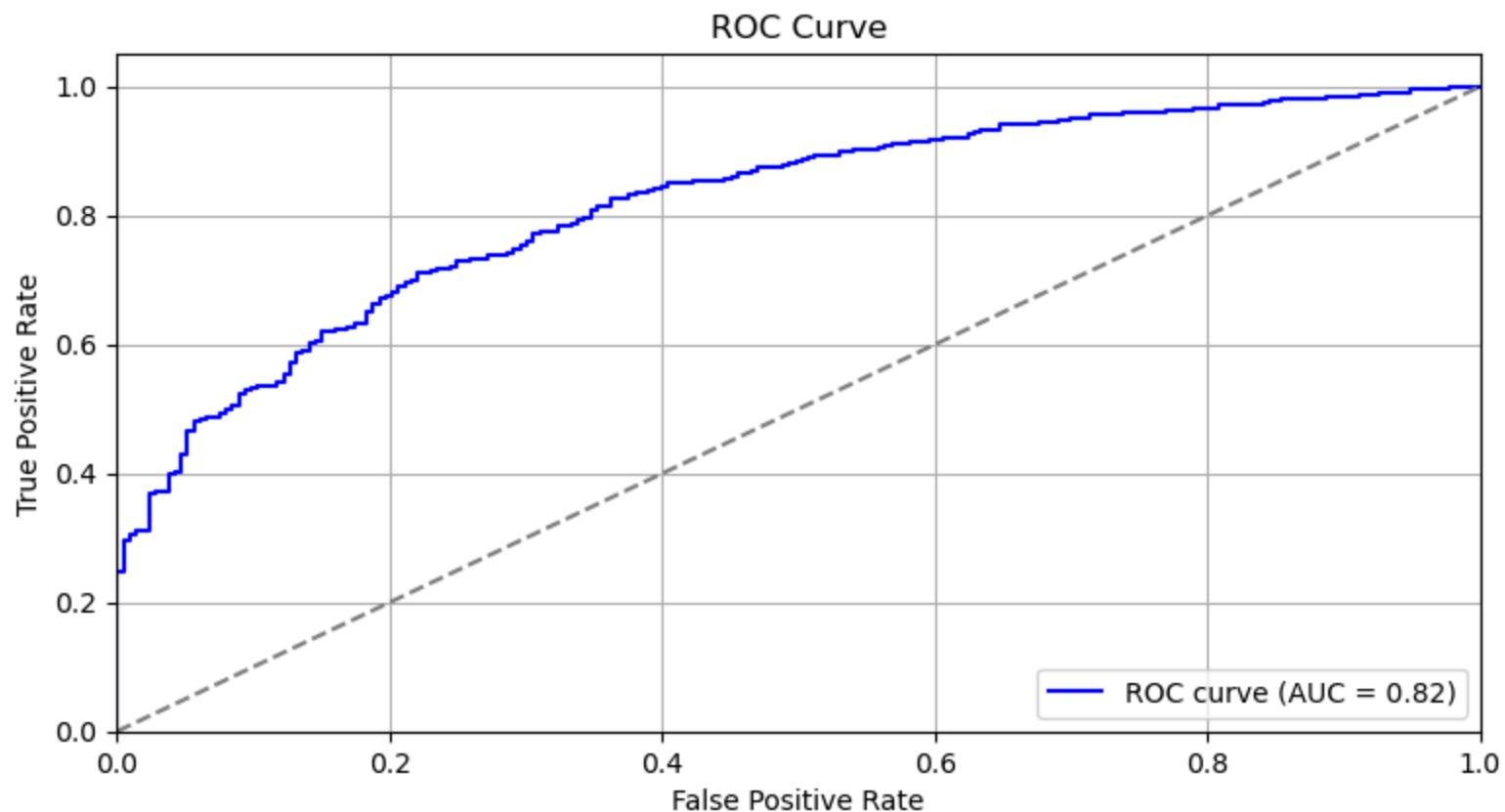
```
In [29]: from sklearn.metrics import roc_curve, auc

# Get the best model
best_model = grid_search_logreg.best_estimator_

# Predict probabilities
y_val_probs = best_model.predict_proba(X_val)[:, 1] # Get the probabilities for the positive class

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_val, y_val_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(9, 4.5))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--') # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Evaluating the Model's Performance

```
In [30]: from sklearn.metrics import roc_auc_score, classification_report, confusion_matrix

roc_auc = roc_auc_score(y_val, y_val_probs)
print(f"Validation ROC AUC: {roc_auc}")

# Predict classes
y_val_pred = best_model.predict(X_val)

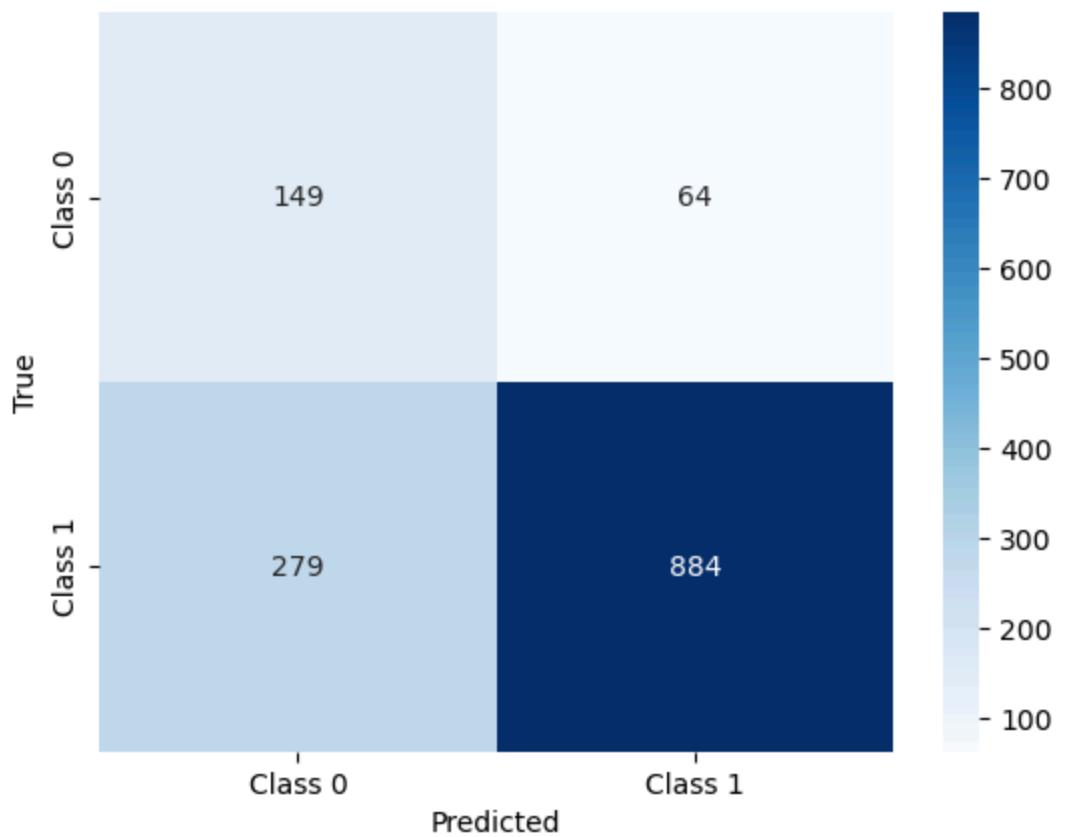
# Classification report
print(classification_report(y_val, y_val_pred))

# plot the confusion matrix
conf_matrix = confusion_matrix(y_val, y_val_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'], yticklabels=[],
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Validation ROC AUC: 0.8172082076869357

	precision	recall	f1-score	support
0	0.35	0.70	0.46	213
1	0.93	0.76	0.84	1163
accuracy			0.75	1376
macro avg	0.64	0.73	0.65	1376
weighted avg	0.84	0.75	0.78	1376

Confusion Matrix



Submission Attempt 1 with Baseline Regularized Logistic (Kaggle Score: 0.61132)

In [31]:

```
# make predictions on the test set
y_test_pred = best_model.predict(df_test_final)
y_test_pred

# create a submission file
submission = pd.DataFrame({
    'id': df_test_final.index,
    'output': y_test_pred
})

assert submission.shape[0] == df_test_final.shape[0]

# # save the submission file to a csv
submission.to_csv('submission_Baseline_Logistic_Regularized_2.csv', index=False)
```

```
[CV] END ..... C=0.01, penalty=l1, solver=saga; total time= 10.3s
[CV] END ..... C=0.5, penalty=l1, solver=saga; total time= 40.8s
[CV] END ..... C=10, penalty=l1, solver=saga; total time= 58.1s
[CV] END ..... C=0.01, penalty=l1, solver=saga; total time= 10.5s
[CV] END ..... C=0.5, penalty=l1, solver=saga; total time= 51.5s
[CV] END ..... C=10, penalty=l1, solver=saga; total time= 51.1s
[CV] END ..... C=0.01, penalty=l1, solver=saga; total time= 6.3s
[CV] END ..... C=0.5, penalty=l1, solver=saga; total time= 51.0s
[CV] END ..... C=10, penalty=l1, solver=saga; total time= 59.1s
[CV] END ..... C=0.01, penalty=l1, solver=saga; total time= 9.6s
[CV] END ..... C=0.5, penalty=l1, solver=saga; total time= 51.5s
[CV] END ..... C=10, penalty=l1, solver=saga; total time= 58.9s
[CV] END ..... C=0.01, penalty=l1, solver=saga; total time= 10.1s
[CV] END ..... C=0.5, penalty=l1, solver=saga; total time= 52.0s
[CV] END ..... C=10, penalty=l1, solver=saga; total time= 58.0s
[CV] END ..... C=0.1, penalty=l1, solver=saga; total time= 23.9s
[CV] END ..... C=1, penalty=l1, solver=saga; total time= 56.3s
[CV] END ..... C=100, penalty=l1, solver=saga; total time= 49.5s
[CV] END ..... C=0.1, penalty=l1, solver=saga; total time= 33.7s
[CV] END ..... C=1, penalty=l1, solver=saga; total time= 48.2s
[CV] END ..... C=100, penalty=l1, solver=saga; total time= 49.9s
[CV] END ..... C=0.1, penalty=l1, solver=saga; total time= 33.6s
[CV] END ..... C=1, penalty=l1, solver=saga; total time= 57.2s
[CV] END ..... C=100, penalty=l1, solver=saga; total time= 41.9s
[CV] END ..... C=0.1, penalty=l1, solver=saga; total time= 32.7s
[CV] END ..... C=1, penalty=l1, solver=saga; total time= 56.5s
[CV] END ..... C=100, penalty=l1, solver=saga; total time= 48.3s
[CV] END ..... C=0.1, penalty=l1, solver=saga; total time= 34.2s
[CV] END ..... C=1, penalty=l1, solver=saga; total time= 57.0s
[CV] END ..... C=100, penalty=l1, solver=saga; total time= 47.3s
```

Submission Attempt 2: With an adjusted threshold aiming for Highest Accuracy (kaggle Score: 0.54091)

The approach here is to find and apply the best probability threshold to generate class predictions

```
In [32]: y_probs = best_model.predict_proba(X_val)[:, 1]

# Define the range of thresholds you want to test
thresholds = np.arange(0.0, 1.05, 0.05) # Thresholds from 0.0 to 1.0 in increments of 0.05

# Initialize a list to store accuracy values
accuracy_scores = []

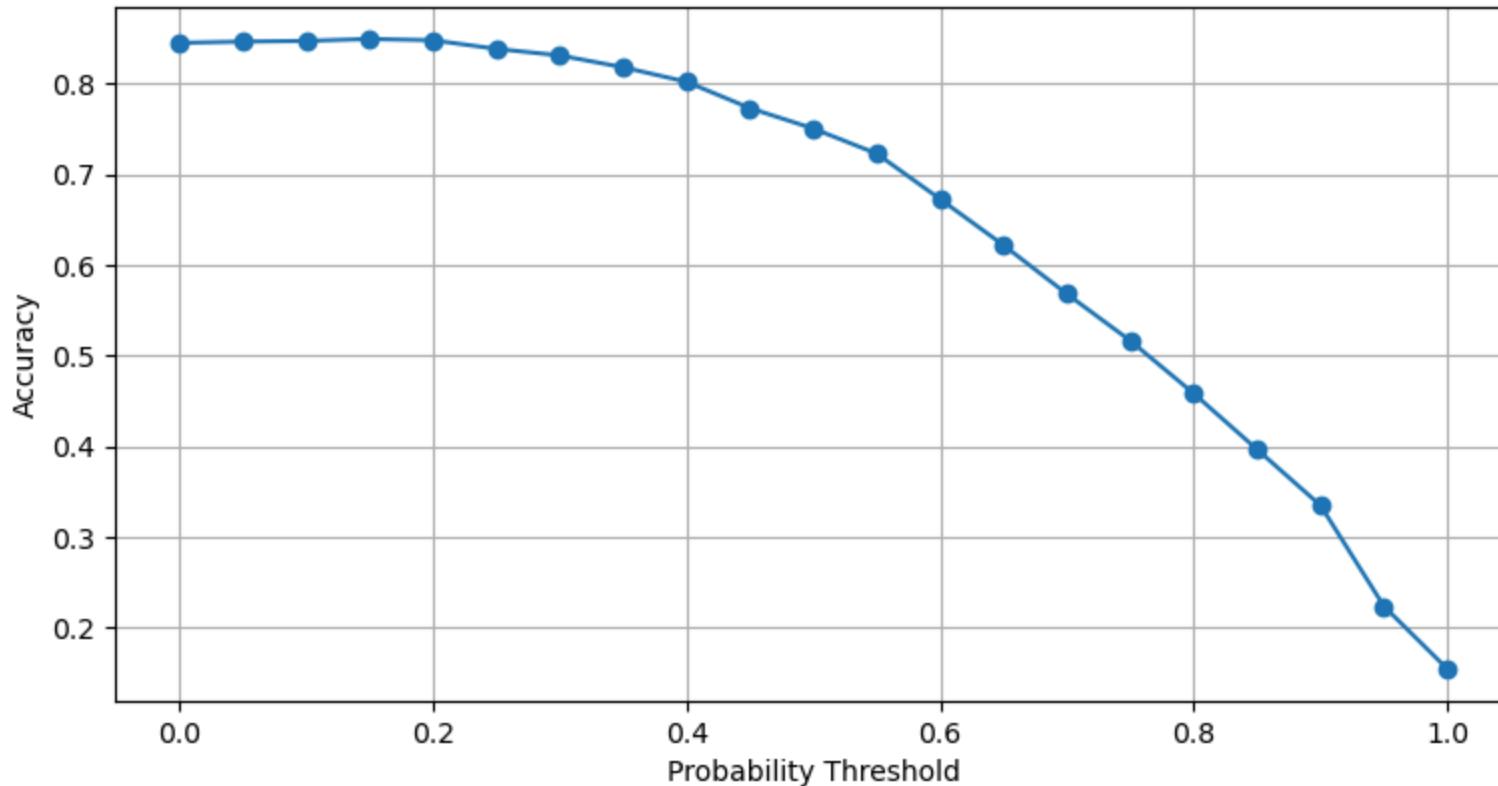
# Loop through each threshold and calculate accuracy
for threshold in thresholds:
    # Apply the threshold to get binary predictions
    y_pred_adjusted = (y_probs >= threshold).astype(int)

    # Calculate accuracy for this threshold
    accuracy = accuracy_score(y_val, y_pred_adjusted)

    # Append the accuracy score to the list
    accuracy_scores.append(accuracy)

# Plot the threshold values against accuracy scores
plt.figure(figsize=(9, 4.5))
plt.plot(thresholds, accuracy_scores, marker='o')
plt.title('Threshold vs Accuracy on Validation Set')
plt.xlabel('Probability Threshold')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

Threshold vs Accuracy on Validation Set



```
In [33]: best_threshold = thresholds[np.argmax(accuracy_scores)]
```

```
print(f"Best Probability Threshold that gives the \nhighest Accuracy on the validation set: {best_threshold}")
```

Best Probability Threshold that gives the
highest Accuracy on the validation set: 0.15000000000000002

```
In [34]: # Get predicted probabilities for the test set
y_test_prob = best_model.predict_proba(df_test_final)[:, 1]

# Apply the best threshold to generate class predictions
y_test_pred = (y_test_prob >= best_threshold).astype(int)

# Create a submission file with predicted class values
submission = pd.DataFrame({
    'id': df_test_final.index,
    'output': y_test_pred           # The class predictions (0 or 1)
})

# Ensure the submission matches the test set size
assert submission.shape[0] == df_test_final.shape[0]

# Save the submission file to a CSV
submission.to_csv('submission_Baseline_Logistic_Regularized_w_Threshold.csv', index=False)

print("Submission file with class values created successfully!")
```

Submission file with class values created successfully!

Submission Attempt 3: With an adjusted threshold aiming for Highest ROC_AUC Score (Kaggle Score: 0.63976)

The approach here is to find and apply the best probability threshold to generate class predictions.

```
In [35]: # Get the predicted probabilities (for class 1) on the validation set
y_prob = best_model.predict_proba(X_val)[:, 1]

# Define the range of thresholds you want to test
thresholds = np.arange(0.0, 1.05, 0.05)

# Initialize a list to store ROC AUC values
roc_auc_scores = []

# Loop through each threshold and calculate ROC AUC
for threshold in thresholds:
    # Apply the threshold to get binary predictions
    y_pred_adjusted = (y_prob >= threshold).astype(int)

    # Calculate ROC AUC for this threshold
    roc_auc = roc_auc_score(y_val, y_pred_adjusted)

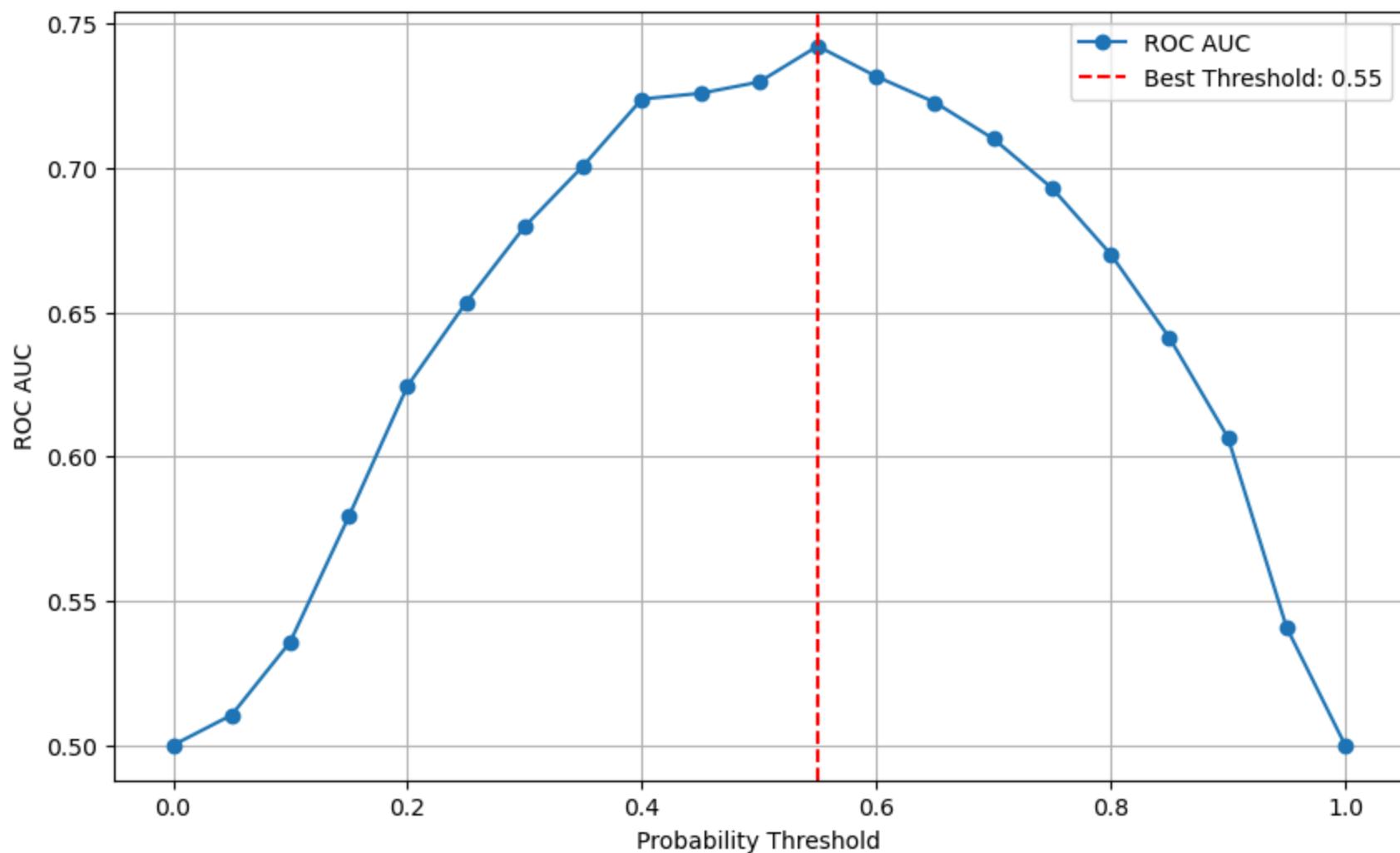
    # Append the ROC AUC score to the list
    roc_auc_scores.append(roc_auc)

# Find the best threshold based on maximum ROC AUC
best_threshold = thresholds[np.argmax(roc_auc_scores)]
print(f"Best threshold based on validation ROC AUC: {best_threshold}")

# Plot the threshold values against ROC AUC scores
plt.figure(figsize=(10, 6))
plt.plot(thresholds, roc_auc_scores, marker='o', label='ROC AUC')
plt.axvline(x=best_threshold, color='r', linestyle='--', label=f'Best Threshold: {best_threshold:.2f}')
plt.title('Threshold vs ROC AUC on Validation Set')
plt.xlabel('Probability Threshold')
plt.ylabel('ROC AUC')
plt.grid(True)
plt.legend()
plt.show()
```

Best threshold based on validation ROC AUC: 0.55

Threshold vs ROC AUC on Validation Set



```
In [36]: # Define the range of thresholds you want to test
thresholds = np.arange(0.0, 1.05, 0.05)

# Initialize lists to store Accuracy and ROC AUC values
accuracy_scores = []
roc_auc_scores = []

# Loop through each threshold and calculate both Accuracy and ROC AUC
for threshold in thresholds:
    # Apply the threshold to get binary predictions
    y_pred_adjusted = (y_prob >= threshold).astype(int)

    # Calculate Accuracy for this threshold
    accuracy = accuracy_score(y_val, y_pred_adjusted)
    accuracy_scores.append(accuracy)

    # Calculate ROC AUC for this threshold
    roc_auc = roc_auc_score(y_val, y_pred_adjusted)
    roc_auc_scores.append(roc_auc)

# Find the best threshold based on maximum ROC AUC
best_threshold = thresholds[np.argmax(roc_auc_scores)]
print(f"Best threshold based on validation ROC AUC: {best_threshold}")

# Plot the threshold values against Accuracy and ROC AUC scores
plt.figure(figsize=(10, 6))

# Plot Accuracy
plt.plot(thresholds, accuracy_scores, label='Accuracy', marker='o', color='blue')

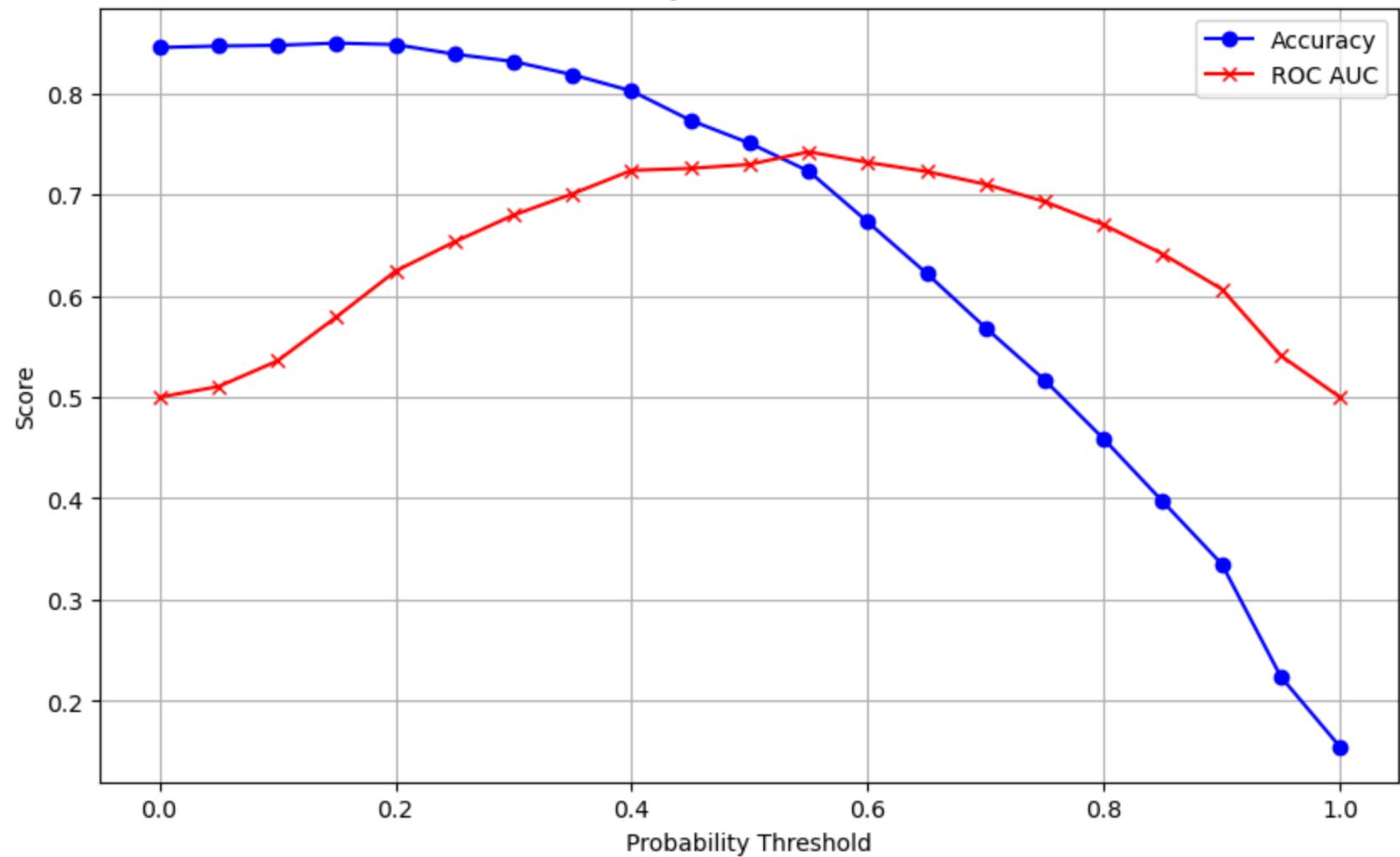
# Plot ROC AUC
plt.plot(thresholds, roc_auc_scores, label='ROC AUC', marker='x', color='red')

# Add labels and title
plt.title('Threshold vs Accuracy and ROC AUC on Validation Set')
plt.xlabel('Probability Threshold')
plt.ylabel('Score')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```

Best threshold based on validation ROC AUC: 0.55

Threshold vs Accuracy and ROC AUC on Validation Set



In [37]: # Find the best threshold based on maximum ROC AUC

```
best_threshold_index = np.argmax(roc_auc_scores)
best_threshold = thresholds[best_threshold_index]
print(f"Best threshold based on validation ROC AUC: {best_threshold}")

# Calculate and print the ROC AUC and accuracy for the best threshold
y_pred_best_threshold = (y_prob >= best_threshold).astype(int)
best_threshold_roc_auc = roc_auc_score(y_val, y_pred_best_threshold)
best_threshold_accuracy = accuracy_score(y_val, y_pred_best_threshold)

print(f"ROC AUC for best threshold ({best_threshold}): {best_threshold_roc_auc}")
print(f"Accuracy for best threshold ({best_threshold}): {best_threshold_accuracy}")

# Now calculate for the default threshold of 0.5
y_pred_default = (y_prob >= 0.5).astype(int)
default_threshold_roc_auc = roc_auc_score(y_val, y_pred_default)
default_threshold_accuracy = accuracy_score(y_val, y_pred_default)

print(f"ROC AUC for default threshold (0.5): {default_threshold_roc_auc}")
print(f"Accuracy for default threshold (0.5): {default_threshold_accuracy}")
```

```
Best threshold based on validation ROC AUC: 0.55
ROC AUC for best threshold (0.55): 0.7422422179970047
Accuracy for best threshold (0.55): 0.7231104651162791
ROC AUC for default threshold (0.5): 0.729816848929634
Accuracy for default threshold (0.5): 0.7507267441860465
```

In [38]:

```
# Now use the best threshold to make predictions on the test set
y_test_prob = best_model.predict_proba(df_test_final)[:, 1]

# Apply the best threshold to generate class predictions
y_test_pred = (y_test_prob >= best_threshold).astype(int)

# Create a submission file with predicted class values
submission = pd.DataFrame({
    'id': df_test_final.index,
    'output': y_test_pred      # The class predictions (0 or 1)
})

# Ensure the submission matches the test set size
assert submission.shape[0] == df_test_final.shape[0]

# Save the submission file to a CSV
submission.to_csv('submission_Baseline_Logistic_Regularized_w_ROC_thresh.csv', index=False)

print("Submission file with class values created successfully!")
```

Submission file with class values created successfully!

Final Notes:

The above analysis evaluated the performance of a binary classification model by exploring how different probability thresholds affect key metrics such as Accuracy and ROC AUC. It tested a range of thresholds from 0.0 to 1.0 in increments of 0.05 and adjusted the predicted probabilities to generate binary predictions. For each threshold, we calculated the corresponding Accuracy and ROC AUC scores to understand how model performance varied. We also identified the threshold that maximized the ROC AUC score and compared its impact on both metrics. Finally visualized these relationships in a plot, providing insights into how the threshold influences the model's performance.

Maximizing the ROC AUC is important for this task because it provides a comprehensive measure of the model's ability to distinguish between classes across different threshold values. A higher ROC AUC indicates better discrimination between positive and negative classes, which is crucial for tasks where minimizing false positives and false negatives is important. By identifying the threshold that maximizes ROC AUC, we can ensure the model achieves the best possible balance between sensitivity and specificity, leading to more reliable and accurate predictions in the held out test set.