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


import pandas as pd
import numpy as np
from plotnine import *
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression # Linear Regression
Model
from sklearn.metrics import mean_squared_error, r2_score,
accuracy_score,mean_absolute_error #model evaluation
from sklearn.metrics import accuracy_score, confusion_matrix,
plot_confusion_matrix,\
 f1_score, recall_score, plot_roc_curve, precision_score,
roc_auc_score
from sklearn.model_selection import train_test_split # simple TT split
cv
from sklearn.model_selection import KFold # k-fold cv
from sklearn.model_selection import LeaveOneOut #LOO cv
from sklearn.model_selection import cross_val_score # cross validation
metrics
from sklearn.model_selection import cross_val_predict # cross
validation metrics
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
```

1)

```
#Reading in the dataset,dropping missing values, as well as making
dummy variables for the State column
Recall=pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleri
ti/CPSC392ParlettPelleriti/master/Data/HW2.csv")
Recall=Recall.dropna()

Recall=pd.get_dummies(Recall,columns=['State'])
predictors = ["WordList", "Trial.Number", "out_degree", "in_degree",
              "Length",
"Log_Freq_HAL","Ortho_N","Phono_N","Concreteness_Rating","Age_Of_Acqui
sition","Female","State_CA","State_ID","State_NM","State_OR","State_WA
"]
```

```
X=Recall[predictors]
y=Recall["correct"]

#Train test split
X_train, X_test, y_train, y_test =
train_test_split(Recall[predictors], Recall["correct"], test_size=0.1)

#Zscoring
z=StandardScaler()
ContinuousVars=
["Trial.Number","out_degree","in_degree","Length","Log_Freq_HAL","Orth
o_N","Phono_N","Concreteness_Rating","Age_Of_Acquisition"]

X_train[ContinuousVars] = z.fit_transform(X_train[ContinuousVars])
X_test[ContinuousVars] = z.transform(X_test[ContinuousVars])
```

```
<bound method NDFrame.head of        WordList  Trial.Number
out_degree  in_degree    Length  Log_Freq_HAL  \
14438          1      1.279009    2.822312  -0.131754  0.640612
0.174321
10977          1     -0.075643    1.416728   4.529228 -1.858713
2.040326
696            0     -0.277399    0.573378   1.732639 -1.858713
1.056696
10771          1      1.596055   -0.832206  -0.570435 -1.858713
0.855885
14056          0      1.250187    0.011144  -0.405929 -0.609051
0.510423
...          ...           ...         ...        ...       ...
...
11599          0      1.106075    0.011144   0.964948 -0.609051     -
0.517453
5672           0      0.789029    0.292261   0.087586  0.640612     -
0.399179
1321           1      1.711345    0.011144  -0.460765 -0.609051     -
0.914819
6586           1      0.587272   -0.832206  -0.570435 -1.858713     -
1.501934
14047          1     -0.277399    0.011144  -0.296259 -0.609051
0.723146

        Ortho_N   Phono_N  Concreteness_Rating  Age_Of_Acquisition
Female  \
14438 -1.150770  0.067703             1.093258           -0.507038
1
10977  1.330578  1.042758             1.223997           -0.998890
0
696    1.123799  2.126151             1.200226           -1.490742
0
10771  2.157693  1.692794             0.439565            0.078500
1
```

```
14056 -0.530433 -1.232369            0.403909           1.425238
1
...         ...       ...                 ...                ...
...
11599  0.917020 -0.148975           -0.784624          -1.402911
1
5672  -0.943991 -0.365654            0.320712           0.810423
0
1321   0.710241 -0.799011           -0.237898           1.343263
1
6586   1.330578  0.717740           -0.439949           0.382980
1
14047  0.710241  0.717740           -0.012077           2.209859
1

       State_CA  State_ID  State_NM  State_OR  State_WA
14438         0         0         0         1         0
10977         0         1         0         0         0
696           0         0         1         0         0
10771         0         0         1         0         0
14056         1         0         0         0         0
...         ...       ...       ...       ...       ...
11599         0         0         1         0         0
5672          0         0         1         0         0
1321          1         0         0         0         0
6586          0         0         1         0         0
14047         0         0         0         1         0

[12512 rows x 16 columns]>
```

```
#Building logistic regression
lr=LogisticRegression()
lr.fit(X_train, y_train)
```

```
LogisticRegression()
```

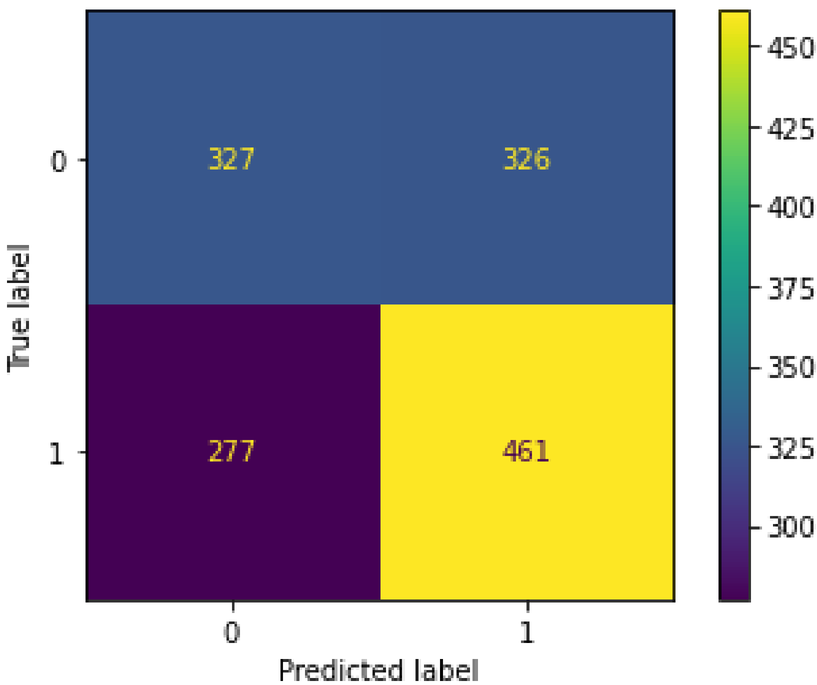
```
#Various accuracy metrics for the logistic regression
acc_train = []
acc_test = []
roc_train = []
roc_test = []
precision_train=[]
precision_test=[]
tpred=lr.predict(X_train)
tepred=lr.predict(X_test)
acc_train.append(accuracy_score(y_train, lr.predict(X_train)))
acc_test.append(accuracy_score(y_test, lr.predict(X_test)))
roc_train.append(roc_auc_score(y_train, lr.predict_proba(X_train)
[:,1]))
roc_test.append(roc_auc_score(y_test, lr.predict_proba(X_test)[:,1]))
precision_train.append(precision_score(y_train, tpred,
```

```
average='macro'))
precision_test.append(precision_score(y_test, tepred,
average='macro'))
print("Logistic Regression train accuracy is: ",acc_train)
print("Logistic Regression test accuracy is: ",acc_test)
print("Logistic Regression ROC AUC train is: ",np.round(roc_train,2))
print("Logistic Regression ROC AUC test is: ",np.round(roc_test,2))
print("Logistic Regression train precision is: ",precision_train)
print("Logistic Regression test precision is: ",precision_test)
```

```
Logistic Regression train accuracy is:  [0.5656969309462916]
Logistic Regression test accuracy is:  [0.5664989216391085]
Logistic Regression ROC AUC train is:  [0.59]
Logistic Regression ROC AUC test is:  [0.58]
Logistic Regression train precision is:  [0.5611222542516777]
Logistic Regression test precision is:  [0.5635797352676355]
```

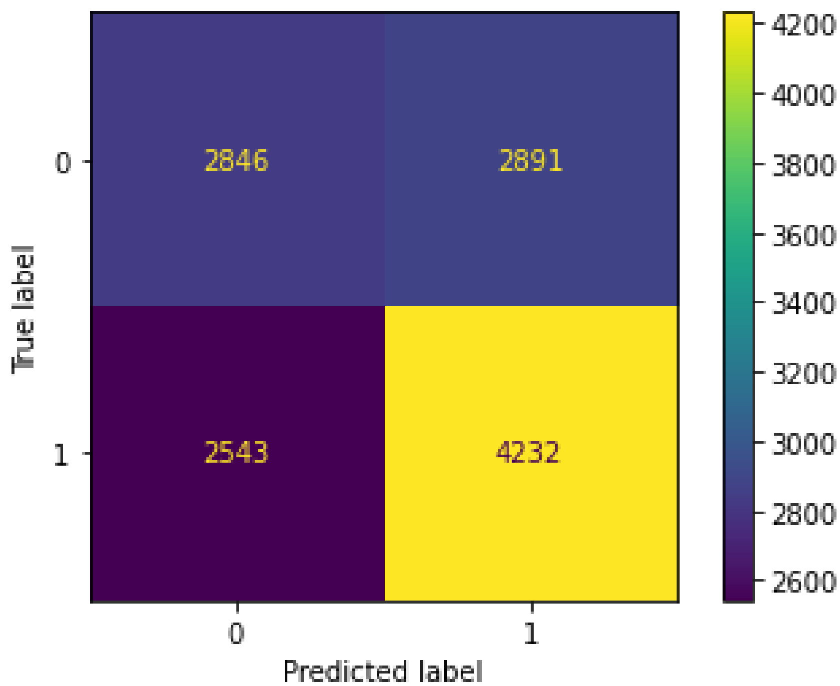
```
#Confusion matrix for test
plot_confusion_matrix(lr,X_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7fc8f76bd0>
```




```
#Confusion matrix for train
plot_confusion_matrix(lr,X_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7fc66cac10>
```




```
#Decision tree empty model creation and using the pipeline and grid
search to find the ideal min_samples_leaf
tree = DecisionTreeClassifier()

newz=make_column_transformer((StandardScaler(),ContinuousVars),remaind
er='passthrough')
tree=DecisionTreeClassifier()
pipe=make_pipeline(newz,tree)
leaves={"decisiontreeclassifier__min_samples_leaf":range(0,100)}
grid=GridSearchCV(pipe,leaves,scoring='accuracy',cv=5,refit=True)
grid.fit(X_train,y_train)

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('columntransformer',

ColumnTransformer(remainder='passthrough',

transformers=[('standardscaler',

StandardScaler(),

['Trial.Number',

'out_degree',

'in_degree',

'Length',
```

```
'Log_Freq_HAL',

'Ortho_N',

'Phono_N',

'Concreteness_Rating',

'Age_Of_Acquisition'])])),
                                       ('decisiontreeclassifier',
                                        DecisionTreeClassifier())]),
             param_grid={'decisiontreeclassifier__min_samples_leaf':
range(0, 100)},
             scoring='accuracy')
```

```
#Getting the chosen value, and creating the tree with that parameter
that was assigned by the gridsearch
#and creating the corresponding tree and fitting it as well
#I also printed the chosen value
chosen=grid.best_estimator_.get_params()
["decisiontreeclassifier__min_samples_leaf"]
myTree=DecisionTreeClassifier(min_samples_leaf=chosen)
myTree.fit(X_train,y_train)
treetrainpred=myTree.predict(X_train)
treetestpred=myTree.predict(X_test)
print("Chosen is: ",chosen)
```

```
Chosen is:  1
```

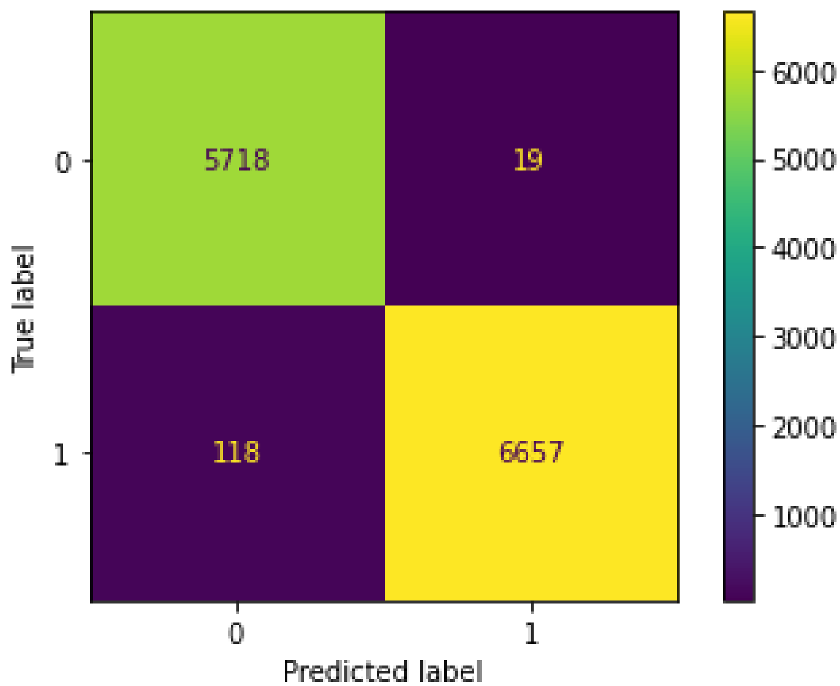
```
#Tree accuracy metrics
tree_acc_train=[]
tree_acc_test=[]
tree_roc_train = []
tree_roc_test = []
tree_precision_train=[]
tree_precision_test=[]
tree_acc_train.append(accuracy_score(y_train,treetrainpred))
tree_acc_test.append(accuracy_score(y_test, treetestpred))
tree_roc_train.append(roc_auc_score(y_train,
myTree.predict_proba(X_train)[:,1]))
tree_roc_test.append(roc_auc_score(y_test,
myTree.predict_proba(X_test)[:,1]))
tree_precision_train.append(precision_score(y_train, treetrainpred,
average='macro'))
tree_precision_test.append(precision_score(y_test, treetestpred,
average='macro'))
print("Tree train accuracy is: ",tree_acc_train)
print("Tree test accuracy is: ",tree_acc_test)
print("Tree ROC AUC train is: ",np.round(tree_roc_train,2))
```

```python
print("Tree ROC AUC test is: ",np.round(tree_roc_test,2))
print("Tree train precision is: ",tree_precision_train)
print("Tree test precision is: ",tree_precision_test)
```

```
Tree train accuracy is:  [0.9890505115089514]
Tree test accuracy is:  [0.7088425593098491]
Tree ROC AUC train is:  [1.]
Tree ROC AUC test is:  [0.71]
Tree train precision is:  [0.9884673280573749]
Tree test precision is:  [0.7077271970705725]
```

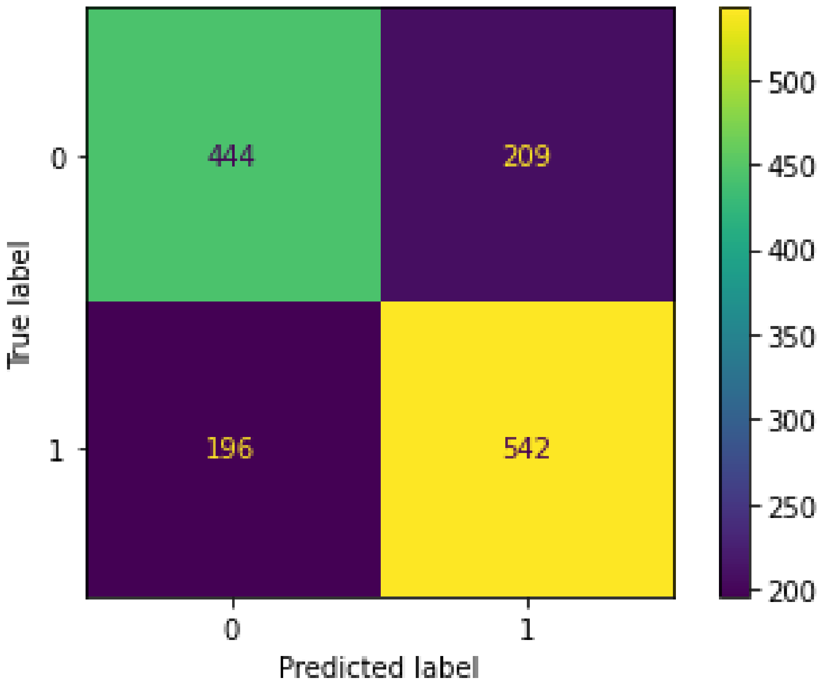
```python
#Tree train confusion matrix
plot_confusion_matrix(myTree,X_train,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7fc6622b90>
```




```python
#Tree test confusion matrix
plot_confusion_matrix(myTree,X_test,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7fc645f090>
```




```
#Creating the contintuous predictors for the KNN
newxt=X_train[ContinuousVars]
newxte=X_test[ContinuousVars]

#KNN grid search and printing the chosen value
knn2 = KNeighborsClassifier()

# choose potential values of k
ks = {"n_neighbors": range(1,30)}

# use grid search to find best parameters
grid = GridSearchCV(knn2, ks, scoring = "accuracy", cv = 5, refit =
True)

grid.fit(newxt, y_train)
chosen=grid.best_estimator_.get_params()["n_neighbors"]
print(chosen)
```

```
1
```

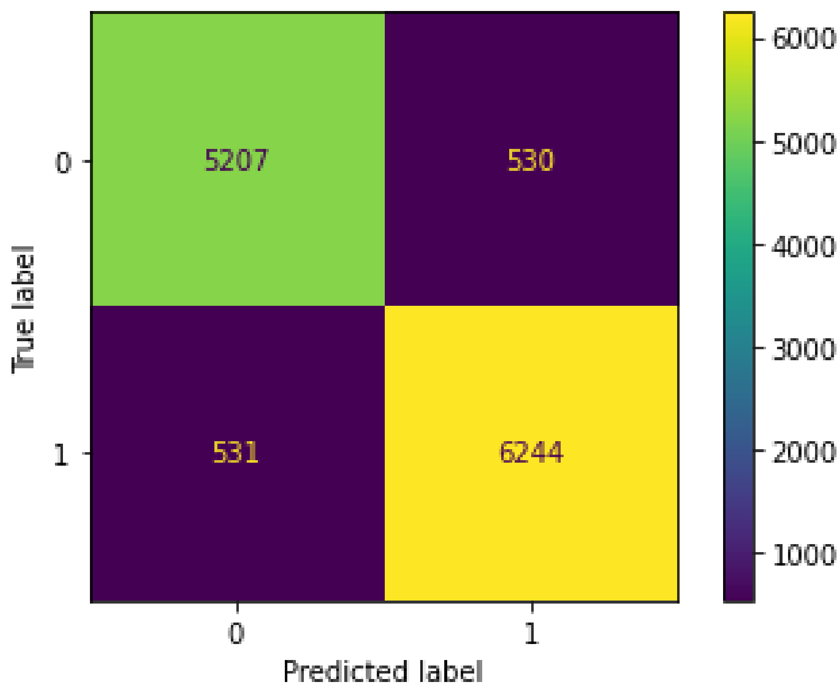
```
#Creating the corresponding KNN model and fitting it with the chosen
value and the continuous variables as predictors
knn=KNeighborsClassifier(n_neighbors=1)
knn.fit(newxt,y_train)
knnpredt=knn.predict(newxt)
knnpredte=knn.predict(newxte)

#KNN accuracy metrics
knn_acc_train=[]
```

```python
knn_acc_test=[]
knn_roc_train = []
knn_roc_test = []
knn_score_train=[]
knn_score_test=[]
knn_acc_train.append(accuracy_score(y_train,knnpredt))
knn_acc_test.append(accuracy_score(y_test, knnpredte))
knn_roc_train.append(roc_auc_score(y_train, knn.predict_proba(newxt)
[:,1]))
knn_roc_test.append(roc_auc_score(y_test, knn.predict_proba(newxte)
[:,1]))
knn_score_train.append(knn.score(newxt, y_train))
knn_score_test.append(knn.score(newxte, y_test))
print("KNN train accuracy is: ",knn_acc_train)
print("KNN test accuracy is: ",knn_acc_test)
print("KNN ROC AUC train is: ",np.round(knn_roc_train,2))
print("KNN ROC AUC test is: ",np.round(knn_roc_test,2))
print("KNN score train is: ",knn_score_train)
print("KNN score test is: ",knn_score_test)
```

```
KNN train accuracy is:  [0.9152014066496164]
KNN test accuracy is:  [0.7368799424874192]
KNN ROC AUC train is:  [0.91]
KNN ROC AUC test is:  [0.74]
KNN score train is:  [0.9152014066496164]
KNN score test is:  [0.7368799424874192]
```

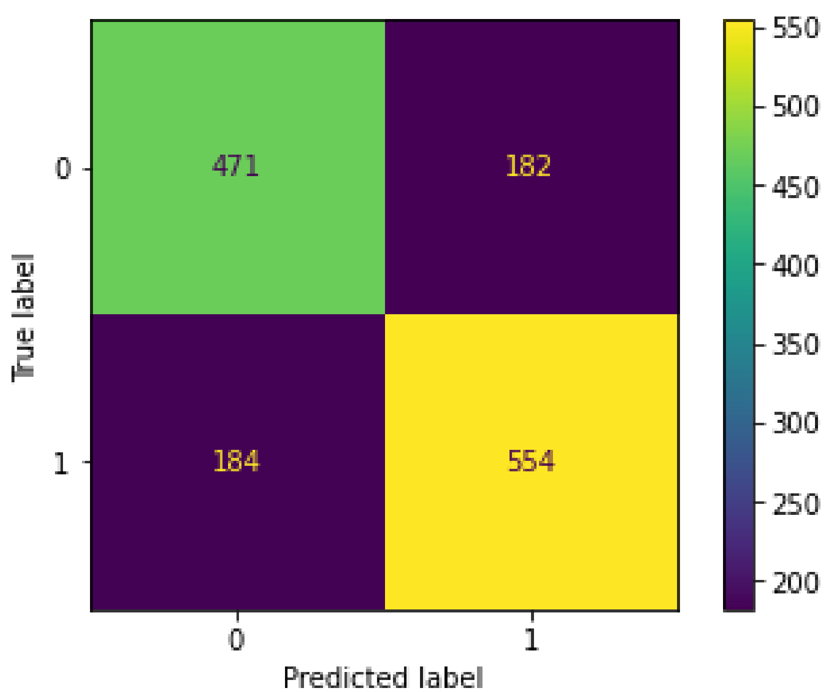
```python
#KNN train confusion metrics
plot_confusion_matrix(knn,newxt,y_train)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7fc636aa10>
```




```
#KNN test confusion metrics
plot_confusion_matrix(knn,newxte,y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7fc62a93d0>
```



2a)

I think the KNN model did the best of all three models. All three models were iffy from the start. The logistic regression was fitted correctly but hovered around .6 in all metrics which meant that it couldn't predict very reliably. Although the contents of this study aren't inherently high stake like predicting stroke in the last project, these predictions are high stake, solely for the fact that it is still a scientific research study. The decision tree and KNN both had very similar results. Both were overfit but were able to predict on unseen data much better than the logistic regression model. The reason I pick the KNN is because it is clearly less overfit than the decision tree because the decision tree has near perfect (or perfect) accuracy with the training set and nearly the same accuracy as the KNN for the test set.

As a result, we are able to see that the KNN model is less overfit because it does worse on the train set than the decision tree but still does similarly well on the unseen data in the test set.

The KNN model has around .9 accuracy on the training set and around .74 accuracy in the unseen testing set. While that isn't great, it is a lot better than the 60% of the logistic regression.

We can clearly see this model is overfit because it is about 16% better at accuractely predicting the training set than it is the testing set, but still, .74 isn't a bad correct prediction rate.

For this question, accuracy score speaks the most to me because it is checking for how many labels in the true y values are the exact same as in the predicted y values. Ideally we would want our model to be less overfit and have higher test accuracy but with the three models we have, KNN is the one that I would call the best.

2b)

I would trust this model and push it to production. I am pretty satisfied with a 3/4 chance of being correct when predicting on new, unseen data. If a subject corretly recalls a certain word isn't a high stakes scenario and nothing bad would happen if someone's recall was incorrectly predicted to have recalled a word when they didn't.

False negatives and positives are equally important because in the false negative case, the model would predict that the subject did not recall the word when they actually did and a false positive would be when the model predicted they recalled the word when they truly didn't.

Neither is weighted more heavy than the other because there are no real world implications if the model makes a mistake. In the example of our last big project with stroke data, there are major real world consequences if a prediction is incorrect.

To conclude, I would trust this model the reliably predict on future, unseen data. Although it is overfit, it has a pretty reliable prediction rate on new data of 74%.