In [742…

```
#handle imports
import sys
!{sys.executable} -m pip install numpy
!{sys.executable} -m pip install matplotlib
!{sys.executable} -m pip install sklearn
!{sys.executable} -m pip install pandas
!{sys.executable} -m pip install seaborn
!{sys.executable} -m pip install scipy
!{sys.executable} -m pip install datetime
!{sys.executable} -m pip install arff

%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import arff
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set_style('white')

from sklearn.datasets import load_digits, make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, plot_confusion_matrix
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler, StandardScaler, La
from sklearn.preprocessing import RobustScaler, Normalizer, QuantileTransformer,
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.utils.multiclass import type_of_target
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score


from datetime import datetime
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.7/site-packages
(1.19.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/site-packa
ges (3.3.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /usr/
local/lib/python3.7/site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/site-pack
ages (from matplotlib) (1.19.5)
Requirement already satisfied: python-dateutil>=2.1 in /Users/timothynordahl/Lib
rary/Python/3.7/lib/python/site-packages (from matplotlib) (2.8.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/sit
e-packages (from matplotlib) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/site-pa
ckages (from matplotlib) (8.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/site-pac
kages (from matplotlib) (0.10.0)
Requirement already satisfied: six>=1.5 in /usr/local/Cellar/protobuf/3.7.1/libe
xec/lib/python3.7/site-packages (from python-dateutil>=2.1->matplotlib) (1.12.0)
```

```
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/site-packages
(0.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/site-pac
kages (from sklearn) (0.24.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/site-pac
kages (from scikit-learn->sklearn) (1.0.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/site-pa
ckages (from scikit-learn->sklearn) (1.19.5)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/site-pa
ckages (from scikit-learn->sklearn) (1.6.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/
site-packages (from scikit-learn->sklearn) (2.1.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/site-packages
(1.2.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/site-pac
kages (from pandas) (2020.1)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.7/site-pa
ckages (from pandas) (1.19.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /Users/timothynordahl/L
ibrary/Python/3.7/lib/python/site-packages (from pandas) (2.8.0)
Requirement already satisfied: six>=1.5 in /usr/local/Cellar/protobuf/3.7.1/libe
xec/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas) (1.12.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/site-packages
(0.11.1)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.7/site-pac
kages (from seaborn) (1.2.1)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/site-
packages (from seaborn) (3.3.4)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/site-pack
ages (from seaborn) (1.19.5)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/site-packa
ges (from seaborn) (1.6.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/site-pac
kages (from pandas>=0.23->seaborn) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /Users/timothynordahl/L
ibrary/Python/3.7/lib/python/site-packages (from pandas>=0.23->seaborn) (2.8.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/site-pa
ckages (from matplotlib>=2.2->seaborn) (8.1.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/sit
e-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /usr/
local/lib/python3.7/site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/site-pac
kages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: six>=1.5 in /usr/local/Cellar/protobuf/3.7.1/libe
xec/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas>=0.23->seab
orn) (1.12.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/site-packages
(1.6.0)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.7/site-pa
ckages (from scipy) (1.19.5)
Requirement already satisfied: datetime in /usr/local/lib/python3.7/site-package
s (4.3)
Requirement already satisfied: pytz in /usr/local/lib/python3.7/site-packages (f
rom datetime) (2020.1)
Requirement already satisfied: zope.interface in /usr/local/lib/python3.7/site-p
ackages (from datetime) (5.2.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/site-packa
ges (from zope.interface->datetime) (41.0.1)
Requirement already satisfied: arff in /usr/local/lib/python3.7/site-packages
(0.9)
```

In [950…

```
#import letter dataframe
letter_df = pd.read_csv("/Users/timothynordahl/Desktop/COGS118AFinalProject/lett
```

```
letter_df.head()
```

Out[950…

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

In [951…

```
#set A-M as 1 and N-Z as 0
dict = {'A': 1,
     'B': 2,
     'C': 3,
     'D': 4,
     'E': 5,
     'F': 6,
     'G': 7,
     'H': 8,
     'I': 9,
     'J': 10,
     'K': 11,
     'L': 12,
     'M': 13,
     'N': 14,
     'O': 15,
     'P': 16,
     'Q': 17,
     'R': 18,
     'S': 19,
     'T': 20,
     'U': 21,
     'V': 22,
     'W': 23,
     'X': 24,
     'Y': 25,
     'Z': 26,
}
for i in range(letter_df[0].size):
    letter_df[0][i] = dict[letter_df[0][i]]
    if letter_df[0][i] <= 13:
        letter_df[0][i] = 1
    else:
        letter_df[0][i] = 0

letter_df = letter_df.astype('int')
```

In [243…

```
letter_df.shape
```

Out[243… (20000, 17)

In [862…

```
#Check #positive
letter_df_pos = letter_df[letter_df[0] == 1]
```

```
letter_df_pos.shape
```

Out[862… (9940, 17)

In [126…

```
#import occupancy data
occupancy_df = pd.read_csv("/Users/timothynordahl/Desktop/COGS118AFinalProject/o
```

In [127…

```
#convert datetime data to cyclic seconds past midnight, drop date
datetime_object = datetime.fromisoformat('2015-02-04 17:51:00')
pd.options.mode.chained_assignment = None  # default='warn'

for i in range ((occupancy_df['date']).size):
    occupancy_df['date'][i+1] = datetime.fromisoformat(occupancy_df['date'][i+1]

time_sec = 0
occupancy_df['sec'] = occupancy_df['CO2']
for i in range ((occupancy_df['date']).size):
    time_sec = occupancy_df['date'][i+1]
    occupancy_df['sec'][i+1] = (time_sec - time_sec.replace(hour=0, minute=0, se

seconds_in_day = 24*60*60

occupancy_df['sin_sec'] = np.sin(2*np.pi*occupancy_df.sec/seconds_in_day)
occupancy_df['cos_sec'] = np.cos(2*np.pi*occupancy_df.sec/seconds_in_day)

del occupancy_df['date']
del occupancy_df['sec']

occupancy_df.head()
```

Out[127…

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy	sin_sec	cos_sec
1	23.18	27.2720	426.0	721.25	0.004793	1	-0.999229	-0.039260
2	23.15	27.2675	429.5	714.00	0.004783	1	-0.999388	-0.034972
3	23.15	27.2450	426.0	713.50	0.004779	1	-0.999534	-0.030539
4	23.15	27.2000	426.0	708.25	0.004772	1	-0.999657	-0.026177
5	23.10	27.2000	426.0	704.50	0.004757	1	-0.999762	-0.021815

In [128…

```
occupancy_df.shape
```

Out[128… (8143, 8)

In [129…

```
#set occupancy as first column
occupancy_df = occupancy_df[['Occupancy','Temperature','Humidity','Light','CO2',
occupancy_df.head()
```

Out[129…

	Occupancy	Temperature	Humidity	Light	CO2	HumidityRatio	sin_sec	cos_sec
1	1	23.18	27.2720	426.0	721.25	0.004793	-0.999229	-0.039260
2	1	23.15	27.2675	429.5	714.00	0.004783	-0.999388	-0.034972

	Occupancy	Temperature	Humidity	Light	CO2	HumidityRatio	sin_sec	cos_sec
3	1	23.15	27.2450	426.0	713.50	0.004779	-0.999534	-0.030539
4	1	23.15	27.2000	426.0	708.25	0.004772	-0.999657	-0.026177
5	1	23.10	27.2000	426.0	704.50	0.004757	-0.999762	-0.021815

In [860…

```
#get positive number for pos rate
occupancy_df_pos = occupancy_df[occupancy_df['Occupancy']==1]
occupancy_df_pos.shape
```

Out[860…

```
(1729, 8)
```

In [382…

```
#import eeg eye state data
EEG_eye_df = pd.read_csv("/Users/timothynordahl/Desktop/COGS118AFinalProject/EEG
EEG_eye_df.shape
```

Out[382…

```
(14980, 15)
```

In [441…

```
EEG_eye_df.dropna()
EEG_eye_df.head()
```

Out[441…

	0	1	2	3	4	5	6	7	8	9
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10

In [858…

```
EEG_eye_df_pos = EEG_eye_df[EEG_eye_df[14]==1]
EEG_eye_df_pos.shape
```

Out[858…

```
(6723, 15)
```

In [494…

```
#import avila data
avila_df = pd.read_csv("/Users/timothynordahl/Desktop/COGS118AFinalProject/avila
avila_df.head()
```

Out[494…

	0	1	2	3	4	5	6	7	
0	0.266074	-0.165620	0.320980	0.483299	0.172340	0.273364	0.371178	0.929823	0.25
1	0.130292	0.870736	-3.210528	0.062493	0.261718	1.436060	1.465940	0.636203	0.28
2	-0.116585	0.069915	0.068476	-0.783147	0.261718	0.439463	-0.081827	-0.888236	-0.12
3	0.031541	0.297600	-3.210528	-0.583590	-0.721442	-0.307984	0.710932	1.051693	0.59

	0	1	2	3	4	5	6	7
4	0.229043	0.807926	-0.052442	0.082634	0.261718	0.148790	0.635431	0.051062	0.03:

In [495…

```
avila_df.shape
```

Out[495… (20867, 11)

In [496…

```
avila_df = avila_df.replace('A',1)
avila_df = avila_df.replace('B',1)
avila_df = avila_df.replace('C',1)
avila_df = avila_df.replace('D',1)
avila_df = avila_df.replace('E',1)
avila_df = avila_df.replace('F',1)
avila_df = avila_df.replace('G',0)
avila_df = avila_df.replace('H',0)
avila_df = avila_df.replace('I',0)
avila_df = avila_df.replace('W',0)
avila_df = avila_df.replace('X',0)
avila_df = avila_df.replace('Y',0)

avila_df[10].unique()
```

Out[496… array([1, 0])

In [856…

```
#get avila pos number
avila_df_pos = avila_df[avila_df[10]==1]
avila_df_pos.shape
```

Out[856… (15606, 11)

In [498…

```
# Establish list of dataframes and the associated classification column
data_list = [(avila_df,10),(EEG_eye_df, 14),(occupancy_df, 'Occupancy'), (letter

# Add Name property to dataframes
occupancy_df.name = "Occupancy"
EEG_eye_df.name = "EEG Eye"
letter_df.name = "Letter"
avila_df.name = "Avila"

# Create dict of dataframes
dict = {
    'Occupancy Dataset' : occupancy_df,
    'EEG Eye Dataset' : EEG_eye_df,
    'Letters Dataset' : letter_df,
    'Avila Dataset' : avila_df,
}
```

In [765…

```
# Create pipeline packaging a standard scaler and logistic regression classifier
pipe = Pipeline([('std', StandardScaler()),
                 ('classifier', LogisticRegression())])
```

```
# Create search space of hyperparameters for logreg model
search_space = [{'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['saga'],
                 'classifier__penalty': ['l1', 'l2'],
                 'classifier__C': np.logspace(-8, 4, 13),
                 'classifier__random_state':[1000]},
                {'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['lbfgs','newton-cg','sag'],
                 'classifier__penalty': ['none'],
                 'classifier__random_state':[1000]},
                {'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['lbfgs','newton-cg','sag'],
                 'classifier__penalty': ['l2'],
                 'classifier__C': np.logspace(-8, 4, 13),
                 'classifier__random_state':[1000]},
                {'classifier': [LogisticRegression(max_iter=5000)],
                 'classifier__solver': ['liblinear'],
                 'classifier__penalty': ['l1','l2'],
                 'classifier__C': np.logspace(-8, 4, 13),
                 'classifier__random_state':[1000]}
                ]

# instantiate lists to store data from loop
best_logreg_trials = []
best_logreg_by_data = []
best_logreg_metrics = []

# Loop through each dataframe, collecting the df and Y column
for data_set, target_name in data_list:

    #print line for monitoring purposes
    print("Now working on: ", data_set.name)
    #reset list for later iteration
    best_logreg_trials = []

    for i in range(5):

        #reset list for later iteration
        best_logreg_metrics = []

        #Honestly I got frustrated working with gathering params and setting par
        #while unnecessary, it did save time overall as debugging was taking lon
        #for lack of elegance/scalability
        clf = [GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5, s
                    scoring=['accuracy'], refit='accuracy', verbose=0,n_jobs=-1),
               GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5, s
                    scoring=['roc_auc_ovr'], refit='roc_auc_ovr', verbose=0,n_job
               GridSearchCV(pipe, search_space, cv=StratifiedKFold(n_splits=5, s
                    scoring=['f1_micro'], refit='f1_micro', verbose=0,n_jobs=-1)
              ]

        print('Onto trial: ', i + 1)

        # Set X to a sample of 5000 from the current dataset, make Y the relevan
        X = data_set.sample(n = 5000, random_state = i * 5, axis = 0)
        Y = X[target_name]
        X = X.drop([target_name],axis=1)

        #fit each gridsearch instance and add it to the metric list
        for j in clf:
            best_logreg_metrics.append(j.fit(X,Y))
```

```
        #add the results from each trial to the trial list
        best_logreg_trials.append(best_logreg_metrics)

    #add the results from each set of trials data list
    best_logreg_by_data.append(best_logreg_trials)

print("Finished!")
```

```
Now working on:  Avila
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  EEG Eye
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  Occupancy
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  Letter
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Finished!
```

In [930…

```
# instantiate standard scaler object
std = StandardScaler()

# establish iteration lists for both the test data and training data
log_reg_predict_metric = []
log_reg_predict_trial = []
log_reg_predict_data = []

log_reg_sample_metric = []
log_reg_sample_trial = []
log_reg_sample_data = []

j = 0

#iterate through the data list again
for data_set, target_name in data_list:

    #print to let me know where the program is running
    print("Now working on: ", data_set.name)

    #reset lists to collect trial data
    log_reg_predict_trial = []
    log_reg_sample_trial = []

    for i in range(5):

        #print to let me know where the program is running
```

```
        print("Onto trial:", i+1)

        #reset lists to collect trial data
        log_reg_sample_metric = []
        log_reg_predict_metric= []

        for k in range(3):

            # collect the best estimator from the clf collection
            log_reg = best_logreg_by_data[j][i][k].best_estimator_[1]

            # set X and Y to the same data sample as before, this time scaling i
            X = data_set.sample(n = 5000, random_state = i * 5, axis = 0)
            Y = X[target_name]
            X = X.drop([target_name], axis=1)

            std.fit(X)
            X = std.transform(X)

            # fit the best model for each metric on the training data
            log_reg.fit(X,Y)

            # Predict and score the model on the same training data
            if k % 3 == 0:
                log_reg_sample_metric.append(accuracy_score(Y,log_reg.predict(X)
            elif k % 3 == 1:
                log_reg_sample_metric.append(roc_auc_score(Y,log_reg.predict(X))
            else:
                log_reg_sample_metric.append(f1_score(Y,log_reg.predict(X)))

            #Set X and Y as the whole dataset, again scaling X
            X = data_set.drop([target_name], axis=1)
            Y = data_set[target_name]
            std.fit(X)
            X = std.transform(X)

            # Using the same fit model from above, predict and score it for the
            if k % 3 == 0:
                log_reg_predict_metric.append(accuracy_score(Y,log_reg.predict(X
            elif k % 3 == 1:
                log_reg_predict_metric.append(roc_auc_score(Y,log_reg.predict(X)
            else:
                log_reg_predict_metric.append(f1_score(Y,log_reg.predict(X)))

        #Record results for the trial
        log_reg_predict_trial.append(log_reg_predict_metric)
        log_reg_sample_trial.append(log_reg_sample_metric)

    #record results for the dataset
    log_reg_predict_data.append(log_reg_predict_trial)
    log_reg_sample_data.append(log_reg_sample_trial)
    j+=1
print("Finished!")
```

```
LogisticRegression(C=10.0, max_iter=5000, penalty='l1', random_state=1000,
                   solver='saga')
Now working on:  Avila
Onto trial: 1
Onto trial: 2
Onto trial: 3
Onto trial: 4
```

```
Onto trial: 5
Now working on:  EEG Eye
Onto trial: 1
Onto trial: 2
Onto trial: 3
Onto trial: 4
Onto trial: 5
Now working on:  Occupancy
Onto trial: 1
Onto trial: 2
Onto trial: 3
Onto trial: 4
Onto trial: 5
Now working on:  Letter
Onto trial: 1
Onto trial: 2
Onto trial: 3
Onto trial: 4
Onto trial: 5
Finished!
```

In [955…

```
#instantiate lists to be used for ttests here were working with scores for the w
log_reg_avg_metric = []
log_reg_avg_data = []
log_acc = []
log_roc = []
log_f1 = []
temp = []
log_data1 = []

#loop through the score data for the whole datasets
for i in log_reg_predict_data:

    #scrub lists
    log_reg_avg_metric = []
    temp = []

    #iterate for each trial
    for j in range(5):

        #Get the error metric values into their respective lists
        log_acc.append(i[j][0])
        log_roc.append(i[j][1])
        log_f1.append(i[j][2])

        #temp is used to format data1 so the error metrics go in in lists of 3
        for k in range(3):
            temp.append(i[j][k])

    #append temp to data1 which will hold all of the data for each dataset and t
    log_data1.append(temp)

    #gather the means of each metric for all trials
    log_reg_avg_metric.append(np.mean([i[0][0],i[1][0],i[2][0],i[3][0],i[4][0]])
    log_reg_avg_metric.append(np.mean([i[0][1],i[1][1],i[2][1],i[3][1],i[4][1]])
    log_reg_avg_metric.append(np.mean([i[0][2],i[1][2],i[2][2],i[3][2],i[4][2]])
    log_reg_avg_data.append(log_reg_avg_metric)

print(log_data1)
```

```
[[0.8607370489289309, 0.744731689153132, 0.9131084798469083, 0.8607370489289309,
0.744731689153132, 0.9131084798469083, 0.8607370489289309, 0.744731689153132, 0.
```

```
9131084798469083, 0.8607370489289309, 0.744731689153132, 0.9131084798469083, 0.8
607370489289309, 0.744731689153132, 0.9131084798469083], [0.8607370489289309, 0.
744731689153132, 0.9131084798469083, 0.8607370489289309, 0.744731689153132, 0.91
31084798469083, 0.8607370489289309, 0.744731689153132, 0.9131084798469083, 0.860
7370489289309, 0.744731689153132, 0.9131084798469083, 0.8607370489289309, 0.7447
31689153132, 0.9131084798469083], [0.8607370489289309, 0.744731689153132, 0.9131
084798469083, 0.8607370489289309, 0.744731689153132, 0.9131084798469083, 0.86073
70489289309, 0.744731689153132, 0.9131084798469083, 0.8607370489289309, 0.744731
689153132, 0.9131084798469083, 0.8607370489289309, 0.744731689153132, 0.91310847
98469083], [0.8607370489289309, 0.744731689153132, 0.9131084798469083, 0.8607370
489289309, 0.744731689153132, 0.9131084798469083, 0.8607370489289309, 0.74473168
9153132, 0.9131084798469083, 0.8607370489289309, 0.744731689153132, 0.9131084798
469083, 0.8607370489289309, 0.744731689153132, 0.9131084798469083]]
```

In [944…

```
#instantiate lists to be used for ttests here were working with scores for the s
log_reg_sample_avg_metric = []
log_reg_sample_avg_data = []
log_sample_acc = []
log_sample_roc = []
log_sample_f1 = []
temp = []
log_data2 = []

#loop through the score data for the sampled datasets
for i in log_reg_sample_data:

    #scrub lists
    log_reg_sample_avg_metric = []
    temp = []

    #iterate for each trial
    for j in range(5):

        #Get the error metric values into their respective lists
        log_sample_acc.append(i[j][0])
        log_sample_roc.append(i[j][1])
        log_sample_f1.append(i[j][2])

        #temp is used to format data1 so the error metrics go in in lists of 3
        for k in range(3):
            temp.append(i[j][k])

    #append temp to data1 which will hold all of the data for each dataset and t
    log_data2.append(temp)

    #gather the means of each metric for all trials
    log_reg_sample_avg_metric.append(np.mean([i[0][0],i[1][0],i[2][0],i[3][0],i[
    log_reg_sample_avg_metric.append(np.mean([i[0][1],i[1][1],i[2][1],i[3][1],i[
    log_reg_sample_avg_metric.append(np.mean([i[0][2],i[1][2],i[2][2],i[3][2],i[
    log_reg_sample_avg_data.append(log_reg_sample_avg_metric)
print(log_reg_sample_avg_data)
```

```
[[0.8612, 0.7510454600786453, 0.9129907934591974], [0.64276, 0.628695333450083,
0.5527368053801064], [0.98864, 0.9870964623764469, 0.9741458661548628], [0.72972
00000000001, 0.7298269699162919, 0.7303042008734786]]
```

In [830…

```
# Create decision tree classifier
dec_tree = DecisionTreeClassifier()

# Create search space of hyperparameters for dectree model
search_space = [{
```

```
                    'criterion': ['gini','entropy'],
                    'splitter': ['best', 'random'],
                    'max_depth': [None,2,4,6,8,10,12],
                    'min_samples_split': [2,4,6,8,10],
                    'min_samples_leaf': [2,4,6,8,10],
                    'max_features': ['auto','sqrt','log2',None],
                    'random_state':[1000]}
                   ]

# instantiate lists to store data from loop
best_dectree_trials = []
best_dectree_by_data = []



# Loop through each dataframe, collecting the df and Y column
for data_set, target_name in data_list:

    #print line for monitoring purposes
    print("Now working on: ", data_set.name)
    #reset list for later iteration
    best_dectree_trials = []

    for i in range(5):

        #set up gridsearch instance for three performance metrics
        clf = GridSearchCV(dec_tree, search_space, cv=StratifiedKFold(n_splits=5
                     scoring=['accuracy', 'roc_auc_ovr', 'f1_micro'], refit=False,
                     verbose=0,n_jobs=-1)
        print('Onto trial: ', i + 1)

        # Set X to a sample of 5000 from the current dataset, make Y the relevan
        X = data_set.sample(n = 5000, random_state = i * 5, axis = 0)
        Y = X[target_name]
        X = X.drop([target_name],axis=1)

        #add the results from each trial to the trial list
        best_dectree_trials.append(clf.fit(X, Y))

    #add the results from each set of trials data list
    best_dectree_by_data.append(best_dectree_trials)

print("Finished!")
```

```
Now working on:  Avila
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  EEG Eye
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  Occupancy
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
```

```
Now working on:  Letter
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Finished!
```

In [929…

```
# establish iteration lists for both the test data and training data
dec_tree_predict_metric = []
dec_tree_predict_trial = []
dec_tree_predict_data = []

dec_tree_sample_metric = []
dec_tree_sample_trial = []
dec_tree_sample_data = []
i=0

#iterate through the data list again
for data_set, target_name in data_list:

    #print to let me know where the program is running
    print("Now Working On: ", data_set.name)

    #reset lists to collect trial data
    dec_tree_predict_trial = []
    dec_tree_sample_trial = []

    for j in range(5):

        #print to let me know where the program is running
        print("Onto Trial: ", j + 1)
        temp = best_dectree_by_data[i][j].cv_results_

        #gather best params from gridsearch by performance metric
        error_metric = []
        error_metric.append(np.where(temp['rank_test_accuracy'] == 1))
        error_metric.append(np.where(temp['rank_test_roc_auc_ovr'] == 1))
        error_metric.append(np.where(temp['rank_test_f1_micro'] == 1))

        #reset lists to collect trial data
        dec_tree_predict_metric = []
        dec_tree_sample_metric = []

        for k in range(3):

            #make decision tree instance with best params for each error metric
            error_metric[k] = error_metric[k][0][0]
            best_temp = temp['params'][error_metric[k]]
            dec_tree = DecisionTreeClassifier(criterion=best_temp['criterion'],m

            # set X and Y to the same data sample as before
            X = data_set.sample(n = 5000, random_state = i * 5, axis = 0)
            Y = X[target_name]
            X = X.drop([target_name], axis=1)
            dec_tree.fit(X,Y)

            # Predict and score the model on the same training data
            if k % 3 == 0:
                dec_tree_sample_metric.append(accuracy_score(Y,dec_tree.predict(
            elif k % 3 == 1:
```

```
                dec_tree_sample_metric.append(roc_auc_score(Y,dec_tree.predict(X
            else:
                dec_tree_sample_metric.append(f1_score(Y,dec_tree.predict(X)))


            #Set X and Y as the whole dataset
            X = data_set.drop([target_name], axis=1)
            Y = data_set[target_name]

            # Using the same fit model from above, predict and score it for the
            if k % 3 == 0:
                dec_tree_predict_metric.append(accuracy_score(Y,dec_tree.predict
            elif k % 3 == 1:
                dec_tree_predict_metric.append(roc_auc_score(Y,dec_tree.predict(
            else:
                dec_tree_predict_metric.append(f1_score(Y,dec_tree.predict(X)))

        #Record results for the trial
        dec_tree_sample_trial.append(dec_tree_sample_metric)
        dec_tree_predict_trial.append(dec_tree_predict_metric)

    #record results for the dataset
    dec_tree_predict_data.append(dec_tree_predict_trial)
    dec_tree_sample_data.append(dec_tree_sample_trial)
    i+=1
print("Finished!")
```

```
Now Working On:  Avila
Onto Trial:  1
Onto Trial:  2
Onto Trial:  3
Onto Trial:  4
Onto Trial:  5
Now Working On:  EEG Eye
Onto Trial:  1
Onto Trial:  2
Onto Trial:  3
Onto Trial:  4
Onto Trial:  5
Now Working On:  Occupancy
Onto Trial:  1
Onto Trial:  2
Onto Trial:  3
Onto Trial:  4
Onto Trial:  5
Now Working On:  Letter
Onto Trial:  1
Onto Trial:  2
Onto Trial:  3
Onto Trial:  4
Onto Trial:  5
Finished!
```

In [928…

```
print(dec_tree_sample_data)
```

```
[[[0.9716298461685916, 0.9231172583527412, 0.9809878604920035], [0.9716298461685
916, 0.929538283578492, 0.9809878604920035], [0.9716298461685916, 0.919378998125
3584, 0.9809878604920035], [0.9589782910816121, 0.9143955301309574, 0.9723460618
983006], [0.9716298461685916, 0.8982615420701316, 0.9809878604920035]], [[0.8487
983978638184, 0.8224334727613192, 0.8299932447646926], [0.8391188251001335, 0.81
36398666582864, 0.8187424789410349], [0.859479305740988, 0.8136398666582864, 0.8
418363513411976], [0.814218958611482, 0.8026656525401413, 0.7898829747074368],
```

```
[0.859479305740988, 0.8224334727613192, 0.8418363513411976]], [[0.99607024438167
75, 0.9890084641697068, 0.9907460960092539], [0.9927545130787179, 0.991741875376
3592, 0.9829134086301767], [0.9939825617094437, 0.9874468498366878, 0.9858012170
385396], [0.9942281714355888, 0.9839238396054899, 0.9863491141446412], [0.995824
6346555324, 0.9890964729229709, 0.9901677270098322]], [[0.9008, 0.86348658551707
85, 0.8984335005631207], [0.8936, 0.8699036165301951, 0.8901847455877799], [0.90
08, 0.8699036165301951, 0.8984335005631207], [0.9045, 0.8645381233724414, 0.9029
37290375038], [0.9045, 0.8699036165301951, 0.902937290375038]]]
```

In [911…

```python
#instantiate lists to be used for ttests here were working with scores for the w
dec_tree_avg_metric = []
dec_tree_avg_data = []
dec_acc = []
dec_roc = []
dec_f1 = []
temp = []
dec_data1 = []

#loop through the score data for the whole datasets
for i in dec_tree_predict_data:

    #scrub lists
    temp = []

    #iterate for each trial
    for j in range(5):

        #Get the error metric values into their respective lists
        dec_acc.append(i[j][0])
        dec_roc.append(i[j][1])
        dec_f1.append(i[j][2])

        #temp is used to format data1 so the error metrics go in in lists of 3
        for k in range(3):
            temp.append(i[j][k])

    #append temp to data1 which will hold all of the data for each dataset and t
    dec_data1.append(temp)

    #gather the means of each metric for all trials
    dec_tree_avg_metric = []
    dec_tree_avg_metric.append(np.mean([i[0][0],i[1][0],i[2][0],i[3][0],i[4][0]]
    dec_tree_avg_metric.append(np.mean([i[0][1],i[1][1],i[2][1],i[3][1],i[4][1]]
    dec_tree_avg_metric.append(np.mean([i[0][2],i[1][2],i[2][2],i[3][2],i[4][2]]
    dec_tree_avg_data.append(dec_tree_avg_metric)
print(dec_tree_avg_data)
```

```
[[0.9690995351511956, 0.9169383224515361, 0.979259500773263], [0.84421895861148
2, 0.8149624662758704, 0.824458280219112], [0.9945720250521921, 0.98824350038224
3, 0.9871955125664889], [0.90084, 0.8675471116960211, 0.8985852654928195]]
```

In [945…

```python
#instantiate lists to be used for ttests here were working with scores for the s
dec_tree_sample_avg_metric = []
dec_tree_sample_avg_data = []
dec_sample_acc = []
dec_sample_roc = []
dec_sample_f1 = []
temp = []
dec_data2 = []
```

```
#loop through the score data for the sampled datasets
for i in dec_tree_sample_data:

    #scrub lists
    temp = []

    #iterate for each trial
    for j in range(5):

        #Get the error metric values into their respective lists
        dec_sample_acc.append(i[j][0])
        dec_sample_roc.append(i[j][1])
        dec_sample_f1.append(i[j][2])

        #temp is used to format data1 so the error metrics go in in lists of 3
        for k in range(3):
            temp.append(i[j][k])

    #append temp to data1 which will hold all of the data for each dataset and t
    dec_data2.append(temp)

    #gather the means of each metric for all trials
    dec_tree_sample_avg_metric = []
    dec_tree_sample_avg_metric.append(np.mean([i[0][0],i[1][0],i[2][0],i[3][0],i
    dec_tree_sample_avg_metric.append(np.mean([i[0][1],i[1][1],i[2][1],i[3][1],i
    dec_tree_sample_avg_metric.append(np.mean([i[0][2],i[1][2],i[2][2],i[3][2],i
    dec_tree_sample_avg_data.append(dec_tree_sample_avg_metric)
print(dec_tree_sample_avg_data)
```

```
[[0.99448, 0.9388680987639477, 0.9962634087052418], [0.94176, 0.873127518021153
6, 0.9336664475183228], [0.9963599999999999, 0.9896067697870631, 0.9913051550939
619], [0.97256, 0.9030489952569335, 0.9717230593093819]]
```

In [836…

```
# Create random forest classifier
rand_forest = RandomForestClassifier()

# Create search space of hyperparameters for dectree model
search_space = [{
                'n_estimators': [1024],
                'warm_start':[True,False],
                'criterion': ['gini', 'entropy'],
                'max_features': ['sqrt','log2',None,1,2,4,6,7],
                'random_state' : [1000]
               }]

# instantiate lists to store data from loop
best_rf_trials = []
best_rf_by_data = []



# Loop through each dataframe, collecting the df and Y column
for data_set, target_name in data_list:

    #print line for monitoring purposes
    print("Now working on: ", data_set.name)
    #reset list for later iteration
    best_rf_trials = []

    for i in range(5):
```

```
        #set up gridsearch instance for three performance metrics
        clf = GridSearchCV(rand_forest, search_space, cv=StratifiedKFold(n_split
                     scoring=['accuracy', 'roc_auc_ovr', 'f1_micro'], refit=False,
                     verbose=0,n_jobs=-1)
        print('Onto trial: ', i + 1)

        # Set X to a sample of 5000 from the current dataset, make Y the relevan
        X = data_set.sample(n = 5000, random_state = i * 5, axis = 0)
        Y = X[target_name]
        X = X.drop([target_name],axis=1)

        #add the results from each trial to the trial list
        best_rf_trials.append(clf.fit(X, Y))

    #add the results from each set of trials data list
    best_rf_by_data.append(best_rf_trials)

print("Finished!")
```

```
Now working on:  Avila
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  EEG Eye
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  Occupancy
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Now working on:  Letter
Onto trial:  1
Onto trial:  2
Onto trial:  3
Onto trial:  4
Onto trial:  5
Finished!
```

In [934…

```
# establish iteration lists for both the test data and training data
rf_predict_metric = []
rf_predict_trial = []
rf_predict_data = []

rf_sample_metric = []
rf_sample_trial = []
rf_sample_data = []

i=0

#iterate through the data list again
for data_set, target_name in data_list:

    #print to let me know where the program is running
    print("Now Working On: ", data_set.name)
```

```
    #reset lists to collect trial data
    rf_predict_trial = []
    rf_sample_trial = []

    for j in range(5):

        #print to let me know where the program is running
        print("Onto Trial: ", j + 1)
        temp = best_rf_by_data[i][j].cv_results_

        #gather best params from gridsearch by performance metric
        error_metric = []
        error_metric.append(np.where(temp['rank_test_accuracy'] == 1))
        error_metric.append(np.where(temp['rank_test_roc_auc_ovr'] == 1))
        error_metric.append(np.where(temp['rank_test_f1_micro'] == 1))

        #reset lists to collect trial data
        rf_predict_metric = []
        rf_sample_metric = []

        for k in range(3):

            #make decision tree instance with best params for each error metric
            print("Working on metric: ", k)
            error_metric[k] = error_metric[k][0][0]
            best_temp = temp['params'][error_metric[k]]
            rf = RandomForestClassifier(criterion=best_temp['criterion'],max_fea

            # set X and Y to the same data sample as before
            X = data_set.sample(n = 5000, random_state = i * 5, axis = 0)
            Y = X[target_name]
            X = X.drop([target_name], axis=1)
            rf.fit(X,Y)

            # Predict and score the model on the same training data
            if k % 3 == 0:
                rf_sample_metric.append(accuracy_score(Y,rf.predict(X)))
            elif k % 3 == 1:
                rf_sample_metric.append(roc_auc_score(Y,rf.predict(X)))
            else:
                rf_sample_metric.append(f1_score(Y,rf.predict(X)))

            #Set X and Y as the whole dataset
            X = data_set.drop([target_name], axis=1)
            Y = data_set[target_name]

            # Using the same fit model from above, predict and score it for the
            if k % 3 == 0:
                rf_predict_metric.append(accuracy_score(Y,rf.predict(X)))
            elif k % 3 == 1:
                rf_predict_metric.append(roc_auc_score(Y,rf.predict(X)))
            else:
                rf_predict_metric.append(f1_score(Y,rf.predict(X)))

        #Record results for the trial
        rf_predict_trial.append(rf_predict_metric)
        rf_sample_trial.append(rf_sample_metric)

    #record results for the dataset
    rf_predict_data.append(rf_predict_trial)
```

```
    rf_sample_data.append(rf_sample_trial)
    i+=1
print("Finished!")
```

```
Now Working On:  Avila
Onto Trial:  1
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  2
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  3
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  4
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  5
Working on metric:  0
Working on metric:  1
Working on metric:  2
Now Working On:  EEG Eye
Onto Trial:  1
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  2
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  3
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  4
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  5
Working on metric:  0
Working on metric:  1
Working on metric:  2
Now Working On:  Occupancy
Onto Trial:  1
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  2
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  3
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  4
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  5
```

```
Working on metric:  0
Working on metric:  1
Working on metric:  2
Now Working On:  Letter
Onto Trial:  1
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  2
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  3
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  4
Working on metric:  0
Working on metric:  1
Working on metric:  2
Onto Trial:  5
Working on metric:  0
Working on metric:  1
Working on metric:  2
Finished!
```

In [899…

```
#instantiate lists to be used for ttests here were working with scores for the w
rf_avg_metric = []
rf_avg_data = []
rf_acc = []
rf_roc = []
rf_f1 = []
rf_data1 = []
temp = []

#loop through the score data for the whole datasets
for i in rf_predict_data:

    #scrub lists
    temp = []

    #iterate for each trial
    for j in range(5):

        #Get the error metric values into their respective lists
        rf_acc.append(i[j][0])
        rf_roc.append(i[j][1])
        rf_f1.append(i[j][2])

        #temp is used to format data1 so the error metrics go in in lists of 3
        for k in range(3):
            temp.append(i[j][k])

    #append temp to data1 which will hold all of the data for each dataset and t
    rf_data1.append(temp)

    #gather the means of each metric for all trials
    rf_avg_metric = []
    rf_avg_metric.append(np.mean([i[0][0],i[1][0],i[2][0],i[3][0],i[4][0]]))
    rf_avg_metric.append(np.mean([i[0][1],i[1][1],i[2][1],i[3][1],i[4][1]]))
    rf_avg_metric.append(np.mean([i[0][2],i[1][2],i[2][2],i[3][2],i[4][2]]))
```

```
    rf_avg_data.append(rf_avg_metric)
print(len(rf_data1[3]))
```

15

In [946…

```
#instantiate lists to be used for ttests here were working with scores for the s
rf_sample_avg_metric = []
rf_sample_avg_data = []
rf_sample_acc = []
rf_sample_roc = []
rf_sample_f1 = []
rf_data2 = []
temp = []

#loop through the score data for the sampled datasets
for i in rf_sample_data:

    #scrub lists
    temp = []

    #iterate for each trial
    for j in range(5):

        #Get the error metric values into their respective lists
        rf_sample_acc.append(i[j][0])
        rf_sample_roc.append(i[j][1])
        rf_sample_f1.append(i[j][2])

        #temp is used to format data1 so the error metrics go in in lists of 3
        for k in range(3):
            temp.append(i[j][k])

    #append temp to data1 which will hold all of the data for each dataset and t
    rf_data2.append(temp)

    #gather the means of each metric for all trials
    rf_sample_avg_metric = []
    rf_sample_avg_metric.append(np.mean([i[0][0],i[1][0],i[2][0],i[3][0],i[4][0]
    rf_sample_avg_metric.append(np.mean([i[0][1],i[1][1],i[2][1],i[3][1],i[4][1]
    rf_sample_avg_metric.append(np.mean([i[0][2],i[1][2],i[2][2],i[3][2],i[4][2]
    rf_sample_avg_data.append(rf_sample_avg_metric)
print(rf_sample_avg_data)
```

[[1.0, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 1.0, 1.0]]

In [885…

```
#Find ttest results for each performance metric - rf outperformed for each one
log_acc_p = stats.ttest_rel(log_acc, rf_acc)
log_roc_p = stats.ttest_rel(log_roc, rf_roc)
log_f1_p = stats.ttest_rel(log_f1, rf_f1)
dec_acc_p = stats.ttest_rel(dec_acc, rf_acc)
dec_roc_p = stats.ttest_rel(dec_roc, rf_roc)
dec_f1_p = stats.ttest_rel(dec_f1, rf_f1)
print(log_acc_p)
print(log_roc_p)
print(log_f1_p)
print(dec_acc_p)
print(dec_roc_p)
print(dec_f1_p)
```

```
Ttest_relResult(statistic=-5.974231437564191, pvalue=9.484476701867612e-06)
Ttest_relResult(statistic=-6.791101412138696, pvalue=1.7439128162376142e-06)
Ttest_relResult(statistic=-4.300240917781898, pvalue=0.000386165792042603)
Ttest_relResult(statistic=-4.981181411790205, pvalue=8.291283065958167e-05)
Ttest_relResult(statistic=-7.096573467152287, pvalue=9.47097243970075e-07)
Ttest_relResult(statistic=-4.817784836859598, pvalue=0.00011958662733499019)
```

In [947…

```
log_sample_acc_p = stats.ttest_rel(log_sample_acc, rf_sample_acc)
log_sample_roc_p = stats.ttest_rel(log_sample_roc, rf_sample_roc)
log_sample_f1_p = stats.ttest_rel(log_sample_f1, rf_sample_f1)
dec_sample_acc_p = stats.ttest_rel(dec_sample_acc, rf_sample_acc)
dec_sample_roc_p = stats.ttest_rel(dec_sample_roc, rf_sample_roc)
dec_sample_f1_p = stats.ttest_rel(dec_sample_f1, rf_sample_f1)
print(log_sample_acc_p)
print(log_sample_roc_p)
print(log_sample_f1_p)
print(dec_sample_acc_p)
print(dec_sample_roc_p)
print(dec_sample_f1_p)
```

```
Ttest_relResult(statistic=-6.456542442933555, pvalue=3.452626386121358e-06)
Ttest_relResult(statistic=-7.490032681763721, pvalue=4.39511724667642e-07)
Ttest_relResult(statistic=-5.471867838164947, pvalue=2.8019680512875767e-05)
Ttest_relResult(statistic=-3.6466024646191233, pvalue=0.0017163635816104868)
Ttest_relResult(statistic=-7.297293044233187, pvalue=6.385074685333651e-07)
Ttest_relResult(statistic=-3.6510256040715885, pvalue=0.0016991603325284923)
```

In [912…

```
#Find ttest results for each dataset - rf outperformed for each
onelog_ttest_data = []
dec_ttest_data = []
for i in range(4):
    log_ttest_data.append(stats.ttest_rel(log_data1[i], rf_data1[i]))
    dec_ttest_data.append(stats.ttest_rel(dec_data1[i], rf_data1[i]))
for i in range(4):
    print(log_ttest_data[i])
for i in range(4):
    print(dec_ttest_data[i])
```

```
Ttest_relResult(statistic=-8.356011171444266, pvalue=8.219871082473357e-07)
Ttest_relResult(statistic=-13.53271940374474, pvalue=1.9707318517189897e-09)
Ttest_relResult(statistic=-8.710989480811444, pvalue=5.014367108187927e-07)
Ttest_relResult(statistic=-319.43018872840787, pvalue=1.915883757368666e-28)
Ttest_relResult(statistic=-4.458379785191511, pvalue=0.0005406344713840243)
Ttest_relResult(statistic=-20.494575636540482, pvalue=7.720066362459828e-12)
Ttest_relResult(statistic=-6.918418666417257, pvalue=7.112154071582886e-06)
Ttest_relResult(statistic=-17.287956340982735, pvalue=7.6757347712904e-11)
```

In [948…

```
#Find ttest results for each dataset FOR SAMPLE SCOREs - rf outperformed for eac
log_sample_ttest_data = []
dec_sample_ttest_data = []
for i in range(4):
    log_sample_ttest_data.append(stats.ttest_rel(log_data2[i], rf_data2[i]))
    dec_sample_ttest_data.append(stats.ttest_rel(dec_data2[i], rf_data2[i]))
for i in range(4):
    print(log_ttest_data[i])
for i in range(4):
    print(dec_ttest_data[i])
```

```
Ttest_relResult(statistic=-8.356011171444266, pvalue=8.219871082473357e-07)
Ttest_relResult(statistic=-13.53271940374474, pvalue=1.9707318517189897e-09)
```

```
Ttest_relResult(statistic=-8.710989480811444, pvalue=5.014367108187927e-07)
Ttest_relResult(statistic=-319.43018872840787, pvalue=1.915883757368666e-28)
Ttest_relResult(statistic=-4.458379785191511, pvalue=0.0005406344713840243)
Ttest_relResult(statistic=-20.494575636540482, pvalue=7.720066362459828e-12)
Ttest_relResult(statistic=-6.918418666417257, pvalue=7.112154071582886e-06)
Ttest_relResult(statistic=-17.287956340982735, pvalue=7.6757347712904e-11)
```

In [926…

```

```

```
[0.9755884017919603, 0.9713768041539348, 0.9636563101598299, 0.9755884017919603,
0.9690019079166813, 0.9337696170640155, 0.9337696170640155, 0.9337696170640155,
0.9340553310357682, 0.9337696170640155, 0.9964618858075606, 0.9965398402821474,
0.9971182092815689, 0.9971182092815689, 0.9971182092815689, 0.9618459264533523,
0.9618459264533523, 0.9618459264533523, 0.9617908244696808, 0.9617908244696808]
```

In [ ]:

```

```

In [ ]:

```

```

In [514…

```python
# there were a lot of hyperparameter sets for DT, just used this to make sure no
one = [0,0,0]
two = [0,0,0]
three = [0,0,0]
four = [0,0,0]
five = [0,0,0]
six = [0,0,0]
seven = [0,0,0]
eight = [0,0,0]
nine = [0,0,0]
ten = [0,0,0]

data_i = 0
trial_i = 4

for i in best_dectree_by_data[data_i][trial_i].cv_results_['rank_test_f1_micro']
    if i == 1:
        one[1] += 1
    elif i == 2:
        two[1] += 1
    elif i == 3:
        three[1] += 1
    elif i == 4:
        four[1] += 1
    elif i == 5:
        five[1] += 1
    elif i == 6:
        six[1] += 1
    elif i == 7:
        seven[1] += 1
    elif i == 8:
        eight[1] += 1
    elif i == 9:
        nine[1] += 1
    elif i == 10:
        ten[1] += 1

for i in best_dectree_by_data[data_i][trial_i].cv_results_['rank_test_roc_auc_ov
    if i == 1:
```

```
        one[2] += 1
    elif i == 2:
        two[2] += 1
    elif i == 3:
        three[2] += 1
    elif i == 4:
        four[2] += 1
    elif i == 5:
        five[2] += 1
    elif i == 6:
        six[2] += 1
    elif i == 7:
        seven[2] += 1
    elif i == 8:
        eight[2] += 1
    elif i == 9:
        nine[2] += 1
    elif i == 10:
        ten[2] += 1

for i in best_dectree_by_data[data_i][trial_i].cv_results_['rank_test_accuracy']
    if i == 1:
        one[0] += 1
    elif i == 2:
        two[0] += 1
    elif i == 3:
        three[0] += 1
    elif i == 4:
        four[0] += 1
    elif i == 5:
        five[0] += 1
    elif i == 6:
        six[0] += 1
    elif i == 7:
        seven[0] += 1
    elif i == 8:
        eight[0] += 1
    elif i == 9:
        nine[0] += 1
    elif i == 10:
        ten[0] += 1


print("1: ", one, "\n2: ", two, "\n3: ", three,"\n4: ", four,"\n5: ", five,"\n6:
```

```
1:  [1, 1, 1]
2:  [1, 1, 1]
3:  [1, 1, 1]
4:  [1, 1, 1]
5:  [1, 1, 1]
6:  [1, 1, 1]
7:  [1, 1, 1]
8:  [1, 1, 1]
9:  [1, 1, 1]
10:  [1, 1, 1]
```

In [517…

```
best_rf_by_data[1][4].cv_results_
```

Out[517…

```
{'mean_fit_time': array([11.46271429, 11.29431119, 11.32947607, 11.39440274, 36.
46786246,
```

```
        35.65666256,  6.64373112,  6.63992367,  8.9275959 ,  8.92514262,
        13.80256076, 13.45937891, 17.69512396, 18.04195752, 20.11038904,
        19.91704741, 13.22894216, 13.21112285, 13.55325007, 13.27415004,
        41.46789517, 41.60472412,  7.64556537,  7.87799215, 10.60360079,
        10.58217134, 15.90134616, 16.09247012, 21.05986462, 21.14847856,
        23.72878122, 21.91211243]),
 'std_fit_time': array([0.05539146, 0.04789704, 0.08633709, 0.07707696, 0.461041
1 ,
        0.33881858, 0.02203739, 0.03147888, 0.01746569, 0.06160537,
        0.13066327, 0.14335565, 0.04810801, 0.21255233, 0.18046018,
        0.11438521, 0.03119574, 0.05637657, 0.16985357, 0.03942204,
        0.33390318, 0.25159209, 0.05662933, 0.06693483, 0.07007457,
        0.05309654, 0.1513985 , 0.24411376, 0.10119815, 0.19117601,
        0.06615567, 2.37611958]),
 'mean_score_time': array([0.76102672, 0.77615757, 0.76550784, 0.77270627, 0.728
73096,
        0.72153788, 0.78117929, 0.79397197, 0.74629788, 0.77001119,
        0.75252595, 0.73471928, 0.72581787, 0.72916865, 0.71478944,
        0.72105184, 0.73287768, 0.73211164, 0.7231173 , 0.71555958,
        0.6986361 , 0.70342469, 0.7758111 , 0.80634623, 0.74901295,
        0.7374887 , 0.71855288, 0.72686849, 0.70873437, 0.70243297,
        0.70407619, 0.59780369]),
 'std_score_time': array([0.00697666, 0.01097072, 0.01144377, 0.00581714, 0.0145
0218,
        0.01474566, 0.01924574, 0.01264844, 0.01136249, 0.01783136,
        0.02428909, 0.01045217, 0.02823145, 0.03235582, 0.01271809,
        0.0193456 , 0.02551303, 0.0062784 , 0.02571356, 0.01050658,
        0.02688484, 0.02219419, 0.02756521, 0.01744525, 0.02117474,
        0.00980752, 0.01279551, 0.0174182 , 0.0068577 , 0.00903347,
        0.01827936, 0.11670455]),
 'param_criterion': masked_array(data=['gini', 'gini', 'gini', 'gini', 'gini',
'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_features': masked_array(data=['sqrt', 'sqrt', 'log2', 'log2', None,
None, 1, 1, 2, 2,
                    4, 4, 6, 6, 7, 7, 'sqrt', 'sqrt', 'log2', 'log2', None,
                    None, 1, 1, 2, 2, 4, 4, 6, 6, 7, 7],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[1024, 1024, 1024, 1024, 1024, 1024, 10
24, 1024, 1024,
                    1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
                    1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024,
                    1024, 1024, 1024, 1024, 1024],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_warm_start': masked_array(data=[True, False, True, False, True, False, T
```

```
           rue, False,
                        True, False, True, False, True, False, True, False,
                        True, False, True, False, True, False, True, False,
                        True, False, True, False, True, False, True, False],
                  mask=[False, False, False, False, False, False, False, False,
                        False, False, False, False, False, False, False, False,
                        False, False, False, False, False, False, False, False,
                        False, False, False, False, False, False, False, False],
            fill_value='?',
                 dtype=object),
 'params': [{'criterion': 'gini',
   'max_features': 'sqrt',
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': 'sqrt',
   'n_estimators': 1024,
   'warm_start': False},
  {'criterion': 'gini',
   'max_features': 'log2',
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': 'log2',
   'n_estimators': 1024,
   'warm_start': False},
  {'criterion': 'gini',
   'max_features': None,
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': None,
   'n_estimators': 1024,
   'warm_start': False},
  {'criterion': 'gini',
   'max_features': 1,
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': 1,
   'n_estimators': 1024,
   'warm_start': False},
  {'criterion': 'gini',
   'max_features': 2,
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': 2,
   'n_estimators': 1024,
   'warm_start': False},
  {'criterion': 'gini',
   'max_features': 4,
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': 4,
   'n_estimators': 1024,
   'warm_start': False},
  {'criterion': 'gini',
   'max_features': 6,
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'gini',
   'max_features': 6,
   'n_estimators': 1024,
```

```
 'warm_start': False},
{'criterion': 'gini',
 'max_features': 7,
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'gini',
 'max_features': 7,
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': 'sqrt',
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': 'sqrt',
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': 'log2',
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': 'log2',
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': None,
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': None,
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': 1,
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': 1,
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': 2,
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': 2,
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': 4,
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': 4,
 'n_estimators': 1024,
 'warm_start': False},
{'criterion': 'entropy',
 'max_features': 6,
 'n_estimators': 1024,
 'warm_start': True},
{'criterion': 'entropy',
 'max_features': 6,
 'n_estimators': 1024,
 'warm_start': False},
```

```
  {'criterion': 'entropy',
   'max_features': 7,
   'n_estimators': 1024,
   'warm_start': True},
  {'criterion': 'entropy',
   'max_features': 7,
   'n_estimators': 1024,
   'warm_start': False}],
 'split0_test_accuracy': array([0.888, 0.884, 0.886, 0.892, 0.895, 0.891, 0.874,
0.869, 0.887,
        0.878, 0.892, 0.894, 0.898, 0.896, 0.898, 0.9  , 0.89 , 0.893,
        0.885, 0.891, 0.902, 0.902, 0.874, 0.87 , 0.886, 0.887, 0.896,
        0.899, 0.897, 0.895, 0.895, 0.896]),
 'split1_test_accuracy': array([0.872, 0.873, 0.88 , 0.874, 0.874, 0.869, 0.875,
0.877, 0.879,
        0.883, 0.882, 0.875, 0.877, 0.875, 0.881, 0.88 , 0.879, 0.874,
        0.878, 0.879, 0.876, 0.874, 0.873, 0.875, 0.88 , 0.882, 0.882,
        0.882, 0.881, 0.881, 0.879, 0.882]),
 'split2_test_accuracy': array([0.892, 0.891, 0.896, 0.889, 0.884, 0.881, 0.878,
0.881, 0.889,
        0.888, 0.897, 0.893, 0.893, 0.895, 0.894, 0.894, 0.892, 0.896,
        0.896, 0.896, 0.889, 0.889, 0.879, 0.885, 0.897, 0.89 , 0.897,
        0.894, 0.903, 0.901, 0.903, 0.901]),
 'split3_test_accuracy': array([0.915, 0.915, 0.91 , 0.912, 0.896, 0.898, 0.897,
0.897, 0.906,
        0.914, 0.914, 0.914, 0.908, 0.91 , 0.91 , 0.901, 0.913, 0.915,
        0.911, 0.916, 0.901, 0.901, 0.906, 0.905, 0.91 , 0.909, 0.91 ,
        0.911, 0.912, 0.907, 0.908, 0.906]),
 'split4_test_accuracy': array([0.901, 0.895, 0.895, 0.892, 0.878, 0.884, 0.869,
0.876, 0.888,
        0.887, 0.901, 0.893, 0.897, 0.897, 0.896, 0.894, 0.899, 0.894,
        0.9  , 0.899, 0.891, 0.893, 0.881, 0.874, 0.894, 0.894, 0.906,
        0.905, 0.906, 0.902, 0.908, 0.907]),
 'mean_test_accuracy': array([0.8936, 0.8916, 0.8934, 0.8918, 0.8854, 0.8846, 0.
8786, 0.88  ,
        0.8898, 0.89  , 0.8972, 0.8938, 0.8946, 0.8946, 0.8958, 0.8938,
        0.8946, 0.8944, 0.894 , 0.8962, 0.8918, 0.8918, 0.8826, 0.8818,
        0.8934, 0.8924, 0.8982, 0.8982, 0.8998, 0.8972, 0.8986, 0.8984]),
 'std_test_accuracy': array([0.01423517, 0.01387948, 0.01019019, 0.0121062 , 0.0
0884534,
        0.00976934, 0.00964572, 0.00933809, 0.00884081, 0.012506  ,
        0.01053376, 0.01235152, 0.01009158, 0.01121784, 0.00926067,
        0.007494  , 0.01121784, 0.01300154, 0.01154123, 0.01202331,
        0.00945304, 0.01014692, 0.01207642, 0.01260793, 0.01022937,
        0.00917824, 0.00968297, 0.00990757, 0.01057166, 0.00895321,
        0.0108922 , 0.00909065]),
 'rank_test_accuracy': array([17, 24, 18, 21, 27, 28, 32, 31, 26, 25,  6, 15, 1
0, 10,  9, 15, 10,
        13, 14,  8, 22, 22, 29, 30, 18, 20,  5,  4,  1,  7,  2,  3],
       dtype=int32),
 'split0_test_roc_auc_ovr': array([0.95857031, 0.95916172, 0.95946554, 0.9588538
6, 0.95897134,
        0.95886196, 0.94980232, 0.94962003, 0.95803357, 0.95670896,
        0.95968023, 0.96084484, 0.96196084, 0.96263732, 0.96077192,
        0.9616631 , 0.96104333, 0.96100079, 0.96037494, 0.96201552,
        0.96225047, 0.9622606 , 0.95133758, 0.95085351, 0.95899159,
        0.95875462, 0.9624125 , 0.96269201, 0.9631295 , 0.96258669,
        0.96333609, 0.96296139]),
 'split1_test_roc_auc_ovr': array([0.95317057, 0.95406378, 0.95310576, 0.9534885
6, 0.9476959 ,
        0.94745082, 0.94779109, 0.94845542, 0.95309361, 0.95323538,
        0.95348654, 0.95385718, 0.95278169, 0.95279182, 0.95199786,
        0.95204647, 0.95650034, 0.95662592, 0.95646591, 0.95587044,
        0.95281613, 0.95221255, 0.94914001, 0.95048083, 0.95514129,
        0.95654085, 0.95661579, 0.95702897, 0.956154  , 0.95610741,
```

```
        0.95575702, 0.95560511]),
 'split2_test_roc_auc_ovr': array([0.95915565, 0.95948782, 0.96139575, 0.9608185
1, 0.94857897,
        0.94668724, 0.95672111, 0.95662592, 0.95903615, 0.95908071,
        0.95930958, 0.95887614, 0.95715455, 0.95759203, 0.95644565,
        0.9564092 , 0.96222819, 0.96278113, 0.96228288, 0.962532  ,
        0.9540658 , 0.95402732, 0.95841435, 0.9585622 , 0.96268391,
        0.96110409, 0.96131473, 0.96205401, 0.96040532, 0.96023924,
        0.95986454, 0.95999822]),
 'split3_test_roc_auc_ovr': array([0.96587595, 0.96607241, 0.96649167, 0.9662324
2, 0.95685074,
        0.95704517, 0.96156993, 0.96069496, 0.96587392, 0.96524605,
        0.9670203 , 0.96591848, 0.96428398, 0.96503338, 0.96234364,
        0.96235377, 0.96665573, 0.96756919, 0.96606836, 0.96736462,
        0.96075369, 0.96094813, 0.96179273, 0.96265353, 0.96609874,
        0.96620001, 0.96605418, 0.96697979, 0.96525212, 0.96613317,
        0.96504958, 0.96429208]),
 'split4_test_roc_auc_ovr': array([0.96179775, 0.96155886, 0.96172487, 0.9606782
1, 0.95595101,
        0.95580524, 0.95245065, 0.95278672, 0.95995546, 0.95943719,
        0.96196984, 0.96201235, 0.96188278, 0.96206499, 0.96150825,
        0.96119648, 0.96303472, 0.96298816, 0.96307319, 0.96332827,
        0.96038263, 0.96016398, 0.95382326, 0.95362891, 0.96152647,
        0.96074097, 0.96345379, 0.9644215 , 0.96443365, 0.96325539,
        0.96313594, 0.96394979]),
 'mean_test_roc_auc_ovr': array([0.95971405, 0.96006892, 0.96043672, 0.96001431,
0.95360959,
        0.95317009, 0.95366702, 0.95363661, 0.95919854, 0.95874166,
        0.9602933 , 0.9603018 , 0.95961277, 0.96002391, 0.95861347,
        0.9587338 , 0.96189246, 0.96219304, 0.96165306, 0.96222217,
        0.95805374, 0.95792252, 0.95490159, 0.9552358 , 0.9608884 ,
        0.96066811, 0.9619702 , 0.96263526, 0.96187492, 0.96166438,
        0.96142863, 0.96136132]),
 'std_test_roc_auc_ovr': array([0.00416528, 0.00388547, 0.00433423, 0.00409107,
0.00458289,
        0.00508124, 0.0049543 , 0.00452672, 0.00409359, 0.00393336,
        0.00437582, 0.00395812, 0.00412778, 0.00434299, 0.00388377,
        0.00394649, 0.00328336, 0.00352974, 0.00317643, 0.00369192,
        0.00383829, 0.0040193 , 0.00462359, 0.00470072, 0.00367167,
        0.00320915, 0.00310403, 0.00328138, 0.00329895, 0.00335449,
        0.00329479, 0.0032509 ]),
 'rank_test_roc_auc_ovr': array([19, 16, 13, 18, 31, 32, 29, 30, 21, 22, 15, 14,
20, 17, 24, 23,  5,
         3,  8,  2, 25, 26, 28, 27, 11, 12,  4,  1,  6,  7,  9, 10],
       dtype=int32),
 'split0_test_f1_micro': array([0.888, 0.884, 0.886, 0.892, 0.895, 0.891, 0.874,
0.869, 0.887,
        0.878, 0.892, 0.894, 0.898, 0.896, 0.898, 0.9  , 0.89 , 0.893,
        0.885, 0.891, 0.902, 0.902, 0.874, 0.87 , 0.886, 0.887, 0.896,
        0.899, 0.897, 0.895, 0.895, 0.896]),
 'split1_test_f1_micro': array([0.872, 0.873, 0.88 , 0.874, 0.874, 0.869, 0.875,
0.877, 0.879,
        0.883, 0.882, 0.875, 0.877, 0.875, 0.881, 0.88 , 0.879, 0.874,
        0.878, 0.879, 0.876, 0.874, 0.873, 0.875, 0.88 , 0.882, 0.882,
        0.882, 0.881, 0.881, 0.879, 0.882]),
 'split2_test_f1_micro': array([0.892, 0.891, 0.896, 0.889, 0.884, 0.881, 0.878,
0.881, 0.889,
        0.888, 0.897, 0.893, 0.893, 0.895, 0.894, 0.894, 0.892, 0.896,
        0.896, 0.896, 0.889, 0.889, 0.879, 0.885, 0.897, 0.89 , 0.897,
        0.894, 0.903, 0.901, 0.903, 0.901]),
 'split3_test_f1_micro': array([0.915, 0.915, 0.91 , 0.912, 0.896, 0.898, 0.897,
0.897, 0.906,
        0.914, 0.914, 0.914, 0.908, 0.91 , 0.91 , 0.901, 0.913, 0.915,
        0.911, 0.916, 0.901, 0.901, 0.906, 0.905, 0.91 , 0.909, 0.91 ,
        0.911, 0.912, 0.907, 0.908, 0.906]),
```

```
 'split4_test_f1_micro': array([0.901, 0.895, 0.895, 0.892, 0.878, 0.884, 0.869,
0.876, 0.888,
        0.887, 0.901, 0.893, 0.897, 0.897, 0.896, 0.894, 0.899, 0.894,
        0.9  , 0.899, 0.891, 0.893, 0.881, 0.874, 0.894, 0.894, 0.906,
        0.905, 0.906, 0.902, 0.908, 0.907]),
 'mean_test_f1_micro': array([0.8936, 0.8916, 0.8934, 0.8918, 0.8854, 0.8846, 0.
8786, 0.88  ,
        0.8898, 0.89  , 0.8972, 0.8938, 0.8946, 0.8946, 0.8958, 0.8938,
        0.8946, 0.8944, 0.894 , 0.8962, 0.8918, 0.8918, 0.8826, 0.8818,
        0.8934, 0.8924, 0.8982, 0.8982, 0.8998, 0.8972, 0.8986, 0.8984]),
 'std_test_f1_micro': array([0.01423517, 0.01387948, 0.01019019, 0.0121062 , 0.0
0884534,
        0.00976934, 0.00964572, 0.00933809, 0.00884081, 0.012506  ,
        0.01053376, 0.01235152, 0.01009158, 0.01121784, 0.00926067,
        0.007494  , 0.01121784, 0.01300154, 0.01154123, 0.01202331,
        0.00945304, 0.01014692, 0.01207642, 0.01260793, 0.01022937,
        0.00917824, 0.00968297, 0.00990757, 0.01057166, 0.00895321,
        0.0108922 , 0.00909065]),
 'rank_test_f1_micro': array([17, 24, 18, 21, 27, 28, 32, 31, 26, 25,  6, 15, 1
2, 10,  9, 15, 10,
        13, 14,  8, 22, 22, 29, 30, 18, 20,  5,  4,  1,  7,  2,  3],
       dtype=int32)}
```

In [ ]: