Load Necessary Libraries

```
# importing necessarily libraries for the binary classification task

# libraries imported for data processing and analysis
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split

# libraries imported for learning algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import pipeline

# libraries imported for performance metrics
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
```

Load & Clean Adult Income Dataset

```
# load 'Adult Income Census' data and names into pandas dataframe

# make array of column names (based on adult.names)
column_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',
    'martial-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain',
    'capital-loss', 'hours-per-week', 'native country', 'income>50K']
# load data by using read_csv from .data file
df = pd.read_csv("datasets/Adult_Income/adult.data", names=column_names)

# clean data
# replace all '?' entries with NaN
df = df.replace(to_replace=" ?", value=np.NaN)
# change focus value (adult income > or <=50K) into binary value
df = df.replace(to_replace=" >50K", value=1)
df = df.replace(to_replace=" <=50K", value=0)
# drop all samples with NaN entries
df = df.dropna()

# save new cleaned up data into csv into dataset folder
df.to_csv("datasets/Adult_Income/adult.csv", index=False)

# one-hot encode the dataframe
encoded = pd.get_dummies(df)

# move binary classifier(label) column to the end
# hold column
classifier = encoded['income>50K']
# drop column from dataframe
encoded.drop(columns=['income>50K'], inplace=True)
# reinsert into dataframe at the end
encoded['income>50K'] = classifier

adultDF = encoded
adultDF
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	...	native country_Puerto-Rico	native country_Scotland

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	...	native country_Puerto-Rico	native country_Scotland
0	39	77516	13	2174	0	40	0	0	0	0	...	0	0
1	50	83311	13	0	0	13	0	0	0	0	...	0	0
2	38	215646	9	0	0	40	0	0	1	0	...	0	0
3	53	234721	7	0	0	40	0	0	1	0	...	0	0
4	28	338409	13	0	0	40	0	0	1	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	257302	12	0	0	38	0	0	1	0	...	0	0
32557	40	154374	9	0	0	40	0	0	1	0	...	0	0
32558	58	151910	9	0	0	40	0	0	1	0	...	0	0
32559	22	201490	9	0	0	20	0	0	1	0	...	0	0
32560	52	287927	9	15024	0	40	0	0	0	1	...	0	0

30162 rows × 105 columns

Load & Clean Electrical Grid Stability Dataset

```
# load 'Electrical Grid Stability' data and names into pandas dataframe

# load data by using read_csv from .data file
df = pd.read_csv("datasets/Grid_Stability/grid_stability.csv")

# clean data
# replace string label classifiers into binary values
df = df.replace(to_replace="stable", value=1)
df = df.replace(to_replace="unstable", value=0)
# drop all samples with NaN entries
df = df.dropna()

gridDF = df
gridDF
```

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g1	g2	g3	g4
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395	-1.723086	0.650456	0.859578	0.887445	0.958034
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742	-1.255012	0.413441	0.862414	0.562139	0.781760
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210	-0.920492	0.163041	0.766689	0.839444	0.109853
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944	-0.997374	0.446209	0.976744	0.929381	0.362718
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975	-0.554305	0.797110	0.455450	0.656947	0.820923
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	2.930406	9.487627	2.376523	6.187797	3.343416	-0.658054	-1.449106	-1.236256	0.601709	0.779642	0.813512	0.608385
9996	3.392299	1.274827	2.954947	6.894759	4.349512	-1.663661	-0.952437	-1.733414	0.502079	0.567242	0.285880	0.366120
9997	2.364034	2.842030	8.776391	1.008906	4.299976	-1.380719	-0.943884	-1.975373	0.487838	0.986505	0.149286	0.145984
9998	9.631511	3.994398	2.757071	7.821347	2.514755	-0.966330	-0.649915	-0.898510	0.365246	0.587558	0.889118	0.818391

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g1	g2	g3	g4
9999	6.530527	6.781790	4.349695	8.673138	3.492807	-1.390285	-1.532193	-0.570329	0.073056	0.505441	0.378761	0.942631

10000 rows × 14 columns

Load & Clean Occupancy Dataset

```
# load 'Occupancy' data and names into pandas dataframe

# load data by using read_csv from .data file
first = pd.read_csv("datasets/Occupancy/datatest.csv")
second = pd.read_csv("datasets/Occupancy/datatest2.csv")
third = pd.read_csv("datasets/Occupancy/datatraining.csv")

# clean data
# concatenate all three data csv
df = pd.concat([first, second, third])
# reset index for dataset
df.reset_index(drop=True, inplace=True)
# drop date data (unscalable)
df.drop(columns=['date'], inplace=True)
df.dropna()

occupancyDF = df
occupancyDF
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
0	23.7000	26.2720	585.200000	749.200000	0.004764	1
1	23.7180	26.2900	578.400000	760.400000	0.004773	1
2	23.7300	26.2300	572.666667	769.666667	0.004765	1
3	23.7225	26.1250	493.750000	774.750000	0.004744	1
4	23.7540	26.2000	488.600000	779.000000	0.004767	1
...	...	...	...	...	...	...
20555	21.0500	36.0975	433.000000	787.250000	0.005579	1
20556	21.0500	35.9950	433.000000	789.500000	0.005563	1
20557	21.1000	36.0950	433.000000	798.500000	0.005596	1
20558	21.1000	36.2600	433.000000	820.333333	0.005621	1
20559	21.1000	36.2000	447.000000	821.000000	0.005612	1

20560 rows × 6 columns

Load & Clean HTRU2 Dataset

```
# load 'Electrical Grid Stability' data and names into pandas dataframe

# load data by using read_csv from .data file
df = pd.read_csv("datasets/HTRU2/HTRU_2.csv")

# clean data
# drop all samples with NaN entries
df = df.dropna()
```

```
htru2DF = df
htru2DF
```

	mean_int	stddev_int	excess_int	skew_int	mean_dmsnr	stddev_dmsnr	excess_dmsnr	skew_dmsnr	class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0
...	...	...	...	...	...	...	...	...	...
17893	136.429688	59.847421	-0.187846	-0.738123	1.296823	12.166062	15.450260	285.931022	0
17894	122.554688	49.485605	0.127978	0.323061	16.409699	44.626893	2.945244	8.297092	0
17895	119.335938	59.935939	0.159363	-0.743025	21.430602	58.872000	2.499517	4.595173	0
17896	114.507812	53.902400	0.201161	-0.024789	1.946488	13.381731	10.007967	134.238910	0
17897	57.062500	85.797340	1.406391	0.089520	188.306020	64.712562	-1.597527	1.429475	0

17898 rows × 9 columns

Declare & Initialize Algorithm Parameters and Parameter-Grid

```
# pre-declared values/arrays/functions to be used once inside the trial loop
# C values for logistic regression regularization in range of 10(-8) to 10(4)
Cvals = [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]
# K values for k-nearest neighbors in range of 1 to 105 in steps of 4
Kvals = np.linspace(1, 105, num=26, dtype=int).tolist()
# max feature values for random forest similar to CNM06
max_features = [1, 2, 4, 6, 8, 12, 16, 20]
# max depth values for decision trees (shallower = better)
max_depths = np.linspace(1, 5, num=5, dtype=int).tolist()
# array of performance metrics
scoring = ['accuracy', 'f1_micro', 'roc_auc_ovr']

# build parameter grids to be passed into GridSearchCV
logreg_pgrid = {'classifier__penalty': ['l1','l2','none'], 'classifier__C': Cvals, 'classifier__max_iter': [5
knn_pgrid = {'classifier__weights': ['distance'], 'classifier__n_neighbors': Kvals}
rforest_pgrid = {'classifier__n_estimators': [1024], 'classifier__max_features': max_features}
dtree_pgrid = {'classifier__max_depth': max_depths}
```

Create Data Structure to Store ALL Score Data

```
# most top level dictionary to hold each dataset
## size of 4 (4 datasets) accessed by dataset name key value
top_dict = {}

# second top level array to hold the trials from each dataset
## size of 5 (5 trials) accessed by trial number index
### each index in array holds dictionary
#### dictionary holds each trial's data
##### each trial's data hold algorithms as key value
###### accessing algorithm's key value results in another set of dictionaries
####### those dictionaries hold all training and testing data scores resulted from fitting the
####### model with best parameters for a specific scoring metric
score_array = [{}, {}, {}, {}, {}]
```

```
for df, dataset in zip([adultDF, gridDF, occupancyDF, htru2DF],
                       ['Adult', 'Grid', 'Occupancy', 'HTRU2']):
    # loop through this entire trial FIVE (5) times
    for i in range(5):
        # slice the dataframe to not include the binary classifier (label)
        # last column is the label
        X, y = df.iloc[:,:-1], df.iloc[:,-1]

        # randomly pick 5000 samples with replacement for training set
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=5000, shuffle=True)

        # make pipeline for each algorithms to condense model call
        logreg = pipeline.Pipeline([('scale', StandardScaler()), ('classifier', LogisticRegression(n_jobs=-1)
        knn = pipeline.Pipeline([('scale', StandardScaler()), ('classifier', KNeighborsClassifier(n_jobs=-1)
        rforest = pipeline.Pipeline([('scale', StandardScaler()), ('classifier', RandomForestClassifier(n_jo
        dtree = pipeline.Pipeline([('scale', StandardScaler()), ('classifier', DecisionTreeClassifier())])

        # 5-fold cross validation using Stratified KFold
        k_fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=i)

        # GridSearchCV classifier for each algorithm
        logreg_clf = GridSearchCV(estimator=logreg, param_grid=logreg_pgrid, scoring=scoring,
                                  n_jobs=-1, cv=k_fold, verbose=2, refit=False)
        knn_clf = GridSearchCV(estimator=knn, param_grid=knn_pgrid, scoring=scoring,
                                  n_jobs=-1, cv=k_fold, verbose=2, refit=False)
        rforest_clf = GridSearchCV(estimator=rforest, param_grid=rforest_pgrid, scoring=scoring,
                                  n_jobs=-1, cv=k_fold, verbose=2, refit=False)
        dtree_clf = GridSearchCV(estimator=dtree, param_grid=dtree_pgrid, scoring=scoring,
                                  n_jobs=-1, cv=k_fold, verbose=2, refit=False)



        # for each classifier
        for clf, clf_name in zip([logreg_clf, knn_clf, rforest_clf, dtree_clf],
                    ['LogReg', 'KNN', 'Ran_For', 'Dec_Tree']):
            # fit to training data of 5000 samples
            clf.fit(X_train, y_train)

            # get parameters for each scoring metric's best
            best_acc_param = clf.cv_results_['params'][ np.argmin(clf.cv_results_['rank_test_accuracy']) ]
            best_f1_param = clf.cv_results_['params'][ np.argmin(clf.cv_results_['rank_test_f1_micro']) ]
            best_roc_param = clf.cv_results_['params'][ np.argmin(clf.cv_results_['rank_test_roc_auc_ovr']) ]

            # get pipeline based on current classifier
            if (clf_name == 'LogReg'):
                pipe = logreg
            elif (clf_name == 'KNN'):
                pipe = knn
            elif (clf_name == 'Ran_For'):
                pipe = rforest
            elif (clf_name == 'Dec_Tree'):
                pipe = dtree

            # set pipeline parameters to the parameters for best accuracy
            pipe.set_params(**best_acc_param)
            # fit classifier with training data and new parameters for scoring metric
            pipe.fit(X_train, y_train)
            # get predictions for both training and testing data
            y_train_pred = pipe.predict(X_train)
            y_test_pred = pipe.predict(X_test)

            # get scores for all metrics from both training and testing data
            acc_train = accuracy_score(y_train, y_train_pred)
            f1_train = f1_score(y_train, y_train_pred)
            roc_auc_train = roc_auc_score(y_train, y_train_pred)
```

```
            acc_test = accuracy_score(y_test, y_test_pred)
            f1_test = f1_score(y_test, y_test_pred)
            roc_auc_test = roc_auc_score(y_test, y_test_pred)

            # store all scores into a dictionary for accuracy metric
            acc_dict = {'acc_train': acc_train, 'f1_train': f1_train, 'roc_auc_train': roc_auc_train,
                        'acc_test': acc_test, 'f1_test': f1_test, 'roc_auc_test': roc_auc_test}


            # do ^^^^^ all that for f1 score
            pipe.set_params(**best_f1_param)
            pipe.fit(X_train, y_train)
            y_train_pred = pipe.predict(X_train)
            y_test_pred = pipe.predict(X_test)

            acc_train = accuracy_score(y_train, y_train_pred)
            f1_train = f1_score(y_train, y_train_pred)
            roc_auc_train = roc_auc_score(y_train, y_train_pred)

            acc_test = accuracy_score(y_test, y_test_pred)
            f1_test = f1_score(y_test, y_test_pred)
            roc_auc_test = roc_auc_score(y_test, y_test_pred)

            f1_dict = {'acc_train': acc_train, 'f1_train': f1_train, 'roc_auc_train': roc_auc_train,
                        'acc_test': acc_test, 'f1_test': f1_test, 'roc_auc_test': roc_auc_test}


            # do ^^^^^ all that for roc_auc score
            pipe.set_params(**best_roc_param)
            pipe.fit(X_train, y_train)
            y_train_pred = pipe.predict(X_train)
            y_test_pred = pipe.predict(X_test)

            acc_train = accuracy_score(y_train, y_train_pred)
            f1_train = f1_score(y_train, y_train_pred)
            roc_auc_train = roc_auc_score(y_train, y_train_pred)

            acc_test = accuracy_score(y_test, y_test_pred)
            f1_test = f1_score(y_test, y_test_pred)
            roc_auc_test = roc_auc_score(y_test, y_test_pred)

            roc_auc_dict = {'acc_train': acc_train, 'f1_train': f1_train, 'roc_auc_train': roc_auc_train,
                        'acc_test': acc_test, 'f1_test': f1_test, 'roc_auc_test': roc_auc_test}

            # build final dictionary to store all scores from all three models and their best parameters
            score_array[i][clf_name] = {'acc_dict': acc_dict, 'f1_dict': f1_dict, 'roc_auc_dict': roc_auc_di
    top_dict[dataset] = score_array
```

```
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    6.0s finished
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.4s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   30.0s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   27.2s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   44.2s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.2s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```

```
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    3.8s remaining:    0.2s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    4.1s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.9s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   30.2s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   27.9s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   44.0s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.2s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    3.9s remaining:    0.2s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    4.2s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    6.1s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   30.1s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   27.5s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   44.4s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.2s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    3.7s remaining:    0.2s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    4.0s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.7s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   30.0s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   43.7s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.2s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
```

```
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    3.8s remaining:    0.2s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    4.2s finished
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.9s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   30.2s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   27.7s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   43.8s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.2s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.9s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   10.2s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   22.5s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   25.3s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.9s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.9s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   10.2s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   22.3s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   25.1s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.9s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.9s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   10.2s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   22.2s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   25.2s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

```
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.9s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   10.3s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   22.9s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   25.3s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits[Parallel(n_jobs=-1)]: Using backend LokyBackend
with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    1.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:   10.2s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   22.1s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   25.4s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.8s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    1.8s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   10.8s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   12.0s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.7s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
```

```
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    1.9s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   10.6s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   11.7s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.8s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    1.9s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   10.7s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   11.9s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.7s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    1.9s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   10.8s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   12.0s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.8s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
```

```
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    1.8s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   10.8s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   11.9s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.9s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    2.6s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   34.2s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   38.0s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Fitting 5 folds for each of 39 candidates, totalling 195 fits[Parallel(n_jobs=-1)]: Using backend LokyBackend
with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.9s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    2.3s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   38.3s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   41.9s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.8s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.5s
```

```
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    2.3s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   35.9s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   38.8s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.8s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    2.3s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   36.8s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   39.8s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
Fitting 5 folds for each of 39 candidates, totalling 195 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 180 out of 195 | elapsed:    0.8s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done 195 out of 195 | elapsed:    0.8s finished
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
C:\Users\howar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1320: UserWarning: Setting penal
ty='none' will ignore the C and l1_ratio parameters
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:    2.5s finished
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   35.8s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:   38.9s finished
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  23 out of  25 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
```

```
print(top_dict)
```

```
{'Adult': [{'LogReg': {'acc_dict': {'acc_train': 0.9776, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8
997129791788285, 'acc_test': 0.9796867731431229, 'f1_test': 0.884377758164166, 'roc_auc_test': 0.912640934893
729}, 'f1_dict': {'acc_train': 0.9776, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8997129791788285,
'acc_test': 0.9796867731431229, 'f1_test': 0.884377758164166, 'roc_auc_test': 0.912640934893729}, 'roc_auc_di
ct': {'acc_train': 0.9774, 'f1_train': 0.8592777085927772, 'roc_auc_train': 0.8964596711422457, 'acc_test':
0.9784462707396495, 'f1_test': 0.876114081996435, 'roc_auc_test': 0.9048984847818058}}, 'KNN': {'acc_dict':
{'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.871402
8776978417, 'roc_auc_test': 0.8993555895289623}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_trai
n': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.8714028776978417, 'roc_auc_test': 0.8993555895289623},
'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9772832997363933, 'f1
_test': 0.8663930688554491, 'roc_auc_test': 0.8919980508874011}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0,
'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8754880694143167, 'roc_au
c_test': 0.9141721122267628}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_tes
t': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'roc_auc_test': 0.9148429700807956}, 'roc_auc_dict':
{'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9780586137385641, 'f1_test': 0.877329
8656263545, 'roc_auc_test': 0.9154576307892892}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.978, 'f1_train':
0.86318407960199, 'roc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939370445, 'f1_test': 0.870420017
8731009, 'roc_auc_test': 0.901041929094226}, 'f1_dict': {'acc_train': 0.978, 'f1_train': 0.86318407960199, 'r
```

oc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939370445, 'f1_test': 0.8704200178731009, 'roc_auc_te
st': 0.901041929094226}, 'roc_auc_dict': {'acc_train': 0.9822, 'f1_train': 0.8926417370325693, 'roc_auc_trai
n': 0.9252853992346112, 'acc_test': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'roc_auc_test': 0.9148
429700807956}}}, {'LogReg': {'acc_dict': {'acc_train': 0.982, 'f1_train': 0.8984198645598194, 'roc_auc_trai
n': 0.9198433850392286, 'acc_test': 0.977981082338347, 'f1_test': 0.8703196347031963, 'roc_auc_test': 0.90532
83630179727}, 'f1_dict': {'acc_train': 0.982, 'f1_train': 0.8984198645598194, 'roc_auc_train': 0.919843385039
2286, 'acc_test': 0.977981082338347, 'f1_test': 0.8703196347031963, 'roc_auc_test': 0.9053283630179727}, 'roc
_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8878822197055493, 'roc_auc_train': 0.9131561807510331, 'acc_t
est': 0.9774383625368274, 'f1_test': 0.8658367911479944, 'roc_auc_test': 0.8996284316004942}}, 'KNN': {'acc_d
ict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.
8626023657870792, 'roc_auc_test': 0.9026320304845619}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_au
c_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8626023657870792, 'roc_auc_test': 0.9026320304845
619}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.974879826329663
5, 'f1_test': 0.8439306358381502, 'roc_auc_test': 0.8739148070533131}}, 'Ran_For': {'acc_dict': {'acc_train':
1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9791440533416034, 'f1_test': 0.8806036395916556, 'r
oc_auc_test': 0.9210149438896509}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_
test': 0.9792215847418204, 'f1_test': 0.8808888888888889, 'roc_auc_test': 0.9206717390997874}, 'roc_auc_dic
t': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9791440533416034, 'f1_test': 0.88
06036395916556, 'roc_auc_test': 0.9210149438896509}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.987, 'f1_trai
n': 0.9270482603815937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.9763529229337882, 'f1_test': 0.862
4267027514658, 'roc_auc_test': 0.9055907793652292}, 'f1_dict': {'acc_train': 0.987, 'f1_train': 0.92704826038
15937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.9768181113350907, 'f1_test': 0.8645219755323968, 'r
oc_auc_test': 0.9050748582349099}, 'roc_auc_dict': {'acc_train': 0.9848, 'f1_train': 0.9157427937915743, 'roc
_auc_train': 0.9356227915194346, 'acc_test': 0.9768956427353078, 'f1_test': 0.8669642857142856, 'roc_auc_tes
t': 0.9116765376849232}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9792, 'f1_train': 0.8820861678004535, 'roc
_auc_train': 0.9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test': 0.8787878787878787, 'roc_auc_tes
t': 0.9068000752090111}, 'f1_dict': {'acc_train': 0.9792, 'f1_train': 0.8820861678004535, 'roc_auc_train': 0.
9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test': 0.8787878787878787, 'roc_auc_test': 0.9068000752
090111}, 'roc_auc_dict': {'acc_train': 0.9762, 'f1_train': 0.8627450980392157, 'roc_auc_train': 0.89607226032
08693, 'acc_test': 0.9785238021398667, 'f1_test': 0.871699861046781, 'roc_auc_test': 0.9000903674164247}}, 'K
NN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009,
'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8994061986217511}, 'f1_dict': {'acc_train': 1.0, 'f1_train':
1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009, 'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8
994061986217511}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.975
8102031322685, 'f1_test': 0.8511450381679387, 'roc_auc_test': 0.8797470586863493}}, 'Ran_For': {'acc_dict':
{'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9789889905411692, 'f1_test': 0.875516
7661920074, 'roc_auc_test': 0.904962774454589}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_trai
n': 1.0, 'acc_test': 0.9787563963405179, 'f1_test': 0.8741965105601469, 'roc_auc_test': 0.9044501579972247},
'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9794541789424717, 'f1
_test': 0.8800362154821186, 'roc_auc_test': 0.9125282033791584}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.98
18, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9786013335400837, 'f1_
test': 0.873972602739726, 'roc_auc_test': 0.9062884790289292}, 'f1_dict': {'acc_train': 0.9818, 'f1_train':
0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9782912079392154, 'f1_test': 0.8720292
504570383, 'roc_auc_test': 0.9049637947318712}, 'roc_auc_dict': {'acc_train': 0.9818, 'f1_train': 0.897406989
8534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9789114591409521, 'f1_test': 0.8761384335154827,
'roc_auc_test': 0.9083825981506746}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487
889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'r
oc_auc_test': 0.9098645544982119}, 'f1_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_
train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9
098645544982119}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.91
4858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982
119}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881
376958, 'f1_test': 0.8707789284106257, 'roc_auc_test': 0.9068933676714869}, 'f1_dict': {'acc_train': 1.0, 'f1
_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8707789284106257, 'roc_auc_t
est': 0.9068933676714869}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_tes
t': 0.9743371065281439, 'f1_test': 0.8430535798956852, 'roc_auc_test': 0.8752651229841335}}, 'Ran_For': {'acc
_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799968987439913, 'f1_test':
0.8851291184327694, 'roc_auc_test': 0.9184290808638039}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_
auc_train': 1.0, 'acc_test': 0.9798418359435571, 'f1_test': 0.8841354723707666, 'roc_auc_test': 0.91758090444
31084}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9795317103426
888, 'f1_test': 0.8823529411764706, 'roc_auc_test': 0.916647396509485}}, 'Dec_Tree': {'acc_dict': {'acc_trai
n': 0.9798, 'f1_train': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.97705070553574
2, 'f1_test': 0.8655767484105358, 'roc_auc_test': 0.9011694615089376}, 'f1_dict': {'acc_train': 0.9798, 'f1_t
rain': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.977050705535742, 'f1_test': 0.8
655767484105358, 'roc_auc_test': 0.9011694615089376}, 'roc_auc_dict': {'acc_train': 0.9834, 'f1_train': 0.907
6751946607342, 'roc_auc_train': 0.9400641639532655, 'acc_test': 0.9784462707396495, 'f1_test': 0.878177037686
2401, 'roc_auc_test': 0.9206271453655979}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767
123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536
485, 'roc_auc_test': 0.9025865080719201}, 'f1_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'r
oc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_te

st': 0.9025865080719201}, 'roc_auc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8566001899335232, 'roc_auc_test': 0.8823090659229057}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9801519615444255, 'f1_test': 0.8848920863309352, 'roc_auc_test': 0.9156374373670301}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9802294929446426, 'f1_test': 0.8854961832061069, 'roc_auc_test': 0.916445122666026}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799193673437743, 'f1_test': 0.8834907782276203, 'roc_auc_test': 0.9147444423939172}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772832997363933, 'f1_test': 0.8692547969656403, 'roc_auc_test': 0.9102340560861778}, 'f1_dict': {'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772057683361761, 'f1_test': 0.8685152057245079, 'roc_auc_test': 0.9090438555562133}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8908489525909592, 'roc_auc_train': 0.9318768979416184, 'acc_test': 0.977981082338347, 'f1_test': 0.8737777777777778, 'roc_auc_test': 0.9140605866984228}}}], 'Grid': [{'LogReg': {'acc_dict': {'acc_train': 0.9776, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8997129791788285, 'acc_test': 0.9796867731431229, 'f1_test': 0.884377758164166, 'roc_auc_test': 0.912640934893729}, 'f1_dict': {'acc_train': 0.9776, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8997129791788285, 'acc_test': 0.9796867731431229, 'f1_test': 0.884377758164166, 'roc_auc_test': 0.912640934893729}, 'roc_auc_dict': {'acc_train': 0.9774, 'f1_train': 0.8592777082777082, 'roc_auc_train': 0.8964596711422457, 'acc_test': 0.9784462707396495, 'f1_test': 0.876114081996435, 'roc_auc_test': 0.9048984847818058}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.8714028776978417, 'roc_auc_test': 0.8993555895289623}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.8714028776978417, 'roc_auc_test': 0.8993555895289623}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9772832997363933, 'f1_test': 0.8663930688554491, 'roc_auc_test': 0.8919980508874011}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484884881376958, 'f1_test': 0.8754880694143167, 'roc_auc_test': 0.9141721122267628}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'roc_auc_test': 0.9148429700807956}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9780586137385641, 'f1_test': 0.8773298656263545, 'roc_auc_test': 0.9154576307892892}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.978, 'f1_train': 0.86318407960199, 'roc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939370445, 'f1_test': 0.8704200178731009, 'roc_auc_test': 0.901041929094226}, 'f1_dict': {'acc_train': 0.978, 'f1_train': 0.86318407960199, 'roc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939370445, 'f1_test': 0.8704200178731009, 'roc_auc_test': 0.901041929094226}, 'roc_auc_dict': {'acc_train': 0.9822, 'f1_train': 0.8926417370325693, 'roc_auc_train': 0.9252853992346112, 'acc_test': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'roc_auc_test': 0.9148429700807956}}}, {'LogReg': {'acc_dict': {'acc_train': 0.982, 'f1_train': 0.8984198645598194, 'roc_auc_train': 0.9198433850392286, 'acc_test': 0.977981082338347, 'f1_test': 0.8703196347031963, 'roc_auc_test': 0.9053283630179727}, 'f1_dict': {'acc_train': 0.982, 'f1_train': 0.8984198645598194, 'roc_auc_train': 0.9198433850392286, 'acc_test': 0.977981082338347, 'f1_test': 0.8703196347031963, 'roc_auc_test': 0.9053283630179727}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8878822197055493, 'roc_auc_train': 0.9131561807510331, 'acc_test': 0.9774383625368274, 'f1_test': 0.8658367911479944, 'roc_auc_test': 0.8996284316004942}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8626023657870792, 'roc_auc_test': 0.9026320304845619}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8626023657870792, 'roc_auc_test': 0.9026320304845619}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9748798263296635, 'f1_test': 0.8439306358381502, 'roc_auc_test': 0.8739148070533131}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9791440533416034, 'f1_test': 0.8806036395916556, 'roc_auc_test': 0.9210149438896509}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9792215847418204, 'f1_test': 0.8808888888888889, 'roc_auc_test': 0.9206717390997874}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9791440533416034, 'f1_test': 0.8806036395916556, 'roc_auc_test': 0.9210149438896509}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.987, 'f1_train': 0.9270482603815937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.9763529229337882, 'f1_test': 0.8624267027514658, 'roc_auc_test': 0.9055907793652292}, 'f1_dict': {'acc_train': 0.987, 'f1_train': 0.9270482603815937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.9768181113350907, 'f1_test': 0.8645219755323968, 'roc_auc_test': 0.9050748582349099}, 'roc_auc_dict': {'acc_train': 0.9848, 'f1_train': 0.9157427937915743, 'roc_auc_train': 0.9356227915194346, 'acc_test': 0.9768956427353078, 'f1_test': 0.8669642857142856, 'roc_auc_test': 0.9116765376849232}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9792, 'f1_train': 0.8820861678004535, 'roc_auc_train': 0.9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test': 0.8787878787878787, 'roc_auc_test': 0.9068000752090111}, 'f1_dict': {'acc_train': 0.9792, 'f1_train': 0.8820861678004535, 'roc_auc_train': 0.9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test': 0.8787878787878787, 'roc_auc_test': 0.9068000752090111}, 'roc_auc_dict': {'acc_train': 0.9762, 'f1_train': 0.8627450980392157, 'roc_auc_train': 0.8960722603208693, 'acc_test': 0.9785238021398667, 'f1_test': 0.8716998610467 81, 'roc_auc_test': 0.9000903674164247}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009, 'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8994061986217511}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009, 'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8994061986217511}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9758102031322685, 'f1_test': 0.8511450381679387, 'roc_auc_test':

0.8797470586863493}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9789889905411692, 'f1_test': 0.8755167661920074, 'roc_auc_test': 0.904962774454589}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9787563963405179, 'f1_test': 0.8741965105601469, 'roc_auc_test': 0.9044501579972247}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9794541789424717, 'f1_test': 0.8800362154821186, 'roc_auc_test': 0.9125282033791584}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.9818, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9786013335400837, 'f1_test': 0.873972602739726, 'roc_auc_test': 0.9062884790289292}, 'f1_dict': {'acc_train': 0.9818, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9782912079392154, 'f1_test': 0.8720292504570383, 'roc_auc_test': 0.9049637947318712}, 'roc_auc_dict': {'acc_train': 0.9818, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9789114591409521, 'f1_test': 0.8761384335154827, 'roc_auc_test': 0.9083825981506746}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982119}, 'f1_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982119}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982119}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8707789284106257, 'roc_auc_test': 0.9068933676714869}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8707789284106257, 'roc_auc_test': 0.9068933676714869}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9743371065281439, 'f1_test': 0.8430535798956852, 'roc_auc_test': 0.8752651229841335}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799968987439913, 'f1_test': 0.8851291184327694, 'roc_auc_test': 0.9184290808638039}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9798418359435571, 'f1_test': 0.8841354723707666, 'roc_auc_test': 0.9175809044431084}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9795317103426888, 'f1_test': 0.8823529411764706, 'roc_auc_test': 0.916647396509485}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.9798, 'f1_train': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.977050705535742, 'f1_test': 0.8655767484105358, 'roc_auc_test': 0.9011694615089376}, 'f1_dict': {'acc_train': 0.9798, 'f1_train': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.977050705535742, 'f1_test': 0.8655767484105358, 'roc_auc_test': 0.9011694615089376}, 'roc_auc_dict': {'acc_train': 0.9834, 'f1_train': 0.9076751946607342, 'roc_auc_train': 0.9400641639532655, 'acc_test': 0.9784462707396495, 'f1_test': 0.8781770376862401, 'roc_auc_test': 0.9206271453655979}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}, 'f1_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}, 'roc_auc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8566001899335232, 'roc_auc_test': 0.8823090659229057}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9801519615444255, 'f1_test': 0.8848920863309352, 'roc_auc_test': 0.9156374373670301}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9802294929446426, 'f1_test': 0.8854961832061069, 'roc_auc_test': 0.916445122666026}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799193673437743, 'f1_test': 0.8834907782276203, 'roc_auc_test': 0.9147444423939172}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772832997363933, 'f1_test': 0.8692547969656403, 'roc_auc_test': 0.9102340560861778}, 'f1_dict': {'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772057683361761, 'f1_test': 0.8685152057245079, 'roc_auc_test': 0.9090438555562133}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8908489525909592, 'roc_auc_train': 0.9318768979416184, 'acc_test': 0.977981082338347, 'f1_test': 0.8737777777777778, 'roc_auc_test': 0.9140605866984228}}}], 'Occupancy': [{'LogReg': {'acc_dict': {'acc_train': 0.9776, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8997129791788285, 'acc_test': 0.9796867731431229, 'f1_test': 0.884377758164166, 'roc_auc_test': 0.912640934893729}, 'f1_dict': {'acc_train': 0.9776, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8997129791788285, 'acc_test': 0.9796867731431229, 'f1_test': 0.884377758164166, 'roc_auc_test': 0.912640934893729}, 'roc_auc_dict': {'acc_train': 0.9774, 'f1_train': 0.8592777085927772, 'roc_auc_train': 0.8964596711422457, 'acc_test': 0.9784462707396495, 'f1_test': 0.876114081996435, 'roc_auc_test': 0.9048984847818058}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.8714028776978417, 'roc_auc_test': 0.8993555895289623}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.8714028776978417, 'roc_auc_test': 0.8993555895289623}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9772832997363933, 'f1_test': 0.8663930688554491, 'roc_auc_test': 0.8919980508874011}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8754880694143167, 'roc_auc_test': 0.9141721122267628}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'roc_auc_test': 0.9148429700807956}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9780586137385641, 'f1_test': 0.8773298656263545, 'roc_auc_test': 0.9154576307892892}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.978, 'f1_train': 0.86318407960199, 'roc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939370445, 'f1_test': 0.870420017

8731009, 'roc_auc_test': 0.901041929094226}, 'f1_dict': {'acc_train': 0.978, 'f1_train': 0.86318407960199, 'r
oc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939370445, 'f1_test': 0.8704200178731009, 'roc_auc_te
st': 0.901041929094226}, 'roc_auc_dict': {'acc_train': 0.9822, 'f1_train': 0.8926417370325693, 'roc_auc_trai
n': 0.9252853992346112, 'acc_test': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'roc_auc_test': 0.9148
429700807956}}}, {'LogReg': {'acc_dict': {'acc_train': 0.982, 'f1_train': 0.8984198645598194, 'roc_auc_trai
n': 0.9198433850392286, 'acc_test': 0.977981082338347, 'f1_test': 0.8703196347031963, 'roc_auc_test': 0.90532
83630179727}, 'f1_dict': {'acc_train': 0.982, 'f1_train': 0.8984198645598194, 'roc_auc_train': 0.919843385039
2286, 'acc_test': 0.977981082338347, 'f1_test': 0.8703196347031963, 'roc_auc_test': 0.9053283630179727}, 'roc
_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8878822197055493, 'roc_auc_train': 0.9131561807510331, 'acc_t
est': 0.9774383625368274, 'f1_test': 0.8658367911479944, 'roc_auc_test': 0.8996284316004942}}, 'KNN': {'acc_d
ict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.
8626023657870792, 'roc_auc_test': 0.9026320304845619}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_au
c_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8626023657870792, 'roc_auc_test': 0.9026320304845
619}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.974879826329663
5, 'f1_test': 0.8439306358381502, 'roc_auc_test': 0.8739148070533131}}, 'Ran_For': {'acc_dict': {'acc_train':
1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9791440533416034, 'f1_test': 0.8806036395916556, 'r
oc_auc_test': 0.9210149438896509}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_
test': 0.9792215847418204, 'f1_test': 0.8808888888888889, 'roc_auc_test': 0.9206717390997874}, 'roc_auc_dic
t': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9791440533416034, 'f1_test': 0.88
06036395916556, 'roc_auc_test': 0.9210149438896509}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.987, 'f1_trai
n': 0.9270482603815937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.9763529229337882, 'f1_test': 0.862
4267027514658, 'roc_auc_test': 0.9055907793652292}, 'f1_dict': {'acc_train': 0.987, 'f1_train': 0.92704826038
15937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.9768181113350907, 'f1_test': 0.8645219755323968, 'r
oc_auc_test': 0.9050748582349099}, 'roc_auc_dict': {'acc_train': 0.9848, 'f1_train': 0.9157427937915743, 'roc
_auc_train': 0.9356227915194346, 'acc_test': 0.9768956427353078, 'f1_test': 0.8669642857142856, 'roc_auc_tes
t': 0.9116765376849232}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9792, 'f1_train': 0.8820861678004535, 'roc
_auc_train': 0.9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test': 0.8787878787878787, 'roc_auc_tes
t': 0.9068000752090111}, 'f1_dict': {'acc_train': 0.9792, 'f1_train': 0.8820861678004535, 'roc_auc_train': 0.
9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test': 0.8787878787878787, 'roc_auc_test': 0.9068000752
090111}, 'roc_auc_dict': {'acc_train': 0.9762, 'f1_train': 0.8627450980392157, 'roc_auc_train': 0.89607226032
08693, 'acc_test': 0.9785238021398667, 'f1_test': 0.871699861046781, 'roc_auc_test': 0.9000903674164247}}, 'K
NN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009,
'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8994061986217511}, 'f1_dict': {'acc_train': 1.0, 'f1_train':
1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009, 'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8
994061986217511}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.975
8102031322685, 'f1_test': 0.8511450381679387, 'roc_auc_test': 0.8797470586863493}}, 'Ran_For': {'acc_dict':
{'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9789889905411692, 'f1_test': 0.875516
7661920074, 'roc_auc_test': 0.9049627744545 89}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_trai
n': 1.0, 'acc_test': 0.9787563963405179, 'f1_test': 0.8741965105601469, 'roc_auc_test': 0.9044501579972247},
'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9794541789424717, 'f1
_test': 0.8800362154821186, 'roc_auc_test': 0.9125282033791584}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.98
18, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9786013335400837, 'f1_
test': 0.873972602739726, 'roc_auc_test': 0.9062884790289292}, 'f1_dict': {'acc_train': 0.9818, 'f1_train':
0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9782912079392154, 'f1_test': 0.8720292
504570383, 'roc_auc_test': 0.9049637947318712}, 'roc_auc_dict': {'acc_train': 0.9818, 'f1_train': 0.897406989
8534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9789114591409521, 'f1_test': 0.8761384335154827,
'roc_auc_test': 0.9083825981506746}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487
889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'r
oc_auc_test': 0.9098645544982119}, 'f1_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_
train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9
098645544982119}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.91
4858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982
119}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881
376958, 'f1_test': 0.8707789284106257, 'roc_auc_test': 0.9068933676714869}, 'f1_dict': {'acc_train': 1.0, 'f1
_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8707789284106257, 'roc_auc_t
est': 0.9068933676714869}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_tes
t': 0.9743371065281439, 'f1_test': 0.8430535798956852, 'roc_auc_test': 0.8752651229841335}}, 'Ran_For': {'acc
_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799968987439913, 'f1_test':
0.8851291184327694, 'roc_auc_test': 0.9184290808638039}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_
auc_train': 1.0, 'acc_test': 0.9798418359435571, 'f1_test': 0.8841354723707666, 'roc_auc_test': 0.91758090444
31084}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9795317103426
888, 'f1_test': 0.8823529411764706, 'roc_auc_test': 0.916647396509485}}, 'Dec_Tree': {'acc_dict': {'acc_trai
n': 0.9798, 'f1_train': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.97705070553574
2, 'f1_test': 0.8655767484105358, 'roc_auc_test': 0.9011694615089376}, 'f1_dict': {'acc_train': 0.9798, 'f1_t
rain': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.977050705535742, 'f1_test': 0.8
655767484105358, 'roc_auc_test': 0.9011694615089376}, 'roc_auc_dict': {'acc_train': 0.9834, 'f1_train': 0.907
6751946607342, 'roc_auc_train': 0.9400641639532655, 'acc_test': 0.9784462707396495, 'f1_test': 0.878177037686
2401, 'roc_auc_test': 0.9206271453655979}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767
123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536
485, 'roc_auc_test': 0.9025865080719201}, 'f1_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'r

oc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_te
st': 0.9025865080719201}, 'roc_auc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_trai
n': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025
865080719201}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9
781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'f1_dict': {'acc_train':
1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'r
oc_auc_test': 0.9099382288318842}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0,
'acc_test': 0.9765855171344394, 'f1_test': 0.8566001899335232, 'roc_auc_test': 0.8823090659229057}}, 'Ran_Fo
r': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9801519615444255, 'f
1_test': 0.8848920863309352, 'roc_auc_test': 0.9156374373670301}, 'f1_dict': {'acc_train': 1.0, 'f1_train':
1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9802294929446426, 'f1_test': 0.8854961832061069, 'roc_auc_test': 0.9
16445122666026}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799
193673437743, 'f1_test': 0.8834907782276203, 'roc_auc_test': 0.9147444423939172}}, 'Dec_Tree': {'acc_dict':
{'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.97728
32997363933, 'f1_test': 0.8692547969656403, 'roc_auc_test': 0.9102340560861778}, 'f1_dict': {'acc_train': 0.9
85, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772057683361761, 'f1_
test': 0.8685152057245079, 'roc_auc_test': 0.9090438555562133}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_tra
in': 0.8908489525909592, 'roc_auc_train': 0.9318768979416184, 'acc_test': 0.977981082338347, 'f1_test': 0.873
7777777777778, 'roc_auc_test': 0.9140605866984228}}}], 'HTRU2': [{'LogReg': {'acc_dict': {'acc_train': 0.977
6, 'f1_train': 0.8613861386138614, 'roc_auc_train': 0.8997129791788285, 'acc_test': 0.9796867731431229, 'f1_t
est': 0.884377758164166, 'roc_auc_test': 0.912640934893729}, 'f1_dict': {'acc_train': 0.9776, 'f1_train': 0.8
613861386138614, 'roc_auc_train': 0.8997129791788285, 'acc_test': 0.9796867731431229, 'f1_test': 0.8843777581
64166, 'roc_auc_test': 0.912640934893729}, 'roc_auc_dict': {'acc_train': 0.9774, 'f1_train': 0.859277708592777
72, 'roc_auc_train': 0.8964596711422457, 'acc_test': 0.9784462707396495, 'f1_test': 0.876114081996435, 'roc_a
uc_test': 0.9048984847818058}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0,
'acc_test': 0.9778260195379128, 'f1_test': 0.8714028776978417, 'roc_auc_test': 0.8993555895289623}, 'f1_dic
t': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9778260195379128, 'f1_test': 0.87
14028776978417, 'roc_auc_test': 0.8993555895289623}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc
_auc_train': 1.0, 'acc_test': 0.9772832997363933, 'f1_test': 0.8663930688554491, 'roc_auc_test': 0.8919980508
874011}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.977
7484881376958, 'f1_test': 0.8754880694143167, 'roc_auc_test': 0.9141721122267628}, 'f1_dict': {'acc_train':
1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9782912079392154, 'f1_test': 0.8782608695652174, 'r
oc_auc_test': 0.9148429700807956}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0,
'acc_test': 0.9780586137385641, 'f1_test': 0.8773298656263545, 'roc_auc_test': 0.9154576307892892}}, 'Dec_Tre
e': {'acc_dict': {'acc_train': 0.978, 'f1_train': 0.86318407960199, 'roc_auc_train': 0.8988839430498801, 'acc
_test': 0.9775158939370445, 'f1_test': 0.8704200178731009, 'roc_auc_test': 0.901041929094226}, 'f1_dict': {'a
cc_train': 0.978, 'f1_train': 0.86318407960199, 'roc_auc_train': 0.8988839430498801, 'acc_test': 0.9775158939
370445, 'f1_test': 0.8704200178731009, 'roc_auc_test': 0.901041929094226}, 'roc_auc_dict': {'acc_train': 0.98
22, 'f1_train': 0.8926417370325693, 'roc_auc_train': 0.9252853992346112, 'acc_test': 0.9782912079392154, 'f1_
test': 0.8782608695652174, 'roc_auc_test': 0.9148429700807956}}}, {'LogReg': {'acc_dict': {'acc_train': 0.98
2, 'f1_train': 0.8984198645598194, 'roc_auc_train': 0.9198433850392286, 'acc_test': 0.977981082338347, 'f1_te
st': 0.8703196347031963, 'roc_auc_test': 0.9053283630179727}, 'f1_dict': {'acc_train': 0.982, 'f1_train': 0.8
984198645598194, 'roc_auc_train': 0.9198433850392286, 'acc_test': 0.977981082338347, 'f1_test': 0.87031963470
31963, 'roc_auc_test': 0.9053283630179727}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8878822197055
493, 'roc_auc_train': 0.9131561807510331, 'acc_test': 0.9774383625368274, 'f1_test': 0.8658367911479944, 'roc
_auc_test': 0.8996284316004942}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.
0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8626023657870792, 'roc_auc_test': 0.9026320304845619}, 'f1_di
ct': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8
626023657870792, 'roc_auc_test': 0.9026320304845619}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'ro
c_auc_train': 1.0, 'acc_test': 0.9748798263296635, 'f1_test': 0.8439306358381502, 'roc_auc_test': 0.873914807
0533131}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.97
91440533416034, 'f1_test': 0.8806036395916556, 'roc_auc_test': 0.9210149438896509}, 'f1_dict': {'acc_train':
1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9792215847418204, 'f1_test': 0.8808888888888889, 'r
oc_auc_test': 0.9206717390997874}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0,
'acc_test': 0.9791440533416034, 'f1_test': 0.8806036395916556, 'roc_auc_test': 0.9210149438896509}}, 'Dec_Tre
e': {'acc_dict': {'acc_train': 0.987, 'f1_train': 0.9270482603815937, 'roc_auc_train': 0.9368374558303886, 'a
cc_test': 0.9763529229337882, 'f1_test': 0.8624267027514658, 'roc_auc_test': 0.9055907793652292}, 'f1_dict':
{'acc_train': 0.987, 'f1_train': 0.9270482603815937, 'roc_auc_train': 0.9368374558303886, 'acc_test': 0.97681
81113350907, 'f1_test': 0.8645219755323968, 'roc_auc_test': 0.9050748582349099}, 'roc_auc_dict': {'acc_trai
n': 0.9848, 'f1_train': 0.9157427937915743, 'roc_auc_train': 0.9356227915194346, 'acc_test': 0.97689564273530
78, 'f1_test': 0.8669642857142856, 'roc_auc_test': 0.9116765376849232}}}, {'LogReg': {'acc_dict': {'acc_trai
n': 0.9792, 'f1_train': 0.8820861678004535, 'roc_auc_train': 0.9120637315362212, 'acc_test': 0.97953171034268
88, 'f1_test': 0.8787878787878787, 'roc_auc_test': 0.9068000752090111}, 'f1_dict': {'acc_train': 0.9792, 'f1_
train': 0.8820861678004535, 'roc_auc_train': 0.9120637315362212, 'acc_test': 0.9795317103426888, 'f1_test':
0.8787878787878787, 'roc_auc_test': 0.9068000752090111}, 'roc_auc_dict': {'acc_train': 0.9762, 'f1_train': 0.
8627450980392157, 'roc_auc_train': 0.8960722603208693, 'acc_test': 0.9785238021398667, 'f1_test': 0.871699861
046781, 'roc_auc_test': 0.9000903674164247}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_au
c_train': 1.0, 'acc_test': 0.9786788649403009, 'f1_test': 0.8722712494194148, 'roc_auc_test': 0.8994061986217
511}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9786788649403009, 'f
1_test': 0.8722712494194148, 'roc_auc_test': 0.8994061986217511}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_trai

n': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9758102031322685, 'f1_test': 0.8511450381679387, 'roc_auc_test': 0.8797470586863493}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9789889905411692, 'f1_test': 0.8755167661920074, 'roc_auc_test': 0.904962774454589}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9787563963405179, 'f1_test': 0.8741965105601469, 'roc_auc_test': 0.9044501579972247}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9794541789424717, 'f1_test': 0.8800362154821186, 'roc_auc_test': 0.9125282033791584}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.9818, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9786013335400837, 'f1_test': 0.873972602739726, 'roc_auc_test': 0.9062884790289292}, 'f1_dict': {'acc_train': 0.9818, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9782912079392154, 'f1_test': 0.8720292504570383, 'roc_auc_test': 0.9049637947318712}, 'roc_auc_dict': {'acc_train': 0.9818, 'f1_train': 0.8974069898534386, 'roc_auc_train': 0.9221000179290827, 'acc_test': 0.9789114591409521, 'f1_test': 0.8761384335154827, 'roc_auc_test': 0.9083825981506746}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982119}, 'f1_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982119}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8858131487889274, 'roc_auc_train': 0.914858264700249, 'acc_test': 0.9789889905411692, 'f1_test': 0.8777627424447452, 'roc_auc_test': 0.9098645544982119}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8707789284106257, 'roc_auc_test': 0.9068933676714869}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9777484881376958, 'f1_test': 0.8707789284106257, 'roc_auc_test': 0.9068933676714869}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9743371065281439, 'f1_test': 0.8430535798956852, 'roc_auc_test': 0.8752651229841335}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799968987439913, 'f1_test': 0.8851291184327694, 'roc_auc_test': 0.9184290808638039}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9798418359435571, 'f1_test': 0.8841354723707666, 'roc_auc_test': 0.9175809044431084}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9795317103426888, 'f1_test': 0.8823529411764706, 'roc_auc_test': 0.916647396509485}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.9798, 'f1_train': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.977050705535742, 'f1_test': 0.8655767484105358, 'roc_auc_test': 0.9011694615089376}, 'f1_dict': {'acc_train': 0.9798, 'f1_train': 0.8835063437139561, 'roc_auc_train': 0.9136611760199195, 'acc_test': 0.977050705535742, 'f1_test': 0.8655767484105358, 'roc_auc_test': 0.9011694615089376}, 'roc_auc_dict': {'acc_train': 0.9834, 'f1_train': 0.9076751946607342, 'roc_auc_train': 0.9400641639532655, 'acc_test': 0.9784462707396495, 'f1_test': 0.8781770376862401, 'roc_auc_test': 0.9206271453655979}}}, {'LogReg': {'acc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}, 'f1_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}, 'roc_auc_dict': {'acc_train': 0.9784, 'f1_train': 0.8767123287671234, 'roc_auc_train': 0.9114908805973062, 'acc_test': 0.9786788649403009, 'f1_test': 0.8737953189536485, 'roc_auc_test': 0.9025865080719201}}, 'KNN': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9781361451387812, 'f1_test': 0.8733153638814016, 'roc_auc_test': 0.9099382288318842}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9765855171344394, 'f1_test': 0.8566001899335232, 'roc_auc_test': 0.8823090659229057}}, 'Ran_For': {'acc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9801519615444255, 'f1_test': 0.8848920863309352, 'roc_auc_test': 0.9156374373670301}, 'f1_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9802294929446426, 'f1_test': 0.8854961832061069, 'roc_auc_test': 0.916445122666026}, 'roc_auc_dict': {'acc_train': 1.0, 'f1_train': 1.0, 'roc_auc_train': 1.0, 'acc_test': 0.9799193673437743, 'f1_test': 0.8834907782276203, 'roc_auc_test': 0.9147444423939172}}, 'Dec_Tree': {'acc_dict': {'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772832997363933, 'f1_test': 0.8692547969656403, 'roc_auc_test': 0.9102340560861778}, 'f1_dict': {'acc_train': 0.985, 'f1_train': 0.9160134378499439, 'roc_auc_train': 0.9393703606202136, 'acc_test': 0.9772057683361761, 'f1_test': 0.8685152057245079, 'roc_auc_test': 0.9090438555562133}, 'roc_auc_dict': {'acc_train': 0.9802, 'f1_train': 0.8908489525909592, 'roc_auc_train': 0.9318768979416184, 'acc_test': 0.977981082338347, 'f1_test': 0.8737777777777778, 'roc_auc_test': 0.9140605866984228}}}]}

Analyze Problem Dataset (Table 1)

```
# array of dataset strings
STRINGarray = ['ADULT', 'GRID', 'HTRU2', 'OCCUPANCY']

# get attributes size
adultATTR = adultDF.shape[1]
gridATTR = gridDF.shape[1]
htru2ATTR = htru2DF.shape[1]
occupancyATTR = occupancyDF.shape[1]
# process adult attributes to add attributes from non-encode and encode
adultATTR = '14/' + str(adultATTR - 1)
# store into array
```

```
ATTRarray = [adultATTR, gridATTR - 1, htru2ATTR - 1, occupancyATTR - 1]

# get test size
adultSIZE = adultDF.shape[0]
gridSIZE = gridDF.shape[0]
htru2SIZE = htru2DF.shape[0]
occupancySIZE = occupancyDF.shape[0]
# store into array
SIZEarray = [adultSIZE, gridSIZE, htru2SIZE, occupancySIZE]

# get number of pos from each dataset
adultPOS = adultDF.loc[adultDF['income>50K'] == 1].shape[0]
gridPOS = gridDF.loc[gridDF['stabf'] == 1].shape[0]
htru2POS = htru2DF.loc[htru2DF['class'] == 1].shape[0]
occupancyPOS = occupancyDF.loc[occupancyDF['Occupancy'] == 1].shape[0]
# process POS to get percentage
adultPOS = str(int((adultPOS/adultSIZE) * 100)) + '%'
gridPOS = str(int((gridPOS/gridSIZE) * 100)) + '%'
htru2POS = str(int((htru2POS/htru2SIZE) * 100)) + '%'
occupancyPOS = str(int((occupancyPOS/occupancySIZE) * 100)) + '%'
# store into array
POSarray = [adultPOS, gridPOS, htru2POS, occupancyPOS]

t1 = {'PROBLEM': STRINGarray, '#ATTR': ATTRarray, 'TRAIN SIZE': [5000,5000,5000,5000],
        'TEST SIZE': SIZEarray, '%POS': POSarray}
t1 = pd.DataFrame.from_dict(t1)
t1.set_index('PROBLEM', inplace=True)
t1
```

PROBLEM	#ATTR	TRAIN SIZE	TEST SIZE	%POS
ADULT	14/104	5000	30162	24%
GRID	13	5000	10000	36%
HTRU2	8	5000	17898	9%
OCCUPANCY	5	5000	20560	23%

Break Down and Analyze Scores Data

Break Down Adult's Testing Data Scores

```
# get adult dataset score array
adultDATA = top_dict['Adult']

# make array for ADULT dataset BEST PARAM ACCURACY TEST Scores (for all algorithms)
adult_logreg_acc = []
adult_knn_acc = []
adult_rforest_acc = []
adult_dtree_acc = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    adult_logreg_acc.append(adultDATA[trial]['LogReg']['acc_dict']['acc_test'])
    adult_knn_acc.append(adultDATA[trial]['KNN']['acc_dict']['acc_test'])
    adult_rforest_acc.append(adultDATA[trial]['Ran_For']['acc_dict']['acc_test'])
    adult_dtree_acc.append(adultDATA[trial]['Dec_Tree']['acc_dict']['acc_test'])


# make array for ADULT dataset BEST PARAM F1 TEST Scores (for all algorithms)
adult_logreg_f1 = []
adult_knn_f1 = []
```

```
adult_rforest_f1 = []
adult_dtree_f1 = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    adult_logreg_f1.append(adultDATA[trial]['LogReg']['f1_dict']['f1_test'])
    adult_knn_f1.append(adultDATA[trial]['KNN']['f1_dict']['f1_test'])
    adult_rforest_f1.append(adultDATA[trial]['Ran_For']['f1_dict']['f1_test'])
    adult_dtree_f1.append(adultDATA[trial]['Dec_Tree']['f1_dict']['f1_test'])


# make array for ADULT dataset BEST PARAM ROC AUC TEST Scores (for all algorithms)
adult_logreg_roc_auc = []
adult_knn_roc_auc = []
adult_rforest_roc_auc = []
adult_dtree_roc_auc = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    adult_logreg_roc_auc.append(adultDATA[trial]['LogReg']['roc_auc_dict']['roc_auc_test'])
    adult_knn_roc_auc.append(adultDATA[trial]['KNN']['roc_auc_dict']['roc_auc_test'])
    adult_rforest_roc_auc.append(adultDATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_test'])
    adult_dtree_roc_auc.append(adultDATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_test'])
```

Break Down Grid Testing Data Scores

```
# get GRID dataset score array
gridDATA = top_dict['Grid']

# make array for ADULT dataset BEST PARAM ACCURACY TEST Scores (for all algorithms)
grid_logreg_acc = []
grid_knn_acc = []
grid_rforest_acc = []
grid_dtree_acc = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    grid_logreg_acc.append(gridDATA[trial]['LogReg']['acc_dict']['acc_test'])
    grid_knn_acc.append(gridDATA[trial]['KNN']['acc_dict']['acc_test'])
    grid_rforest_acc.append(gridDATA[trial]['Ran_For']['acc_dict']['acc_test'])
    grid_dtree_acc.append(gridDATA[trial]['Dec_Tree']['acc_dict']['acc_test'])


# make array for GRID dataset BEST PARAM F1 TEST Scores (for all algorithms)
grid_logreg_f1 = []
grid_knn_f1 = []
grid_rforest_f1 = []
grid_dtree_f1 = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    grid_logreg_f1.append(gridDATA[trial]['LogReg']['f1_dict']['f1_test'])
    grid_knn_f1.append(gridDATA[trial]['KNN']['f1_dict']['f1_test'])
    grid_rforest_f1.append(gridDATA[trial]['Ran_For']['f1_dict']['f1_test'])
    grid_dtree_f1.append(gridDATA[trial]['Dec_Tree']['f1_dict']['f1_test'])


# make array for GRID dataset BEST PARAM ROC AUC TEST Scores (for all algorithms)
grid_logreg_roc_auc = []
grid_knn_roc_auc = []
grid_rforest_roc_auc = []
grid_dtree_roc_auc = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    grid_logreg_roc_auc.append(gridDATA[trial]['LogReg']['roc_auc_dict']['roc_auc_test'])
    grid_knn_roc_auc.append(gridDATA[trial]['KNN']['roc_auc_dict']['roc_auc_test'])
    grid_rforest_roc_auc.append(gridDATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_test'])
    grid_dtree_roc_auc.append(gridDATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_test'])
```

Break Down HTRU2 Testing Data Scores

```
# get HTRU2 dataset score array
htru2DATA = top_dict['HTRU2']

# make array for HTRU2 dataset BEST PARAM ACCURACY TEST Scores (for all algorithms)
htru2_logreg_acc = []
htru2_knn_acc = []
htru2_rforest_acc = []
htru2_dtree_acc = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    htru2_logreg_acc.append(htru2DATA[trial]['LogReg']['acc_dict']['acc_test'])
    htru2_knn_acc.append(htru2DATA[trial]['KNN']['acc_dict']['acc_test'])
    htru2_rforest_acc.append(htru2DATA[trial]['Ran_For']['acc_dict']['acc_test'])
    htru2_dtree_acc.append(htru2DATA[trial]['Dec_Tree']['acc_dict']['acc_test'])


# make array for HTRU2 dataset BEST PARAM F1 TEST Scores (for all algorithms)
htru2_logreg_f1 = []
htru2_knn_f1 = []
htru2_rforest_f1 = []
htru2_dtree_f1 = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    htru2_logreg_f1.append(htru2DATA[trial]['LogReg']['f1_dict']['f1_test'])
    htru2_knn_f1.append(htru2DATA[trial]['KNN']['f1_dict']['f1_test'])
    htru2_rforest_f1.append(htru2DATA[trial]['Ran_For']['f1_dict']['f1_test'])
    htru2_dtree_f1.append(htru2DATA[trial]['Dec_Tree']['f1_dict']['f1_test'])


# make array for HTRU2 dataset BEST PARAM ROC AUC TEST Scores (for all algorithms)
htru2_logreg_roc_auc = []
htru2_knn_roc_auc = []
htru2_rforest_roc_auc = []
htru2_dtree_roc_auc = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    htru2_logreg_roc_auc.append(htru2DATA[trial]['LogReg']['roc_auc_dict']['roc_auc_test'])
    htru2_knn_roc_auc.append(htru2DATA[trial]['KNN']['roc_auc_dict']['roc_auc_test'])
    htru2_rforest_roc_auc.append(htru2DATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_test'])
    htru2_dtree_roc_auc.append(htru2DATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_test'])
```

Break Down Occupancy Testing Data Scores

```
# get OCCUPANCY dataset score array
occupancyDATA = top_dict['Occupancy']

# make array for OCCUPANCY dataset BEST PARAM ACCURACY TEST Scores (for all algorithms)
occupancy_logreg_acc = []
occupancy_knn_acc = []
occupancy_rforest_acc = []
occupancy_dtree_acc = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    occupancy_logreg_acc.append(occupancyDATA[trial]['LogReg']['acc_dict']['acc_test'])
    occupancy_knn_acc.append(occupancyDATA[trial]['KNN']['acc_dict']['acc_test'])
    occupancy_rforest_acc.append(occupancyDATA[trial]['Ran_For']['acc_dict']['acc_test'])
    occupancy_dtree_acc.append(occupancyDATA[trial]['Dec_Tree']['acc_dict']['acc_test'])


# make array for OCCUPANCY dataset BEST PARAM F1 TEST Scores (for all algorithms)
occupancy_logreg_f1 = []
occupancy_knn_f1 = []
```

```
occupancy_rforest_f1 = []
occupancy_dtree_f1 = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    occupancy_logreg_f1.append(occupancyDATA[trial]['LogReg']['f1_dict']['f1_test'])
    occupancy_knn_f1.append(occupancyDATA[trial]['KNN']['f1_dict']['f1_test'])
    occupancy_rforest_f1.append(occupancyDATA[trial]['Ran_For']['f1_dict']['f1_test'])
    occupancy_dtree_f1.append(occupancyDATA[trial]['Dec_Tree']['f1_dict']['f1_test'])


# make array for OCCUPANCY dataset BEST PARAM ROC AUC TEST Scores (for all algorithms)
occupancy_logreg_roc_auc = []
occupancy_knn_roc_auc = []
occupancy_rforest_roc_auc = []
occupancy_dtree_roc_auc = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    occupancy_logreg_roc_auc.append(occupancyDATA[trial]['LogReg']['roc_auc_dict']['roc_auc_test'])
    occupancy_knn_roc_auc.append(occupancyDATA[trial]['KNN']['roc_auc_dict']['roc_auc_test'])
    occupancy_rforest_roc_auc.append(occupancyDATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_test'])
    occupancy_dtree_roc_auc.append(occupancyDATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_test'])
```

Get Accuracy Averages

```
# get average of logistic regression accuracy average
# get average of trials for adult logreg accuracy
adult_logreg_acc_avg = sum(adult_logreg_acc)/len(adult_logreg_acc)
# get average of trials for grid logreg accuracy
grid_logreg_acc_avg = sum(grid_logreg_acc)/len(grid_logreg_acc)
# get average of trials for htru2 logreg accuracy
htru2_logreg_acc_avg = sum(htru2_logreg_acc)/len(htru2_logreg_acc)
# get average of trials for occupancy logreg accuracy
occupancy_logreg_acc_avg = sum(occupancy_logreg_acc)/len(occupancy_logreg_acc)
# get average of all logreg accuracy average
logreg_accuracy_avg = (adult_logreg_acc_avg + grid_logreg_acc_avg + htru2_logreg_acc_avg +
                       occupancy_logreg_acc_avg) / 4
print(logreg_accuracy_avg)


# get average of knn accuracy average
# get average of trials for adult knn accuracy
adult_knn_acc_avg = sum(adult_knn_acc)/len(adult_knn_acc)
# get average of trials for grid knn accuracy
grid_knn_acc_avg = sum(grid_knn_acc)/len(grid_knn_acc)
# get average of trials for htru2 knn accuracy
htru2_knn_acc_avg = sum(htru2_knn_acc)/len(htru2_knn_acc)
# get average of trials for occupancy knn accuracy
occupancy_knn_acc_avg = sum(occupancy_knn_acc)/len(occupancy_knn_acc)
# get average of all knn accuracy average
knn_accuracy_avg = (adult_knn_acc_avg + grid_knn_acc_avg + htru2_knn_acc_avg +
                       occupancy_knn_acc_avg) / 4
print(knn_accuracy_avg)


# get average of random forest accuracy average
# get average of trials for adult logreg accuracy
adult_rforest_acc_avg = sum(adult_rforest_acc)/len(adult_rforest_acc)
# get average of trials for grid logreg accuracy
grid_rforest_acc_avg = sum(grid_rforest_acc)/len(grid_rforest_acc)
# get average of trials for htru2 logreg accuracy
htru2_rforest_acc_avg = sum(htru2_rforest_acc)/len(htru2_rforest_acc)
# get average of trials for occupancy logreg accuracy
occupancy_rforest_acc_avg = sum(occupancy_rforest_acc)/len(occupancy_rforest_acc)
# get average of all logreg accuracy average
rforest_accuracy_avg = (adult_rforest_acc_avg + grid_rforest_acc_avg + htru2_rforest_acc_avg +
```

```
                        occupancy_rforest_acc_avg) / 4
print(rforest_accuracy_avg)


# get average of decision tree accuracy average
# get average of trials for adult logreg accuracy
adult_dtree_acc_avg = sum(adult_dtree_acc)/len(adult_dtree_acc)
# get average of trials for grid logreg accuracy
grid_dtree_acc_avg = sum(grid_dtree_acc)/len(grid_dtree_acc)
# get average of trials for htru2 logreg accuracy
htru2_dtree_acc_avg = sum(htru2_dtree_acc)/len(htru2_dtree_acc)
# get average of trials for occupancy logreg accuracy
occupancy_dtree_acc_avg = sum(occupancy_dtree_acc)/len(occupancy_dtree_acc)
# get average of all logreg accuracy average
dtree_accuracy_avg = (adult_dtree_acc_avg + grid_dtree_acc_avg + htru2_dtree_acc_avg +
                        occupancy_dtree_acc_avg) / 4
print(dtree_accuracy_avg)
```

```
0.9789734842611258
0.9777950069778261
0.9792060784617771
0.9773608311366104
```

Get F1 Score Averages

```
# get average of logistic regression f1 average
# get average of trials for adult logreg f1
adult_logreg_f1_avg = sum(adult_logreg_f1)/len(adult_logreg_f1)
# get average of trials for grid logreg f1
grid_logreg_f1_avg = sum(grid_logreg_f1)/len(grid_logreg_f1)
# get average of trials for htru2 logreg f1
htru2_logreg_f1_avg = sum(htru2_logreg_f1)/len(htru2_logreg_f1)
# get average of trials for occupancy logreg f1
occupancy_logreg_f1_avg = sum(occupancy_logreg_f1)/len(occupancy_logreg_f1)
# get average of all logreg f1 average
logreg_f1_avg = (adult_logreg_f1_avg + grid_logreg_f1_avg + htru2_logreg_f1_avg +
                        occupancy_logreg_f1_avg) / 4
print(logreg_f1_avg)


# get average of knn f1 average
# get average of trials for adult knn f1
adult_knn_f1_avg = sum(adult_knn_f1)/len(adult_knn_f1)
# get average of trials for grid knn f1
grid_knn_f1_avg = sum(grid_knn_f1)/len(grid_knn_f1)
# get average of trials for htru2 knn f1
htru2_knn_f1_avg = sum(htru2_knn_f1)/len(htru2_knn_f1)
# get average of trials for occupancy knn f1
occupancy_knn_f1_avg = sum(occupancy_knn_f1)/len(occupancy_knn_f1)
# get average of all knn f1 average
knn_f1_avg = (adult_knn_f1_avg + grid_knn_f1_avg + htru2_knn_f1_avg +
                        occupancy_knn_f1_avg) / 4
print(knn_f1_avg)


# get average of random forest f1 average
# get average of trials for adult random forest f1
adult_rforest_f1_avg = sum(adult_rforest_f1)/len(adult_rforest_f1)
# get average of trials for grid random forest f1
grid_rforest_f1_avg = sum(grid_rforest_f1)/len(grid_rforest_f1)
# get average of trials for htru2 random forest f1
htru2_rforest_f1_avg = sum(htru2_rforest_f1)/len(htru2_rforest_f1)
# get average of trials for occupancy random forest f1
occupancy_rforest_f1_avg = sum(occupancy_rforest_f1)/len(occupancy_rforest_f1)
# get average of all random forest f1 average
rforest_f1_avg = (adult_rforest_f1_avg + grid_rforest_f1_avg + htru2_rforest_f1_avg +
```

```
                        occupancy_rforest_f1_avg) / 4
print(rforest_f1_avg)


# get average of decision tree f1 average
# get average of trials for adult decision tree f1
adult_dtree_f1_avg = sum(adult_dtree_f1)/len(adult_dtree_f1)
# get average of trials for grid decision tree f1
grid_dtree_f1_avg = sum(grid_dtree_f1)/len(grid_dtree_f1)
# get average of trials for htru2 decision tree f1
htru2_dtree_f1_avg = sum(htru2_dtree_f1)/len(htru2_dtree_f1)
# get average of trials for occupancy decision tree f1
occupancy_dtree_f1_avg = sum(occupancy_dtree_f1)/len(occupancy_dtree_f1)
# get average of all decision tree f1 average
dtree_f1_avg = (adult_dtree_f1_avg + grid_dtree_f1_avg + htru2_dtree_f1_avg +
                        occupancy_dtree_f1_avg) / 4
print(dtree_f1_avg)
```

```
0.877008666610727
0.8700741570392726
0.8805955849182254
0.8682126395995159
```

Get ROC AUC Averages

```
# get average of logistic regression roc_auc average
# get average of trials for adult logreg roc_auc
adult_logreg_roc_auc_avg = sum(adult_logreg_roc_auc)/len(adult_logreg_roc_auc)
# get average of trials for grid logreg roc_auc
grid_logreg_roc_auc_avg = sum(grid_logreg_roc_auc)/len(grid_logreg_roc_auc)
# get average of trials for htru2 logreg roc_auc
htru2_logreg_roc_auc_avg = sum(htru2_logreg_roc_auc)/len(htru2_logreg_roc_auc)
# get average of trials for occupancy logreg roc_auc
occupancy_logreg_roc_auc_avg = sum(occupancy_logreg_roc_auc)/len(occupancy_logreg_roc_auc)
# get average of all logreg roc_auc average
logreg_roc_auc_avg = (adult_logreg_roc_auc_avg + grid_logreg_roc_auc_avg + htru2_logreg_roc_auc_avg +
                        occupancy_logreg_roc_auc_avg) / 4
print(logreg_roc_auc_avg)


# get average of knn roc_auc average
# get average of trials for adult knn roc_auc
adult_knn_roc_auc_avg = sum(adult_knn_roc_auc)/len(adult_knn_roc_auc)
# get average of trials for grid knn roc_auc
grid_knn_roc_auc_avg = sum(grid_knn_roc_auc)/len(grid_knn_roc_auc)
# get average of trials for htru2 knn roc_auc
htru2_knn_roc_auc_avg = sum(htru2_knn_roc_auc)/len(htru2_knn_roc_auc)
# get average of trials for occupancy knn roc_auc
occupancy_knn_roc_auc_avg = sum(occupancy_knn_roc_auc)/len(occupancy_knn_roc_auc)
# get average of all knn roc_auc average
knn_roc_auc_avg = (adult_knn_roc_auc_avg + grid_knn_roc_auc_avg + htru2_knn_roc_auc_avg +
                        occupancy_knn_roc_auc_avg) / 4
print(knn_roc_auc_avg)


# get average of random forest roc_auc average
# get average of trials for adult random forest roc_auc
adult_rforest_roc_auc_avg = sum(adult_rforest_roc_auc)/len(adult_rforest_roc_auc)
# get average of trials for grid random forest roc_auc
grid_rforest_roc_auc_avg = sum(grid_rforest_roc_auc)/len(grid_rforest_roc_auc)
# get average of trials for htru2 random forest roc_auc
htru2_rforest_roc_auc_avg = sum(htru2_rforest_roc_auc)/len(htru2_rforest_roc_auc)
# get average of trials for occupancy random forest roc_auc
occupancy_rforest_roc_auc_avg = sum(occupancy_rforest_roc_auc)/len(occupancy_rforest_roc_auc)
# get average of all random forest roc_auc average
rforest_roc_auc_avg = (adult_rforest_roc_auc_avg + grid_rforest_roc_auc_avg + htru2_rforest_roc_auc_avg +
```

```
                        occupancy_rforest_roc_auc_avg) / 4
print(rforest_roc_auc_avg)


# get average of decision tree roc_auc average
# get average of trials for adult decision tree roc_auc
adult_dtree_roc_auc_avg = sum(adult_dtree_roc_auc)/len(adult_dtree_roc_auc)
# get average of trials for grid decision tree roc_auc
grid_dtree_roc_auc_avg = sum(grid_dtree_roc_auc)/len(grid_dtree_roc_auc)
# get average of trials for htru2 decision tree roc_auc
htru2_dtree_roc_auc_avg = sum(htru2_dtree_roc_auc)/len(htru2_dtree_roc_auc)
# get average of trials for occupancy decision tree roc_auc
occupancy_dtree_roc_auc_avg = sum(occupancy_dtree_roc_auc)/len(occupancy_dtree_roc_auc)
# get average of all decision tree roc_auc average
dtree_roc_auc_avg = (adult_dtree_roc_auc_avg + grid_dtree_roc_auc_avg + htru2_dtree_roc_auc_avg +
                        occupancy_dtree_roc_auc_avg) / 4
print(dtree_roc_auc_avg)
```

```
0.9034136692737714
0.8806468211068206
0.9160785233923001
0.9139179675960829
```

Analyze Score Data by SCORES and MODEL (Table 2)

```
# row labels
models = ['LogReg', 'KNN', 'RF', 'DT']
# accuracy column
t2_acc = [logreg_accuracy_avg, knn_accuracy_avg, rforest_accuracy_avg, dtree_accuracy_avg]
# f1 column
t2_f1 = [logreg_f1_avg, knn_f1_avg, rforest_f1_avg, dtree_f1_avg]
# roc auc column
t2_roc_auc = [logreg_roc_auc_avg, knn_roc_auc_avg, rforest_roc_auc_avg, dtree_roc_auc_avg]
# get average of rows
t2_logreg_mean = (logreg_accuracy_avg + logreg_f1_avg + logreg_roc_auc_avg)/3
t2_knn_mean = (knn_accuracy_avg + knn_f1_avg + knn_roc_auc_avg)/3
t2_rforest_mean = (rforest_accuracy_avg + rforest_f1_avg + rforest_roc_auc_avg)/3
t2_dtree_mean = (dtree_accuracy_avg + dtree_f1_avg + dtree_roc_auc_avg)/3
# mean column
t2_mean = [t2_logreg_mean, t2_knn_mean, t2_rforest_mean, t2_dtree_mean]
# make dictionary and dataframe
t2 = {'MODEL': models, 'ACC': t2_acc, 'F1': t2_f1, 'ROC AUC': t2_roc_auc, 'MEAN': t2_mean}
t2 = pd.DataFrame.from_dict(t2)
t2.sort_values(by='MEAN', ascending=False, inplace=True)
t2
```

	MODEL	ACC	F1	ROC AUC	MEAN
2	RF	0.979206	0.880596	0.916079	0.925293
3	DT	0.977361	0.868213	0.913918	0.919830
0	LogReg	0.978973	0.877009	0.903414	0.919799
1	KNN	0.977795	0.870074	0.880647	0.909505

Analyze Score Data by DATASET and MODEL (Table 3)

```
# adult column
adult_logreg_avg = (adult_logreg_acc_avg + adult_logreg_f1_avg + adult_logreg_roc_auc_avg) / 3
adult_knn_avg = (adult_knn_acc_avg + adult_knn_f1_avg + adult_knn_roc_auc_avg) / 3
adult_rforest_avg = (adult_rforest_acc_avg + adult_rforest_f1_avg + adult_rforest_roc_auc_avg) / 3
adult_dtree_avg = (adult_dtree_acc_avg + adult_dtree_f1_avg + adult_dtree_roc_auc_avg) / 3
t3_adult = [adult_logreg_avg, adult_knn_avg, adult_rforest_avg, adult_dtree_avg]
```

```
# grid column
grid_logreg_avg = (grid_logreg_acc_avg + grid_logreg_f1_avg + grid_logreg_roc_auc_avg) / 3
grid_knn_avg = (grid_knn_acc_avg + grid_knn_f1_avg + grid_knn_roc_auc_avg) / 3
grid_rforest_avg = (grid_rforest_acc_avg + grid_rforest_f1_avg + grid_rforest_roc_auc_avg) / 3
grid_dtree_avg = (grid_dtree_acc_avg + grid_dtree_f1_avg + grid_dtree_roc_auc_avg) / 3
t3_grid = [grid_logreg_avg, grid_knn_avg, grid_rforest_avg, grid_dtree_avg]

# htru2 column
htru2_logreg_avg = (htru2_logreg_acc_avg + htru2_logreg_f1_avg + htru2_logreg_roc_auc_avg) / 3
htru2_knn_avg = (htru2_knn_acc_avg + htru2_knn_f1_avg + htru2_knn_roc_auc_avg) / 3
htru2_rforest_avg = (htru2_rforest_acc_avg + htru2_rforest_f1_avg + htru2_rforest_roc_auc_avg) / 3
htru2_dtree_avg = (htru2_dtree_acc_avg + htru2_dtree_f1_avg + htru2_dtree_roc_auc_avg) / 3
t3_htru2 = [htru2_logreg_avg, htru2_knn_avg, htru2_rforest_avg, htru2_dtree_avg]

# occupancy column
occupancy_logreg_avg = (occupancy_logreg_acc_avg + occupancy_logreg_f1_avg + occupancy_logreg_roc_auc_avg) /
occupancy_knn_avg = (occupancy_knn_acc_avg + occupancy_knn_f1_avg + occupancy_knn_roc_auc_avg) / 3
occupancy_rforest_avg = (occupancy_rforest_acc_avg + occupancy_rforest_f1_avg + occupancy_rforest_roc_auc_avg
occupancy_dtree_avg = (occupancy_dtree_acc_avg + occupancy_dtree_f1_avg + occupancy_dtree_roc_auc_avg) / 3
t3_occupancy = [occupancy_logreg_avg, occupancy_knn_avg, occupancy_rforest_avg, occupancy_dtree_avg]

# mean column
# get average of rows
t3_logreg_mean = (adult_logreg_avg + grid_logreg_avg + htru2_logreg_avg + occupancy_logreg_avg)/4
t3_knn_mean = (adult_knn_avg + grid_knn_avg + htru2_knn_avg + occupancy_knn_avg)/4
t3_rforest_mean = (adult_rforest_avg + grid_rforest_avg + htru2_rforest_avg + occupancy_rforest_avg)/4
t3_dtree_mean = (adult_dtree_avg + grid_dtree_avg + htru2_dtree_avg + occupancy_dtree_avg)/4
# mean column
t3_mean = [t3_logreg_mean, t3_knn_mean, t3_rforest_mean, t3_dtree_mean]

# make dictionary and dataframe
t3 = {'MODEL': models, 'ADULT': t3_adult, 'GRID': t3_grid, 'HTRU2': t3_htru2, 'OCCUPANCY': t3_occupancy, 'ME
t3 = pd.DataFrame.from_dict(t3)
t3.sort_values(by='MEAN', ascending=False, inplace=True)
t3
```

	MODEL	ADULT	GRID	HTRU2	OCCUPANCY	MEAN
2	RF	0.925293	0.925293	0.925293	0.925293	0.925293
3	DT	0.919830	0.919830	0.919830	0.919830	0.919830
0	LogReg	0.919799	0.919799	0.919799	0.919799	0.919799
1	KNN	0.909505	0.909505	0.909505	0.909505	0.909505

Secondary Results Analysis

Break Down Adult Training Data Scores

```
# get adult dataset score array
adultDATA = top_dict['Adult']

# make array for ADULT dataset BEST PARAM ACCURACY train Scores (for all algorithms)
adult_logreg_acc_train = []
adult_knn_acc_train = []
adult_rforest_acc_train = []
adult_dtree_acc_train = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    adult_logreg_acc_train.append(adultDATA[trial]['LogReg']['acc_dict']['acc_train'])
    adult_knn_acc_train.append(adultDATA[trial]['KNN']['acc_dict']['acc_train'])
    adult_rforest_acc_train.append(adultDATA[trial]['Ran_For']['acc_dict']['acc_train'])
```

```
    adult_dtree_acc_train.append(adultDATA[trial]['Dec_Tree']['acc_dict']['acc_train'])


# make array for ADULT dataset BEST PARAM F1 train Scores (for all algorithms)
adult_logreg_f1_train = []
adult_knn_f1_train = []
adult_rforest_f1_train = []
adult_dtree_f1_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    adult_logreg_f1_train.append(adultDATA[trial]['LogReg']['f1_dict']['f1_train'])
    adult_knn_f1_train.append(adultDATA[trial]['KNN']['f1_dict']['f1_train'])
    adult_rforest_f1_train.append(adultDATA[trial]['Ran_For']['f1_dict']['f1_train'])
    adult_dtree_f1_train.append(adultDATA[trial]['Dec_Tree']['f1_dict']['f1_train'])


# make array for ADULT dataset BEST PARAM ROC AUC train Scores (for all algorithms)
adult_logreg_roc_auc_train = []
adult_knn_roc_auc_train = []
adult_rforest_roc_auc_train = []
adult_dtree_roc_auc_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    adult_logreg_roc_auc_train.append(adultDATA[trial]['LogReg']['roc_auc_dict']['roc_auc_train'])
    adult_knn_roc_auc_train.append(adultDATA[trial]['KNN']['roc_auc_dict']['roc_auc_train'])
    adult_rforest_roc_auc_train.append(adultDATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_train'])
    adult_dtree_roc_auc_train.append(adultDATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_train'])
```

Break Down Grid Training Data Scores

```
# get grid dataset score array
gridDATA = top_dict['Grid']

# make array for grid dataset BEST PARAM ACCURACY train Scores (for all algorithms)
grid_logreg_acc_train = []
grid_knn_acc_train = []
grid_rforest_acc_train = []
grid_dtree_acc_train = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    grid_logreg_acc_train.append(gridDATA[trial]['LogReg']['acc_dict']['acc_train'])
    grid_knn_acc_train.append(gridDATA[trial]['KNN']['acc_dict']['acc_train'])
    grid_rforest_acc_train.append(gridDATA[trial]['Ran_For']['acc_dict']['acc_train'])
    grid_dtree_acc_train.append(gridDATA[trial]['Dec_Tree']['acc_dict']['acc_train'])


# make array for grid dataset BEST PARAM F1 train Scores (for all algorithms)
grid_logreg_f1_train = []
grid_knn_f1_train = []
grid_rforest_f1_train = []
grid_dtree_f1_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    grid_logreg_f1_train.append(gridDATA[trial]['LogReg']['f1_dict']['f1_train'])
    grid_knn_f1_train.append(gridDATA[trial]['KNN']['f1_dict']['f1_train'])
    grid_rforest_f1_train.append(gridDATA[trial]['Ran_For']['f1_dict']['f1_train'])
    grid_dtree_f1_train.append(gridDATA[trial]['Dec_Tree']['f1_dict']['f1_train'])


# make array for grid dataset BEST PARAM ROC AUC train Scores (for all algorithms)
grid_logreg_roc_auc_train = []
grid_knn_roc_auc_train = []
grid_rforest_roc_auc_train = []
grid_dtree_roc_auc_train = []
# get best F1 scores (for all algorithms)
```

```
for trial in range(5):
    grid_logreg_roc_auc_train.append(gridDATA[trial]['LogReg']['roc_auc_dict']['roc_auc_train'])
    grid_knn_roc_auc_train.append(gridDATA[trial]['KNN']['roc_auc_dict']['roc_auc_train'])
    grid_rforest_roc_auc_train.append(gridDATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_train'])
    grid_dtree_roc_auc_train.append(gridDATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_train'])
```

Break Down HTRU2 Training Data Scores

```
# get htru2 dataset score array
htru2DATA = top_dict['HTRU2']

# make array for htru2 dataset BEST PARAM ACCURACY train Scores (for all algorithms)
htru2_logreg_acc_train = []
htru2_knn_acc_train = []
htru2_rforest_acc_train = []
htru2_dtree_acc_train = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    htru2_logreg_acc_train.append(htru2DATA[trial]['LogReg']['acc_dict']['acc_train'])
    htru2_knn_acc_train.append(htru2DATA[trial]['KNN']['acc_dict']['acc_train'])
    htru2_rforest_acc_train.append(htru2DATA[trial]['Ran_For']['acc_dict']['acc_train'])
    htru2_dtree_acc_train.append(htru2DATA[trial]['Dec_Tree']['acc_dict']['acc_train'])


# make array for htru2 dataset BEST PARAM F1 train Scores (for all algorithms)
htru2_logreg_f1_train = []
htru2_knn_f1_train = []
htru2_rforest_f1_train = []
htru2_dtree_f1_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    htru2_logreg_f1_train.append(htru2DATA[trial]['LogReg']['f1_dict']['f1_train'])
    htru2_knn_f1_train.append(htru2DATA[trial]['KNN']['f1_dict']['f1_train'])
    htru2_rforest_f1_train.append(htru2DATA[trial]['Ran_For']['f1_dict']['f1_train'])
    htru2_dtree_f1_train.append(htru2DATA[trial]['Dec_Tree']['f1_dict']['f1_train'])


# make array for htru2 dataset BEST PARAM ROC AUC train Scores (for all algorithms)
htru2_logreg_roc_auc_train = []
htru2_knn_roc_auc_train = []
htru2_rforest_roc_auc_train = []
htru2_dtree_roc_auc_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    htru2_logreg_roc_auc_train.append(htru2DATA[trial]['LogReg']['roc_auc_dict']['roc_auc_train'])
    htru2_knn_roc_auc_train.append(htru2DATA[trial]['KNN']['roc_auc_dict']['roc_auc_train'])
    htru2_rforest_roc_auc_train.append(htru2DATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_train'])
    htru2_dtree_roc_auc_train.append(htru2DATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_train'])
```

Break Down Occupancy Training Data Scores

```
# get occupancy dataset score array
occupancyDATA = top_dict['Occupancy']

# make array for occupancy dataset BEST PARAM ACCURACY train Scores (for all algorithms)
occupancy_logreg_acc_train = []
occupancy_knn_acc_train = []
occupancy_rforest_acc_train = []
occupancy_dtree_acc_train = []
# get best ACCURACY scores (for all algorithms)
for trial in range(5):
    occupancy_logreg_acc_train.append(occupancyDATA[trial]['LogReg']['acc_dict']['acc_train'])
    occupancy_knn_acc_train.append(occupancyDATA[trial]['KNN']['acc_dict']['acc_train'])
    occupancy_rforest_acc_train.append(occupancyDATA[trial]['Ran_For']['acc_dict']['acc_train'])
```

```
    occupancy_dtree_acc_train.append(occupancyDATA[trial]['Dec_Tree']['acc_dict']['acc_train'])


# make array for occupancy dataset BEST PARAM F1 train Scores (for all algorithms)
occupancy_logreg_f1_train = []
occupancy_knn_f1_train = []
occupancy_rforest_f1_train = []
occupancy_dtree_f1_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    occupancy_logreg_f1_train.append(occupancyDATA[trial]['LogReg']['f1_dict']['f1_train'])
    occupancy_knn_f1_train.append(occupancyDATA[trial]['KNN']['f1_dict']['f1_train'])
    occupancy_rforest_f1_train.append(occupancyDATA[trial]['Ran_For']['f1_dict']['f1_train'])
    occupancy_dtree_f1_train.append(occupancyDATA[trial]['Dec_Tree']['f1_dict']['f1_train'])


# make array for occupancy dataset BEST PARAM ROC AUC train Scores (for all algorithms)
occupancy_logreg_roc_auc_train = []
occupancy_knn_roc_auc_train = []
occupancy_rforest_roc_auc_train = []
occupancy_dtree_roc_auc_train = []
# get best F1 scores (for all algorithms)
for trial in range(5):
    occupancy_logreg_roc_auc_train.append(occupancyDATA[trial]['LogReg']['roc_auc_dict']['roc_auc_train'])
    occupancy_knn_roc_auc_train.append(occupancyDATA[trial]['KNN']['roc_auc_dict']['roc_auc_train'])
    occupancy_rforest_roc_auc_train.append(occupancyDATA[trial]['Ran_For']['roc_auc_dict']['roc_auc_train'])
    occupancy_dtree_roc_auc_train.append(occupancyDATA[trial]['Dec_Tree']['roc_auc_dict']['roc_auc_train'])
```

Get Accuracy Averages

```
# get average of logistic regression accuracy average
# get average of trials for adult logreg accuracy
adult_logreg_acc_train_avg = sum(adult_logreg_acc_train)/len(adult_logreg_acc_train)
# get average of trials for grid logreg accuracy
grid_logreg_acc_train_avg = sum(grid_logreg_acc_train)/len(grid_logreg_acc_train)
# get average of trials for htru2 logreg accuracy
htru2_logreg_acc_train_avg = sum(htru2_logreg_acc_train)/len(htru2_logreg_acc_train)
# get average of trials for occupancy logreg accuracy
occupancy_logreg_acc_train_avg = sum(occupancy_logreg_acc_train)/len(occupancy_logreg_acc_train)
# get average of all logreg accuracy average
logreg_accuracy_train_avg = (adult_logreg_acc_train_avg + grid_logreg_acc_train_avg + htru2_logreg_acc_train_
                        occupancy_logreg_acc_train_avg) / 4
print(logreg_accuracy_train_avg)


# get average of knn accuracy average
# get average of trials for adult knn accuracy
adult_knn_acc_train_avg = sum(adult_knn_acc_train)/len(adult_knn_acc_train)
# get average of trials for grid knn accuracy
grid_knn_acc_train_avg = sum(grid_knn_acc_train)/len(grid_knn_acc_train)
# get average of trials for htru2 knn accuracy
htru2_knn_acc_train_avg = sum(htru2_knn_acc_train)/len(htru2_knn_acc_train)
# get average of trials for occupancy knn accuracy
occupancy_knn_acc_train_avg = sum(occupancy_knn_acc_train)/len(occupancy_knn_acc_train)
# get average of all knn accuracy average
knn_accuracy_train_avg = (adult_knn_acc_train_avg + grid_knn_acc_train_avg + htru2_knn_acc_train_avg +
                        occupancy_knn_acc_train_avg) / 4
print(knn_accuracy_train_avg)


# get average of random forest accuracy average
# get average of trials for adult random forest accuracy
adult_rforest_acc_train_avg = sum(adult_rforest_acc_train)/len(adult_rforest_acc_train)
# get average of trials for grid random forest accuracy
grid_rforest_acc_train_avg = sum(grid_rforest_acc_train)/len(grid_rforest_acc_train)
```

```
# get average of trials for htru2 random forest accuracy
htru2_rforest_acc_train_avg = sum(htru2_rforest_acc_train)/len(htru2_rforest_acc_train)
# get average of trials for occupancy random forest accuracy
occupancy_rforest_acc_train_avg = sum(occupancy_rforest_acc_train)/len(occupancy_rforest_acc_train)
# get average of all random forest accuracy average
rforest_accuracy_train_avg = (adult_rforest_acc_train_avg + grid_rforest_acc_train_avg + htru2_rforest_acc_tr
                        occupancy_rforest_acc_train_avg) / 4
print(rforest_accuracy_train_avg)


# get average of decision tree accuracy average
# get average of trials for adult decision tree accuracy
adult_dtree_acc_train_avg = sum(adult_dtree_acc_train)/len(adult_dtree_acc_train)
# get average of trials for grid decision tree accuracy
grid_dtree_acc_train_avg = sum(grid_dtree_acc_train)/len(grid_dtree_acc_train)
# get average of trials for htru2 decision tree accuracy
htru2_dtree_acc_train_avg = sum(htru2_dtree_acc_train)/len(htru2_dtree_acc_train)
# get average of trials for occupancy decision tree accuracy
occupancy_dtree_acc_train_avg = sum(occupancy_dtree_acc_train)/len(occupancy_dtree_acc_train)
# get average of all decision tree accuracy average
dtree_accuracy_train_avg = (adult_dtree_acc_train_avg + grid_dtree_acc_train_avg + htru2_dtree_acc_train_avg
                        occupancy_dtree_acc_train_avg) / 4
print(dtree_accuracy_train_avg)
```

```
0.97948
1.0
1.0
0.98232
```

Get F1 Averages

```
# get average of logistic regression f1 average
# get average of trials for adult logreg f1
adult_logreg_f1_train_avg = sum(adult_logreg_f1_train)/len(adult_logreg_f1_train)
# get average of trials for grid logreg f1
grid_logreg_f1_train_avg = sum(grid_logreg_f1_train)/len(grid_logreg_f1_train)
# get average of trials for htru2 logreg f1
htru2_logreg_f1_train_avg = sum(htru2_logreg_f1_train)/len(htru2_logreg_f1_train)
# get average of trials for occupancy logreg f1
occupancy_logreg_f1_train_avg = sum(occupancy_logreg_f1_train)/len(occupancy_logreg_f1_train)
# get average of all logreg f1 average
logreg_f1_train_avg = (adult_logreg_f1_train_avg + grid_logreg_f1_train_avg + htru2_logreg_f1_train_avg +
                        occupancy_logreg_f1_train_avg) / 4
print(logreg_f1_train_avg)


# get average of knn f1 average
# get average of trials for adult knn f1
adult_knn_f1_train_avg = sum(adult_knn_f1_train)/len(adult_knn_f1_train)
# get average of trials for grid knn f1
grid_knn_f1_train_avg = sum(grid_knn_f1_train)/len(grid_knn_f1_train)
# get average of trials for htru2 knn f1
htru2_knn_f1_train_avg = sum(htru2_knn_f1_train)/len(htru2_knn_f1_train)
# get average of trials for occupancy knn f1
occupancy_knn_f1_train_avg = sum(occupancy_knn_f1_train)/len(occupancy_knn_f1_train)
# get average of all knn f1 average
knn_f1_train_avg = (adult_knn_f1_train_avg + grid_knn_f1_train_avg + htru2_knn_f1_train_avg +
                        occupancy_knn_f1_train_avg) / 4
print(knn_f1_train_avg)


# get average of random forest f1 average
# get average of trials for adult random forest f1
adult_rforest_f1_train_avg = sum(adult_rforest_f1_train)/len(adult_rforest_f1_train)
# get average of trials for grid random forest f1
grid_rforest_f1_train_avg = sum(grid_rforest_f1_train)/len(grid_rforest_f1_train)
```

```
# get average of trials for htru2 random forest f1
htru2_rforest_f1_train_avg = sum(htru2_rforest_f1_train)/len(htru2_rforest_f1_train)
# get average of trials for occupancy random forest f1
occupancy_rforest_f1_train_avg = sum(occupancy_rforest_f1_train)/len(occupancy_rforest_f1_train)
# get average of all random forest f1 average
rforest_f1_train_avg = (adult_rforest_f1_train_avg + grid_rforest_f1_train_avg + htru2_rforest_f1_train_avg +
                        occupancy_rforest_f1_train_avg) / 4
print(rforest_f1_train_avg)


# get average of decision tree f1 average
# get average of trials for adult decision tree f1
adult_dtree_f1_train_avg = sum(adult_dtree_f1_train)/len(adult_dtree_f1_train)
# get average of trials for grid decision tree f1
grid_dtree_f1_train_avg = sum(grid_dtree_f1_train)/len(grid_dtree_f1_train)
# get average of trials for htru2 decision tree f1
htru2_dtree_f1_train_avg = sum(htru2_dtree_f1_train)/len(htru2_dtree_f1_train)
# get average of trials for occupancy decision tree f1
occupancy_dtree_f1_train_avg = sum(occupancy_dtree_f1_train)/len(occupancy_dtree_f1_train)
# get average of all decision tree f1 average
dtree_f1_train_avg = (adult_dtree_f1_train_avg + grid_dtree_f1_train_avg + htru2_dtree_f1_train_avg +
                        occupancy_dtree_f1_train_avg) / 4
print(dtree_f1_train_avg)
```

```
0.8808835297060369
1.0
1.0
0.8974318222801845
```

Get ROC AUC Averages

```
# get average of logistic regression roc_auc average
# get average of trials for adult logreg roc_auc
adult_logreg_roc_auc_train_avg = sum(adult_logreg_roc_auc_train)/len(adult_logreg_roc_auc_train)
# get average of trials for grid logreg roc_auc
grid_logreg_roc_auc_train_avg = sum(grid_logreg_roc_auc_train)/len(grid_logreg_roc_auc_train)
# get average of trials for htru2 logreg roc_auc
htru2_logreg_roc_auc_train_avg = sum(htru2_logreg_roc_auc_train)/len(htru2_logreg_roc_auc_train)
# get average of trials for occupancy logreg roc_auc
occupancy_logreg_roc_auc_train_avg = sum(occupancy_logreg_roc_auc_train)/len(occupancy_logreg_roc_auc_train)
# get average of all logreg roc_auc average
logreg_roc_auc_train_avg = (adult_logreg_roc_auc_train_avg + grid_logreg_roc_auc_train_avg + htru2_logreg_roc
                        occupancy_logreg_roc_auc_train_avg) / 4
print(logreg_roc_auc_train_avg)


# get average of knn roc_auc average
# get average of trials for adult knn roc_auc
adult_knn_roc_auc_train_avg = sum(adult_knn_roc_auc_train)/len(adult_knn_roc_auc_train)
# get average of trials for grid knn roc_auc
grid_knn_roc_auc_train_avg = sum(grid_knn_roc_auc_train)/len(grid_knn_roc_auc_train)
# get average of trials for htru2 knn roc_auc
htru2_knn_roc_auc_train_avg = sum(htru2_knn_roc_auc_train)/len(htru2_knn_roc_auc_train)
# get average of trials for occupancy knn roc_auc
occupancy_knn_roc_auc_train_avg = sum(occupancy_knn_roc_auc_train)/len(occupancy_knn_roc_auc_train)
# get average of all knn roc_auc average
knn_roc_auc_train_avg = (adult_knn_roc_auc_train_avg + grid_knn_roc_auc_train_avg + htru2_knn_roc_auc_train_a
                        occupancy_knn_roc_auc_train_avg) / 4
print(knn_roc_auc_train_avg)


# get average of random forest roc_auc average
# get average of trials for adult random forest roc_auc
adult_rforest_roc_auc_train_avg = sum(adult_rforest_roc_auc_train)/len(adult_rforest_roc_auc_train)
# get average of trials for grid random forest roc_auc
grid_rforest_roc_auc_train_avg = sum(grid_rforest_roc_auc_train)/len(grid_rforest_roc_auc_train)
```

```
# get average of trials for htru2 random forest roc_auc
htru2_rforest_roc_auc_train_avg = sum(htru2_rforest_roc_auc_train)/len(htru2_rforest_roc_auc_train)
# get average of trials for occupancy random forest roc_auc
occupancy_rforest_roc_auc_train_avg = sum(occupancy_rforest_roc_auc_train)/len(occupancy_rforest_roc_auc_tra
# get average of all random forest roc_auc average
rforest_roc_auc_train_avg = (adult_rforest_roc_auc_train_avg + grid_rforest_roc_auc_train_avg + htru2_rfores
                        occupancy_rforest_roc_auc_train_avg) / 4
print(rforest_roc_auc_train_avg)


# get average of decision tree roc_auc average
# get average of trials for adult decision tree roc_auc
adult_dtree_roc_auc_train_avg = sum(adult_dtree_roc_auc_train)/len(adult_dtree_roc_auc_train)
# get average of trials for grid decision tree roc_auc
grid_dtree_roc_auc_train_avg = sum(grid_dtree_roc_auc_train)/len(grid_dtree_roc_auc_train)
# get average of trials for htru2 decision tree roc_auc
htru2_dtree_roc_auc_train_avg = sum(htru2_dtree_roc_auc_train)/len(htru2_dtree_roc_auc_train)
# get average of trials for occupancy decision tree roc_auc
occupancy_dtree_roc_auc_train_avg = sum(occupancy_dtree_roc_auc_train)/len(occupancy_dtree_roc_auc_train)
# get average of all decision tree roc_auc average
dtree_roc_auc_train_avg = (adult_dtree_roc_auc_train_avg + grid_dtree_roc_auc_train_avg + htru2_dtree_roc_au
                        occupancy_dtree_roc_auc_train_avg) / 4
print(dtree_roc_auc_train_avg)
```

```
0.9064074515023407
1.0
1.0
0.9309898541156025
```

Analyze Train Score Data by SCORES and MODEL (Secondary)

```
# accuracy column
t4_acc = [logreg_accuracy_train_avg, knn_accuracy_train_avg, rforest_accuracy_train_avg, dtree_accuracy_trai
# f1 column
t4_f1 = [logreg_f1_train_avg, knn_f1_train_avg, rforest_f1_train_avg, dtree_f1_train_avg]
# roc auc column
t4_roc_auc = [logreg_roc_auc_train_avg, knn_roc_auc_train_avg, rforest_roc_auc_train_avg, dtree_roc_auc_trai
# get average of rows
t4_logreg_mean = (logreg_accuracy_train_avg + logreg_f1_train_avg + logreg_roc_auc_train_avg)/3
t4_knn_mean = (knn_accuracy_train_avg + knn_f1_train_avg + knn_roc_auc_train_avg)/3
t4_rforest_mean = (rforest_accuracy_train_avg + rforest_f1_train_avg + rforest_roc_auc_train_avg)/3
t4_dtree_mean = (dtree_accuracy_train_avg + dtree_f1_train_avg + dtree_roc_auc_train_avg)/3
# mean column
t4_mean = [t4_logreg_mean, t4_knn_mean, t4_rforest_mean, t4_dtree_mean]
# make dictionary and dataframe
t4 = {'MODEL': models, 'ACC': t4_acc, 'F1': t4_f1, 'ROC AUC': t4_roc_auc, 'MEAN': t4_mean}
t4 = pd.DataFrame.from_dict(t4)
t4.sort_values(by='MEAN', ascending=False, inplace=True)
t4
```

	MODEL	ACC	F1	ROC AUC	MEAN
1	KNN	1.00000	1.000000	1.000000	1.000000
2	RF	1.00000	1.000000	1.000000	1.000000
3	DT	0.98232	0.897432	0.930990	0.936914
0	LogReg	0.97948	0.880884	0.906407	0.922257

Analyze Train Score Data by DATASET and MODEL (Secondary)

```
# adult column
adult_logreg_train_avg = (adult_logreg_acc_train_avg + adult_logreg_f1_train_avg + adult_logreg_roc_auc_trai
```

```
adult_knn_train_avg = (adult_knn_acc_train_avg + adult_knn_f1_train_avg + adult_knn_roc_auc_train_avg) / 3
adult_rforest_train_avg = (adult_rforest_acc_train_avg + adult_rforest_f1_train_avg + adult_rforest_roc_auc_t
adult_dtree_train_avg = (adult_dtree_acc_train_avg + adult_dtree_f1_train_avg + adult_dtree_roc_auc_train_avg
t5_adult = [adult_logreg_train_avg, adult_knn_train_avg, adult_rforest_train_avg, adult_dtree_train_avg]

# grid column
grid_logreg_train_avg = (grid_logreg_acc_train_avg + grid_logreg_f1_train_avg + grid_logreg_roc_auc_train_avg
grid_knn_train_avg = (grid_knn_acc_train_avg + grid_knn_f1_train_avg + grid_knn_roc_auc_train_avg) / 3
grid_rforest_train_avg = (grid_rforest_acc_train_avg + grid_rforest_f1_train_avg + grid_rforest_roc_auc_trai
grid_dtree_train_avg = (grid_dtree_acc_train_avg + grid_dtree_f1_train_avg + grid_dtree_roc_auc_train_avg) /
t5_grid = [grid_logreg_train_avg, grid_knn_train_avg, grid_rforest_train_avg, grid_dtree_train_avg]

# htru2 column
htru2_logreg_train_avg = (htru2_logreg_acc_train_avg + htru2_logreg_f1_train_avg + htru2_logreg_roc_auc_trai
htru2_knn_train_avg = (htru2_knn_acc_train_avg + htru2_knn_f1_train_avg + htru2_knn_roc_auc_train_avg) / 3
htru2_rforest_train_avg = (htru2_rforest_acc_train_avg + htru2_rforest_f1_train_avg + htru2_rforest_roc_auc_t
htru2_dtree_train_avg = (htru2_dtree_acc_train_avg + htru2_dtree_f1_train_avg + htru2_dtree_roc_auc_train_avg
t5_htru2 = [htru2_logreg_train_avg, htru2_knn_train_avg, htru2_rforest_train_avg, htru2_dtree_train_avg]

# occupancy column
occupancy_logreg_train_avg = (occupancy_logreg_acc_train_avg + occupancy_logreg_f1_train_avg + occupancy_logr
occupancy_knn_train_avg = (occupancy_knn_acc_train_avg + occupancy_knn_f1_train_avg + occupancy_knn_roc_auc_t
occupancy_rforest_train_avg = (occupancy_rforest_acc_train_avg + occupancy_rforest_f1_train_avg + occupancy_r
occupancy_dtree_train_avg = (occupancy_dtree_acc_train_avg + occupancy_dtree_f1_train_avg + occupancy_dtree_r
t5_occupancy = [occupancy_logreg_train_avg, occupancy_knn_train_avg, occupancy_rforest_train_avg, occupancy_d

# mean column
# get average of rows
t5_logreg_mean = (adult_logreg_train_avg + grid_logreg_train_avg + htru2_logreg_train_avg + occupancy_logreg_
t5_knn_mean = (adult_knn_train_avg + grid_knn_train_avg + htru2_knn_train_avg + occupancy_knn_train_avg)/4
t5_rforest_mean = (adult_rforest_train_avg + grid_rforest_train_avg + htru2_rforest_train_avg + occupancy_rfo
t5_dtree_mean = (adult_dtree_train_avg + grid_dtree_train_avg + htru2_dtree_train_avg + occupancy_dtree_train
# mean column
t5_mean = [t5_logreg_mean, t5_knn_mean, t5_rforest_mean, t5_dtree_mean]

# make dictionary and dataframe
t5 = {'MODEL': models, 'ADULT': t5_adult, 'GRID': t5_grid, 'HTRU2': t5_htru2, 'OCCUPANCY': t5_occupancy, 'MEA
t5 = pd.DataFrame.from_dict(t5)
t5.sort_values(by='MEAN', ascending=False, inplace=True)
t5
```

	MODEL	ADULT	GRID	HTRU2	OCCUPANCY	MEAN
1	KNN	1.000000	1.000000	1.000000	1.000000	1.000000
2	RF	1.000000	1.000000	1.000000	1.000000	1.000000
3	DT	0.936914	0.936914	0.936914	0.936914	0.936914
0	LogReg	0.922257	0.922257	0.922257	0.922257	0.922257

Raw Test Data Scores (Secondary)

```
raw_adult_test_scores = {'LR ACC': adult_logreg_acc, 'LR F1': adult_logreg_f1, 'LR AUC': adult_logreg_roc_au
                         'KNN ACC': adult_knn_acc, 'KNN F1': adult_knn_f1, 'KNN AUC': adult_knn_roc_auc,
                         'RF ACC': adult_rforest_acc, 'RF F1': adult_rforest_f1, 'RF AUC': adult_rforest_roc
                         'DT ACC': adult_dtree_acc, 'DT F1': adult_dtree_f1, 'DT AUC': adult_dtree_roc_auc}
raw_adult = pd.DataFrame.from_dict(raw_adult_test_scores)
raw_adult.index = raw_adult.index + 1
raw_adult = raw_adult.transpose()
print('Adult')
print(raw_adult)

raw_grid_test_scores = {'LR ACC': grid_logreg_acc, 'LR F1': grid_logreg_f1, 'LR AUC': grid_logreg_roc_auc,
```

```
                          'KNN ACC': grid_knn_acc, 'KNN F1': grid_knn_f1, 'KNN AUC': grid_knn_roc_auc,
                          'RF ACC': grid_rforest_acc, 'RF F1': grid_rforest_f1, 'RF AUC': grid_rforest_roc_au
                          'DT ACC': grid_dtree_acc, 'DT F1': grid_dtree_f1, 'DT AUC': grid_dtree_roc_auc}
raw_grid = pd.DataFrame.from_dict(raw_grid_test_scores)
raw_grid.index = raw_grid.index + 1
raw_grid = raw_grid.transpose()
print('\nGrid')
print(raw_grid)


raw_htru2_test_scores = {'LR ACC': htru2_logreg_acc, 'LR F1': htru2_logreg_f1, 'LR AUC': htru2_logreg_roc_au
                          'KNN ACC': htru2_knn_acc, 'KNN F1': htru2_knn_f1, 'KNN AUC': htru2_knn_roc_auc,
                          'RF ACC': htru2_rforest_acc, 'RF F1': htru2_rforest_f1, 'RF AUC': htru2_rforest_roc_
                          'DT ACC': htru2_dtree_acc, 'DT F1': htru2_dtree_f1, 'DT AUC': htru2_dtree_roc_auc}
raw_htru2 = pd.DataFrame.from_dict(raw_htru2_test_scores)
raw_htru2.index = raw_htru2.index + 1
raw_htru2 = raw_htru2.transpose()
print('\nHTRU2')
print(raw_htru2)



raw_occupancy_test_scores = {'LR ACC': occupancy_logreg_acc, 'LR F1': occupancy_logreg_f1, 'LR AUC': occupanc
                          'KNN ACC': occupancy_knn_acc, 'KNN F1': occupancy_knn_f1, 'KNN AUC': occupancy_knn_r
                          'RF ACC': occupancy_rforest_acc, 'RF F1': occupancy_rforest_f1, 'RF AUC': occupancy_
                          'DT ACC': occupancy_dtree_acc, 'DT F1': occupancy_dtree_f1, 'DT AUC': occupancy_dtre
raw_occupancy = pd.DataFrame.from_dict(raw_occupancy_test_scores)
raw_occupancy.index = raw_occupancy.index + 1
raw_occupancy = raw_occupancy.transpose()
print('\nOccupancy')
print(raw_occupancy)
```

```
Adult
                1         2         3         4         5
LR ACC   0.979687  0.977981  0.979532  0.978989  0.978679
LR F1    0.884378  0.870320  0.878788  0.877763  0.873795
LR AUC   0.904898  0.899628  0.900090  0.909865  0.902587
KNN ACC  0.977826  0.976586  0.978679  0.977748  0.978136
KNN F1   0.871403  0.862602  0.872271  0.870779  0.873315
KNN AUC  0.891998  0.873915  0.879747  0.875265  0.882309
RF ACC   0.977748  0.979144  0.978989  0.979997  0.980152
RF F1    0.878261  0.880889  0.874197  0.884135  0.885496
RF AUC   0.915458  0.921015  0.912528  0.916647  0.914744
DT ACC   0.977516  0.976353  0.978601  0.977051  0.977283
DT F1    0.870420  0.864522  0.872029  0.865577  0.868515
DT AUC   0.914843  0.911677  0.908383  0.920627  0.914061

Grid
                1         2         3         4         5
LR ACC   0.979687  0.977981  0.979532  0.978989  0.978679
LR F1    0.884378  0.870320  0.878788  0.877763  0.873795
LR AUC   0.904898  0.899628  0.900090  0.909865  0.902587
KNN ACC  0.977826  0.976586  0.978679  0.977748  0.978136
KNN F1   0.871403  0.862602  0.872271  0.870779  0.873315
KNN AUC  0.891998  0.873915  0.879747  0.875265  0.882309
RF ACC   0.977748  0.979144  0.978989  0.979997  0.980152
RF F1    0.878261  0.880889  0.874197  0.884135  0.885496
RF AUC   0.915458  0.921015  0.912528  0.916647  0.914744
DT ACC   0.977516  0.976353  0.978601  0.977051  0.977283
DT F1    0.870420  0.864522  0.872029  0.865577  0.868515
DT AUC   0.914843  0.911677  0.908383  0.920627  0.914061

HTRU2
                1         2         3         4         5
LR ACC   0.979687  0.977981  0.979532  0.978989  0.978679
LR F1    0.884378  0.870320  0.878788  0.877763  0.873795
LR AUC   0.904898  0.899628  0.900090  0.909865  0.902587
KNN ACC  0.977826  0.976586  0.978679  0.977748  0.978136
```

```
KNN F1    0.871403  0.862602  0.872271  0.870779  0.873315
KNN AUC   0.891998  0.873915  0.879747  0.875265  0.882309
RF ACC    0.977748  0.979144  0.978989  0.979997  0.980152
RF F1     0.878261  0.880889  0.874197  0.884135  0.885496
RF AUC    0.915458  0.921015  0.912528  0.916647  0.914744
DT ACC    0.977516  0.976353  0.978601  0.977051  0.977283
DT F1     0.870420  0.864522  0.872029  0.865577  0.868515
DT AUC    0.914843  0.911677  0.908383  0.920627  0.914061

Occupancy
                1         2         3         4         5
LR ACC    0.979687  0.977981  0.979532  0.978989  0.978679
LR F1     0.884378  0.870320  0.878788  0.877763  0.873795
LR AUC    0.904898  0.899628  0.900090  0.909865  0.902587
KNN ACC   0.977826  0.976586  0.978679  0.977748  0.978136
KNN F1    0.871403  0.862602  0.872271  0.870779  0.873315
KNN AUC   0.891998  0.873915  0.879747  0.875265  0.882309
RF ACC    0.977748  0.979144  0.978989  0.979997  0.980152
RF F1     0.878261  0.880889  0.874197  0.884135  0.885496
RF AUC    0.915458  0.921015  0.912528  0.916647  0.914744
DT ACC    0.977516  0.976353  0.978601  0.977051  0.977283
DT F1     0.870420  0.864522  0.872029  0.865577  0.868515
DT AUC    0.914843  0.911677  0.908383  0.920627  0.914061
```

P-Value Tables for Table 2 (Secondary)

```
# import scipay stats to get t-test function
from scipy import stats
```

```
# best performing in ACC column (Random Tree)
# get array of values (5 trials X 4 datasets = 20 values)
best_acc = (adult_rforest_acc + grid_rforest_acc + htru2_rforest_acc + occupancy_rforest_acc)
# best performing in F1 column (Random Tree)
# get array of values (5 trials X 4 datasets = 20 values)
best_f1 = (adult_rforest_f1 + grid_rforest_f1 + htru2_rforest_f1 + occupancy_rforest_f1)
# best performing in ROC AUC column (Random Tree)
# get array of values (5 trials X 4 datasets = 20 values)
best_roc_auc = (adult_rforest_roc_auc + grid_rforest_roc_auc + htru2_rforest_roc_auc + occupancy_rforest_roc_
# best performing in MEAN column (Random Tree)
best_mean = [rforest_accuracy_avg, rforest_f1_avg, rforest_roc_auc_avg]

# get array of values for other rows
# get decision tree arrays
dtree_acc = (adult_dtree_acc + grid_dtree_acc + htru2_dtree_acc + occupancy_dtree_acc)
dtree_f1 = (adult_dtree_f1 + grid_dtree_f1 + htru2_dtree_f1 + occupancy_dtree_f1)
dtree_roc_auc = (adult_dtree_roc_auc + grid_dtree_roc_auc + htru2_dtree_roc_auc + occupancy_dtree_roc_auc)
dtree_mean = [dtree_accuracy_avg, dtree_f1_avg, dtree_roc_auc_avg]
# get logistic regression arrays
logreg_acc = (adult_logreg_acc + grid_logreg_acc + htru2_logreg_acc + occupancy_logreg_acc)
logreg_f1 = (adult_logreg_f1 + grid_logreg_f1 + htru2_logreg_f1 + occupancy_logreg_f1)
logreg_roc_auc = (adult_logreg_roc_auc + grid_logreg_roc_auc + htru2_logreg_roc_auc + occupancy_logreg_roc_au
logreg_mean = [logreg_accuracy_avg, logreg_f1_avg, logreg_roc_auc_avg]
# get logistic regression arrays
knn_acc = (adult_knn_acc + grid_knn_acc + htru2_knn_acc + occupancy_knn_acc)
knn_f1 = (adult_knn_f1 + grid_knn_f1 + htru2_knn_f1 + occupancy_knn_f1)
knn_roc_auc = (adult_knn_roc_auc + grid_knn_roc_auc + htru2_knn_roc_auc + occupancy_knn_roc_auc)
knn_mean = [knn_accuracy_avg, knn_f1_avg, knn_roc_auc_avg]


# make arrays for p-values
# array for ACC column
acc_pvalue = [None] * 4
# array for F1 column
f1_pvalue = [None] * 4
# array for ROC AUC column
roc_auc_pvalue = [None] * 4
```

```
# array for ROC AUC column
mean_pvalue = [None] * 4


# fill the pvalue array for accuracy
for i, acc, f1, roc_auc, mean in zip(range(4), [best_acc, dtree_acc, logreg_acc, knn_acc],
                         [best_f1, dtree_f1, logreg_f1, knn_f1], [best_roc_auc, dtree_roc_auc, logreg_roc_auc
                         [best_mean, dtree_mean, logreg_mean, knn_mean]):
    acc_pvalue[i] = stats.ttest_rel(best_acc, acc)[1]
    f1_pvalue[i] = stats.ttest_rel(best_f1, f1)[1]
    roc_auc_pvalue[i] = stats.ttest_rel(best_roc_auc, roc_auc)[1]
    mean_pvalue[i] = stats.ttest_rel(best_mean, mean)[1]

t6 = {'MODEL': ['RF', 'DT', 'LR', 'KNN'], 'ACC': acc_pvalue, 'F1': f1_pvalue, 'ROC AUC': roc_auc_pvalue, 'ME
t6 = pd.DataFrame.from_dict(t6)
t6
```

	MODEL	ACC	F1	ROC AUC	MEAN
0	RF	NaN	NaN	NaN	NaN
1	DT	0.000004	6.366546e-08	4.645252e-02	0.255234
2	LR	0.441568	5.163279e-02	5.370830e-10	0.277082
3	KNN	0.000017	1.274963e-07	8.191890e-14	0.260682

P-Value Tables for Table 3 (Secondary)

```
# best performing in ADULT column (Random Tree)
# get array of values (5 trials X 3 scores = 15 values)
best_adult = (adult_rforest_acc + adult_rforest_f1 + adult_rforest_roc_auc)
# best performing in GRID column (Random Tree)
# get array of values (5 trials X 3 scores = 15 values)
best_grid = (grid_rforest_acc + grid_rforest_f1 + grid_rforest_roc_auc)
# best performing in HTRU2 column (Random Tree)
# get array of values (5 trials X 3 scores = 15 values)
best_htru2 = (htru2_rforest_acc + htru2_rforest_f1 + htru2_rforest_roc_auc)
# best performing in Occupancy column (Random Tree)
# get array of values (5 trials X 3 scores = 15 values)
best_occupancy = (occupancy_rforest_acc + occupancy_rforest_f1 + occupancy_rforest_roc_auc)
# best performing in MEAN column (Random Tree)
best_mean = [adult_rforest_avg, grid_rforest_avg, htru2_rforest_avg, occupancy_rforest_avg]

# get array of values for other rows
# get decision tree arrays
dtree_adult = (adult_dtree_acc + adult_dtree_f1 + adult_dtree_roc_auc)
dtree_grid = (grid_dtree_acc + grid_dtree_f1 + grid_dtree_roc_auc)
dtree_htru2 = (htru2_dtree_acc + htru2_dtree_f1 + htru2_dtree_roc_auc)
dtree_occupancy = (occupancy_dtree_acc + occupancy_dtree_f1 + occupancy_dtree_roc_auc)
dtree_mean = [adult_dtree_avg, grid_dtree_avg, htru2_dtree_avg, occupancy_dtree_avg]
# get logistic regression arrays
logreg_adult = (adult_logreg_acc + adult_logreg_f1 + adult_logreg_roc_auc)
logreg_grid = (grid_logreg_acc + grid_logreg_f1 + grid_logreg_roc_auc)
logreg_htru2 = (htru2_logreg_acc + htru2_logreg_f1 + htru2_logreg_roc_auc)
logreg_occupancy = (occupancy_logreg_acc + occupancy_logreg_f1 + occupancy_logreg_roc_auc)
logreg_mean = [adult_logreg_avg, grid_logreg_avg, htru2_logreg_avg, occupancy_logreg_avg]
# get logistic regression arrays
knn_adult = (adult_knn_acc + adult_knn_f1 + adult_knn_roc_auc)
knn_grid = (grid_knn_acc + grid_knn_f1 + grid_knn_roc_auc)
knn_htru2 = (htru2_knn_acc + htru2_knn_f1 + htru2_knn_roc_auc)
knn_occupancy = (occupancy_knn_acc + occupancy_knn_f1 + occupancy_knn_roc_auc)
knn_mean = [adult_knn_avg, grid_knn_avg, htru2_knn_avg, occupancy_knn_avg]
```

```
# make arrays for p-values
# array for ADULT column
adult_pvalue = [None] * 4
# array for GRID column
grid_pvalue = [None] * 4
# array for HTRU2 column
htru2_pvalue = [None] * 4
# array for Occupancy column
occupancy_pvalue = [None] * 4
# array for Mean column
mean_pvalue1 = [None] * 4


# fill the pvalue array for accuracy
for i, adult, grid, htru2, occupancy, mean in zip(range(4), [best_adult, dtree_adult, logreg_adult, knn_adult
                        [best_grid, dtree_grid, logreg_grid, knn_grid], [best_htru2, dtree_htru2, logreg_htru
                        [best_occupancy, dtree_occupancy, logreg_occupancy, knn_occupancy], [best_mean, dtree
    adult_pvalue[i] = stats.ttest_rel(best_adult, adult)[1]
    grid_pvalue[i] = stats.ttest_rel(best_grid, grid)[1]
    htru2_pvalue[i] = stats.ttest_rel(best_htru2, htru2)[1]
    occupancy_pvalue[i] = stats.ttest_rel(best_occupancy, occupancy)[1]
    mean_pvalue1[i] = stats.ttest_rel(best_mean, mean)[1]
    print(mean)
    print(stats.ttest_rel(best_mean, mean))

t7 = {'MODEL': ['RF', 'DT', 'LR', 'KNN'], 'ADULT': adult_pvalue, 'GRID': grid_pvalue, 'HTRU2': htru2_pvalue,
t7 = pd.DataFrame.from_dict(t7)
t7
```

```
[0.9252933955907675, 0.9252933955907675, 0.9252933955907675, 0.9252933955907675]
Ttest_relResult(statistic=nan, pvalue=nan)
[0.9198304794440698, 0.9198304794440698, 0.9198304794440698, 0.9198304794440698]
Ttest_relResult(statistic=inf, pvalue=0.0)
[0.9197986067152081, 0.9197986067152081, 0.9197986067152081, 0.9197986067152081]
Ttest_relResult(statistic=inf, pvalue=0.0)
[0.9095053283746397, 0.9095053283746397, 0.9095053283746397, 0.9095053283746397]
Ttest_relResult(statistic=inf, pvalue=0.0)
```

	MODEL	ADULT	GRID	HTRU2	OCCUPANCY	MEAN
0	RF	NaN	NaN	NaN	NaN	NaN
1	DT	0.008321	0.008321	0.008321	0.008321	0.0
2	LR	0.014770	0.014770	0.014770	0.014770	0.0
3	KNN	0.001894	0.001894	0.001894	0.001894	0.0