Ashna Sood COGS 118A Final Project

Imports

In [1]:
```
# Imports
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

import seaborn as sns
sns.set()
sns.set_context('talk')

import scipy.stats as stats

from sklearn.metrics import roc_auc_score, f1_score, accuracy_score, make_scorer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold, GridSearchCV
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

Data Cleanup

Adult Dataset

Although I did not end up keeping the Adult dataset as one of my final datasets (due to the mulitple days it took for the algorithms to run with that dataset), I am still displaying the data cleaning I perfomed to get the dataset ready for potential analysis.

In [2]:
```
# load in adult dataset
adult_df = pd.read_csv("data/adult.data", header=None)
```

In [3]:
```
# Add column names
adult_df.columns = ["Age", "Workclass", "Final Weight", "Education", "Education Num",
                    "Marital Status", "Occupation","Relationship", "Race", "Sex",
                    "Capital Gain", "Capital Loss", "Hours Per Week",
                    "Native Country", "Income"]
```

In [4]:
```
# binarize the income col to 1 for >50K and 0 for <=50K
adult_df["Income"] = adult_df["Income"].apply(lambda x: 1 if x == " >50K" else 0)

# binarize the sex col to 1 - Female and 0 - Male
adult_df["Sex"] = adult_df["Sex"].apply(lambda x: 1 if x == " Female" else 0)
```

In [5]:
```
#Print the counts of the newly binarized columns
print("Income count:\n", adult_df['Income'].value_counts())

print("Sex count:\n", adult_df['Sex'].value_counts())
```

```
Income count:
 0    24720
1     7841
Name: Income, dtype: int64
Sex count:
 0    21790
1    10771
Name: Sex, dtype: int64
```

In [6]:
```
adult_df
```

Out[6]:

	Age	Workclass	Final Weight	Education	Education Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours Per Week	Nati Count
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	0	2174	0	40	Unite Stat
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	0	0	0	13	Unite Stat
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	0	0	0	40	Unite Stat
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	0	0	0	40	Unite Stat
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	1	0	0	40	Cu
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	1	0	0	38	Unite Stat
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	0	0	0	40	Unite Stat
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	1	0	0	40	Unite Stat
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	0	0	0	20	Unite Stat
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	1	15024	0	40	Unite Stat

32561 rows × 15 columns

In [7]:
```
# One hot encoding for nominal data: Workclass, Education, Marital Status,
# Occupation, Relationship, Race, Native Country
workclass_dummies = pd.get_dummies(adult_df["Workclass"], prefix='Work')
adult_df = pd.concat([adult_df, workclass_dummies], axis=1)

edu_dummies = pd.get_dummies(adult_df["Education"], prefix='Edu')
adult_df = pd.concat([adult_df, edu_dummies], axis=1)

married_dummies = pd.get_dummies(adult_df["Marital Status"], prefix='Mar')
adult_df = pd.concat([adult_df, married_dummies], axis=1)

occ_dummies = pd.get_dummies(adult_df["Occupation"], prefix='Occ')
adult_df = pd.concat([adult_df, occ_dummies], axis=1)

rel_dummies = pd.get_dummies(adult_df["Relationship"], prefix='Rel')
adult_df = pd.concat([adult_df, rel_dummies], axis=1)

race_dummies = pd.get_dummies(adult_df["Race"], prefix='Race')
adult_df = pd.concat([adult_df, race_dummies], axis=1)

nativeC_dummies = pd.get_dummies(adult_df["Native Country"], prefix='NativeC')
adult_df = pd.concat([adult_df, nativeC_dummies], axis=1)

adult_df
```

Out[7]:

	Age	Workclass	Final Weight	Education	Education Num	Marital Status	Occupation	Relationship	Race	Sex	...	NativeC_ Portugal	NativeC_ Puerto-Rico	Nat Sco
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	0	...	0	0	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	0	...	0	0	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	0	...	0	0	
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	0	...	0	0	

	Age	Workclass	Final Weight	Education	Education Num	Marital Status	Occupation	Relationship	Race	Sex	...	NativeC_ Portugal	NativeC_ Puerto-Rico	Nat Sco
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	1	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	1	...	0	0	
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	0	...	0	0	
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	1	...	0	0	
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	0	...	0	0	
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	1	...	0	0	

32561 rows × 115 columns

In [8]:
```
# drop the nominal columns that were one hot encoded
adult_df = adult_df.drop(columns=["Workclass", "Education", "Marital Status", "Occupation",
                                  "Relationship", "Race", "Native Country"])
adult_df
```

Out[8]:

	Age	Final Weight	Education Num	Sex	Capital Gain	Capital Loss	Hours Per Week	Income	Work_ ?	Work_ Federal-gov	...	NativeC_ Portugal	NativeC_ Puerto-Rico	NativeC_ Scotland	NativeC_ South	I
0	39	77516	13	0	2174	0	40	0	0	0	...	0	0	0	0	
1	50	83311	13	0	0	0	13	0	0	0	...	0	0	0	0	
2	38	215646	9	0	0	0	40	0	0	0	...	0	0	0	0	
3	53	234721	7	0	0	0	40	0	0	0	...	0	0	0	0	
4	28	338409	13	1	0	0	40	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	257302	12	1	0	0	38	0	0	0	...	0	0	0	0	
32557	40	154374	9	0	0	0	40	1	0	0	...	0	0	0	0	
32558	58	151910	9	1	0	0	40	0	0	0	...	0	0	0	0	
32559	22	201490	9	0	0	0	20	0	0	0	...	0	0	0	0	
32560	52	287927	9	1	15024	0	40	1	0	0	...	0	0	0	0	

32561 rows × 108 columns

In [9]:
```
# move income column to the end of the df as it is the Y (what to predict)
columns = list(adult_df.columns.values)
# remove Income from the list and add back to end of df
columns.pop(columns.index("Income"))
adult_df = adult_df[columns + ["Income"]]
adult_df
```

Out[9]:

	Age	Final Weight	Education Num	Sex	Capital Gain	Capital Loss	Hours Per Week	Work_ ?	Work_ Federal-gov	Work_ Local-gov	...	NativeC_ Puerto-Rico	NativeC_ Scotland	NativeC_ South	NativeC_ Taiwan	N T
0	39	77516	13	0	2174	0	40	0	0	0	...	0	0	0	0	
1	50	83311	13	0	0	0	13	0	0	0	...	0	0	0	0	
2	38	215646	9	0	0	0	40	0	0	0	...	0	0	0	0	
3	53	234721	7	0	0	0	40	0	0	0	...	0	0	0	0	
4	28	338409	13	1	0	0	40	0	0	0	...	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	257302	12	1	0	0	38	0	0	0	...	0	0	0	0	
32557	40	154374	9	0	0	0	40	0	0	0	...	0	0	0	0	
32558	58	151910	9	1	0	0	40	0	0	0	...	0	0	0	0	
32559	22	201490	9	0	0	0	20	0	0	0	...	0	0	0	0	
32560	52	287927	9	1	15024	0	40	0	0	0	...	0	0	0	0	

32561 rows × 108 columns

Letter Dataset

In [10]:
```
# load in letter dataset
letter_df = pd.read_csv("data/letter-recognition.data", header=None)
```

In [11]:
```
# Add column names
letter_df.columns = ["Letter", "X-box", "Y-box", "Width", "Height", "Total Pixels",
                     "X-bar","Y-bar", "X2bar", "Y2bar", "Xybar", "X2ybr", "Xy2br",
                     "X-edge", "X-edgey", "Y-edge", "Y-edgex"]
```

In [12]:
```
letter_df
```

Out[12]:

	Letter	X-box	Y-box	Width	Height	Total Pixels	X-bar	Y-bar	X2bar	Y2bar	Xybar	X2ybr	Xy2br	X-edge	X-edgey	Y-edge	Y-edgex
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19995	D	2	2	3	3	2	7	7	7	6	6	6	4	2	8	3	7
19996	C	7	10	8	8	4	4	8	6	9	12	9	13	2	9	3	7
19997	T	6	9	6	7	5	6	11	3	7	11	9	5	2	12	2	4
19998	S	2	3	4	2	1	8	7	2	6	10	6	8	1	9	5	8
19999	A	4	9	6	6	2	9	5	3	1	8	1	8	2	7	2	8

20000 rows × 17 columns

In [13]:
```
# binarize the Letter col to make letters A-M as positive - 1 and N-Z as negative - 0
A_M = list(map(chr, range(65, 78)))

letter_df["Letter"] = letter_df["Letter"].apply(lambda x: 1 if x in A_M else 0)
# check the counts of the newly binarized column
letter_df["Letter"].value_counts()
```

Out[13]:
```
0    10060
1     9940
Name: Letter, dtype: int64
```

In [14]:
```
letter_df
```

Out[14]:

	Letter	X-box	Y-box	Width	Height	Total Pixels	X-bar	Y-bar	X2bar	Y2bar	Xybar	X2ybr	Xy2br	X-edge	X-edgey	Y-edge	Y-edgex
0	0	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	1	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	1	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	0	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	1	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19995	1	2	2	3	3	2	7	7	7	6	6	6	4	2	8	3	7
19996	1	7	10	8	8	4	4	8	6	9	12	9	13	2	9	3	7
19997	0	6	9	6	7	5	6	11	3	7	11	9	5	2	12	2	4
19998	0	2	3	4	2	1	8	7	2	6	10	6	8	1	9	5	8
19999	1	4	9	6	6	2	9	5	3	1	8	1	8	2	7	2	8

20000 rows × 17 columns

Covertype Dataset

In [15]:
```
# load in Covertype dataset
covertype_df = pd.read_csv("data/covtype.data", header=None)
```

In [16]:
```
# Add / rename column names
covertype_df = covertype_df.rename(columns = {0: "Elevation", 1: "Aspect", 2: "Slope",
                    3: "Horizontal_Distance_To_Hydrology", 4: "Vertical_Distance_To_Hydrology",
                    5: "Horizontal_Distance_To_Roadways", 6: "Hillshade_9am", 7: "Hillshade_Noon",
                    8: "Hillshade_3pm", 9: "Horizontal_Distance_To_Fire_Points", 54: "Cover_Type"})
```

In [17]:
```
covertype_df
```

Out[17]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways
0	2596	51	3	258	0	510
1	2590	56	2	212	-6	390
2	2804	139	9	268	65	3180
3	2785	155	18	242	118	3090
4	2595	45	2	153	-1	391
...	...	...	...	...	...	...
581007	2396	153	20	85	17	108
581008	2391	152	19	67	12	95
581009	2386	159	17	60	7	90
581010	2384	170	15	60	5	90
581011	2383	165	13	60	4	67

581012 rows × 55 columns

In [18]:
```
# see which class has the highest frequency and make that the positive class and the rest negative
covertype_df["Cover Type"].value_counts()
```

Out[18]:
```
2    283301
1    211840
3     35754
7     20510
6     17367
5      9493
4      2747
Name: Cover Type, dtype: int64
```

In [19]:
```
# binarize the cover type based on the highest frequency class
covertype_df["Cover Type"] = covertype_df["Cover Type"].apply(lambda x: 1 if x == 2 else 0)
# check the counts of the newly binarized column
covertype_df["Cover Type"].value_counts()
```

Out[19]:
```
0    297711
1    283301
Name: Cover Type, dtype: int64
```

In [20]:
```
covertype_df
```

Out[20]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways
0	2596	51	3	258	0	510
1	2590	56	2	212	-6	390
2	2804	139	9	268	65	3180
3	2785	155	18	242	118	3090
4	2595	45	2	153	-1	391
...	...	...	...	...	...	...
581007	2396	153	20	85	17	108
581008	2391	152	19	67	12	95
581009	2386	159	17	60	7	90
581010	2384	170	15	60	5	90
581011	2383	165	13	60	4	67

581012 rows × 55 columns

California Housing Dataset

In [21]:
```
# load in CalHousing dataset
calhousing_df = pd.read_csv("data/cal_housing.data", header=None)
```

In [22]:
```
# Add column names
calhousing_df.columns = ["Longitude", "Lattitude", "Housing Median Age",
                         "Total Rooms", "Total Bedrooms", "Population",
                         "Households", "Median Income", "Median House Val"]
```

In [23]:
```
# binarize output variable so that median housing value > $130,000 is 1 and <= $130,000 is 0
calhousing_df["Median House Val"] = calhousing_df["Median House Val"].apply(lambda x: 1
                                                                             if x > 130000 else 0)
# check the counts of the newly binarized column
calhousing_df["Median House Val"].value_counts()
```

Out[23]:
```
1    14728
0     5912
Name: Median House Val, dtype: int64
```

In [24]:
```
calhousing_df
```

Out[24]:

	Longitude	Lattitude	Housing Median Age	Total Rooms	Total Bedrooms	Population	Households	Median Income	Median House Val
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	1
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	1
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	1
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	1
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	1
...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	0
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	0
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	0
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	0
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	0

20640 rows × 9 columns

Dry Beans Dataset

In [25]:
```
# load in the Dry Beans dataset
beans_df = pd.read_csv("data/Dry_Beans_Dataset.csv", header=0)
```

In [26]:
```
# see which 2 classes have the highest frequency to make those the positive class
beans_df["Class"].value_counts()
```

Out[26]:
```
DERMASON    3546
SIRA        2636
SEKER       2027
HOROZ       1928
CALI        1630
BARBUNYA    1322
BOMBAY       522
Name: Class, dtype: int64
```

In [27]:
```
# Binarize the classes - make the two highest classes the postive class & the rest negative
beans_df["Class"] = beans_df["Class"].apply(lambda x: 1 if x == "DERMASON" or x == "SIRA" else 0)
# check the counts of the newly binarized column
beans_df["Class"].value_counts()
```

Out[27]:
```
0    7429
1    6182
Name: Class, dtype: int64
```

In [28]:
```
beans_df
```

Out[28]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidit
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.98885
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272751	0.783968	0.98498
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.98955
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.97669
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.99089
...	...	...	...	...	...	...	...	...	...	.
13606	42097	759.696	288.721612	185.944705	1.552728	0.765002	42508	231.515799	0.714574	0.99033
13607	42101	757.499	281.576392	190.713136	1.476439	0.735702	42494	231.526798	0.799943	0.99075
13608	42139	759.321	281.539928	191.187979	1.472582	0.734065	42569	231.631261	0.729932	0.98989
13609	42147	763.779	283.382636	190.275731	1.489326	0.741055	42667	231.653248	0.705389	0.98781
13610	42159	772.237	295.142741	182.204716	1.619841	0.786693	42600	231.686223	0.788962	0.98964

13611 rows × 17 columns

Training

The four algorithms I will be training are Logistic Regression, Support Vector Machines (SVM), K Nearest Neighbors (KNN), and Random Forests (RF).

X & y splits for all datasets

In [29]:
```
# define the X and y for all datasets

# was originally using adult instead of beans, but now only using beans
#X_adult = adult_df.iloc[:, :-1]
#y_adult = adult_df.iloc[:, -1:]

X_beans  = beans_df.iloc[:, :-1]
y_beans = beans_df.iloc[:, -1:]

X_letter = letter_df.iloc[:, 1:]
y_letter = letter_df.iloc[:, 0]

X_covertype = covertype_df.iloc[:, :-1]
y_covertype = covertype_df.iloc[:, -1:]

X_calhousing = calhousing_df.iloc[:, :-1]
y_calhousing = calhousing_df.iloc[:, -1:]

# create an array with all the X and y splits to pass through algorithm loops
X_total = [X_beans, X_letter, X_covertype, X_calhousing]
y_total = [y_beans, y_letter, y_covertype, y_calhousing]
```

Logistic Regression Model

In [30]:
```
# Logistic Regression model

# store metrics of all 4 datasets
lr_metrics_train = []
lr_metrics_test = []
for X, y in zip(X_total, y_total):
    # create 3 lists for the AUC, Accuracy, and F1 Scores across the 5 trials for train & test
    lr_AUC_train, lr_Acc_train, lr_F1_train = [], [], []
    lr_AUC_test, lr_Acc_test, lr_F1_test = [], [], []
    for trial in range(5):
        # for each trial, randomly select 5000 samples for the training set & rest as test
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=5000)
        C_vals = [10**-8, 10**-7, 10**-6, 10**-5, 10**-4, 10**-3, 10**-2, 10**-1,
                  10**0, 10**1, 10**2, 10**3, 10**4]
        # metrics to evaluate model on
        scoring = {"Accuracy": make_scorer(accuracy_score), "F1_Score": "f1", "AUC": "roc_auc"}
        # Create a pipeline
        pipe = Pipeline([('std', StandardScaler()),
                         ('classifier', LogisticRegression(max_iter = 2000))])
        # Create search space of candidate learning algorithms and their hyperparameters
        search_space = [{'classifier': [LogisticRegression(max_iter = 2000)],
                         'classifier__C': C_vals, 'classifier__penalty': ["l2"]},
                        {'classifier': [LogisticRegression()], 'classifier__penalty': ["none"]}]
        # Grid Search with stratified 5 folds cross validation to find best hyperparameters
        search_results = GridSearchCV(pipe, search_space, scoring = scoring, refit = False,
                                      cv=StratifiedKFold(n_splits = 5))
        # train models
        search_results.fit(X_train, y_train)

        # find optimal hyperparameters for each metric
        AUC_rank = search_results.cv_results_["rank_test_AUC"]
        # find index of #1 in the rank array to find best hyperparameter
        AUC_ind = np.argmin(AUC_rank)
        # save best hyperparameters
        opt_AUC = search_results.cv_results_["params"][AUC_ind]

        Acc_rank = search_results.cv_results_["rank_test_Accuracy"]
        # find index of #1 in the array to find best hyperparameter
        Acc_ind = np.argmin(Acc_rank)
        # save best hyperparameters
        opt_Acc = search_results.cv_results_["params"][Acc_ind]
```

```
        F1_rank = search_results.cv_results_["rank_test_F1_Score"]
        # find index of #1 in the array to find best hyperparameter
        F1_ind = np.argmin(F1_rank)
        # save best hyperparameters
        opt_F1 = search_results.cv_results_["params"][F1_ind]

        # determine the optimal parameters for the three models
        opt_models = [opt_AUC, opt_Acc, opt_F1]
        # initialize models -- will now fill in with best parameters
        lr_AUC_model =  LogisticRegression(max_iter = 2000)
        lr_Acc_model = LogisticRegression(max_iter = 2000)
        lr_F1_model =  LogisticRegression(max_iter = 2000)
        models = [lr_AUC_model, lr_Acc_model, lr_F1_model]
        # create 3 optimal models each with best parameters for that metric
        for opt, model in zip(opt_models, models):
            if opt["classifier__penalty"] == "none":
                model = LogisticRegression(max_iter = 2000, penalty = opt["classifier__penalty"])
            else:
                # l2 regularization
                model = LogisticRegression(max_iter = 2000, C = opt["classifier__C"],
                                           penalty = opt["classifier__penalty"])

        # train 3 models with the optimal parameters -- one model for each metric
        # AUC model
        lr_AUC_model.fit(X_train, y_train)
        # make predictions & calculate AUC on both training and testing sets
        lr_AUC_pred_train = (lr_AUC_model.predict_proba(X_train)[:,1])
        lr_AUC_score_train = roc_auc_score(y_train, lr_AUC_pred_train)
        lr_AUC_pred_test = (lr_AUC_model.predict_proba(X_test)[:,1])
        lr_AUC_score_test = roc_auc_score(y_test, lr_AUC_pred_test)
        # add AUC for current trial
        lr_AUC_train.append(lr_AUC_score_train)
        lr_AUC_test.append(lr_AUC_score_test)

        # Accuracy model
        lr_Acc_model.fit(X_train, y_train)
        # calculate accuracy on both training and testing sets
        lr_Acc_score_train = lr_Acc_model.score(X_train, y_train)
        lr_Acc_score_test = lr_Acc_model.score(X_test, y_test)
        # add Accuracy for current trial
        lr_Acc_train.append(lr_Acc_score_train)
        lr_Acc_test.append(lr_Acc_score_test)
```

```
        # F1 score model
        lr_F1_model.fit(X_train, y_train)
        # make predictions & calculate F1 score on both training and testing sets
        lr_F1_pred_train = (lr_F1_model.predict(X_train))
        lr_F1_score_train = f1_score(y_train, lr_F1_pred_train)
        lr_F1_pred_test = (lr_F1_model.predict(X_test))
        lr_F1_score_test = f1_score(y_test, lr_F1_pred_test)
        # add F1 score for current trial
        lr_F1_train.append(lr_F1_score_train)
        lr_F1_test.append(lr_F1_score_test)

    # average train & test AUC, Accuracy, and F1 score across all 5 trials
    lr_AUC_train_m, lr_AUC_test_m = np.mean(lr_AUC_train), np.mean(lr_AUC_test)
    lr_Acc_train_m, lr_Acc_test_m = np.mean(lr_Acc_train), np.mean(lr_Acc_test)
    lr_F1_train_m, lr_F1_test_m = np.mean(lr_F1_train), np.mean(lr_F1_test)
    # combine average training metrics into one array for current trial
    lr_trial_metrics_train = [lr_AUC_train_m, lr_Acc_train_m, lr_F1_train_m]
    # combine average testing metrics into one array for current trial
    lr_trial_metrics_test = [lr_AUC_test_m, lr_Acc_test_m, lr_F1_test_m]
    # add average train and test metrics for current trial
    lr_metrics_train.append(lr_trial_metrics_train)
    lr_metrics_test.append(lr_trial_metrics_test)

    # print raw test metric values for current trial
    print("Raw test values")
    print("AUC:", lr_AUC_test)
    print("Acc:", lr_Acc_test)
    print("F1:", lr_F1_test)
    # print average metrics for train and test for current trial
    print("train trial metrics:", np.round(lr_trial_metrics_train, 3))
    print("test trial metrics:", np.round(lr_trial_metrics_test, 3))

# final metrics from all 4 datasets
lr_metrics_train = np.array(lr_metrics_train)
print("Logistic Regression metrics train:\n", lr_metrics_train)
lr_metrics_train_m = np.mean(lr_metrics_train, axis=0)
print("Average Logistic Regression train metrics:\n", lr_metrics_train_m)
lr_metrics_test = np.array(lr_metrics_test)
print("Logistic Regression metrics test:\n", lr_metrics_test)
lr_metrics_test_m = np.mean(lr_metrics_test, axis=0)
print("Average Logistic Regression test metrics:\n", lr_metrics_test_m)
# calculate average across metrics
lr_metrics_test_m_2 = np.mean(lr_metrics_test, axis=1)
```

```
print("Average across Logistic Regression test metrics:\n", lr_metrics_test_m_2)
```

```
Raw test values
AUC: [0.9871307469647064, 0.9857381429308004, 0.9869650302790988, 0.8966795174688855, 0.98631077232
37451]
Acc: [0.9466960864011148, 0.9440250841946348, 0.9499477412611775, 0.7841133433979793, 0.94843804436
18627]
F1: [0.94211123723042, 0.9399451781709445, 0.9455464308275426, 0.7664866222836327, 0.94356888662938
49]
train trial metrics: [0.971 0.918 0.912]
test trial metrics: [0.969 0.915 0.908]
Raw test values
AUC: [0.8124646253359503, 0.8090583525898403, 0.8127006535506116, 0.8108140997392252, 0.81012193627
48475]
Acc: [0.7278, 0.7242666666666666, 0.7278666666666667, 0.7229333333333333, 0.7224666666666667]
F1: [0.7292619852794907, 0.7271767810026385, 0.7315533342101802, 0.7231179213857429, 0.723774135757
4149]
train trial metrics: [0.818 0.732 0.735]
test trial metrics: [0.811 0.725 0.727]
Raw test values
AUC: [0.7990187939641502, 0.7998214888174716, 0.7914906556858601, 0.7931409413303765, 0.79887721095
64317]
Acc: [0.7307556092581404, 0.7262140372075582, 0.7167142351201017, 0.71641215808004, 0.7310993520968
313]
F1: [0.7252286024473145, 0.7268721856598546, 0.7243508961602785, 0.7250278267400594, 0.734808275748
7141]
train trial metrics: [0.801 0.73  0.736]
test trial metrics: [0.796 0.724 0.727]
Raw test values
AUC: [0.8908112553124239, 0.8915175930655257, 0.8879267443196939, 0.8899060754110683, 0.88907442649
11231]
Acc: [0.8307544757033248, 0.8338874680306906, 0.8294117647058824, 0.8312659846547314, 0.82979539641
94373]
F1: [0.8850380021715526, 0.8872108666406283, 0.8831259856316804, 0.8846002363135094, 0.884771881222
405]
train trial metrics: [0.893 0.83  0.884]
test trial metrics: [0.89  0.831 0.885]
Logistic Regression metrics train:
 [[0.9707294  0.91772    0.91199135]
 [0.81849992 0.73228    0.73480448]
 [0.80135044 0.73048    0.73638799]
 [0.89311518 0.83032    0.88398199]]
Average Logistic Regression train metrics:
 [0.87092374 0.8027     0.81679145]
```

```
Logistic Regression metrics test:
 [[0.96856484 0.91464406 0.90753167]
 [0.81103193 0.72506667 0.72697683]
 [0.79646982 0.72423908 0.72725756]
 [0.88984722 0.83102302 0.88494939]]
Average Logistic Regression test metrics:
 [0.86647845 0.79874321 0.81167886]
Average across Logistic Regression test metrics:
 [0.93024686 0.75435848 0.74932215 0.86860654]
```

These are the full arrays for the lr values in both tables 2 and 3, and will be used to calculate the t tests and p values.

In [ ]:

```
# table 2 lr arrays (cols of Lr metrics test)
lr_auc = [0.96856484, 0.81103193, 0.79646982, 0.88984722]
lr_acc = [0.91464406, 0.72506667, 0.72423908, 0.83102302]
lr_f1 = [0.90753167, 0.72697683, 0.72725756, 0.88494939]
lr_mean_1 = [0.86647845, 0.79874321, 0.81167886]

# table 3 lr arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
lr_d1 = [0.9586460235320803, 0.9565694684321265, 0.9608197341226062,
         0.8157598277168324, 0.959439234438331]
lr_d2 = [0.7565088702051469, 0.7535006000863818, 0.7573735514758195,
         0.7522884514861006, 0.752120912899643]
lr_d3 = [0.751667668556535, 0.7509692372282948, 0.74418526232208,
         0.7448603087168252, 0.7549282796006591]
lr_d4 = [0.8688679110624338, 0.8708719759122815, 0.8668214982190855,
         0.8685907654597697, 0.8678805680443218]
lr_mean_2 = [0.93024686, 0.75435848, 0.74932215, 0.86860654]
```

SVM Model

In [31]:
```
# SVM model
# store metrics of all 4 datasets
svm_metrics_train = []
svm_metrics_test = []
for X, y in zip(X_total, y_total):
    # create 3 lists for the AUC, Accuracy, and F1 Scores across the 5 trials
    svm_AUC_train, svm_Acc_train, svm_F1_train = [], [], []
    svm_AUC_test, svm_Acc_test, svm_F1_test = [], [], []
    for trial in range(5):
        print("trial:", trial)
        # for each trial, randomly select 5000 samples for the training set & rest as test
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=5000)
        # hyperparameters
        C_vals = [10**-7, 10**-6, 10**-5, 10**-4, 10**-3, 10**-2, 10**-1,
                  10**0, 10**1, 10**2, 10**3]
        degree = [2, 3]
        gamma = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 2]
        # metrics to evaluate model on
        scoring = {"Accuracy": make_scorer(accuracy_score), "F1_Score": "f1", "AUC": "roc_auc"}
        # Create a pipeline
        pipe = Pipeline([('std', StandardScaler()), ('classifier', SVC(probability=True))])

        # Create search space of candidate learning algorithms and their hyperparameters
        search_space = [{'classifier': [SVC(probability=True)], 'classifier__C': C_vals,
                        'classifier__kernel': ["linear"]},
                        {'classifier': [SVC(probability=True)], 'classifier__C': C_vals,
                         'classifier__kernel': ["poly"],'classifier__degree': degree},
                        {'classifier': [SVC(probability=True)], 'classifier__C': C_vals,
                        'classifier__kernel': ["rbf"], 'classifier__gamma': gamma}]
        # Grid Search with stratified 5 folds cross validation to find best hyperparmeters
        search_results = GridSearchCV(pipe, search_space, scoring = scoring, refit = False,
                                      cv=StratifiedKFold(n_splits = 5))
        # train models
        search_results.fit(X_train, y_train)

        # find optimal hyperparameters for each metric
        AUC_rank = search_results.cv_results_["rank_test_AUC"]
        # find index of #1 in the rank array to find best hyperparameter
        AUC_ind = np.argmin(AUC_rank)
        # save best hyperparameters
        opt_AUC = search_results.cv_results_["params"][AUC_ind]
```

```
        Acc_rank = search_results.cv_results_["rank_test_Accuracy"]
        # find index of #1 in the rank array to find best hyperparameter
        Acc_ind = np.argmin(Acc_rank)
        # save best hyperparameters
        opt_Acc = search_results.cv_results_["params"][Acc_ind]

        F1_rank = search_results.cv_results_["rank_test_F1_Score"]
        # find index of #1 in the rank array to find best hyperparameter
        F1_ind = np.argmin(F1_rank)
        # save best hyperparameters
        opt_F1 = search_results.cv_results_["params"][F1_ind]

        # determine the optimal parameters for the three models
        opt_models = [opt_AUC, opt_Acc, opt_F1]
        # initialize models -- will now fill in with best parameters
        svm_AUC_model = SVC(probability=True)
        svm_Acc_model = SVC(probability=True)
        svm_F1_model, = SVC(probability=True)
        models = [svm_AUC_model, svm_Acc_model, svm_F1_model]
        # create 3 optimal models each with best parameters for that metric
        for opt, model in zip(opt_models, models):
            if opt["classifier__kernel"] == "poly":
                model = SVC(kernel = opt["classifier__kernel"], C = opt["classifier__C"],
                            degree = opt["classifier__degree"], probability=True)
            elif opt["classifier__kernel"] == "gamma":
                model = SVC(kernel = opt["classifier__kernel"], C = opt["classifier__C"],
                            gamma = opt["classifier__gamma"], probability=True)
            else:
                model = SVC(kernel = opt["classifier__kernel"], C = opt["classifier__C"],
                            probability=True)

        # train 3 models with the optimal parameters -- one model for each metric
        # AUC model
        svm_AUC_model.fit(X_train, y_train)
        # make predictions & calculate AUC on both training and testing sets
        svm_AUC_pred_train = (svm_AUC_model.predict_proba(X_train)[:,1])
        svm_AUC_score_train = roc_auc_score(y_train, svm_AUC_pred_train)
        svm_AUC_pred_test = (svm_AUC_model.predict_proba(X_test)[:,1])
        svm_AUC_score_test = roc_auc_score(y_test, svm_AUC_pred_test)
        # add AUC for current trial
        svm_AUC_train.append(svm_AUC_score_train)
        svm_AUC_test.append(svm_AUC_score_test)
```

```
        # Accuracy model
        svm_Acc_model.fit(X_train, y_train)
        # calculate accuracy on both training and testing sets
        svm_Acc_score_train = svm_Acc_model.score(X_train, y_train)
        svm_Acc_score_test = svm_Acc_model.score(X_test, y_test)
        # add Accuracy for current trial
        svm_Acc_train.append(svm_Acc_score_train)
        svm_Acc_test.append(svm_Acc_score_test)

        # F1 score model
        svm_F1_model.fit(X_train, y_train)
        # make predictions & calculate F1 score on both training and testing sets
        svm_F1_pred_train = (svm_F1_model.predict(X_train))
        svm_F1_score_train = f1_score(y_train, svm_F1_pred_train)
        svm_F1_pred_test = (svm_F1_model.predict(X_test))
        svm_F1_score_test = f1_score(y_test, svm_F1_pred_test)
        # add F1 score for current trial
        svm_F1_train.append(svm_F1_score_train)
        svm_F1_test.append(svm_F1_score_test)

    # average AUC, Accuracy, and F1 score across all 5 trials
    svm_AUC_train_m, svm_AUC_test_m = np.mean(svm_AUC_train), np.mean(svm_AUC_test)
    svm_Acc_train_m, svm_Acc_test_m = np.mean(svm_Acc_train), np.mean(svm_Acc_test)
    svm_F1_train_m, svm_F1_test_m = np.mean(svm_F1_train), np.mean(svm_F1_test)
    # combine average training metrics into one array for current trial
    svm_trial_metrics_train = [svm_AUC_train_m, svm_Acc_train_m, svm_F1_train_m]
    # combine average testing metrics into one array for current trial
    svm_trial_metrics_test = [svm_AUC_test_m, svm_Acc_test_m, svm_F1_test_m]
    # add average train and test metrics for current trial
    svm_metrics_train.append(svm_trial_metrics_train)
    svm_metrics_test.append(svm_trial_metrics_test)

    # print raw test metric values for current trial
    print("Raw test values")
    print("AUC:", svm_AUC_test)
    print("Acc:", svm_Acc_test)
    print("F1:", svm_F1_test)
    # print average metrics for train and test for current trial
    print("train trial metrics:", np.round(svm_trial_metrics_train, 3))
    print("test trial metrics:", np.round(svm_trial_metrics_test, 3))

# final metrics from all 4 datasets
svm_metrics_train = np.array(svm_metrics_train)
```

```
print("SVM metrics train:\n", svm_metrics_train)
svm_metrics_train_m = np.mean(svm_metrics_train, axis=0)
print("Average SVM train metrics:\n", svm_metrics_train_m)
svm_metrics_test = np.array(svm_metrics_test)
print("SVM metrics test:\n", svm_metrics_test)
svm_metrics_test_m = np.mean(svm_metrics_test, axis=0)
print("Average SVM test metrics:\n", svm_metrics_test_m)
# calculate average across metrics
svm_metrics_test_m_2 = np.mean(svm_metrics_test, axis=1)
print("Average across SVM test metrics:\n", svm_metrics_test_m_2)
```

```
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.8604035759252292, 0.8639090242455969, 0.8660821741915149, 0.8611989958814702, 0.86383309918
27154]
Acc: [0.7823713854372315, 0.7813262106607827, 0.784693996051562, 0.7799326442921845, 0.785855301358
7272]
F1: [0.7924695459579181, 0.7892085525579312, 0.7922456297624383, 0.7893274041133963, 0.795474711623
7799]
train trial metrics: [0.867 0.788 0.797]
test trial metrics: [0.863 0.783 0.792]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9524253553565456, 0.9531144230829234, 0.9511932994979699, 0.9516859665068053, 0.95125047666
31635]
Acc: [0.8795333333333333, 0.8789333333333333, 0.8767333333333334, 0.8803333333333333, 0.88126666666
66666]
F1: [0.8786678305244076, 0.8792392605399655, 0.8755301245371928, 0.8803253550236683, 0.879098499762
4057]
train trial metrics: [0.96  0.89  0.888]
test trial metrics: [0.952 0.879 0.879]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
```

```
Raw test values
AUC: [0.7562672149943853, 0.7579740737064173, 0.751593227687614, 0.7517994243494166, 0.755366628669
9963]
Acc: [0.6935549953820407, 0.6978847662895912, 0.6916123275209545, 0.6946435143712284, 0.69574939410
98449]
F1: [0.6768857908533104, 0.6939217734814987, 0.6760264890999651, 0.6835179743883634, 0.681494351473
5602]
train trial metrics: [0.763 0.703 0.688]
test trial metrics: [0.755 0.695 0.682]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.8253671841943817, 0.7514659022459358, 0.8064260426450938, 0.7844186635179224, 0.78675154898
54504]
Acc: [0.7119565217391305, 0.7125319693094629, 0.7118925831202046, 0.7140664961636829, 0.71624040920
71611]
F1: [0.8317460317460318, 0.8321135175504107, 0.8317023978486591, 0.8331841241420471, 0.834662096714
1049]
train trial metrics: [0.815 0.714 0.833]
test trial metrics: [0.791 0.713 0.833]
SVM metrics train:
 [[0.8669318  0.78764    0.79741423]
 [0.9600166  0.88956    0.88776708]
 [0.76303965 0.70272    0.68767883]
 [0.81530286 0.714      0.83306661]]
Average SVM train metrics:
 [0.85132273 0.77348    0.80148168]
SVM metrics test:
 [[0.86308537 0.78283591 0.79174517]
 [0.9519339  0.87936    0.87857221]
 [0.75460011 0.694689   0.68236928]
 [0.79088587 0.7133376  0.83268163]]
Average SVM test metrics:
 [0.84012632 0.76755563 0.79634207]
Average across SVM test metrics:
 [0.81255548 0.90328871 0.7105528  0.77896837]
```

These are the full arrays for the svm values in both tables 2 and 3, and will be used to calculate the t tests and p values.

In [ ]:
```
# table 2 svm arrays (cols of svm metrics test)
svm_auc = [0.86308537, 0.9519339, 0.75460011, 0.79088587]
svm_acc = [0.78283591, 0.87936, 0.694689, 0.7133376]
svm_f1 = [0.79174517, 0.87857221, 0.68236928, 0.83268163]
svm_mean_1 = [0.84012632, 0.76755563, 0.79634207]

# table 3 svm arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
svm_d1 = [0.8117481691067928, 0.8114812624881037, 0.8143406000018384,
          0.8101530147623502, 0.8150543707217408]
svm_d2 = [0.9035421730714287, 0.9037623389854074, 0.9011522524561654,
          0.9041148849546022, 0.9038718810307453]
svm_d3 = [0.7089026670765787, 0.7165935378258358, 0.7064106814361778,
          0.7099869710363361, 0.7108701247511338]
svm_d4 = [0.7896899125598481, 0.7653704630352699, 0.7833403412046526,
          0.7772230946078841, 0.7792180183022389]
svm_mean_2 = [0.81255548, 0.90328871, 0.7105528, 0.77896837]
```

KNN Model

In [32]:
```
# KNN model
# store metrics of all 4 datasets
knn_metrics_train = []
knn_metrics_test = []
for X, y in zip(X_total, y_total):
    # create 3 lists for the AUC, Accuracy, and F1 Scores across the 5 trials
    knn_AUC_train, knn_Acc_train, knn_F1_train = [], [], []
    knn_AUC_test, knn_Acc_test, knn_F1_test = [], [], []
    for trial in range(5):
        print("trial:", trial)
        # for each trial, randomly select 5000 samples for the training set & rest as test
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=5000)
        # hyperparameters
        n_neighbors = [1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57,
                       61, 65, 69, 73, 77, 81, 85, 89, 93, 97, 101]
        weights = ["uniform", "distance"]
        algorithms = ["ball_tree", "kd_tree", "brute"]
        # metrics to evaluate model on
        scoring = {"Accuracy": make_scorer(accuracy_score), "F1_Score": "f1", "AUC": "roc_auc"}

        # Create a pipeline
        pipe = Pipeline([('std', StandardScaler()),
                         ('classifier', KNeighborsClassifier(metric="euclidean"))])
        # Create search space of candidate learning algorithms and their hyperparameters
        search_space = [{'classifier': [KNeighborsClassifier(metric="euclidean")],
                         'classifier__n_neighbors': n_neighbors, 'classifier__weights': weights,
                         'classifier__algorithm': algorithms}]
        # Grid Search with stratified 5 folds cross validation to find best hyperparmeters
        search_results = GridSearchCV(pipe, search_space, scoring = scoring, refit = False,
                                      cv=StratifiedKFold(n_splits = 5))
        # train models
        search_results.fit(X_train, y_train)

        # find optimal hyperparameters for each metric
        AUC_rank = search_results.cv_results_["rank_test_AUC"]
        # find index of #1 in the rank array to find best hyperparameter
        AUC_ind = np.argmin(AUC_rank)
        # save best hyperparameters
        opt_AUC = search_results.cv_results_["params"][AUC_ind]

        Acc_rank = search_results.cv_results_["rank_test_Accuracy"]
        # find index of #1 in the rank array to find best hyperparameter
```

```
        Acc_ind = np.argmin(Acc_rank)
        # save best hyperparameters
        opt_Acc = search_results.cv_results_["params"][Acc_ind]

        F1_rank = search_results.cv_results_["rank_test_F1_Score"]
        # find index of #1 in the rank array to find best hyperparameter
        F1_ind = np.argmin(F1_rank)
        # save best hyperparameters
        opt_F1 = search_results.cv_results_["params"][F1_ind]

        # train 3 models with the optimal parameters -- one model for each metric
        # AUC model
        knn_AUC_model = KNeighborsClassifier(metric="euclidean",
                                             n_neighbors = opt_AUC["classifier__n_neighbors"],
                                             weights = opt_AUC["classifier__weights"],
                                             algorithm = opt_AUC["classifier__algorithm"])
        knn_AUC_model.fit(X_train, y_train)
        # make predictions & calculate AUC on both training and testing sets
        knn_AUC_pred_train = (knn_AUC_model.predict_proba(X_train)[:,1])
        knn_AUC_score_train = roc_auc_score(y_train, knn_AUC_pred_train)
        knn_AUC_pred_test = (knn_AUC_model.predict_proba(X_test)[:,1])
        knn_AUC_score_test = roc_auc_score(y_test, knn_AUC_pred_test)
        # add AUC for current trial
        knn_AUC_train.append(knn_AUC_score_train)
        knn_AUC_test.append(knn_AUC_score_test)

        # Accuracy model
        knn_Acc_model = KNeighborsClassifier(metric="euclidean",
                                             n_neighbors = opt_Acc["classifier__n_neighbors"],
                                             weights = opt_Acc["classifier__weights"],
                                             algorithm = opt_Acc["classifier__algorithm"])
        knn_Acc_model.fit(X_train, y_train)
        # calculate accuracy on both training and testing sets
        knn_Acc_score_train = knn_Acc_model.score(X_train, y_train)
        knn_Acc_score_test = knn_Acc_model.score(X_test, y_test)
        # add Accuracy for current trial
        knn_Acc_train.append(knn_Acc_score_train)
        knn_Acc_test.append(knn_Acc_score_test)

        # F1 Score model
        knn_F1_model = KNeighborsClassifier(metric="euclidean",
                                            n_neighbors = opt_F1["classifier__n_neighbors"],
                                            weights = opt_F1["classifier__weights"],
```

```
                                                algorithm = opt_F1["classifier__algorithm"])
        knn_F1_model.fit(X_train, y_train)
        # make predictions & calculate F1 score on both training and testing sets
        knn_F1_pred_train = (knn_F1_model.predict(X_train))
        knn_F1_score_train = f1_score(y_train, knn_F1_pred_train)
        knn_F1_pred_test = (knn_F1_model.predict(X_test))
        knn_F1_score_test = f1_score(y_test, knn_F1_pred_test)
        # add F1 score for current trial
        knn_F1_train.append(knn_F1_score_train)
        knn_F1_test.append(knn_F1_score_test)

    # average AUC, Accuracy, and F1 score across all 5 trials
    knn_AUC_train_m, knn_AUC_test_m = np.mean(knn_AUC_train), np.mean(knn_AUC_test)
    knn_Acc_train_m, knn_Acc_test_m = np.mean(knn_Acc_train), np.mean(knn_Acc_test)
    knn_F1_train_m, knn_F1_test_m = np.mean(knn_F1_train), np.mean(knn_F1_test)
    # combine average training metrics into one array for current trial
    knn_trial_metrics_train = [knn_AUC_train_m, knn_Acc_train_m, knn_F1_train_m]
    # combine average testing metrics into one array for current trial
    knn_trial_metrics_test = [knn_AUC_test_m, knn_Acc_test_m, knn_F1_test_m]
    # add average train and test metrics for current trial
    knn_metrics_train.append(knn_trial_metrics_train)
    knn_metrics_test.append(knn_trial_metrics_test)

    # print raw test metric values for current trial
    print("Raw test values")
    print("AUC:", knn_AUC_test)
    print("Acc:", knn_Acc_test)
    print("F1:", knn_F1_test)
    # print average metrics for train and test for current trial
    print("train trial metrics:", knn_trial_metrics_train)
    print("test trial metrics:", np.round(knn_trial_metrics_test, 3))

# final metrics from all 4 datasets
knn_metrics_train = np.array(knn_metrics_train)
print("KNN metrics train:\n", knn_metrics_train)
knn_metrics_train_m = np.mean(knn_metrics_train, axis=0)
print("Average KNN train metrics:\n", knn_metrics_train_m)
knn_metrics_test = np.array(knn_metrics_test)
print("KNN metrics test:\n", knn_metrics_test)
knn_metrics_test_m = np.mean(knn_metrics_test, axis=0)
print("Average KNN test metrics:\n", knn_metrics_test_m)
# calculate average across metrics
knn_metrics_test_m_2 = np.mean(knn_metrics_test, axis=1)
```

```
print("Average across KNN test metrics:\n", knn_metrics_test_m_2)
```

```
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9186584716544333, 0.9107685411788818, 0.9117882994902321, 0.9232789366433118, 0.91641949988
38217]
Acc: [0.8418302171640925, 0.7989780513296946, 0.8409011729183602, 0.8324236441760539, 0.84949483219
13831]
F1: [0.8347087378640777, 0.7993508751593833, 0.8375622480436329, 0.8277837450769783, 0.842374118219
4114]
train trial metrics: [1.0, 0.91544, 0.9115126875396428]
test trial metrics: [0.916 0.833 0.828]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9912599000264377, 0.9901761003301686, 0.9903323285507352, 0.9904655045553582, 0.99079132473
58082]
Acc: [0.959, 0.9572666666666667, 0.9574666666666667, 0.9575333333333333, 0.9560666666666666]
F1: [0.9590355025644441, 0.9572238905572238, 0.9571697099892588, 0.9576490924805532, 0.955762905282
9428]
train trial metrics: [1.0, 1.0, 1.0]
test trial metrics: [0.991 0.957 0.957]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.8588672610437202, 0.8585651485056803, 0.8612744085839801, 0.8605461430320636, 0.86119680693
27682]
Acc: [0.7768154135677728, 0.779607022075929, 0.7820392630709082, 0.7792702235370097, 0.785240585265
5847]
F1: [0.7787280697979831, 0.7772639879286427, 0.7762673927919708, 0.7754759796034943, 0.783827990661
3915]
train trial metrics: [1.0, 1.0, 1.0]
test trial metrics: [0.86  0.781 0.778]
trial: 0
```

```
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.6697572508645547, 0.6661878008514497, 0.6680496603664368, 0.6648510068387052, 0.65436213917
69073]
Acc: [0.6968030690537085, 0.6914322250639386, 0.6792838874680307, 0.6925191815856777, 0.69565217391
30435]
F1: [0.8127480457005413, 0.8023751023751022, 0.8121778350515463, 0.8038343871099327, 0.811225925777
0095]
train trial metrics: [1.0, 0.9513999999999999, 0.9688661114670083]
test trial metrics: [0.665 0.691 0.808]
KNN metrics train:
 [[1.         0.91544    0.91151269]
 [1.         1.         1.        ]
 [1.         1.         1.        ]
 [1.         0.9514     0.96886611]]
Average KNN train metrics:
 [1.        0.96671   0.9700947]
KNN metrics test:
 [[0.91618275 0.83272558 0.82835594]
 [0.99060503 0.95746667 0.95736822]
 [0.86008995 0.7805945  0.77831268]
 [0.66464157 0.69113811 0.80847226]]
Average KNN test metrics:
 [0.85787983 0.81548121 0.84312728]
Average across KNN test metrics:
 [0.85908809 0.96847997 0.80633238 0.72141731]
```

These are the full arrays for the knn values in both tables 2 and 3, and will be used to calculate the t tests and p values.

In [ ]:
```
# table 2 knn arrays (cols of knn metrics test)
knn_auc = [0.91618275, 0.99060503, 0.86008995, 0.66464157]
knn_acc = [0.83272558, 0.95746667, 0.7805945, 0.69113811]
knn_f1 = [0.82835594, 0.95736822, 0.77831268, 0.80847226]
knn_mean_1 = [0.85787983, 0.81548121, 0.84312728]

# table 3 knn arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
knn_d1 = [0.8650658088942013, 0.8363658225559866, 0.8634172401507417,
          0.8611621086321147, 0.8694294834315387]
knn_d2 = [0.9697651341969605, 0.9682222191846863, 0.9683229017355536,
          0.9685493101230817, 0.9675402988951393]
knn_d3 = [0.8048035814698253, 0.8051453861700839, 0.8065270214822863,
          0.8050974487241892, 0.8100884609532483]
knn_d4 = [0.7264361218729348, 0.7199983760968302, 0.7198371276286712,
          0.7204015251781053, 0.7204134129556534]
knn_mean_2 = [0.85908809, 0.96847997, 0.80633238, 0.72141731]
```

Random Forests Model

In [33]:
```
# Random Forests model
# store metrics of all 4 datasets
rf_metrics_train = []
rf_metrics_test = []
for X, y in zip(X_total, y_total):
    # create 3 lists for the AUC, Accuracy, and F1 Scores across the 5 trials
    rf_AUC_train, rf_Acc_train, rf_F1_train = [], [], []
    rf_AUC_test, rf_Acc_test, rf_F1_test = [], [], []
    for trial in range(5):
        print("trial:", trial)
        # for each trial, randomly select 5000 samples for the training set & rest as test
        X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=5000)
        max_features = [1, 2, 4, 6, 8, 12, 16, 20]
        # metrics to evaluate model on
        scoring = {"Accuracy": make_scorer(accuracy_score), "F1_Score": "f1", "AUC": "roc_auc"}

        # Create a pipeline
        pipe = Pipeline([('std', StandardScaler()),
                         ('classifier', RandomForestClassifier(n_estimators=1024))])
        # Create search space of candidate learning algorithms and their hyperparameters
        search_space = [{'classifier': [RandomForestClassifier(n_estimators=1024)],
                         'classifier__max_features': max_features}]
        # Grid Search with stratified 5 folds cross validation to find best hyperparmeters
        search_results = GridSearchCV(pipe, search_space, scoring = scoring,
                                      refit = False, cv=StratifiedKFold(n_splits = 5))
        # train models
        search_results.fit(X_train, y_train)

        # find optimal hyperparameters for each metric
        AUC_rank = search_results.cv_results_["rank_test_AUC"]
        # find index of #1 in the rank array to find best hyperparameter
        AUC_ind = np.argmin(AUC_rank)
        # save best hyperparameters
        opt_AUC = search_results.cv_results_["params"][AUC_ind]

        Acc_rank = search_results.cv_results_["rank_test_Accuracy"]
        # find index of #1 in the rank array to find best hyperparameter
        Acc_ind = np.argmin(Acc_rank)
        # save best hyperparameters
        opt_Acc = search_results.cv_results_["params"][Acc_ind]

        F1_rank = search_results.cv_results_["rank_test_F1_Score"]
```

```
        # find index of #1 in the rank array to find best hyperparameter
        F1_ind = np.argmin(F1_rank)
        # save best hyperparameters
        opt_F1 = search_results.cv_results_["params"][F1_ind]

        # train 3 models with the optimal parameters -- one model for each metric
        # AUC model
        rf_AUC_model = RandomForestClassifier(n_estimators=1024,
                                              max_features = opt_AUC["classifier__max_features"])
        rf_AUC_model.fit(X_train, y_train)
        # make predictions & calculate AUC on both training and testing sets
        rf_AUC_pred_train = (rf_AUC_model.predict_proba(X_train)[:,1])
        rf_AUC_score_train = roc_auc_score(y_train, rf_AUC_pred_train)
        rf_AUC_pred_test = (rf_AUC_model.predict_proba(X_test)[:,1])
        rf_AUC_score_test = roc_auc_score(y_test, rf_AUC_pred_test)
        # add AUC for current trial
        rf_AUC_train.append(rf_AUC_score_train)
        rf_AUC_test.append(rf_AUC_score_test)

        # Accuracy model
        rf_Acc_model = RandomForestClassifier(n_estimators=1024,
                                              max_features = opt_Acc["classifier__max_features"])
        rf_Acc_model.fit(X_train, y_train)
        # calculate accuracy on both training and testing sets
        rf_Acc_score_train = rf_Acc_model.score(X_train, y_train)
        rf_Acc_score_test = rf_Acc_model.score(X_test, y_test)
        # add Accuracy for current trial
        rf_Acc_train.append(rf_Acc_score_train)
        rf_Acc_test.append(rf_Acc_score_test)

        # F1 Score model
        rf_F1_model = RandomForestClassifier(n_estimators=1024,
                                             max_features = opt_F1["classifier__max_features"])
        rf_F1_model.fit(X_train, y_train)
        # make predictions & calculate F1 score on both training and testing sets
        rf_F1_pred_train = (rf_F1_model.predict(X_train))
        rf_F1_score_train = f1_score(y_train, rf_F1_pred_train)
        rf_F1_pred_test = (rf_F1_model.predict(X_test))
        rf_F1_score_test = f1_score(y_test, rf_F1_pred_test)
        # add F1 score for current trial
        rf_F1_train.append(rf_F1_score_train)
        rf_F1_test.append(rf_F1_score_test)
```

```
    # average AUC, Accuracy, and F1 score across all 5 trials
    rf_AUC_train_m, rf_AUC_test_m = np.mean(rf_AUC_train), np.mean(rf_AUC_test)
    rf_Acc_train_m, rf_Acc_test_m = np.mean(rf_Acc_train), np.mean(rf_Acc_test)
    rf_F1_train_m, rf_F1_test_m = np.mean(rf_F1_train), np.mean(rf_F1_test)
    # combine average training metrics into one array for current trial
    rf_trial_metrics_train = [rf_AUC_train_m, rf_Acc_train_m, rf_F1_train_m]
    # combine average testing metrics into one array for current trial
    rf_trial_metrics_test = [rf_AUC_test_m, rf_Acc_test_m, rf_F1_test_m]
    # add average train and test metrics for current trial
    rf_metrics_train.append(rf_trial_metrics_train)
    rf_metrics_test.append(rf_trial_metrics_test)

    # print raw test metric values for current trial
    print("Raw test values")
    print("AUC:", rf_AUC_test)
    print("Acc:", rf_Acc_test)
    print("F1:", rf_F1_test)
    # print average metrics for train and test for current trial
    print("train trial metrics:", rf_trial_metrics_train)
    print("test trial metrics:", np.round(rf_trial_metrics_test, 3))

# final metrics from all 4 datasets
rf_metrics_train = np.array(rf_metrics_train)
print("Random Forests metrics train:\n", rf_metrics_train)
rf_metrics_train_m = np.mean(rf_metrics_train, axis=0)
print("Average Random Forests train metrics:\n", rf_metrics_train_m)
rf_metrics_test = np.array(rf_metrics_test)
print("Random Forests metrics test:\n", rf_metrics_test)
rf_metrics_test_m = np.mean(rf_metrics_test, axis=0)
print("Average Random Forests test metrics:\n", rf_metrics_test_m)
# calculate average across metrics
rf_metrics_test_m_2 = np.mean(rf_metrics_test, axis=1)
print("Average across Random Forests test metrics:\n", rf_metrics_test_m_2)
```

```
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9956086459416813, 0.9955518649409411, 0.9953060760871978, 0.9953704261712993, 0.99600150030
15847]
Acc: [0.9727093252816166, 0.9742190221809314, 0.9725931947509, 0.9739867611194983, 0.97363836952734
```

```
87]
F1: [0.9707955689828802, 0.9721115537848606, 0.9695729992329327, 0.9719743621968078, 0.971005236939
5835]
train trial metrics: [1.0, 1.0, 1.0]
test trial metrics: [0.996 0.973 0.971]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9906156045037205, 0.9914715482242094, 0.9910284925030058, 0.9905158181316122, 0.99079865717
41171]
Acc: [0.9460666666666666, 0.9483333333333334, 0.9463333333333334, 0.9458, 0.9492]
F1: [0.9459694116075603, 0.9482874412357287, 0.947354302193012, 0.9455153949129853, 0.9481182795698
925]
train trial metrics: [1.0, 1.0, 1.0]
test trial metrics: [0.991 0.947 0.947]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9020488873165246, 0.9020604394859583, 0.8986109577799153, 0.9010217851335058, 0.89936777883
29375]
Acc: [0.8232554182898968, 0.8195714672610988, 0.8185801684687125, 0.8200992340437352, 0.82126934855
52385]
F1: [0.8226898891495652, 0.8190312510960681, 0.8178036213183585, 0.816701029466987, 0.8185276971786
889]
train trial metrics: [1.0, 1.0, 1.0]
test trial metrics: [0.901 0.821 0.819]
trial: 0
trial: 1
trial: 2
trial: 3
trial: 4
Raw test values
AUC: [0.9584702330896755, 0.9589003064393685, 0.9561498555580314, 0.9576883362044625, 0.96126368941
68683]
Acc: [0.9015984654731458, 0.9040281329923273, 0.8998721227621483, 0.8966112531969309, 0.90524296675
19182]
F1: [0.9316575722336524, 0.9326083080763059, 0.9305074202434905, 0.9284089388465141, 0.933936489452
```

```
2327]
train trial metrics: [1.0, 1.0, 1.0]
test trial metrics: [0.958 0.901 0.931]
Random Forests metrics train:
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Average Random Forests train metrics:
 [1. 1. 1.]
Random Forests metrics test:
 [[0.9955677  0.97342933 0.97109194]
 [0.99088602 0.94714667 0.94704897]
 [0.90062197 0.82055513 0.8189507 ]
 [0.95849448 0.90147059 0.93142375]]
Average Random Forests test metrics:
 [0.96139255 0.91065043 0.91712884]
Average across Random Forests test metrics:
 [0.98002966 0.96169389 0.84670926 0.93046294]
```

These are the full arrays for the rf values in both tables 2 and 3, and will be used to calculate the t tests and p values.

In [ ]:

```
# table 2 rf arrays (cols of rf metrics test)
rf_auc = [0.9955677, 0.99088602, 0.90062197, 0.95849448]
rf_acc = [0.97342933, 0.94714667, 0.82055513, 0.90147059]
rf_f1 = [0.97109194, 0.94704897, 0.8189507, 0.93142375]
rf_mean_1 = [0.96139255, 0.91065043, 0.91712884]

# table 3 rf arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
rf_d1 = [0.9797045134020594, 0.9806274803022443, 0.9791574233570102,
         0.9804438498292019, 0.9802150355895057]
rf_d2 = [0.9608838942593159, 0.9626974409310906, 0.9615720426764504,
         0.9606104043481992, 0.9627056455813365]
rf_d3 = [0.8493313982519956, 0.8468877192810417, 0.8449982491889955,
         0.8459406828814093, 0.8463882748556216]
rf_d4 = [0.9305754235988246, 0.9318455825026671, 0.9288431328545568,
         0.9275695094159692, 0.9334810485403398]
rf_mean_2 = [0.98002966, 0.96169389, 0.84670926, 0.93046294]
```

Final Metrics

Table 2

In [34]:
```
# table 2
print("lr:", lr_metrics_test_m)
print("svm:", svm_metrics_test_m)
print("knn:", knn_metrics_test_m)
print("rf:", rf_metrics_test_m)

final_metrics = np.array([lr_metrics_test_m, svm_metrics_test_m,
                          knn_metrics_test_m, rf_metrics_test_m])
# models in rows and metrics in columns
print("Average Metrics across Datasets per Model:\n", final_metrics)
```

```
lr: [0.86647845 0.79874321 0.81167886]
svm: [0.84012632 0.76755563 0.79634207]
knn: [0.85787983 0.81548121 0.84312728]
rf: [0.96139255 0.91065043 0.91712884]
Average Metrics across Datasets per Model:
 [[0.86647845 0.79874321 0.81167886]
 [0.84012632 0.76755563 0.79634207]
 [0.85787983 0.81548121 0.84312728]
 [0.96139255 0.91065043 0.91712884]]
```

In [35]:
```
# calculate mean values across metrics from table 2
t2_mean_metrics = np.mean(final_metrics, axis = 1)

print("mean values:", t1_mean_metrics)
```

```
mean values: [0.82563351 0.80134134 0.83882944 0.92972394]
```

Table 3

In [40]:
```
# table 3
# calculate average metrics for each algorithm across metrics
print("lr:", lr_metrics_test_m_2)
print("svm:", svm_metrics_test_m_2)
print("knn:", knn_metrics_test_m_2)
print("rf:", rf_metrics_test_m_2)

final_metrics_2 = np.array([lr_metrics_test_m_2, svm_metrics_test_m_2,
                            knn_metrics_test_m_2, rf_metrics_test_m_2])
# models in rows and datasets in columns
print("Average Metrics for all Algorithms:\n", final_metrics_2)
```

```
lr: [0.93024686 0.75435848 0.74932215 0.86860654]
svm: [0.81255548 0.90328871 0.7105528  0.77896837]
knn: [0.85908809 0.96847997 0.80633238 0.72141731]
rf: [0.98002966 0.96169389 0.84670926 0.93046294]
Average Metrics for all Algorithms:
 [[0.93024686 0.75435848 0.74932215 0.86860654]
 [0.81255548 0.90328871 0.7105528  0.77896837]
 [0.85908809 0.96847997 0.80633238 0.72141731]
 [0.98002966 0.96169389 0.84670926 0.93046294]]
```

In [41]:
```
# calculate mean values across datasets from table 1
t3_mean_datasets = np.mean(final_metrics_2, axis = 1)

print("mean values:", t2_mean_datasets)
```

```
mean values: [0.82563351 0.80134134 0.83882944 0.92972394]
```

Calculating T-tests and P values

In [165]:
```
# table 2 lr arrays (cols of Lr metrics test)
lr_auc = [0.96856484, 0.81103193, 0.79646982, 0.88984722]
lr_acc = [0.91464406, 0.72506667, 0.72423908, 0.83102302]
lr_f1 = [0.90753167, 0.72697683, 0.72725756, 0.88494939]
lr_mean_1 = [0.86647845, 0.79874321, 0.81167886]

# table 3 lr arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
lr_d1 = [0.9586460235320803, 0.9565694684321265, 0.9608197341226062,
         0.8157598277168324, 0.959439234438331]
lr_d2 = [0.7565088702051469, 0.7535006000863818, 0.7573735514758195,
         0.7522884514861006, 0.752120912899643]
lr_d3 = [0.751667668556535, 0.7509692372282948, 0.74418526232208,
         0.7448603087168252, 0.7549282796006591]
lr_d4 = [0.8688679110624338, 0.8708719759122815, 0.8668214982190855,
         0.8685907654597697, 0.8678805680443218]
lr_mean_2 = [0.93024686, 0.75435848, 0.74932215, 0.86860654]
```

In [166]:
```
# table 2 svm arrays (cols of svm metrics test)
svm_auc = [0.86308537, 0.9519339, 0.75460011, 0.79088587]
svm_acc = [0.78283591, 0.87936, 0.694689, 0.7133376]
svm_f1 = [0.79174517, 0.87857221, 0.68236928, 0.83268163]
svm_mean_1 = [0.84012632, 0.76755563, 0.79634207]

# table 3 svm arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
svm_d1 = [0.8117481691067928, 0.8114812624881037, 0.8143406000018384,
          0.8101530147623502, 0.8150543707217408]
svm_d2 = [0.9035421730714287, 0.9037623389854074, 0.9011522524561654,
          0.9041148849546022, 0.9038718810307453]
svm_d3 = [0.7089026670765787, 0.7165935378258358, 0.7064106814361778,
          0.7099869710363361, 0.7108701247511338]
svm_d4 = [0.7896899125598481, 0.7653704630352699, 0.7833403412046526,
          0.7772230946078841, 0.7792180183022389]
svm_mean_2 = [0.81255548, 0.90328871, 0.7105528, 0.77896837]
```

In [167]:
```
# table 2 knn arrays (cols of knn metrics test)
knn_auc = [0.91618275, 0.99060503, 0.86008995, 0.66464157]
knn_acc = [0.83272558, 0.95746667, 0.7805945, 0.69113811]
knn_f1 = [0.82835594, 0.95736822, 0.77831268, 0.80847226]
knn_mean_1 = [0.85787983, 0.81548121, 0.84312728]

# table 3 knn arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
knn_d1 = [0.8650658088942013, 0.8363658225559866, 0.8634172401507417,
          0.8611621086321147, 0.8694294834315387]
knn_d2 = [0.9697651341969605, 0.9682222191846863, 0.9683229017355536,
          0.9685493101230817, 0.9675402988951393]
knn_d3 = [0.8048035814698253, 0.8051453861700839, 0.8065270214822863,
          0.8050974487241892, 0.8100884609532483]
knn_d4 = [0.7264361218729348, 0.7199983760968302, 0.7198371276286712,
          0.7204015251781053, 0.7204134129556534]
knn_mean_2 = [0.85908809, 0.96847997, 0.80633238, 0.72141731]
```

In [168]:
```
# table 2 rf arrays (cols of rf metrics test)
rf_auc = [0.9955677, 0.99088602, 0.90062197, 0.95849448]
rf_acc = [0.97342933, 0.94714667, 0.82055513, 0.90147059]
rf_f1 = [0.97109194, 0.94704897, 0.8189507, 0.93142375]
rf_mean_1 = [0.96139255, 0.91065043, 0.91712884]

# table 3 rf arrays (averaged AUC, ACC, and F1 metrics for all trials per dataset)
rf_d1 = [0.9797045134020594, 0.9806274803022443, 0.9791574233570102,
         0.9804438498292019, 0.9802150355895057]
rf_d2 = [0.9608838942593159, 0.9626974409310906, 0.9615720426764504,
         0.9606104043481992, 0.9627056455813365]
rf_d3 = [0.8493313982519956, 0.8468877192810417, 0.8449982491889955,
         0.8459406828814093, 0.8463882748556216]
rf_d4 = [0.9305754235988246, 0.9318455825026671, 0.9288431328545568,
         0.9275695094159692, 0.9334810485403398]
rf_mean_2 = [0.98002966, 0.96169389, 0.84670926, 0.93046294]
```

Based off of table 2's values, it is evident that the Random Forests model performed the best in all three metrics, so those will be the values compared to in the Independent t-test. For example, the p value between rf_auc and lr_auc calculation is shown below.

In [185]:
```
# Compute Independent t-test
stat, p_val = stats.ttest_ind(np.array(rf_auc), np.array(lr_auc), equal_var = False)
p_val
```

Out[185]:
```
0.09458285528558459
```

Based off of table 3's values, it is evident that the Random Forests model performed the best in datasets 1, 3, and 4, with the exception of dataset 2 where KNN slightly outperformed. So those will be the values compared to in the Independent t-test. For example, the p value between rf_d1 and lr_d1 calculation is shown below.

In [220]:
```
# Compute Independent t-test
stat, p_val = stats.ttest_ind(np.array(rf_d1), np.array(lr_d1), equal_var = False)
p_val
```

Out[220]:
```
0.15705374922671014
```

Secondary Tables

In [38]:
```
# secondary table 1 -- mean training set performance
print("lr train mean:", lr_metrics_train_m)
print("svm train mean:", svm_metrics_train_m)
print("knn train mean:", knn_metrics_train_m)
print("rf train mean:", rf_metrics_train_m)

final_metrics_train = np.array([lr_metrics_train_m, svm_metrics_train_m,
                                knn_metrics_train_m, rf_metrics_train_m])
# models in rows and metrics in columns
print("Average Training set Metrics across Datasets per Model:\n", final_metrics_train)
```

```
lr train mean: [0.87092374 0.8027     0.81679145]
svm train mean: [0.85132273 0.77348    0.80148168]
knn train mean: [1.         0.96671   0.9700947]
rf train mean: [1. 1. 1.]
Average Training set Metrics across Datasets per Model:
 [[0.87092374 0.8027     0.81679145]
 [0.85132273 0.77348    0.80148168]
 [1.         0.96671    0.9700947 ]
 [1.         1.         1.        ]]
```

In [39]:
```
# calculate mean values across metrics from secondary table 1
st1_mean_datasets = np.mean(final_metrics_train, axis = 1)

print("mean values:", st1_mean_datasets)
```

```
mean values: [0.8301384  0.80876147 0.9789349  1.        ]
```

Student's I discussed with:

- Anjali Ramesh
- Urmi Suresh
- Harmeena Sandhu