Lab package 3: Supervised classification and regression

Wednesday, October 6, 2021 1:43 PM

Supervised classification is in general easier to understand than unsupervised classification.
Data for unsupervised classification have measures for features (X).
Data for supervised classfication have measures for features (X) and the true labels (y).

The general procedures for supervised classification:

1. Clean data
2. Select appropriate features as the input variables
3. Randomly split data into a training set and a test set: commonly 70/30, 75/25, or 80/20 split
4. Scale and transform data:
   a. Use training features to fit a scaler
   b. Transform both training features and test features.
5. Select, initial, and fit a machine learning model with training features and training labels
6. Fit the model with test features to predict test labels
7. Compare test labels and and predict test labels to evaluate the model performance
8. Calculate additional evaluation metrics
9. Run different models to select the one with the best performance
10. Some may opt to use the entire data set to finalize the model (especially for software production)

Q#1: Which of the following statement is false:

A. Clustering analysis is an unsupervised classification
B. Hotspot analysis is an unsupervised classification
C. Facial recognition is a supervised classification
D. Toponym resolution is a supervised classification
E. None of the above

Q#2: Which of the following statement is false in comparison between training and validation during machine learning:

A. The training explores the space of a chosen hypothesis class
B. The validation seeks the best possible hypothesis in a chosen hypothesis class
C. More data samples should be used for training than for validation
D. Validation data are part of the training data
E. None of the above

Strictly speaking, the training procedures should include two separate data sets: training data and validation data. A more rigorous appraoch includes the following:

1. Further split the training data into training data and validation data (commonly, 80/20). If considered validation, we usually have the training/testing split to 80/20; and then have training/validation split to 80/20; resulting training/validation/testing split at 60/20/20
2. Most machine learning methods involve training hyperparameters and parameters. Validation data are used as a part of the training process to fine tune parameters and assess the model skills.

Classification is to classify observations into discrete groups (y) based on measurable features (X). Groups are discrete labels.
Regression is to make predictions (y) based on measurable features (X). Predictions are continuous values.

Procedures wise, classification and regression are similar, but mathematically they are distinct.

We will practice supervised classification and regression in Lab package 3.

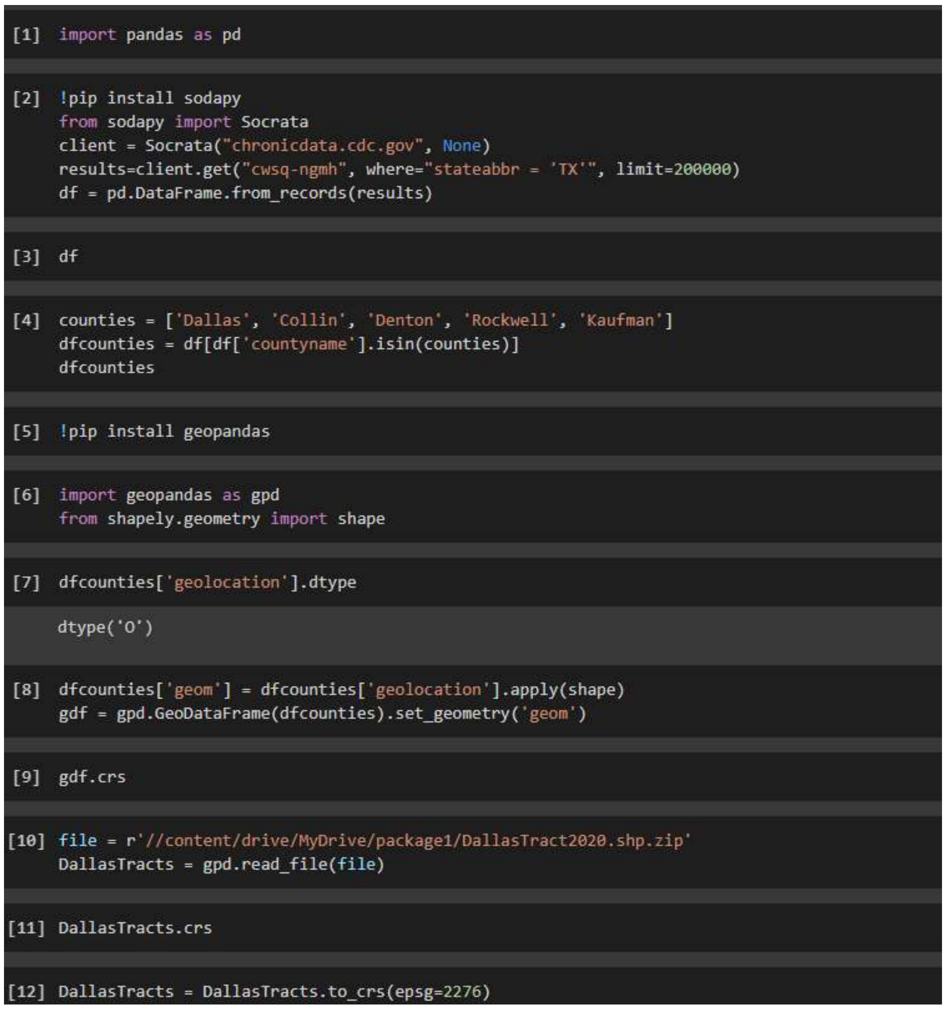
Q#3: If we want to do a 5-fold cross-validation, what is the proportion of training data to validation data?

A. 5 to 1
B. 4 to 1
C. 3 to 2
D. 2 to 3
E. All of the above are possible

Objectives:

1. Are there some census tracts with distinctively more population of high cholestrol than the other tracts in the city of Dallas?
2. If so, how well can we group the census tracts according the percentage of people with high cholestrol in each tract, and what is the optimal number of groups?
3. How may food consumptions predict which group of percent population with high choletrol for a census tract in Dallas?
4. How may food consumptions predict the percent population with high cholestrol in Dallas?

The python script:



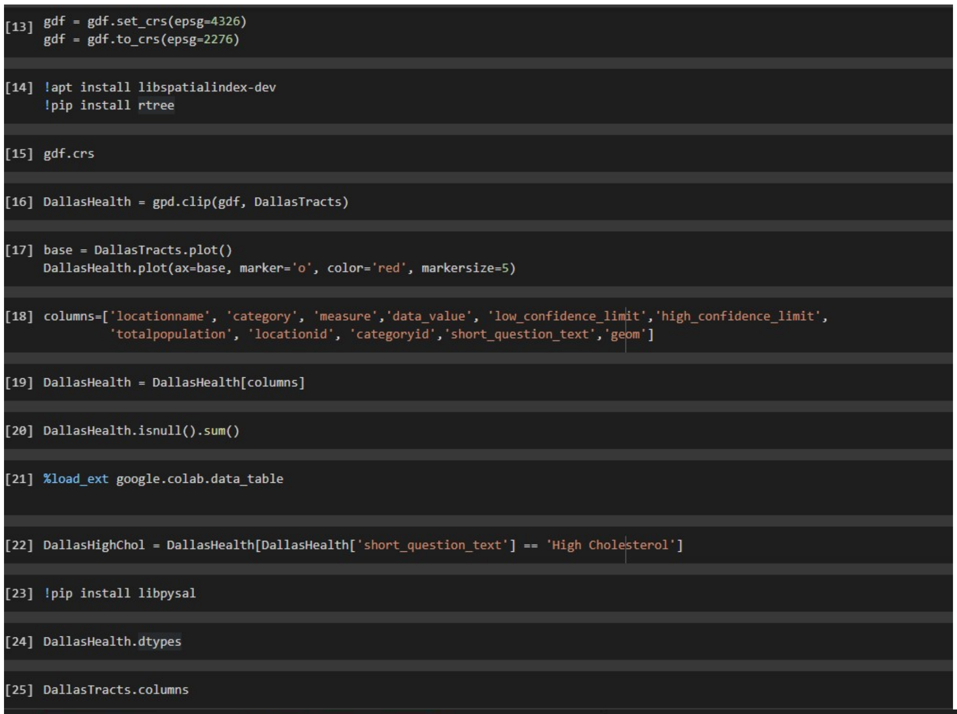
```
[1]  import pandas as pd

[2]  !pip install sodapy
     from sodapy import Socrata
     client = Socrata("chronicdata.cdc.gov", None)
     results=client.get("cwsq-ngmh", where="stateabbr = 'TX'", limit=200000)
     df = pd.DataFrame.from_records(results)

[3]  df

[4]  counties = ['Dallas', 'Collin', 'Denton', 'Rockwell', 'Kaufman']
     dfcounties = df[df['countyname'].isin(counties)]
     dfcounties

[5]  !pip install geopandas

[6]  import geopandas as gpd
     from shapely.geometry import shape

[7]  dfcounties['geolocation'].dtype

     dtype('O')

[8]  dfcounties['geom'] = dfcounties['geolocation'].apply(shape)
     gdf = gpd.GeoDataFrame(dfcounties).set_geometry('geom')

[9]  gdf.crs

[10] file = r'//content/drive/MyDrive/package1/DallasTract2020.shp.zip'
     DallasTracts = gpd.read_file(file)

[11] DallasTracts.crs

[12] DallasTracts = DallasTracts.to_crs(epsg=2276)
```

Q#4: The health data are from the Center for Disease Control https://dev.socrata.com/foundry/chronicdata.cdc.gov/cwsq-ngmh. The data include ____________(a number) measures at four geographic area units: county, place, census tracts, and ____________. The measures are calculated from three sources. Specifically, high cholestrol estimates are from ________________(a year)_____________ (a data source - using the acronym).

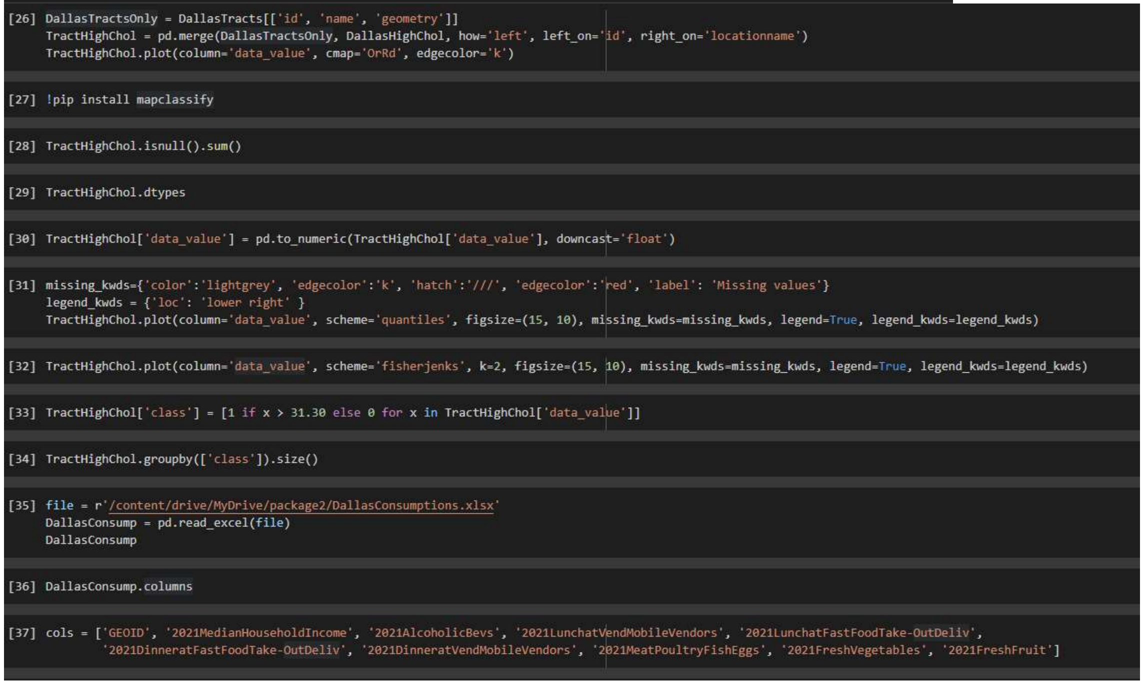
Q#5: Why do we need to do apply(shape) in line [8]?

A. Assign geolocation to a shape
B. Convert geolocation from data type object to data type shape
C. Create a new data type shape from geolocation
D. Transform the shape of geolocation
E. None of the above

Q#6: DallasHealth has ______ (a number) records and ______ (a number) features. There are _________ (a number) of records with missing data. The field ____________(a field name) indicates the records with data values for high cholesterol.



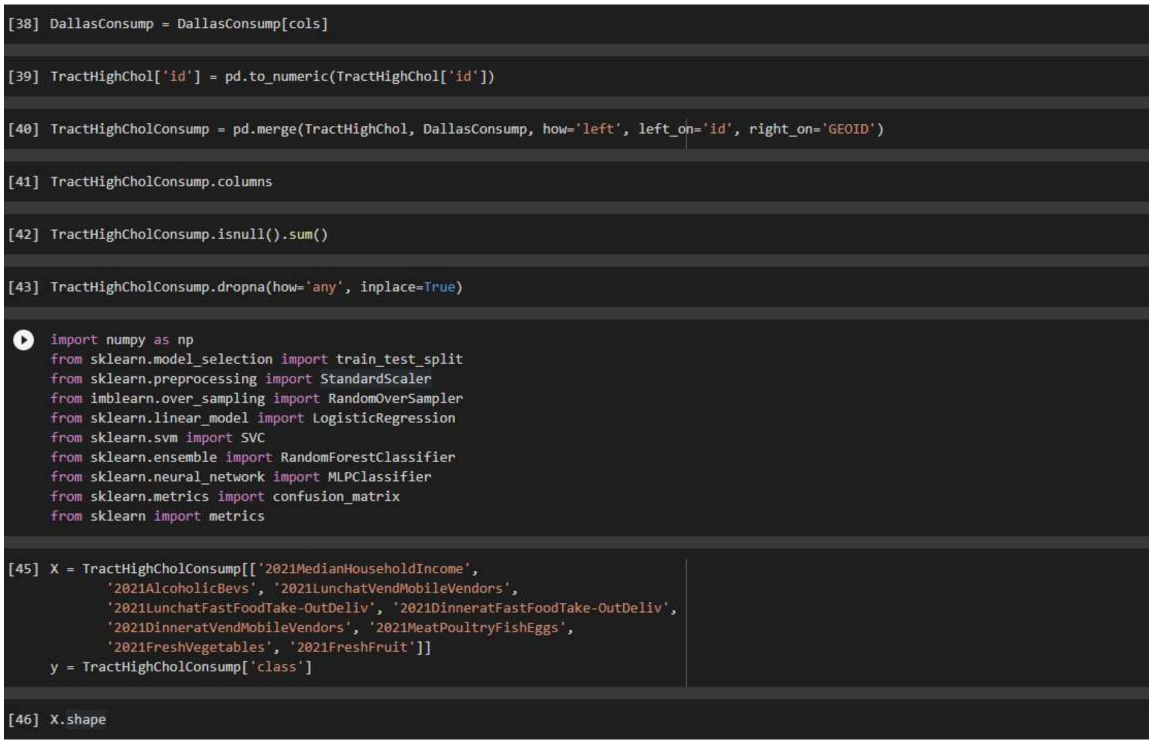
```
[13] gdf = gdf.set_crs(epsg=4326)
     gdf = gdf.to_crs(epsg=2276)

[14] !apt install libspatialindex-dev
     !pip install rtree

[15] gdf.crs

[16] DallasHealth = gpd.clip(gdf, DallasTracts)

[17] base = DallasTracts.plot()
     DallasHealth.plot(ax=base, marker='o', color='red', markersize=5)

[18] columns=['locationname', 'category', 'measure','data_value', 'low_confidence_limit','high_confidence_limit',
          'totalpopulation', 'locationid', 'categoryid','short_question_text','geom']

[19] DallasHealth = DallasHealth[columns]

[20] DallasHealth.isnull().sum()

[21] %load_ext google.colab.data_table

[22] DallasHighChol = DallasHealth[DallasHealth['short_question_text'] == 'High Cholesterol']

[23] !pip install libpysal

[24] DallasHealth.dtypes

[25] DallasTracts.columns
```



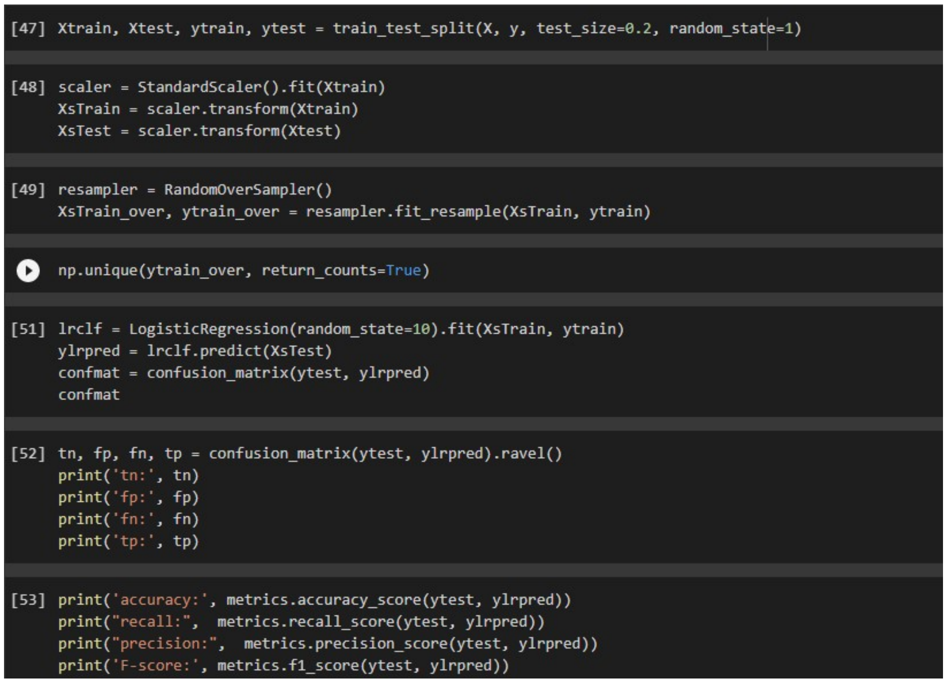
```
[26] DallasTractsOnly = DallasTracts[['id', 'name', 'geometry']]
     TractHighChol = pd.merge(DallasTractsOnly, DallasHighChol, how='left', left_on='id', right_on='locationname')
     TractHighChol.plot(column='data_value', cmap='OrRd', edgecolor='k')

[27] !pip install mapclassify

[28] TractHighChol.isnull().sum()

[29] TractHighChol.dtypes

[30] TractHighChol['data_value'] = pd.to_numeric(TractHighChol['data_value'], downcast='float')

[31] missing_kwds={'color':'lightgrey', 'edgecolor':'k', 'hatch':'///', 'edgecolor':'red', 'label': 'Missing values'}
     legend_kwds = {'loc': 'lower right' }
     TractHighChol.plot(column='data_value', scheme='quantiles', figsize=(15, 10), missing_kwds=missing_kwds, legend=True, legend_kwds=legend_kwds)

[32] TractHighChol.plot(column='data_value', scheme='fisherjenks', k=2, figsize=(15, 10), missing_kwds=missing_kwds, legend=True, legend_kwds=legend_kwds)

[33] TractHighChol['class'] = [1 if x > 31.30 else 0 for x in TractHighChol['data_value']]

[34] TractHighChol.groupby(['class']).size()

[35] file = r'/content/drive/MyDrive/package2/DallasConsumptions.xlsx'
     DallasConsump = pd.read_excel(file)
     DallasConsump

[36] DallasConsump.columns

[37] cols = ['GEOID', '2021MedianHouseholdIncome', '2021AlcoholicBevs', '2021LunchatVendMobileVendors', '2021LunchatFastFoodTake-OutDeliv',
         '2021DinneratFastFoodTake-OutDeliv', '2021DinneratVendMobileVendors', '2021MeatPoultryFishEggs', '2021FreshVegetables', '2021FreshFruit']
```

Q#7: Why do pd.to_numeric in line [30]?

A. We need to downcast data_value from float to integer.
B. We need to downcast data_value from integer to float.
C. We need to downcast data_value from string to float.
D. We need to downcast data_value from object to float.



```
[38] DallasConsump = DallasConsump[cols]

[39] TractHighChol['id'] = pd.to_numeric(TractHighChol['id'])

[40] TractHighCholConsump = pd.merge(TractHighChol, DallasConsump, how='left', left_on='id', right_on='GEOID')

[41] TractHighCholConsump.columns

[42] TractHighCholConsump.isnull().sum()

[43] TractHighCholConsump.dropna(how='any', inplace=True)

     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from imblearn.over_sampling import RandomOverSampler
     from sklearn.linear_model import LogisticRegression
     from sklearn.svm import SVC
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neural_network import MLPClassifier
     from sklearn.metrics import confusion_matrix
     from sklearn import metrics

[45] X = TractHighCholConsump[['2021MedianHouseholdIncome',
            '2021AlcoholicBevs', '2021LunchatVendMobileVendors',
            '2021LunchatFastFoodTake-OutDeliv', '2021DinneratFastFoodTake-OutDeliv',
            '2021DinneratVendMobileVendors', '2021MeatPoultryFishEggs',
            '2021FreshVegetables', '2021FreshFruit']]
     y = TractHighCholConsump['class']

[46] X.shape
```

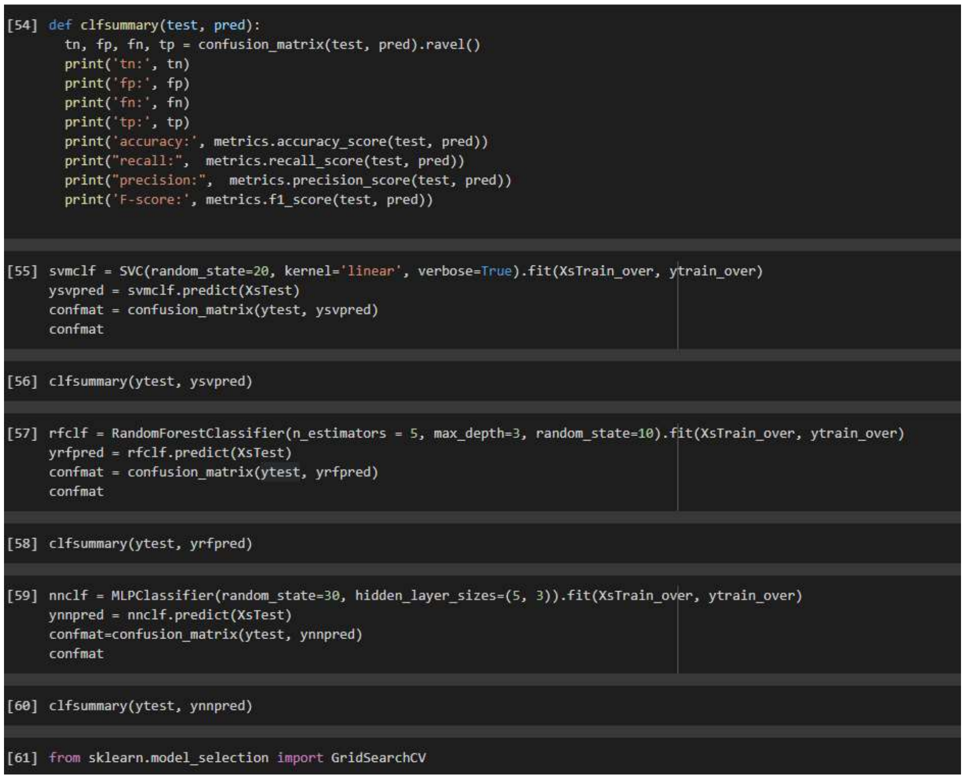


```
[47] Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=1)

[48] scaler = StandardScaler().fit(Xtrain)
     XsTrain = scaler.transform(Xtrain)
     XsTest = scaler.transform(Xtest)

[49] resampler = RandomOverSampler()
     XsTrain_over, ytrain_over = resampler.fit_resample(XsTrain, ytrain)

     np.unique(ytrain_over, return_counts=True)

[51] lrclf = LogisticRegression(random_state=10).fit(XsTrain, ytrain)
     ylrpred = lrclf.predict(XsTest)
     confmat = confusion_matrix(ytest, ylrpred)
     confmat

[52] tn, fp, fn, tp = confusion_matrix(ytest, ylrpred).ravel()
     print('tn:', tn)
     print('fp:', fp)
     print('fn:', fn)
     print('tp:', tp)

[53] print('accuracy:', metrics.accuracy_score(ytest, ylrpred))
     print("recall:",  metrics.recall_score(ytest, ylrpred))
     print("precision:",  metrics.precision_score(ytest, ylrpred))
     print('F-score:', metrics.f1_score(ytest, ylrpred))
```

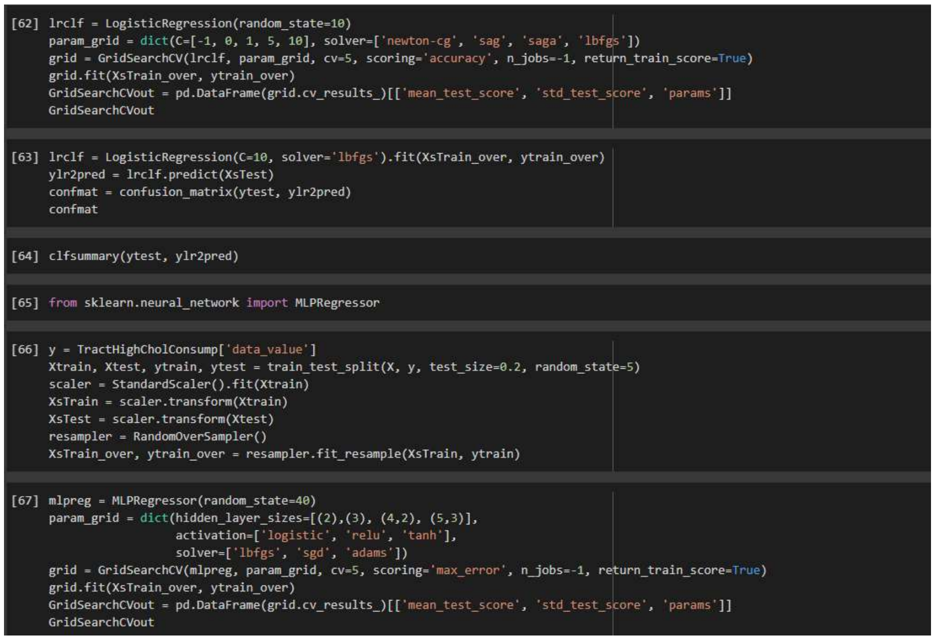
Q#8: After train_test_split, we have ___________ (a number) training samples and ______________(a number) test samples. There are __________ (more or less) training samples in class 0 than in class 1. Based on the numbers of training samples in classes 0 and 1, if an algorithm blindly makes all predictions to class 1, the algorithm's prediction accuracy will be ____________(a ratio, rount up to two digits, such as 90%).

Q#9: Which of the statements is false:

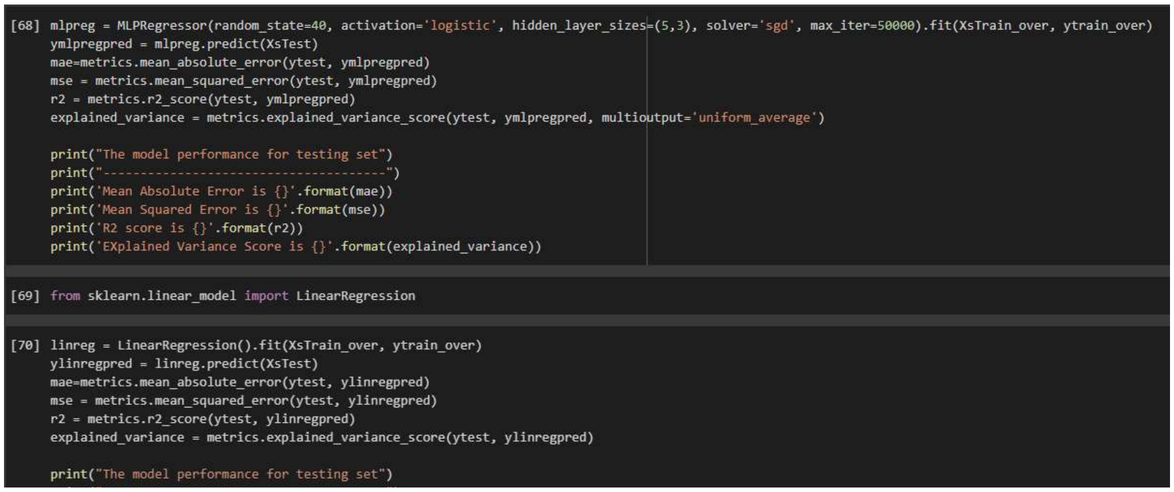
A. If training data have high imbalanced distributions among classes, the prediction results are easier to get high accuracy.
B. If a prediction has high accuracy, we can have high trust in the prediction skills of the algorithm.
C. A prediction can be at a high accuracy but low precision if the prediction produces high false positive cases.
D. A prediction has a high recall if the algorithm is good at predicting positive cases.
E. None of the above.



```
[54] def clfsummary(test, pred):
       tn, fp, fn, tp = confusion_matrix(test, pred).ravel()
       print('tn:', tn)
       print('fp:', fp)
       print('fn:', fn)
       print('tp:', tp)
       print('accuracy:', metrics.accuracy_score(test, pred))
       print("recall:",  metrics.recall_score(test, pred))
       print("precision:",  metrics.precision_score(test, pred))
       print('F-score:', metrics.f1_score(test, pred))

[55] svmclf = SVC(random_state=20, kernel='linear', verbose=True).fit(XsTrain_over, ytrain_over)
     ysvpred = svmclf.predict(XsTest)
     confmat = confusion_matrix(ytest, ysvpred)
     confmat

[56] clfsummary(ytest, ysvpred)

[57] rfclf = RandomForestClassifier(n_estimators = 5, max_depth=3, random_state=10).fit(XsTrain_over, ytrain_over)
     yrfpred = rfclf.predict(XsTest)
     confmat = confusion_matrix(ytest, yrfpred)
     confmat

[58] clfsummary(ytest, yrfpred)

[59] nnclf = MLPClassifier(random_state=30, hidden_layer_sizes=(5, 3)).fit(XsTrain_over, ytrain_over)
     ynnpred = nnclf.predict(XsTest)
     confmat=confusion_matrix(ytest, ynnpred)
     confmat

[60] clfsummary(ytest, ynnpred)

[61] from sklearn.model_selection import GridSearchCV
```



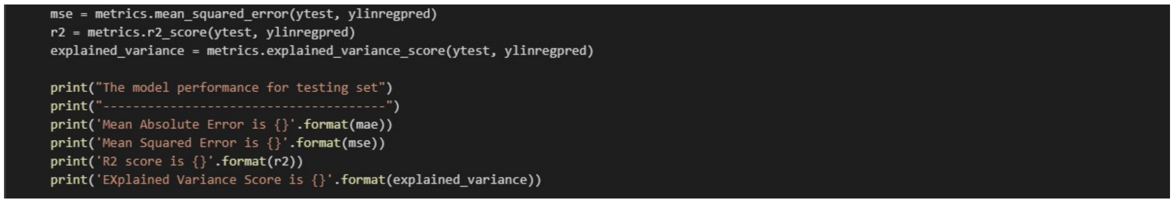
```
[62] lrclf = LogisticRegression(random_state=10)
     param_grid = dict(C=[-1, 0, 1, 5, 10], solver=['newton-cg', 'sag', 'saga', 'lbfgs'])
     grid = GridSearchCV(lrclf, param_grid, cv=5, scoring='accuracy', n_jobs=-1, return_train_score=True)
     grid.fit(XsTrain_over, ytrain_over)
     GridSearchCVout = pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
     GridSearchCVout

[63] lrclf = LogisticRegression(C=10, solver='lbfgs').fit(XsTrain_over, ytrain_over)
     ylr2pred = lrclf.predict(XsTest)
     confmat = confusion_matrix(ytest, ylr2pred)
     confmat

[64] clfsummary(ytest, ylr2pred)

[65] from sklearn.neural_network import MLPRegressor

[66] y = TractHighCholConsump['data_value']
     Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=5)
     scaler = StandardScaler().fit(Xtrain)
     XsTrain = scaler.transform(Xtrain)
     XsTest = scaler.transform(Xtest)
     resampler = RandomOverSampler()
     XsTrain_over, ytrain_over = resampler.fit_resample(XsTrain, ytrain)

[67] mlpreg = MLPRegressor(random_state=40)
     param_grid = dict(hidden_layer_sizes=[(2),(3), (4,2), (5,3)],
                       activation=['logistic', 'relu', 'tanh'],
                       solver=['lbfgs', 'sgd', 'adams'])
     grid = GridSearchCV(mlpreg, param_grid, cv=5, scoring='max_error', n_jobs=-1, return_train_score=True)
     grid.fit(XsTrain_over, ytrain_over)
     GridSearchCVout = pd.DataFrame(grid.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
     GridSearchCVout
```



```
[68] mlpreg = MLPRegressor(random_state=40, activation='logistic', hidden_layer_sizes=(5,3), solver='sgd', max_iter=50000).fit(XsTrain_over, ytrain_over)
     ymlpregpred = mlpreg.predict(XsTest)
     mae=metrics.mean_absolute_error(ytest, ymlpregpred)
     mse = metrics.mean_squared_error(ytest, ymlpregpred)
     r2 = metrics.r2_score(ytest, ymlpregpred)
     explained_variance = metrics.explained_variance_score(ytest, ymlpregpred, multioutput='uniform_average')

     print("The model performance for testing set")
     print("--------------------------------------")
     print('Mean Absolute Error is {}'.format(mae))
     print('Mean Squared Error is {}'.format(mse))
     print('R2 score is {}'.format(r2))
     print('EXplained Variance Score is {}'.format(explained_variance))

[69] from sklearn.linear_model import LinearRegression

[70] linreg = LinearRegression().fit(XsTrain_over, ytrain_over)
     ylinregpred = linreg.predict(XsTest)
     mae=metrics.mean_absolute_error(ytest, ylinregpred)
     mse = metrics.mean_squared_error(ytest, ylinregpred)
     r2 = metrics.r2_score(ytest, ylinregpred)
     explained_variance = metrics.explained_variance_score(ytest, ylinregpred)

     print("The model performance for testing set")
```

Q#10: Which of the statements is false based on the MLPregressor and the linear regressor in lines 68-70:

A. The MLPregressor predicts better than the Linear regressor.
B. The MLPregressor consists of many logistic regressors.
C. The Linear regressor includes 9 independent variables.
D. The Linear regressor can only explain about



```
mse = metrics.mean_squared_error(ytest, ylinregpred)
r2 = metrics.r2_score(ytest, ylinregpred)
explained_variance = metrics.explained_variance_score(ytest, ylinregpred)

print("The model performance for testing set")
print("--------------------------------------")
print('Mean Absolute Error is {}'.format(mae))
print('Mean Squared Error is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('EXplained Variance Score is {}'.format(explained_variance))
```

B. The MLPregressor consists of many logistic regressors.
C. The Linear regressor includes 9 independent variables.
D. The Linear regressor can only explain about 20% of the variance in the test data.
E. None of the above.