In [1]:
```
#---------------------------------------Welcome to Exam 2: Part 2--------------------
```

In [2]:
```
#Note the following:

#1. Add your Student Name and ID below
#2. Remember that each question is for 1 point
#3. Remember to save your file before uploading to Canvas
#4. Read the instructions per cell and run the cells accordingly to help your answers/o
#5. Refer to the relevant Python Jupyter Notebooks for weeks 10 - 15 in Canvas for help
#6. Send me an email at abel.iyasele@colorado.edu if you have questions
```

In [ ]:
```

```

In [26]:
```
#Student Name: Julia Troni
#julia.troni@colorado.edu
# Student ID: 109280095
```

In [ ]:
```

```

In [ ]:
```

```

In [4]:
```
#Questions 1 - 5 walk you through the process of training a Linear Discriminant Analysi
# and a Quadratic Discriminant Analysis algorithm on the Iris dataset in scikit learn


#Run this cell first before attempting Questions 1 - 5

#Step 1: Load Necessary Libraries
#First, we'll load the necessary functions and libraries for this example

from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

#load iris dataset
iris = datasets.load_iris()

X = iris.data
y = iris.target
```

In [ ]:
```

```

In [5]:
```
#Question 1:

#Write Python code to:

#1.Create an instance of the Quadratic Discriminant Analysis function, and
#2. fit a Quadratic Discriminant Analysis model to X and y as defined in the cell above

#Use the object model_QDA to call the QuadraticDiscriminantAnalysis() function and to d
# on your X and y

#Write your answer below:
model_QDA = QuadraticDiscriminantAnalysis()
model_QDA.fit(X, y)
```

Out[5]:
```
QuadraticDiscriminantAnalysis()
```

In [6]:
```
#Question 2:

#Write Python code to:

#1.Create an instance of the Linear Discriminant Analysis function and
#2. fit a Linear Discriminant Analysis model to X and y as defined above

#Use the object model_LDA to call the LinearDiscriminantAnalysis() function and to driv
# on your X and y

#Write your answer below:

model_LDA = LinearDiscriminantAnalysis()
model_LDA.fit(X, y)
```

Out[6]:
```
LinearDiscriminantAnalysis()
```

In [7]:
```
#Once we've fit the model using our data, we can evaluate how well the model performed
#by using repeated stratified k-fold cross validation.

#We'll use 15 folds and 4 repeats:

#This cell defines a method to evaluate both the QDA and LDA models.
#We use RepeatedStratifiedKFold as defined below
#Run this cell before you continue with Questions 3,4 and 5

#Define method to evaluate model. Use RepeatedStratifiedKFold
cv = RepeatedStratifiedKFold(n_splits=15, n_repeats=4, random_state=1)
```

In [8]:
```
#Question 3:

#Write Python code to evaluate your QDA model (stored in model_QDA) using cross_val_sco

#set the scoring hyperparameter to 'accuracy'
#set your n_jobs hyperparameter to use all the cores of your CPU
#save your cross_val_score function in an object called scores_QDA
# set cv = cv
```

```
#Write your answer below:
#Evaluate model
scores_QDA = cross_val_score(model_QDA, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

In [9]:

```
#Question 4:

#Write Python code to evaluate your LDA model (stored in model_LDA) using cross_val_sco

#set the scoring hyperparameter to 'accuracy'
#set your n_jobs hyperparameter to use all the cores of your CPU
#save your cross_val_score function in an object called scores_LDA
#set cv = cv

#Write your answer below:

scores_LDA = cross_val_score(model_LDA, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

In [10]:

```
#Question 5:

#Write Python code to:

#the mean of the scores you stored in scores_QDA
#the mean of the scores you stored in scores_LDA

#Include all your code in this cell

#Write your code lines below:
print(np.mean(scores_QDA))

print(np.mean(scores_LDA))
```

```
0.9716666666666666
0.98
```

In [ ]:

In [11]:

```
#Questions 6 - 11 is about working with the Ridge Regression algorithm
# on a dataset imported from the World Wide Web

# importing libraries
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import LeaveOneOut          #Load LeaveOneOut from sklearn
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
```

In [12]:

```
#Run the next two cells before attempting the questions that follow
```

In [13]:
```
# Load dataset from the Web

url = 'https://vincentarelbundock.github.io/Rdatasets/csv/AER/GrowthSW.csv'
df = pd.read_csv(url,index_col=0).dropna()

#display dataframe df

df
```

Out[13]:

	growth	rgdp60	tradeshare	education	revolutions	assassinations
India	1.915168	765.999817	0.140502	1.45	0.133333	0.866667
Argentina	0.617645	4462.001465	0.156623	4.99	0.933333	1.933333
Japan	4.304759	2953.999512	0.157703	6.71	0.000000	0.200000
Brazil	2.930097	1783.999878	0.160405	2.89	0.100000	0.100000
United States	1.712265	9895.003906	0.160815	8.66	0.000000	0.433333
...	...	...	...	...	...	...
Cyprus	5.384184	2037.000366	0.979355	4.29	0.100000	0.166667
Malaysia	4.114544	1420.000244	1.105364	2.34	0.033333	0.033333
Belgium	2.651335	5495.001953	1.115917	7.46	0.000000	0.000000
Mauritius	3.024178	2861.999268	1.127937	2.44	0.000000	0.000000
Malta	6.652838	1374.000000	1.992616	5.64	0.000000	0.000000

65 rows × 6 columns

In [14]:
```
# Split df into input and output elements

#Note that X is the set of inputs and y is the target variable

X = df[["rgdp60","tradeshare","education","revolutions","assassinations"]]
y = df[["growth"]]


# scaling the inputs
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)

# Split data into train and test portions
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.3)
```

In [ ]:
```

```

In [15]:
```
#Question 6:

#Write Python code to create an instance of the Ridge() function/ train a model using t

#alpha = 0.5
```

```
#normalize = False
#tol = 0.001
#solver = 'auto'
#random_state = 42

#Store your results in an object called ridge_model

#Write your answer below:


ridge_model = Ridge(alpha = 0.5, normalize = False, tol = 0.001,solver ='auto', random_
```

In [ ]:

In [16]:
```
#Question 7:

#Write Python code to fit a model on the training data obtained from the Train/Test spl

#Use your trained object (i.e.ridge_model) from Question 6 for the .fit() function call


#Write your answer below:
ridge_model.fit(X_train, y_train)
```

Out[16]:
```
Ridge(alpha=0.5, random_state=42)
```

In [ ]:

In [17]:
```
#Question 8:

#Write Python code to predict the values of the target variable for data in X_test

#Use your trained ridge_model for the .predict() function call

#Store your predictions in y_pred

#Write your answer below:
# predicting the y_test
y_pred = ridge_model.predict(X_test)
```

In [18]:
```
##Just run this cell to view mean absolute error (MAE) from your result in Question 8 a

mean_absolute_error(y_test, y_pred)
```

Out[18]:
```
0.9155742073250481
```

In [19]:
```
#we define Leave One Out - using the LeaveOneOut() function - as the
#cross-validation method to use and store in object cv

#Make sure to run this cell before proceeding to Questions 9, 10 and 11
```

```
cv = LeaveOneOut()
```

In [20]:
```
#Question 9:

#Since Questions 6 - 9 is a regression scenario, we can use Leave One Out Cross Validat
# (LOOCV) to evaluate our model


#Write Python code to evaluate your model from Question 6 using LOOCV

#Use the following hyperparameter settings for your cross_val_score() function call:

#scoring = 'neg_mean_absolute_error'
#cv = cv
#n_jobs = -1

#Store your results in an object called ridge_scores

#Write your answer below:

#use LOOCV to evaluate model
ridge_scores = cross_val_score(ridge_model, X, y, scoring='neg_mean_absolute_error',
                           cv=cv, n_jobs=-1)
```

In [21]:
```
#Just run this cell to view mean absolute error (MAE) stored in your object scores from

mean(absolute(ridge_scores))
```

Out[21]:
```
1.2368924826583882
```

In [ ]:
```

```

In [22]:
```
#Question 10:

#Write Python code to find the difference between the mean absolute score you obtained
# and the mean absolute error after Question 8 above. Do not assign your code to an obj

#Write your answer below:
mean(absolute(ridge_scores)) - mean_absolute_error(y_test, y_pred)
```

Out[22]:
```
0.3213182753333401
```

In [ ]:
```

```

In [23]:
```
#Question 11:

#Using your Ridge() call from Question 6 above, how would you show a friend that
#You can convert a Ridge Regression to regular Linear Regression?What is the one hyperp
#would change?
```

```
#Write Python code to create an instance of the Ridge() function/ train a model that is

#Make the hyperparameter change that converts your Ridge Regression model in Question 6

#Store your results in an object called ridge_model_2

#Write your answer below:

ridge_model_2 = Ridge(alpha = 0, normalize = False, tol = 0.001,solver ='auto', random_
#When alpha=0 the model is equivalent to that which is solved by LinearRegression
```

In [ ]:

```
```

In [24]:

```
#Questions 12 - 13 examine the use of Cross Validation for evaluating a Regression mode

#Run this cell first to answer Questions 12 - 15 below:

from sklearn.model_selection import train_test_split
from sklearn.model_selection import LeaveOneOut          #Load LeaveOneOut from sklearn
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd
from sklearn.model_selection import cross_validate #load cross_validate from sklearn
from sklearn.linear_model import LinearRegression

#Create the Dataframe of numeric variables
#We create a pandas DataFrame that contains three predictor variables, x1, x2, and x3 a


df = pd.DataFrame({'y': [6, 8, 12, 14, 14, 15, 17, 22, 24, 23],
                   'x1': [2, 5, 4, 3, 4, 6, 7, 5, 8, 9],
                   'x2':[-3,3,4,13,11,6,8,7,9,15],
                   'x3': [14, 12, 12, 13, 7, 8, 7, 4, 6, 5]})



#define predictor and response variables
X = df[['x1', 'x2','x3']]
y = df['y']

#define cross-validation method to use
cv = LeaveOneOut()
```

In [ ]:

```
```

In [25]:

```
#Recall that cross_validate() is a multi-metric cross validation option relative to cro

#Recall that you can use various performance metrics with cross_val_score() including t

‘accuracy’
‘balanced_accuracy’
```

```
‘roc_auc’
‘f1’
‘neg_mean_absolute_error’
‘neg_root_mean_squared_error’
‘r2’
```

```
  File "C:\Users\julia\AppData\Local\Temp/ipykernel_30528/671164659.py", line 5
    ‘accuracy’
    ^
SyntaxError: invalid character '‘' (U+2018)
```

In [ ]:

In [ ]:
```
#Question 12:

#Write Python code to create a list of the folloing performance metrics:

'neg_mean_absolute_error'
'neg_root_mean_squared_error'
'r2'

#Save your list in an object called metrics


#Write your answer below:
metrics = ['neg_mean_absolute_error', 'neg_root_mean_squared_error', 'r2']
```

In [ ]:

In [ ]:
```
#Just run this cell to create an instance of Linear Regression

LR_model = LinearRegression()
```

In [ ]:
```
#Question 13:

#Write Python code to use cross_validate() to fit a Linear Regression model of y on X u
#hyperparameters:

#cv = cv
#scoring = metrics

#Remember to include the object LR_model (from the cell above) in your cross_validate()

#Store your results in an object called LR_scores

#Note that you might get a pink warning box when you run this cell. Ignore it.



#Write your answer below:
LR_scores = cross_validate(LR_model, X, y, cv=cv, scoring=metrics)
```

In [ ]:

In [ ]:
```
#Just run this cell to display the average and standard deviation of the MAE

mae_scores = LR_scores['test_neg_mean_absolute_error']

print("Mean mae of %0.2f with a standard deviation of %0.2f" % (absolute(mae_scores.mea
```

In [ ]:
```

```

In [ ]:
```
#Questions 14 - 16 below are about the KNN classification algorithm


#Run this cell first before attempting Questions 14 - 16

import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns #Import Python Seaborn
from seaborn import load_dataset
from sklearn.model_selection import train_test_split

# Load and display the first rows of the penguins dataset

df = load_dataset('penguins')
df.head()
```

In [ ]:
```

```

In [ ]:
```
#Run this cell next to:

#1. Drop NaN values in our dataset
#2. Splitting our DataFrame into features and target variable

df = df.dropna()

X = df[['flipper_length_mm']]
y = df['species']
```

In [ ]:
```

```

In [ ]:

```
#Question 14:

#Using the train_test_split() function, write Python code to split the data in your cel

#Use the follwoing hyperparameters:

#test_size = 0.25
#random_state = 100

#Write your answer below:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_stat
```

In [ ]:

```

```

In [ ]:

```
#Recall the KNN Classifier function has the structure below:
#Do not run this cell. Just note it to help you answer Question 15

KNeighborsClassifier(
    n_neighbors=5,          # The number of neighbours to consider
    weights='uniform',      # How to weight distances (e.g uniform or distance)
    algorithm='auto',       # Algorithm to compute the neighbors
    leaf_size=30,           # The leaf size to speed up searches
    p=2,                    # The power parameter for the Minkowski metric. 2 = Euclide
    metric='minkowski',     # The type of distance to use. This generalizes Euclidean a
    metric_params=None,     # Keyword arguments for the metric function
    n_jobs=None             # How many parallel jobs to run
)
```

In [ ]:

```
#Question 15:


#Copy and paste the entire KNeighborsClassifier function above into this cell to build

#Make the following changes to the function:

#1. Use Manhattan distance rather than Euclidean distance
#2. Use all the cores in your computer CPU
#3. Use 7 nearest neighbors
#4. Use distance as your weights
#5. Use leaf size 15

#Store your entire result in an object called KNN



#Write your answer below:

KNN= KNeighborsClassifier(
    n_neighbors=7,          # The number of neighbours to consider
    weights='distance',      # How to weight distances (e.g uniform or distance)
    algorithm='auto',       # Algorithm to compute the neighbors
    leaf_size=15,           # The leaf size to speed up searches
    p=1,                    # The power parameter for the Minkowski metric. 2 = Euclide
    metric='manhattan',     # The type of distance to use. This generalizes Euclidean a
    metric_params=None,     # Keyword arguments for the metric function
```

```
    n_jobs=-1             # How many parallel jobs to run
)
```

In [ ]:

In [ ]:
```
#Question 16:

#Use your object KNN from Question 15 above to fit a model on X_train and y_train from j



#Write your answer below:
KNN.fit(X_train, y_train)
```

In [ ]:

In [ ]:

In [ ]:
```
#Questions 17 - 20 is about ensemble models

#Run this cell first before attempting Questions 17 - 20.

#Note that it might take a while for !pip install xgboost to complete


#Install xgboost

!pip install xgboost

from sklearn import datasets

# Load the dataset
diabetes = datasets.load_diabetes()

# importing utility modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import datasets

# importing machine learning models for prediction
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
```

In [ ]:

In [ ]:
```
#Make sure you run this cell before you continue to Questions 17 - 20 below:

# Load the dataset
```

```
df = datasets.load_diabetes()

# getting target data from the dataframe

X = df["data"]
y = df["target"]


# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [ ]:

In [ ]:

```
#Question 17:


#Write Python code to initialize :

#1. a Linear Regression model
#2. an XGBoost regression model
#3. a Random Forest regression model
#4. a Gradient Boosting regression model

#using their default parameters

#Store the models as model_1, model_2, model_3 and model_4 respectively

#Write your code below:




model_1 = LinearRegression()
model_2 =  xgb.XGBRegressor()
model_3 = RandomForestRegressor()
model_4= GradientBoostingRegressor()
```

In [ ]:

In [ ]:

```
#Question 18:

#Using the objects model_1, model_2, model_3 and Model_4 from your answer to Question 1
# write Python code to fit a linear regression model, an XGBoost model, a Random Forest
# and a Gradient Boosting model i.e. four(4) models to X_train and y_train obtained fro

#include all your four .fit() calls in this cell, one below the other


#Write your answer below:

model_1.fit(X_train, y_train)
model_2.fit(X_train, y_train)
model_3.fit(X_train, y_train)
model_4.fit(X_train, y_train)
```

In [ ]:

In [ ]:

```
#Question 19:


#Using model_1, model_2, model_3 and Model_4 from your answer to Question 17,
# write Python code to predict outputs for X_test a linear regression model, an XGBoost
# and a Gradient Boosting model i.e. four(4) predictions

#Store the predictions of model_1 in pred_1
#Store the predictions of model_2 in pred_2
#Store the predictions of model_3 in pred_3
#store the predictions of model_4 in pred_4

#include all your four .predict() calls in this cell, one below the other

#Write your answer below:
pred_1 = model_1.predict(X_test)
pred_2 = model_2.predict(X_test)
pred_3 = model_3.predict(X_test)
pred_4= model_4.predict(X_test)
```

In [ ]:

In [ ]:

```
#Question 20:

#Using pred_1, pred_2, pred_3 and pred_4 from Question 19 above,
#write Python code to display the final predictions across your four models using the a

#Write your answer below:
pred_final = (pred_1+pred_2+pred_3+pred_4)/4.0


#display final predictions arr
print("pred_final ", pred_final)

print(" ")
```

```
# printing the mean squared error between real value and predicted value
print("mean squared error ", mean_squared_error(y_test, pred_final))
```

In [ ]:

```

```

In [ ]:

```
#-----------------------------------End of Final Exam /Exam 2 - Part 2---------------
```