1. Using the original data, df, training (X\_train, y\_train) and test (X\_test, y\_test) sets have been created. Complete the code using the data sets in the appropriate places.

**Complete the code to return the output**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train,y\_train)

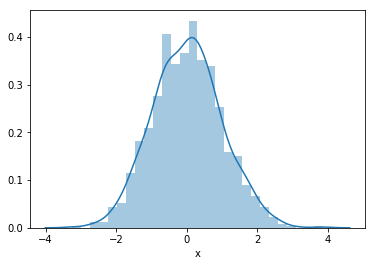
**predictions = lin\_reg.predict(X\_test, y\_test)**

**print("Mean squared error: %.2f" % mean\_squared\_error(lin\_reg, predictions))**

**predictions = lin\_reg.predict(X\_test)**

**print("Mean squared error: %.2f" % mean\_squared\_error(y\_test, predictions))**

1. Calculate the mean of the array x, whose distribution is shown in the plot.



## Complete the code to return the output

import numpy as np

result = np.mean(x)

print(result)

1. A (Poisson) generalized linear model is fitted on the dataset score and stored in this session as model. Using model, apply the prediction method on the test dataset.

**Complete the code to return the output**

import statsmodels.api as sm

from statsmodels.formula.api import glm

model = glm(

'goal ~ player',

data = score,

family = sm.families.Poisson()

).fit()

predictions = model.predict(test)

print(predictions)

Available in this session are the training data X\_train and y\_train for feature and target variables, respectively; as well as the testing data X\_test and y\_test for feature and target variables, respectively.

Using sklearn, fit a classification random forest model on X\_train and y\_train.

## Complete the code to return the output

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

**model = SKlear(n\_estimators=10, random\_state=1)**

**model.predict(X\_train, y\_train)**

**model = RandomForestClassifier(n\_estimators=10, random\_state=1)**

**model.fit(X\_train, y\_train)**

y\_pred = model.predict(X\_test)

print(accuracy\_score(y\_test, y\_pred))

Available in this session are the training data X\_train and y\_train for feature and target variables, respectively; as well as the testing data X\_test and y\_test.

Using sklearn, fit a logistic regression model on X\_train and y\_train.

## Complete the code to return the output

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(random\_state=1)

**model.predict(X\_train, y\_train)**

**model.fit(X\_train, y\_train)**

print(model.score(X\_test, y\_test))

Create and fit random forest classifier with 15 trees to predict whether a patient has or does not have heart disease. The data has already been split into train and test sets (X\_train, X\_test, y\_train, y\_test).

**Complete the code to return the output**

**from sklearn.metrics import RandomForestClassifier**

**from sklearn.ensemble import RandomForestClassifier**

from sklearn.metrics import accuracy\_score

model = RandomForestClassifier(n\_estimators=15,

random\_state=1)

**model.predict(X\_train, y\_train)**

**model.fit(X\_train, y\_train)**

y\_pred = model.predict(X\_test)

print(accuracy\_score(y\_test, y\_pred))

A college is looking at high-school exam scores to determine whether or not new students will pass their first year.

What machine learning model is best suited for this problem?

A: Logistic Regression

Which one of the following statements best describes k-means clustering and its utility?

* 

K-means clustering reduces the amount of variables in the model to determine which features have the most variability.

* 

K-means clustering groups data into relatively distinct groups by using a pre-determined number of clusters and iterating cluster assignments.

* 

K-means clustering is a supervised learning algorithm that predicts continuous variables as a function of categorical variables (called clusters).

* 

K-means clustering is a supervised learning algorithm that predicts categorical variables (called clusters) as a function of continuous variables.

In order to compare the performance of two different linear models, calculate the mean squared error for each model. Perform the calculation based on actual test data values against pred1 and pred2 - the predictions for each model.

**Complete the code to return the output**

from sklearn import metrics

**mse1=metrics.predict(actual, pred\_1)**

**mse2=metrics.predict(actual, pred\_2)**

**mse1=metrics.mean\_squared\_error(actual, pred\_1)**

**mse2=metrics.mean\_squared\_error(actual, pred\_2)**

print("MSE for model 1 is", str(mse1.round(2)),

"and for model 2 is", str(mse2.round(2)))

The Pandas DataFrame df is loaded on your working session. Using the scipy package, implement k-means clustering (with two centroids) on the columns x\_scaled and y\_scaled.

**Complete the code to return the output**

from scipy.cluster.vq import kmeans, vq

**clusters = mean(**

**clusters = kmeans(**

df[['x\_scaled', 'y\_scaled']],

2

)

print(clusters)

How does a random forest improve upon a decision tree?

A random forest reduces variance by combining decision trees that…

1. 

have been trained on different bootstrap samples of the training set.

1. 

have been trained sequentially where each tree improves on the previous tree.

1. 

each use all the available features in the training set.

1. 

each have the highest accuracy on the same training set.

A group of scientists are looking at weather patterns to identify groups of similar hurricanes.

What model should they use for this task?

* 

Decision trees.

* 

Logistic regression.

* 

K-means clustering

* 

Ada boost.

Which one of the following statements describe supervised learning?

* 

A machine learning problem where we seek to understand whether observations fit into distinct groups.

* 

A machine learning problem where we want to decrease the number of variables so that we only preserve the ones with most variability.

* 

A machine learning problem where we fit a model to a response variable as a function of a set of predictor variables.

* 

A machine learning problem that deals with the importing and cleaning of a dataset, such that it is tidy.

The following is a preview of the data df. One-hot encode the Animal variable.

Age Animal Weight

0 22 dog 68.7

1 38 cat 89.8

2 56 cat 77.4

3 21 dog 77.3

4 43 fish 82.8

## Complete the code to return the output

import pandas as pd

**cluster = pd.predict(df, columns=['Animal'])**

**encoded = pd.get\_dummies(df, columns=['Animal'])**

print(encoded)

x\_train and y\_train are explanatory and response variables, respectively, used to fit a linear model, stored in the variable reg. Use reg to predict the value of the response variable y\_test as a function of the explanatory variable x\_test.

**Complete the code to return the output**

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

reg = LinearRegression()

reg.fit(x\_train, y\_train)

predictions = reg.predict(x\_test)

print(predictions)

1)Fit a Logistic Regression model to the binary data.

x y

0 0

1 1

2 0

3 0

...

**Complete the code to return the output**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(random\_state=1)

model.fit(x, y)

print(model.score(x, y))

2) Available in this session are the training data X\_train and y\_train for feature and target variables, respectively; as well as the testing data X\_test and y\_test for feature and target variables, respectively.

Using sklearn, fit a classification decision tree model on X\_train and y\_train.

**Complete the code to return the output**

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

**model = LogisticRegression(max\_depth=4, random\_state=1)**

**model = DecisionTreeClassifier(max\_depth=4, random\_state=1)**

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(accuracy\_score(y\_test, y\_pred))

3) A GradientBoostingClassifier model is fitted on the training data X\_train and y\_train, and stored as model. Use model and the X\_test feature data to predict values for the response variable, and store it in y\_pred.

**Complete the code to return the output**

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score

model = GradientBoostingClassifier(n\_estimators=300, max\_depth=1, random\_state=1)

model.fit(X\_train, y\_train)

**y\_pred = predict.model(X\_test)**

**y\_pred = model.predict(X\_test)**

print(accuracy\_score(y\_test, y\_pred))

4) A LogisticRegression model is fitted on the training data X\_train and y\_train, and stored in model. Use model and the X\_test feature data to predict values for the response variable, and store it in y\_pred. Then get the model score using X\_test and y\_test.

**Complete the code to return the output**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(random\_state=1)

model.fit(X\_train, y\_train)

**y\_pred = model.predict()**

**score = model.(X\_test, y\_test)**

**y\_pred = model.predict(X\_test)**

**score = model.score(X\_test, y\_test)**

print(score)

5) Available in this session are the training data X\_train and y\_train for feature and target variables, respectively; as well as the testing data X\_test and y\_test.

Using sklearn, fit a logistic regression model on X\_train and y\_train.

**Complete the code to return the output**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(random\_state=1)

model.fit(X\_train, y\_train)

print(model.score(X\_test, y\_test))

6)

Overfitting means:

* 

We have built a model that will only perform well on previously unseen data.

* 

We have built a model that will only perform well on the training data.

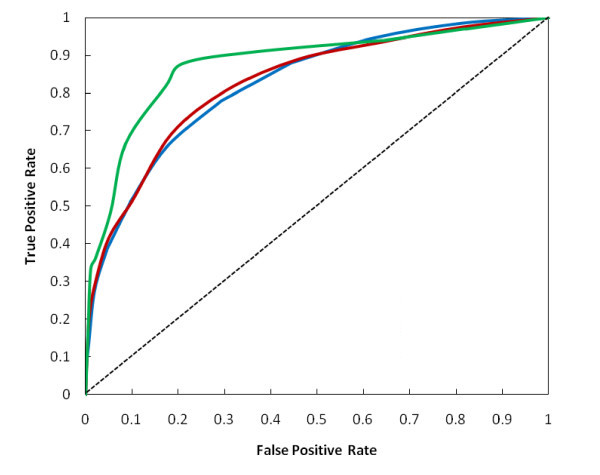
* 

We have spent a lot of time building a model with no significant improvements.

* 

We have built a model that will give reasonable results on any data.

7) As part of a project performing credit risk modeling on a banking DataFrame - you created a few models and wanted to check their ROC curves and compare between them.



Which model should you use and how would you quantify your decision?

Graphical user interface, text, application, email

Description automatically generated

8) A linear model has been fitted to training data and evaluated on the test data. Use the model to predict the output on new\_data.

**Complete the code to return the output**

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

new\_data = np.array([[8, 9]]).reshape(-1,1)

**predictions = model.fit(new\_data)**

**predictions = model.predict(new\_data)**

print(predictions)

9) Calculate the mean of the array x, whose distribution is shown in the plot.

Histogram

Description automatically generated

**Complete the code to return the output**

import numpy as np

result = np.mean(x)

print(result)

10) Before we fit a model to our data we should consider centering and scaling the data so that:

* 

It's easier to interpret the model output because the variables are the same values

* 

A feature does not have more influence on the model because of larger or smaller values

* 

We can remove outliers from the data because they will all be on the same range

* 

We can use both the original and scaled version of the variables in our model

11) Which of the following metrics would not be used when assessing the performance of a classification model?

* 

Area under the curve.

* 

Precision.

* 

Recall.

* 

Median absolute error.

12) You have created a k-means clustering model customer\_model. The model groups customers into 5 classes. You now receive new data describing a set of 5 customers (X\_new). Determine to which clusters they belong.

**Complete the code to return the output**

**predictions = kmeans(X\_new)**

**predictions = customer\_model.predict(X\_new)**

print(predictions)

13) What is the main difference between hierarchical clustering and k-means clustering?

* 

K-means clustering is a supervised learning algorithm, while hierarchical clustering is an unsupervised learning algorithm.

* 

K-means clustering is an unsupervised learning algorithm, while hierarchical clustering is a supervised learning algorithm.

* 

K-means clustering requires a pre-specified number of clusters, while hierarchical clustering creates multiple clusters across all of the data.

* 

K-means clustering yields a dendrogram representation of the results, while hierarchical clustering requires a pre-specified number of clusters.

14) Using the original data, df, training (X\_train, y\_train) and test (X\_test, y\_test) sets have been created. Complete the code using the data sets in the appropriate places.

**Complete the code to return the output**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train,y\_train)

**predictions = lin\_reg.predict(X\_test, y\_test)**

**print("Mean squared error: %.2f" % mean\_squared\_error(, ))**

**predictions = lin\_reg.predict(X\_test)**

**print("Mean squared error: %.2f" % mean\_squared\_error(y\_test, predictions))**

15) Grid Search is:

* 

The process of testing all combinations of possible hyper-parameters to find the most suitable values.

* 

The process of randomly testing combinations of possible hyper-parameters to find suitable values.

* 

The process of re-fitting a model to several random subsets of the original data.

* 

The process of fitting a model to the complete data set to find the final model parameters.

The data df contains a total of 32 rows. Create a train/test split, where 80% of the data is in the training data.

**Complete the code to return the output**

from sklearn import model\_selection

**X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.8, random\_state=42)**

**X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, train\_size=0.8, random\_state=42)**

print("X\_train shape: ", X\_train.shape)

print("X\_test shape: ", X\_test.shape)

print("y\_train shape: ",y\_train.shape)

print("y\_test shape: ",y\_test.shape)

A dataset has been prepared for you and split into test and training sets (X\_train, X\_test, y\_train, y\_test).

Use sklearn to fit a classification gradient boosting model on the training data **with 300 estimators and 0.01 learning rate**

## Complete the code to return the output

from sklearn.ensemble import GradientBoostingClassifier

**model = GradientBoostingClassifier(n\_estimators=300, max\_depth=0.01, random\_state=42)**

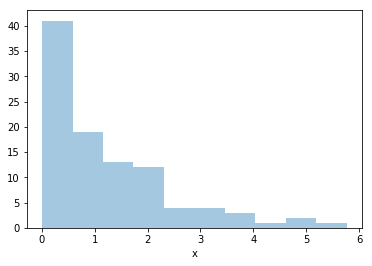
**model = GradientBoostingClassifier(n\_estimators=300, learning\_rate=0.01, random\_state=42)**

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(model.score(X\_test, y\_test))

Consider the variable x in the Pandas DataFrame df shown in the plot below. Apply a Box-Cox Transformation to the variable x.



## Complete the code to return the output

from sklearn.preprocessing import PowerTransformer

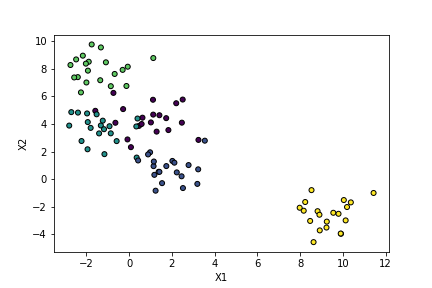
**log = PowerTransformer(method=Box-Cox)**

**log = PowerTransformer(method='box-cox')**

df['log\_x'] = log.fit\_transform(df[['x']])

print(df['log\_x'].head())

Consider a dataset, X with 2 features and 100 samples. Use K-Means clustering to group the data into 5 clusters.



## Complete the code to return the output

**from sklearn.scipy.cluster.vq import KMeans**

**from sklearn.cluster import KMeans**

**kmeans = KMeans(, random\_state=42)**

**kmeans = KMeans(n\_clusters=5, random\_state=42)**

kmeans.fit(X)

centers = kmeans.cluster\_centers\_

print(centers)

A LogisticRegression model, clf has been fitted to data where the target variable is binary.

Use an appropriate metric to determine the percentage of correct predictions made.

## Complete the code to return the output

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25, random\_state=42)

clf.fit(X\_train,y\_train)

y\_pred=clf.predict(X\_test)

**score = metrics.clf(y\_test, y\_pred)**

**score = metrics.accuracy\_score(y\_test, y\_pred)**

print(score)

You are supplied with continuous variables in the NumPy arrays X and y.

Build an appropriate model on the target array y using the features of X.

Display the coefficient and intercept for the resulting model.

## Complete the code to return the output

from sklearn.linear\_model import LinearRegression

**model = LogisticRegression(random\_state=1)**

**model = LinearRegression()**

model.fit(X,y)

**print("Regression coefficients: {}".format(X))**

**print("Regression intercept: {}".format(y))**

**print("Regression coefficients: {}".format(model.coef\_))**

**print("Regression intercept: {}".format(model.intercept\_))**

You are supplied with a dataset assigned to the variable scaled\_data.

This dataset exhibits high dimensionality, you need to reduce it to **3** features.

**Complete the code to return the output**

from sklearn.decomposition import PCA

**reducing\_fn = PCA**

**reducing\_fn = PCA(n\_components=3)**

**Scale\_data**

**reducing\_fn.fit(scaled\_data)**

reduced\_ft = reducing\_fn.transform(scaled\_data)

print(reduced\_ft.shape)

Consider the first five rows of the data frame df shown below. Apply a pre-processing step to standardize all numeric features.

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

1 5.1 3.5 1.4 0.2 setosa

2 4.9 3.0 1.4 0.2 setosa

3 4.7 3.2 1.3 0.2 setosa

4 4.6 3.1 1.5 0.2 setosa

5 5.0 3.6 1.4 0.2 setosa

6 5.4 3.9 1.7 0.4 setosa

**Complete the code to return the output**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

**df\_scaled = pd.DataFrame(scaler(df), columns=df.columns)**

**df\_scaled = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)**

print(df\_scaled.head())

Consider the variable x in the Pandas DataFrame df shown in the plot below. Note that the data contains positive and negative values. Apply a suitable transformation to the data.

Chart, histogram

Description automatically generated

**Complete the code to return the output**

from sklearn.preprocessing import PowerTransformer

**log = PowerTransformer(method='box-cox')**

**log = PowerTransformer(method='yeo-johnson')**

df['log\_x'] = log.fit\_transform(df[['x']])

print(df['log\_x'].head())

Determine if 50, 150 or 250 is the best value for the n\_estimators hyperparameter of a random forest classifier.

**Complete the code to return the output**

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

model\_params = {'n\_estimators': [50, 150, 250]}

rf = RandomForestClassifier(random\_state=42)

**clf = (rf, , cv=5)**

**clf = GridSearchCV(rf, model\_params, cv=5)**

clf.fit(X\_train, y\_train)

print(clf.best\_params\_)

A dataset has been prepared for you and split into test and training sets (X\_train, X\_test, y\_train, y\_test).

Show the **importance of each feature** in the gradient boosting model that is fitted to the data

**Complete the code to return the output**

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

**importances = gradient boosting model(**

**importances = model.feature\_importances\_**

**feature\_importances = pd.Series(, wine\_feat.columns)**

**feature\_importances = pd.Series(importances, wine\_feat.columns)**

print(feature\_importances)

The test\_data dataset contains grades from a midterm exam, as well as the time taken by the students to write the exam. There was no time limit on the test so students had no time pressure to finish. Fit a Ridge Regression model with alpha=0.1 and determine the R-squared value for the model.

>>> test\_data.head()

Grade Time

0 85.0 127

1 79.2 152

2 74.2 115

3 71.7 152

4 80.0 152

**Complete the code to return the output**

from sklearn.linear\_model import Ridge

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

X = test\_data[['Grade']]

y = test\_data['Time']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25, random\_state=42)

**ridge =**

**model.fit(X\_train, y\_train)**

**ridge = Ridge(alpha=0.1)**

**ridge.fit(X\_train, y\_train)**

y\_pred = ridge.predict(X\_test)

**r\_squared =**

**r\_squared = ridge.score(X\_test, y\_test)**

print("R-squared: %.3f" % r\_squared)

Use five fold cross-validation to evaluate a fitted k-nearest neighbors classifier. X contains the features and y is the target variable.

**Complete the code to return the output**

from sklearn import model\_selection

**scores = model\_selection.knearest(knn, X, y, cv=5, scoring='accuracy')**

**scores = model\_selection.cross\_val\_score(knn, X, y, cv=5, scoring='accuracy')**

print(scores)

# Create a pd.Series of features importances

importances = pd.Series(data=rf.feature\_importances\_,

                        index= X\_train.columns)

# Sort importances

importances\_sorted = importances.sort\_values()

# Draw a horizontal barplot of importances\_sorted

importances\_sorted.plot(kind='barh', color='lightgreen')

plt.title('Features Importances')

plt.show()

# Import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier

# Import AdaBoostClassifier

from sklearn.ensemble import AdaBoostClassifier

# Instantiate dt

dt = DecisionTreeClassifier(max\_depth=2, random\_state=1)

# Instantiate ada

ada = AdaBoostClassifier(base\_estimator=dt, n\_estimators=180, random\_state=1)

# Import mean\_squared\_error as MSE

from sklearn.metrics import mean\_squared\_error as MSE

# Compute MSE

mse\_test = MSE(y\_test, y\_pred)

# Compute RMSE

rmse\_test = mse\_test\*\*(1/2)

# Print RMSE

print('Test set RMSE of gb: {:.3f}'.format(rmse\_test))

# Import GradientBoostingRegressor

from sklearn.ensemble import GradientBoostingRegressor

# Instantiate sgbr

sgbr = GradientBoostingRegressor(max\_depth=4,

                                 subsample=0.9,

                                 max\_features=0.75,

                                 n\_estimators=200,

                                 random\_state=2)

# Import GridSearchCV

from sklearn.model\_selection import GridSearchCV

# Instantiate grid\_dt

grid\_dt = GridSearchCV(estimator=dt,

                       param\_grid=params\_dt,

                       scoring='roc\_auc',

                       cv=5,

                       n\_jobs=-1)

# Import roc\_auc\_score from sklearn.metrics

from sklearn.metrics import roc\_auc\_score

# Extract the best estimator

best\_model = grid\_dt.best\_estimator\_

# Predict the test set probabilities of the positive class

y\_pred\_proba = best\_model.predict\_proba(X\_test)[:,1]

# Compute test\_roc\_auc

test\_roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

# Print test\_roc\_auc

print('Test set ROC AUC score: {:.3f}'.format(test\_roc\_auc))

# Import GridSearchCV

from sklearn.model\_selection import GridSearchCV

# Instantiate grid\_rf

grid\_rf = GridSearchCV(estimator=rf,

                       param\_grid=params\_rf,

                       scoring='neg\_mean\_squared\_error',

                       cv=3,

                       verbose=1,

                       n\_jobs=-1)

A random forest model has been fitted to train data.

Use the results from a random forest classifier to determine the importance of each feature in determining whether a patient does or does not have heart disease.

## Complete the code to return the output

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=15,

random\_state=1)

model.fit(X\_train, y\_train)

# Create a DataFrame with the feature importances

feature\_importances = pd.DataFrame(

**{"feature": list(X.columns), "importance": list(Y.columns)}**

**{"feature": list(X.columns), "importance": model.feature\_importances\_}**

).sort\_values("importance", ascending=False)

sns.barplot(data=feature\_importances, x="importance", y="feature")

plt.show()

Available in this session are the training (X\_train, y\_train) and test (X\_test, y\_test) sets.

Implement a Lasso Regression model with alpha equal to 0.01. Determine the RMSE achieved by the model.

## Complete the code to return the output

import numpy as np

from sklearn.metrics import mean\_squared\_error, r2\_score

**from sklearn.linear\_model import LassoRegression**

**from sklearn.linear\_model import Lasso**

**lasso\_model = LassoRegression(Alpha=0.01)**

**lasso\_model = Lasso(alpha=0.01)**

lasso\_model.fit(X\_train, y\_train)

lasso\_predictions = lasso\_model.predict(X\_test)

print("RMSE: ", np.sqrt(mean\_squared\_error(y\_test,lasso\_predictions)))

You are supplied with 2 sets of variables; y\_pred are predicted using the model supplied and y\_test are the actual response values.

Print the main classification metrics for the model.

## Complete the code to return the output

from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics

model = DecisionTreeClassifier(max\_depth=4, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

**print(metrics.y\_pred)**

**print(metrics.classification\_report(y\_test, y\_pred))**

Create a bagging classifier using 10 decision tree classifiers. The data has already been split into train and test sets (X\_train, X\_test, y\_train, y\_test)

**Complete the code to return the output**

**from sklearn. import DecisionTreeClassifier**

**from sklearn. import BaggingClassifier**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import BaggingClassifier**

from sklearn.metrics import accuracy\_score

dt = DecisionTreeClassifier(random\_state=1)

bc = BaggingClassifier(base\_estimator=dt,

n\_estimators=10,

random\_state=1)

**.fit(X\_train, y\_train)**

**bc.fit(X\_train, y\_train)**

**y\_pred = .predict(X\_test)**

**y\_pred = bc.predict(X\_test)**

print(accuracy\_score(y\_test, y\_pred))

Using the binary predicted values y\_pred, and known actual values, y\_test, determine the size of the area under the ROC curve.

**Complete the code to return the output**

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

clf=LogisticRegression(solver='newton-cg', random\_state=42)

clf.fit(X\_train,y\_train)

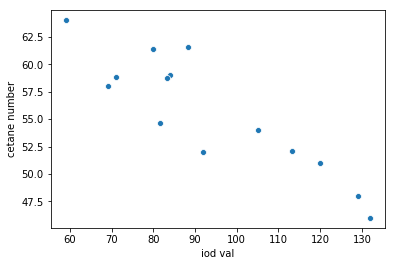
y\_pred=clf.predict(X\_test)

**size = metrics.accuracy\_score(y\_test, y\_pred)**

**size = metrics.roc\_auc\_score(y\_test, y\_pred)**

print(size)

The scatterplot shows data for a sample of 14 biofuels. Fit a linear regression model and print the intercept and coefficient.



* x = iodine value (g)
* y = cetane number

**Complete the code to return the output**

from sklearn.linear\_model import LinearRegression

model = LinearRegression(fit\_intercept=True)

model.fit(x, y)

**print("Regression coefficients: {}".format(model.coef))**

**print("Regression intercept: {}".format(model.intercept))**

**print("Regression coefficients: {}".format(model.coef\_))**

**print("Regression intercept: {}".format(model.intercept\_\_))**

A random forest model has been fitted to train data.

Use the results from a random forest classifier to determine the importance of each feature in determining whether a patient does or does not have heart disease.

## Complete the code to return the output

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=15,

random\_state=1)

model.fit(X\_train, y\_train)

# Create a DataFrame with the feature importances

feature\_importances = pd.DataFrame(

**{"feature": list(X.columns), "importance": model.feature\_importances}**

**{"feature": list(X.columns), "importance": model.feature\_importances\_}**

).sort\_values("importance", ascending=False)

sns.barplot(data=feature\_importances, x="importance", y="feature")

plt.show()

Available in your working session is the dataset scaled\_samples. Instantiate a principal component analysis model object with 2 components, and fit the model to the scaled\_samples object.

**Complete the code to return the output**

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

**pca.fit()**

**pca.fit(scaled\_samples)**

pca\_features = pca.transform(scaled\_samples)

print(pca\_features.shape)

The Pandas DataFrame df is loaded on your working session. Using the scipy package, compute the cluster distance (with linkage method complete) between the observations the columns x\_scaled and y\_scaled.

**Complete the code to return the output**

from scipy.cluster.hierarchy import linkage

distances = linkage(

df[['x\_scaled', 'y\_scaled']],

**,**

**method = 'complete',**

metric = 'euclidean'

)

print(distances[:5])

An array X with two features has been loaded for you along with the target variable array y. Fit a multiple linear regression model with **interaction terms** on the target variable y as a function of the feature variables contained in X.

**Complete the code to return the output**

import pandas as pd

from sklearn.linear\_model import LinearRegression

**from sklearn.preprocessing import PowerTransformer**

**from sklearn.preprocessing import PolynomialFeatures**

**interaction\_term = PowerTransformer(degree=2, interaction\_only=X, include\_bias=False)**

**interaction\_term = PolynomialFeatures(degree=2, interaction\_only=True, include\_bias=False)**

X\_inter = interaction\_term.fit\_transform(X)

model = LinearRegression()

**model.fit(X,y)**

**model.fit(X\_inter,y)**

print("Regression coefficients: {}".format(model.coef\_))

A dataset has been prepared for you and fed into a random forest model.

Use sklearn to show the **predicted probabilities** of a new data point belonging to each class.

>>> print(new)

Alcohol Malic.acid Phenols Flavanoids

13.64 3.10 2.70 3.01

## Complete the code to return the output

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

**print(model.new)**

**print(model.predict\_proba(new))**

You have a New York house price dataset named housing that contains the following columns. Label Encode the Borough column so that each Borough has an associated number to it.

LotArea Borough SalePrice

0 220 Bronx 150000

1 140 Manhattan 450000

2 200 Queens 190000

3 150 Brooklyn 240000

4 340 Staten Island 200000

## Complete the code to return the output

**from sklearn.preprocessing import PowerTransformer**

**from sklearn.preprocessing import LabelEncoder**

**le = PowerTransformer()**

**housing['Borough'] = le.(housing[])**

**le = LabelEncoder()**

**housing['Borough'] = le.fit\_transform(housing['Borough'])**

print(housing)