Python Machine Learning Mini Course

https://machinelearningmastery.com/python-machine-learning-mini-course/
(https://machinelearningmastery.com/python-machine-learning-mini-course/)

I'm using a mix of descriptions used in the course along with my own commentary in code comments and markdown blocks to learn and help display understanding

Lesson 01

In [1]:
```
# Lesson 01
import sys
import scipy
import numpy as np
import matplotlib as plt
import pandas as pd
import sklearn as sk
# giving the imports nicknames; removed the prints to keep it cleaner
```

Lesson 02

In [2]:
```
# Lesson 02
myarray = np.array([[1, 2, 3], [4, 5, 6]])
rownames = ['a', 'b']
colnames = ['one', 'two', 'three']
mydataframe = pd.DataFrame(myarray, index=rownames, columns=colnames)

print(mydataframe)
```

```
   one  two  three
a    1    2      3
b    4    5      6
```

Lesson 03

In [3]:
```
# Lesson 03
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indian
s-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'clas
s']
data = pd.read_csv(url, names=names)

print(data.shape)
```

```
(768, 9)
```

names: list of the columns being imported/read as a csv

9 columns, 768 rows

Below are other methods of importing a csv file

In [4]:
```python
# import csv
# with open('example.csv', 'r') as file:
#  csv_reader = csv.reader(file)

# import numpy as np
# data = np.loadtxt('example.csv', delimiter=',')

# import pandas as pd
# data = pd.read_csv('example.csv')
```

Lesson 04

In [5]:
```python
# Lesson 04
# Statistical Summary
data = pd.read_csv(url, names=names)
data.describe()
```

Out[5]:

	preg	plas	pres	skin	test	mass	pedi	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.

In [6]:
```python
data.head()
```

Out[6]:

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [7]:
```
data.shape
```
Out[7]:
```
(768, 9)
```

In [8]:
```
data.dtypes
```
Out[8]:
```
preg        int64
plas        int64
pres        int64
skin        int64
test        int64
mass      float64
pedi      float64
age         int64
class       int64
dtype: object
```

In [9]:
```
data.describe()
```
Out[9]:

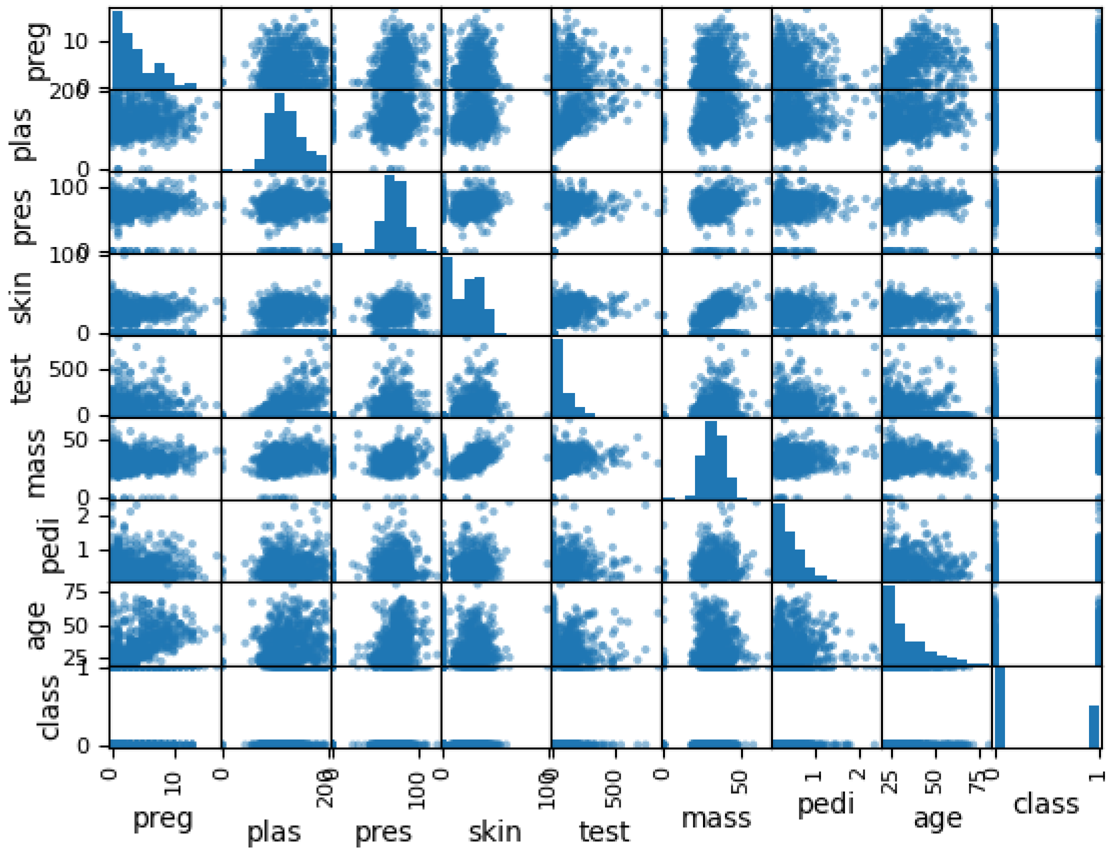
	preg	plas	pres	skin	test	mass	pedi	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.2
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.0

In [10]:
```
data.corr()
```
Out[10]:

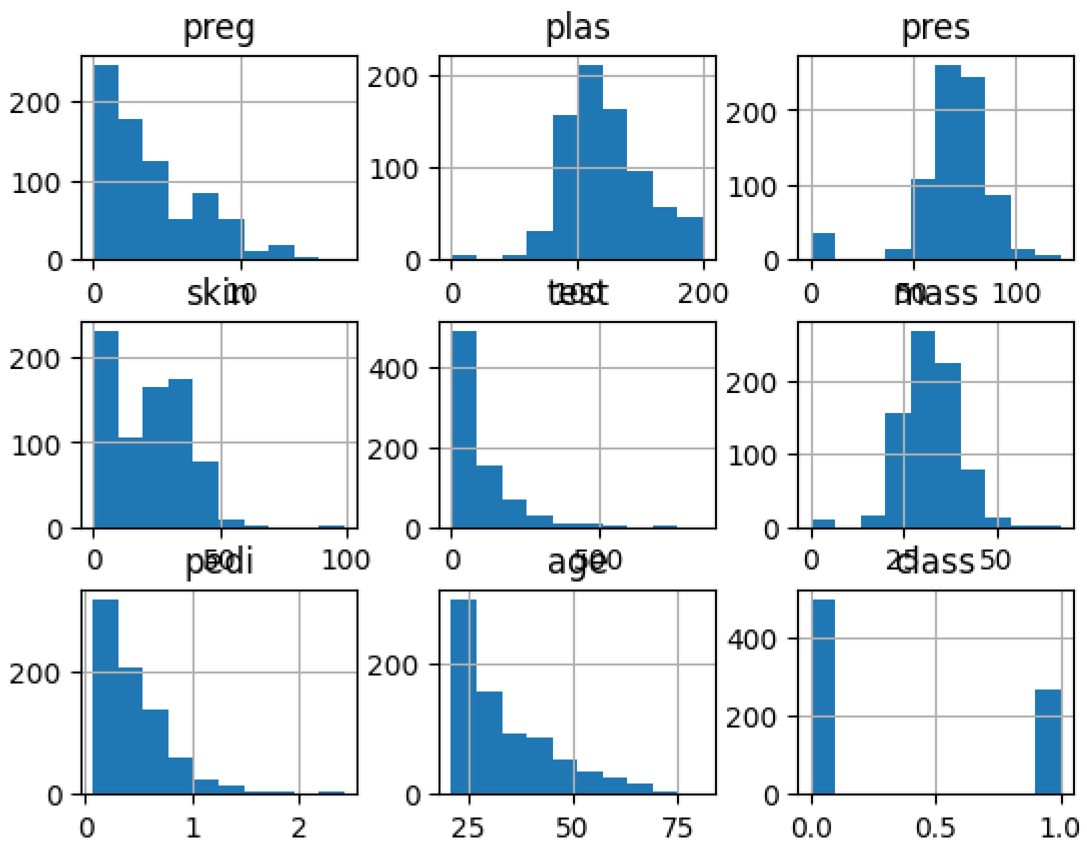
	preg	plas	pres	skin	test	mass	pedi	age	clas
preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.22189
plas	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.46658
pres	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.06506
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.07475
test	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.13054
mass	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.29269
pedi	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.17384
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.23835
class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.00000

Lesson 05

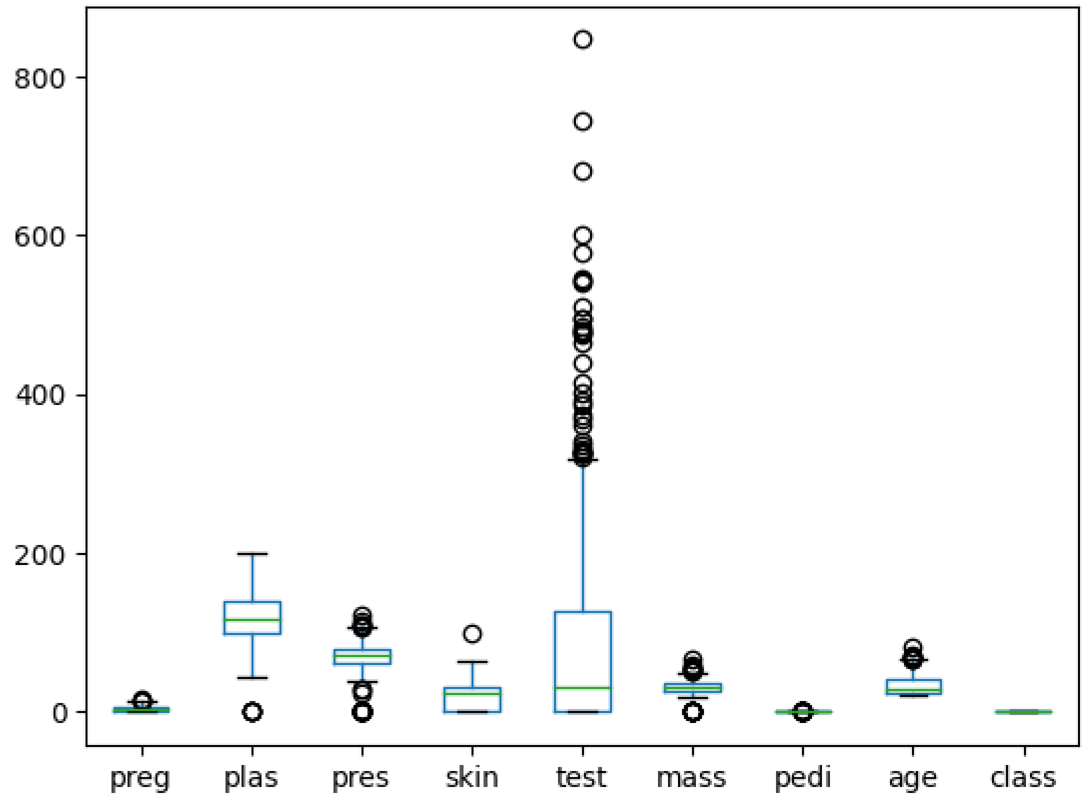
In [11]:
```
# Lesson 05
# Scatter Plot Matrix
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix

scatter_matrix(data)
plt.show()
```



In [12]:
```
# Histogram of the data
data.hist()
plt.show()
```



In [13]:
```
# Box plot of the data
data.plot(kind='box')
plt.show()
```



Lesson 06

In [14]:
```
# Lesson 06
# Prepare the data for modeling by preprocessing it
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler

dataframe = pd.read_csv(url, names=names)
array = dataframe.values

# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]

# calculate params needed to standardize data
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)

# summarize transformed data
np.set_printoptions(precision=3)

print(rescaledX[0:5,:])
```

```
[[ 0.64   0.848  0.15   0.907 -0.693  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.604 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.921 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.485 -0.02 ]]
```

The above snippet:

- Loads the Pima Indians onset of diabetes dataset
- Calculates the parameters needed to standardize the data, transforming the data to have a mean of 0 and a standard deviation of 1
- Then creates a standardized copy of the input data. StandardScaler() is used to do that and assign it to a new variable

In [15]:
```
# Standardize numerical data (e.g. mean of 0 and standard deviation of 1) usin
g the scale and center options.

from sklearn.preprocessing import MinMaxScaler

# Normalize data (0 to 1)
minmax_scaler = MinMaxScaler(feature_range=(0, 1))
normalizedX = minmax_scaler.fit_transform(X)

# Summarize transformed data
print(normalizedX[0:5,:])
```

```
[[0.353 0.744 0.59  0.354 0.    0.501 0.234 0.483]
 [0.059 0.427 0.541 0.293 0.    0.396 0.117 0.167]
 [0.471 0.92  0.525 0.    0.    0.347 0.254 0.183]
 [0.059 0.447 0.541 0.232 0.111 0.419 0.038 0.   ]
 [0.    0.688 0.328 0.354 0.199 0.642 0.944 0.2  ]]
```

The difference between MinMaxScaler() and StandardScaler() is that MinMaxScaler will standardize the data between a range of [0,1] while StandardScaler will standardize it so it has a mean of 0 and standard deviation of 1

Lesson 07

In [16]:
```
# Lesson 07

# Evaluate using Cross Validation
from pandas import read_csv
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression

dataframe = read_csv(url, names=names)
array = dataframe.values
# split the data's features and target values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = LogisticRegression(solver='liblinear')
# test and training sets are handled by cross_val_score
results = cross_val_score(model, X, Y, cv=kfold)

print(f"Accuracy: {results.mean()*100:.3f}% ({results.std()*100:.3f}%)")
```

```
Accuracy: 77.086% (5.091%)
```

The lesson suggests to attempt the following actions:

- Split a dataset into training and test sets.
- Estimate the accuracy of an algorithm using k-fold cross validation.
- Estimate the accuracy of an algorithm using leave one out cross validation.

It's kind of there already, just a few minor changes to be made

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_m
atrix

# Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, rand
om_state=7)

# Train the model on the training set
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)

# Predict the outcomes on the test set
Y_pred = model.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {accuracy * 100:.3f}%")
print(f"\nClassification Report:\n{classification_report(Y_test, Y_pred)}")
print(f"\nConfusion Matrix:\n{confusion_matrix(Y_test, Y_pred)}")
```

```
Accuracy: 79.221%

Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.94      0.85        97
         1.0       0.84      0.54      0.66        57

    accuracy                           0.79       154
   macro avg       0.81      0.74      0.76       154
weighted avg       0.80      0.79      0.78       154


Confusion Matrix:
[[91  6]
 [26 31]]
```

A similar method for the above cell was used in Step 5.1 of the other assignment. I also came back to this after Lesson 08 to apply the evaluation metrics

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)

In [18]:
```
# Estimate the accuracy of an algorithm using k-fold cross validation

kfold = KFold(n_splits=10, random_state=7, shuffle=True)
results = cross_val_score(model, X, Y, cv=kfold)
print(results)
```

```
[0.831 0.714 0.714 0.779 0.792 0.766 0.688 0.857 0.803 0.763]
```

In [19]:
```
from sklearn.model_selection import LeaveOneOut

# Estimate the accuracy of an algorithm using leave one out cross validation.
loo = LeaveOneOut()
results_loo = cross_val_score(model, X, Y, scoring='accuracy', cv=loo)
print(f"Accuracy: {results_loo.mean()*100:.3f}% ({results_loo.std()*100:.3
f}%)")
```

```
Accuracy: 76.823% (42.196%)
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html)

Example under "LOOCV to Evaluate Machine Learning Models" section

https://machinelearningmastery.com/loocv-for-evaluating-machine-learning-algorithms/ (https://machinelearningmastery.com/loocv-for-evaluating-machine-learning-algorithms/)

Lesson 08

In [20]:
```
# Lesson 08

# Cross Validation Classification LogLoss
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# I added shuffle=True to the kFold, or else it would error out
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = LogisticRegression(solver='liblinear')
scoring = 'neg_log_loss'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
# I had to move one of the ')' to the end. Originally it was in the wrong loca
tion
print(f"Logloss: {results.mean():.3f} ({results.std():.3f})")
```

```
Logloss: -0.494 (0.042)
```

- Practice using the Accuracy and LogLoss metrics on a classification problem.
- Practice generating a confusion matrix and a classification report.
- Practice using RMSE and RSquared metrics on a regression problem.

In [21]:
```
# Practice using the Accuracy and LogLoss metrics on a classification problem.
results_acc = cross_val_score(model, X, Y, cv=kfold, scoring='accuracy')
print(f"Logloss: {results.mean()*100:.3f}% ({results.std()*100:.3f}%)")
```

```
Logloss: -49.367% (4.207%)
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)

https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

Above is a link to different options accepted in scoring for model evaluation.

In [22]:
```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Practice generating a confusion matrix and a classification report.
# This was done in the last cell of the other assignment

# X and Y variables are created a couple cells above
# create an 80/20 split training and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, rand
om_state=7)
model.fit(X_train, Y_train)
predictions = model.predict(X_test)

print(f"\nClassification Report:\n{classification_report(Y_test, prediction
s)}")
print(f"\nConfusion Matrix:\n{confusion_matrix(Y_test, predictions)}")
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.94      0.85        97
         1.0       0.84      0.54      0.66        57

    accuracy                           0.79       154
   macro avg       0.81      0.74      0.76       154
weighted avg       0.80      0.79      0.78       154


Confusion Matrix:
[[91  6]
 [26 31]]
```

In [23]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# practice using RMSE and RSquared metrics on a regression problem.

# references:
# https://machinelearningmastery.com/regression-metrics-for-machine-learning/
# https://www.statology.org/r-squared-in-python/

regression_model = LinearRegression()

# fit the regression model
regression_model.fit(X_train, Y_train)
Y_pred = regression_model.predict(X_test)

# calculate RMSE
rmse = mean_squared_error(Y_test, Y_pred, squared=False)

# calculate RSquared of regression modell
r_squared = regression_model.score(X_train, Y_train)

print(f"RSquared:\t{r_squared:.3f}\nRMSE:\t\t{rmse:.3f}")
```

```
RSquared:       0.298
RMSE:           0.401
```

Lesson 09

In [24]:
```
# Lesson 09

# KNN Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.dat
a"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
X', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = read_csv(url, delim_whitespace=True, names=names)
array = dataframe.values
# split into features and target variables
X = array[:,0:13]
Y = array[:,13]
# shuffle=True was required to be added again
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = KNeighborsRegressor()
# note: i linked list of accepted scoring strings in Lesson 08 along with what
each does
# Mean squared error regression loss
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
-38.852320266666666
```

In [38]:
```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Spot check linear algorithms on a dataset (e.g. linear regression, logistic
regression and linear discriminate analysis).

checks = {
    LinearRegression(): 'neg_mean_squared_error',
    LogisticRegression(solver='liblinear'): None,
    LinearDiscriminantAnalysis(): 'accuracy'
}
# loop through above to apply the estimator and scoring to the X and Y created
earlier
for check, score in checks.items():
    results = cross_val_score(check, X, Y, cv=kfold, scoring=score)
    print(f"{str(check).split('(')[0]}: {results.mean()}")
```

```
LinearRegression: -0.1632073857590707
LogisticRegression: 0.7708646616541353
LinearDiscriminantAnalysis: 0.7669685577580315
```

In [26]:
```
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor

# Spot check some non-linear algorithms on a dataset (e.g. KNN, SVM and CART).

checks = {
    SVR(): 'neg_mean_squared_error',
    DecisionTreeRegressor(): 'neg_mean_squared_error',
    KNeighborsRegressor(): 'neg_mean_squared_error'
}
for check, score in checks.items():
    results = cross_val_score(check, X, Y, cv=kfold, scoring=score)
    print(f"{str(check).split('(')[0]}: {results.mean()}")
```

```
SVR: -67.64140705473743
DecisionTreeRegressor: -20.25612431372549
KNeighborsRegressor: -38.852320266666666
```

For spot checking algorithms, I don't know if using the same scoring is required throughout. Using certain combinations of algo and evaluation score would cause errors. Atleast with the method of application I am using, it is easy to update with the correct value.

Lesson 10

In [27]:
```python
# Lesson 10

# Compare Algorithms
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indian
s-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'clas
s']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# prepare models to be compared
linear_models = [('LR', LogisticRegression(solver='liblinear')),
                 ('LDA', LinearDiscriminantAnalysis())]
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in linear_models:
    kfold = KFold(n_splits=10, random_state=7, shuffle=True)
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    # i hate original way of printing vars compared to using f-strings
    msg = f"{name}: {cv_results.mean():.6f} ({cv_results.std():.6f})"
    print(msg)
```

```
LR: 0.770865 (0.050905)
LDA: 0.766969 (0.047966)
```

- Compare linear algorithms to each other on a dataset.
- Compare nonlinear algorithms to each other on a dataset.
- Compare different configurations of the same algorithm to each other.
- Create plots of the results comparing algorithms.

Linear

```
LinearRegression(): 'neg_mean_squared_error'
LogisticRegression(solver='liblinear'): None
LinearDiscriminantAnalysis(): 'accuracy'
```

Non Linear

```
SVR(): 'neg_mean_squared_error'
DecisionTreeRegressor(): 'neg_mean_squared_error'
KNeighborsRegressor(): 'neg_mean_squared_error'
```

Different configurations

```
LinearRegression(): 'neg_mean_squared_error'
LinearRegression(): 'accuracy'
```

In [28]:
```
nonlinear_models = [('SVR', SVR()),
                    ('DTR', DecisionTreeRegressor())]

results = []
names = []
scoring = 'neg_mean_squared_error'
for name, model in nonlinear_models:
    kfold = KFold(n_splits=10, random_state=7, shuffle=True)
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    # i hate original way of printing vars compared to using f-strings
    msg = f"{name}: {cv_results.mean():.6f} ({cv_results.std():.6f})"
    print(msg)
```

```
SVR: -0.174617 (0.020384)
DTR: -0.305878 (0.048066)
```

In [29]:
```
config_model = [('LR netwon-cg', LogisticRegression(solver='newton-cg')),
                ('LR liblinear', LogisticRegression(solver='liblinear'))]
results = []
names = []
scoring = 'neg_mean_squared_error'
# loop through each option, both using LogisticRegression but with different s
olvers
for name, model in config_model:
    kfold = KFold(n_splits=10, random_state=7, shuffle=True)
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    # i hate original way of printing vars compared to using f-strings
    msg = f"{name}: {cv_results.mean():.6f} ({cv_results.std():.6f})"
    print(msg)
```

```
LR netwon-cg: -0.227837 (0.049684)
LR liblinear: -0.229135 (0.050905)
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

Lesson 11

In [30]:
```
# Lesson 11

# Grid Search for Algorithm Tuning
from pandas import read_csv
import numpy
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indian
s-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'clas
s']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]

# define alpha values for the grid
alphas = numpy.array([1,0.1,0.01,0.001,0.0001,0])
param_grid = dict(alpha=alphas)

# grid search for ridge regression algo
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)
grid.fit(X, Y)
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

```
0.27961755931297233
1.0
```

- Tune the parameters of an algorithm using a grid search that you specify.
- Tune the parameters of an algorithm using a random search.

```
In [31]: from sklearn.model_selection import RandomizedSearchCV

         # define alpha values for the grid
         alphas = numpy.array([1,0.1,0.01,0.001,0.0001,0])
         param_random = dict(alpha=alphas)

         # grid search for ridge regression algo
         model = Ridge()

         # setup random search with 3-fold cross validation
         random_search = RandomizedSearchCV(estimator=model, param_distributions=param_
         random, n_iter=6)
         random_search.fit(X, Y)
         # not defining n_iter gives a warning because it defaults to 10 while there is
         6 alpha values

         print(random_search.best_score_)
         print(random_search.best_estimator_.alpha)
```

```
0.27610844129292433
1.0
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

Lesson 12

```
In [32]: # Lesson 12

         # Random Forest Classification
         from pandas import read_csv
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         from sklearn.ensemble import RandomForestClassifier
         url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indian
         s-diabetes.data.csv"
         names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'clas
         s']
         dataframe = read_csv(url, names=names)
         array = dataframe.values
         X = array[:,0:8]
         Y = array[:,8]
         num_trees = 100
         max_features = 3
         kfold = KFold(n_splits=10, random_state=7, shuffle=True)
         model = RandomForestClassifier(n_estimators=num_trees, max_features=max_featur
         es)
         results = cross_val_score(model, X, Y, cv=kfold)
         print(results.mean())
```

```
0.7643369788106631
```

- Practice bagging ensembles with the random forest and extra trees algorithms.
- Practice boosting ensembles with the gradient boosting machine and AdaBoost algorithms.
- Practice voting ensembles using by combining the predictions from multiple models together.

In [33]:
```
from sklearn.ensemble import ExtraTreesClassifier
model_extratrees = ExtraTreesClassifier(n_estimators=num_trees, max_features=m
ax_features)
results_extratrees = cross_val_score(model_extratrees, X, Y, cv=kfold)
print(f"Extra Trees: {results_extratrees.mean()}")
```

```
Extra Trees: 0.7682330827067669
```

In [34]:
```
from sklearn.ensemble import GradientBoostingClassifier
model_gradient = GradientBoostingClassifier(n_estimators=num_trees, max_featur
es=max_features)
results_gradient = cross_val_score(model_gradient, X, Y, cv=kfold)
print(f"Gradient Boosting: {results_gradient.mean()}")
```

```
Gradient Boosting: 0.764354066985646
```

In [35]:
```
from sklearn.ensemble import AdaBoostClassifier
model_ada = AdaBoostClassifier(n_estimators=num_trees)
results_ada = cross_val_score(model_ada, X, Y, cv=kfold)
print(f"AdaBoost: {results_ada.mean()}")
```

```
AdaBoost: 0.7578605604921395
```

Extra Trees: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html)

Gradient Boosting Machine: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html)

AdaBoost: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)

Lesson 13

In [36]:
```
# Lesson 13

# Save Model Using Pickle
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pickle
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indian
s-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'clas
s']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)
# Fit the model on 67%
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))

# some time later...

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, Y_test)
print(result)
```

```
0.7559055118110236
```

In [37]:
```
import pickle

# Save model
# with open('model.pkl', 'wb') as file:
#     pickle.dump(model, file)

# Reload model
# with open('model.pkl', 'rb') as file:
#     model_reloaded = pickle.load(file)
```

In [ ]: