CAP 5768 - Data Science - Dr. Marques - Fall 2020[¶](#gjdgxs)

Assignment 4: Machine Learning[¶](#30j0zll)

## STARTER CODE[¶](#1fob9te)

Goals[¶](#3znysh7)

* To learn how to implement a Data Science / Machine Learning workflow in Python (using Pandas, Scikit-learn, Matplotlib, and Numpy)
* To learn how to use perform linear regression by least squares using Python and scikit-learn.
* To appreciate that the same linear regression coefficients may be the best fit for dramatically different data distributions -- as illustrated by the Anscombe's quartet.
* To practice with different types of regularization (*lasso* and *ridge*) and understand when to use them.
* To learn how to implement several different machine learning classification models in Python
* To learn how to evaluate and fine-tune the performance of a model using cross-validation
* To learn how to test a model and produce a set of plots and performance measures
* To expand upon the prior experience of manipulating, summarizing, and visualizing representative datasets in data science and machine learning

Instructions[¶](#2et92p0)

* This assignment is structured in 3 parts, each using their own dataset(s).
* As usual, there will be some Python code to be written and questions to be answered.
* At the end, you should export your notebook to PDF format; it will "automagically" become your report.
* Submit the report (PDF), notebook (.ipynb file), and the link to the "live" version of your solution on Google Colaboratory via Canvas.
* **The number of points is indicated next to each part. They add up to 100.**
* **There are additional (10 points worth of) bonus items**, which are, of course optional.

Important[¶](#tyjcwt)

* For the sake of reproducibility, use random\_state=0 (or equivalent) in all functions that use random number generation.
* It is OK to attempt the bonus points, but please **do not overdo it!**

Imports + Google Drive[¶](#3dy6vkm)

In [ ]:

**!** pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

In [112]:

*# Imports*  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** pandas **as** pd  
**%matplotlib** inline  
**import** seaborn **as** sns; sns**.**set()  
**import** scipy.stats **as** ss  
**from** pandas\_profiling **import** ProfileReport  
  
**from** tensorflow **import** keras  
**from** tensorflow.keras **import** layers

In [113]:

*# Mount Google Drive*  
**from** google.colab **import** drive  
drive**.**mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

Part 1: Regression Analysis[¶](#1t3h5sf)

1a. Linear regression by least squares[¶](#4d34og8)

In this part, we will look at the correlation between female literacy and fertility (defined as the average number of children born per woman) throughout the world. For ease of analysis and interpretation, we will work with the *illiteracy* rate.

The Python code below plots the fertility versus illiteracy and computes the Pearson correlation coefficient. The Numpy array illiteracy has the illiteracy rate among females for most of the world's nations. The array fertility has the corresponding fertility data.

In [114]:

df **=** pd**.**read\_csv('/content/drive/My Drive/Colab Notebooks/data/female\_literacy\_fertility.csv')   
*# df = pd.read\_csv('data/female\_literacy\_fertility.csv')*  
illiteracy **=** 100 **-** df['female literacy']  
fertility **=** df['fertility']  
  
**def** pearson\_r(x, y):  
 """Compute Pearson correlation coefficient between two arrays."""  
 *# Compute correlation matrix: corr\_mat*  
 corr\_mat **=** np**.**corrcoef(x, y)  
  
 *# Return entry [0,1]*  
 **return** corr\_mat[0,1]  
  
*# Plot the illiteracy rate versus fertility*  
\_ **=** plt**.**plot(illiteracy, fertility, marker**=**'.', linestyle**=**'none')  
  
*# Set the margins and label axes*  
plt**.**margins(0.02)  
\_ **=** plt**.**xlabel('% illiterate')  
\_ **=** plt**.**ylabel('fertility')  
  
*# Show the plot*  
plt**.**show()  
  
*# Show the Pearson correlation coefficient*  
print('Pearson correlation coefficient between illiteracy and fertility: {:.5f}'**.**format(pearson\_r(illiteracy, fertility)))

Pearson correlation coefficient between illiteracy and fertility: 0.80413

In [115]:

df**.**head()

Out[115]:

|  | **Country** | **Continent** | **female literacy** | **fertility** | **population** |
| --- | --- | --- | --- | --- | --- |
| **0** | Chine | ASI | 90.5 | 1.769 | 1,324,655,000 |
| **1** | Inde | ASI | 50.8 | 2.682 | 1,139,964,932 |
| **2** | USA | NAM | 99.0 | 2.077 | 304,060,000 |
| **3** | Indonésie | ASI | 88.8 | 2.132 | 227,345,082 |
| **4** | Brésil | LAT | 90.2 | 1.827 | 191,971,506 |

1.1 Your turn! (5 points)[¶](#2s8eyo1)

We will assume that fertility is a linear function of the female illiteracy rate: f=ai+b, where a is the slope and b is the intercept.

We can think of the intercept as the minimal fertility rate, probably somewhere between one and two.

The slope tells us how the fertility rate varies with illiteracy. We can find the best fit line .

Write code to plot the data and the best fit line (using np.polyfit()) and print out the slope and intercept.

In [116]:

*# ENTER YOUR CODE HERE*  
*#p(x) = p[0] \* x\*\*deg + ... + p[deg]*  
*#y= m\*x^(1)+c*  
a **=** np**.**polyfit(illiteracy, fertility, 1)  
*#numpy.polyfit(x, y, deg)*  
slope **=** a[0]  
intercept **=** a[1]  
print("slope =",a[0] )  
print("intercept =",a[1] )

slope = 0.04979854809063423  
intercept = 1.888050610636557

1b. Anscombe's quartet[¶](#17dp8vu)

The Anscombe's quartet is a collection of four small data sets that have nearly identical simple descriptive statistics, yet have very different distributions. Each dataset consists of 11 (x,y) points. The quartet was created in 1973 by the statistician Francis Anscombe to demonstrate: the importance of visualization and exploratory data analysis (EDA), the effect of outliers and other influential observations on statistical properties, and the limitations of summary statistics (\*).

(\*) See <https://heap.io/blog/data-stories/anscombes-quartet-and-why-summary-statistics-dont-tell-the-whole-story> if you're interested.

The Python code below performs a linear regression on the data set from Anscombe's quartet that is most reasonably interpreted with linear regression.

In [117]:

x1 **=** [10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0]  
y1 **=** [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68]  
  
x2 **=** [10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0]  
y2 **=** [9.14, 8.14, 8.74, 8.77, 9.26, 8.10, 6.13, 3.10, 9.13, 7.26, 4.74]  
  
x3 **=** [10.0, 8.0, 13.0, 9.0, 11.0, 14.0, 6.0, 4.0, 12.0, 7.0, 5.0]  
y3 **=** [7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5.73]  
  
x4 **=** [8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 19.0, 8.0, 8.0, 8.0]  
y4 **=** [6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.50, 5.56, 7.91, 6.89]

In [118]:

*# Perform linear regression: a, b*  
a, b **=** np**.**polyfit(x1, y1, 1)  
  
*# Print the slope and intercept*  
print('slope =', a)  
print('intercept =', b)  
  
*# Generate theoretical x and y data: x\_theor, y\_theor*  
x\_theor **=** np**.**array([3, 15])  
y\_theor **=** a **\*** x\_theor **+** b  
  
*# Plot the Anscombe data and theoretical line*  
\_ **=** plt**.**plot(x1, y1, marker**=**'.', linestyle**=**'none')  
\_ **=** plt**.**plot(x\_theor, y\_theor)  
  
*# Label the axes*  
plt**.**xlabel('x')  
plt**.**ylabel('y')  
  
*# Show the plot*  
plt**.**show()

slope = 0.5000909090909095  
intercept = 3.000090909090909

1.2 Your turn! (5 points)[¶](#3rdcrjn)

### Linear regression on all Anscombe data[¶](#26in1rg)

Write code to verify that all four of the Anscombe data sets have the same slope and intercept from a linear regression, i.e. compute the slope and intercept for each set.

The data are stored in lists (anscombe\_x = [x1, x2, x3, x4] and anscombe\_y = [y1, y2, y3, y4]), corresponding to the $x$ and $y$ values for each Anscombe data set.

In [119]:

anscombe\_x **=** [x1, x2, x3, x4]  
anscombe\_y **=** [y1, y2, y3, y4]  
*# ENTER YOUR CODE HERE*  
  
  
  
ax\_1 **=** np**.**polyfit(anscombe\_x[0], anscombe\_y[0], 1)  
slope\_1 **=** ax\_1[0]  
intercept\_1 **=** ax\_1[1]  
print("slope1 =",ax\_1[0],"intercept1 =",ax\_1[1] )  
ax\_2 **=** np**.**polyfit(anscombe\_x[1], anscombe\_y[1], 1)  
slope\_2 **=** ax\_2[0]  
intercept\_2 **=** ax\_2[1]  
print("slope2 =",ax\_2[0],"intercept2 =",ax\_2[1] )  
ax\_3 **=** np**.**polyfit(anscombe\_x[2], anscombe\_y[2], 1)  
slope\_3 **=** ax\_3[0]  
intercept\_3 **=** ax\_3[1]  
print("slope3 =",ax\_3[0],"intercept3 =",ax\_3[1] )  
ax\_4 **=** np**.**polyfit(anscombe\_x[3], anscombe\_y[3], 1)  
slope\_4 **=** ax\_4[0]  
intercept\_4 **=** ax\_4[1]  
print("slope4 =",ax\_4[0],"intercept4 =",ax\_4[1] )

slope1 = 0.5000909090909095 intercept1 = 3.000090909090909  
slope2 = 0.5000000000000004 intercept2 = 3.0009090909090896  
slope3 = 0.4997272727272731 intercept3 = 3.0024545454545453  
slope4 = 0.4999090909090908 intercept4 = 3.0017272727272735

* slope1 = 0.5000909090909095 intercept1 = 3.000090909090909
* slope2 = 0.5000000000000004 intercept2 = 3.0009090909090896
* slope3 = 0.4997272727272731 intercept3 = 3.0024545454545453
* slope4 = 0.4999090909090908 intercept4 = 3.0017272727272735
* """Anscombe data has almost same slope and intercept """

1c. Regression using scikit-learn[¶](#lnxbz9)

Now that we know the basics of linear regression, we will switch to scikit-learn, a powerful, workflow-oriented library for data science and machine learning.

The Python code below shows a simple linear regression example using scikit-learn. Note the use of the fit() and predict() methods.

In [120]:

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
*# Generate random data around the y = ax+b line where a=3 and b=-2*  
rng **=** np**.**random**.**RandomState(42)  
x **=** 10 **\*** rng**.**rand(50)  
y **=** 3 **\*** x **-** 2 **+** rng**.**randn(50)  
  
**from** sklearn.linear\_model **import** LinearRegression  
  
*# Note: If you get a "ModuleNotFoundError: No module named 'sklearn'" error message, don't panic.*  
*# It probably means you'll have to install the module by hand if you're using pip.*   
*# If you're using conda, you should not see any error message.*  
  
model **=** LinearRegression(fit\_intercept**=True**)  
  
X **=** x[:, np**.**newaxis]  
X**.**shape  
  
model**.**fit(X, y)  
print(model**.**coef\_)  
print(model**.**intercept\_)  
  
xfit **=** np**.**linspace(**-**1, 11)  
Xfit **=** xfit[:, np**.**newaxis]  
yfit **=** model**.**predict(Xfit)  
  
plt**.**scatter(x, y)  
plt**.**plot(xfit, yfit);

[2.9776566]  
-1.903310725531119

1d. Polynomial regression[¶](#35nkun2)

One way to adapt linear regression to nonlinear relationships between variables is to transform the data according to *basis functions*.

The idea is to take the multidimensional linear model: $$ y = a\_0 + a\_1 x\_1 + a\_2 x\_2 + a\_3 x\_3 + \cdots $$ and build the $x\_1, x\_2, x\_3,$ and so on, from our single-dimensional input $x$. That is, we let $x\_n = f\_n(x)$, where $f\_n()$ is some function that transforms our data.

For example, if $f\_n(x) = x^n$, our model becomes a polynomial regression: $$ y = a\_0 + a\_1 x + a\_2 x^2 + a\_3 x^3 + \cdots $$ Notice that this is *still a linear model*—the linearity refers to the fact that the coefficients $a\_n$ never multiply or divide each other. What we have effectively done is taken our one-dimensional $x$ values and projected them into a higher dimension, so that a linear fit can fit more complicated relationships between $x$ and $y$.

The code below shows a simple example of polynomial regression using the PolynomialFeatures transformer in scikit-learn. Concretely, it shows how we can use polynomial features with a polynomial of degree seven, i.e. $$y = a\_0 + a\_1 x + a\_2 x^2 + a\_3 x^3 + \cdots + a\_7 x^7$$

It also introduces the notion of a *pipeline* in scikit-learn. "The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters." (<https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>)

In [121]:

**from** sklearn.preprocessing **import** PolynomialFeatures  
**from** sklearn.pipeline **import** make\_pipeline  
poly\_model **=** make\_pipeline(PolynomialFeatures(7),  
 LinearRegression())  
  
rng **=** np**.**random**.**RandomState(1)  
x **=** 10 **\*** rng**.**rand(100)  
y **=** np**.**sin(x) **+** 0.1 **\*** rng**.**randn(100)  
  
poly\_model**.**fit(x[:, np**.**newaxis], y)  
yfit **=** poly\_model**.**predict(xfit[:, np**.**newaxis])  
  
plt**.**scatter(x, y)  
plt**.**plot(xfit, yfit);  
  
print('The R^2 score for the fit is: ', poly\_model**.**score(x[:, np**.**newaxis], y))

The R^2 score for the fit is: 0.9806993128749489

Our linear model, through the use of 7th-order polynomial basis functions, can provide an excellent fit to this non-linear data!

1.3 Your turn! (10 points)[¶](#1ksv4uv)

Write code to find the best degree/order for the polynomial basis functions (between 1 and 15) by computing the quality of the fit using a suitable metric, in this case the $R^2$ coefficient (which can be computer using the score() function).

Remember that **the best possible score is 1.0**. The score can be negative (because the model can be arbitrarily worse). A score of 0 suggests a constant model that always predicts the expected value of y, disregarding the input features.

Hint: If you plot the score against the degree/order of the polynomial, you should see something like this:

In [122]:

rng **=** np**.**random**.**RandomState(1)  
x **=** 10 **\*** rng**.**rand(100)  
y **=** np**.**sin(x) **+** 0.1 **\*** rng**.**randn(100)  
  
R\_score**=**[]  
**for** i **in** range(0, 16):  
 poly\_model **=** make\_pipeline(PolynomialFeatures(i), LinearRegression())  
 poly\_model**.**fit(x[:, np**.**newaxis], y)  
 yfit **=** poly\_model**.**predict(xfit[:, np**.**newaxis])  
 R\_score**.**append(poly\_model**.**score(x[:, np**.**newaxis], y))  
  
R\_score  
plt**.**plot(R\_score)  
plt**.**ylabel("R^2 score")  
plt**.**xlabel("Polynomial degree")  
plt**.**show()

1e. Regularization[¶](#44sinio)

The use of polynomial regression with high-order polynomials can very quickly lead to over-fitting. In this part, we will look into the use of regularization to address potential overfitting.

The code below shows an attempt to fit a 15th degree polynomial to a sinusoidal shaped data. The fit is excellent ($R^2$ > 0.98), but might raise suspicions that it will lead to overfitting.

In [123]:

model **=** make\_pipeline(PolynomialFeatures(15), LinearRegression())  
  
model**.**fit(x[:, np**.**newaxis], y)  
  
plt**.**scatter(x, y)  
plt**.**plot(xfit, model**.**predict(xfit[:, np**.**newaxis]))  
  
plt**.**xlim(0, 10)  
plt**.**ylim(**-**1.5, 1.5);  
  
score **=** poly\_model**.**score(x[:, np**.**newaxis], y)  
print(score)

0.9810106927307878

In [123]:

1.4 Your turn! (5 points)[¶](#2jxsxqh)

Write Python code to perform Ridge regression ($L\_2$ Regularization), plot the resulting fit, and compute the $R^2$ score.

Hints:

1. This type of penalized model is built into Scikit-Learn with the Ridge estimator.
2. In the beginning, use all default values for its parameters.
3. After you get your code to work, spend some time trying to fine-tune the model, i.e., experimenting with the regularization parameters.

In [124]:

*# ENTER YOUR CODE HERE*  
  
rng **=** np**.**random**.**RandomState(1)  
x **=** 10 **\*** rng**.**rand(100)  
y **=** np**.**sin(x) **+** 0.05 **\*** rng**.**randn(100)  
  
**from** sklearn.linear\_model **import** Ridge  
  
Ridge\_model **=** make\_pipeline(PolynomialFeatures(15), Ridge())  
Ridge\_model**.**fit(x[:, np**.**newaxis], y)  
  
plt**.**scatter(x, y)  
plt**.**plot(xfit, Ridge\_model**.**predict(xfit[:, np**.**newaxis]))  
  
plt**.**xlim(0, 10)  
plt**.**ylim(**-**1.5, 1.5);  
  
score **=** Ridge\_model**.**score(x[:, np**.**newaxis], y)  
print(score)

0.9922588607527568

1.5 Your turn! (5 points)[¶](#z337ya)

Write Python code to perform Lasso regression ($L\_1$ Regularization), plot the resulting fit, and compute the $R^2$ score.

Hints:

1. This type of penalized model is built into Scikit-Learn with the Lasso estimator.
2. In the beginning, use Lasso(alpha=0.1, tol=0.2)
3. After you get your code to work, spend some time trying to fine-tune the model, i.e., experimenting with the regularization parameters.

In [125]:

*# ENTER YOUR CODE HERE*  
  
rng **=** np**.**random**.**RandomState(1)  
x **=** 10 **\*** rng**.**rand(100)  
y **=** np**.**sin(x) **+** 0.1 **\*** rng**.**randn(100)  
  
**from** sklearn.linear\_model **import** Lasso  
  
lasso\_model **=** make\_pipeline(PolynomialFeatures(15), Lasso(alpha**=**0.1, tol**=.**1))  
lasso\_model**.**fit(x[:, np**.**newaxis], y)  
  
plt**.**scatter(x, y)  
plt**.**plot(xfit, lasso\_model**.**predict(xfit[:, np**.**newaxis]))  
  
plt**.**xlim(0, 10)  
plt**.**ylim(**-**1.5, 1.5)  
  
score **=** lasso\_model**.**score(x[:, np**.**newaxis], y)  
print(score)

/usr/local/lib/python3.6/dist-packages/sklearn/linear\_model/\_coordinate\_descent.py:476: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 7.427004190467991, tolerance: 4.2480145591765615  
 positive)

0.7547973847410004

1f. The housing problem[¶](#3j2qqm3)

The Boston housing dataset is a classic dataset used in linear regression examples. (See <https://scikit-learn.org/stable/datasets/index.html#boston-dataset> for more)

The Python code below:

* Loads the Boston dataset (using scikit-learn's load\_boston()) and converts it into a Pandas dataframe
* Selects two features to be used for fitting a model that will then be used to make predictions: LSTAT (% lower status of the population) and RM (average number of rooms per dwelling) (\*)
* Splits the data into train and test sets

(\*) See <https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155> for details.

In [126]:

**from** sklearn.datasets **import** load\_boston  
boston\_dataset **=** load\_boston()  
boston **=** pd**.**DataFrame(boston\_dataset**.**data, columns**=**boston\_dataset**.**feature\_names)  
boston**.**head()

Out[126]:

|  | **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** | **LSTAT** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 |
| **1** | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 |
| **2** | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 |
| **3** | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2.94 |
| **4** | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5.33 |

In [127]:

boston['MEDV'] **=** boston\_dataset**.**target  
X **=** pd**.**DataFrame(np**.**c\_[boston['LSTAT'], boston['RM']], columns **=** ['LSTAT','RM'])  
y **=** boston['MEDV']  
  
**from** sklearn.model\_selection **import** train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size **=** 0.2, random\_state**=**5)  
print(X\_train**.**shape)  
print(X\_test**.**shape)  
print(y\_train**.**shape)  
print(y\_test**.**shape)

(404, 2)  
(102, 2)  
(404,)  
(102,)

(OPTIONAL EDA)[¶](#1y810tw)

The (innocent-looking) lines of code below use Pandas Profiling to produce rich reports, plots and insights on the dataset.

Read more about it:

* <https://pypi.org/project/pandas-profiling/>
* <https://www.datacourses.com/pandas-1150/>
* <https://pandas-profiling.github.io/pandas-profiling/docs/master/index.html>
* <https://medium.com/analytics-vidhya/pandas-profiling-5ecd0b977ecd>

In [128]:

*# fun with pandas\_profiling*  
profile **=** ProfileReport(boston, title**=**'Pandas Profiling Report for Boston Housing Dataset', explorative**=True**)

In [129]:

profile**.**to\_notebook\_iframe()

Output hidden; open in https://colab.research.google.com to view.

1.6 Bonus! (10 points)[¶](#4i7ojhp)

Write Python code to:

1. Fit a linear model to the data.
2. Compute and print the RMSE and $R^2$ score for both train and test datasets.
3. Fit a polynomial model (of degree 4) to the data.
4. Compute and print the RMSE and $R^2$ score for both train and test datasets.
5. Apply Ridge regression to the polynomial model.
6. Compute and print the RMSE and $R^2$ score for both train and test datasets.

In [130]:

*# ENTER YOUR CODE HERE*  
*# ...*  
*# ...*  
*# ...*

Part 2: Classification[¶](#2xcytpi)

2a. The Iris dataset[¶](#1ci93xb)

The Python code below will load a dataset containing information about three types of Iris flowers that had the size of its petals and sepals carefully measured.

The Fisher’s Iris dataset contains 150 observations with 4 features each:

* sepal length in cm;
* sepal width in cm;
* petal length in cm; and
* petal width in cm.

The class for each instance is stored in a separate column called “species”. In this case, the first 50 instances belong to class Setosa, the following 50 belong to class Versicolor and the last 50 belong to class Virginica.

See: <https://archive.ics.uci.edu/ml/datasets/Iris> for additional information.

In [131]:

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns  
iris **=** sns**.**load\_dataset("iris")  
iris**.**head()

Out[131]:

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

Histograms, pair plots and summary statistics[¶](#3whwml4)

The code below:

1. Computes and displays relevant summary statistics for the whole dataset.
2. Displays the pair plots for all (4) attributes for all (3) categories / species / classes in the Iris dataset.

In [132]:

*# Display pair plot*  
sns**.**pairplot(iris, hue**=**'species', height**=**2.5);  
  
*# Display summary statistics for the whole dataset*  
iris**.**describe()

Out[132]:

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** |
| --- | --- | --- | --- | --- |
| **count** | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| **mean** | 5.843333 | 3.057333 | 3.758000 | 1.199333 |
| **std** | 0.828066 | 0.435866 | 1.765298 | 0.762238 |
| **min** | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| **25%** | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| **50%** | 5.800000 | 3.000000 | 4.350000 | 1.300000 |
| **75%** | 6.400000 | 3.300000 | 5.100000 | 1.800000 |
| **max** | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

2.1 Your turn! (15 points)[¶](#2bn6wsx)

Write code to:

1. Build a decision tree classifier using scikit-learn's DecisionTreeClassifier (using the default options). Check documentation at <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
2. Plot the resulting decision tree. (Note: if graphviz gives you headaches, a text-based 'plot'-- using export\_text -- should be OK.)
3. Perform k-fold cross-validation using k=3 and display the results.

In [184]:

*# ENTER YOUR CODE HERE*  
  
**from** sklearn.datasets **import** load\_iris  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn **import** tree  
  
*# class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max\_depth=None, min\_samples\_split=2,*   
*# min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0,*   
*# max\_features=None, random\_state=None, max\_leaf\_nodes=None, class\_weight=None, presort=False)*  
  
  
iris **=** sns**.**load\_dataset("iris")  
sns**.**reset\_orig() *# reset to orginal data set*  
  
*# sns data into np*  
iris **=** load\_iris()  
df **=** pd**.**DataFrame(iris['data'], columns**=**iris['feature\_names'])   
df['target'] **=** iris['target'] *#Original Pandas df (features + target)*  
  
*#75% of the data into a training set and 25% of the data into a test set.*  
X\_train, X\_test, Y\_train, Y\_test **=** train\_test\_split(df[iris**.**feature\_names], df['target'], random\_state**=**0)   
  
*# Make an instance of the Model*  
classified\_tree **=** DecisionTreeClassifier()  
  
*#Predict labels of unseen (test) data*  
classified\_tree**.**fit(X\_train, Y\_train)   
  
*#labelling*  
fn**=**['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']  
cn**=**['setosa', 'versicolor', 'virginica']  
  
*#add graphics to tree image*  
fig, axes **=** plt**.**subplots(nrows **=** 1,ncols **=** 1,figsize **=** (4,4), dpi**=**200)  
  
*#plot tree*  
tree**.**plot\_tree(classified\_tree,feature\_names **=** fn, class\_names**=**cn, filled **=** **True**); *# ; need otherwise so many texts*  
  
*#tree.plot\_tree(classified\_tree); # plot withoiut labeling*

2b. Digit classification[¶](#qsh70q)

The MNIST handwritten digit dataset consists of a training set of 60,000 examples, and a test set of 10,000 examples. Each image in the dataset has 28$\times$28 pixels.

The Python code below loads the images from the MNIST dataset, flattens them, normalizes them (i.e., maps the intensity values from [0..255] to [0..1]), and displays a few images from the training set.

In [133]:

**from** keras.datasets **import** mnist *# load data set*  
  
*# Model / data parameters*  
num\_classes **=** 10 *# 0-10 number as classes*  
input\_shape **=** (28, 28, 1) *# pixes 28\*28 1 = gray cycle (blck and white)*   
  
*# the data, split between train and validation sets*  
(X\_train, y\_train), (X\_valid, y\_valid) **=** mnist**.**load\_data() *# y\_train = output , xtrain images , x valid - to check ,*

In [134]:

X\_train**.**shape

Out[134]:

(60000, 28, 28)

In [135]:

y\_train**.**shape *# labels 60000*

Out[135]:

(60000,)

In [136]:

*#tranign set interm of numbers*  
y\_train[0:12] *#just for view*

Out[136]:

array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5], dtype=uint8)

In [137]:

*#traning set interm of picutres / numbers # just for view*  
plt**.**figure(figsize**=**(5,5))  
**for** k **in** range(12):  
 plt**.**subplot(3, 4, k**+**1)  
 plt**.**imshow(X\_train[k], cmap**=**'Greys')  
 plt**.**axis('off')  
plt**.**tight\_layout()  
plt**.**show()

In [138]:

X\_valid**.**shape

Out[138]:

(10000, 28, 28)

In [139]:

y\_valid**.**shape

Out[139]:

(10000,)

In [140]:

y\_valid[0]

Out[140]:

7

In [141]:

*#validation test*  
plt**.**imshow(X\_valid[0], cmap**=**'Greys')  
plt**.**axis('off')  
plt**.**show()

In [142]:

X\_valid[0]**.**max() *# 8 bit images 0-225*

Out[142]:

255

In [142]:

In [142]:

In [143]:

*# Reshape (flatten) images*   
*#28\*28 = 784*  
*#convert images to float32*  
X\_train\_reshaped **=** X\_train**.**reshape(60000, 784)**.**astype('float32') *# data feeding method,*   
X\_valid\_reshaped **=** X\_valid**.**reshape(10000, 784)**.**astype('float32')  
  
*# Scale images to the [0, 1] range*  
''' 8 bit images normalization X\_train\_scaled\_reshaped value is between 0-1 (because diveided by 255)'''  
X\_train\_scaled\_reshaped **=** X\_train\_reshaped **/** 255   
X\_valid\_scaled\_reshaped **=** X\_valid\_reshaped **/** 255   
  
*# Renaming for conciseness*  
X\_training **=** X\_train\_scaled\_reshaped   
X\_validation **=** X\_valid\_scaled\_reshaped  
  
print("X\_training shape (after reshaping + scaling):", X\_training**.**shape)  
print(X\_training**.**shape[0], "train samples")  
print("X\_validation shape (after reshaping + scaling):", X\_validation**.**shape)  
print(X\_validation**.**shape[0], "validation samples")

X\_training shape (after reshaping + scaling): (60000, 784)  
60000 train samples  
X\_validation shape (after reshaping + scaling): (10000, 784)  
10000 validation samples

In [143]:

In [144]:

*# convert class vectors to binary class matrices*  
''' singale valvue "8" making a catogorical vector [one hot conversion] see below '''  
y\_training **=** keras**.**utils**.**to\_categorical(y\_train, num\_classes)  
y\_validation **=** keras**.**utils**.**to\_categorical(y\_valid, num\_classes)

In [145]:

print(y\_valid[0])  
print(y\_validation[0])  
''' [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.] = 8th number so "8" '''

7  
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]

Out[145]:

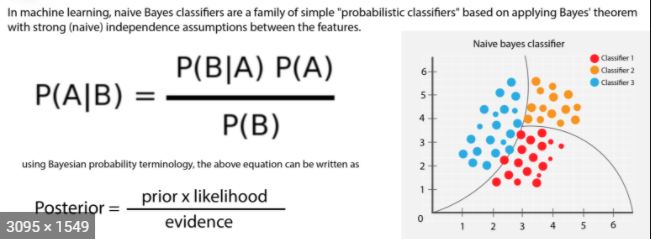
' [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.] = 8th number so "8" '

2.2 Your turn! (10 points)[¶](#3as4poj)

Write code to:

1. Build and fit a 10-class Naive Bayes classifier using scikit-learn's MultinomialNB() with default options and using the raw pixel values as features.
2. Make predictions on the test data, compute the overall accuracy and plot the resulting confusing matrix.

Hint: your accuracy will be around 83.5%



In [146]:

*# ENTER YOUR CODE HERE*  
*#Naive Bayes calssificatioin*   
**from** sklearn.naive\_bayes **import** MultinomialNB   
  
NBclassifier **=** MultinomialNB()   
NBclassifier**.**fit(X\_training , y\_train) *# categorize*   
  
predicted\_labels **=** NBclassifier**.**predict(X\_validation) *# predict*

In [147]:

predicted\_labels

Out[147]:

array([7, 2, 1, ..., 9, 8, 6], dtype=uint8)

In [148]:

predicted\_labels**.**shape

Out[148]:

(10000,)

In [170]:

*# ENTER YOUR CODE HERE*  
  
**from** sklearn.metrics **import** confusion\_matrix  
**import** seaborn **as** sns  
  
matrix **=** confusion\_matrix(y\_valid , predicted\_labels)  
  
*#sns.heatmap(matrix.T, fmt='d', cmap="YlGnBu", cbar=False, square=True, annot=True)*  
sns**.**heatmap(matrix**.**T, square**=True**, annot**=True**, fmt**=**'d',cbar**=False**, cmap**=**"YlGnBu")  
  
plt**.**xlabel('true label')  
plt**.**ylabel('predicted label')  
  
  
print(matrix)

[[ 973 0 1 0 0 2 1 1 2 0]  
 [ 0 1127 3 1 0 1 0 1 2 0]  
 [ 4 0 1015 0 1 0 0 6 6 0]  
 [ 0 0 2 995 0 3 0 6 4 0]  
 [ 0 0 3 0 966 0 4 0 2 7]  
 [ 2 0 0 5 1 878 2 1 2 1]  
 [ 4 2 0 0 2 3 946 0 1 0]  
 [ 0 3 10 1 1 0 0 1004 2 7]  
 [ 1 0 1 4 1 2 0 2 960 3]  
 [ 3 3 2 6 9 2 0 5 6 973]]

In [172]:

**from** sklearn.metrics **import** accuracy\_score  
accuracy\_score(y\_valid, predicted\_labels)

Out[172]:

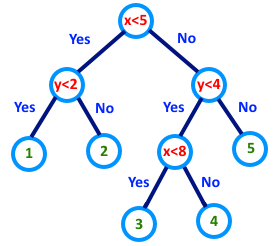
0.9837

2.3 Your turn! (10 points)[¶](#1pxezwc)

Write code to:

1. Build and fit a 10-class Random Forests classifier using scikit-learn's RandomForestClassifier() with default options (don't forget random\_state=0) and using the raw pixel values as features.
2. Make predictions on the test data, compute the overall accuracy and plot the resulting confusing matrix.

Hint: your accuracy should be > 90%



In [151]:

**from** sklearn.ensemble **import** RandomForestClassifier  
RFClassified\_model **=** RandomForestClassifier(random\_state**=**0)  
  
RFClassified\_model**.**fit(X\_training , y\_train) *#catagorizing*  
  
predicted\_labels2 **=** RFClassified\_model**.**predict(X\_validation) *# predict*

In [169]:

*# ENTER YOUR CODE HERE*  
  
**from** sklearn.metrics **import** confusion\_matrix  
**import** seaborn **as** sns  
  
matrix2 **=** confusion\_matrix(y\_valid , predicted\_labels2)  
  
*#sns.heatmap(matrix.T, fmt='d', cmap="YlGnBu", cbar=False, square=True, annot=True)*  
sns**.**heatmap(matrix2**.**T, square**=True**, annot**=True**, fmt**=**'d',cbar**=False**, cmap**=**"YlGnBu")  
  
plt**.**xlabel('true label')  
plt**.**ylabel('predicted label')  
  
  
print(matrix2)

[[ 969 0 0 0 0 2 5 1 3 0]  
 [ 0 1124 2 3 0 2 2 1 1 0]  
 [ 5 0 999 7 2 0 3 9 7 0]  
 [ 1 0 10 971 0 10 0 8 8 2]  
 [ 1 0 1 0 958 0 6 0 2 14]  
 [ 3 0 0 11 0 862 8 2 4 2]  
 [ 6 3 0 0 3 4 939 0 3 0]  
 [ 1 3 20 1 0 0 0 989 3 11]  
 [ 5 0 5 6 4 9 4 4 928 9]  
 [ 8 5 1 9 8 3 1 4 4 966]]

In [153]:

**from** sklearn.metrics **import** accuracy\_score  
accuracy\_score(y\_valid, predicted\_labels2)

Out[153]:

0.9705

2.4 Your turn! (10 points)[¶](#49x2ik5)

Write code to:

1. Build and fit a 10-class classifier of your choice, with sensible initialization options, and using the raw pixel values as features.
2. Make predictions on the test data, compute the overall accuracy and plot the resulting confusing matrix.

Hint: A variation of the Random Forests classifier from 2.2 above is acceptable. In that case, document your selection of (hyper)parameters and your rationale for choosing them.

In [154]:

print(type(y\_train))

<class 'numpy.ndarray'>

In [155]:

**from** sklearn **import** datasets, svm, metrics  
  
param\_C **=** 5  
param\_gamma **=** 0.05  
classifier **=** svm**.**SVC(C**=**param\_C,gamma**=**param\_gamma)  
  
  
*########################################################*  
*# Now predict the value of the test*  
expected **=** y\_valid  
  
classifier**.**fit(X\_training , y\_train)   
  
predicted\_labels **=** classifier**.**predict(X\_validation)

In [168]:

**from** sklearn.metrics **import** confusion\_matrix  
**import** seaborn **as** sns  
  
matrix2 **=** confusion\_matrix(y\_valid , predicted\_labels)  
  
*#sns.heatmap(matrix.T, fmt='d', cmap="YlGnBu", cbar=False, square=True, annot=True)*  
sns**.**heatmap(matrix2**.**T, square**=True**, annot**=True**, fmt**=**'d',cbar**=False**, cmap**=**"YlGnBu")  
  
plt**.**xlabel('true label')  
plt**.**ylabel('predicted label')  
print(matrix2)

[[ 973 0 1 0 0 2 1 1 2 0]  
 [ 0 1127 3 1 0 1 0 1 2 0]  
 [ 4 0 1015 0 1 0 0 6 6 0]  
 [ 0 0 2 995 0 3 0 6 4 0]  
 [ 0 0 3 0 966 0 4 0 2 7]  
 [ 2 0 0 5 1 878 2 1 2 1]  
 [ 4 2 0 0 2 3 946 0 1 0]  
 [ 0 3 10 1 1 0 0 1004 2 7]  
 [ 1 0 1 4 1 2 0 2 960 3]  
 [ 3 3 2 6 9 2 0 5 6 973]]

In [173]:

**from** sklearn.metrics **import** accuracy\_score  
accuracy\_score(y\_valid, predicted\_labels)

Out[173]:

0.9837

more info: <https://github.com/ksopyla/svm_mnist_digit_classification/blob/master/svm_mnist_classification.py>

Part 3: Face Recognition using PCA (eigenfaces)[¶](#2p2csry)

In this part you will build a face recognition solution.

We will use a subset of the Labeled Faces in the Wild (LFW) people dataset: <https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_lfw_people.html>

The Python code below loads a dataset of 1867 images (resized to 62 $\times$ 47 pixels) from the dataset and displays some of them.

In [158]:

**from** sklearn.datasets **import** fetch\_lfw\_people  
faces **=** fetch\_lfw\_people(min\_faces\_per\_person**=**40)  
print(faces**.**target\_names)  
print(faces**.**images**.**shape)

['Ariel Sharon' 'Arnold Schwarzenegger' 'Colin Powell' 'Donald Rumsfeld'  
 'George W Bush' 'Gerhard Schroeder' 'Gloria Macapagal Arroyo'  
 'Hugo Chavez' 'Jacques Chirac' 'Jean Chretien' 'Jennifer Capriati'  
 'John Ashcroft' 'Junichiro Koizumi' 'Laura Bush' 'Lleyton Hewitt'  
 'Luiz Inacio Lula da Silva' 'Serena Williams' 'Tony Blair'  
 'Vladimir Putin']  
(1867, 62, 47)

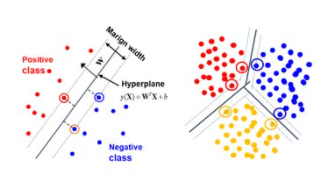
In [159]:

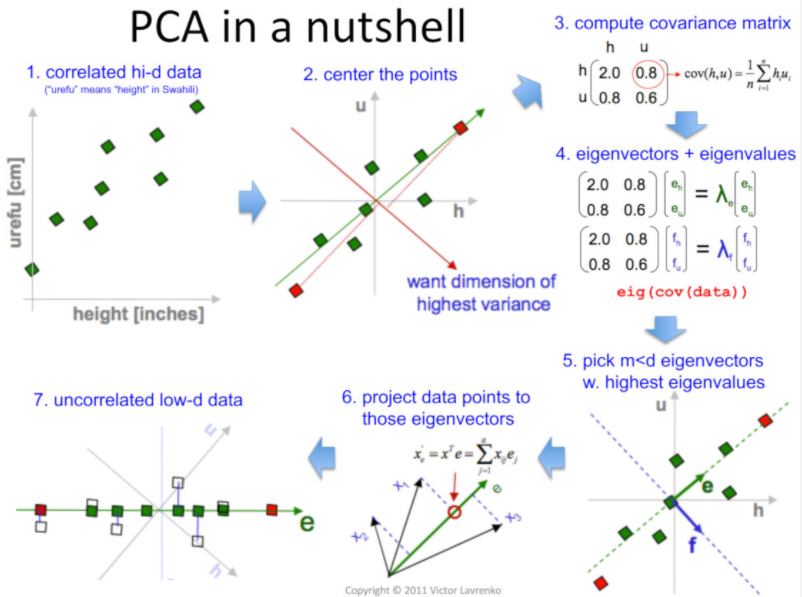
plt**.**rcParams["figure.figsize"]**=**15,15  
fig, ax **=** plt**.**subplots(3, 5)  
**for** i, axi **in** enumerate(ax**.**flat):  
 axi**.**imshow(faces**.**images[i], cmap**=**'bone')  
 axi**.**set(xticks**=**[], yticks**=**[],  
 xlabel**=**faces**.**target\_names[faces**.**target[i]])

3.1 Your turn! (15 points)[¶](#147n2zr)

Write code to:

1. Use Principal Component Analysis (PCA) to reduce the dimensionality of each face to the first 120 components.
2. Build and fit a multi-class SVM classifier, with sensible initialization options, and using the PCA-reduced features.
3. Make predictions on the test data, compute the precision, recall and f1 score for each category, compute the overall accuracy, and plot the resulting confusing matrix.
4. Display examples of correct and incorrect predictions (at least 5 of each).

* svm classifier 



In [160]:

*# ENTER YOUR CODE HERE*  
  
**from** sklearn.svm **import** SVC  
**from** sklearn.decomposition **import** PCA  
**from** sklearn.pipeline **import** make\_pipeline  
  
pca **=** PCA(n\_components**=**120, random\_state**=**0, whiten**=True**)  
svc **=** SVC(kernel**=**'rbf', C**=**1.0, class\_weight**=**'balanced')  
svc\_model **=** make\_pipeline(pca, svc)  
  
pca**.**fit(faces**.**data)

Out[160]:

PCA(copy=True, iterated\_power='auto', n\_components=120, random\_state=0,  
 svd\_solver='auto', tol=0.0, whiten=True)

In [161]:

**from** sklearn.model\_selection **import** train\_test\_split  
X\_train2, X\_valid2, y\_train2, y\_valid2 **=** train\_test\_split(faces**.**data, faces**.**target,   
 random\_state**=**0)

In [162]:

svc\_model**.**fit(X\_train2, y\_train2)

Out[162]:

Pipeline(memory=None,  
 steps=[('pca',  
 PCA(copy=True, iterated\_power='auto', n\_components=120,  
 random\_state=0, svd\_solver='auto', tol=0.0, whiten=True)),  
 ('svc',  
 SVC(C=1.0, break\_ties=False, cache\_size=200,  
 class\_weight='balanced', coef0=0.0,  
 decision\_function\_shape='ovr', degree=3, gamma='scale',  
 kernel='rbf', max\_iter=-1, probability=False,  
 random\_state=None, shrinking=True, tol=0.001,  
 verbose=False))],  
 verbose=False)

In [163]:

y\_fit **=**svc\_model**.**predict(X\_valid2) *#fitting*

In [164]:

*#fitting and dispay*  
plt**.**rcParams["figure.figsize"]**=**10,10  
fig, ax **=** plt**.**subplots(6, 6)  
**for** i, axi **in** enumerate(ax**.**flat):  
 axi**.**imshow(X\_valid2[i]**.**reshape(62,47), cmap**=**'bone')  
 axi**.**set(xticks**=**[], yticks**=**[])  
 axi**.**set\_ylabel(faces**.**target\_names[y\_fit[i]]**.**split()[**-**1],   
 color**=**'black' **if** y\_fit[i]**==**y\_valid2[i] **else** 'red')  
  
''' wrong predictions are in red '''

Out[164]:

' wrong predictions are in red '

In [165]:

**from** sklearn.metrics **import** classification\_report  
print(classification\_report(y\_valid2, y\_fit, target\_names**=**faces**.**target\_names))

precision recall f1-score support  
  
 Ariel Sharon 0.67 0.67 0.67 18  
 Arnold Schwarzenegger 0.25 0.08 0.12 13  
 Colin Powell 0.57 0.83 0.68 54  
 Donald Rumsfeld 0.77 0.85 0.81 27  
 George W Bush 0.78 0.89 0.83 131  
 Gerhard Schroeder 0.77 0.80 0.79 30  
 Gloria Macapagal Arroyo 1.00 0.73 0.84 11  
 Hugo Chavez 0.94 0.68 0.79 25  
 Jacques Chirac 0.71 0.45 0.56 11  
 Jean Chretien 1.00 0.55 0.71 11  
 Jennifer Capriati 1.00 0.50 0.67 12  
 John Ashcroft 0.86 0.38 0.52 16  
 Junichiro Koizumi 1.00 0.86 0.93 22  
 Laura Bush 1.00 0.64 0.78 14  
 Lleyton Hewitt 1.00 0.55 0.71 11  
Luiz Inacio Lula da Silva 1.00 0.80 0.89 10  
 Serena Williams 0.42 0.73 0.53 11  
 Tony Blair 0.55 0.65 0.60 26  
 Vladimir Putin 0.46 0.43 0.44 14  
  
 accuracy 0.73 467  
 macro avg 0.78 0.63 0.68 467  
 weighted avg 0.76 0.73 0.73 467

In [166]:

**from** sklearn.metrics **import** confusion\_matrix  
**import** seaborn **as** sns  
  
matrix3 **=** confusion\_matrix(y\_valid2 , y\_fit)  
  
*#sns.heatmap(matrix.T, fmt='d', cmap="YlGnBu", cbar=False, square=True, annot=True)*  
sns**.**heatmap(matrix3**.**T, square**=True**, annot**=True**, fmt**=**'d',cbar**=False**, cmap**=**"YlGnBu")  
  
plt**.**xlabel('true label')  
plt**.**ylabel('predicted label')  
  
  
print(matrix3)

[[ 12 0 3 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0  
 1]  
 [ 0 1 1 0 5 2 0 0 0 0 0 0 0 0 0 0 1 2  
 1]  
 [ 2 0 45 0 4 0 0 0 1 0 0 1 0 0 0 0 0 1  
 0]  
 [ 1 0 1 23 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 2]  
 [ 1 0 10 1 116 0 0 0 0 0 0 0 0 0 0 0 1 1  
 1]  
 [ 0 1 1 0 2 24 0 0 0 0 0 0 0 0 0 0 0 2  
 0]  
 [ 0 0 1 0 2 0 8 0 0 0 0 0 0 0 0 0 0 0  
 0]  
 [ 0 0 3 1 2 1 0 17 0 0 0 0 0 0 0 0 0 1  
 0]  
 [ 0 0 0 3 1 1 0 0 5 0 0 0 0 0 0 0 1 0  
 0]  
 [ 1 0 2 0 0 0 0 0 0 6 0 0 0 0 0 0 1 0  
 1]  
 [ 0 1 2 0 2 0 0 0 0 0 6 0 0 0 0 0 1 0  
 0]  
 [ 1 0 4 0 1 2 0 0 0 0 0 6 0 0 0 0 0 1  
 1]  
 [ 0 0 0 0 1 1 0 0 0 0 0 0 19 0 0 0 0 1  
 0]  
 [ 0 0 1 0 0 0 0 0 0 0 0 0 0 9 0 0 3 1  
 0]  
 [ 0 0 0 1 2 0 0 0 0 0 0 0 0 0 6 0 2 0  
 0]  
 [ 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 8 0 1  
 0]  
 [ 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 8 0  
 0]  
 [ 0 1 3 0 5 0 0 0 0 0 0 0 0 0 0 0 0 17  
 0]  
 [ 0 0 0 1 3 0 0 0 1 0 0 0 0 0 0 0 0 3  
 6]]

Conclusions (10 points)[¶](#3o7alnk)

Write your conclusions and make sure to address the issues below:

* What have you learned from this assignment?
* Which parts were the most fun, time-consuming, enlightening, tedious?
* What would you do if you had an additional week to work on this?

Things I learned

* To perform linear regression and non linear regression using numpy
* To perform linear regression and non linear regression using sklearn
* sklearn Lasso regression parameters
* panda profiling
* classification methods (DecisionTreeClassifier / MultinomialNB /RandomForestClassifier )
* validation sets and trainging set
* How to train data over differnt methods
* How to validate data
* diffrence between DecisionTreeClassifier / MultinomialNB /RandomForestClassifier )
* Component Analysis (PCA)
* multi-class SVM classifier
* Display correct and incorrect predictions for SVM classifier

The most interesting part was to divide the data into a training set and a validation set. The time-consuming part was to understand the code and how the classification methods work. The enlightening part was to understan difference between each classification methods, and the tedious part is to manipulate codes. I would pay more attention to the mathematics side of each classification method.

In [ ]:

**%cd** /content/drive/My\ Drive/Colab\ Notebooks/  
  
**!**sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-generic-recommended  
  
**!**jupyter nbconvert --to html nkumarachchi2019\_CAP5768\_Fall2020\_Assignment4\_STARTER.ipynb

/content/drive/My Drive/Colab Notebooks  
Reading package lists... Done  
Building dependency tree   
Reading state information... Done  
texlive-fonts-recommended is already the newest version (2017.20180305-1).  
texlive-generic-recommended is already the newest version (2017.20180305-1).  
texlive-xetex is already the newest version (2017.20180305-1).  
The following packages were automatically installed and are no longer required:  
 fig2dev libdbus-glib-1-2 libgtkmm-2.4-1v5 libgtkspell0 python-bs4  
 python-chardet python-html5lib python-lxml python-pkg-resources python-scour  
 python-six python-webencodings scour transfig  
Use 'sudo apt autoremove' to remove them.  
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.

In [ ]: