Qbus1040 Python notes

Foundations of Business Analytics (University of Sydney)

Python Basic

Print:
Print('qbus1040')

List:
numbers = [4, 7, 2, 0]
to_buy = ['carrot', 'fruit']

Float → decimal number
String formation:
print('Pi to 2 decimal places: {:.2f}'.format(pi))

<u>Calculator/Math library:</u>
+ addition, - subtraction, * multiplication, / division, % modulus, ** power
Import math
Print(math.factorial(4)) → 24

Time library:
import time
start = time.time()
 # Your program
end = time.time()
elapsed_time = end – start

Numpy library:

Input: collect information from user
variable = **input('**message_displayed_to_user '**)**

- Don't forget that input values are always strings!
- You might need to convert to *int()* , *float()* , *np.fromstring(variable, dtype=int, sep=' ')* when handling numeric values.

  variable = input(*message_displayed_to_user*)
  e.g. x = int(input('Enter your number: '))
- np.fromstring → convert input to vector

np.random.seed(0) → every time generate same set of random #
np.random.rand(int/float/nth)(low (inclusive), high (exclusive), size)
np.inner → inner product between 2 vectors (a^Tb) → scalar
np.array → store vectors and matrices
np.shape/ a.shape() → show dimensions
np.concatenate((a,b, axis=0 or 1))

np.array([2, 1 ,3])
*array*__[__*index*__]__
*array*__[__*start*:*end*__]__

- start index inclusive, end exlcusive
- A[2:4, :3] → starting from 2 to 3, starting from 0 up to 2

range(*end***) – end is exclusive**
range(*start***,** *end***)**
range(*start***,** *end***,** *step_size***)**

np.zeros(*dimension***)**

- Np.zeros((n,1)) → print zero matrix
- Np.zeros(n) → print n-vector

np.ones(*dimension***)**

.min(), .max(), .mean(), .sum()
.argmin(), .argmax() → show the index of the min/max values

np.isclose() → check whether two values are similar

```
if condition:
    # code you execute if condition is true
else:
    # code you execute if condition is false
```

Elif statement

```
if condition_1:
    # code to execute if condition 1 is true
elif condition_2:
    # code to execute if condition 2 is true
elif condition_3:
    # code to execute if condition 3 is true
...
else:
    # code to execute if none of the conditions are true
```

```
np.eye(n)
np.tril (matrix)
np.triu (matrix)
np.diag(np.array([3.7, 2.5, -1.2, 4.5]))
```

Brackets:

- () Used for functions such as print()
- [] Used for lists.
- {} Used for string formatting.

Root mean Square(RMS)

```
def rms(x):
    norm = np.sqrt(np.inner(x,x))
    sqrt_n = np.sqrt(x.shape[0])
    rms = norm/sqrt_n

    return rms
```

Tut 7 Gram-schmidt A → Q
Tut 8 QR factorisation A → Q, R
Tut 9 Back Sub Rx = b → get x
Tut 10 Solve via Back Sub A, b → get x

Back substitution: Rx = b

- Get x from R, b

```
def back_sub(R, b):
   n, n = R.shape
   x = np.zeros(n)
   for i in range(n):
      j = n - 1 - i
      x[j] = (b[j] - np.inner(R[j], x)) / R[j, j]
   return x
```

Solve via Back Substitution:

- Get $\hat{x}$ from R, $Q^T b$
- A → Q, R
- B → Q^Tb
- Perform back sub (R, Q^Tb)

$$\hat{x} = A^{\dagger} b = R^{-1} Q^T b$$
$$R \hat{x} = R R^{-1} Q^T b$$
$$R \hat{x} = Q^T b$$

```
def solve_via_back_sub(A, b):
   linearly_independent, Q_transpose = gram_schmidt(A.T) → why on rows?
   R = Q_transpose @ A
   Q = Q_transpose.T
   x = back_sub(R, Q_transpose @ b)
   return x


def polyfit(x, y, degree):
   A = vandermonde(x, degree+1)
   theta_hat = solve_via_back_sub(A, y)
   return theta_hat

def polyeval(x, theta_hat, degree):
   A = vandermonde(x, degree+1)
   f_hat_x = A @ theta_hat
   return f_hat_x
```

Least Squares Data Fitting

- Constant model:
A = np.ones((y_d.shape[0],1))

- Straight-line fit:
A = np.concatenate((ones, x_d), axis=1)

- De-trended time series:

```
year = [1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015]
location = [0, 5, 10, 15, 20, 25, 30, 35]
plt.xticks(location, year)
```

- Polynomial fit:

```
A = funcs.vandermonde(x, degree+1)
theta_hat = funcs.solve_via_back_sub(A, y_d)
```

  - Sometimes there are errors with python, whens some value in A are too large, the p_inv of A will be too small, python can't handle so we use solve via back sub, instead of np.linalg.pinv(A) @ y_d

- Auto Regressive model:

```
A = np.zeros((T-M, M)) → no. of predictions corresponds to # of rows in zeros

for i in range(M):
    A[:, i] = z[M-1-i:T-1-i]
          # A[:, 0] = z[M-1-0:T-1-0]
          # A[:, 1] = z[M-1-1:T-1-1]
          # A[:, 2] = z[M-1-2:T-1-2]
y_d = z[M:].reshape(-1, 1) → true data (after memory)
```

----

np.linspace(start, stop, number) → generate a vector of equally spaced x values for testing

- The easiest way to visualise your model is to "plot points".
- This involves constructing a new matrix *A* for lots of different *x* values, and then obtaining the corresponding *y*^ and then joining the dots!

```
theta_hat = np.linalg.pinv(A) @ y_d → find model parameter
f_hat_x = A @ theta_hat → estimate/predicted y_d
```

Loading Data

```
Fold1 = np.load('fold1.npy')
red_df = pd.read_csv('red-wine.csv').to_numpy() → convert to np array
```

Extracting Data

1. x.reshape(-1, 1)

- This function reshape a 1D array into 2D array with one column, and as many rows as needed to contain all elements
- Convert a row vector (array) to column vector

2.
```
x_data = data[:, 0] #extracting all rows but only 1st col
y_data = data[:, 1] #extracting all rows but only 2nd col
```

3.
```
allfolds = [0, 1, 2, 3, 4]
selected = allfolds[1:] #starting from 1 (inclusive till the end)
[1, 2, 3, 4]
```

4.
```
f = ['fold0', 'fold1', 'fold2', 'fold3', 'fold4']
i = 2
print(f[:i] + f[i+1:])
#selecting all folds, except fold i
['fold0', 'fold1', 'fold3', 'fold4']
```

Cross Validation

- Split data into training and test data
- Obtain theta hat from train data (x,y)
- Use theta hat obtained from train data, generate f_hat_x for test data (A_test @ theta_hat)
- Compare predicted f_hat_y to true test_y → generate residuals, rmse

```
#calculate the rmse of the test/train data of that split
def rmse_split(all_folds, split, degree):

    train = np.concatenate(all_folds[:split] + all_folds[split+1:], axis=0)
    test = all_folds[split]
    #pull out all training except for fold split, and make split the test d

    train_x = train[:, 0]; train_y = train[:, 1]
    test_x = test[:, 0]; test_y = test[:, 1]
```

```
    theta_hat = funcs.polyfit(train_x, train_y, degree)

    test_y_hat = funcs.polyeval(test_x, theta_hat, degree)
    test_residuals = test_y - test_y_hat
    test_rmse = funcs.rms(test_residuals)

    train_y_hat = funcs.polyeval(train_x, theta_hat, degree)
    train_residuals = train_y - train_y_hat
    train_rmse = funcs.rms (train_residuals)

    return test_rmse, train_rmse

#Test function
all_folds = [fold0, fold1, fold2, fold3, fold4]
split = 0
degree = 1
test = rmse_split(all_folds, split, degree)

#creating for-loop calculate all rmses of all splits
rmses = np.zeros(5)
for i in range(5):
    split = i
    test = rmse_split(all_folds, split ,degree)
    rmses[i] = test

rms_cv = funcs.rms(rmses)
```

Visualizing Data **(Matplotlib)**

Import matplotlib.pyplot as plt

plt.figure(figsize=(9, 4)) → create new figure with width9, height4
plt.subplot(1, 2, 1)
plt.subplot(1, 2, 2)

plt.scatter(x values, y values, s=10) → scatter plots
plt.plot(x values, y values, 'o-') → line plots

- label = 'y^{d_1}' → y^{d_1}

plt.axhline(1, color='red') → adding straight line
plt.quiver(0, 0, 3, 4, angles='xy', scale_units='xy', scale=1) → showing vector displacement

plt.grid(); plt.axis('square') → help visualisation
Plt.xlim([x_min, x_max]), plt.ylim([y_min, y_max])
plt.xlabel('x1'); plt.ylabel('x2')
plt.title('Example scatter graph')
plt.legend()
plt.savefig('filename')