Getting started with Python for Data Science

Meenal Jhajharia

Meenal Jhajharia. she/her. 19.

CS and Math undergrad student

University of Delhi, India

PyMC Core Contributor | GSoC student - PyMC (under NUMFOCUS)

Web - [mjhajharia.com (https://mjhajharia.com)](https://mjhajharia.com)

You will find this useful if

- You already know an object oriented programming language
- You want to learn Python
- You're interested in Scientific Computing

Quick facts about Python

- Useful for quick prototyping
- Dynamically Typed
- Interpreted Language
- High level data types
- There's a large number of scientific open source software in Python

Best Place to learn more : [Official Python documentation (https://docs.python.org/3/tutorial/introduction.html)](https://docs.python.org/3/tutorial/introduction.html)

Sneak peek into

- Basic Python Syntax
- Numpy and Pandas
- PyMC

Takeaways

- A taste of reading Python code in large libraries
- This will help you contribute to Open Source + write better code

When browsing code from libraries you aren't familiar with, we'll be focusing on "how" the code works, and be less concerned with the actual use or thought behind it, think of it as, we pick up a random novel to learn how to read or speak, we don't care about the story being told (**only** right now)

If you're on a UNIX system, great, if you're on windows ensure you have a terminal, if not use [this (https://www.microsoft.com/en-in/p/windows-terminal/9n0dx20hk701?rtc=1&activetab=pivot:overviewtab)](https://www.microsoft.com/en-in/p/windows-terminal/9n0dx20hk701?rtc=1&activetab=pivot:overviewtab)

```
$ conda install notebook
$ jupyter-notebook
```

Let's get started right away! so for the first half of this session I'll quickly go over basic data cleaning/manipulation in pandas/numpy. Data is one of the most crucial things in any form of scientific analysis, naturally, you cannot make reasonable predictions without meaningful data.

Pandas and Numpy are the most fundamental libraries that you'll be using in Python

To make the process less mundane, I'm going to over the data I had to clean for [a model I made in PyMC for Fantasy Premier League predictions, I gave a talk on this at PyData about two weeks ago! (https://github.com/mjhajharia/pydata21/blob/main/fpl-analysis-model.ipynb)](https://github.com/mjhajharia/pydata21/blob/main/fpl-analysis-model.ipynb)

In [1]:
```
import pandas as pd
import numpy as np

!curl -o 2020.csv https://raw.githubusercontent.com/vaastav/Fantasy-Premier-Leagu
e/master/data/2020-21/gws/merged_gw.csv
data20 = pd.read_csv("2020.csv")

!curl -o 2021.csv https://raw.githubusercontent.com/vaastav/Fantasy-Premier-Leagu
e/master/data/2020-21/gws/merged_gw.csv
data = pd.read_csv("2021.csv")

data20 = pd.read_csv("2020.csv")
data = pd.read_csv("2021.csv")
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 3494k  100 3494k    0     0  2385k      0  0:00:01  0:00:01 --:--:-- 2383k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100 3494k  100 3494k    0     0  3643k      0 --:--:-- --:--:-- --:--:-- 3640k
```

So here's the data elements that we do have, now we don't need all of them

In [2]:
```
data.columns
```

Out[2]:
```
Index(['name', 'position', 'team', 'xP', 'assists', 'bonus', 'bps',
       'clean_sheets', 'creativity', 'element', 'fixture', 'goals_conceded',
       'goals_scored', 'ict_index', 'influence', 'kickoff_time', 'minutes',
       'opponent_team', 'own_goals', 'penalties_missed', 'penalties_saved',
       'red_cards', 'round', 'saves', 'selected', 'team_a_score',
       'team_h_score', 'threat', 'total_points', 'transfers_balance',
       'transfers_in', 'transfers_out', 'value', 'was_home', 'yellow_cards',
       'GW'],
      dtype='object')
```

We want to use two years' data so we pick the players that play in both years

In [3]:

```
players21 = set(data.name)
players20 = set(data20.name)
players20 = players20.intersection(players21)
data20 = data20[data20['name'].isin(players20)]
data = data[data['name'].isin(players20)]
```

now we want to append the columns `total_points` and `GW` of last year to this year's data. We could join the dataframes directly, but we're not sure if the indexes will match so we do it manually, and fix an index using `data.set_index(cols)`

In [4]:

```
data20['total_points_20'] = data20['total_points']
data20['GW_20'] = data20['GW']
data20['name_20'] = data20['name']
data20 = data20[['name_20','GW_20','total_points_20']]
cols = ['name','GW']
cols20 = ['name_20','GW_20']
data20.set_index(cols20)
data.set_index(cols)
data = data.join(data20)
data = data[(data['name']==data['name_20']) & (data['GW']==data['GW_20'])]
```

Map `opponent_team` id's to team_names, we need to do this to inculcate some external data defined in terms of team_names

In [5]:
```
team_names = sorted(list(set(list((data.team)))))
team_ids = np.arange(1,21)
data['opponent_team'] = data['opponent_team'].map(dict(zip(team_ids, team_names)))
```

External data of last year

In [6]:

```
#in previous year
goals_conceded_by_team = {'Arsenal': 39,'Aston Villa': 46,'Brighton': 46,'Burnley'
: 55,
                          'Chelsea': 36,'Crystal Palace': 66,'Everton': 48,'Fulha
m': 55,
                          'Leeds': 54,'Leicester': 50,'Liverpool': 42,'Man City':
32,
                          'Man Utd': 44,'Newcastle': 62,'Sheffield Utd': 63,'South
ampton': 68,
                          'Spurs': 45,'West Brom': 76,'West Ham': 47,'Wolves': 52}

goals_scored= {'Man City':83,'Man Utd':73,'Leicester':68,
               'Liverpool':68,'Spurs':68,'Leeds':62,'West Ham':62,
               'Chelsea':58,'Arsenal':55,'Aston Villa':55,'Southampton':48,
               'Everton':47,'Newcastle':46,'Brighton':41,'Crystal Palace':41,
               'Wolves':36,'West Brom':35,'Burnley':33,'Fulham':27,'Sheffield Utd'
:20}

team_wins = {'Man City': 27,'Man Utd': 21, 'Leicester': 20,'Liverpool': 20,
             'Chelsea': 19,'West Ham': 19,'Arsenal': 18,'Leeds': 18,'Spurs': 18,
             'Everton': 17,'Aston Villa': 16,'Crystal Palace': 12,'Newcastle': 12,
             'Southampton': 12,'Wolves': 12,'Burnley': 10,'Brighton': 9,
             'Sheffield Utd': 7,'Fulham': 5,'West Brom': 5}
```

Create columns for this new information by mapping with `team_name` , additionally rank these points as `1` , `2` , `3` or `4` this is better than absolute ranks because it clusters teams together and reduces computational load on our final model

In [7]:
```
#we measure goals_conceded as an inverse indicator of defense strength
data['opp_defense_rank'] = data['opponent_team'].map(goals_conceded_by_team)
data.sort_values(by =['opp_defense_rank'], inplace = True)
data['opp_defense_rank']= pd.qcut(data['opp_defense_rank'],q = 4, labels = False)

#Goals scored are an attribute of team strength
data['opp_attack_rank'] = data['opponent_team'].map(goals_scored)
data.sort_values(by =['opp_attack_rank'], inplace = True)
data['opp_attack_rank']= pd.qcut(data['opp_attack_rank'],q = 4, labels = False)

#Final rankings naturally give an idea of the overall team strength/quality

data['team_cluster_rank'] = data['opponent_team'].map(goals_scored)
data.sort_values(by =['team_cluster_rank'], inplace = True)
data['team_cluster_rank']= pd.qcut(data['team_cluster_rank'],q = 4, labels = False
)

data['opp_cluster_rank'] = data['opponent_team'].map(goals_scored)
data.sort_values(by =['opp_cluster_rank'], inplace = True)
data['opp_cluster_rank']= pd.qcut(data['opp_cluster_rank'],q = 4, labels = False)
```

We map the names back to integers

In [8]:
```
data['opponent_team'] = data['opponent_team'].map(dict(zip(team_names, team_ids)))
data['team'] = data['team'].map(dict(zip(team_names, team_ids)))
```

We store the initial valuation of each player as a metric for prediction

In [9]:
```
initval = data[data['GW']==1][["name","value"]]
data['initval'] = data['name'].map(dict(zip(initval.name, initval.value)))
```

One hot encode the positions

In [10]:

```
data['pos_id'] = data['position']
data = pd.get_dummies(data,columns=['position'])
pos_ids = np.array([k for k in data['pos_id'].unique()])
data['pos_id']=data['pos_id'].apply(lambda x : np.where(x == pos_ids)[0][0])
```

Average (expected) scores of players' last year data

In [11]:
```
data['game_avg_7']= data.groupby(['name']
                                 )['total_points_20'].rolling(7).mean().reset_index
(0,drop=True)

data.sort_values(by =['game_avg_7'], inplace = True)
data['rank']= pd.qcut(data['game_avg_7'],q = 4, labels = False)
data['diff_from_avg'] = data['total_points'] - data['game_avg_7']
```

Eliminate very low scoring players to avoid skewing data

In [12]:
```
pts=data.groupby(['name']).sum()
pts = pts[pts['total_points']>50]
chosen_players = pts.index
data = data[data['name'].isin(chosen_players)]
```

One hot encode `was_home` and retain the original column as well

In [13]:
```
data['was_homee']=data['was_home']
data = pd.get_dummies(data,columns=['was_home'])
data['was_home']=data['was_homee']
data = data.dropna()
```

Select useful columns and store final data

In [14]:
```
data = data[['name', 'total_points','position_DEF','GW','team',
             'opponent_team','position_FWD', 'position_GK',
             'position_MID', 'game_avg_7' ,'pos_id',
             'initval','rank','diff_from_avg',
            'opp_defense_rank','opp_attack_rank','team_cluster_rank',
            'opp_cluster_rank','was_home','total_points_20']]
data.sort_values(by=['GW'],inplace=True)
data.to_csv('data.csv',index=False)
```

```
import pymc as pm
import pandas as pd
import numpy as np
import arviz as az
from sklearn.metrics import mean_absolute_error
```

```
You are running the v4 development version of Py
MC which currently still lacks key features. You
probably want to use the stable v3 instead which
you can either install via conda or find on the
v3 GitHub branch: https://github.com/pymc-devs/p
ymc/tree/v3
```

```
with pm.Model() as model:
    nu = pm.Exponential('nu minus one', 1/29,shape=2) + 1
    err = pm.Uniform('std dev based on rank', 0, 100, shape=ranks)
    err_b = pm.Uniform('std dev based on rank b', 0, 100, shape=ranks)
```

```
data = pd.read_csv('data.csv')
train = data[:2985]
test = data[2986:]

ranks=4
team_number = 20
player_names = set(train.name)
opp_defense_rank_no = train.opp_defense_rank.max()
opp_attack_rank_no = train.opp_attack_rank.max()
team_cluster_rank_no = train.team_cluster_rank.max()
opp_cluster_rank_no = train.opp_cluster_rank.max()
num_positions = 4
N = len(train)
```

```
with model:
    team_cluster_rank = pm.Data('team_cluster_rank',np.asarray((train['team_cluster_rank']).values, dtype = int))
    opp_cluster_rank = pm.Data('opp_cluster_rank',np.asarray((train['opp_cluster_rank']).values, dtype = int))
    opp_defense_rank = pm.Data('opp_defense_rank',np.asarray((train['opp_defense_rank']).values, dtype = int))
    opp_attack_rank = pm.Data('opp_attack_rank',np.asarray((train['opp_attack_rank']).values, dtype = int))
    initval = pm.Data('initval',np.asarray((train['initval']).values, dtype = int))
    player_home = pm.Data('player_home',np.asarray(train['was_home'].values, dtype = int))
    player_avg = pm.Data('player_avg',np.asarray((train['game_avg_7']).values, dtype = float))
    player_opp = pm.Data('player_opp',np.asarray((train['opponent_team']).values, dtype = int))
    player_team = pm.Data('player_team',np.asarray((train['team']).values, dtype = int))
    player_rank = pm.Data('player_rank',np.asarray((train['rank']-1).values, dtype = int))
    position_FWD = pm.Data('position_FWD',np.asarray((train['position_FWD']).values.astype(int),
                                        dtype = int))
    position_MID = pm.Data('position_MID',np.asarray((train['position_MID']).values.astype(int),
                                        dtype = int))
    position_GK = pm.Data('position_GK',np.asarray((train['position_GK']).values.astype(int),
                                       dtype = int))
    position_DEF = pm.Data('position_DEF',np.asarray((train['position_DEF']).values.astype(int),
                                        dtype = int))
    pos_id = pm.Data('pos_id',np.asarray((train['pos_id']).values, dtype = int))
```

```
with model:

    mu = (position_GK*pos_home_gk[player_rank]*(player_home) +
          position_MID*pos_home_mid[player_rank]*(player_home) +
          position_FWD*pos_home_fwd[player_rank]*(player_home) +
          position_DEF*pos_home_defe[player_rank]*(player_home) +
          position_GK*pos_away_gk[player_rank]*(1-player_home) +
          position_MID*pos_away_mid[player_rank]*(1-player_home) +
          position_FWD*pos_away_fwd[player_rank]*(1-player_home) +
          position_DEF*pos_away_defe[player_rank]*(1-player_home) +
          team_strength_effects + opp_strength_team_effects +
          opp_strength_attack_team_effects +
          opp_strength_defense_team_effects +
          initval*initval_coeff+player_avg + def_effect)

    like2 = pm.StudentT('Score', mu=mu, sd=err[player_rank],
                        nu=nu[0], observed=train['total_points'])
```

```
with model:
#     trace = pm.sample(10000, pm.sample())
    trace=az.from_netcdf('data')
    assert all(az.rhat(trace) < 1.03)
```

```
with model:
    pm.set_data({'team_cluster_rank': np.asarray((train['team_cluster_rank']).values, dtype = int)})
    pm.set_data({'opp_cluster_rank': np.asarray((test['opp_cluster_rank']).values, dtype = int)})
    pm.set_data({'opp_defense_rank': np.asarray((test['opp_defense_rank']).values, dtype = int)})
    pm.set_data({'opp_attack_rank': np.asarray((test['opp_attack_rank']).values, dtype = int)})
    pm.set_data({'initval': np.asarray((test['initval']).values, dtype = int)})
    pm.set_data({'player_home': np.asarray(test['was_home'].values, dtype = int)})
    pm.set_data({'player_avg': np.asarray((test['game_avg_7']).values, dtype = float)})
    pm.set_data({'player_opp': np.asarray((test['opponent_team']).values, dtype = int)})
    pm.set_data({'player_team': np.asarray((test['team']).values, dtype = int)})
    pm.set_data({'player_rank': np.asarray((test['rank']-1).values, dtype = int)})
    pm.set_data({'position_FWD': np.asarray((test['position_FWD']).values.astype(int),dtype = int)})
    pm.set_data({'position_MID': np.asarray((test['position_MID']).values.astype(int), dtype = int)})
    pm.set_data({'position_GK': np.asarray((test['position_GK']).values.astype(int),dtype = int)})
    pm.set_data({'position_DEF': np.asarray((test['position_DEF']).values.astype(int), dtype = int)})
    pm.set_data({'pos_id': np.asarray((test['pos_id']).values, dtype = int)})

    ppc=pm.sample_posterior_predictive(trace, samples=44000)
```

100.00% [44000/44000 01:36<00:00]

```
test['pred_points'] = ppc['Score'][0].tolist()
pts=test.groupby(['name']).sum()
pts.sort_values(by =['pred_points'], inplace = True,ascending=False)
pred = set(pts[:15].index)
pts=test.groupby(['name']).sum()
pts.sort_values(by =['total_points'], inplace = True,ascending=False)
truth = set(pts[:15].index)
len(pred.intersection(truth))
```

7

```
mean_absolute_error(test.loc[:,'total_points'].values, ppc['Score'].mean(axis=0))
```

```
2.5331934828587186
```

```
pred
```

```
{'Callum Wilson',
 'Christian Benteke',
 'Dominic Calvert-Lewin',
 'Emiliano Martínez',
 'Harry Kane',
 'Heung-Min Son',
 'Hugo Lloris',
 'Jordan Pickford',
 'Matheus Pereira',
 'Mohamed Salah',
 'Patrick Bamford',
 'Pierre-Emerick Aubameyang',
 'Roberto Firmino',
 'Rodrigo Moreno',
 'Stuart Dallas'}
```

truth

```
{'Emiliano Martínez',
 'Harry Kane',
 'Illan Meslier',
 'Jordan Pickford',
 'Kelechi Iheanacho',
 'Leandro Trossard',
 'Lewis Dunk',
 'Lucas Digne',
 'Matheus Pereira',
 'Mohamed Salah',
 'Nicolas Pépé',
 'Patrick Bamford',
 'Sam Johnstone',
 'Stuart Dallas',
 'Trent Alexander-Arnold'}
```

References

1. https://srome.github.io/Bayesian-Hierarchical-Modeling-Applied-to-Fantasy-Football-Projections-for-Increased-Insight-and-Confidence/
2. https://www.degruyter.com/document/doi/10.1515/jqas-2017-0066/html
3. https://pymc-examples.readthedocs.io/en/latest/case_studies/multilevel_modeling.html

Now we're going to go into the second half of this tutorial, where we try to understand how python open source packages work. Let's try to break-down a very simple distribution in PyMC. - Yes it's Normal

These are the relevant files - distribution.py (https://github.com/pymc-devs/pymc/blob/main/pymc/distributions/distribution.py) and continuous.py (https://github.com/pymc-devs/pymc/blob/31b4a37076668a9a12ef856852e563ae8faf02f2/pymc/distributions/conti

---

```python
class Normal(Continuous):
    r"""
    Univariate normal log-likelihood.

    The pdf of this distribution is

    .. math::

       f(x \mid \mu, \tau) =
           \sqrt{\frac{\tau}{2\pi}}
           \exp\left\{ -\frac{\tau}{2} (x-\mu)^2 \right\}

    Normal distribution can be parameterized either in terms of precision
    or standard deviation. The link between the two parametrizations is
    given by

    .. math::

       \tau = \dfrac{1}{\sigma^2}
```

```
.. plot::

    import matplotlib.pyplot as plt
    import numpy as np
    import scipy.stats as st
    import arviz as az
    plt.style.use('arviz-darkgrid')
    x = np.linspace(-5, 5, 1000)
    mus = [0., 0., 0., -2.]
    sigmas = [0.4, 1., 2., 0.4]
    for mu, sigma in zip(mus, sigmas):
        pdf = st.norm.pdf(x, mu, sigma)
        plt.plot(x, pdf, label=r'$\mu$ = {}, $\sigma$ = {}'.format(mu, sigma))
    plt.xlabel('x', fontsize=12)
    plt.ylabel('f(x)', fontsize=12)
    plt.legend(loc=1)
    plt.show()
```

```
========  ==========================================
Support   :math:`x \in \mathbb{R}`
Mean      :math:`\mu`
Variance  :math:`\dfrac{1}{\tau}` or :math:`\sigma^2`
========  ==========================================

Parameters
----------
mu: float
    Mean.
sigma: float
    Standard deviation (sigma > 0) (only required if tau is not specified).
tau: float
    Precision (tau > 0) (only required if sigma is not specified).
```

```
Examples
--------
.. code-block:: python

    with pm.Model():
        x = pm.Normal('x', mu=0, sigma=10)

    with pm.Model():
        x = pm.Normal('x', mu=0, tau=1/23)
"""
```

```python
rv_op = normal

@classmethod
def dist(cls, mu=0, sigma=None, tau=None, sd=None, no_assert=False, **kwargs):
    if sd is not None:
        sigma = sd
    tau, sigma = get_tau_sigma(tau=tau, sigma=sigma)
    sigma = at.as_tensor_variable(sigma)
```

```python
        if not no_assert:
            assert_negative_support(sigma, "sigma", "Normal")

        return super().dist([mu, sigma], **kwargs)

    def get_moment(rv, size, mu, sigma):
        mu, _ = at.broadcast_arrays(mu, sigma)
        if not rv_size_is_none(size):
            mu = at.full(size, mu)
        return mu
```

```python
def logcdf(value, mu, sigma):
    """
    Compute the log of the cumulative distribution function for Normal distribution
    at the specified value.

    Parameters
    ----------
    value: numeric or np.ndarray or `TensorVariable`
        Value(s) for which log CDF is calculated. If the log CDF for multiple
        values are desired the values must be provided in a numpy array or `TensorVariable`.

    Returns
    -------
    TensorVariable
    """
    return bound(
        normal_lcdf(mu, sigma, value),
        0 < sigma,
    )
```