Project DotA Documentation

Akshat Gupta

https://github.com/neonash/ProjectDotA

Feb 2020

Index

1. Introduction

The documentation includes the Software Engineering aspect of the project and highlights the key parameters designed to create it.

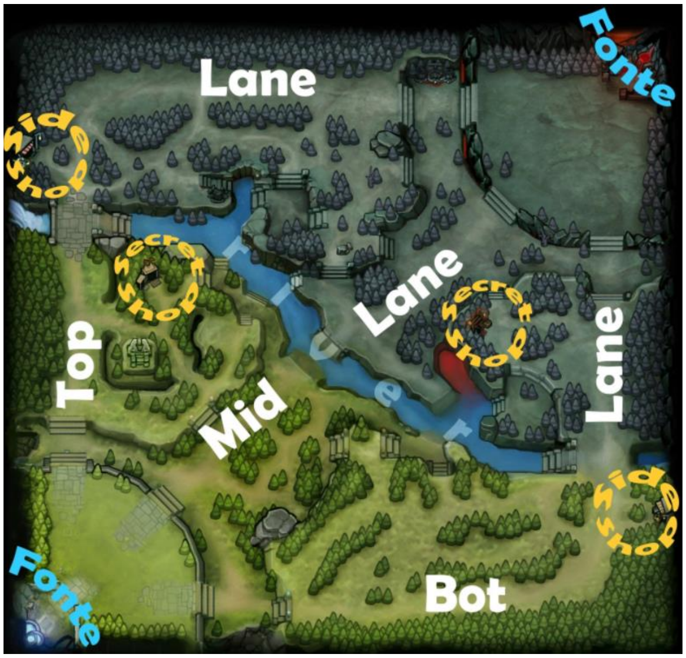
This is a Django based Web App – Project DotA which is designed for DotA Players and Analysts to predict the match results using in game statistics.

A. Game Details

To give an introduction of how a DotA game looks like – It's a 5 vs 5 mutliplayer game, consisting of 2 teams (5 players each in team) where the objective is to defeat the enemy by destroying their most important building which is called 'Ancient'. The two teams are called Radiant and Dire and they have to defend their ancients – hence the name Defense of the Ancients (DotA).

In the game there are various other elements of choosing a hero, role, gaining economy-gold, levels of characters, the spells and powers of the characters/heroes, the hitpoints/life (hp) of the hero, the mana of the heroes, various items which are used to enhance abilities and add new ones

The map always remains the same and consists of bases of team on each side(fountain), 3 lanes on each side, Shops to purchase items, Roshans den, tower structures and various other elements



During the game the players upgrade their heroes and levels, buy different items, kill opponent heroes, farm creeps/deny creeps. The goal of the game is to destroy enemy Ancient which is near the fountain, hence no draws are possible for a game.

B. Model Details

The features being used from the data are the match details and statistics which are fed into a Light GBM based model to predict the game outcome. The initial dataset consisted of a set of around 250 Features which have been evaluated and customized and trimmed down to finally 50 Features based on the variable importance of the feature set used for the AUC improvement. The notebook for feature evaluation is available in the app path :

dota_analytics/prediction_model/dota_lgb_init.ipynb

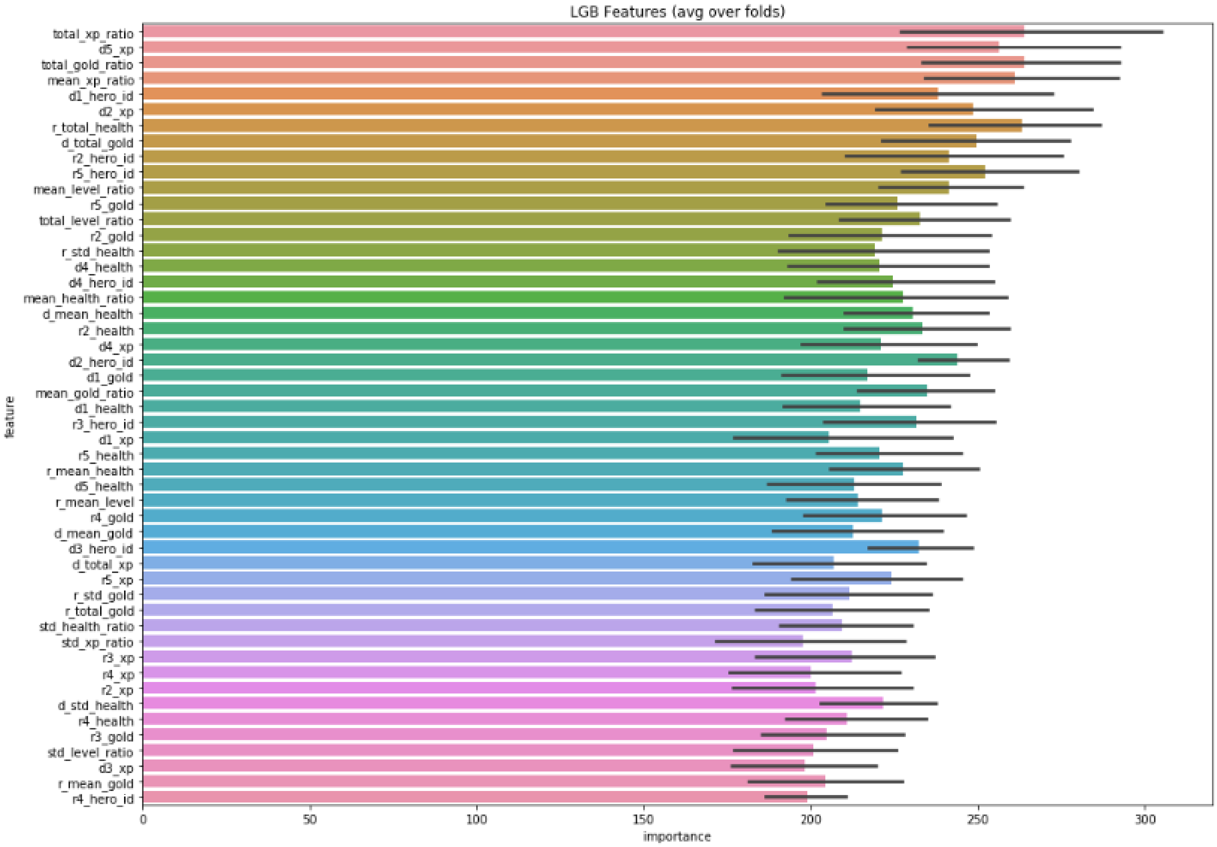
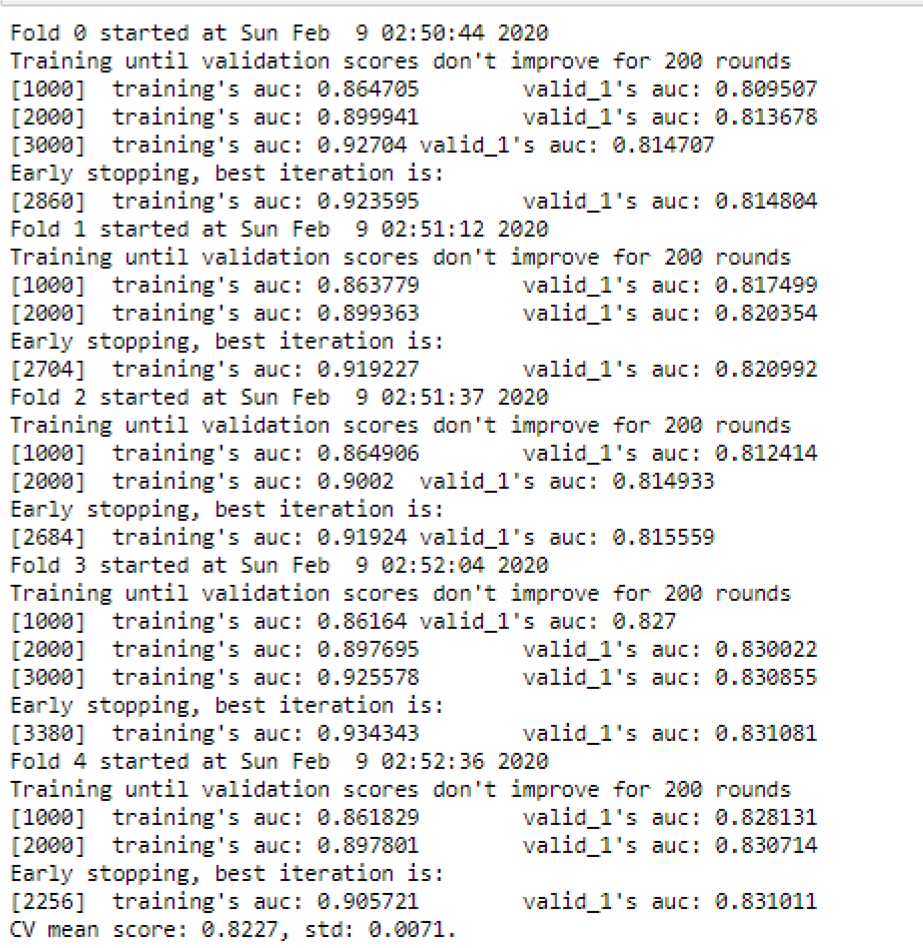


Figure : Variable Importance based on LGB

The training data consists of 40k matches on which the model is trained. The test data consists of 10k matches.

The model was cross validated using 5 folds and an average CV score of 82.27% was achieved for the predictions which is impressive for such a highly complex strategic game.



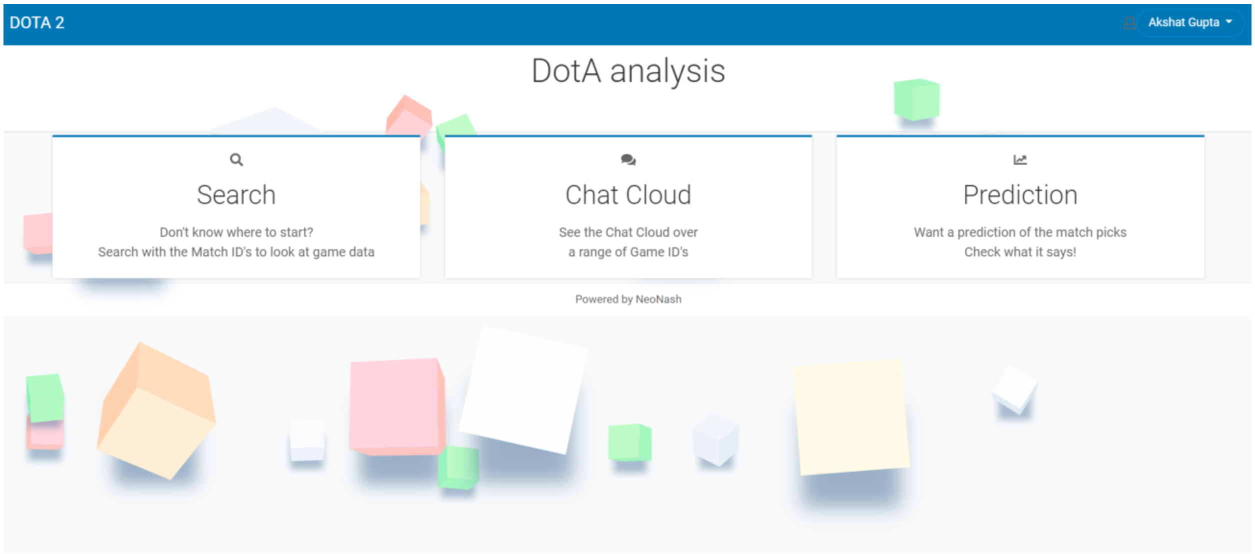
```
Fold 0 started at Sun Feb  9 02:50:44 2020
Training until validation scores don't improve for 200 rounds
[1000]  training's auc: 0.864705        valid_1's auc: 0.809507
[2000]  training's auc: 0.899941        valid_1's auc: 0.813678
[3000]  training's auc: 0.92704 valid_1's auc: 0.814707
Early stopping, best iteration is:
[2860]  training's auc: 0.923595        valid_1's auc: 0.814804
Fold 1 started at Sun Feb  9 02:51:12 2020
Training until validation scores don't improve for 200 rounds
[1000]  training's auc: 0.863779        valid_1's auc: 0.817499
[2000]  training's auc: 0.899363        valid_1's auc: 0.820354
Early stopping, best iteration is:
[2704]  training's auc: 0.919227        valid_1's auc: 0.820992
Fold 2 started at Sun Feb  9 02:51:37 2020
Training until validation scores don't improve for 200 rounds
[1000]  training's auc: 0.864906        valid_1's auc: 0.812414
[2000]  training's auc: 0.9002  valid_1's auc: 0.814933
Early stopping, best iteration is:
[2684]  training's auc: 0.91924 valid_1's auc: 0.815559
Fold 3 started at Sun Feb  9 02:52:04 2020
Training until validation scores don't improve for 200 rounds
[1000]  training's auc: 0.86164 valid_1's auc: 0.827
[2000]  training's auc: 0.897695        valid_1's auc: 0.830022
[3000]  training's auc: 0.925578        valid_1's auc: 0.830855
Early stopping, best iteration is:
[3380]  training's auc: 0.934343        valid_1's auc: 0.831081
Fold 4 started at Sun Feb  9 02:52:36 2020
Training until validation scores don't improve for 200 rounds
[1000]  training's auc: 0.861829        valid_1's auc: 0.828131
[2000]  training's auc: 0.897801        valid_1's auc: 0.830714
Early stopping, best iteration is:
[2256]  training's auc: 0.905721        valid_1's auc: 0.831011
CV mean score: 0.8227, std: 0.0071.
```

Figure : Average Cross Validation AUC score

C. Application Details

The application is based on the Django Web app framework with MySQL as backend database. Its based on MTV(Model Template View) Architecture and is created based on service pipeline where the services are being implemented to run the predictions and fetching the data from the models via database. The architecture details are not followed as strictly. The front end is based on HTML/ Bootstrap with Jquery/Javascript as scripting framework. The frontend resides inside the Views which serves the HTML renderer and connects the Model to the requests via the views on the server side. Architecture explained more in detail in section 4. Clean Code Development – (8. Architecure)

The landing page after the user logs into the app looks as below



DOTA 2

Akshat Gupta

DotA analysis

Search

Don't know where to start?
Search with the Match ID's to look at game data

Chat Cloud

See the Chat Cloud over
a range of Game ID's

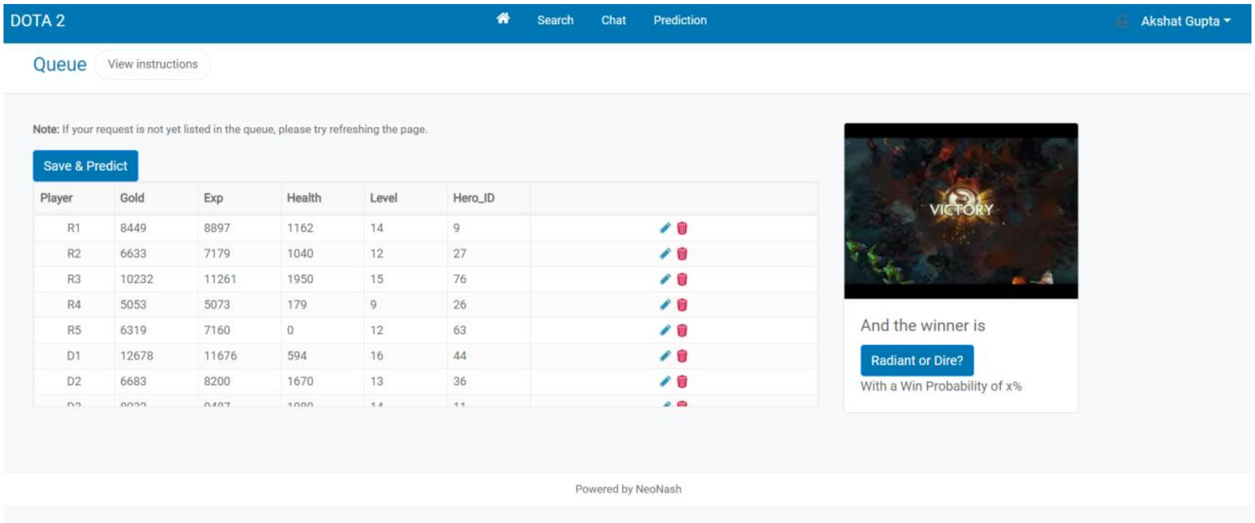
Prediction

Want a prediction of the match picks
Check what it says!

Powered by NeoNash

Figure : Landing page of the application Dashboard

The user can use the Prediction tab to give his inputs of the match details or use a prepopulated sample input and predict the outcome of the game



DOTA 2

Queue View instructions

Note: If your request is not yet listed in the queue, please try refreshing the page.

Save & Predict

Player	Gold	Exp	Health	Level	Hero_ID
R1	8449	8897	1162	14	9
R2	6633	7179	1040	12	27
R3	10232	11261	1950	15	76
R4	5053	5073	179	9	26
R5	6319	7160	0	12	63
D1	12678	11676	594	16	44
D2	6683	8200	1670	13	36
D3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

And the winner is

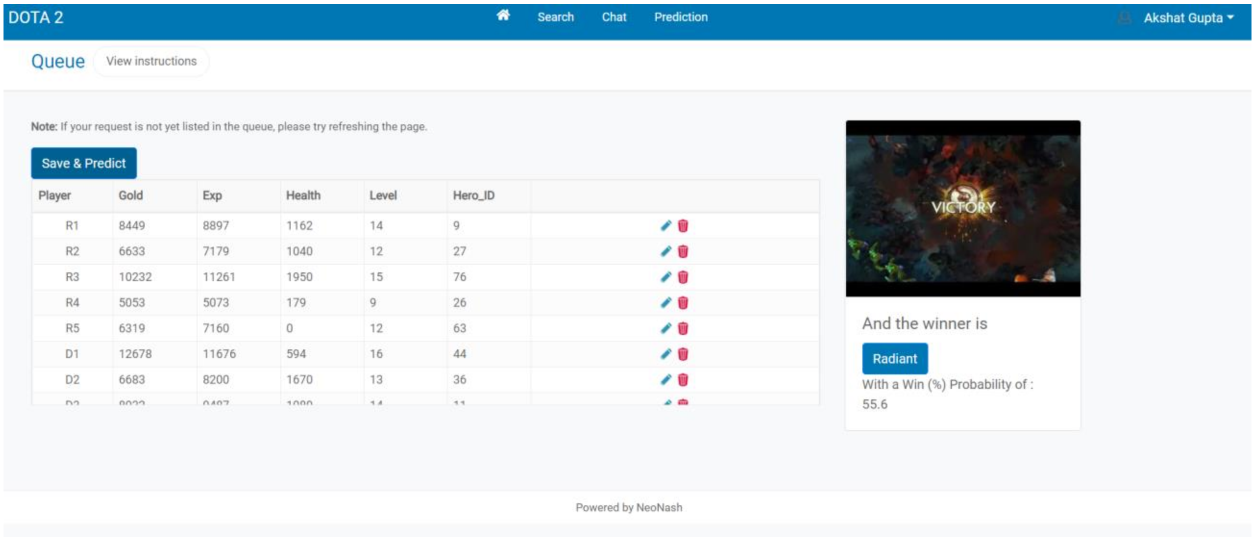
Radiant or Dire?

With a Win Probability of x%

Powered by NeoNash

Figure : Prediction queue landing page

After the user saves the updated details and clicks on Predict the Model evaluates the given data and predicts the game outcome i.e. The team which is going to win (Radiant or Dire) along with the probability of winning (0-100%).



Note: If your request is not yet listed in the queue, please try refreshing the page.

Player	Gold	Exp	Health	Level	Hero_ID
R1	8449	8897	1162	14	9
R2	6633	7179	1040	12	27
R3	10232	11261	1950	15	76
R4	5053	5073	179	9	26
R5	6319	7160	0	12	63
D1	12678	11676	594	16	44
D2	6683	8200	1670	13	36

And the winner is

With a Win (%) Probability of : 55.6

Figure : Output of the predictions for the given data

In this case the model predicts the winner as *Radiant* with a probability of *55.6%* which is correct when compared with the real match statistics

2. UML Diagrams

A. Class Diagram

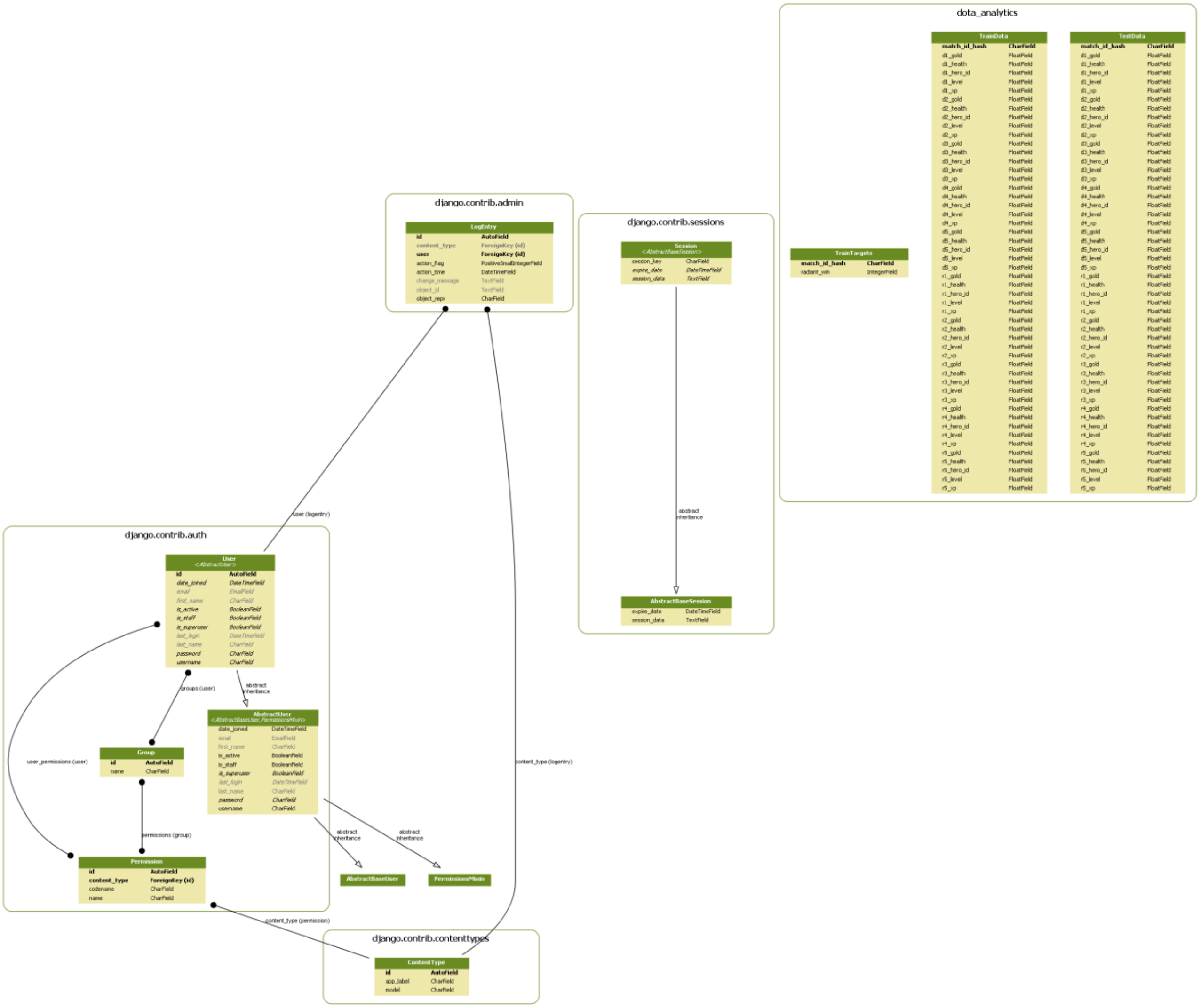


Figure : Class Diagram for the Database objects and Models

B. Use Case Diagram

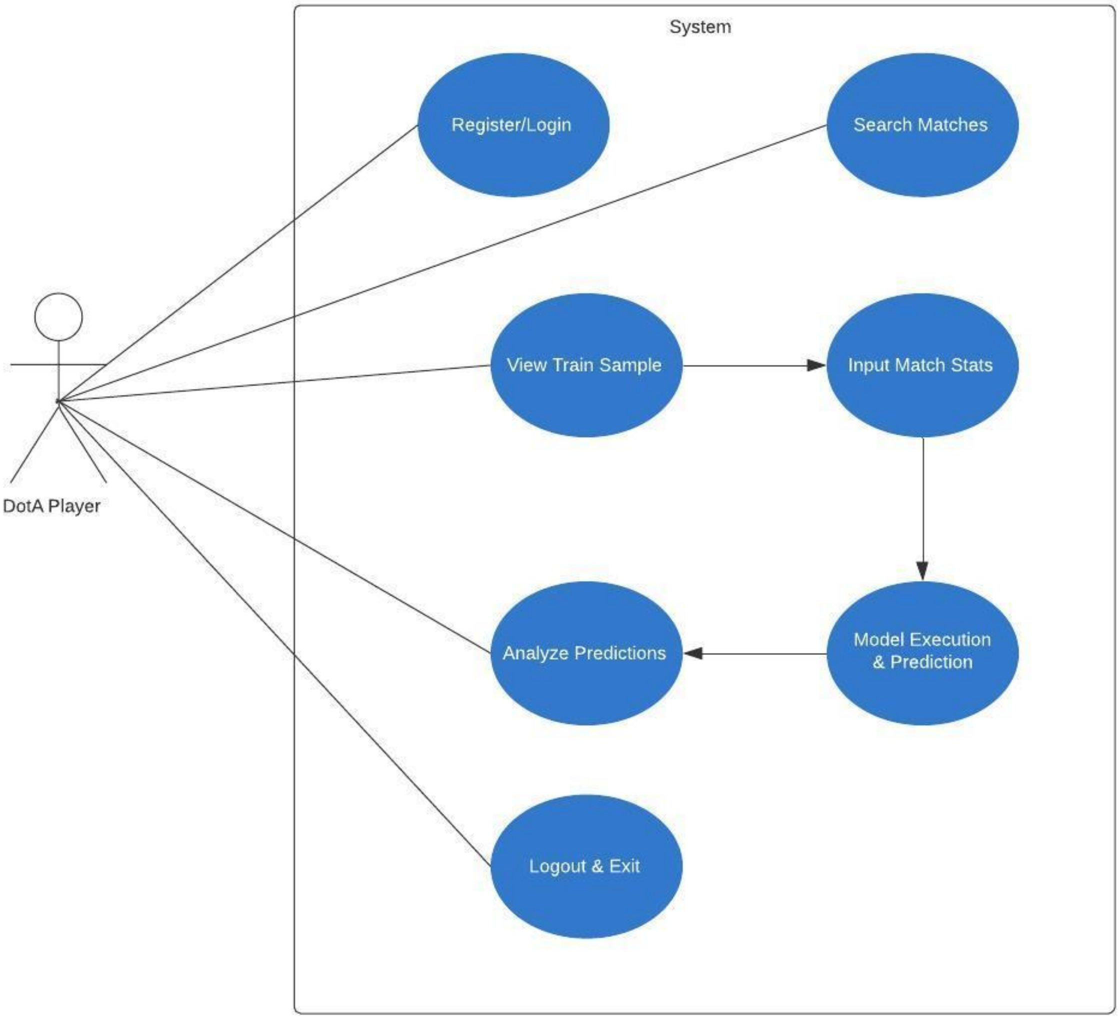


Figure : Use Case Diagram showing Interaction of User with the System

C. Activity Diagram

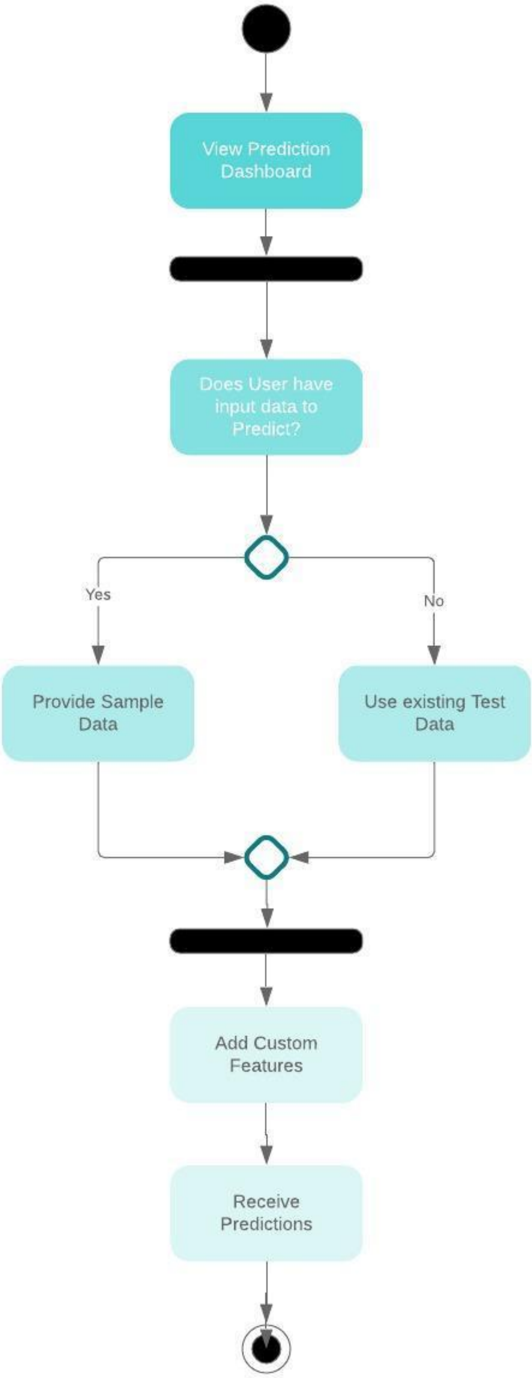


Figure : Activity Diagram to represent the Predictive Generation activity

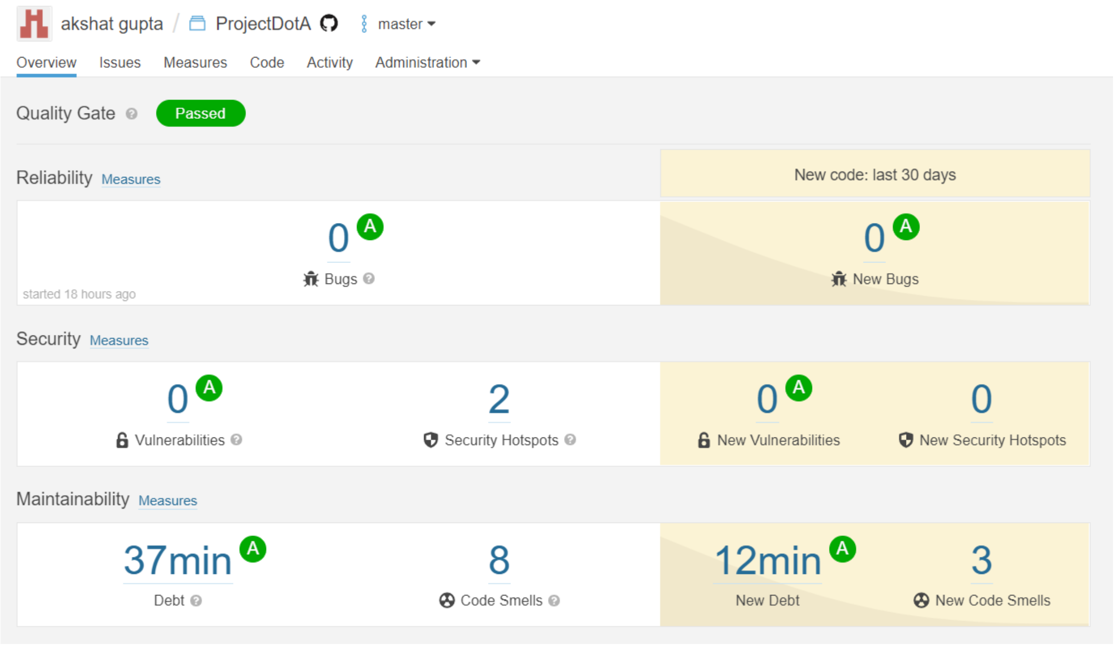
3. Metrics

There are 2 Metrics Performance Tools used, Sonarcloud and Codacy

A. SonarCloud

The metrics for SonarCloud are available here : https://sonarcloud.io/dashboard?id=neonash_ProjectDotA

The metrics were Passed with Grade A



akshat gupta / ProjectDotA master

Overview Issues Measures Code Activity Administration

Quality Gate Passed

Reliability Measures

New code: last 30 days

0 A Bugs

started 18 hours ago

0 A New Bugs

Security Measures

0 A Vulnerabilities

2 Security Hotspots

0 A New Vulnerabilities

0 New Security Hotspots

Maintainability Measures

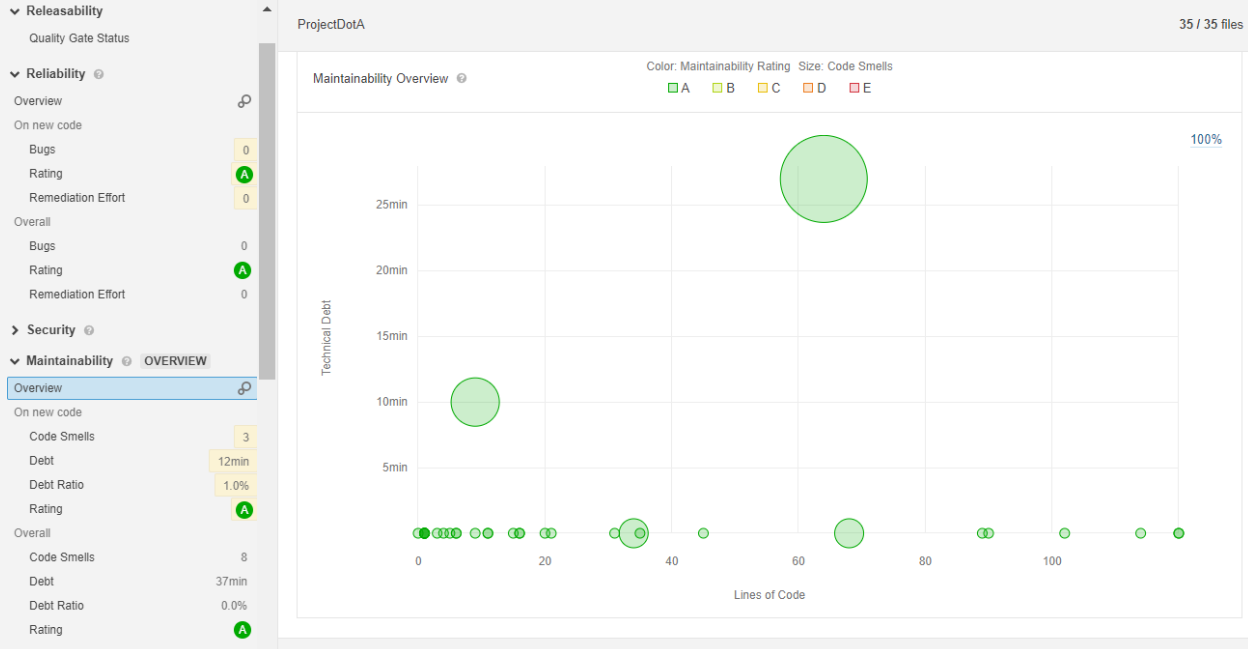
37min A Debt

8 Code Smells

12min A New Debt

3 New Code Smells

Figure : Overview of SonarCloud Dashboard



Releasability

Quality Gate Status

Reliability

Overview	
On new code	
Bugs	0
Rating	A
Remediation Effort	0
Overall	
Bugs	0
Rating	A
Remediation Effort	0

Security

Maintainability OVERVIEW

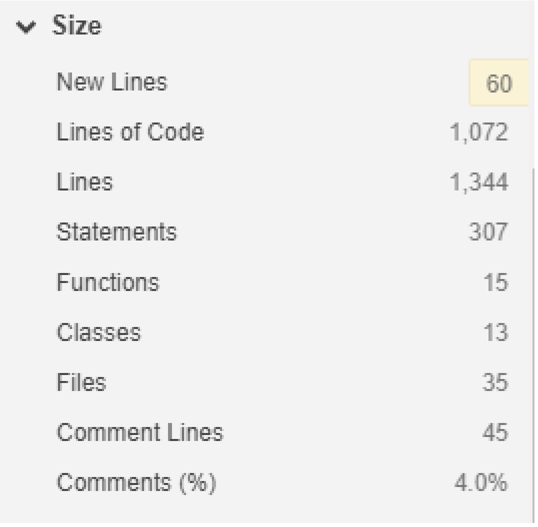
Overview	
On new code	
Code Smells	3
Debt	12min
Debt Ratio	1.0%
Rating	A
Overall	
Code Smells	8
Debt	37min
Debt Ratio	0.0%
Rating	A

ProjectDotA

35 / 35 files

Maintainability Overview

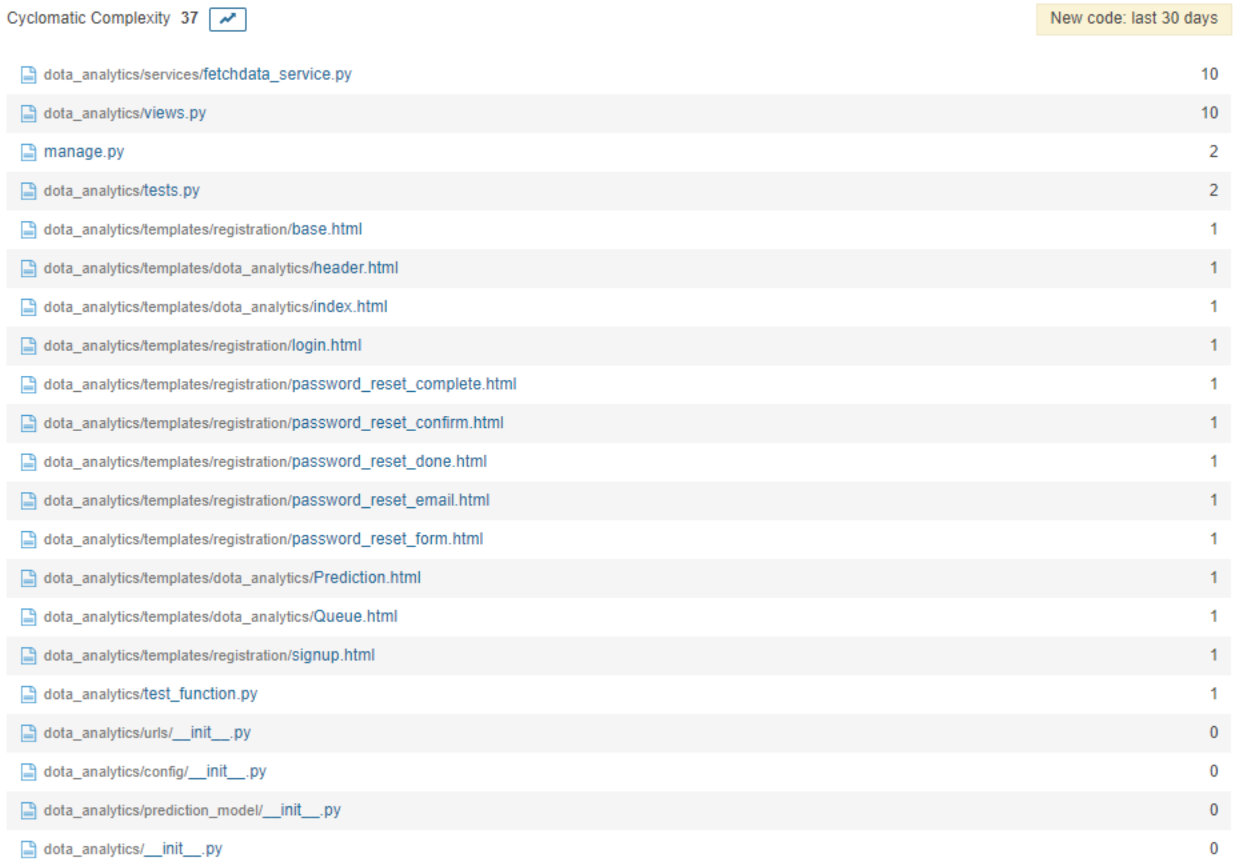
Figure : Overview of Maintainability



Size

New Lines	60
Lines of Code	1,072
Lines	1,344
Statements	307
Functions	15
Classes	13
Files	35
Comment Lines	45
Comments (%)	4.0%

Figure : Overview of Code Size Measures



Cyclomatic Complexity 37

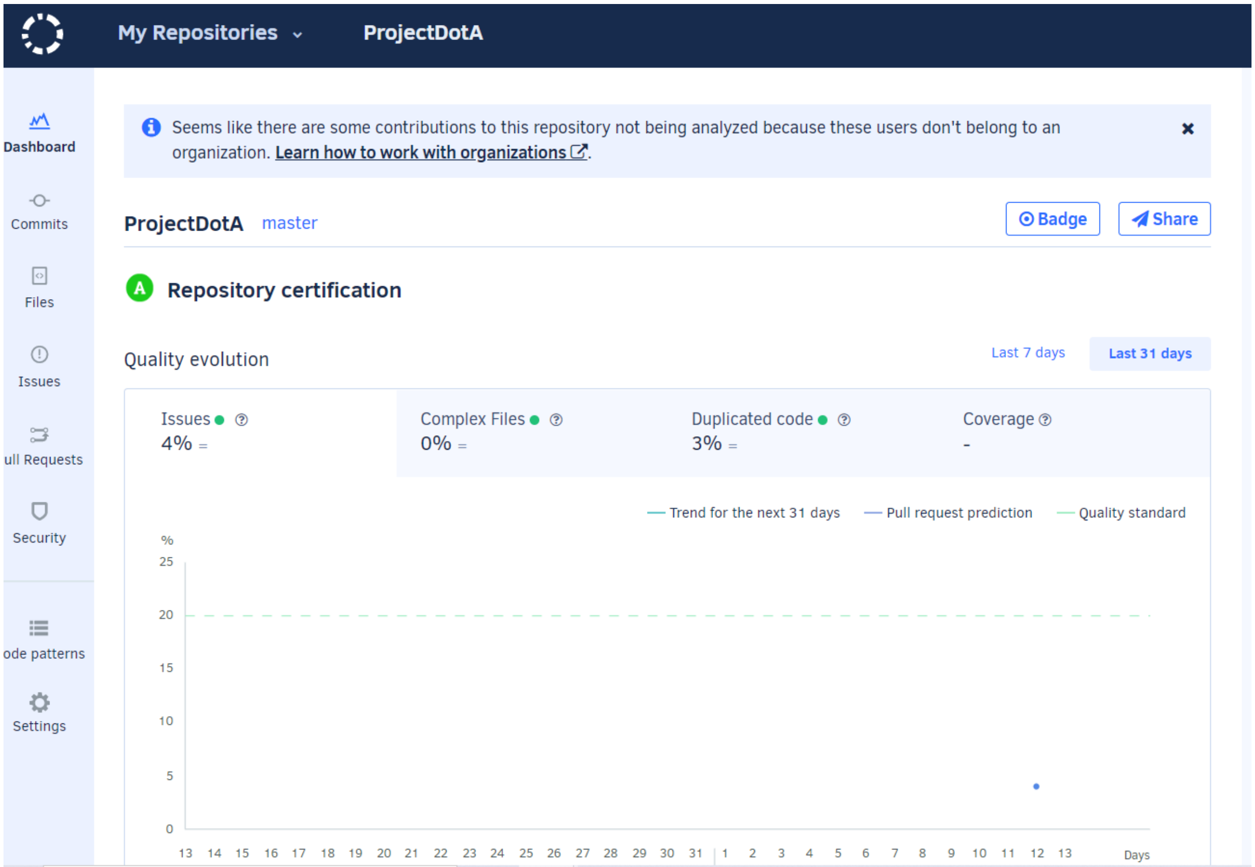
New code: last 30 days

dota_analytics/services/fetchdata_service.py	10
dota_analytics/views.py	10
manage.py	2
dota_analytics/tests.py	2
dota_analytics/templates/registration/base.html	1
dota_analytics/templates/dota_analytics/header.html	1
dota_analytics/templates/dota_analytics/index.html	1
dota_analytics/templates/registration/login.html	1
dota_analytics/templates/registration/password_reset_complete.html	1
dota_analytics/templates/registration/password_reset_confirm.html	1
dota_analytics/templates/registration/password_reset_done.html	1
dota_analytics/templates/registration/password_reset_email.html	1
dota_analytics/templates/registration/password_reset_form.html	1
dota_analytics/templates/dota_analytics/Prediction.html	1
dota_analytics/templates/dota_analytics/Queue.html	1
dota_analytics/templates/registration/signup.html	1
dota_analytics/test_function.py	1
dota_analytics/urls/__init__.py	0
dota_analytics/config/__init__.py	0
dota_analytics/prediction_model/__init__.py	0
dota_analytics/__init__.py	0

Figure : Overview of Cyclomatic Complexity

B. Codacy

The link to the Codacy measurements is here : https://app.codacy.com/manual/neonash/ProjectDotA/dashboard



Seems like there are some contributions to this repository not being analyzed because these users don't belong to an organization. Learn how to work with organizations.

ProjectDotA master

Repository certification A

Quality evolution

Issues ● 4% =

Complex Files ● 0% =

Duplicated code ● 3% =

Coverage -

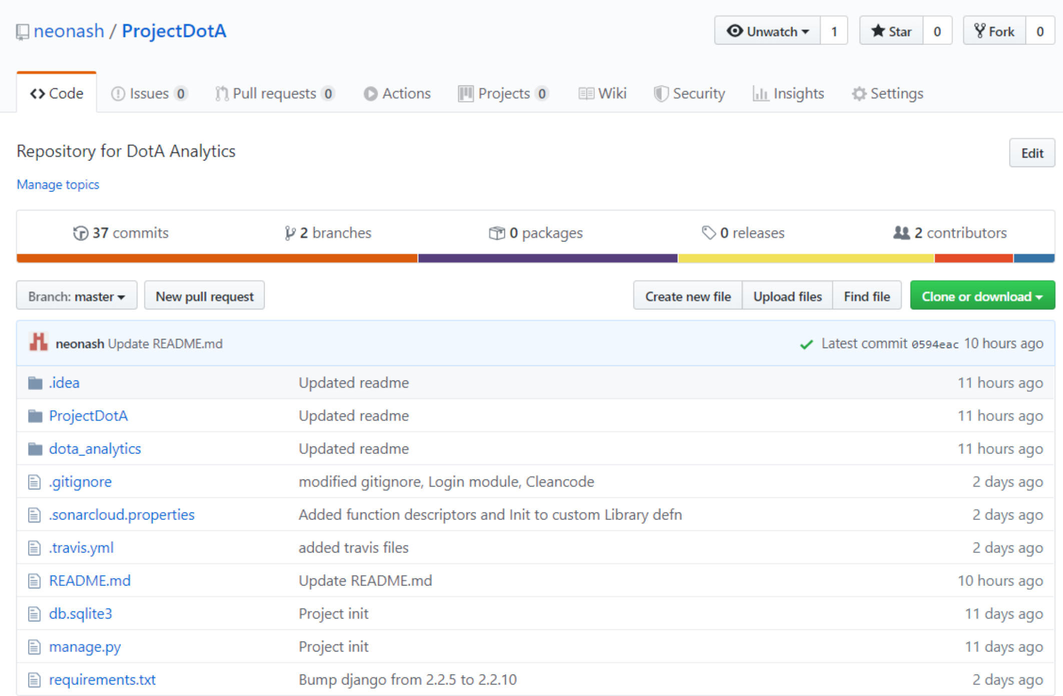
Figure : Overview of Codacy Dashboard

4. Clean Code Development

The Clean code development guide was followed and many concepts from the cheatsheet were used, some of the highlights shown below : *https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf*

A. Version Control

Github Repository was used for VCS and maintainability of the code and the project



Repository for DotA Analytics

37 commits | 2 branches | 0 packages | 0 releases | 2 contributors

neonash Update README.md — Latest commit 0594eac 10 hours ago

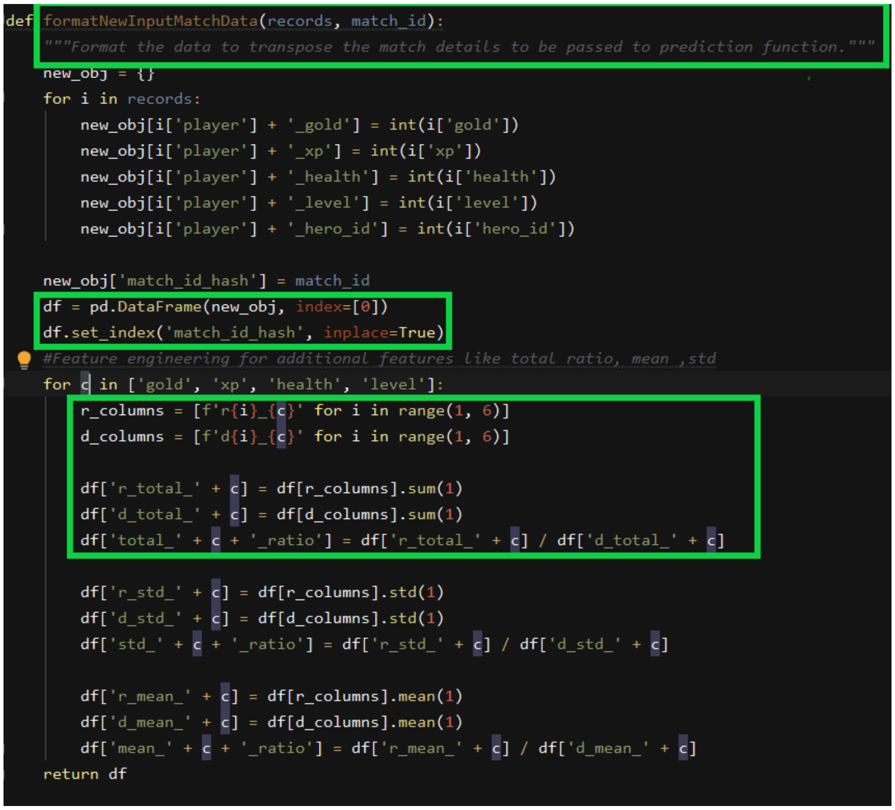
.idea	Updated readme	11 hours ago
ProjectDotA	Updated readme	11 hours ago
dota_analytics	Updated readme	11 hours ago
.gitignore	modified gitignore, Login module, Cleancode	2 days ago
.sonarcloud.properties	Added function descriptors and Init to custom Library defn	2 days ago
.travis.yml	added travis files	2 days ago
README.md	Update README.md	10 hours ago
db.sqlite3	Project init	11 days ago
manage.py	Project init	11 days ago
requirements.txt	Bump django from 2.2.5 to 2.2.10	2 days ago

B. Naming

Descriptive and Unambiguous Class, function names with their definitions were used, and example shown in the figure below – Highlighted in green

C. Source Code Structure

Vertical Separation methodology was followed where the variable declaration was kept close to its usage to minimize the scope – Highlighted in green

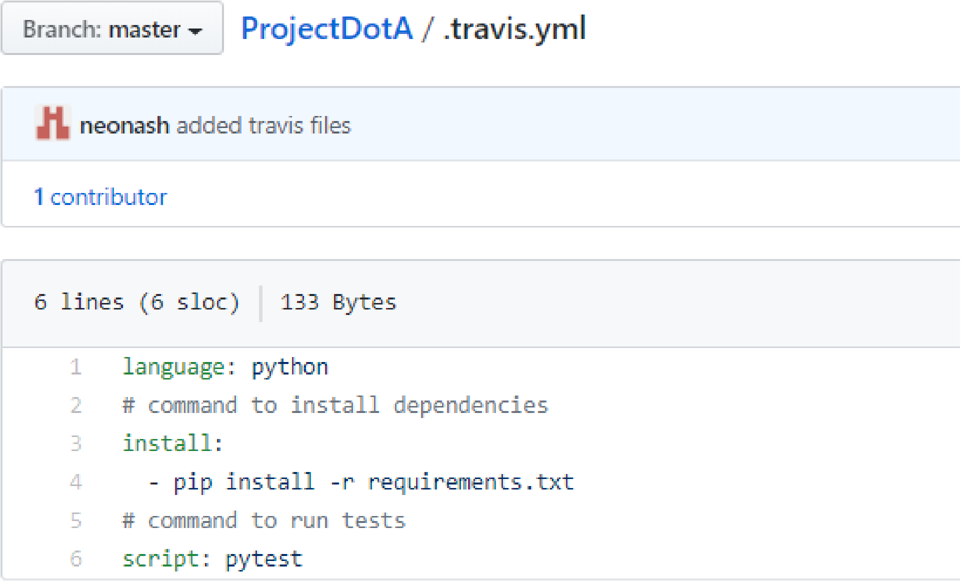


```
def formatNewInputMatchData(records, match_id):
    """Format the data to transpose the match details to be passed to prediction function."""
    new_obj = {}
    for i in records:
        new_obj[i['player'] + '_gold'] = int(i['gold'])
        new_obj[i['player'] + '_xp'] = int(i['xp'])
        new_obj[i['player'] + '_health'] = int(i['health'])
        new_obj[i['player'] + '_level'] = int(i['level'])
        new_obj[i['player'] + '_hero_id'] = int(i['hero_id'])

    new_obj['match_id_hash'] = match_id
    df = pd.DataFrame(new_obj, index=[0])
    df.set_index('match_id_hash', inplace=True)
    #Feature engineering for additional features like total ratio, mean ,std
    for c in ['gold', 'xp', 'health', 'level']:
        r_columns = [f'r{i}_{c}' for i in range(1, 6)]
        d_columns = [f'd{i}_{c}' for i in range(1, 6)]

        df['r_total_' + c] = df[r_columns].sum(1)
        df['d_total_' + c] = df[d_columns].sum(1)
        df['total_' + c + '_ratio'] = df['r_total_' + c] / df['d_total_' + c]

        df['r_std_' + c] = df[r_columns].std(1)
        df['d_std_' + c] = df[d_columns].std(1)
        df['std_' + c + '_ratio'] = df['r_std_' + c] / df['d_std_' + c]

        df['r_mean_' + c] = df[r_columns].mean(1)
        df['d_mean_' + c] = df[d_columns].mean(1)
        df['mean_' + c + '_ratio'] = df['r_mean_' + c] / df['d_mean_' + c]
    return df
```

Figure : Method Naming and Vertical Separation

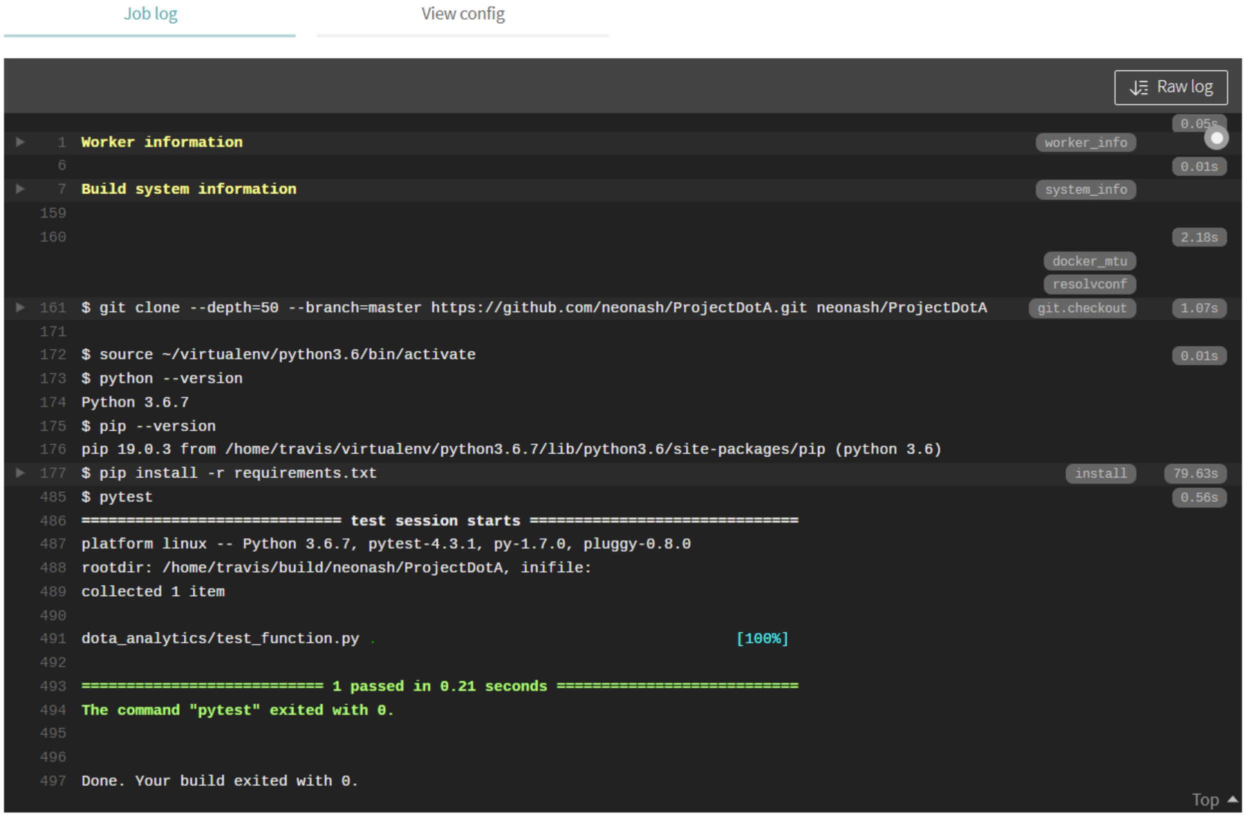
D. Continuous Integration

Travis CI was the platform which was used for building,testing and integration of the code with code analysis by SonarCloud. The *.travis.yml* file from github is shown along with the travis build logs



Branch: master
ProjectDotA / .travis.yml
neonash added travis files
1 contributor
6 lines (6 sloc) | 133 Bytes

```yaml
language: python
# command to install dependencies
install:
  - pip install -r requirements.txt
# command to run tests
script: pytest
```

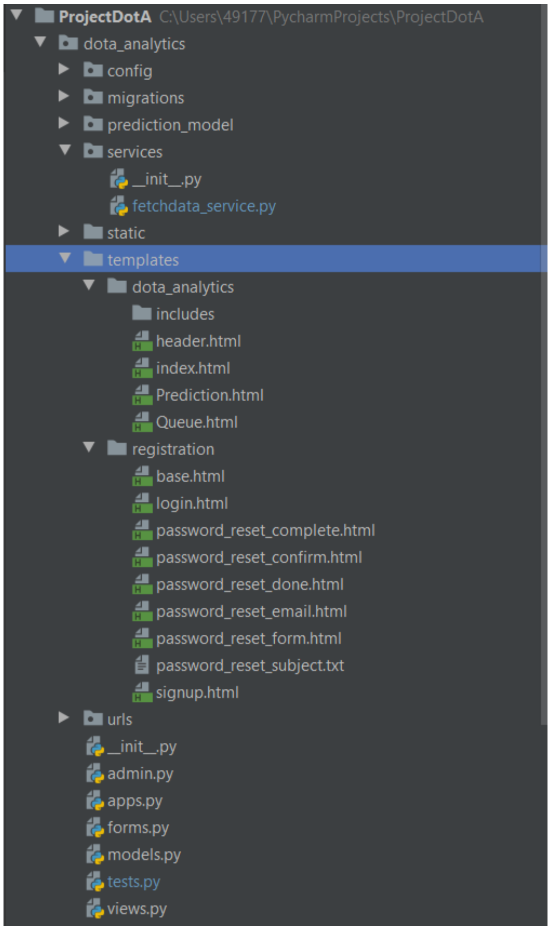


```
Worker information

Build system information


$ git clone --depth=50 --branch=master https://github.com/neonash/ProjectDotA.git neonash/ProjectDotA

$ source ~/virtualenv/python3.6/bin/activate
$ python --version
Python 3.6.7
$ pip --version
pip 19.0.3 from /home/travis/virtualenv/python3.6.7/lib/python3.6/site-packages/pip (python 3.6)
$ pip install -r requirements.txt
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.6.7, pytest-4.3.1, py-1.7.0, pluggy-0.8.0
rootdir: /home/travis/build/neonash/ProjectDotA, inifile:
collected 1 item

dota_analytics/test_function.py .                                       [100%]

=========================== 1 passed in 0.21 seconds ===========================
The command "pytest" exited with 0.


Done. Your build exited with 0.
```

E. Understandability

The Project structure along with variable and method naming convention has been kept consistent according to Camel Case to better understand the code.

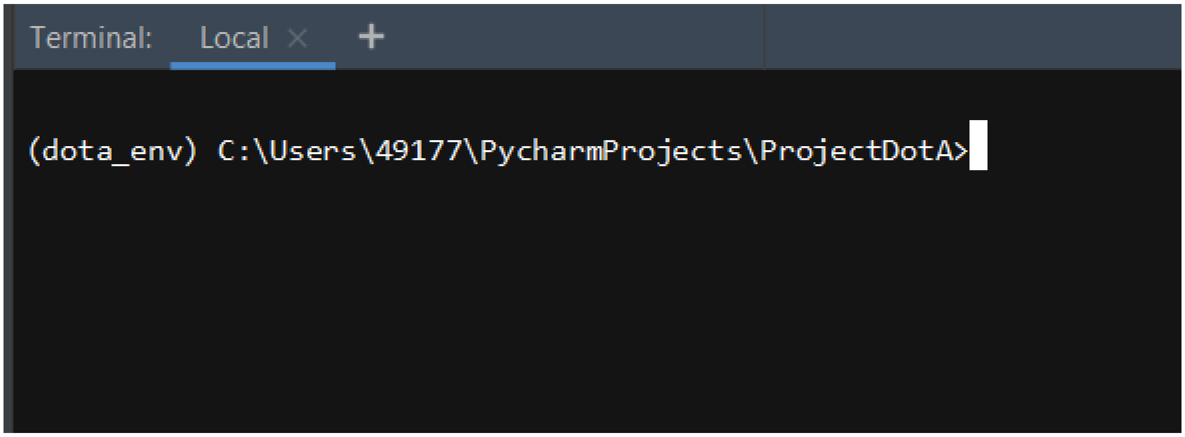
The files for the same purposes have been grouped together. An example shown below for templating front end files together with the split being in registration and the app folder itself. This methodology has been followed in the entire project structure



```
▼ ProjectDotA C:\Users\49177\PycharmProjects\ProjectDotA
  ▼ dota_analytics
    ▶ config
    ▶ migrations
    ▶ prediction_model
    ▼ services
        __init__.py
        fetchdata_service.py
    ▶ static
    ▼ templates
      ▼ dota_analytics
          includes
          header.html
          index.html
          Prediction.html
          Queue.html
      ▼ registration
          base.html
          login.html
          password_reset_complete.html
          password_reset_confirm.html
          password_reset_done.html
          password_reset_email.html
          password_reset_form.html
          password_reset_subject.txt
          signup.html
    ▶ urls
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      views.py
```

F. Environment

The entire project was build into a single virtual environment so that the packages are separate and easy to differentiate from the users base packages. A clean environment paradigm was followed. Conda activated ‘dota_env’ shown below:

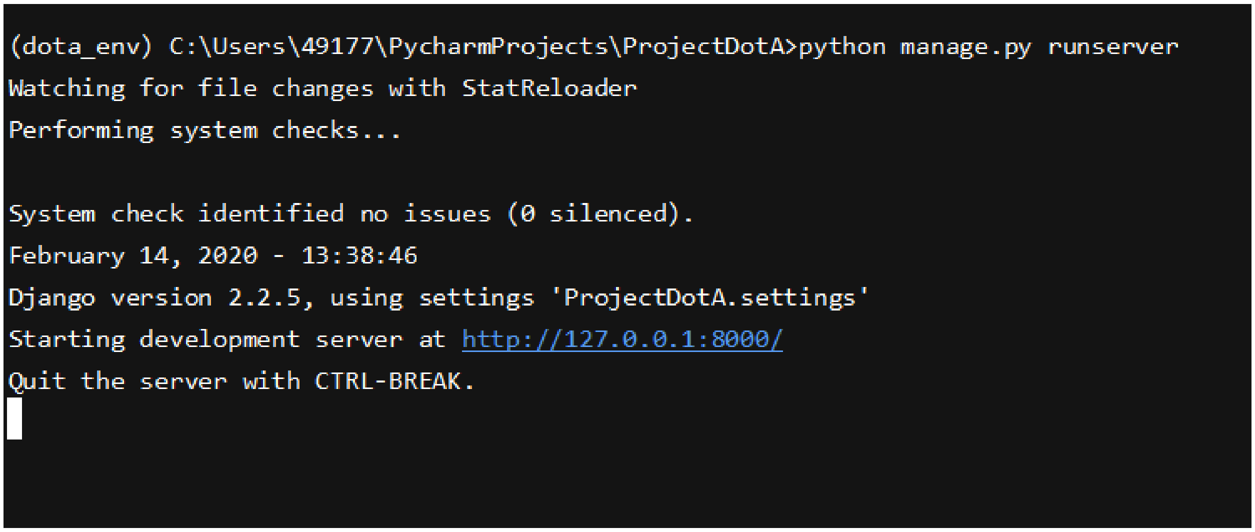


```
(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>
```

G. Environment execution requires only one step

The command which checks for the errors and shows if there are any issues else the project server and the app runs successfully – '*python manage.py runserver*'

All the DB connections, migrations, Project settings are checked with this single command



```
(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 14, 2020 - 13:38:46
Django version 2.2.5, using settings 'ProjectDotA.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

H. Architecture, Class and Separation of Concerns

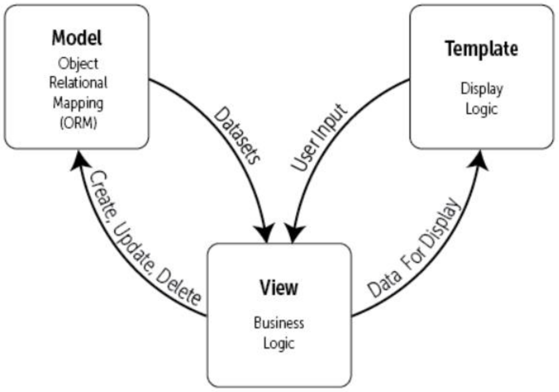
The project was objectively organized with a goal to Predict the match outcomes from the given statistics. This was carried out by following an MTV (Model-Template-View) architecture where each layer had its own data handling and displaying roles. An illustration of how the stack looks like from the client and server side is shown below as well.

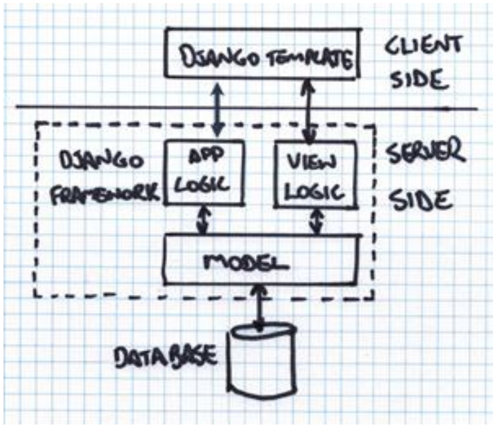
The **Templating** logic from Django is being handled by the front end at client side by HTML/CSS/JS

The Business and App logic is being handled by the **Views** via services and controls at the server side in the framework.

The Database ORM module is logically implemented as **Models**. The models here consists of 3 main classes of Training, Testing and TrainTargets

The template module triggers the actions from the display which triggers the View and the Model logic thus a cycle of interactions is built consisting of event and actions





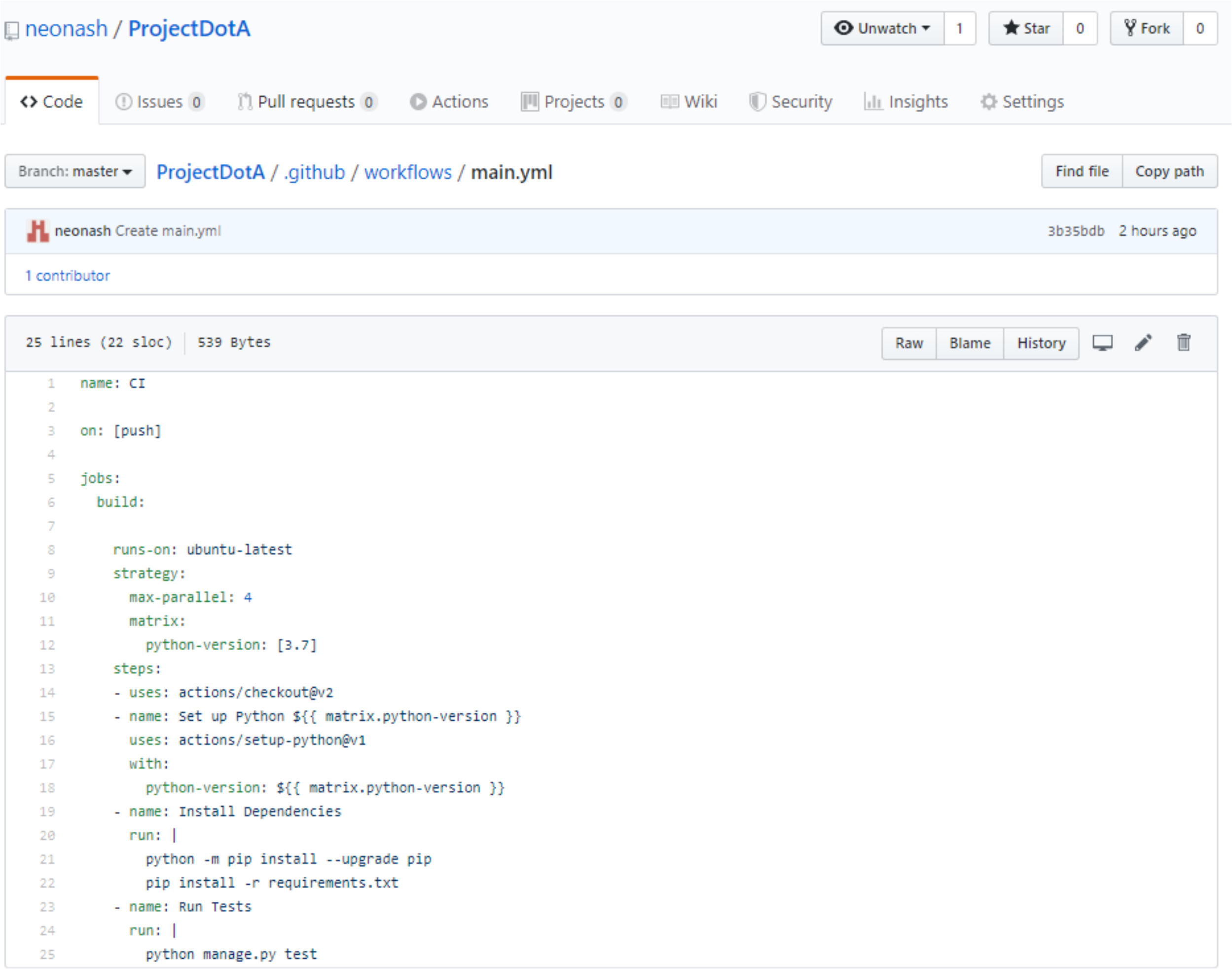
5. Build Management

The build management was implemented using Github Actions and Workflows. The *main.yaml* file from the Actions is listed below with the expansion for each command.

Initially a Python version 3.7 on which the project was developed is setup

Then, the requirements and dependency in Python are installed using “pip install -r requirements.txt”. And the pip version is upgraded if not up to date

The test cases are run using “python manage.py test“

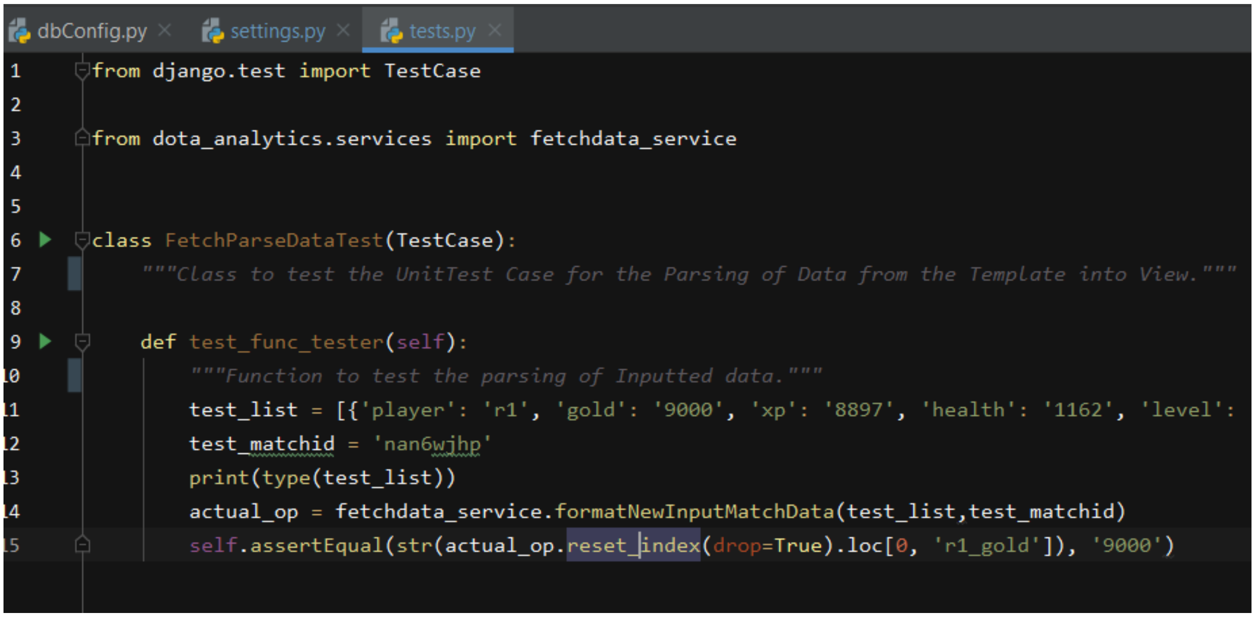


```yaml
name: CI

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      max-parallel: 4
      matrix:
        python-version: [3.7]
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v1
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install Dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Run Tests
      run: |
        python manage.py test
```

Figure : main.yaml file from Github Actions build

6. Unit Tests

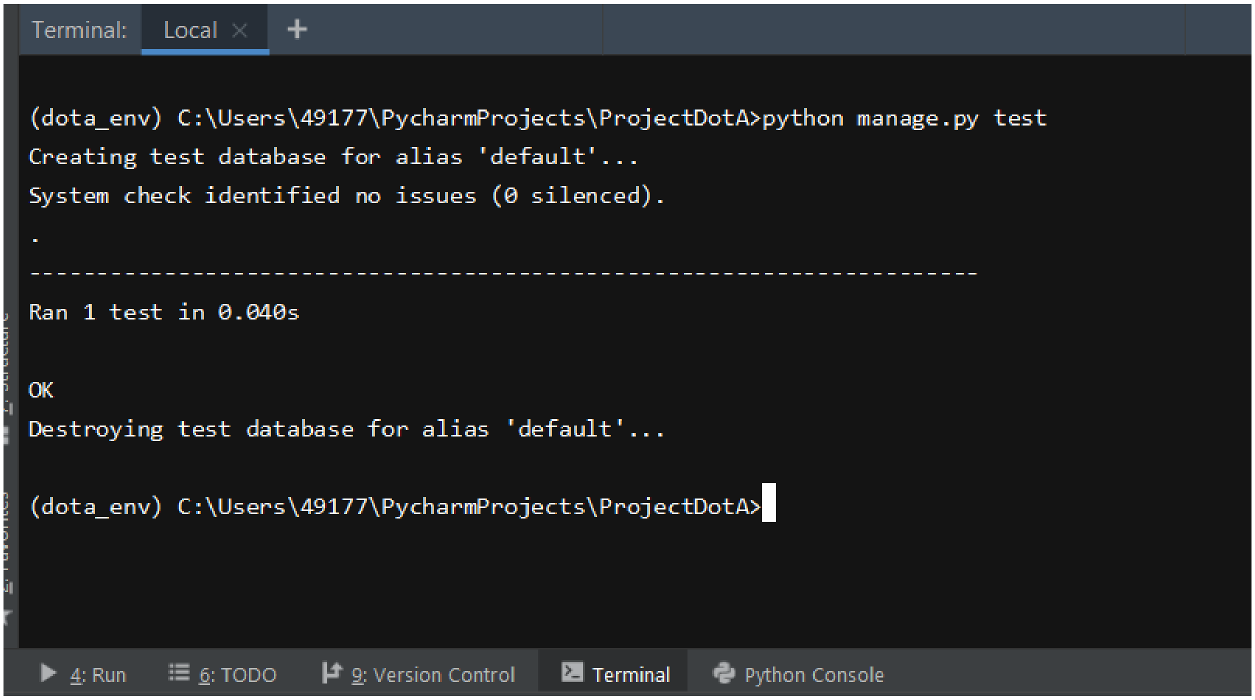
The testing framework was used as a part of Django framework. The library used for such tests is the default library being shipped with the Python Interpreter along with the Django framework. A model test for the parsing of the data and formatting is shown next which is a crucial test as this involves data integrity and synchronization amongst the core services for prediction.



```python
from django.test import TestCase

from dota_analytics.services import fetchdata_service


class FetchParseDataTest(TestCase):
    """Class to test the UnitTest Case for the Parsing of Data from the Template into View."""

    def test_func_tester(self):
        """Function to test the parsing of Inputted data."""
        test_list = [{'player': 'r1', 'gold': '9000', 'xp': '8897', 'health': '1162', 'level':
        test_matchid = 'nan6wjhp'
        print(type(test_list))
        actual_op = fetchdata_service.formatNewInputMatchData(test_list,test_matchid)
        self.assertEqual(str(actual_op.reset_index(drop=True).loc[0, 'r1_gold']), '9000')
```

Figure : Unit Test for Data Parsing from the template for Prediction

The model test case results are shown below



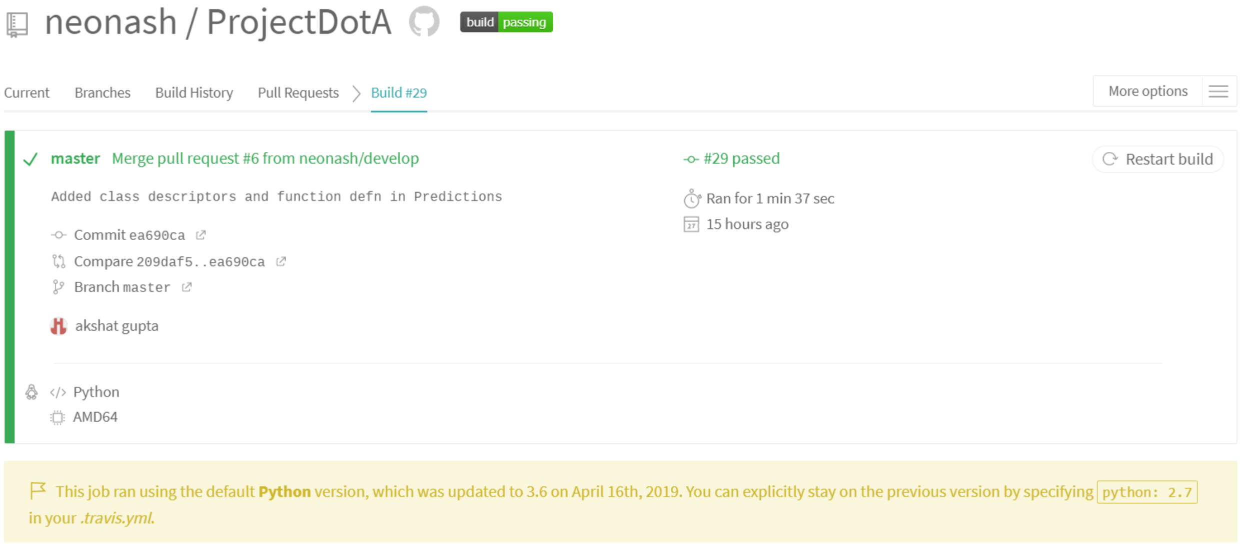
```
(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.040s

OK
Destroying test database for alias 'default'...

(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>
```

Figure : Successful test run

7. Continuous Delivery

The system for Continuous Integration and Delivery which was used was Travis CI. The .travis.yml was configured and available in Github. The link to dashboard is below:

https://travis-ci.org/neonash/ProjectDotA/



✓ **master** Merge pull request #6 from neonash/develop

`Added class descriptors and function defn in Predictions`

Commit `ea690ca`

Compare `209daf5..ea690ca`

Branch `master`

akshat gupta

#29 passed

Ran for 1 min 37 sec

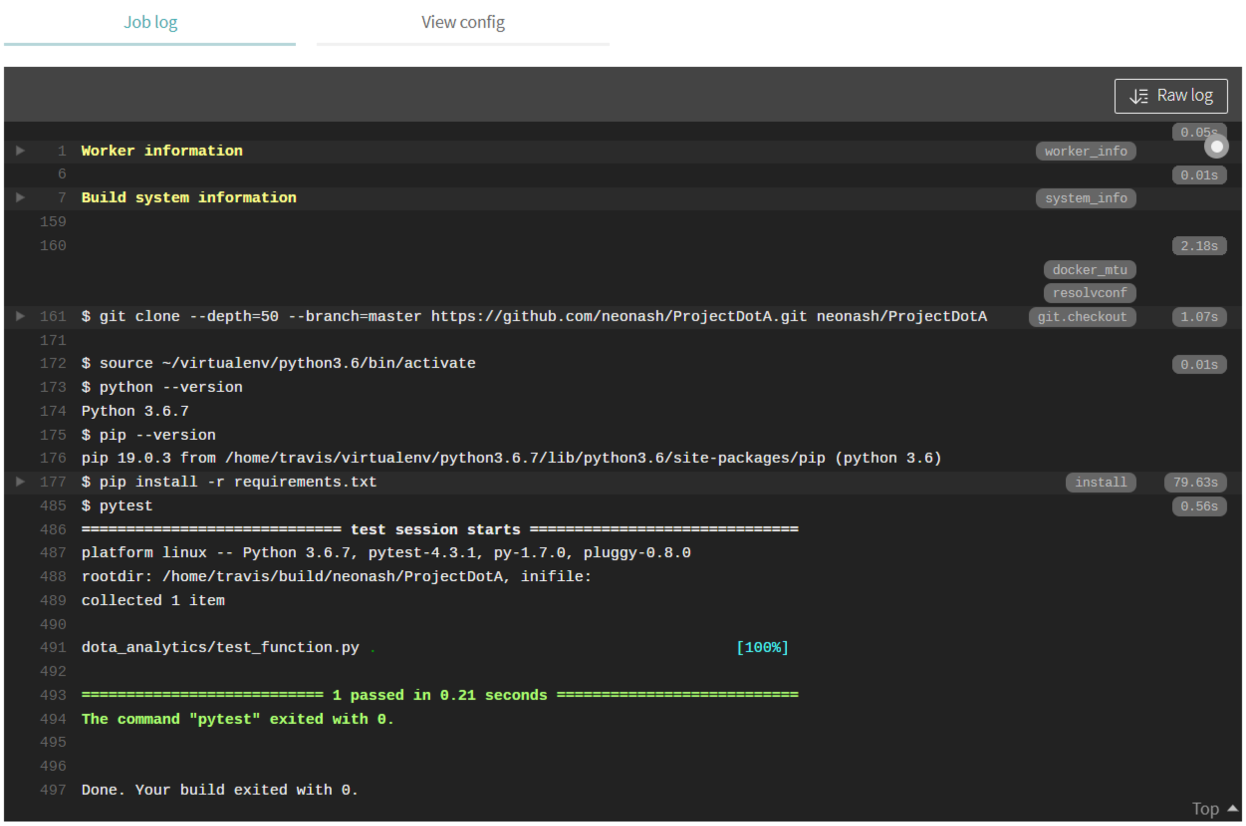
15 hours ago

Python

AMD64

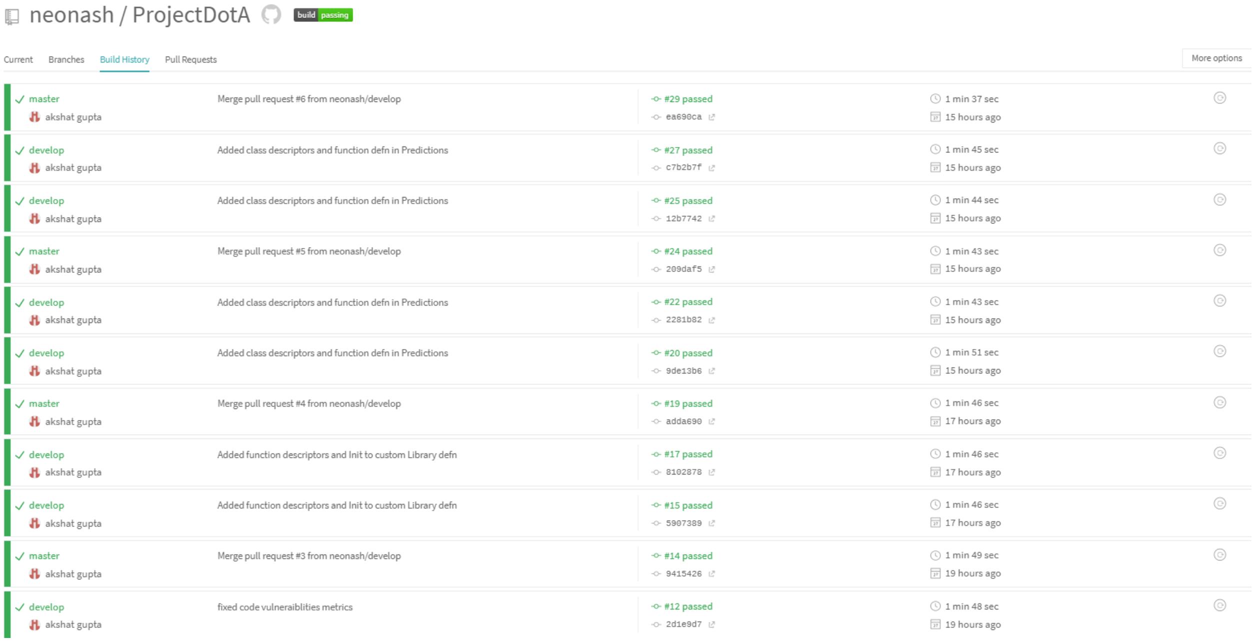
This job ran using the default **Python** version, which was updated to 3.6 on April 16th, 2019. You can explicitly stay on the previous version by specifying `python: 2.7` in your *.travis.yml*.

Figure : Build Dashboard



```
Worker information

Build system information



$ git clone --depth=50 --branch=master https://github.com/neonash/ProjectDotA.git neonash/ProjectDotA

$ source ~/virtualenv/python3.6/bin/activate
$ python --version
Python 3.6.7
$ pip --version
pip 19.0.3 from /home/travis/virtualenv/python3.6.7/lib/python3.6/site-packages/pip (python 3.6)
$ pip install -r requirements.txt
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.6.7, pytest-4.3.1, py-1.7.0, pluggy-0.8.0
rootdir: /home/travis/build/neonash/ProjectDotA, inifile:
collected 1 item

dota_analytics/test_function.py .                                        [100%]

=========================== 1 passed in 0.21 seconds ===========================
The command "pytest" exited with 0.


Done. Your build exited with 0.
```

Figure : Job Log

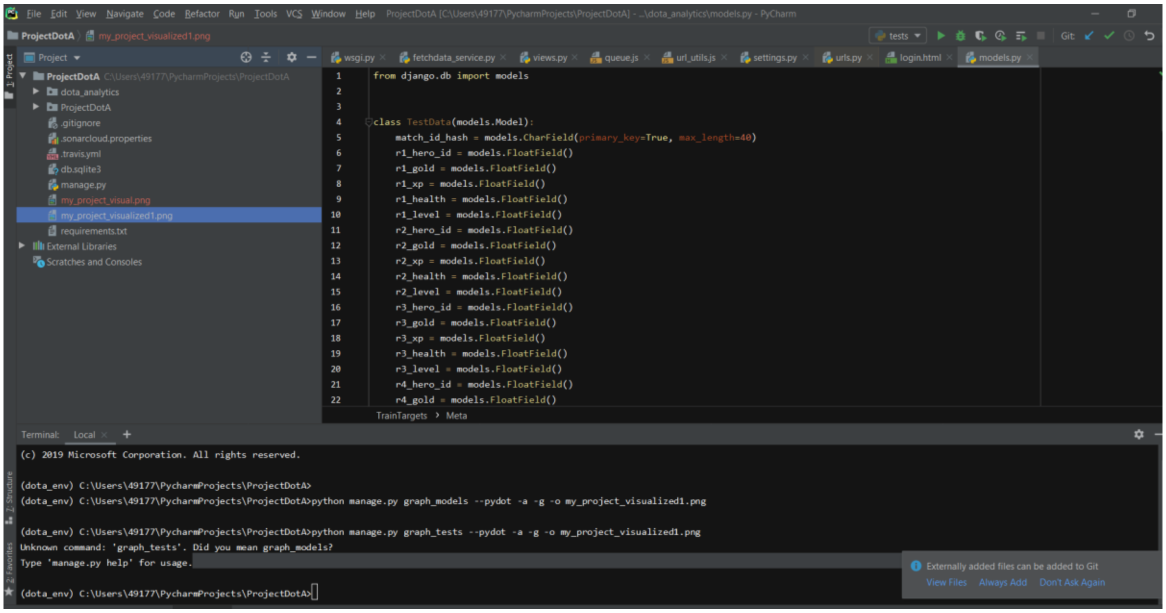


✓ master akshat gupta	Merge pull request #6 from neonash/develop	#29 passed ea690ca	1 min 37 sec 15 hours ago
✓ develop akshat gupta	Added class descriptors and function defn in Predictions	#27 passed c7b2b7f	1 min 45 sec 15 hours ago
✓ develop akshat gupta	Added class descriptors and function defn in Predictions	#25 passed 12b7742	1 min 44 sec 15 hours ago
✓ master akshat gupta	Merge pull request #5 from neonash/develop	#24 passed 209daf5	1 min 43 sec 15 hours ago
✓ develop akshat gupta	Added class descriptors and function defn in Predictions	#22 passed 2281b82	1 min 43 sec 15 hours ago
✓ develop akshat gupta	Added class descriptors and function defn in Predictions	#20 passed 9de13b6	1 min 51 sec 15 hours ago
✓ master akshat gupta	Merge pull request #4 from neonash/develop	#19 passed adda690	1 min 46 sec 17 hours ago
✓ develop akshat gupta	Added function descriptors and Init to custom Library defn	#17 passed 8102878	1 min 46 sec 17 hours ago
✓ develop akshat gupta	Added function descriptors and Init to custom Library defn	#15 passed 5907389	1 min 46 sec 17 hours ago
✓ master akshat gupta	Merge pull request #3 from neonash/develop	#14 passed 9415426	1 min 49 sec 19 hours ago
✓ develop akshat gupta	fixed code vulneraiblities metrics	#12 passed 2d1e9d7	1 min 48 sec 19 hours ago

Figure : Build History

8. Integrated Dev Environment

The IDE used here is PyCharm from Jetbrains. It is a very comprehensive and instructive IDE for Django Web app development in Python. Dark mode is my preference here , with some cool and handy shortcuts which I frequently used are listed below:



```
from django.db import models


class TestData(models.Model):
    match_id_hash = models.CharField(primary_key=True, max_length=40)
    r1_hero_id = models.FloatField()
    r1_gold = models.FloatField()
    r1_xp = models.FloatField()
    r1_health = models.FloatField()
    r1_level = models.FloatField()
    r2_hero_id = models.FloatField()
    r2_gold = models.FloatField()
    r2_xp = models.FloatField()
    r2_health = models.FloatField()
    r2_level = models.FloatField()
    r3_hero_id = models.FloatField()
    r3_gold = models.FloatField()
    r3_xp = models.FloatField()
    r3_health = models.FloatField()
    r3_level = models.FloatField()
    r4_hero_id = models.FloatField()
    r4_gold = models.FloatField()
```

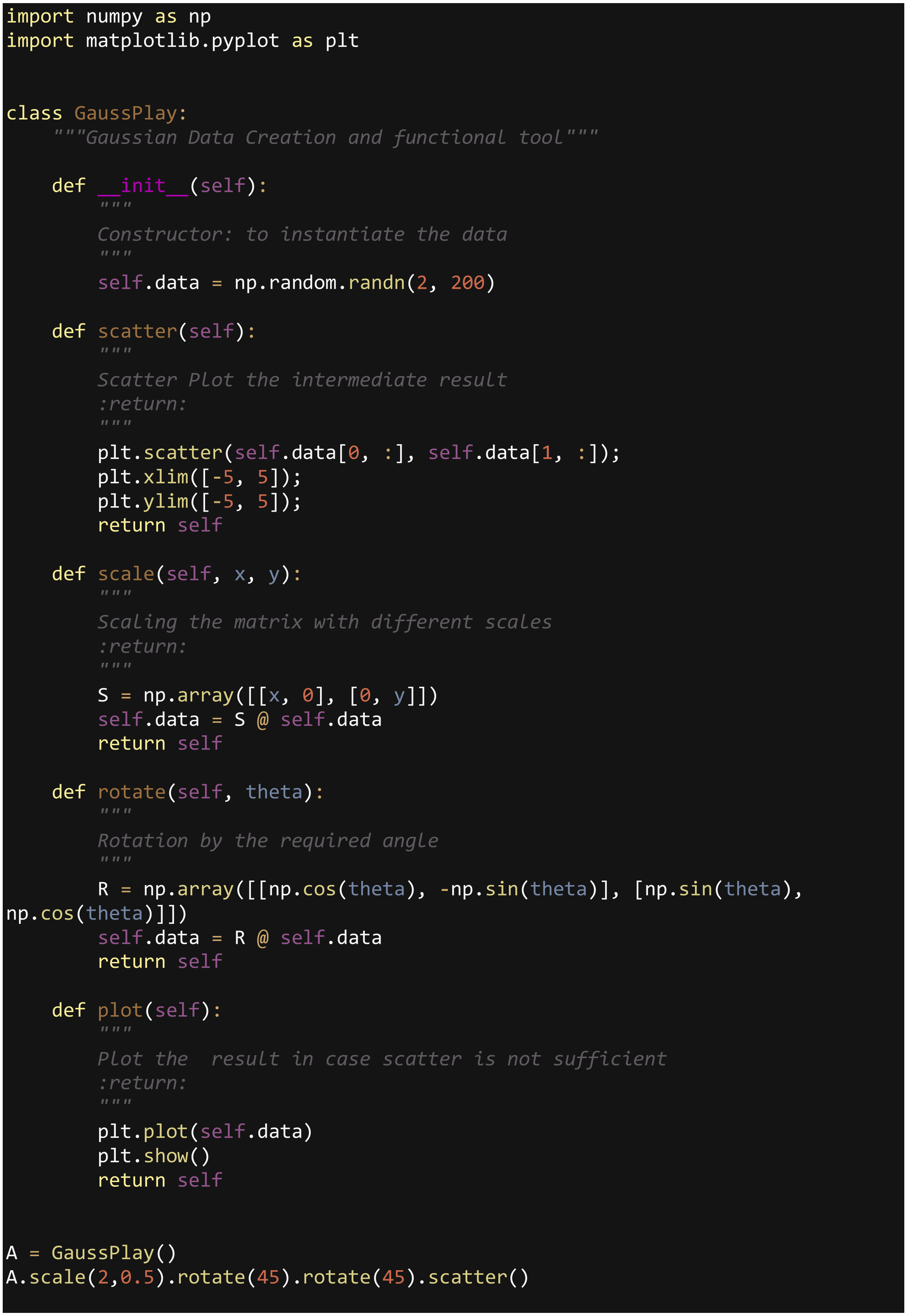
```
(c) 2019 Microsoft Corporation. All rights reserved.

(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>
(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>python manage.py graph_models --pydot -a -g -o my_project_visualized1.png

(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>python manage.py graph_tests --pydot -a -g -o my_project_visualized1.png
Unknown command: 'graph_tests'. Did you mean graph_models?
Type 'manage.py help' for usage.

(dota_env) C:\Users\49177\PycharmProjects\ProjectDotA>
```

Figure : A visual of the IDE

1. Double Shift : Find any file in the entire project
2. Ctrl + Shift + F : Find any text in the entire project
3. Ctrl + D : To duplicate the current line of code
4. Ctrl + G : To Goto a particular line or column in the file
5. Ctrl + R : Replace the content with another content or regex based checks

9. Domain Specific Language

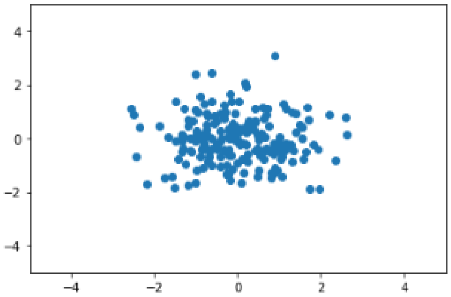
A DSL sample has been implemented for a random Gaussian Data which although is not related to the project but has been implemented separately in the code. This is an object oriented approach of building a DSL where the pipeline is generated by transforming the data repeatedly on different functions. The data generated is passed through a series of Class methods to obtain the desired result. The module is located under '*dota_analytics/dsl*'



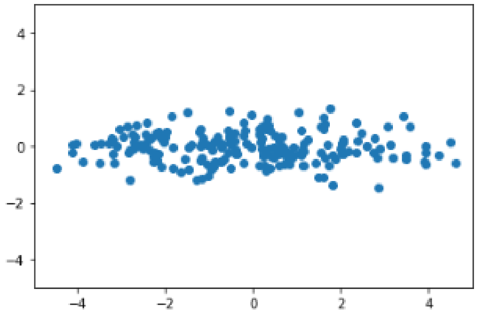
```
import numpy as np
import matplotlib.pyplot as plt


class GaussPlay:
    """Gaussian Data Creation and functional tool"""

    def __init__(self):
        """
        Constructor: to instantiate the data
        """
        self.data = np.random.randn(2, 200)

    def scatter(self):
        """
        Scatter Plot the intermediate result
        :return:
        """
        plt.scatter(self.data[0, :], self.data[1, :]);
        plt.xlim([-5, 5]);
        plt.ylim([-5, 5]);
        return self

    def scale(self, x, y):
        """
        Scaling the matrix with different scales
        :return:
        """
        S = np.array([[x, 0], [0, y]])
        self.data = S @ self.data
        return self

    def rotate(self, theta):
        """
        Rotation by the required angle
        """
        R = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta),
np.cos(theta)]])
        self.data = R @ self.data
        return self

    def plot(self):
        """
        Plot the  result in case scatter is not sufficient
        :return:
        """
        plt.plot(self.data)
        plt.show()
        return self


A = GaussPlay()
A.scale(2,0.5).rotate(45).rotate(45).scatter()
```

The example shown below generates the data , scales it twice on the X-axis and halves it in the Y-axis, then rotates it 2 times by 45 degree.

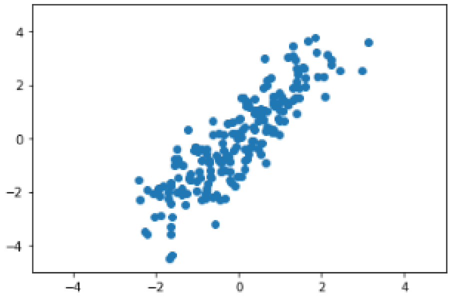
The steps are shown below:



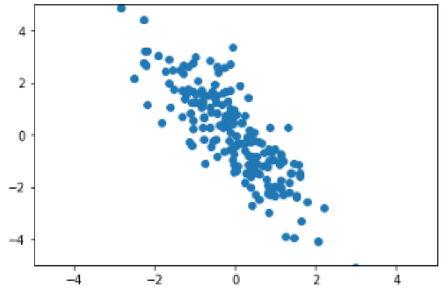
Original Data : A



After scaling one time



After first Rotation



After second Rotation

10. Functional Programming

The Architecture was based on Object Oriented Programming but the concepts of Functional programming were applied as well.

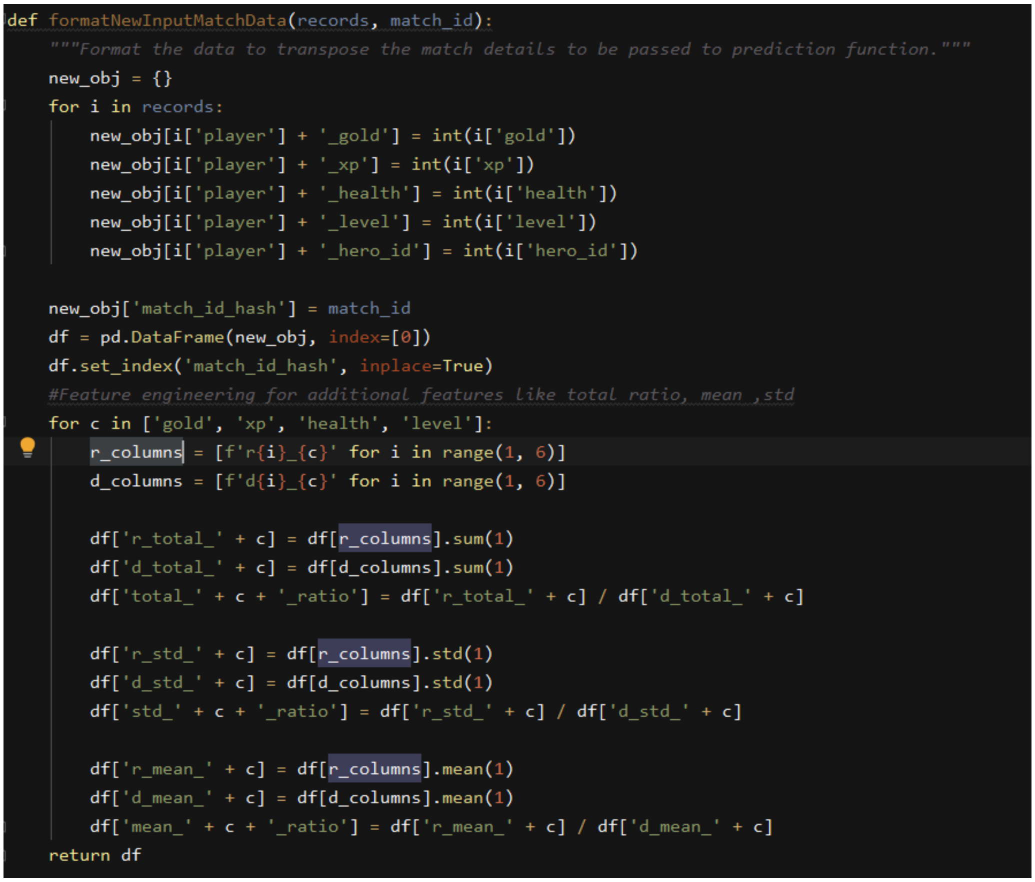
A. Final Data Structures

Since Python is not purely functional programming language, the mutability concept was a bit relaxed but the datastructures which were used were final data structures based on lists and dictionaries to pass the data across functions

B. Side Effect Free Functions

The functions were designed as Pure Functions which are stateless and will not mutate the data. Example of one such function is given below from the fetchdata_service.py

The *formatNewInputMatchData()* function can be run multiple times with any side effects and it will return the same value without any mutation of the existing variables

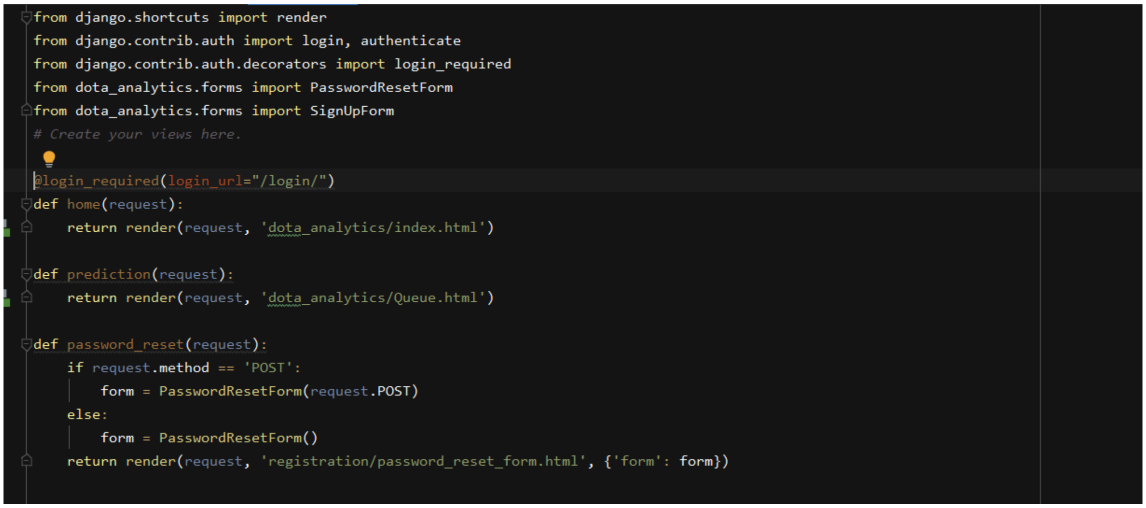


```
def formatNewInputMatchData(records, match_id):
    """Format the data to transpose the match details to be passed to prediction function."""
    new_obj = {}
    for i in records:
        new_obj[i['player'] + '_gold'] = int(i['gold'])
        new_obj[i['player'] + '_xp'] = int(i['xp'])
        new_obj[i['player'] + '_health'] = int(i['health'])
        new_obj[i['player'] + '_level'] = int(i['level'])
        new_obj[i['player'] + '_hero_id'] = int(i['hero_id'])

    new_obj['match_id_hash'] = match_id
    df = pd.DataFrame(new_obj, index=[0])
    df.set_index('match_id_hash', inplace=True)
    #Feature engineering for additional features like total ratio, mean ,std
    for c in ['gold', 'xp', 'health', 'level']:
        r_columns = [f'r{i}_{c}' for i in range(1, 6)]
        d_columns = [f'd{i}_{c}' for i in range(1, 6)]

        df['r_total_' + c] = df[r_columns].sum(1)
        df['d_total_' + c] = df[d_columns].sum(1)
        df['total_' + c + '_ratio'] = df['r_total_' + c] / df['d_total_' + c]

        df['r_std_' + c] = df[r_columns].std(1)
        df['d_std_' + c] = df[d_columns].std(1)
        df['std_' + c + '_ratio'] = df['r_std_' + c] / df['d_std_' + c]

        df['r_mean_' + c] = df[r_columns].mean(1)
        df['d_mean_' + c] = df[d_columns].mean(1)
        df['mean_' + c + '_ratio'] = df['r_mean_' + c] / df['d_mean_' + c]
    return df
```

Figure : Function showing no Side effects

C. Higher Order Functions

The decorators were used to implement the high order functionality. It is explained in more detail in the AOP (Aspect oriented programming section). This is available in *views.py* section of the Django app – *dota_analytics*



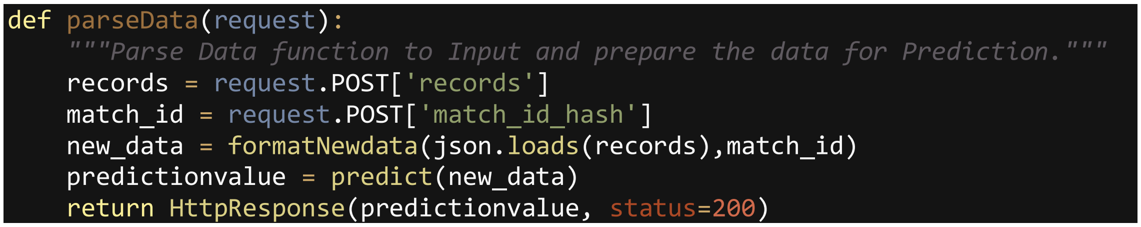
```python
from django.shortcuts import render
from django.contrib.auth import login, authenticate
from django.contrib.auth.decorators import login_required
from dota_analytics.forms import PasswordResetForm
from dota_analytics.forms import SignUpForm
# Create your views here.

@login_required(login_url="/login/")
def home(request):
    return render(request, 'dota_analytics/index.html')

def prediction(request):
    return render(request, 'dota_analytics/Queue.html')

def password_reset(request):
    if request.method == 'POST':
        form = PasswordResetForm(request.POST)
    else:
        form = PasswordResetForm()
    return render(request, 'registration/password_reset_form.html', {'form': form})
```

Figure : Decorator login_required in views.py

D. Functions as Parameters and Return Values

All the functions which have been implemented in the code whether in Python or in JavaScript is using the proper function definitions, Parameters and Return values.

An example of such function is shown from Python function *parseData()* where the model output is being predicted- fetchdata_service.py. The function itself is a collection of other functions which themselves follow the parametric and return concept.



```python
def parseData(request):
    """Parse Data function to Input and prepare the data for Prediction."""
    records = request.POST['records']
    match_id = request.POST['match_id_hash']
    new_data = formatNewdata(json.loads(records),match_id)
    predictionvalue = predict(new_data)
    return HttpResponse(predictionvalue, status=200)
```

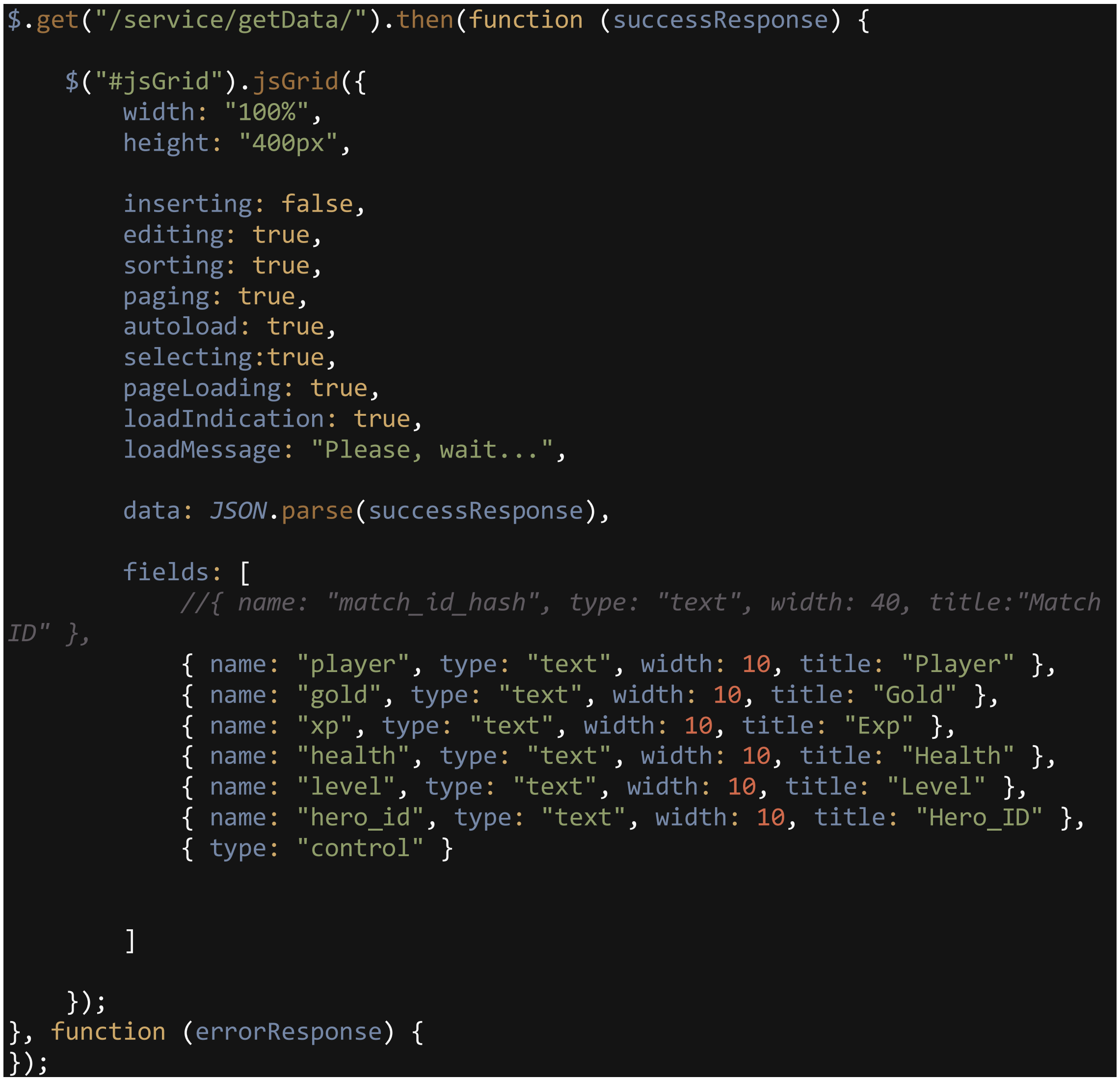
Figure : Function as Parameters and Return values

E. Anonymous Functions

The use of Anonymous function can be demonstrated in the JavaScript where the function is being invoked and implemented anonymously.

Here the API service which is used to get the data is being wrapped inside an anonymous function implementation to achieve a functionally composed pipeline.

This has been followed at across multiple levels as well, an example is within the same queue.js file where the entire snippet itself is an anonymous function call



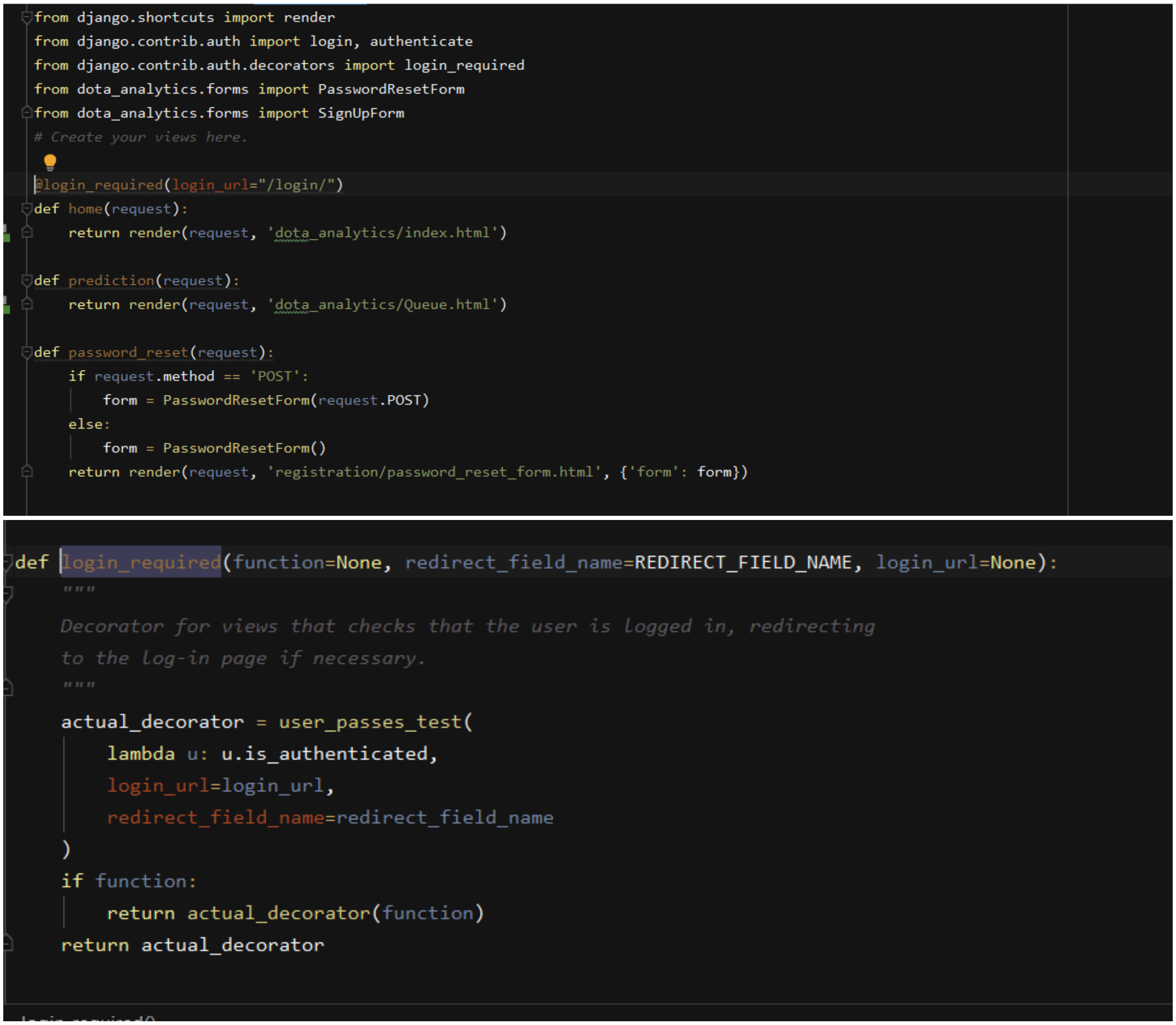
```
$.get("/service/getData/").then(function (successResponse) {

    $("#jsGrid").jsGrid({
        width: "100%",
        height: "400px",

        inserting: false,
        editing: true,
        sorting: true,
        paging: true,
        autoload: true,
        selecting:true,
        pageLoading: true,
        loadIndication: true,
        loadMessage: "Please, wait...",

        data: JSON.parse(successResponse),

        fields: [
            //{ name: "match_id_hash", type: "text", width: 40, title:"Match
ID" },
            { name: "player", type: "text", width: 10, title: "Player" },
            { name: "gold", type: "text", width: 10, title: "Gold" },
            { name: "xp", type: "text", width: 10, title: "Exp" },
            { name: "health", type: "text", width: 10, title: "Health" },
            { name: "level", type: "text", width: 10, title: "Level" },
            { name: "hero_id", type: "text", width: 10, title: "Hero_ID" },
            { type: "control" }


        ]

    });
}, function (errorResponse) {
});
```

Figure : queue.js anonymous function demonstration

11. AOP

Aspect Oriented Programming was applied in this project via using Decorators in Python. Decorators alter the functionality of a method dynamically without having to make subclass or change the source code of the decorated structure. This makes the code cleaner and helps in maintainability and reduce boilerplate code. These are high order functions which wrap the decorated functionality.

An example of the decorator used here is *@login_required* which is used before the user enters the web app. This helps in restricting access and security vulnerabilities hence providing an effective authentication mechanism



```
from django.shortcuts import render
from django.contrib.auth import login, authenticate
from django.contrib.auth.decorators import login_required
from dota_analytics.forms import PasswordResetForm
from dota_analytics.forms import SignUpForm
# Create your views here.

@login_required(login_url="/login/")
def home(request):
    return render(request, 'dota_analytics/index.html')

def prediction(request):
    return render(request, 'dota_analytics/Queue.html')

def password_reset(request):
    if request.method == 'POST':
        form = PasswordResetForm(request.POST)
    else:
        form = PasswordResetForm()
    return render(request, 'registration/password_reset_form.html', {'form': form})
```

```
def login_required(function=None, redirect_field_name=REDIRECT_FIELD_NAME, login_url=None):
    """
    Decorator for views that checks that the user is logged in, redirecting
    to the log-in page if necessary.
    """
    actual_decorator = user_passes_test(
        lambda u: u.is_authenticated,
        login_url=login_url,
        redirect_field_name=redirect_field_name
    )
    if function:
        return actual_decorator(function)
    return actual_decorator
```

Figure : Decorator Declaration and Definition