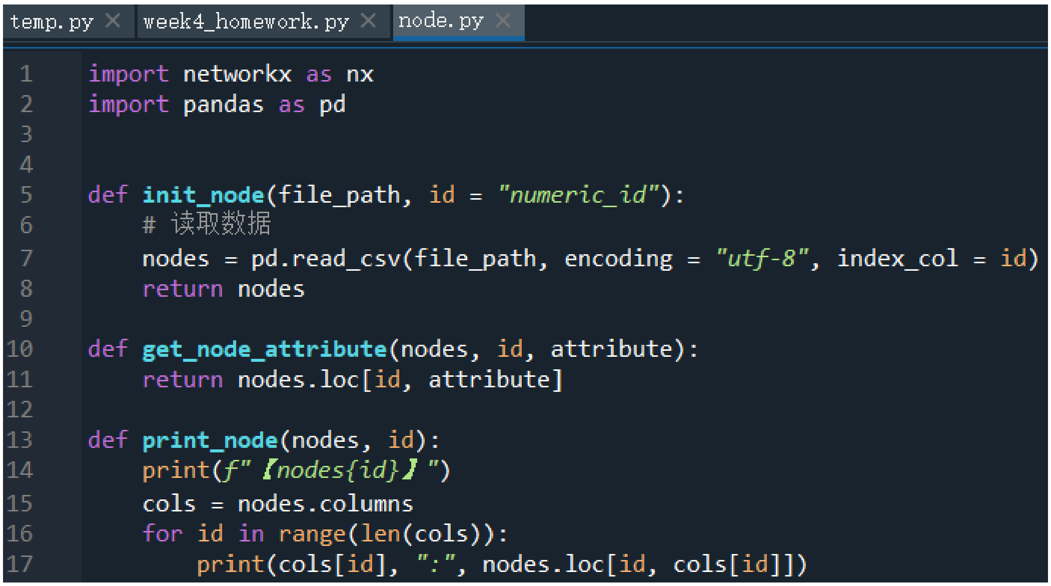
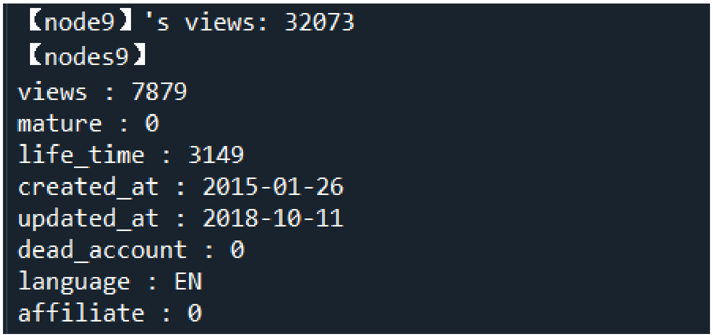
1.建立包 GraphStat

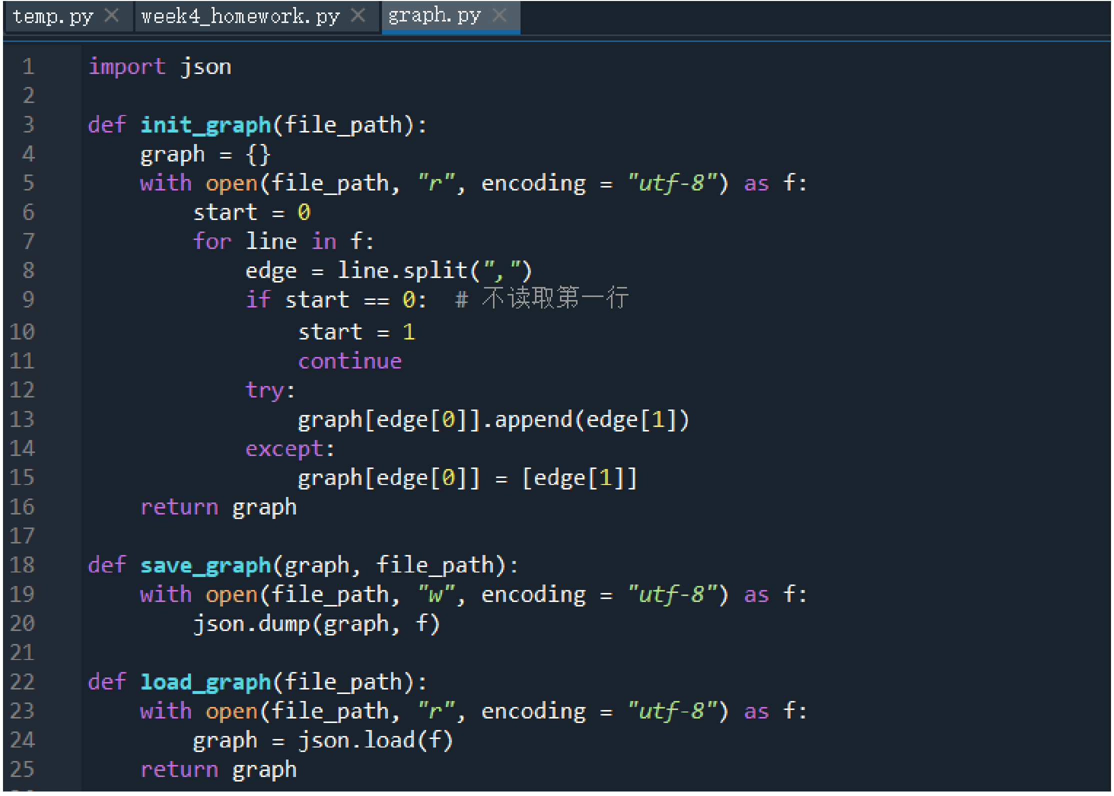
1.1 Graph



```python
import networkx as nx
import pandas as pd


def init_node(file_path, id = "numeric_id"):
    # 读取数据
    nodes = pd.read_csv(file_path, encoding = "utf-8", index_col = id)
    return nodes

def get_node_attribute(nodes, id, attribute):
    return nodes.loc[id, attribute]

def print_node(nodes, id):
    print(f"【nodes{id}】")
    cols = nodes.columns
    for id in range(len(cols)):
        print(cols[id], ":", nodes.loc[id, cols[id]])
```



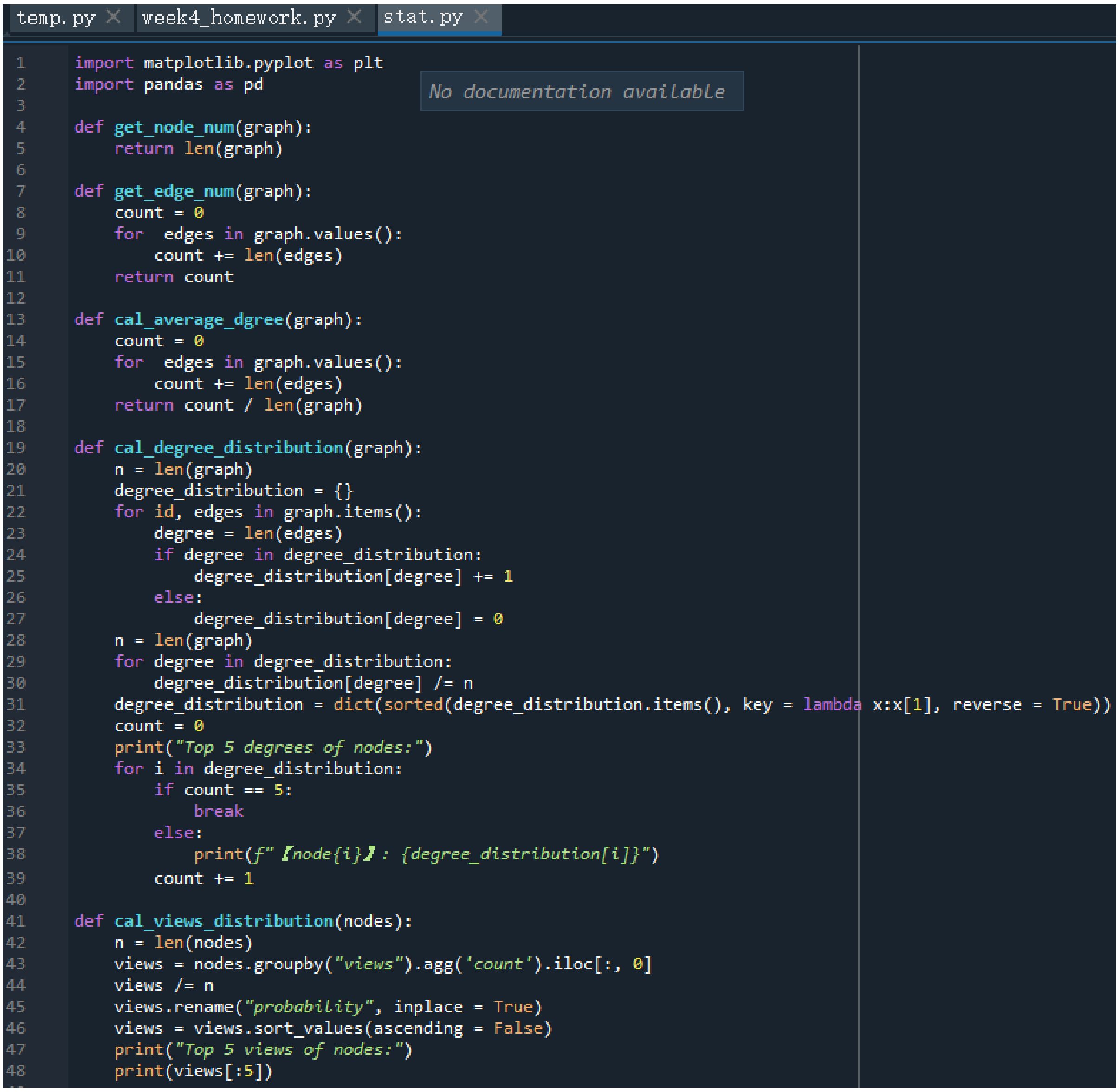
```
 【node9】's views: 32073
 【nodes9】
views : 7879
mature : 0
life_time : 3149
created_at : 2015-01-26
updated_at : 2018-10-11
dead_account : 0
language : EN
affiliate : 0
```



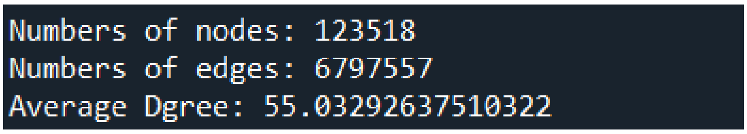
```python
import json

def init_graph(file_path):
    graph = {}
    with open(file_path, "r", encoding = "utf-8") as f:
        start = 0
        for line in f:
            edge = line.split(",")
            if start == 0:  # 不读取第一行
                start = 1
                continue
            try:
                graph[edge[0]].append(edge[1])
            except:
                graph[edge[0]] = [edge[1]]
    return graph

def save_graph(graph, file_path):
    with open(file_path, "w", encoding = "utf-8") as f:
        json.dump(graph, f)

def load_graph(file_path):
    with open(file_path, "r", encoding = "utf-8") as f:
        graph = json.load(f)
    return graph
```

代码解析：使用 DataFrame 储存节点属性，便于索引和透视；使用字典以邻接表数据结构储存网络图，便于查询，但与 DaraFrame 作 index 索引相比，没有 pandas 各种函数的灵活性和简洁性。

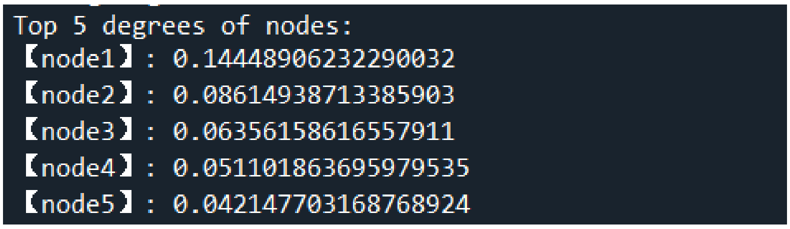
考虑到可以使程序除了应用于作业提供的数据外，还能应用于其他网络数据，在构建函数时尽量将 file_path，id，attribute 等设置为参数而非固定值，从而拓展了程序的可应用性。



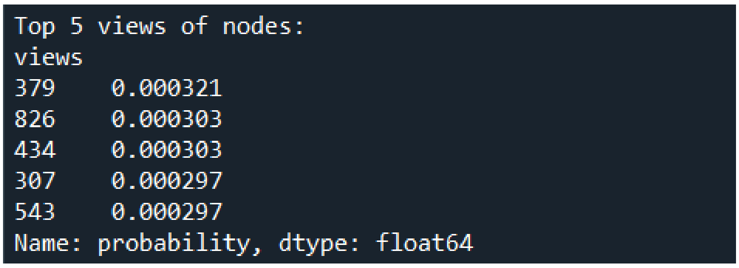
```python
import matplotlib.pyplot as plt
import pandas as pd

def get_node_num(graph):
    return len(graph)

def get_edge_num(graph):
    count = 0
    for  edges in graph.values():
        count += len(edges)
    return count

def cal_average_dgree(graph):
    count = 0
    for  edges in graph.values():
        count += len(edges)
    return count / len(graph)

def cal_degree_distribution(graph):
    n = len(graph)
    degree_distribution = {}
    for id, edges in graph.items():
        degree = len(edges)
        if degree in degree_distribution:
            degree_distribution[degree] += 1
        else:
            degree_distribution[degree] = 0
    n = len(graph)
    for degree in degree_distribution:
        degree_distribution[degree] /= n
    degree_distribution = dict(sorted(degree_distribution.items(), key = lambda x:x[1], reverse = True))
    count = 0
    print("Top 5 degrees of nodes:")
    for i in degree_distribution:
        if count == 5:
            break
        else:
            print(f"【node{i}】: {degree_distribution[i]}")
        count += 1

def cal_views_distribution(nodes):
    n = len(nodes)
    views = nodes.groupby("views").agg('count').iloc[:, 0]
    views /= n
    views.rename("probability", inplace = True)
    views = views.sort_values(ascending = False)
    print("Top 5 views of nodes:")
    print(views[:5])
```



```
Numbers of nodes: 123518
Numbers of edges: 6797557
Average Dgree: 55.03292637510322
```

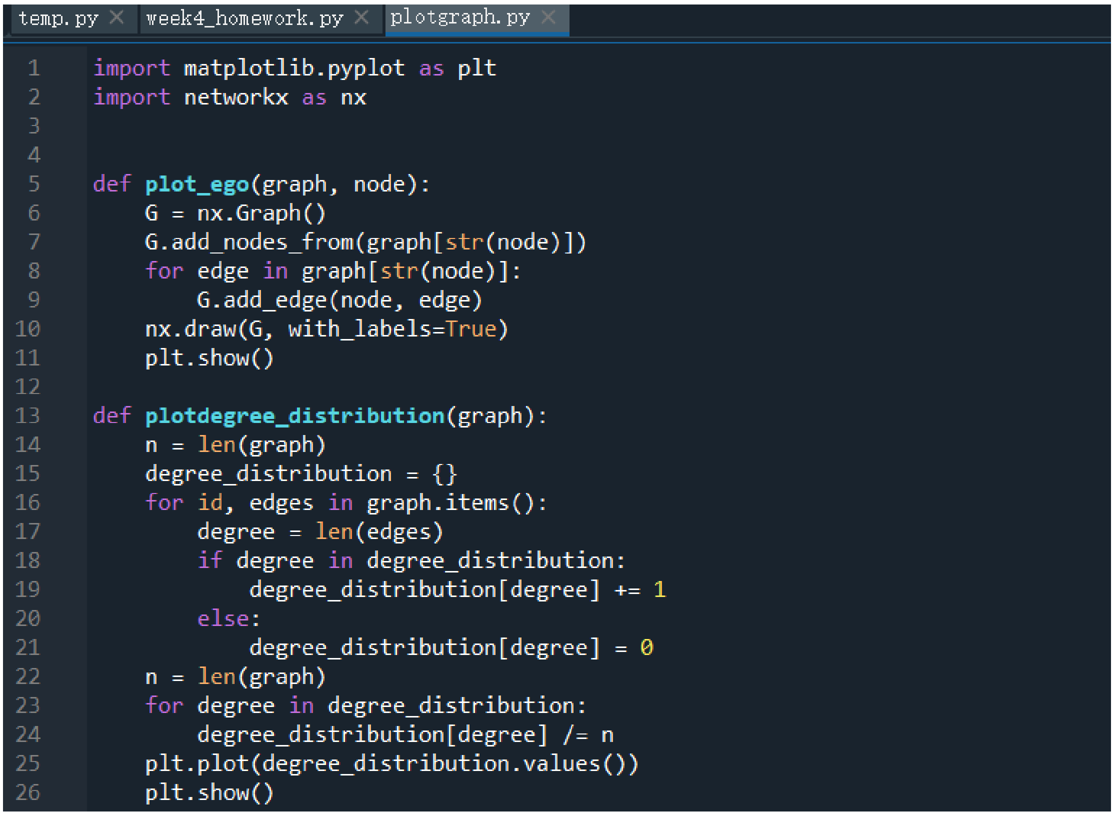


```
Top 5 degrees of nodes:
【node1】: 0.14448906232290032
【node2】: 0.08614938713385903
【node3】: 0.06356158616557911
【node4】: 0.051101863695979535
【node5】: 0.042147703168768924
```

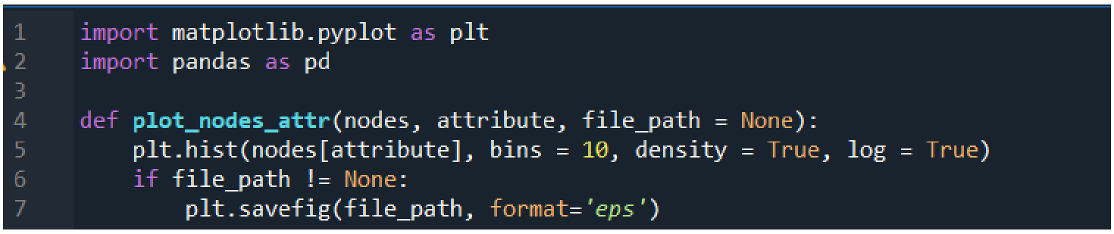


```
Top 5 views of nodes:
views
379    0.000321
826    0.000303
434    0.000303
307    0.000297
543    0.000297
Name: probability, dtype: float64
```

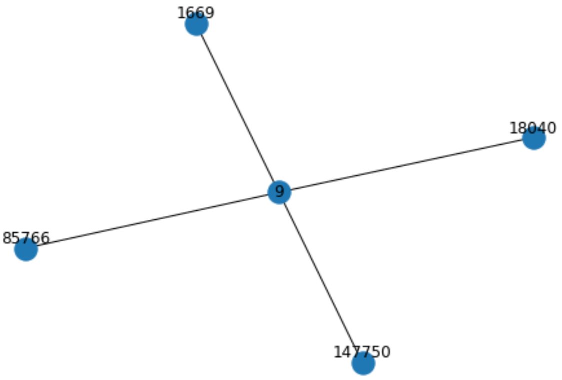
1.2 Visualization

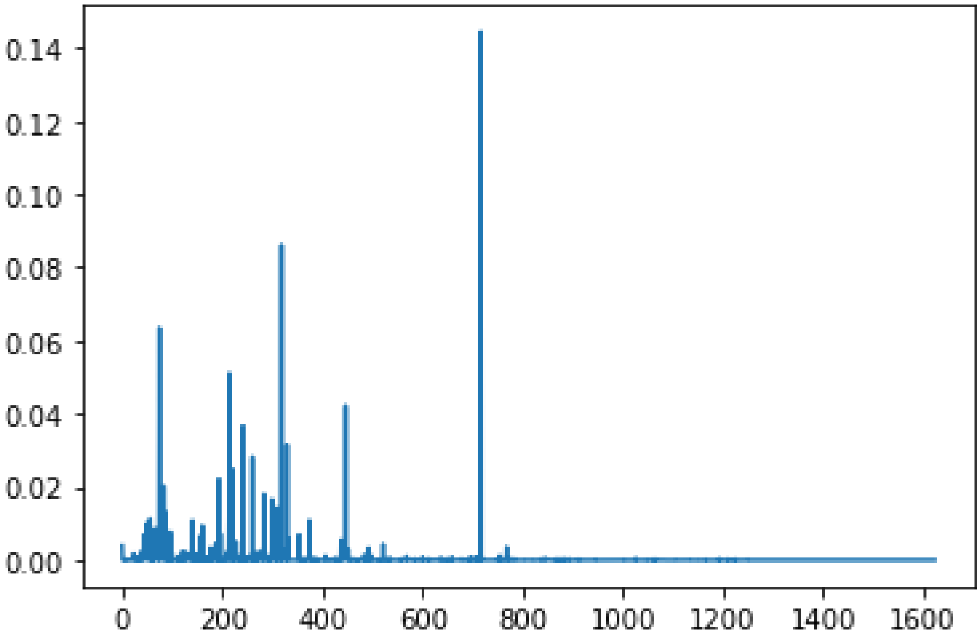


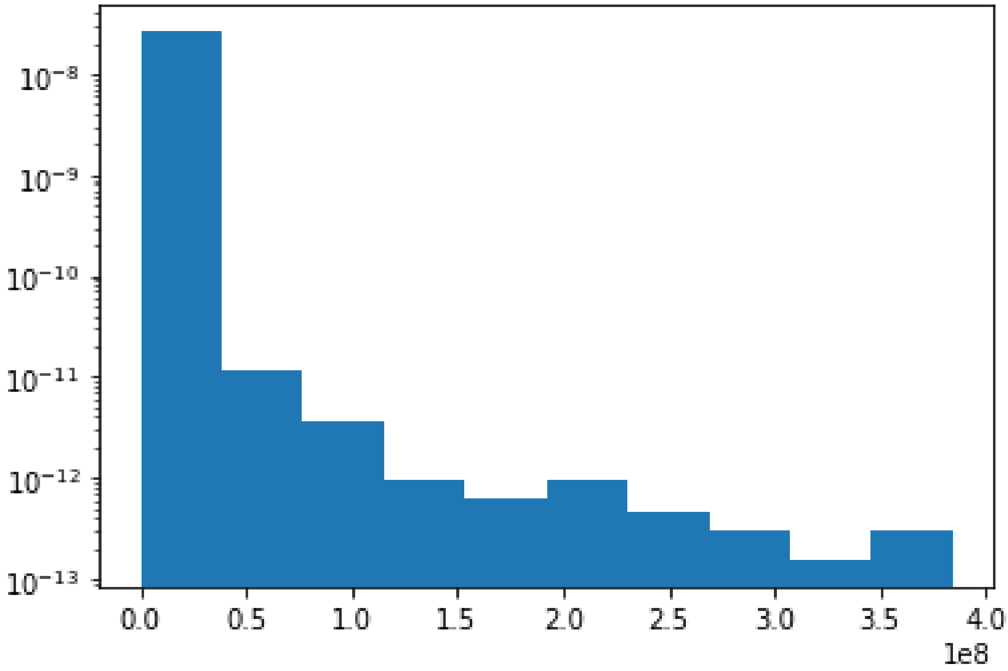
```python
import matplotlib.pyplot as plt
import networkx as nx


def plot_ego(graph, node):
    G = nx.Graph()
    G.add_nodes_from(graph[str(node)])
    for edge in graph[str(node)]:
        G.add_edge(node, edge)
    nx.draw(G, with_labels=True)
    plt.show()

def plotdegree_distribution(graph):
    n = len(graph)
    degree_distribution = {}
    for id, edges in graph.items():
        degree = len(edges)
        if degree in degree_distribution:
            degree_distribution[degree] += 1
        else:
            degree_distribution[degree] = 0
    n = len(graph)
    for degree in degree_distribution:
        degree_distribution[degree] /= n
    plt.plot(degree_distribution.values())
    plt.show()
```

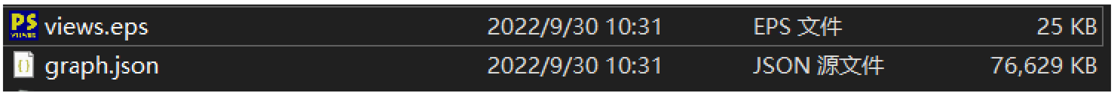


```python
import matplotlib.pyplot as plt
import pandas as pd

def plot_nodes_attr(nodes, attribute, file_path = None):
    plt.hist(nodes[attribute], bins = 10, density = True, log = True)
    if file_path != None:
        plt.savefig(file_path, format='eps')
```



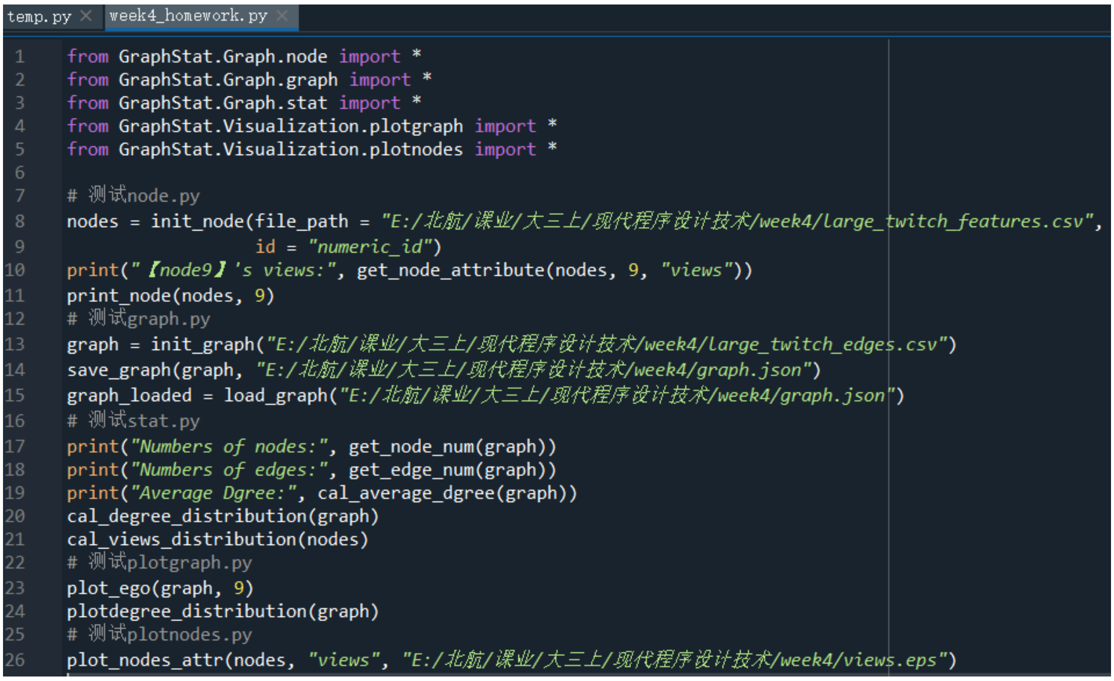






views.eps	2022/9/30 10:31	EPS 文件	25 KB
graph.json	2022/9/30 10:31	JSON 源文件	76,629 KB

2. 测试包



```python
from GraphStat.Graph.node import *
from GraphStat.Graph.graph import *
from GraphStat.Graph.stat import *
from GraphStat.Visualization.plotgraph import *
from GraphStat.Visualization.plotnodes import *

# 测试node.py
nodes = init_node(file_path = "E:/北航/课业/大三上/现代程序设计技术/week4/large_twitch_features.csv",
                  id = "numeric_id")
print("【node9】's views:", get_node_attribute(nodes, 9, "views"))
print_node(nodes, 9)
# 测试graph.py
graph = init_graph("E:/北航/课业/大三上/现代程序设计技术/week4/large_twitch_edges.csv")
save_graph(graph, "E:/北航/课业/大三上/现代程序设计技术/week4/graph.json")
graph_loaded = load_graph("E:/北航/课业/大三上/现代程序设计技术/week4/graph.json")
# 测试stat.py
print("Numbers of nodes:", get_node_num(graph))
print("Numbers of edges:", get_edge_num(graph))
print("Average Dgree:", cal_average_dgree(graph))
cal_degree_distribution(graph)
cal_views_distribution(nodes)
# 测试plotgraph.py
plot_ego(graph, 9)
plotdegree_distribution(graph)
# 测试plotnodes.py
plot_nodes_attr(nodes, "views", "E:/北航/课业/大三上/现代程序设计技术/week4/views.eps")
```

3. 数据分析

可以看到，网络度的分布和节点属性“views”的分布都呈左偏形态，平均度为 55，度值主要分布在[50,450]，其中 730 左右的度值异常增高，而周围度值并无增高趋势，很可能是数据有误，有待进一步排查；“views”主要分布在$[0, 0.4 \times 10^8]$，峰值左偏形态明显。

对市场营销者来说，若想推广一款新产品，若经费充足，可以面向度值在[50,450]的节点，若经费有限，可面向度值在[200,270]的节点。