AWS Global Power Rankings

S3 - Athena - SageMaker – Lambda – API Gateway

Introduction

- S3: Storage services
- Athena: Querying tables in S3 bucket
- SageMaker: Analyzing data retrieved from Athena queries
- Lambda: Migrating code and get ready to deploy
- API Gateway: Deploy lambda function to the endpoint

Endpoints and Ranking Algorithms

- Three required endpoints:

- /global-rankings (?number_of_teams)
- /tournament-rankings (?tournament_id)
- /team-rankings (?team_ids)

- Global rankings use regional strength and features interactive weights
- Tournament ranking
- Team ranking uses win/loss ratios of teams internationally and domestically to create regional strength metrics and reliability factors

Global Rankings

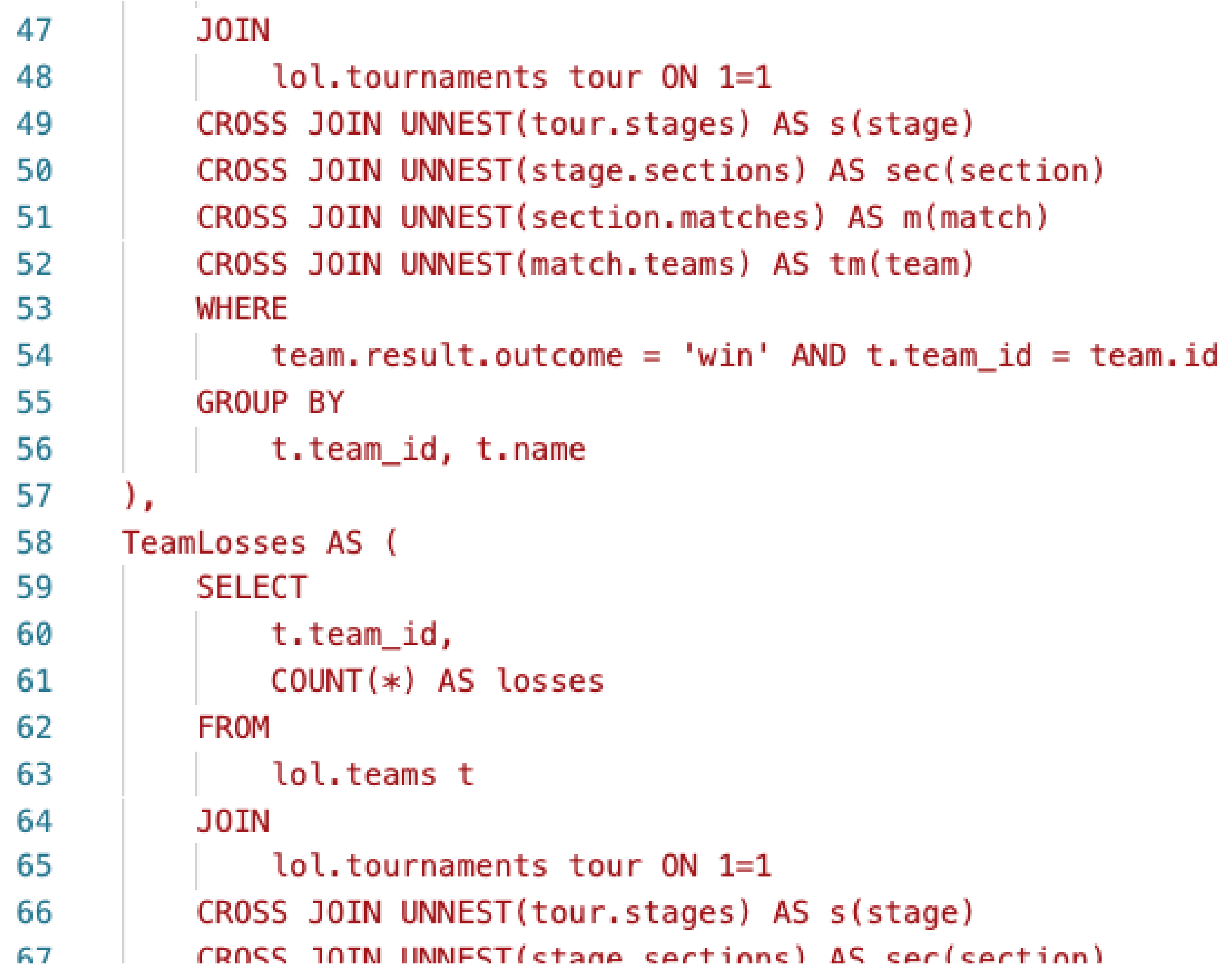
Data Retrieval

SQL query to retrieve team stats across all regions in the lol.tournaments table.

Includes team_id, number of wins, number of losses, and region.

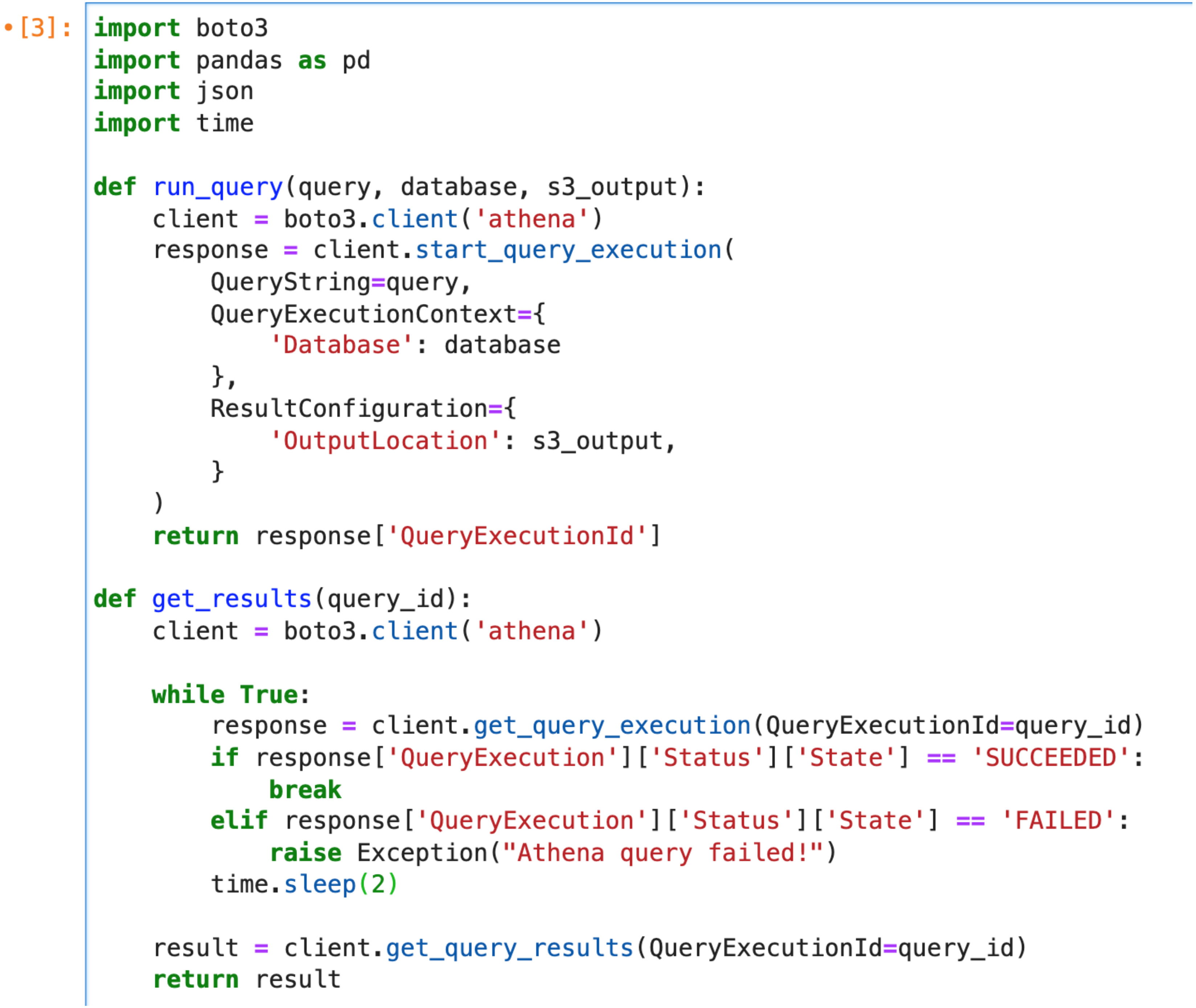
THE CROSS JOIN UNNEST

When an object has an attribute that contains a tuple and you want to include it in the data of the object itself, you can **cross join unnest**



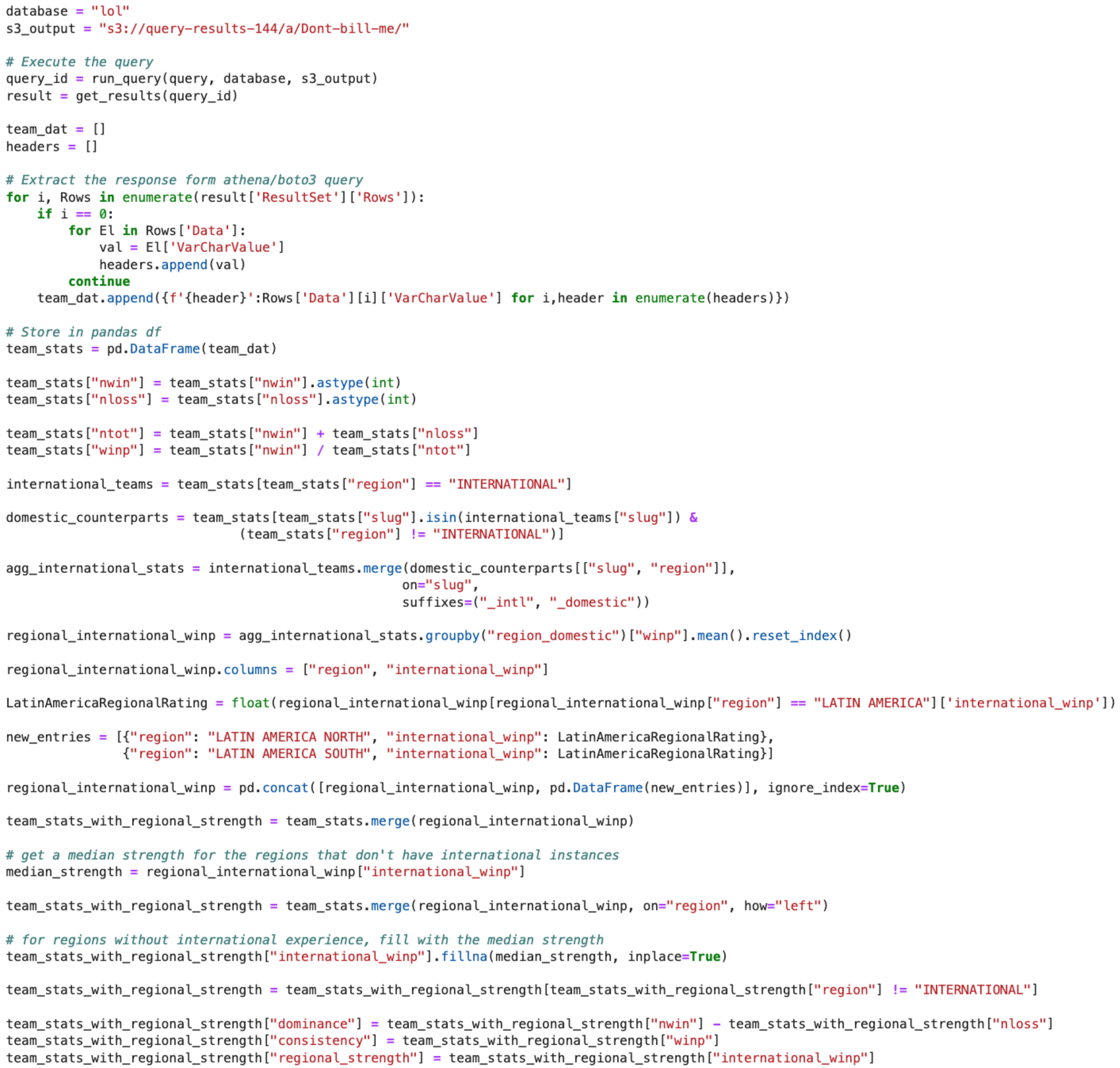
```
    JOIN
        lol.tournaments tour ON 1=1
    CROSS JOIN UNNEST(tour.stages) AS s(stage)
    CROSS JOIN UNNEST(stage.sections) AS sec(section)
    CROSS JOIN UNNEST(section.matches) AS m(match)
    CROSS JOIN UNNEST(match.teams) AS tm(team)
    WHERE
        team.result.outcome = 'win' AND t.team_id = team.id
    GROUP BY
        t.team_id, t.name
),
TeamLosses AS (
    SELECT
        t.team_id,
        COUNT(*) AS losses
    FROM
        lol.teams t
    JOIN
        lol.tournaments tour ON 1=1
    CROSS JOIN UNNEST(tour.stages) AS s(stage)
    CROSS JOIN UNNEST(stage.sections) AS sec(section)
```

Running the queries from Python using boto3



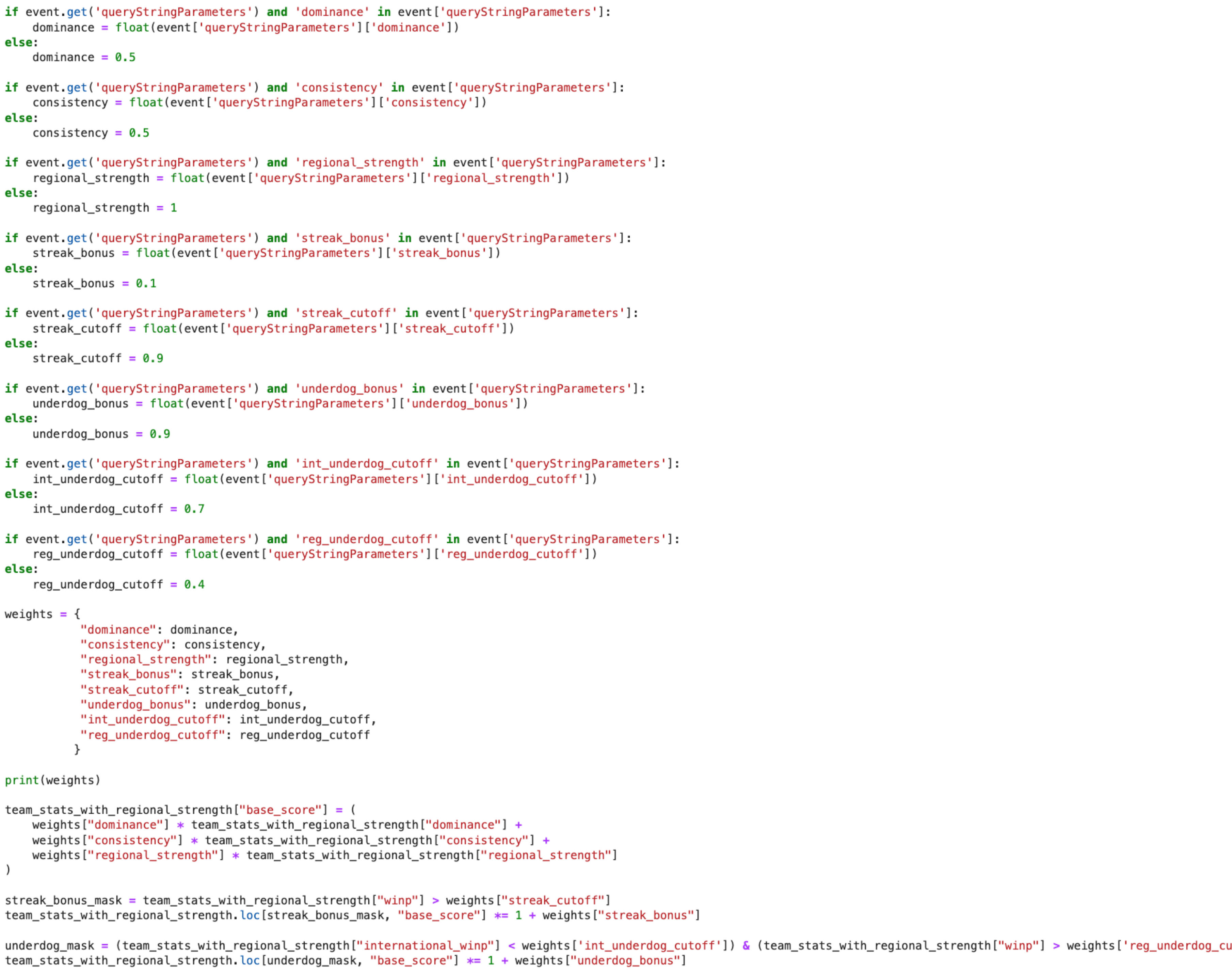
```
import boto3
import pandas as pd
import json
import time

def run_query(query, database, s3_output):
    client = boto3.client('athena')
    response = client.start_query_execution(
        QueryString=query,
        QueryExecutionContext={
            'Database': database
        },
        ResultConfiguration={
            'OutputLocation': s3_output,
        }
    )
    return response['QueryExecutionId']

def get_results(query_id):
    client = boto3.client('athena')

    while True:
        response = client.get_query_execution(QueryExecutionId=query_id)
        if response['QueryExecution']['Status']['State'] == 'SUCCEEDED':
            break
        elif response['QueryExecution']['Status']['State'] == 'FAILED':
            raise Exception("Athena query failed!")
        time.sleep(2)

    result = client.get_query_results(QueryExecutionId=query_id)
    return result
```

Extracting the Athena response to Pandas and defining basic metrics



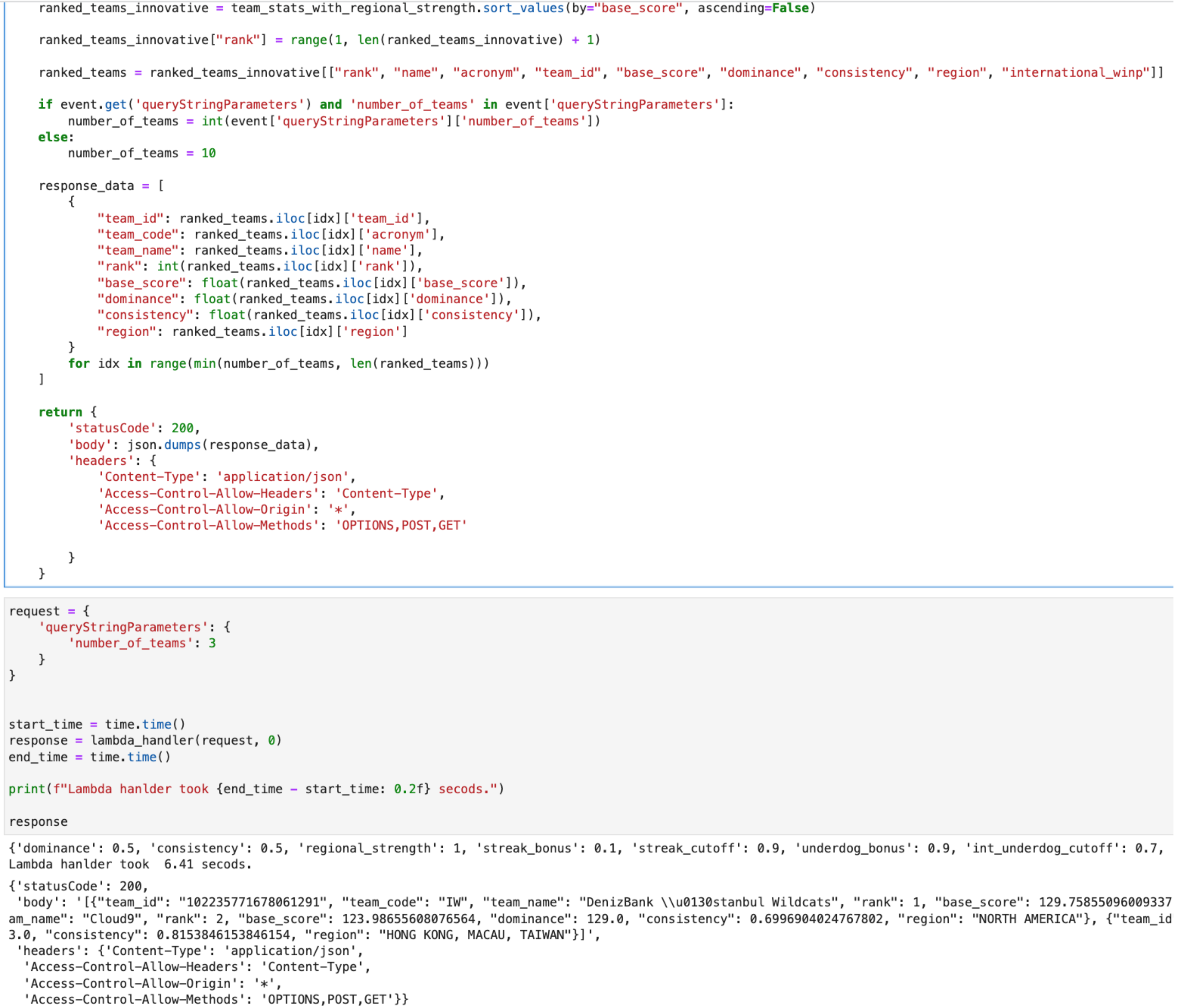
```
database = "lol"
s3_output = "s3://query-results-144/a/Dont-bill-me/"

# Execute the query
query_id = run_query(query, database, s3_output)
result = get_results(query_id)

team_dat = []
headers = []

# Extract the response form athena/boto3 query
for i, Rows in enumerate(result['ResultSet']['Rows']):
    if i == 0:
        for El in Rows['Data']:
            val = El['VarCharValue']
            headers.append(val)
        continue
    team_dat.append({f'{header}':Rows['Data'][i]['VarCharValue'] for i,header in enumerate(headers)})

# Store in pandas df
team_stats = pd.DataFrame(team_dat)

team_stats["nwin"] = team_stats["nwin"].astype(int)
team_stats["nloss"] = team_stats["nloss"].astype(int)

team_stats["ntot"] = team_stats["nwin"] + team_stats["nloss"]
team_stats["winp"] = team_stats["nwin"] / team_stats["ntot"]

international_teams = team_stats[team_stats["region"] == "INTERNATIONAL"]

domestic_counterparts = team_stats[team_stats["slug"].isin(international_teams["slug"]) &
                              (team_stats["region"] != "INTERNATIONAL")]

agg_international_stats = international_teams.merge(domestic_counterparts[["slug", "region"]],
                                                     on="slug",
                                                     suffixes=("_intl", "_domestic"))

regional_international_winp = agg_international_stats.groupby("region_domestic")["winp"].mean().reset_index()

regional_international_winp.columns = ["region", "international_winp"]

LatinAmericaRegionalRating = float(regional_international_winp[regional_international_winp["region"] == "LATIN AMERICA"]['international_winp'])

new_entries = [{"region": "LATIN AMERICA NORTH", "international_winp": LatinAmericaRegionalRating},
               {"region": "LATIN AMERICA SOUTH", "international_winp": LatinAmericaRegionalRating}]

regional_international_winp = pd.concat([regional_international_winp, pd.DataFrame(new_entries)], ignore_index=True)

team_stats_with_regional_strength = team_stats.merge(regional_international_winp)

# get a median strength for the regions that don't have international instances
median_strength = regional_international_winp["international_winp"]

team_stats_with_regional_strength = team_stats.merge(regional_international_winp, on="region", how="left")

# for regions without international experience, fill with the median strength
team_stats_with_regional_strength["international_winp"].fillna(median_strength, inplace=True)

team_stats_with_regional_strength = team_stats_with_regional_strength[team_stats_with_regional_strength["region"] != "INTERNATIONAL"]

team_stats_with_regional_strength["dominance"] = team_stats_with_regional_strength["nwin"] - team_stats_with_regional_strength["nloss"]
team_stats_with_regional_strength["consistency"] = team_stats_with_regional_strength["winp"]
team_stats_with_regional_strength["regional_strength"] = team_stats_with_regional_strength["international_winp"]
```

Get input parameters from the endpoints and compute base score using weights



```
if event.get('queryStringParameters') and 'dominance' in event['queryStringParameters']:
    dominance = float(event['queryStringParameters']['dominance'])
else:
    dominance = 0.5

if event.get('queryStringParameters') and 'consistency' in event['queryStringParameters']:
    consistency = float(event['queryStringParameters']['consistency'])
else:
    consistency = 0.5

if event.get('queryStringParameters') and 'regional_strength' in event['queryStringParameters']:
    regional_strength = float(event['queryStringParameters']['regional_strength'])
else:
    regional_strength = 1

if event.get('queryStringParameters') and 'streak_bonus' in event['queryStringParameters']:
    streak_bonus = float(event['queryStringParameters']['streak_bonus'])
else:
    streak_bonus = 0.1

if event.get('queryStringParameters') and 'streak_cutoff' in event['queryStringParameters']:
    streak_cutoff = float(event['queryStringParameters']['streak_cutoff'])
else:
    streak_cutoff = 0.9

if event.get('queryStringParameters') and 'underdog_bonus' in event['queryStringParameters']:
    underdog_bonus = float(event['queryStringParameters']['underdog_bonus'])
else:
    underdog_bonus = 0.9

if event.get('queryStringParameters') and 'int_underdog_cutoff' in event['queryStringParameters']:
    int_underdog_cutoff = float(event['queryStringParameters']['int_underdog_cutoff'])
else:
    int_underdog_cutoff = 0.7

if event.get('queryStringParameters') and 'reg_underdog_cutoff' in event['queryStringParameters']:
    reg_underdog_cutoff = float(event['queryStringParameters']['reg_underdog_cutoff'])
else:
    reg_underdog_cutoff = 0.4

weights = {
           "dominance": dominance,
           "consistency": consistency,
           "regional_strength": regional_strength,
           "streak_bonus": streak_bonus,
           "streak_cutoff": streak_cutoff,
           "underdog_bonus": underdog_bonus,
           "int_underdog_cutoff": int_underdog_cutoff,
           "reg_underdog_cutoff": reg_underdog_cutoff
          }

print(weights)

team_stats_with_regional_strength["base_score"] = (
    weights["dominance"] * team_stats_with_regional_strength["dominance"] +
    weights["consistency"] * team_stats_with_regional_strength["consistency"] +
    weights["regional_strength"] * team_stats_with_regional_strength["regional_strength"]
)

streak_bonus_mask = team_stats_with_regional_strength["winp"] > weights["streak_cutoff"]
team_stats_with_regional_strength.loc[streak_bonus_mask, "base_score"] *= 1 + weights["streak_bonus"]

underdog_mask = (team_stats_with_regional_strength["international_winp"] < weights['int_underdog_cutoff']) & (team_stats_with_regional_strength["winp"] > weights['reg_underdog_cu
team_stats_with_regional_strength.loc[underdog_mask, "base_score"] *= 1 + weights["underdog_bonus"]
```

Basic testing



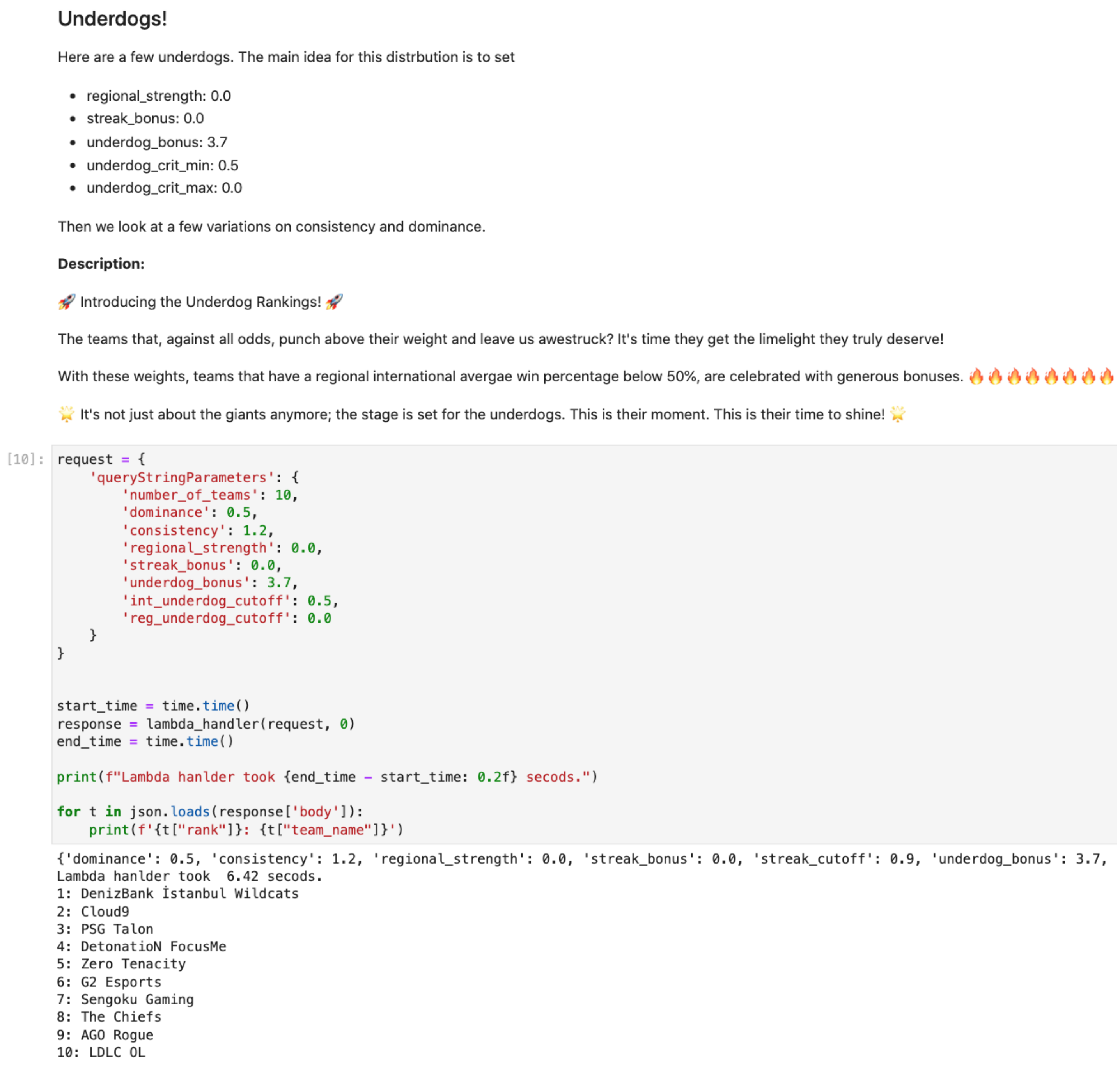
```
    ranked_teams_innovative = team_stats_with_regional_strength.sort_values(by="base_score", ascending=False)

    ranked_teams_innovative["rank"] = range(1, len(ranked_teams_innovative) + 1)

    ranked_teams = ranked_teams_innovative[["rank", "name", "acronym", "team_id", "base_score", "dominance", "consistency", "region", "international_winp"]]

    if event.get('queryStringParameters') and 'number_of_teams' in event['queryStringParameters']:
        number_of_teams = int(event['queryStringParameters']['number_of_teams'])
    else:
        number_of_teams = 10

    response_data = [
        {
            "team_id": ranked_teams.iloc[idx]['team_id'],
            "team_code": ranked_teams.iloc[idx]['acronym'],
            "team_name": ranked_teams.iloc[idx]['name'],
            "rank": int(ranked_teams.iloc[idx]['rank']),
            "base_score": float(ranked_teams.iloc[idx]['base_score']),
            "dominance": float(ranked_teams.iloc[idx]['dominance']),
            "consistency": float(ranked_teams.iloc[idx]['consistency']),
            "region": ranked_teams.iloc[idx]['region']
        }
        for idx in range(min(number_of_teams, len(ranked_teams)))
    ]

    return {
        'statusCode': 200,
        'body': json.dumps(response_data),
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'

        }
    }
```

```
request = {
    'queryStringParameters': {
        'number_of_teams': 3
    }
}


start_time = time.time()
response = lambda_handler(request, 0)
end_time = time.time()

print(f"Lambda hanlder took {end_time - start_time: 0.2f} secods.")

response
```

```
{'dominance': 0.5, 'consistency': 0.5, 'regional_strength': 1, 'streak_bonus': 0.1, 'streak_cutoff': 0.9, 'underdog_bonus': 0.9, 'int_underdog_cutoff': 0.7,
Lambda hanlder took  6.41 secods.
```

```
{'statusCode': 200,
 'body': '[{"team_id": "102235771678061291", "team_code": "IW", "team_name": "DenizBank \\u0130stanbul Wildcats", "rank": 1, "base_score": 129.75855096009337
am_name": "Cloud9", "rank": 2, "base_score": 123.98655608076564, "dominance": 129.0, "consistency": 0.6996904024767802, "region": "NORTH AMERICA"}, {"team_id
3.0, "consistency": 0.8153846153846154, "region": "HONG KONG, MACAU, TAIWAN"}]',
 'headers': {'Content-Type': 'application/json',
  'Access-Control-Allow-Headers': 'Content-Type',
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'}}
```

Testing with input weights.



Underdogs!

Here are a few underdogs. The main idea for this distrbution is to set

- regional_strength: 0.0
- streak_bonus: 0.0
- underdog_bonus: 3.7
- underdog_crit_min: 0.5
- underdog_crit_max: 0.0

Then we look at a few variations on consistency and dominance.

Description:

🚀 Introducing the Underdog Rankings! 🚀

The teams that, against all odds, punch above their weight and leave us awestruck? It's time they get the limelight they truly deserve!

With these weights, teams that have a regional international avergae win percentage below 50%, are celebrated with generous bonuses. 🔥🔥🔥🔥🔥🔥🔥🔥

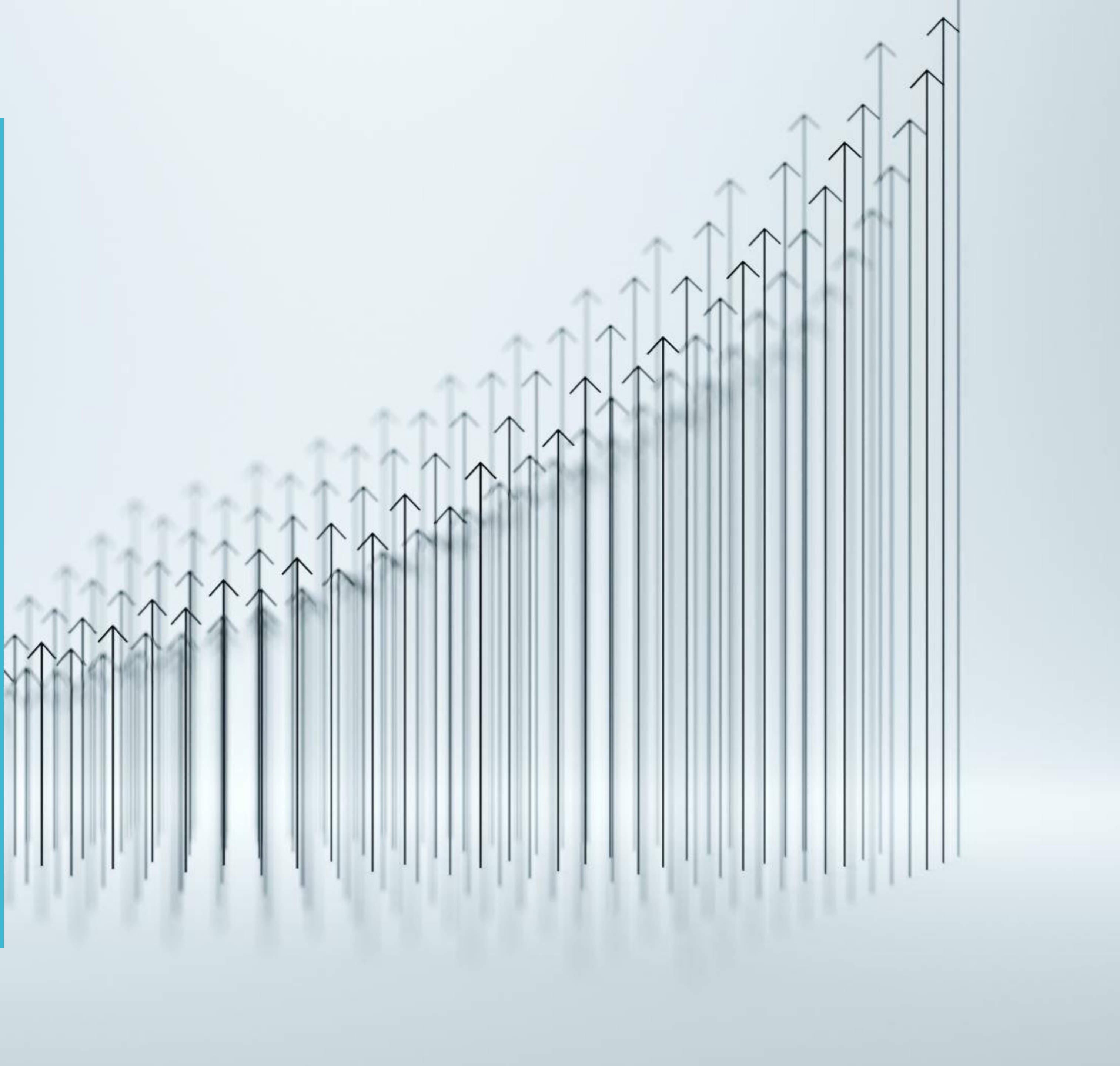
🌟 It's not just about the giants anymore; the stage is set for the underdogs. This is their moment. This is their time to shine! 🌟

[10]:

```python
request = {
    'queryStringParameters': {
        'number_of_teams': 10,
        'dominance': 0.5,
        'consistency': 1.2,
        'regional_strength': 0.0,
        'streak_bonus': 0.0,
        'underdog_bonus': 3.7,
        'int_underdog_cutoff': 0.5,
        'reg_underdog_cutoff': 0.0
    }
}


start_time = time.time()
response = lambda_handler(request, 0)
end_time = time.time()

print(f"Lambda hanlder took {end_time - start_time: 0.2f} secods.")

for t in json.loads(response['body']):
    print(f'{t["rank"]}: {t["team_name"]}')
```

```
{'dominance': 0.5, 'consistency': 1.2, 'regional_strength': 0.0, 'streak_bonus': 0.0, 'streak_cutoff': 0.9, 'underdog_bonus': 3.7,
Lambda hanlder took  6.42 secods.
1: DenizBank İstanbul Wildcats
2: Cloud9
3: PSG Talon
4: DetonatioN FocusMe
5: Zero Tenacity
6: G2 Esports
7: Sengoku Gaming
8: The Chiefs
9: AGO Rogue
10: LDLC OL
```

Tournament Rankings



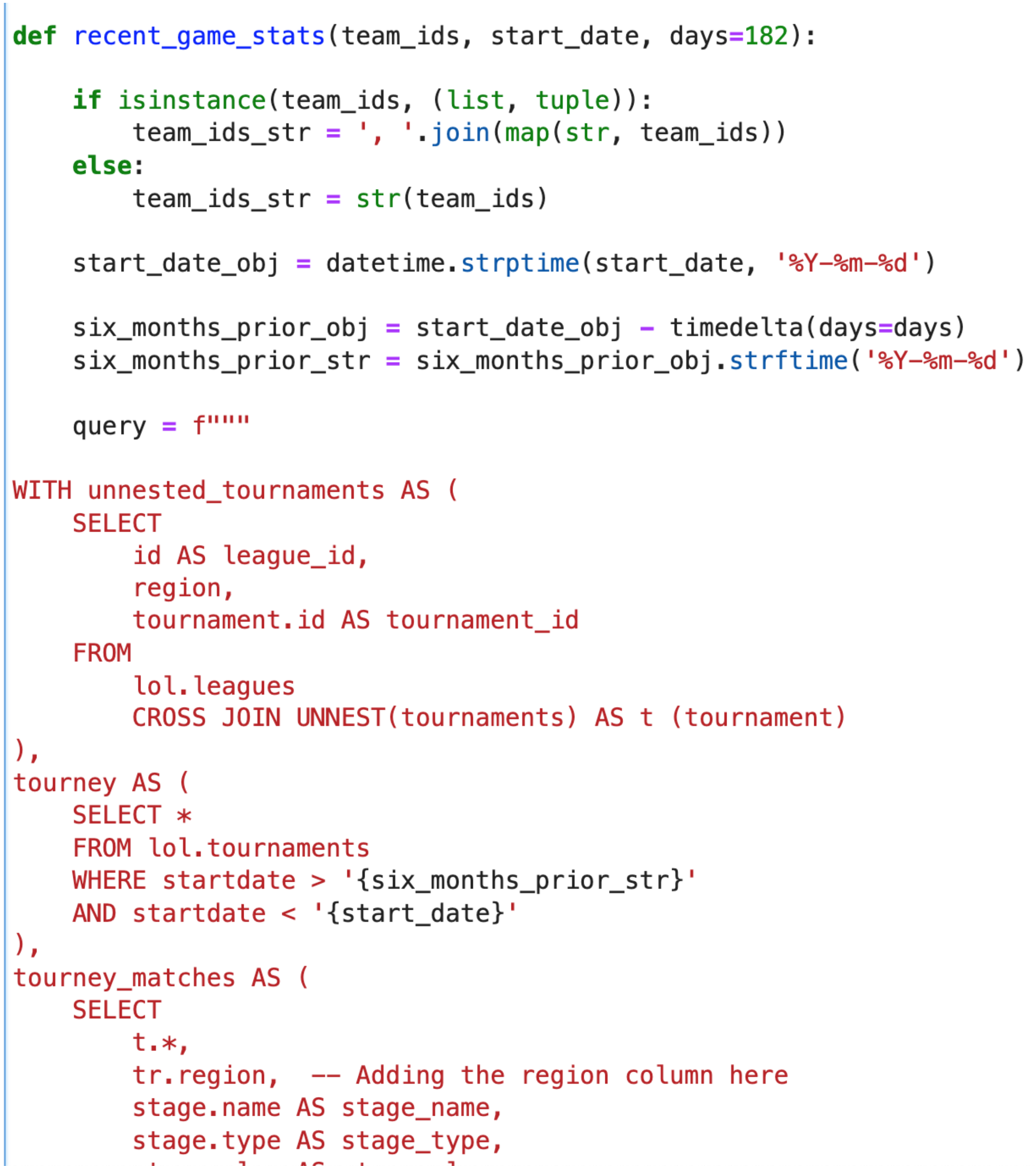
Data Retrieval

League tournaments and team performance metrics.

Core Functions:

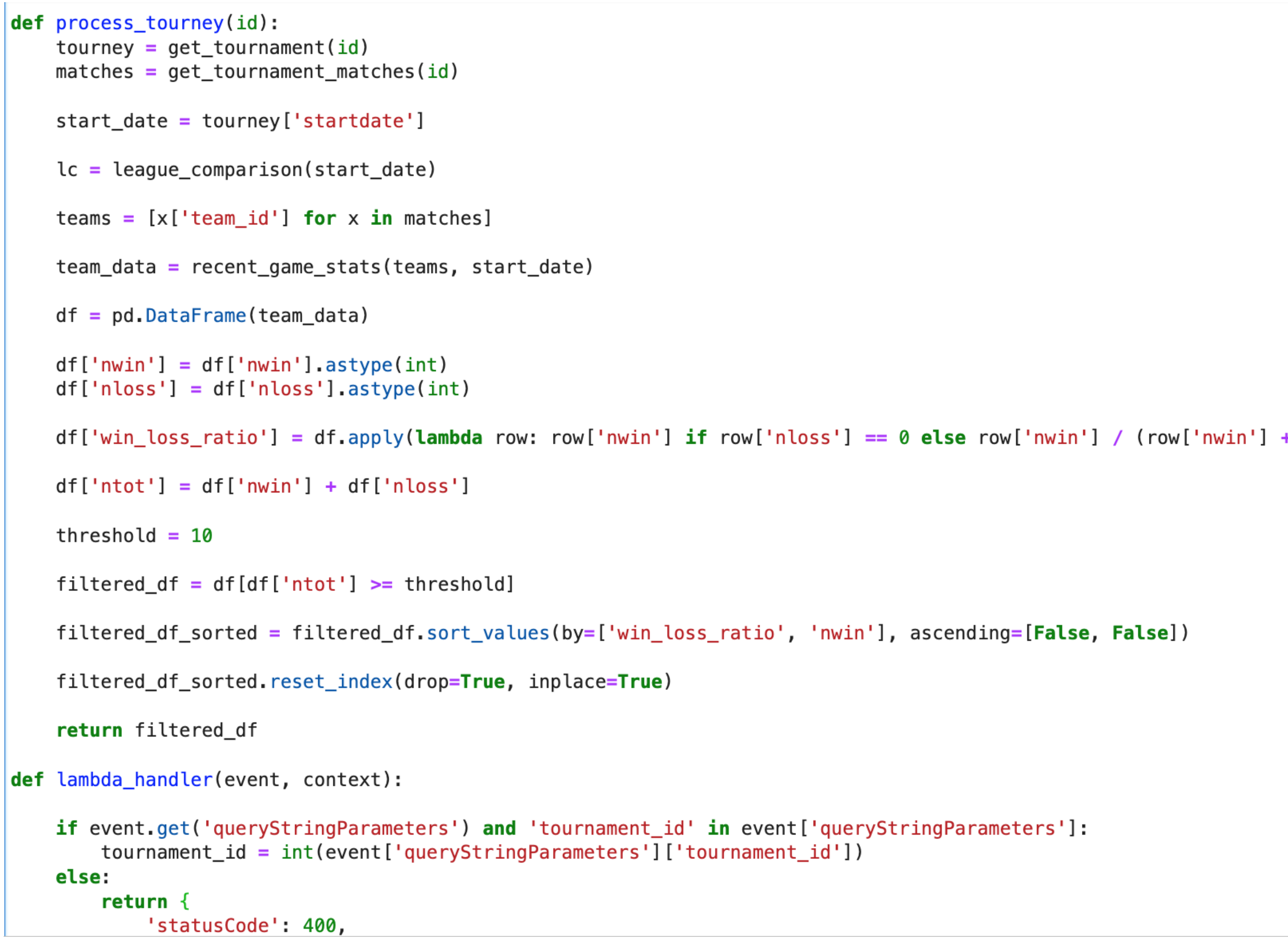
- **get_tournament**: Retrieves basic information about a specific tournament using its ID.
- **get_tournament_matches**: Retrieves match details (teams and their performance) for a specific tournament.
- **recent_game_stats**: Retrieves statistics for a set of teams from games played during the six months leading up
- **league_comparison**: Compares the number of games played by region during the six months leading up

Recent game stats function for finding games within six months



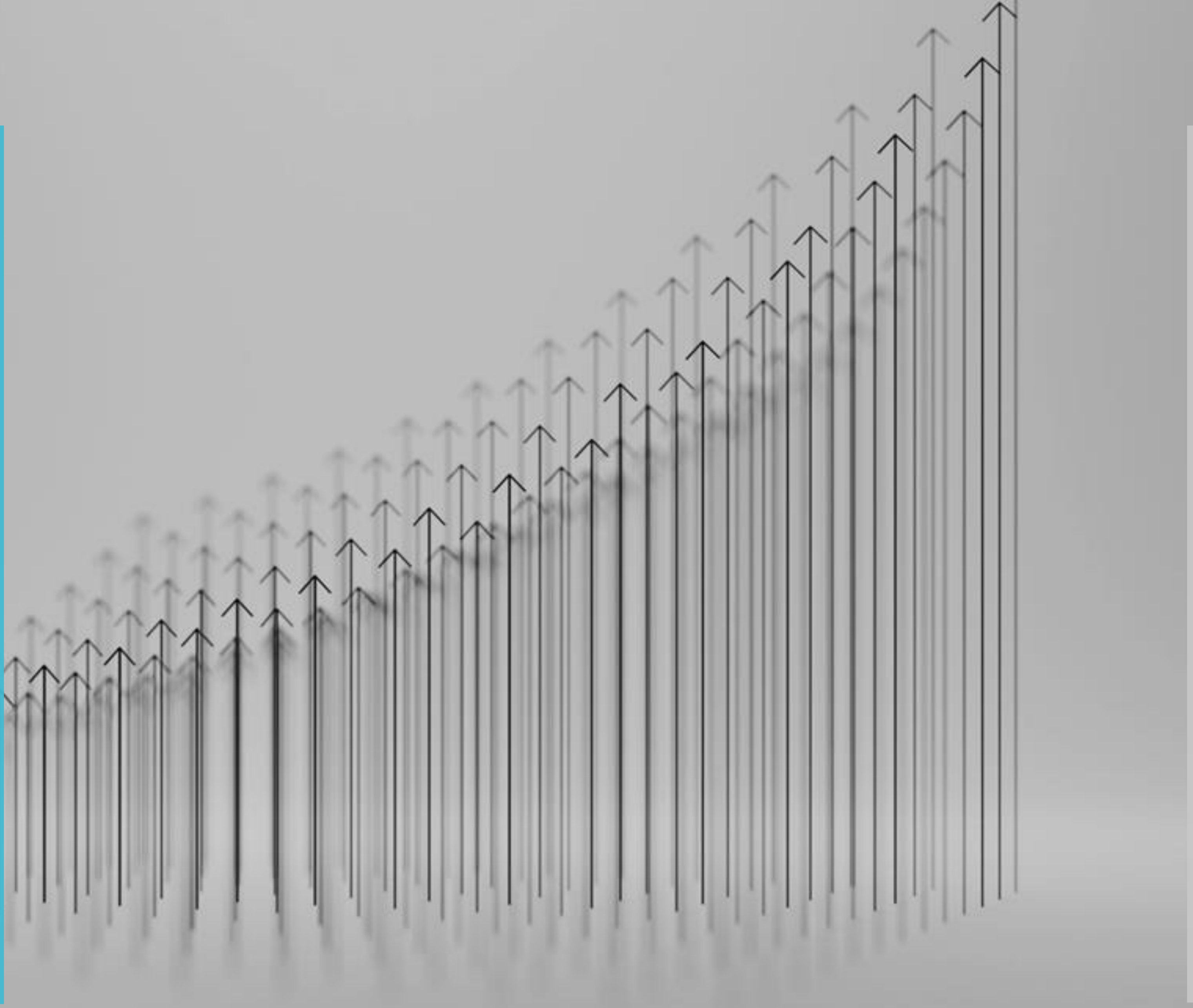
```
def recent_game_stats(team_ids, start_date, days=182):

    if isinstance(team_ids, (list, tuple)):
        team_ids_str = ', '.join(map(str, team_ids))
    else:
        team_ids_str = str(team_ids)

    start_date_obj = datetime.strptime(start_date, '%Y-%m-%d')

    six_months_prior_obj = start_date_obj - timedelta(days=days)
    six_months_prior_str = six_months_prior_obj.strftime('%Y-%m-%d')

    query = f"""

WITH unnested_tournaments AS (
    SELECT
        id AS league_id,
        region,
        tournament.id AS tournament_id
    FROM
        lol.leagues
        CROSS JOIN UNNEST(tournaments) AS t (tournament)
),
tourney AS (
    SELECT *
    FROM lol.tournaments
    WHERE startdate > '{six_months_prior_str}'
    AND startdate < '{start_date}'
),
tourney_matches AS (
    SELECT
        t.*,
        tr.region,  -- Adding the region column here
        stage.name AS stage_name,
        stage.type AS stage_type,
```

Tournament ranking function that holds most of the logic



```python
def process_tourney(id):
    tourney = get_tournament(id)
    matches = get_tournament_matches(id)

    start_date = tourney['startdate']

    lc = league_comparison(start_date)

    teams = [x['team_id'] for x in matches]

    team_data = recent_game_stats(teams, start_date)

    df = pd.DataFrame(team_data)

    df['nwin'] = df['nwin'].astype(int)
    df['nloss'] = df['nloss'].astype(int)

    df['win_loss_ratio'] = df.apply(lambda row: row['nwin'] if row['nloss'] == 0 else row['nwin'] / (row['nwin'] +

    df['ntot'] = df['nwin'] + df['nloss']

    threshold = 10

    filtered_df = df[df['ntot'] >= threshold]

    filtered_df_sorted = filtered_df.sort_values(by=['win_loss_ratio', 'nwin'], ascending=[False, False])

    filtered_df_sorted.reset_index(drop=True, inplace=True)

    return filtered_df

def lambda_handler(event, context):

    if event.get('queryStringParameters') and 'tournament_id' in event['queryStringParameters']:
        tournament_id = int(event['queryStringParameters']['tournament_id'])
    else:
        return {
            'statusCode': 400,
```

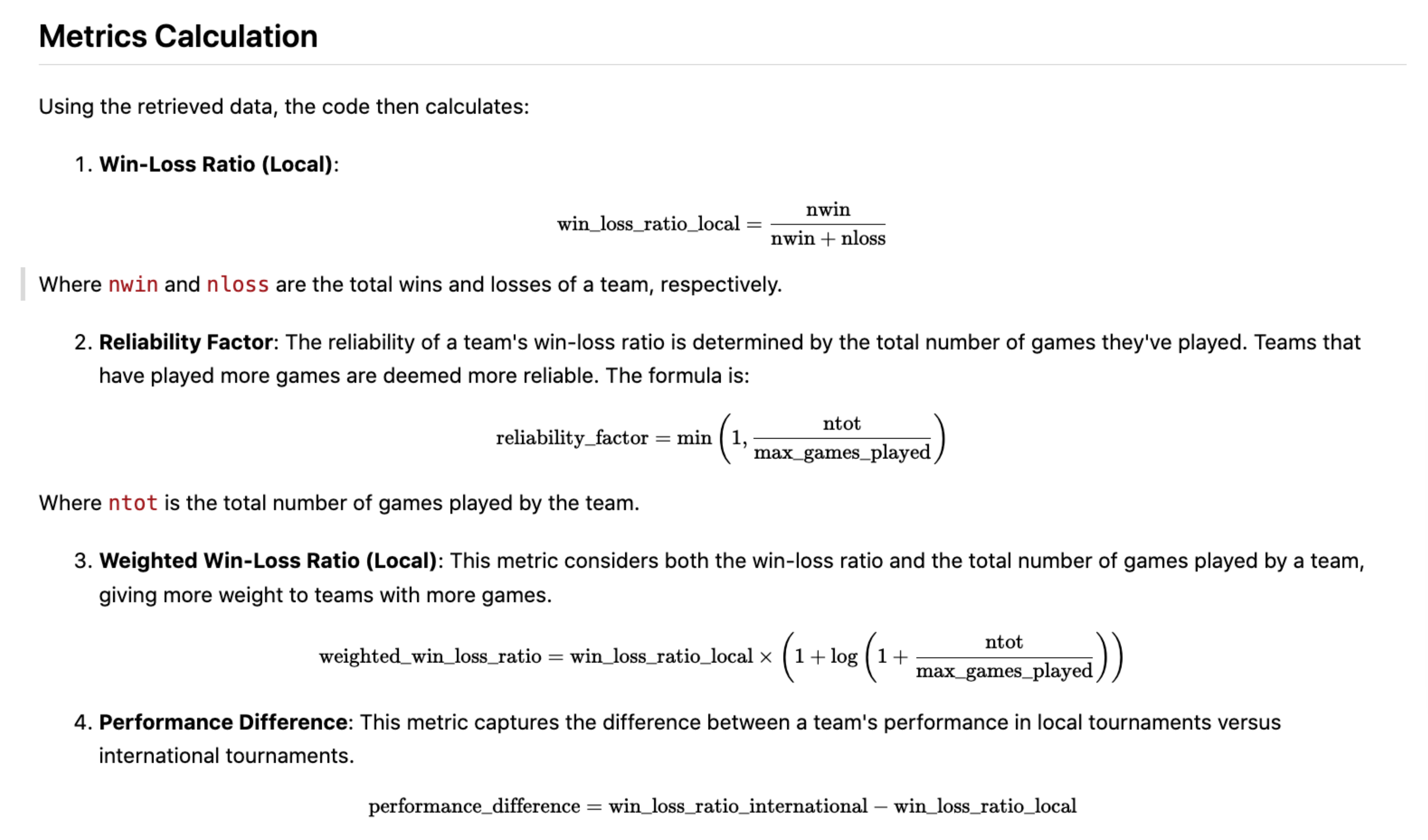
Team Rankings



Data Retrieval

- Collect tournament games across the entire database to construct regional strengths
- Filter based on games that have occurred in the last six months

Metrics



Metrics Calculation

Using the retrieved data, the code then calculates:

1. **Win-Loss Ratio (Local)**:

$$\text{win\_loss\_ratio\_local} = \frac{\text{nwin}}{\text{nwin} + \text{nloss}}$$

> Where `nwin` and `nloss` are the total wins and losses of a team, respectively.

2. **Reliability Factor**: The reliability of a team's win-loss ratio is determined by the total number of games they've played. Teams that have played more games are deemed more reliable. The formula is:

$$\text{reliability\_factor} = \min\left(1, \frac{\text{ntot}}{\text{max\_games\_played}}\right)$$

Where `ntot` is the total number of games played by the team.

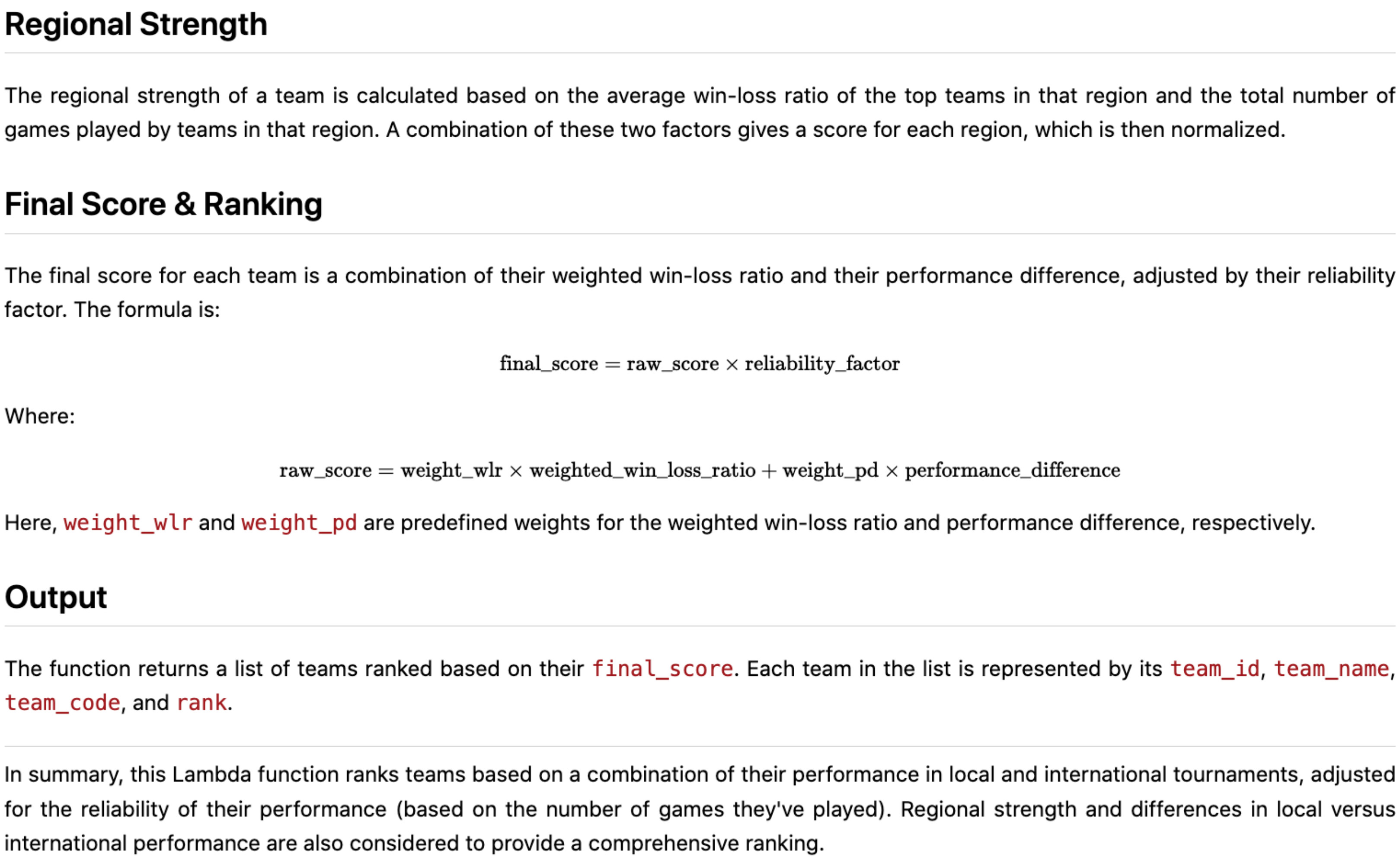
3. **Weighted Win-Loss Ratio (Local)**: This metric considers both the win-loss ratio and the total number of games played by a team, giving more weight to teams with more games.

$$\text{weighted\_win\_loss\_ratio} = \text{win\_loss\_ratio\_local} \times \left(1 + \log\left(1 + \frac{\text{ntot}}{\text{max\_games\_played}}\right)\right)$$

4. **Performance Difference**: This metric captures the difference between a team's performance in local tournaments versus international tournaments.

$$\text{performance\_difference} = \text{win\_loss\_ratio\_international} - \text{win\_loss\_ratio\_local}$$

Regional Strength and Rinal Ranking



Regional Strength

The regional strength of a team is calculated based on the average win-loss ratio of the top teams in that region and the total number of games played by teams in that region. A combination of these two factors gives a score for each region, which is then normalized.

Final Score & Ranking

The final score for each team is a combination of their weighted win-loss ratio and their performance difference, adjusted by their reliability factor. The formula is:

$$\text{final\_score} = \text{raw\_score} \times \text{reliability\_factor}$$

Where:

$$\text{raw\_score} = \text{weight\_wlr} \times \text{weighted\_win\_loss\_ratio} + \text{weight\_pd} \times \text{performance\_difference}$$

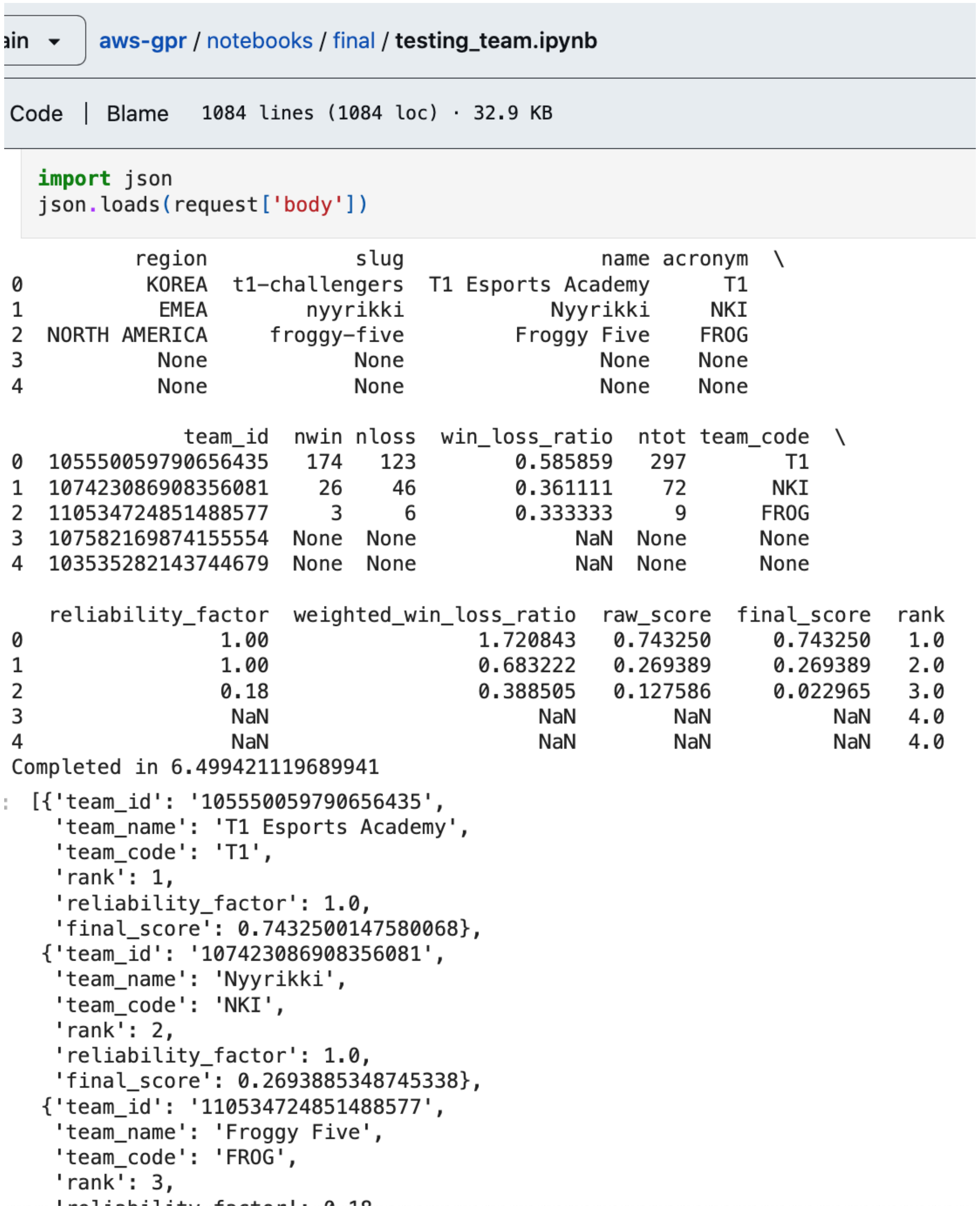
Here, `weight_wlr` and `weight_pd` are predefined weights for the weighted win-loss ratio and performance difference, respectively.

Output

The function returns a list of teams ranked based on their `final_score`. Each team in the list is represented by its `team_id`, `team_name`, `team_code`, and `rank`.

In summary, this Lambda function ranks teams based on a combination of their performance in local and international tournaments, adjusted for the reliability of their performance (based on the number of games they've played). Regional strength and differences in local versus international performance are also considered to provide a comprehensive ranking.

Testing the team ranking endpoint



ain ▾ aws-gpr / notebooks / final / testing_team.ipynb

Code | Blame 1084 lines (1084 loc) · 32.9 KB

```python
import json
json.loads(request['body'])
```

```
          region                slug                  name acronym  \
0          KOREA      t1-challengers    T1 Esports Academy      T1
1           EMEA            nyyrikki              Nyyrikki     NKI
2  NORTH AMERICA         froggy-five           Froggy Five    FROG
3           None                None                  None    None
4           None                None                  None    None

              team_id  nwin nloss  win_loss_ratio  ntot team_code  \
0  105550059790656435   174   123        0.585859   297        T1
1  107423086908356081    26    46        0.361111    72       NKI
2  110534724851488577     3     6        0.333333     9      FROG
3  107582169874155554  None  None             NaN  None      None
4  103535282143744679  None  None             NaN  None      None

   reliability_factor  weighted_win_loss_ratio  raw_score  final_score  rank
0                1.00                 1.720843   0.743250     0.743250   1.0
1                1.00                 0.683222   0.269389     0.269389   2.0
2                0.18                 0.388505   0.127586     0.022965   3.0
3                 NaN                      NaN        NaN          NaN   4.0
4                 NaN                      NaN        NaN          NaN   4.0
Completed in 6.499421119689941
```

```
[{'team_id': '105550059790656435',
  'team_name': 'T1 Esports Academy',
  'team_code': 'T1',
  'rank': 1,
  'reliability_factor': 1.0,
  'final_score': 0.7432500147580068},
 {'team_id': '107423086908356081',
  'team_name': 'Nyyrikki',
  'team_code': 'NKI',
  'rank': 2,
  'reliability_factor': 1.0,
  'final_score': 0.2693885348745338},
 {'team_id': '110534724851488577',
  'team_name': 'Froggy Five',
  'team_code': 'FROG',
  'rank': 3,
  'reliability_factor': 0.18
```

Workflow goes top to bottom

(Athena is also being used inside Lambda)

- S3: Setup external tables to main S3 bucket
- Athena: Explore the data and construct queries
- SageMaker: Process data and prepare for Lambda
- Lambda: Finalizing the endpoint codes
- API Gateway: Deploying the ranking endpoints

Athena

Q: What is Athena?

- Interactive query service
- Analyze s3 bucket using SQL

Q: Why use it?

- To explore data and construct queries
- Flexibility when testing, modifying, and running

Lifecycle:

1. Load the external tables.
2. Explore the data
3. Create query for extracting meaningful data
4. Migrate the query into SageMaker Python

Q: What is SageMaker?

- Fully managed service for building, training, and deploying ML models
- Analyze s3 bucket using SQL

Q: Why use it?

- Familiar and easy-to-use Jupyter interface
- Quickly write code to process the query results

Lifecycle:

- Migrate queries into SageMaker
- Process the resulting data to produce rankings
- Test functions to ensure functionality

SageMaker

Q: What is a Lamdba?

- Run code in cloud

Q: Why use it?

- Provides responses for API Gateway

Lifecycle:

1. Migrate the SageMaker codes into a new Lambda function
2. Set permissions
3. Enable AWSSDKPandas-Python311 Layer
4. Deploy and test the function

Lambda

Q: What is API Gateway?

- Managed service for APIs
- Includes features like traffic management, auth, and VCS.

Q: Why use it?

- Serves lambda functions to endpoints

Benefits:

1. Can handle quick increases in traffic and concurrent API calls.
2. Efficient Data Transfer
3. Cache

Lifecycle:

1. Create endpoint and associated method.
2. Link to lambda.

Lambda

IAM: User, Roles, and Permissions

- Separate users in IAM for collaboration with others on the same account.
- Permissions need to be set for API Gateway to call Lambda.
- Permissions are set on the role for the Lambda.
- Roles need full access to S3 and Athena permissions set.
- SageMaker has a role that requires S3 and Athena permissions.

Other services

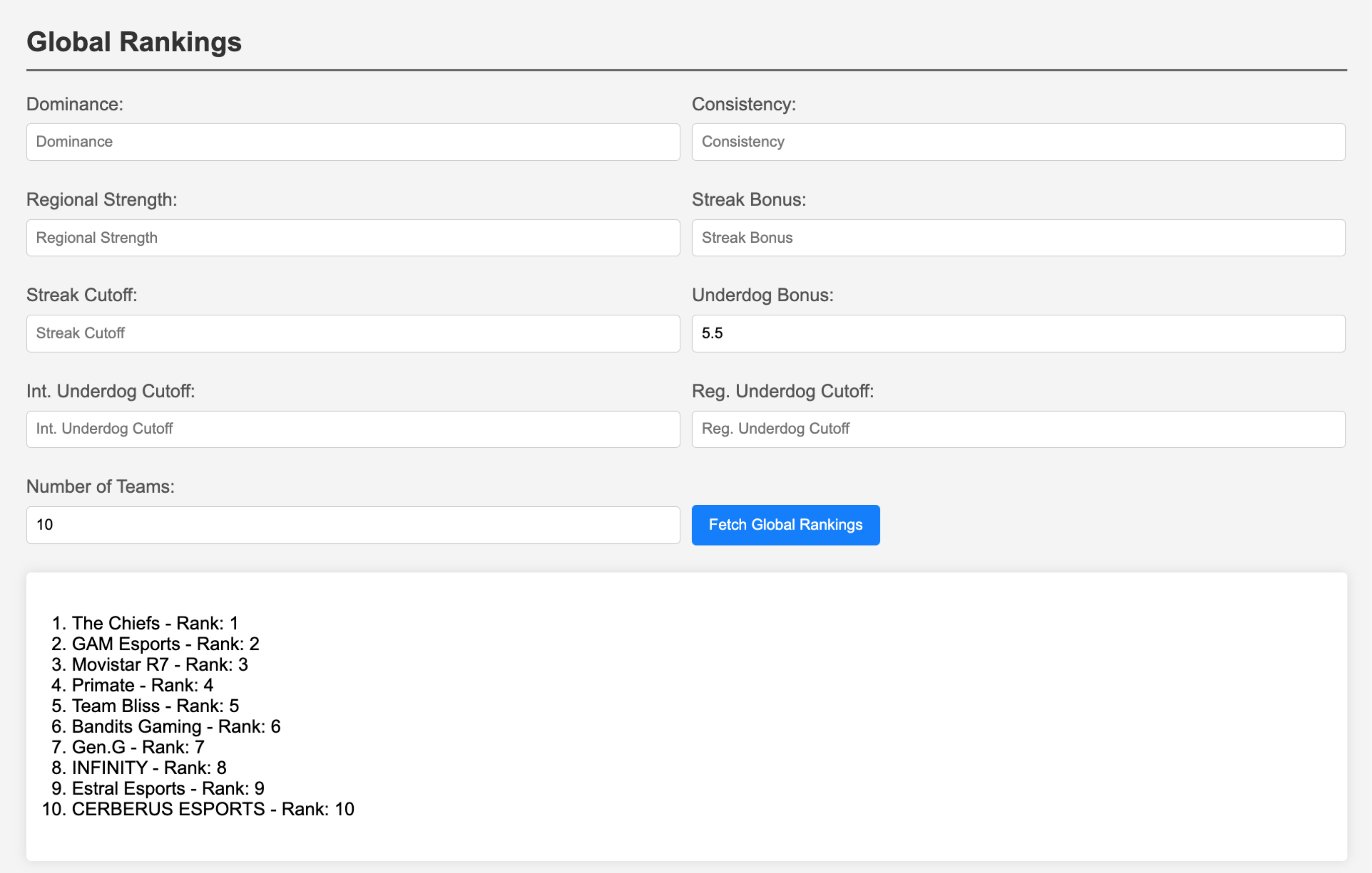
- Two other services were investigated

- Glue: Partition the games table to provide easy access for games based on date and teams.
- DynamoDB: Provide date lookup for games.

- Neither ended up being used. However, I think glue had great potential to enable me to efficiently query the games table.

Deployment and Testing

Simple frontend usage of API.



Global Rankings

Dominance: Dominance

Consistency: Consistency

Regional Strength: Regional Strength

Streak Bonus: Streak Bonus

Streak Cutoff: Streak Cutoff

Underdog Bonus: 5.5

Int. Underdog Cutoff: Int. Underdog Cutoff

Reg. Underdog Cutoff: Reg. Underdog Cutoff

Number of Teams: 10

1. The Chiefs - Rank: 1
2. GAM Esports - Rank: 2
3. Movistar R7 - Rank: 3
4. Primate - Rank: 4
5. Team Bliss - Rank: 5
6. Bandits Gaming - Rank: 6
7. Gen.G - Rank: 7
8. INFINITY - Rank: 8
9. Estral Esports - Rank: 9
10. CERBERUS ESPORTS - Rank: 10

Further work…

1. Take the league information into account. For example, some of the ratings for challenger league teams from a strong region may be inflated.
2. Improve the tournament ranking. Currently makes multiple calls to Athena. Furthermore, the ranking is poor quality in general and often teams are missing due to their games not being found in the tournaments table. This could be fixed by considering games from the games table.
3. None of the in game is data is used, which is sad, because there is so much. It would be fun to continue working on this project and incorporate it.
4. Incorporate data from external sources like Riot Games developer API, online 3d character models, and images.
5. Optimize the tournament ranking. Currently it makes multiple queries to the database which is not good for performance (~17s).
6. Improve the teams ranking by looking up the team if their data was not found in the tournaments table. This way all of the team names and codes will be present in the response. Additionally, if the games table is used then we could include their rank.

Conclusion

Overall,

- Happy with the rankings and progress I made towards learning the AWS infrastructure.
- Thankful to the organizers for providing invaluable resources for getting started with Athena.
- Thankful to AWS for providing credit during the tournaments. This gave me the confidence to initialize and deploy powerful services.