Athlete Performance Analysis Web App

1.Overview

In the era of big data, the sports industry has greatly benefited from the continued advancements in data capture, storage, and analysis. With this project, we aim to collect and analyze data from four popular sports: basketball, American football, soccer, and tennis. Users can access and learn about athletes' performance through an emulated distributed file system, which allows for easy data retrieval and analysis. This app will also help users identify each athlete's strengths and weaknesses.

We used three different databases and Django as a web design application framework to build a web project that can screen and analyze athlete data.

2.Data

2.1data description

We use four datasets in total which are NBA, Soccer, Tennis and Football datasets:

NBA player stats, 2018-2021. The dataset contains more than 2729 rows and 25 columns, which are Player, Pos, Age, Team, Games, Minutes Played, Fields Goal, Fields Goal Attempted, 3-points Field Goal, 3-points Field Goal Attempted, 2-points Field Goal, 2-points Field Goal Attempted, Free Throws, Free Throws Attempted, Offensive Rebounds, Defensive Rebounds Total Rebounds, Assists, Steals, Blocks, Turnovers, Personal Fouls, Points, Rank, Year. The dataset is derived from the website Kaggle, which is the

https://www.kaggle.com/datasets/sumitredekar/nba-stats-2018-2021m/datasets/patrasaurabh/csgoplayer-and-team-stats.

Global Soccer player statistics. Including data on 17,341 unique Soccer players and 53 columns of basic and performance information, which are Name, Nationality, National_Position, National_Kit, Club, Club_Position, Club_Kit, Club_Joining, Contract_Expiry, Rating, Height, Weight, Preffered_Foot, Birth_Date, Age, Preffered_Position, Work_Rate, Weak_foot, Skill_Moves, Ball_Control, Dribbling, Marking, Sliding_Tackle, Standing_Tackle, Aggression, Reactions, Attacking_Position, Interceptions, Vision, Composure, Crossing, Short_Pass, Long_Pass, Acceleration, Speed, Stamina, Strength, Balance, Agility, Jumping, Heading, Shot_Power, Finishing, Long_Shots, Curve, Freekick_Accuracy, Penalties, Volleys, GK（

Goalkeepers）_Positioning, GK_Diving, GK_Kicking, GK_Handling, GK_Reflexes Continent. The dataset is downloaded from

https://www.kaggle.com/datasets/antoinekrajnc/soccer-players-statistics

2017 ATP World Tour tennis player statistics. This dataset contains Player, Age, Elo , HardRaw, ClayRaw, GrassRaw, hard court elo rating, clay-court elo rating, grass-court elo rating, Peak Match, Peak Age, Peak Elo, Gender, Rank, which is downloaded from

https://www.kaggle.com/datasets/anupangadi/tennis-players-ranks-prediction-using-atp-elo.

National Football League offensive statistics for all players from 2019-2022, including Passing yards, rushing yards, receiving yards within 19974 rows and 69 columns. The data is collected from Kaggle: https://www.kaggle.com/datasets/mrframm/nfl-2020-combine

2.2 Data preprocessing

We create ER plots and database plots and then generate ID for all tables. We delete data with negative values and more than 50% missing values' rows, then get 2719, 17341, 784 and 19974 rows of data respectively.

2.3 Database architecture

We generated 8 tables for Soccer and Tennis data, 2 json files for NBA and NFL.

3. Workflow

Our workflow is shown below(Figure 1).

To do task1, we create an **emulated distributed file system** by writing a Tree.py (Figure 2), so that we could realize mkdir (add_dirs), ls (print), rm (delete), put (generate), get partition locations (getPartitionLocations), read partition (readPartition). **Spark** is used to import data into MySQL by connectMySQL.py, which is used to connect spark (Figure 3).

Task 2 will be described in section 4, and task 3 will be described in section 5.

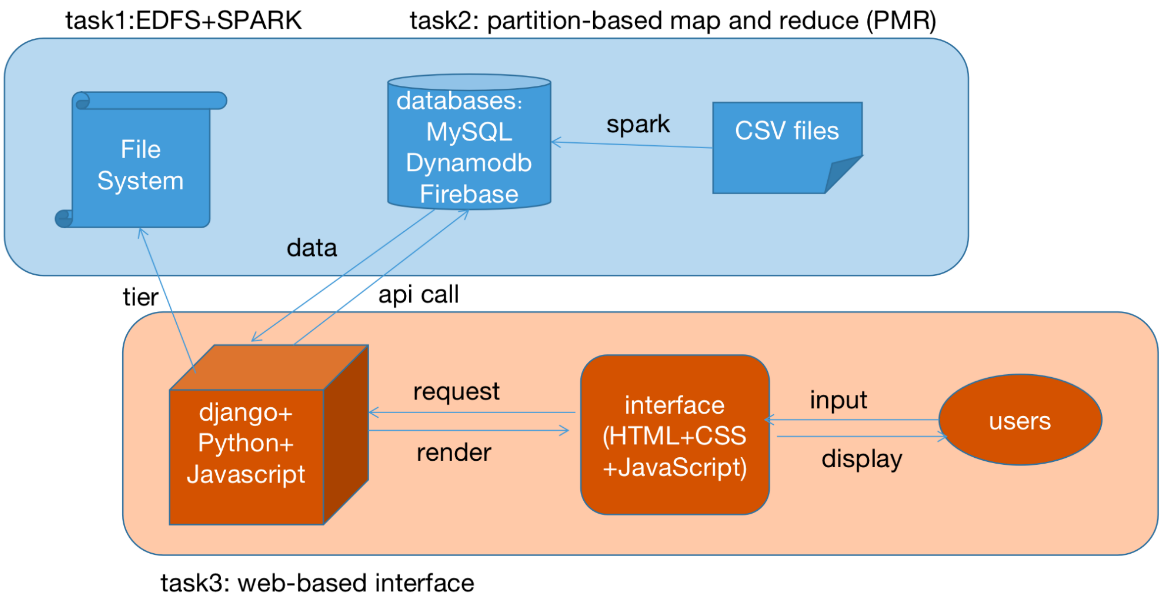
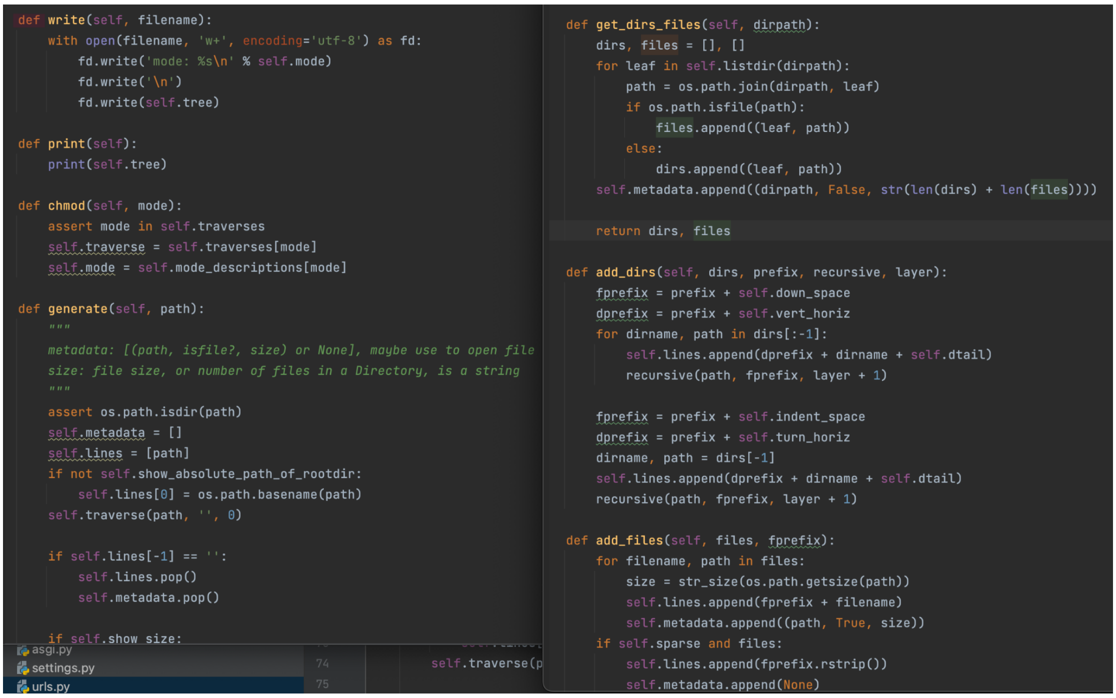


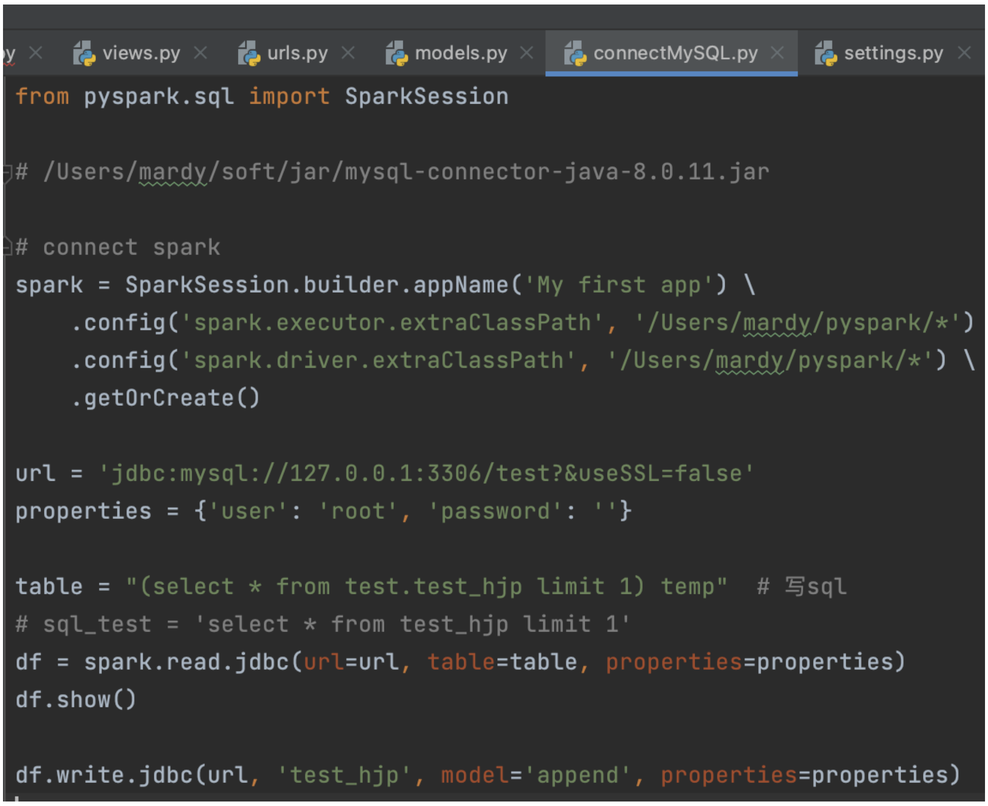
figure1. workflow



```python
def write(self, filename):
    with open(filename, 'w+', encoding='utf-8') as fd:
        fd.write('mode: %s\n' % self.mode)
        fd.write('\n')
        fd.write(self.tree)

def print(self):
    print(self.tree)

def chmod(self, mode):
    assert mode in self.traverses
    self.traverse = self.traverses[mode]
    self.mode = self.mode_descriptions[mode]

def generate(self, path):
    """
    metadata: [(path, isfile?, size) or None], maybe use to open file
    size: file size, or number of files in a Directory, is a string
    """
    assert os.path.isdir(path)
    self.metadata = []
    self.lines = [path]
    if not self.show_absolute_path_of_rootdir:
        self.lines[0] = os.path.basename(path)
    self.traverse(path, '', 0)

    if self.lines[-1] == '':
        self.lines.pop()
        self.metadata.pop()

    if self.show_size:
```

```python
def get_dirs_files(self, dirpath):
    dirs, files = [], []
    for leaf in self.listdir(dirpath):
        path = os.path.join(dirpath, leaf)
        if os.path.isfile(path):
            files.append((leaf, path))
        else:
            dirs.append((leaf, path))
    self.metadata.append((dirpath, False, str(len(dirs) + len(files))))

    return dirs, files

def add_dirs(self, dirs, prefix, recursive, layer):
    fprefix = prefix + self.down_space
    dprefix = prefix + self.vert_horiz
    for dirname, path in dirs[:-1]:
        self.lines.append(dprefix + dirname + self.dtail)
        recursive(path, fprefix, layer + 1)

    fprefix = prefix + self.indent_space
    dprefix = prefix + self.turn_horiz
    dirname, path = dirs[-1]
    self.lines.append(dprefix + dirname + self.dtail)
    recursive(path, fprefix, layer + 1)

def add_files(self, files, fprefix):
    for filename, path in files:
        size = str_size(os.path.getsize(path))
        self.lines.append(fprefix + filename)
        self.metadata.append((path, True, size))
    if self.sparse and files:
        self.lines.append(fprefix.rstrip())
        self.metadata.append(None)
```

Figure 2. Tree.py



```
from pyspark.sql import SparkSession

# /Users/mardy/soft/jar/mysql-connector-java-8.0.11.jar

# connect spark
spark = SparkSession.builder.appName('My first app') \
    .config('spark.executor.extraClassPath', '/Users/mardy/pyspark/*') \
    .config('spark.driver.extraClassPath', '/Users/mardy/pyspark/*') \
    .getOrCreate()

url = 'jdbc:mysql://127.0.0.1:3306/test?&useSSL=false'
properties = {'user': 'root', 'password': ''}

table = "(select * from test.test_hjp limit 1) temp"  # 写sql
# sql_test = 'select * from test_hjp limit 1'
df = spark.read.jdbc(url=url, table=table, properties=properties)
df.show()

df.write.jdbc(url, 'test_hjp', model='append', properties=properties)
```

Figure 3 connectMySQL.py (connect Spark)

4. Database

4.1 database creating

We use MySQL to store Soccer and Tennis datasets, Firebase to store NBA dataset, and Dynamodb to store NFL dataset.
To store Soccer dataset in MySQL, we firstly create four tables in models.py (Figure 4) in django. In order to apply **Partition** in this dataset, we then use the alter function to add the partition method in those tables(Figure 5). For the Soccer_athlete table, we use Range partition by Athlete_ID, for others we use Hash partition by their primary key.

```python
class Soccer_athlete(models.Model):
    #Athlete_ID = models.IntegerField(unique=True)
    Name = models.CharField(max_length=50, blank=True)
    Nationality = models.CharField(max_length=50, blank=True)
    Continent = models.CharField(max_length=50, blank=True)
    Birthday = models.DateField(blank=True)
    Rating = models.IntegerField(blank=True)
    National_Position = models.CharField(max_length=10, blank=True)
    National_Kit = models.CharField(max_length=10,blank=True)

class Soccer_club(models.Model):
    Club_ID = models.IntegerField(unique=True)
    Club = models.CharField(max_length=50, blank=True)
    Club_alias = models.CharField(max_length=10, blank=True)
class Soccer_contract(models.Model):
    Contract_ID = models.IntegerField(unique=True)
    Club_Joining = models.DateField(blank=True)
    Contract_Expiry = models.IntegerField(blank=True)
class Soccer_affiliation(models.Model):...
class Tennis_athlete(models.Model):...
class Tennis_hard(models.Model):...
class Tennis_clay(models.Model):...
class Tennis_grass(models.Model):...
class userInfo2(models.Model):...
```

Figure 4. tables created in django

```sql
ALTER TABLE Athletes_new_app01_Soccer_athlete PARTITION BY RANGE(Athlete_ID)  (
        PARTITION p0 VALUES LESS THAN (1000),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN (3000),
        PARTITION p3 VALUES LESS THAN (4000),
        PARTITION p4 VALUES LESS THAN (5000),
        PARTITION p5 VALUES LESS THAN (6000),
        PARTITION p6 VALUES LESS THAN (7000),
        PARTITION p7 VALUES LESS THAN (8000),
        PARTITION p8 VALUES LESS THAN (9000),
        PARTITION p9 VALUES LESS THAN (10000),
        PARTITION p10 VALUES LESS THAN (11000),
        PARTITION p11 VALUES LESS THAN (12000),
        PARTITION p12 VALUES LESS THAN (13000),
        PARTITION p13 VALUES LESS THAN (14000),
        PARTITION p14 VALUES LESS THAN (15000),
        PARTITION p15 VALUES LESS THAN (16000),
        PARTITION p16 VALUES LESS THAN (17000),
        PARTITION p17 VALUES LESS THAN MAXVALUE
);


ALTER TABLE Athletes_new_app01_Soccer_club
PARTITION BY Hash(Club_ID)
partitions 6;


ALTER TABLE Athletes_new_app01_Soccer_contract
partition by hash(Contract_ID)
partitions 7;


ALTER TABLE Athletes_new_app01_Soccer_affiliation
PARTITION BY hash(pl_id)
partitions 20;
```

Figure 5. Map partition

To store data in firebase, we firstly created data into json file by python(Figure 6), and use codes below to upload data to firebase(Figure 7) (which is not public) : https://group-47a89-default-rtdb.firebaseio.com

db = firebase.database()

storage = firebase.storage()

db.child("users").child(userUniqueId).set(json, user['idToken'])

```python
import csv
import json


# Function to convert a CSV to JSON
# Takes the file paths as arguments
def make_json(csvFilePath, jsonFilePath):

    # create a dictionary
    data = {}

    # Open a csv reader called DictReader
    with open(csvFilePath, encoding='utf-8') as csvf:
        csvReader = csv.DictReader(csvf)

        # Convert each row into a dictionary
        # and add it to data
        for rows in csvReader:

            # Assuming a column named 'No' to
            # be the primary key
            key = rows['ID']
            data[key] = rows

    # Open a json writer, and use the json.dumps()
    # function to dump data
    with open(jsonFilePath, 'w', encoding='utf-8') as jsonf:
        jsonf.write(json.dumps(data, indent=4))

# Driver Code

# Decide the two file paths according to your
# computer system
csvFilePath = r'./NBA_Data_final.csv'
jsonFilePath = r'./CSV2JSON.json'

# Call the make_json function
make_json(csvFilePath, jsonFilePath)
```

Figure 6 change data into json

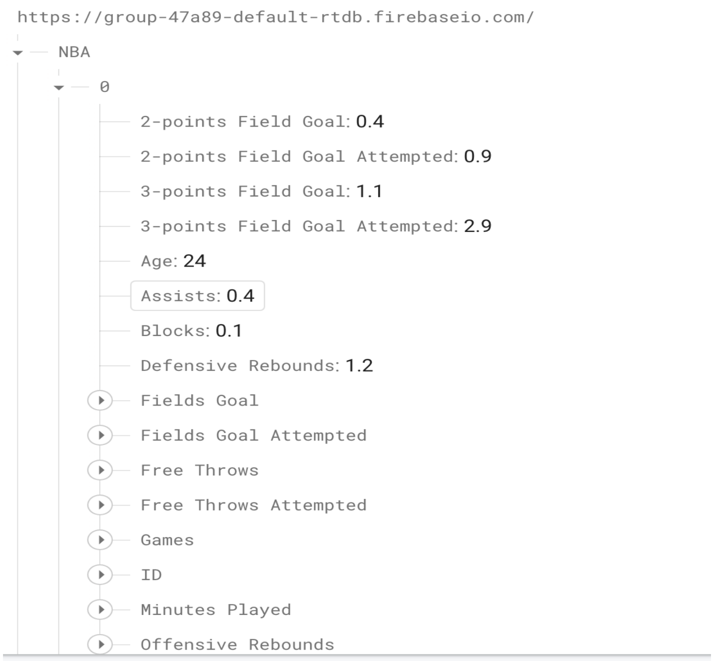


Figure 7. NBA data stored in firebase

We added an Accumulator using SPARK and used it to implement a counter (MapReduce) or a sum to calculate the total points scored by NBA players. Based on this SPARK function, we are able to analyze and summarize the scoring of players then.

To store NFL data in dynamodb, we need json files as well.(Figure 8)

ID	3Cone	40yd	Bench	Broad Ju
244	7.12	4.63	10	122
230	null	4.49	null	null
54	6.88	null	10	126
287	7.05	4.46	16	124
241	null	null	23	null
329	7.65	4.85	24	121
296	null	4.57	15	123

Figure 8 NFL data stored in dynamodb

4.2 pros and cons

DynamoDB: Because DynamoDB is part of AWS, it leverages the breadth of Amazon DynamoDB to enable synchronized data updates across multiple platforms. As a NoSQL database, DynamoDB is more flexible.

However, because DynamoDB is a key-value store database, each query must provide a partitioned key. It results in a restricted query language. Most importantly, DynamoDB uses a flexible pricing technique, which makes it difficult to predict the amount of money the database will generate.

MySQL Database:

MySQL is a relational (SQL) database, which stores tables of data with relations connecting the tables. If we have the relationships of tables, it is a good way to use MySQL. It is an open-source and well-performing database. However, since MySQL is not a standard language, it can not ensure safety.

Firebase Database:

Firebase Database is a cloud-hosted NoSQL database that lets you instantly store and synchronize data between users. It can store large datasets by hosting them. But Storage format is entirely different to that of SQL, (Firebase uses JSON),which means it can not migrate easily. Firebase is better suited for applications that have a centralized database updated by a large number of users, otherwise firebase does not have much of an advantage.

5. Web-based interface

5.1 Configuration

Pycharm pro, python 3.9, django, bootstrap5, html, css, javascript

5.2 system design

For our web-based interface, we use the python and Django frameworks for portable front-end presentation and back-end administration interfaces. Athletes Career Database needs to create table_display and data query interfaces for each dataset.

In Django, we firstly design interface style using HTML+CSS+Javascript+Bootstrap. We define functions for each page, and create path in url.py. In each function(Figure 9), we can receive request from websites by POST and GET method, which is used to create functions for searching data and generating data.

Figure 9. define functions for each web page and connect

```python
def search_soccer_page(request):
    table_col = ['Name', 'Club', 'Contract_ID', 'Reactions', 'Composure', 'Short Pass', 'Vision', 'Longpass
    rows = []
    returnprint =''
    dplayerid = np.nan
    if request.method == 'GET':
        return render(request, 'search_soccer_page.html')
    else:
        post_receive = request.POST
        searchByName = post_receive['searchByName']
        searchByClub = post_receive['searchByClub']
        checkDistribution = post_receive['checkDistribution']
        # print('####',checkDistribution)
        if searchByName:
            if searchByClub:
                print('no')
            else:...
        else:
            if searchByClub:...
        if checkDistribution:...

        return render(request, 'search_soccer_page.html', {'table_col':table_col, 'table_body':rows, 'retur
        # return HttpResponse('Welcome')
```

Figure 10. process requests

Figure 10 shows how we receive data from search_soccer_page, we could receive by request.POST/ request.GET.

To connect django with databases, we need to do configurations. For MySQL, we need to add DATABASES in settings.py file, for firebase and for dynamodb we need to put its config information into views.py file (Figure 11).

```python
#connect mysql
DATABASES = {
    'default':{
    'ENGINE': 'django.db.backends.mysql',
    'NAME':'athletes_new',
    'USER':'root',
    'PASSWORD':'',
    'HOST':'127.0.0.1',
    'PORT':3306,
    }
```

```python
module_dir = os.path.dirname(__file__)
img_folder = os.path.join(module_dir, './static/img')
config = {
        'apiKey': "AIzaSyD9czrjBtKsISm3Aq4GDhPQNFos9kIkeHY",
        'authDomain': "group-47a89.firebaseapp.com",
        'databaseURL': "https://group-47a89-default-rtdb.firebaseio.com"
        'projectId': "group-47a89",
        'storageBucket': "group-47a89.appspot.com",
        'messagingSenderId': "96044158076",
        'appId': "1:96044158076:web:60adf913dd198cd8a60df9",
        'measurementId': "G-4EZGB1NCPS"
    }
firebase = pyrebase.initialize_app(config)
authe = firebase.auth()
database = firebase.database()
```

```python
db3 = boto3.resource('dynamodb',
                     aws_secret_access_key='hY5mEvmLARSu+rRdJ/jhefw3U7xjk8DpF02DbWDF',
                     aws_access_key_id='AKIA4GIDTGJLI27Q56QC',
                     region_name='us-west-1')
table = db3.Table('NFL_551')
response = table.get_item( Key = {'ID': '1'})
print(response['Item'])
```

Figure 11. configurations of three databases

Figure 12 shows how we call data from these three databases.

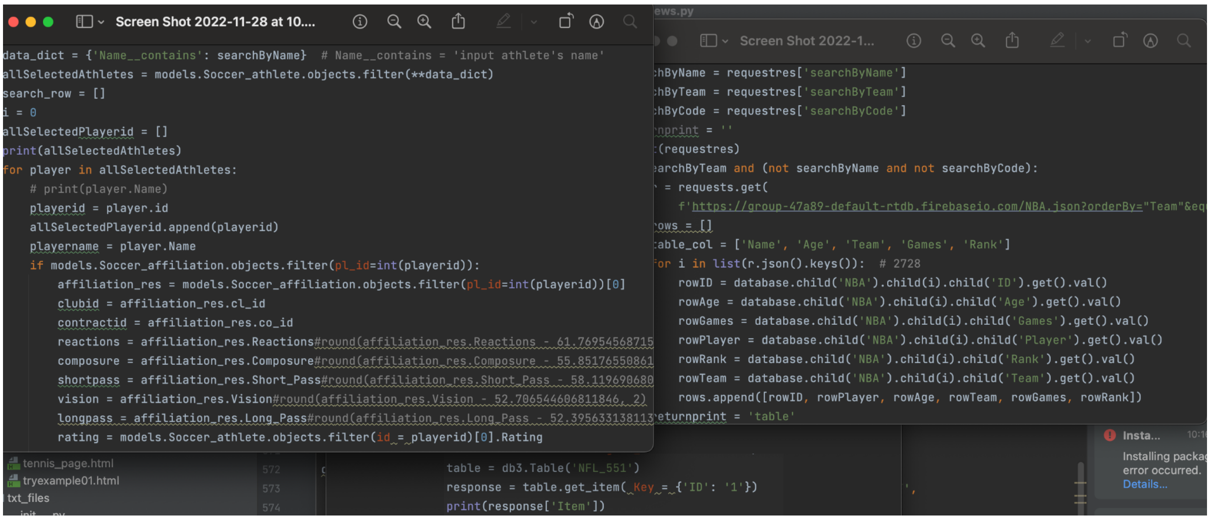


Figure 12 databases api call

After connecting databases to django, we will filter data in def functions(Figure 13):

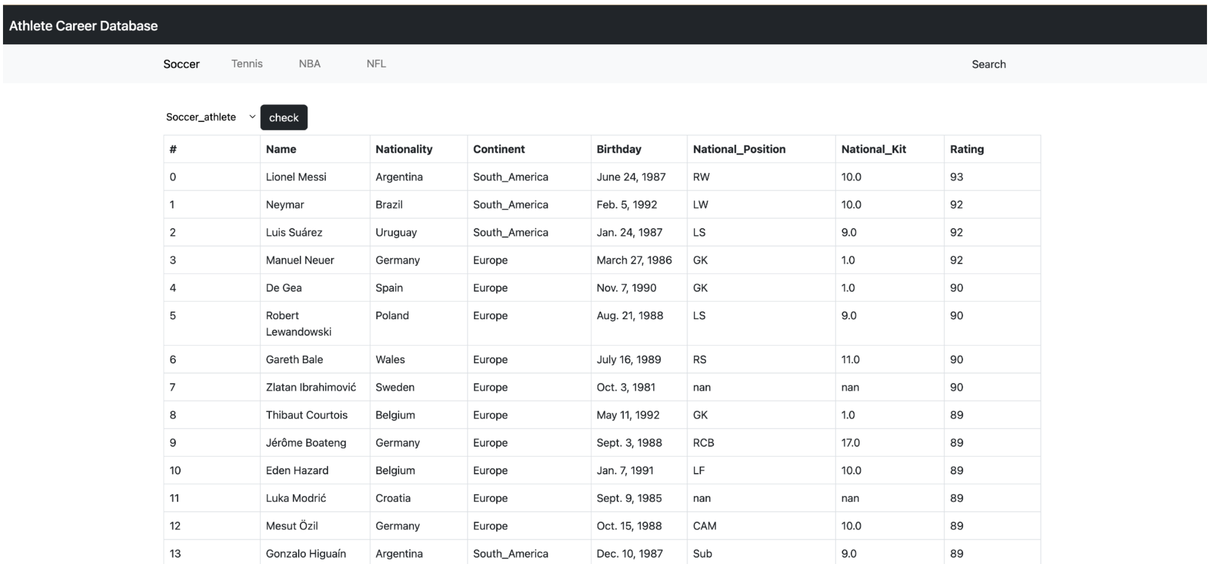
```python
def search_nba_page(request):
    if request.method =='POST':
        requestres = request.POST
        searchByName = requestres['searchByName']
        searchByTeam = requestres['searchByTeam']
        searchByCode = requestres['searchByCode']
        returnprint = ''
        print(requestres)
        if searchByTeam and (not searchByName and not searchByCode):
            r = requests.get(
                f'https://group-47a89-default-rtdb.firebaseio.com/NBA.json?orderBy="Team"&equalTo="{search
            rows = []
            table_col = ['Name', 'Age', 'Team', 'Games', 'Rank']
            for i in list(r.json().keys()):  # 2728
                rowID = database.child('NBA').child(i).child('ID').get().val()
                rowAge = database.child('NBA').child(i).child('Age').get().val()
                rowGames = database.child('NBA').child(i).child('Games').get().val()
                rowPlayer = database.child('NBA').child(i).child('Player').get().val()
                rowRank = database.child('NBA').child(i).child('Rank').get().val()
                rowTeam = database.child('NBA').child(i).child('Team').get().val()
                rows.append([rowID, rowPlayer, rowAge, rowTeam, rowGames, rowRank])
            returnprint = 'table'

            return render(request,'search_nba_page.html',{'returnprint':returnprint,'table_col':table_col,
        if searchByName and (not searchByTeam and not searchByCode):
            #ID and Name =>picture
            rows = []
            table_col = ['Name', 'Team','Fields Goal', 'Fields Goal Attempted', '3-points Field Goal', '2-
            r = requests.get(f'https://group-47a89-default-rtdb.firebaseio.com/NBA.json?orderBy="Player"&e
            for i in list(r.json().keys()):
                ID = database.child('NBA').child(i).child('ID').get().val()
                Player = database.child('NBA').child(i).child('Player').get().val()
```

Figure 13. functions in search_nba_page

5.3 Interface

We write 8 pages in total, which are athletesDB/, athletesDB/Tennis/, athletesDB/NBA/, athletesDB/NFL/, athletesDB/Search_Soccer/, athletesDB/Search_Tennis/, athletesDB/Search_NBA/, athletesDB/Search_NFL/.

All the table pages are similar, only showing tables in each dataset with a button for selecting table names (Figure 14).



#	Name	Nationality	Continent	Birthday	National_Position	National_Kit	Rating
0	Lionel Messi	Argentina	South_America	June 24, 1987	RW	10.0	93
1	Neymar	Brazil	South_America	Feb. 5, 1992	LW	10.0	92
2	Luis Suárez	Uruguay	South_America	Jan. 24, 1987	LS	9.0	92
3	Manuel Neuer	Germany	Europe	March 27, 1986	GK	1.0	92
4	De Gea	Spain	Europe	Nov. 7, 1990	GK	1.0	90
5	Robert Lewandowski	Poland	Europe	Aug. 21, 1988	LS	9.0	90
6	Gareth Bale	Wales	Europe	July 16, 1989	RS	11.0	90
7	Zlatan Ibrahimović	Sweden	Europe	Oct. 3, 1981	nan	nan	90
8	Thibaut Courtois	Belgium	Europe	May 11, 1992	GK	1.0	89
9	Jérôme Boateng	Germany	Europe	Sept. 3, 1988	RCB	17.0	89
10	Eden Hazard	Belgium	Europe	Jan. 7, 1991	LF	10.0	89
11	Luka Modrić	Croatia	Europe	Sept. 9, 1985	nan	nan	89
12	Mesut Özil	Germany	Europe	Oct. 15, 1988	CAM	10.0	89
13	Gonzalo Higuaín	Argentina	South_America	Dec. 10, 1987	Sub	9.0	89

Figure 14 mainpage of Athlete Career Database project

Clicking search, we could go to the search page of each sport.(Figure 15)

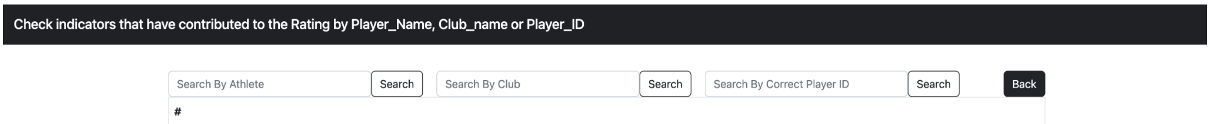
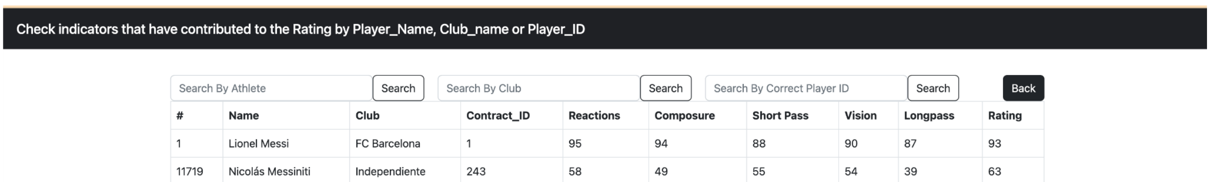


Figure 15 search page of soccer

In each search page, we could check data by Athlete name, Club name, or player ID. For example, searching Messi in Search By Athlete, we will get two pieces of information, since the filter function we use in Name_contains, which will search any players with Messi in their name. (Figure 16)



#	Name	Club	Contract_ID	Reactions	Composure	Short Pass	Vision	Longpass	Rating
1	Lionel Messi	FC Barcelona	1	95	94	88	90	87	93
11719	Nicolás Messiniti	Independiente	243	58	49	55	54	39	63

Figure 16 search by “Messi”

If we search by club, it will generate players who belong to this club. (Figure 17)

Check indicators that have contributed to the Rating by Player_Name, Club_name or Player_ID

Search By Athlete | Search | Search By Club | Search | Search By Correct Player ID | Search | Back

#	Name	Club	Contract_ID	Reactions	Composure	Short Pass	Vision	Longpass	Rating
1	Lionel Messi	FC Barcelona	1	95	94	88	90	87	93
32	Iniesta	FC Barcelona	31	88	88	92	94	86	88
38	Ivan Rakitić	FC Barcelona	37	79	81	87	87	92	87
41	Piqué	FC Barcelona	39	84	76	81	62	80	87
49	Sergio Busquets	FC Barcelona	47	82	82	88	84	79	86
51	Jordi Alba	FC Barcelona	11	82	75	79	68	70	86
115	Arda Turan	FC Barcelona	102	81	83	85	86	81	84
116	Javier Mascherano	FC Barcelona	23	83	86	79	68	75	84
124	André Gomes	FC Barcelona	108	82	78	85	83	84	83
143	Marc-André ter Stegen	FC Barcelona	58	80	66	30	57	38	83
204	Samuel Umtiti	FC Barcelona	166	75	66	78	56	71	82
311	Jasper Cillessen	FC Barcelona	243	78	70	47	60	33	81
375	Jérémy Mathieu	FC Barcelona	290	76	70	66	42	76	81
417	Denis Suárez	FC Barcelona	318	82	65	84	81	79	80
426	Rafinha	FC Barcelona	323	76	76	86	80	74	80

Figure 17. search by club= “FC Barcelona”

If we search by player id, it will generate distribution of these five skill scores among all players, with a vertical line of the player you search.(Figure 18)

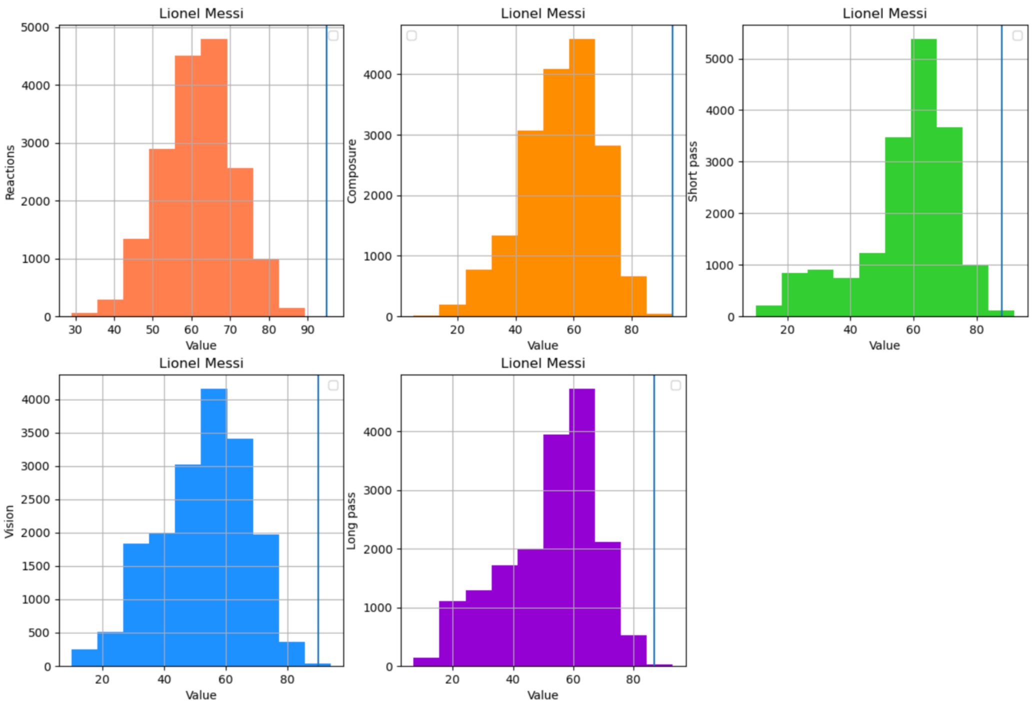


Figure 18. search by playerid = 1

6. Analysis

NBA and Soccer datasets include players' skill score and rating. Therefore, we would like to analyze which skill score contributed to their rating a lot. We use correlation matrix to find out which score has higher R score with rating and select top 5. For Soccer, they are Reactions, Composure, Short pass, Vision and Long pass. For NBA, they are Fields goal, Fields goal attempted, 3-points field goal, 2-points field goal attempted, and free throws. Figure 19 shows how we generated these results using python.

```
X = soccer_df[['Skill_Moves', 'Ball_Control', 'Dribbling', 'Marking',
        'Sliding_Tackle', 'Standing_Tackle', 'Aggression', 'Reactions','Attacking_Position',
         'Interceptions', 'Vision', 'Composure',
        'Crossing', 'Short_Pass', 'Long_Pass', 'Acceleration','Speed','Stamina', 'Strength', 'Balance',
         'Agility', 'Jumping', 'Heading',
        'Shot_Power','Finishing', 'Long_Shots', 'Curve', 'Freekick_Accuracy',
        'Penalties', 'Volleys', 'GK_Positioning', 'GK_Diving', 'GK_Kicking',
        'GK_Handling', 'GK_Reflexes','Rating']]
df_corr = X.corr()
```

```
df_corr.sort_values('Rating',ascending=False).iloc[1:].index.tolist()[:5]
```

```
['Reactions', 'Composure', 'Short_Pass', 'Vision', 'Long_Pass']
```

From every plot, we could compare the player you choose to all other players of these five indicators.

7. Learning Experience

This is our first time to clean the data from scratch and import it into a database. We also learned a lot about importing data, database construction, linking databases to the web, and building web pages.

In the use of spark clusters, it is really helpful in dealing with big datasets. Spark uses an in-memory(RAM) computing system whereas Hadoop uses local memory space to store data. So, compared with Hadoop, it has a higher speed in processing. And it also has low-latency in-memory data processing. But it also has some limitations, Spark is based on other database platforms which is not easy to implement. And for some small scale datasets, there is not a big difference in the data processing, so there is no need to use spark clusters to solve the problem. And for the analysis part, Spark does not have enough computational analysis methods and algorithms.

Link to our DemoVideo:

https://youtu.be/Va7toAUG5vQ

Responsibilities:

Yin Xu, Lingduo Luo, Junhong Duan