Project: NBA player Analyzer

Overview of the project:

In this project, I want to take NBA player data from the 2024-2025 season and use it to make some unique rankings based on different situations. I will do this by normalizing the statistics through min-max normalization and then also filter out outlier results such as removing players that only played less than 15-20 games, and players that were riding the bench (low minutes). After filtering the list of players I will then create weights for each statistic for different scenarios. This could then be used to see who is the best player overall for these situations such as “best clutch scorer”, “Best defensive player” “best playmaker” and more. I could also potentially add a way for users to make their own weights so they could see their own unique rankings. I also want to add a compare players feature where the users could select two specific players and compare them. With all of these unique rankings I want to create brand new csv’s displaying the standard csv’s along with some of the user made csv’s. If there is time I might even add a plotly application which will show a bar chart for the top ten players of each category and if there is additional time, a radar chart that could help compare players and a situation ranking visualization.

Idea of project:

1. Filtering players that barely played (no point in including them)
2. Normalization (use the max-min formula to normalize all the stats and remove potential outliers)
3. Weighted rankings (there are set weights, and the user can make custom weights that will be used to multiply the normalized stats that are included in the weights to give a score for each player for each situation
4. Player comparison (Users can select 2 specific players and do a detailed comparison for specific situations)
5. Data Export (users can create new CSV’s that show the ranked players for each situation) (there will be a top 20 and a full CSV of all the filtered players as the output)
6. Visualizations (This will utilize plotly to create graphs that show the top players for each situation, comparison between two players, and a relationship between players compared to the relationship of two different statistics.

Sample datasheet:

Player	Age	Team	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS

Folder overview:
Final_project.py: contains all the code for this project

CS final project starter CSV - Sheet1.csv (contains all the initial data for the project)
This pdf
Extra CSVs that were tests to see if the exporting aspect of the project worked as intended

What is needed to make this project a reality:
Imports:
Import csv
Import plotly.express as px
Import plotly.graph_objects as go
Import pandas as pd

Classes:

Player(self, name: str, team:str, position: str, age: int, games_played: int, min_played: int, stats: dict):

Attributes:

- Name
- Team
- Position
- Age
- Games_played
- min_played
- Stats (raw stats dictionary)
- Norm_stats (normalized stats dictionary)
- Overall score (set to 0)
- self.id (in case players have two teams)

Methods:

- get_stat(self, self_name):
  - Used to get a specific value of a specific stat
- set_stat(self, stat_name, value):
  - Used to create new stats and also update previous stats
- Calculated score (self, weights):
  - Done by using normalized stats and the weight dictionary to generate a score
- to_dict(self)
  - Returns data (raw or normalized) in dictionary form for exporting

RankingSystem(self, min_games_played: int = 20, min_minutes_played: int = 15, weights: dict = None)

Attributes

- players (list of all the player objects)
- min_games_played (thresholds to filter players)
- min_minutes_played (another threshold to filter players)
- norm_min (dictionary of all the min values for each stat that is normalized)

- norm_max (dictionary of all max values for each stat that is normalized)
- weights (weights dictionary)

Methods:

- add_player(self, player_or_list):
  - Adds the player objects to the players list
- normalize_value(self, value, min_value, max_value):
  - Normalization formula
- calculate_min_max(self):
  - Uses the normalization formula for all the stats each player has
- apply_weights(self)
  - Applies the weights to the normalized stats and calculates the score
- rank_player(self, top_n = 10)
  - Ranks the players based of the score from the weights
- normalize_and_rank(self)
  - Combines the previous three functions to normalize and rank all the players
- rank_by_situation(self, weights, top_n=10)
  - Used to change the weights that the players are ranked by
- to_dict_list(self, top_n = None):
  - Exports either the top_N players or all the players into a list with dictionaries

Situational weight dictionaries

- Default_weights (points, total rebounds, assists, steals, blocks, and turnovers (negative))
- Clutch_scoring_weights (points, FG%, eFG%, FT%, TOV (negative))
- Defensive_weights (defensive rebounds, blocks, steals)
- Playmaking_weights (assists, turnovers, offensive rebounds, personal fouls (negative) efficiency)
- Custom_weights (user created)

Other helper functions

- read_csv_as_dicts(filename):
  - This will read the csv and return all the values as a dictionary (each row is a new key) (also convert stat types if necessary)
- create_custom _weights()
  - Can ask the user what stats should be used along with how much they should be weighted
- compare _players(p1, p2, weights, scenario_name=”Custom”):
  - Used to do a detailed comparison of two players' normalized and weighted statistics
  - Has a calculated_weighted(player) function to help differentiate the two players based of their normal and weighted stats and their score
- export_rankings_to_csv(players, filename)

    - Used to create new CSVs where each player has their raw data, but also their ranking, so they are listed in order of their ranking based of the situation (could be however long the list of players is (full or top 20)
- export_top_n_to_csv(players, filename, top_n=20)
    - Used to specifically export CSVs of the top players
- flatten_player_dicts(player_dicts)
    - Used to flatten the nested player dictionaries so Plotly will be able to use them. It does this by combining the raw and normalized stats into one dictionary.
- plot_top_players(player, title=f"Top Players", scenario_name ="")
    - Used to create a bar graph of the top players based of each situation via plotly
- plot_player_comparision(p1, p2)
    - Used to create a radar chart comparing two specific players based on the different stats via plotly
- plot_scatter(players, stat_x, stat_y):
    - Used to create a scatter plot containing all the players and then also show where they will be according to a relationship between two user chosen stats
- find_player_by_name(players, query)
    - Helps the comparison functions by looking for user defined players in the list of all players (case insensitive)
- weight_situation():
    - Allows the user to specifically choose what type of weight catergory they want to use to rank the players
- list_all_players(players)
    - Displays all the players in the list to help the user make a choice for what two players they want to pick
- main_menu()
    - Where everything is displayed:

Essentially the "game_loop" for the project.
Utilizes a while loop to allow the user to do many different things
with the NBA player analyzer. This combines all the functions from earlier to
show top overall players, top players for situations,
compare two players, export rankings to CSV, see the top players for each situation
bar graph, see the radar chart when comparing two players, and see a relationship between
players and two stats via scatterplot. There is also the exit option to ensure the program
doesn't cause an infinite loop

Needed data structures

- Dictionary of all the stats: stat_min_max = {stat_name: (min_val, max_val)} (will be used for the normalization)
- Dictionary of weights (situation and weight values)
- List of player objects

- List of top-ranked players

CSV output stuff (csvs I will create)

- overall_rankings.csv
- situation_rankings_clutch.csv
- situation_rankings_defense.csv
- custom_rankings_userX.csv
- plotly HTML files for charts

This all can be done by using functions to convert player objects to rows and write CSV manually with a file write function or csv.writer