IDS exercise 1

Riccardo Cereghino

Mar 19, 2021

This module generates plots on statistic extracted from the csv file which can be found at: https://www.kaggle.com/martj42/international-football-results-from-1872-to-2017

`indicator.functional.`**`csv_reader`**(*file_name: str*) → Iterator[str]
: Generates an iterator per line from a file encoded in utf8, specified with file_name

`indicator.functional.`**`generate_indicators`**(*file: str*) → Iterator[Dict[str, Union[str, int, float, List[int], List[datetime.date]]]]
: Iterates `generate_rows()` which iterates `generate_rows()` to `update_indicator()` s of teams in the csv file

  After all the iterations, yields the result

`indicator.functional.`**`generate_match_data`**(*row: Iterator[Dict[str, str]]*) → Iterator[Dict[str, Union[str, int, List[datetime.date]]]]
: Given an iterator from `generate_rows()` yield relevant data per team

`indicator.functional.`**`generate_operators`**(*x: Union[dict, List[dict]]*, *op: Callable*, *k: str*, *v: Any*) → Iterator[Callable]
: Auxiliary function of `operators_reader()`, to return multiple operators if x is list

`indicator.functional.`**`generate_rows`**(*file: str*) → Iterator[Dict[str, str]]
: Iterates through a csv file (path), picks the first line to be used as keys for the yielded list of returning dict

`indicator.functional.`**`operators_reader`**(***kwargs: Union[str, int, float, List[int], List[datetime.date]]*) → List[Callable]
: Given any number of kwargs in the form: - input is in the form:

```
team_name__eq="Italy"
avg_goals_scored__gte=1
```

  It returns a list of functions from the operator library, based on the *__eq* section of the kwargs keyword.

  The arguments of the operator function will be, on the left, the value of the key of the yielded dict and on the right the value od the key of the corresponding kwargs value.

  It is used in `select()`.

`indicator.functional.`**`plot`**(*ind: dict*)
: Plots the data of a match

`indicator.functional.`**`prettify`**(*ind: dict*)
: Prints a match indicator human readable

`indicator.functional.`**`prettyficator`**(*it: Iterator[dict]*)
: Prints match indicators human readable

`indicator.functional.`**`row_splitter`**(*row: str*) → List[str]
: Given a string returns a csv row, splits the cells into elements of a list

  - **input is in the form::** row = “a,b,cn”

`indicator.functional.`**`select`**(*it: Iterator[Dict[str, Union[str, int, float, List[int], List[datetime.date]]]]*, ***kwargs: Union[str, int, float, List[int], List[datetime.date]]*) → Iterator[dict]
: Given a dict iterator (it) and any number of kwargs, - in the form:

```
team_name__eq="Italy"
avg_goals_scored__gte=1
```

  Returns a `filter()` iterator, filtering based on the condition specified in kwargs`.

`indicator.functional.`**`selector`**(*ind: Dict[str, Union[str, int, float, List[int], List[datetime.date]]]*, *mode: str*, *operators: List[Callable]*) → bool

Given a List of comparisons (operators, ex: x < 3) returns a boolean expressing a simple logic port for all operators, it is used in *`select()`*

`indicator.functional.`**`update_indicator`**(*ind: Dict[str, Union[str, int, List[int]]]*, *md: Dict[str, Union[str, int, List[int]]]*) → Dict[str, Union[str, int, List[int], List[datetime.date]]]

Updates team indicator (ind) with yielded match data (md)

```python
def row_splitter(row: str) -> List[str]:
    return row[:-1].split(',')
```

```python
def csv_reader(file_name: str) -> Iterator[str]:
    for line in open(file_name, "r", encoding="utf8"):
        yield line
```

```python
def generate_rows(file: str) -> Iterator[Dict[str, str]]:
    csv_gen = csv_reader(file)

    columns = row_splitter(next(csv_gen))

    for row in csv_gen:
        yield dict(zip(columns, row_splitter(row)))
```

```python
def generate_match_data(row: Iterator[Dict[str, str]]) -> Iterator[Dict[str,␣
↪Union[str, int, List[date]]]]:
    for r in iter(row):
        if r["tournament"] == "FIFA World Cup":
            yield {
                "team_name": r["home_team"],
                "home_goals": int(r["home_score"]),
                "away_goals": int(r["away_score"]),
                "date": date.fromisoformat(r["date"])
            }
            yield {
                "team_name": r["away_team"],
                "home_goals": int(r["away_score"]),
                "away_goals": int(r["home_score"]),
                "date": date.fromisoformat(r["date"])
            }
```

```python
def update_indicator(ind: Dict[str, Union[str, int, List['int']]],
                     md: Dict[str, Union[str, int, List['int']]]) -> Dict[str,␣
↪Union[str, int, List[int], List[date]]]:
    ind["date"].append(md["date"])

    ind["goals_scored_list"].append(md["home_goals"])
    ind["goals_taken_list"].append(md["away_goals"])

    if md["home_goals"] > md["away_goals"]:
        ind["wins"] += 1
        ind["win_streaks"][-1] += 1
    else:
        if md["home_goals"] < md["away_goals"]:
            ind["losses"] += 1
        else:
            ind["draws"] += 1
        if ind["win_streaks"][-1] != 0:
            ind["win_streaks"].append(0)
    return ind
```

```python
def generate_indicators(file: str) -> Iterator[Dict[str, Union[str, int, float, List[
↪'int'], List[date]]]]:
    rows = generate_rows(file)
```

(continues on next page)

(continued from previous page)

```
    inds = {}
    for match_data in generate_match_data(rows):
        if inds.get(match_data["team_name"]) is None:
            inds[match_data["team_name"]] = {
                "team_name": match_data["team_name"],
                "date": [],
                "wins": 0,
                "losses": 0,
                "draws": 0,
                "avg_goals_scored": 0,
                "avg_goals_taken": 0,
                "win_streaks": [0],
                "goals_scored_list": [],
                "goals_taken_list": []
            }
        inds[match_data["team_name"]] = update_indicator(inds[match_data["team_name
↪"]], match_data)

    for el in inds.values():
        el["max_win_streak"] = max(el.pop("win_streaks"))
        matches = el["wins"] + el["losses"] + el["draws"]
        el["avg_goals_scored"] = sum(el["goals_scored_list"]) / matches
        el["avg_goals_taken"] = sum(el["goals_taken_list"]) / matches
        yield el
```

```
def selector(ind: Dict[str, Union[str, int, float, List[int], List[date]]],
             mode: str, operators: List[Callable]) -> bool:
    result = False if mode == 'or' else True

    for operation in operators:
        _result = operation(ind)

        if mode == 'or':
            result = _result or result
        else:
            result = _result and result
    return result
```

```
def operators_reader(**kwargs: Union[str, int, float, List[int], List[date]]) ->␣
↪List[Callable]:
    operators = []
    for kw in kwargs:
        if '__' in kw:
            _kw, op = kw.split('__')
            assert op in ('lt', 'le', 'eq', 'ne', 'ge', 'gt')
        else:
            op = 'eq'
            _kw = kw

        _operator = getattr(operator, op)

        operators.append(lambda x: generate_operators(x, _operator, _kw, kwargs[kw]))

    return operators
```

```
def generate_operators(x: Union[dict, List[dict]], op: Callable, k: str, v: Any) ->␣
↪Iterator[Callable]:
    if isinstance(x, list):
        for el in x:
            return op(el.get(k), v)
    else:
        return op(x.get(k), v)
```

```
def select(
        it: Iterator[Dict[str, Union[str, int, float, List[int], List[date]]]],
        **kwargs: Union[str, int, float, List[int], List[date]]
) -> Iterator[dict[str, Union[str, int, float, List[int], List[date]]]]:
    mode = kwargs.pop('mode', 'or')
    operators = operators_reader(**kwargs)
    return filter(lambda el: selector(el, mode, operators), it)
```

```
def prettify(ind: dict[str, Union[str, int, float, list[int], List[date]]]):
    print(
        "{}, wins: {}, losses: {}, draws: {}, scored goals avg: {}, taken goals avg:
↪{}, max_win_streak: {}".format(
            ind["team_name"], ind["wins"], ind["losses"], ind["draws"], ind["avg_
↪goals_scored"],
            ind["avg_goals_taken"], ind["max_win_streak"]
        )
    )
```

```
def prettyficator(it: Iterator[dict[str, Union[str, int, float, List[int],␣
↪List[date]]]]):
    for el in it:
        prettify(el)
```

```
def plot(ind: dict):
    plt.xlabel("Goals")
    plt.ylabel("Date")

    str_dates = [d.isoformat() for d in ind["date"]]

    plt.plot(ind["goals_scored_list"], str_dates, label='Goals Scored')
    plt.title(ind["team_name"])
    plt.plot(ind["goals_taken_list"], str_dates, label='Goals Taken')
    plt.legend()

    plt.show()
```

```
if __name__ == '__main__':
    indicators = generate_indicators(os.path.abspath('indicator/results.csv'))
    ind_1, ind_2, ind_3 = tee(indicators, 3)

    print("Iceland indicators")
    S = list(select(ind_1, team_name__eq="Iceland")).pop()
    prettify(S)
    plot(S)

    search_params = {
        "mode": "and",
```

(continues on next page)

(continued from previous page)

```
        "wins__gt": S.get("wins"),
        "losses__lt": S.get("losses"),
        "avg_goals_scored__gt": S.get("avg_goals_scored"),
        "avg_goals_taken__lt": S.get("avg_goals_taken"),
        "max_win_streak__gt": S.get("max_win_streak"),
        "date_lte": date.today()
    }

    print("Teams with indicators better than Iceland")
    prettyficator(select(ind_2, **search_params))

    print("Italy indicators")
    S = list(select(ind_3, team_name__eq="Italy")).pop()
    prettify(S)
    plot(S)
```