How to beat your friends in fantasy football

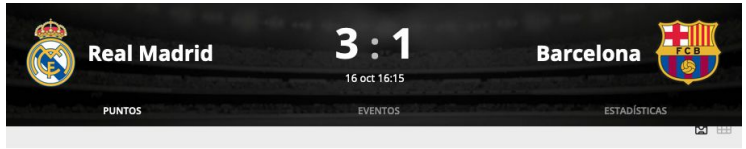
Alex Molas - Senior Data Scientist @ Stuart

@MolasAlex
www.alexmolas.com

What’s fantasy football?

Participants, using a fixed budget, assemble a team of real life players and score points based on those players' actual performance.

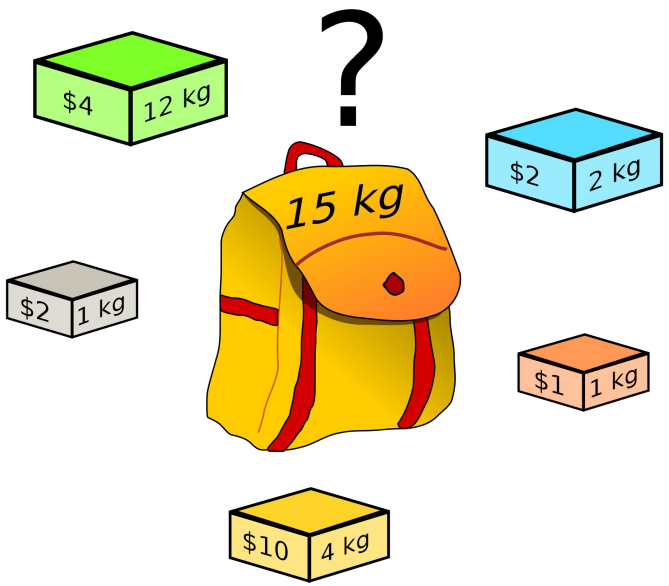
Pos.	Eq.	Jugador	Pts.	VM
DL		Lewandowski	140	65.000.000 €
MC		Valverde	122	30.000.000 €
DL		Griezmann	113	32.000.000 €
DL		Iago Aspas	110	40.000.000 €
MC		Brais Méndez	110	21.000.000 €
MC		Aleix García	108	11.000.000 €
DL		Vinícius Jr	107	54.000.000 €
MC		Pedri	103	43.000.000 €
DL		Dembélé	99	30.000.000 €



Knapsack problem

The problem can be modeled as a **knapsack problem**.

Which boxes should be chosen to maximize the amount of money while still keeping the overall weight under or equal to 15 kg?



Knapsack problem

The problem can be modeled as a **knapsack problem**.

Which boxes should be chosen to maximize the amount of money while still keeping the overall weight under or equal to 15 kg?

Solving knapsack problem

We need

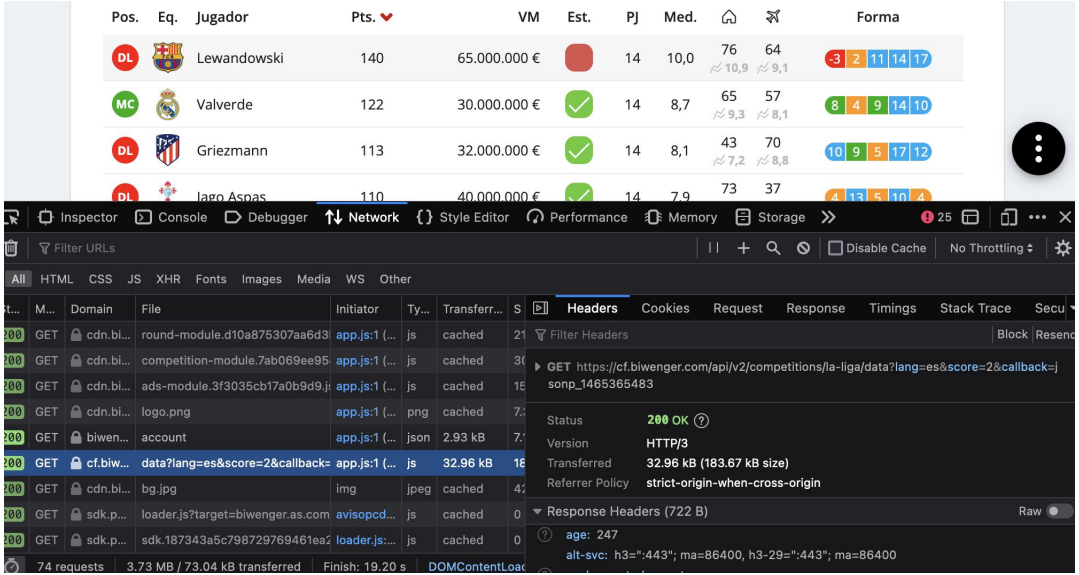
- Set of all players with their characteristics.
- A way to get the value of a player.
- A way to get the price of a player.
- Constraints: max salary we can pay.

```
def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:

```

Players data

Biwenger doesn’t have an open API. Reverse engineering using the **Inspect Tool.**




```python
import requests
def get_data(competition: str = "la-liga"):
    data = requests.get(
        f"{base_path}/competitions/{competition}/data"
        ).json()
    return data
```

Players value & cost

```python
@dataclass(frozen=True)
class Player:
    ...
    price: int
    status: str
    points: int
    games: int
    ...

```

```python
def player_value(p: Player) -> float:
    if p.status == 'ok':
        return p.points / p.games
    return 0.

```

Here you can add some ML!

```python
def player_cost(p: Player) -> int:
    return p.price

```

Solving the problem: ortools

```python
def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:
    # define the solver
    solver = pywraplp.Solver('CoinsGridCLP', pywraplp.Solver.CBC_MIXED_INTEGER_PROGRAMMING)
     ...
```

Solving the problem: ortools

```python
def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:
     ...
     # 0 if player is not selected, 1 if player is selected
     take = [solver.IntVar(0, 1, f'take_{p}') for p in players]
     ...
```

Solving the problem: ortools

```python
def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:
    ...
    # total cost of the team. This variable defines a constraint
    salary = solver.Sum([player_cost(p) * take[i] for i, p in enumerate(players)])
    solver.Add(salary ≤ max_salary)
    ...
```

Solving the problem: ortools

```python
def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:
    ...
    # total value of the team. This is the variable to maximize
    value = solver.Sum([player_value(p) * take[i] for i, p in enumerate(players)])
    solver.Maximize(value)
    ...
```

Solving the problem: ortools

```python
def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:
    ...
    # solve the problem
    solver.Solve()

    # build the team
    team = set()
    for i, p in enumerate(players):
        if take[i].SolutionValue():
            team.add(p)
    return team
```

Solving the problem: ortools

```python
from ortools.linear_solver import pywraplp

def solve(players: Set[Player],
          player_value: Callable[[Player], float],
          player_cost: Callable[[Player], float],
          max_salary: int) → Set[Player]:
    # define the solver
    solver = pywraplp.Solver('CoinsGridCLP', pywraplp.Solver.CBC_MIXED_INTEGER_PROGRAMMING)

    # 0 if player is not selected, 1 if player is selected
    take = [solver.IntVar(0, 1, f'take_{p}') for p in players]

    # total cost of the team. This variable defines a constraint
    salary = solver.Sum([player_cost(p) * take[i] for i, p in enumerate(players)])
    solver.Add(salary ≤ max_salary)

    # total value of the team. This is the variable to maximize
    value = solver.Sum([player_value(p) * take[i] for i, p in enumerate(players)])
    solver.Maximize(value)

    # solve the problem
    solver.Solve()

    # build the team
    team = set()
    for i, p in enumerate(players):
        if take[i].SolutionValue():
            team.add(p)
    return team
```

Best team

- Constraints
  - Salary: 200M
  - 343 / 352 / 433 / 442 / 451 / 532 / 541
- Solution
  - Cost: 198M
  - 343
  - Expected value: 100 points

Thanks for your attention!

@MolasAlex

www.alexmolas.com