Optimiser - Fantasy Football

Aim of the Project

🔗 **App link**: Streamlit App

This project focuses on optimising Fantasy Football teams by maximising predicted points for upcoming gameweeks. The key objectives are:

- Predict total points a user's Fantasy Premier League (FPL) team will score.
- Select the optimal starting XI lineup.
- Choose the best captain for maximum points.
- Recommend player transfers to improve team performance.

Rules of Fantasy Premier League (FPL)

When building and optimizing a Fantasy Premier League squad, the following rules and constraints must be respected to ensure squad validity:

Squad Composition

- **Total squad size:** 15 players
- **Positions:**
  - 2 Goalkeepers
  - 5 Defenders
  - 5 Midfielders
  - 3 Forwards

Starting Lineup Constraints

- **Starting XI size:** 11 players

- **Formation:** The starting lineup must follow one of the allowed FPL formations:
  - 3-4-3, 3-5-2
  - 4-4-2, 4-3-3
  - 5-3-2, 5-4-1
- Exactly 1 goalkeeper must be in the starting XI.

Team Limits

- **Maximum players from one Premier League club:** 3 in the whole squad (including bench and starting XI).

Transfer Rules

- **Transfers per gameweek:** 1 free transfer per gameweek, if not used they carry over to next gameweek (max 3); additional transfers cost points (-4).
- **Squad validity:** All transfers must maintain a valid squad according to the above rules.
- **Transfer limits in optimization:** The model respects allowed number of transfers.

Data Collection

Data is sourced directly from the official Fantasy Premier League API:

```
url = "https://fantasy.premierleague.com/api/bootstrap-static/"
response = requests.get(url).json()

players_df = pd.DataFrame(response['elements'])         # All players
teams_df = pd.DataFrame(response['teams'])              # All teams
positions_df = pd.DataFrame(response['element_types'])  # Position info
```

The API data is clean and structured, requiring minimal preprocessing.

Player History Extraction

A function fetches detailed historical stats for each player across all gameweeks:

```
def get_player_history(player_id):
    url = f"https://fantasy.premierleague.com/api/element-summary/{player_id}/"
    r = requests.get(url).json()
    history_df = pd.DataFrame(r['history'])
    return history_df

```

We loop through all players, gather their histories, and concatenate into a single DataFrame:

```
  all_histories = []
for player_id in players_df['id']:
    df = get_player_history(player_id)
    df['player_id'] = player_id
    all_histories.append(df)

full_history_df = pd.concat(all_histories, ignore_index=True)
```

Finally, player metadata is merged for easier identification:

```
full_history_df = full_history_df.merge(players_df[['id', 'first_name', 'second_name', 'team_name',
'position']],
                                        left_on='player_id', right_on='id')
```

Data size: 27,605 rows × 50 columns.

Feature Engineering

We use **XGBoost** to predict player points for the next gameweek.

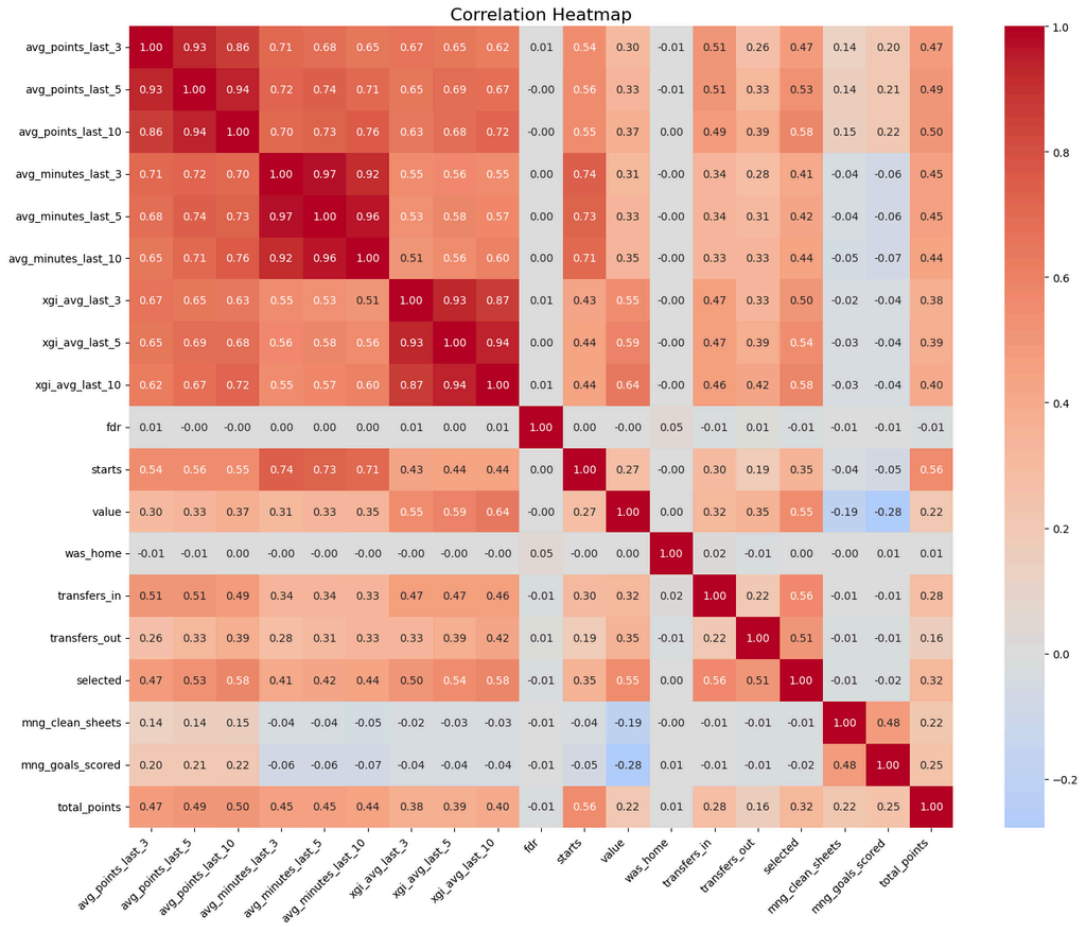
To enhance prediction, rolling averages for recent gameweeks are calculated for key features like points, minutes played, and expected goal involvements (xGI):

```
full_history_df = full_history_df.sort_values(by=["player_id", "round"])
full_history_df['points_shifted'] = full_history_df.groupby('player_id')['total_points'].shift(1)

full_history_df['avg_points_last_3'] = full_history_df.groupby('player_id')['points_shifted'].rolling(window=3,
min_periods=1).mean().reset_index(level=0, drop=True)
full_history_df['avg_points_last_5'] = full_history_df.groupby('player_id')['points_shifted'].rolling(window=5,
min_periods=1).mean().reset_index(level=0, drop=True)
full_history_df['avg_points_last_10'] = full_history_df.groupby('player_id')
['points_shifted'].rolling(window=10, min_periods=1).mean().reset_index(level=0, drop=True)
```

Similar rolling averages are created for minutes and xGI.

Feature Correlation Insights



- High correlation with points:
  - `starts` (0.56)
  - Rolling averages of points (last 3,5,10 gameweeks ~0.47–0.50)
  - Rolling averages of minutes (~0.44–0.45)
  - Rolling averages of xGI (~0.38–0.40)
  - `transfers_in` (0.28), `selected` (0.32)
- Low predictive value features:
  - `fdr` (-0.01), `was_home` (0.01), `transfers_out` (0.16)

Multicollinearity is not a concern due to XGBoost's robustness.

XGBoost Model Training

The dataset is split by gameweek for training and testing:

```
max_round_adj = fpl_data['round_adjusted'].max()
train_data = fpl_data[fpl_data['round_adjusted'] < max_round_adj]
test_data = fpl_data[fpl_data['round_adjusted'] == max_round_adj]

X_train = train_data[features]
y_train = train_data['total_points']

X_test = test_data[features]
y_test = test_data['total_points']
```

Model hyperparameters and training:

```
# Define model with some common hyperparameters
xgb_model = XGBRegressor(
    n_estimators=300,
    max_depth=2,
    learning_rate=0.03,
    subsample=0.6,
    colsample_bytree=0.6,
    reg_alpha=2,
    reg_lambda=3,
    gamma=1,
    random_state=42,
    n_jobs=-1
)

# Train the model
xgb_model.fit(X_train, y_train)
```

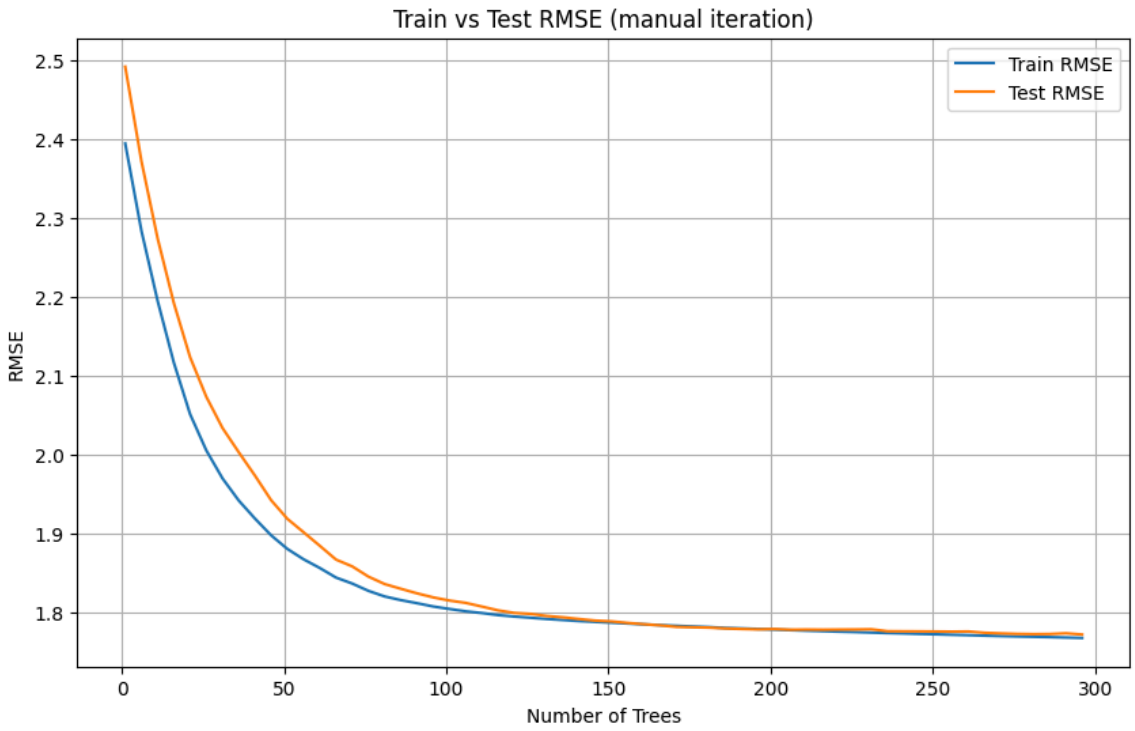
Model Performance:

- R^2 score: 0.476 (47.6% variance explained)
- Mean Squared Error (MSE): 3.335
- Mean Absolute Error (MAE): 0.912

The model shows moderate predictive power with an acceptable error margin, indicating it captures player point patterns reasonably well.

Overfitting Check

Training and testing errors are close, indicating the model generalizes well without overfitting.



From this gplot we see train RMSE is very near Train MSE this means

- The model is **not overfitting** (train error isn't way lower than test error).
- It's **generalizing well** to unseen data.
- The fit is balanced and stable.

Team Optimisation Methods

Predicting Next Gameweek Points

The model predicts each player's points for the upcoming gameweek:

```
fpl_data['next_gw_pred'] = xgb_model.predict(fpl_data[features])
```

Captain Selection

The captain is chosen as the player with the highest predicted points in the starting lineup:

```
    agg_preds = starters_next_gw.groupby('player_id', as_index=False)['next_gw_pred'].sum()

    # Find player with max aggregated predicted points
    best_captain_row = agg_preds.loc[agg_preds['next_gw_pred'].idxmax()]
```

Optimising Starting XI

Given a 15-player squad, players are first grouped by their playing position:

```
        players_by_pos = {'Goalkeeper':[], 'Defender':[], 'Midfielder':[], 'Forward':[]}
        for pid in squad_ids:
            pos = player_info[pid]['position']
            players_by_pos[pos].append(pid)
```

We then ensure there are two goalkeepers, iterating through each as a potential starter. The goalkeeper with the highest predicted points is selected as the starting keeper, and the other is assigned to the bench:

```
        gk_candidates = players_by_pos['Goalkeeper']
        if len(gk_candidates) < 2:
            return None, None

        for gk in gk_candidates:
            bench_gk = [pid for pid in gk_candidates if pid != gk][0]
```

Next, the function iterates through all valid Fantasy Premier League formations to find the optimal lineup. The standard formations considered are:

- 3-4-3
- 3-5-2
- 4-4-2
- 4-3-3
- 5-3-2
- 5-4-1

For each formation, the function generates all possible combinations of defenders, midfielders, and forwards that meet the formation requirements:

```
            for D_req, M_req, F_req in [
                (3, 4, 3), (3, 5, 2),
                (4, 4, 2), (4, 3, 3),
                (5, 3, 2), (5, 4, 1)
            ]:
                if len(def_candidates) < D_req or len(mid_candidates) < M_req or len(fwd_candidates) < F_req:
                    continue

                def_combos = combinations(def_candidates, D_req)
                mid_combos = combinations(mid_candidates, M_req)
                fwd_combos = combinations(fwd_candidates, F_req)
```

For each position, all possible groups of players of the required size are generated to fill the formation slots. The function will iterate through these to find valid squads.

For each valid combination, the lineup is constructed by combining the goalkeeper, defenders, midfielders, and forwards:

```
                mid_combos = list(mid_combos)
                fwd_combos = list(fwd_combos)

                for d_combo in def_combos:
                    for m_combo in mid_combos:
```

```
                        for f_combo in fwd_combos:
                            xi_ids = [gk] + list(d_combo) + list(m_combo) + list(f_combo)

                            team_counts = {}
                            valid = True
                            for pid in xi_ids:
                                team = player_info[pid]['team_name']
                                team_counts[team] = team_counts.get(team, 0) + 1
                                if team_counts[team] > 3:
                                    valid = False
                                    break
                            if not valid:
                                continue
```

This ensures no more than three players from the same real-life team are included in the starting XI.

Among all valid lineups, the one with the highest predicted points is selected as the optimal starting XI.

```
                            pts = total_pred_points(xi_ids)
                            if pts > best_pts:
                                best_pts = pts
                                best_xi = xi_ids
```

Transfer Recommendations

To make informed transfer suggestions, we employ a **Top-K heuristic** to manage computational complexity effectively:

1. **Identify the bottom performers**: We select the bottom **k=6** players from the current squad based on predicted points.
2. **Identify potential top players**: We find the top **k=20** players outside the squad with predicted points greater than 1.

Select the bottom performing players in the current squad based on their predicted points for the next gameweek. These are the prime candidates to be transferred out

```
    # Select the bottom topk_out players by predicted points from the current squad (worst performers)
    bottom_current = current_players_df.nsmallest(topk_out, 'next_gw_pred')['player_id'].tolist()
```

From the pool of all available players outside the current squad, select those with a predicted points value greater than 1 for the upcoming gameweek. From these, take the top performers as potential transfer-in options:

```
    # Potential players to bring in (outside current squad)
    potential_ins_all = set(unique_players['player_id']) - current_player_ids
    potential_ins_df = pred_next_gw[pred_next_gw['player_id'].isin(potential_ins_all)].copy()

    # Filter to include only those with predicted points > 1
    potential_ins_df = potential_ins_df[potential_ins_df['next_gw_pred'] > 1]

    # Select the top topk_in players by predicted points
    top_potential_ins = potential_ins_df.nlargest(topk_in, 'next_gw_pred')['player_id'].tolist()
```

For every number of transfers `k` from 0 up to the allowed number, all combinations of transferring out `k` players from the bottom candidates and transferring in `k` players from the top outside candidates are tested.

- When `k=0`, simply evaluate the current squad as is.
- For `k > 0`, generate all combinations of possible transfers and evaluate each candidate squad:

```
    for k in range(0, transfers_allowed + 1):
        # When k=0: no transfers, just check the current squad
        if k == 0:
            if valid_squad(current_player_ids):
                xi_ids, bench_ids = pick_best_starting_xi(current_player_ids)
                if xi_ids is not None:
                    total_pts = total_pred_points(xi_ids)
                    if total_pts > best_score:
                        best_score = total_pts
                        best_squad = current_player_ids
                        best_starting_xi = xi_ids
                        best_bench = bench_ids
                        best_out = []
                        best_in = []
            continue

        # For k > 0, consider combinations
        outs_combos = combinations(bottom_current, k)
        ins_combos = list(combinations(top_potential_ins, k))

        for out_ids in outs_combos:
            out_set = set(out_ids)
            remaining_players = set(current_squad_list) - out_set

            for in_ids in ins_combos:
                in_set = set(in_ids)

                candidate_squad = remaining_players | in_set

                if not valid_squad(candidate_squad):
                    continue

                xi_ids, bench_ids = pick_best_starting_xi(candidate_squad)
                if xi_ids is None:
                    continue

                total_pts = total_pred_points(xi_ids)  # Transfer penalty

                if total_pts > best_score:
                    best_score = total_pts
                    best_squad = candidate_squad
                    best_starting_xi = xi_ids
                    best_bench = bench_ids
                    best_out = out_ids
                    best_in = in_ids
```

This approach balances thoroughness and computational feasibility by focusing only on the worst-performing players in the current squad and the best available players outside it. By limiting transfer candidates to these Top-K groups, the algorithm efficiently searches for the optimal squad that maximizes predicted points for the next gameweek.

Example Findings

For Gameweek 25, with 3 transfers allowed and Matz Sels as captain:

- Initial predicted points: 38

- After optimisation:
  - Captain changed to Salah
  - Transfers in Minteh, Diaz, Marmoush
  - New predicted points: 87 (an increase of 48 points)

This demonstrates the effectiveness of the model and optimiser in improving team potential.