Football Player Data Analysis

Part 3

Oleksandr Solovei

Project Evolution

Previous Parts

- Part 1: Data Collection & Initial Analysis
- Part 2: Text Analysis & Historical Data

Part 3 Goals

- Advanced text processing & sentiment analysis
- Time series prediction for market values
- Player clustering and network visualization
- Market value trend prediction

Dataset

```
import pandas as pd
import re
import os
import urllib.parse
import json
import nltk
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# import spacy

from collections import Counter
```

```
from wordcloud import WordCloud

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

from unicodedata import normalize
```

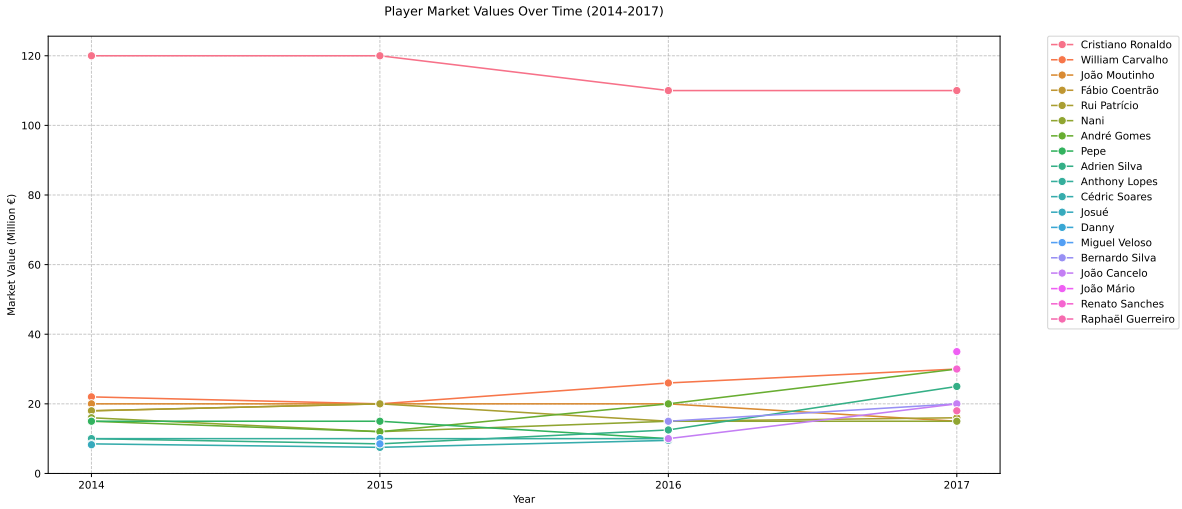
```
years = [2014, 2015, 2016, 2017]
data = list(map(lambda year: pd.read_csv(f'data/portugal_{year}_plus.csv'), years))
data[0].head()
```

	#	Player	Age	Market value	Name	Position	search_re
0	7	Cristiano Ronaldo Centre-Forward	30.0	120000000	Cristiano Ronaldo	CF	8631215
1	6	William Carvalho Defensive Midfield	23.0	22000000	William Carvalho	DM	2809567
2	8	João Moutinho Central Midfield	28.0	20000000	João Moutinho	CM	1431291
3	5	Fábio Coentrão Left-Back	27.0	18000000	Fábio Coentrão	LB	503646
4	12	Rui Patrício Goalkeeper	27.0	18000000	Rui Patrício	GK	1485432

Graphical Representation (Matplotlib)

```
def create_market_value_timeline(data_list, years):
    # Create a DataFrame for timeline plotting
    timeline_data = []

    for year, df in zip(years, data_list):
        year_data = df[['Name', 'Market value']].copy()
        year_data['Year'] = year
        year_data['Market value'] = year_data['Market value'].apply(lambda x: float(str(x).re
        year_data['Market value'] = year_data['Market value'] / 1000000
        timeline_data.append(year_data)

    timeline_df = pd.concat(timeline_data)

    plt.figure(figsize=(16, 7))
```

```
    # Create the line plot
    sns.lineplot(data=timeline_df,
                 x='Year',
                 y='Market value',
                 hue='Name',
                 marker='o',
                 markersize=8)

    # Set integer ticks on x-axis
    plt.xticks(years)

    plt.title('Player Market Values Over Time (2014-2017)', pad=20)
    plt.xlabel('Year')
    plt.ylabel('Market Value (Million €)')

    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.ylim(bottom=0)

    plt.tight_layout()
    return plt

# Create and display the timeline
plt = create_market_value_timeline(data, years)
plt.show()
```

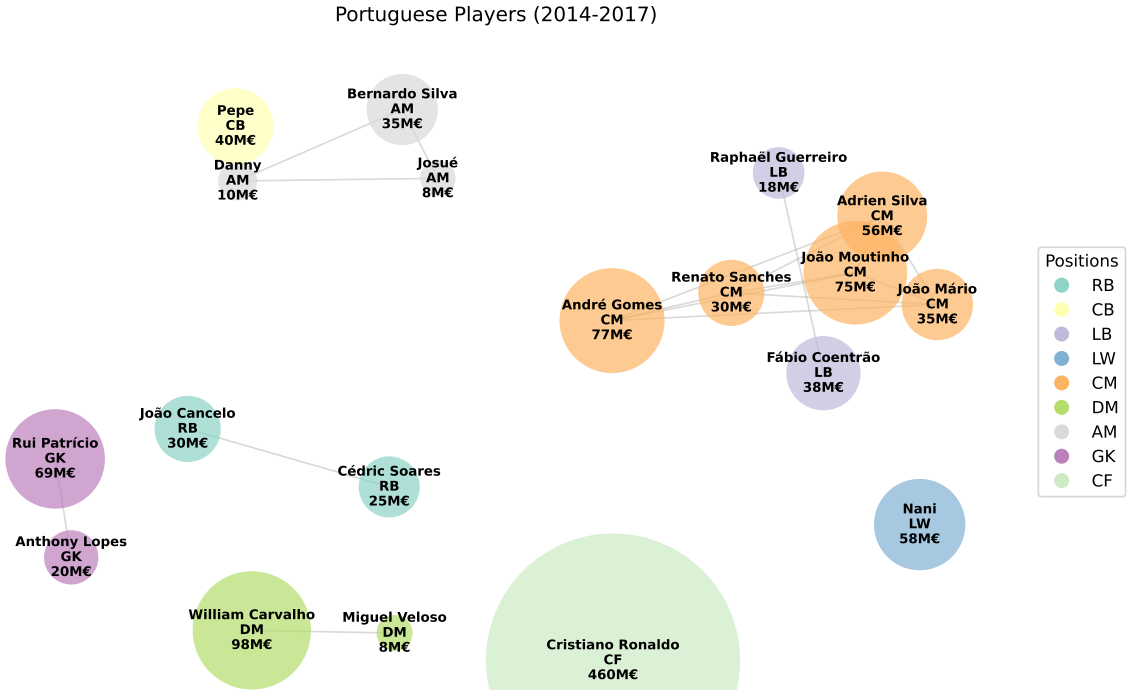


Graphical Representation (NetworkX)

```
merged_data = pd.concat(data)
merged_data = merged_data.groupby('Name').agg({'Market value': 'sum', 'Position': 'first', '
```

```
import networkx as nx
import matplotlib.pyplot as plt

def create_position_network(df):
    # Create a new graph
    G = nx.Graph()

    # Add nodes (players)
    for _, player in df.iterrows():
        # Convert market value to float, removing commas
        market_value = float(str(player['Market value']).replace(',', ''))
        G.add_node(player['Name'],
                   position=player['Position'],
                   market_value=market_value)

    # Connect players with same position
    players = list(G.nodes(data=True))
    for i in range(len(players)):
        for j in range(i + 1, len(players)):
            player1, data1 = players[i]
            player2, data2 = players[j]
            if data1['position'] == data2['position']:
                G.add_edge(player1, player2)

    # Create the visualization
    plt.figure(figsize=(10, 6))

    # Calculate node sizes based on market value (scaled for visibility)
    node_sizes = [G.nodes[player]['market_value']/1000000 * 50 for player in G.nodes()]

    # Create a color map based on positions
    unique_positions = list(set(nx.get_node_attributes(G, 'position').values()))
    color_map = {pos: plt.cm.Set3(i/len(unique_positions)) for i, pos in enumerate(unique_pos
    node_colors = [color_map[G.nodes[player]['position']] for player in G.nodes()]

    # Set up the layout
```

```
    pos = nx.spring_layout(G, k=1, iterations=50)

    # Draw the network
    nx.draw_networkx_nodes(G, pos,
                           node_size=node_sizes,
                           node_color=node_colors,
                           alpha=0.7)
    nx.draw_networkx_edges(G, pos,
                           edge_color='gray',
                           alpha=0.3)

    # Add labels
    labels = {player: f"{player}\n{G.nodes[player]['position']}\n{int(G.nodes[player]['marke
              for player in G.nodes()}

    nx.draw_networkx_labels(G, pos,
                            labels=labels,
                            font_size=8,
                            font_weight='bold')

    # Add legend for positions
    legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
                                  markerfacecolor=color, label=pos,
                                  markersize=10)
                       for pos, color in color_map.items()]
    plt.legend(handles=legend_elements, title='Positions',
              loc='center left', bbox_to_anchor=(1, 0.5))

    plt.title('Portuguese Players (2014-2017)', pad=20)
    plt.axis('off')
    plt.tight_layout()

    return plt

# Create and display the network using merged data, summing up market values of players with
plt = create_position_network(merged_data)
plt.show()
```




```
content_list = []

for year, df in zip(years, data):
    for i, row in df.iterrows():
        fname = urllib.parse.quote_plus(row['Player'])
        if (os.path.isfile(f'snapshots/{year}_{fname}.json')):
                with open(f'snapshots/{year}_{fname}.json','r') as f:
                    snapshot = f.read()
                    arquive_data = json.loads(snapshot)

                    # extract snippets from response_items and add them as a new column
                    # to the dataframe
                    for item in arquive_data['response_items']:
                        snippet_data = {
                            'Player': row['Player'],
                            'title': item.get('title', ''),
                            'url': item.get('originalURL', ''),
                            'snippet': item.get('snippet', '')
                        }
                        content_list.append(snippet_data)

        else:
```

```
            print(f"Snapshot not found for {row['Player']}")
            continue

content_df = pd.DataFrame(content_list)

# add preloaded HTML content

html_content = pd.read_csv('data/url_content_extracted.csv')

#join the column `extracted_text` to the content_df merging by url
content_df = content_df.merge(html_content[['url', 'extracted_text']], on='url', how='left')

# clean rows with missing content
content_df = content_df.dropna(subset=['extracted_text'])
```

Text Processing

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

def advanced_tokenization(text):
    lemmatizer = WordNetLemmatizer()
    if not isinstance(text, str):
        return []
    tokens = word_tokenize(text.lower())
    stop_words = set(stopwords.words('portuguese'))
    # Remove non-alphabetic and stopwords
    tokens = [lemmatizer.lemmatize(t) for t in tokens
              if t.isalpha() and t not in stop_words]
    return tokens

# Apply advanced tokenization to content_df
content_df['tokens'] = content_df['extracted_text'].apply(advanced_tokenization)
```

Word Clouds

```
# group tokens by player (merge arrays of strings)
tokens_by_player = content_df.groupby('Player')['tokens'].sum()

# create a word cloud for each player and show them in 3x4 grid
plt.figure(figsize=(16, 7))
for i, (player, tokens) in enumerate(tokens_by_player.items()):
    wordcloud = WordCloud(width=400, height=175, background_color='white').generate(' '.join
    plt.subplot(3, 4, i+1)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(player)
    plt.axis('off')

#add summary word cloud to the end
wordcloud = WordCloud(width=400, height=175, background_color='white').generate(' '.join(cont
plt.subplot(3, 4, 12)
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Summary')
plt.axis('off')

plt.tight_layout()
```


```
# FOOTBALL_KEYWORDS = [
#     'futebol', 'jogo', 'partida', 'pelada', 'clássico', 'gol', 'chute', 'passe', 'driblar'
# ]

# class FootballClassifier:
#     def __init__(self):
```

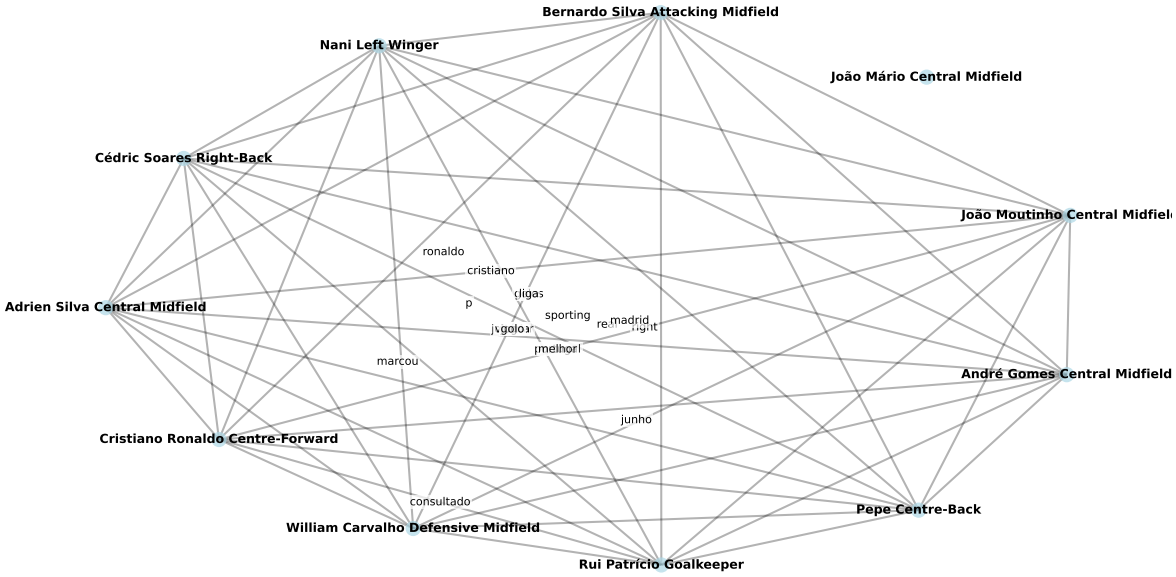
```
#         self.nlp = spacy.load('pt_core_news_sm')
#         self.football_keywords = FOOTBALL_KEYWORDS

#     def preprocess_text(self, text):
#         """
#         Preprocess the text by converting to lowercase and removing special characters
#         """
#         text = text.lower()
#         text = re.sub(r'[^a-záàâãéèêíïóôõöúçñ\s]', '', text)
#         return text

#     def analyze_text(self, text, threshold=0.03):
#         processed_text = self.preprocess_text(text)
#         doc = self.nlp(processed_text)

#         words = [token.lemma_ for token in doc if not token.is_stop and not token.is_punct

#         football_words = [word for word in words if word in self.football_keywords]
#         football_word_count = len(football_words)
#         total_words = len(words)

#         football_ratio = football_word_count / total_words if total_words > 0 else 0

#         return {
#             'is_football_related': football_ratio >= threshold,
#             'confidence_score': football_ratio,
#             'total_words': total_words,
#             'football_words_count': football_word_count
#         }
```

Connections Between Players by Shared Keywords

```
G = nx.Graph()

# Add nodes for players
for player in content_df['Player'].unique():
    G.add_node(player, type='player')

all_words = content_df['tokens'].sum()
most_common_words = [word for word, count in Counter(all_words).most_common(20)]
```

```
# show node between players if they share a word
for word in most_common_words:
    players_with_word = content_df[content_df['tokens'].apply(lambda tokens: word in tokens)]
    for i in range(len(players_with_word)):
        for j in range(i + 1, len(players_with_word)):
            G.add_edge(players_with_word[i], players_with_word[j], word=word)

# Create the visualization with wider aspect ratio
plt.figure(figsize=(12, 7))

# Set up the layout with adjusted parameters
pos = nx.spring_layout(G,
                       k=15,
                       iterations=200,
                       seed=126784)

# Draw the network
nx.draw_networkx_nodes(G, pos,
                       node_size=100,
                       node_color='lightblue',
                       alpha=0.7)

nx.draw_networkx_edges(G, pos,
                       alpha=0.3,
                       width=1.5)

# Add labels
labels = {node: node for node in G.nodes()}

nx.draw_networkx_labels(G, pos,
                        labels=labels,
                        font_size=9,
                        font_weight='bold')

# Add word labels with adjusted position
for word in most_common_words:
    players_with_word = content_df[content_df['tokens'].apply(lambda tokens: word in tokens)]
    if len(players_with_word) > 1:
        x = sum(pos[player][0] for player in players_with_word) / len(players_with_word)
        y = sum(pos[player][1] for player in players_with_word) / len(players_with_word)
        plt.text(x, y, word,
                fontsize=8,
```

```
                ha='center',
                va='center',
                bbox=dict(facecolor='white',
                          alpha=0.7,
                          edgecolor='none',
                          pad=1))

plt.axis('off')

plt.tight_layout()
plt.show()
```



Neural Network for Market Value Prediction

```
# create dataframe of player name, market value and tokens
player_data = content_df.groupby('Player').agg({'tokens': 'sum'}).reset_index()

# merge with market value data
player_data = player_data.merge(merged_data[['Player', 'Market value']], on='Player', how='le
player_data.head()
```

	Player	tokens	Market value
0	Adrien Silva Central Midfield	[sapo, início, início, atualidade, desporto, e...	56000000
1	André Gomes Central Midfield	[sapo, início, início, atualidade, desporto, e...	77000000
2	Bernardo Silva Attacking Midfield	[vésperas, estreia, rúben, amorim, treinador, ...	35000000
3	Cristiano Ronaldo Centre-Forward	[cristiano, ronaldo, página, apresenta, trecho...	460000000
4	Cédric Soares Right-Back	[confimação, saída, nuno, coelho, surge, possi...	25500000

Predictor

```python
class PlayerValuePredictor:
    def __init__(self):

        # Create pipeline with Portuguese-specific TF-IDF
        self.pipeline = Pipeline([
            ('tfidf', TfidfVectorizer(
                max_features=1000,
                ngram_range=(1, 2),
                min_df=2
            )),
            ('scaler', StandardScaler(with_mean=False)),
            ('model', RandomForestRegressor(
                n_estimators=500,
                max_depth=None,
                min_samples_split=3,
                min_samples_leaf=2,
                max_features='sqrt',
                random_state=126784
            ))
        ])

    def preprocess_portuguese_text(self, text):
        """Preprocess Portuguese text"""
        text = text.lower()
        text = normalize('NFKD', text).encode('ASCII', 'ignore').decode('ASCII')
        text = re.sub(r'[^a-zA-Z0-9\s]', ' ', text)
        text = ' '.join(text.split())
        return text

    def train(self, X_keywords, y_values, perform_grid_search=True):
        """Train the model with optional grid search"""
```

```
        # Join list of keywords into space-separated strings if needed
        if isinstance(X_keywords.iloc[0], list):
            X_keywords = X_keywords.apply(' '.join)

        # Log transform the target values if they're skewed
        y_log = np.log1p(y_values)

        # Split the data
        X_train, X_test, y_train, y_test = train_test_split(
            X_keywords, y_log, test_size=0.2, random_state=42
        )

        if perform_grid_search:
            # Define parameter grid
            param_grid = {
                'tfidf__max_features': [500, 1000, 2000],
                'tfidf__ngram_range': [(1, 1), (1, 2)],
                'model__n_estimators': [100, 200],
                'model__max_depth': [5, 10, 15],
                'model__min_samples_split': [2, 5, 10]
            }

            # Perform grid search
            grid_search = GridSearchCV(
                self.pipeline,
                param_grid,
                cv=5,
                scoring='neg_mean_squared_error',
                n_jobs=-1
            )
            grid_search.fit(X_train, y_train)
            self.pipeline = grid_search.best_estimator_
        else:
            # Train with default parameters
            self.pipeline.fit(X_train, y_train)

    def predict(self, keywords):
        """Predict market value for new keywords"""
        # Handle both single string and list of keywords
        if isinstance(keywords, list):
            keywords = ' '.join(keywords)

```

```
        # Convert to DataFrame series to match training format
        keywords_series = pd.Series([keywords])

        # Make prediction and transform back from log scale
        predicted_value_log = self.pipeline.predict(keywords_series)
        predicted_value = np.expm1(predicted_value_log)[0]

        return predicted_value
```

```
    predictor = PlayerValuePredictor()
    predictor.train(player_data['tokens'], player_data['Market value'])
```

Usage Example

Radio Cadena Voces @RCVHonduras

Portugal se impuso este sábado por 3-0 a Turquía y se clasificó para los octavos de final de la Eurocopa 2024 como primera del Grupo F, gracias a un gol de Bernardo Silva, otro de Samet Akaydin en propia puerta y un tanto de Bruno Fernández. #RCVNoticias

rcv.hn pic.x.com/wK0Vn2PLAK

```
text = 'Portugal se impuso este sábado por 3-0 a Turquía y se clasificó para los octavos de
new_keywords = predictor.preprocess_portuguese_text(text)
predicted_value = predictor.predict(new_keywords)
print(f"\nPredicted value for new player: ${predicted_value:,.2f}")
```

```
Predicted value for new player: $65,251,703.39
```

Conclusions

- Advanced text processing & sentiment analysis
- Visualizations of player connections
- Market value trend prediction

Future Research Directions

- Potential for real-time market value predictions
- Expansion to other football leagues and languages
- Integration with broader sports analytics systems

References

- Sozen, Y. (2023). Predicting Football Players Market Value Using Machine Learning
- Transfermarkt Documentation
- Arquivo.pt API Documentation