Fantasy Football Analysis Tool

This notebook provides comprehensive fantasy football analysis tools including:

- Data loading and cleaning
- Custom scoring calculations based on league configuration
- Position-based rankings and tiers
- Value-based drafting (VBD) calculations
- Comparison with expert rankings and ADP
- Draft simulation capabilities
- Interactive visualizations

Table of Contents

1. Setup and Data Loading

In [1]:
```python
# Import required libraries
import pandas as pd
import numpy as np
import yaml
import matplotlib.pyplot as plt
import seaborn as sns
from typing import Dict, List, Optional, Tuple, Union
import warnings
from pathlib import Path
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Configure display settings
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
pd.set_option('display.width', None)
warnings.filterwarnings('ignore')
```

```
# Set plotting style
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

print("Libraries imported successfully!")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent cal
l last)
Cell In[1], line 2
      1 # Import required libraries
----> 2 import pandas as pd
      3 import numpy as np
      4 import yaml

ModuleNotFoundError: No module named 'pandas'
```

In [2]:

```
class FantasyFootballAnalyzer:
    """Comprehensive fantasy football analysis tool."""

    def __init__(self, csv_path: str, config_path: str):
        """Initialize the analyzer with data and configuration.

        Args:
            csv_path: Path to the CSV file containing player data
            config_path: Path to the YAML configuration file
        """
        self.csv_path = Path(csv_path)
        self.config_path = Path(config_path)
        self.config = None
        self.raw_data = None
        self.clean_data = None
        self.scored_data = None

        # Load data and configuration
        self._load_config()
        self._load_data()
        self._clean_data()

    def _load_config(self) -> None:
        """Load league configuration from YAML file."""
        try:
            with open(self.config_path, 'r') as file:
                self.config = yaml.safe_load(file)
            print(f"✓ Configuration loaded: {self.config['league_na
            print(f"  - Teams: {self.config['basic_settings']['team
            print(f"  - Roster size: {self.config['roster']['total_
            print(f"  - Scoring: {self.config['basic_settings']['sc
        except Exception as e:
            raise ValueError(f"Error loading configuration: {e}")

    def _load_data(self) -> None:
        """Load player data from CSV file."""
        try:
            # The CSV has a complex header structure, so we'll read
            self.raw_data = pd.read_csv(self.csv_path, header=1, lo
            print(f"✓ Data loaded: {len(self.raw_data)} rows, {len(
        except Exception as e:
            raise ValueError(f"Error loading data: {e}")

    def _clean_data(self) -> None:
        """Clean and prepare the data for analysis."""
        df = self.raw_data.copy()

        # Filter out rows without player names
        df = df[df['Player'].notna() & (df['Player'] != '')].copy()

        # Key columns to keep and clean
        key_columns = [
            'Rank', 'Player', 'Position', 'Team', 'Bye', 'Age',
            'ESPN', 'NFL', 'Yahoo', 'ECR', 'Dynasty',
            'ESPN ADP', 'NFL ADP', 'Yahoo ADP', 'Sleeper ADP', 'FFC
            'Boris', 'BS Val', 'VBD', 'ECR Prj', 'Salary'
        ]

        # Keep only columns that exist in the data
        available_columns = [col for col in key_columns if col in d
        df = df[available_columns].copy()

        # Convert numeric columns
        numeric_columns = ['Rank', 'Age', 'ESPN', 'NFL', 'Yahoo', '
                           'ESPN ADP', 'NFL ADP', 'Yahoo ADP', 'Slee
                           'VBD', 'ECR Prj']

        for col in numeric_columns:
            if col in df.columns:
                df[col] = pd.to_numeric(df[col], errors='coerce')

        # Clean salary column if it exists
        if 'Salary' in df.columns:
            df['Salary'] = df['Salary'].astype(str).str.replace('$'
            df['Salary'] = pd.to_numeric(df['Salary'], errors='coer

        # Filter for main fantasy positions
        fantasy_positions = ['QB', 'RB', 'WR', 'TE', 'DEF', 'K']
        df = df[df['Position'].isin(fantasy_positions)].copy()

        # Add derived columns
        df['Bye'] = pd.to_numeric(df['Bye'], errors='coerce')
        df['Full_Name'] = df['Player'] + ' (' + df['Position'] + ',

        self.clean_data = df.reset_index(drop=True)

        print(f"✓ Data cleaned: {len(self.clean_data)} players acro
        print(f"  Position breakdown: {dict(df['Position'].value_co

    def calculate_custom_scoring(self, projections: Optional[Dict[s
        """Calculate custom fantasy points based on league scoring

        Args:
            projections: Optional custom projections. If None, uses

        Returns:
            DataFrame with custom fantasy points calculated
        """
        df = self.clean_data.copy()

        # Use ECR projections as base if no custom projections prov
        if projections is None:
            df['Custom_Points'] = df['ECR Prj']
        else:
            # Calculate custom points based on provided projections
            df['Custom_Points'] = 0.0

            for idx, row in df.iterrows():
                player_name = row['Player']
                position = row['Position']

                if player_name in projections:
                    proj = projections[player_name]
                    points = self._calculate_points_for_player(proj
                    df.loc[idx, 'Custom_Points'] = points

        # Calculate positional ranks
        for pos in df['Position'].unique():
            pos_mask = df['Position'] == pos
            df.loc[pos_mask, f'{pos}_Rank'] = df.loc[pos_mask, 'Cus

        self.scored_data = df
        return df

    def _calculate_points_for_player(self, projections: Dict, posit
        """Calculate fantasy points for a single player based on pr

        Args:
            projections: Dictionary containing player projections
            position: Player position

        Returns:
            Total fantasy points
        """
        scoring = self.config['scoring']
        points = 0.0

        # Passing stats (mainly QB)
        if 'pass_yds' in projections:
            points += projections['pass_yds'] * scoring['passing'][
        if 'pass_td' in projections:
            points += projections['pass_td'] * scoring['passing']['
        if 'pass_int' in projections:
            points += projections['pass_int'] * scoring['passing'][

        # Rushing stats
        if 'rush_yds' in projections:
            points += projections['rush_yds'] * scoring['rushing'][
        if 'rush_td' in projections:
            points += projections['rush_td'] * scoring['rushing']['

        # Receiving stats
        if 'rec_yds' in projections:
            points += projections['rec_yds'] * scoring['receiving']
        if 'rec_td' in projections:
            points += projections['rec_td'] * scoring['receiving'][

        # Miscellaneous
        if 'fumbles_lost' in projections:
            points += projections['fumbles_lost'] * scoring['miscel

        return points

    def calculate_vbd(self, method: str = 'baseline') -> pd.DataFra
        """Calculate Value-Based Drafting scores.

        Args:
            method: Method for VBD calculation ('baseline' or 'repl

        Returns:
            DataFrame with VBD scores
        """
        if self.scored_data is None:
            raise ValueError("Must calculate custom scoring first")

        df = self.scored_data.copy()

        # Get replacement levels from config
        replacement_levels = self.config['replacement_level']

        # Calculate VBD for each position
        for position in df['Position'].unique():
            if position not in replacement_levels:
                continue

            pos_data = df[df['Position'] == position].copy()
            pos_data = pos_data.sort_values('Custom_Points', ascend

            replacement_idx = min(replacement_levels[position] - 1,
            baseline_points = pos_data.iloc[replacement_idx]['Custo

            # Calculate VBD
            pos_mask = df['Position'] == position
            df.loc[pos_mask, 'VBD_Custom'] = df.loc[pos_mask, 'Cust

        # Fill NaN values with 0
        df['VBD_Custom'] = df['VBD_Custom'].fillna(0)

        # Calculate overall rank based on VBD
        df['VBD_Rank'] = df['VBD_Custom'].rank(method='dense', asce

        self.scored_data = df
        return df

    def get_position_rankings(self, position: str, top_n: int = 20)
```

```
        """Get top players at a specific position.

        Args:
            position: Position to rank (QB, RB, WR, TE, DEF, K)
            top_n: Number of top players to return

        Returns:
            DataFrame with top players at position
        """
        if self.scored_data is None:
            df = self.clean_data.copy()
            sort_col = 'ECR Prj'
        else:
            df = self.scored_data.copy()
            sort_col = 'Custom_Points'

        pos_data = df[df['Position'] == position].copy()
        pos_data = pos_data.sort_values(sort_col, ascending=False,

        return pos_data.head(top_n)

    def compare_rankings(self) -> pd.DataFrame:
        """Compare custom rankings with expert consensus and ADP.

        Returns:
            DataFrame with ranking comparisons
        """
        if self.scored_data is None:
            raise ValueError("Must calculate custom scoring first")

        df = self.scored_data.copy()

        # Calculate overall rankings
        df['Custom_Rank'] = df['Custom_Points'].rank(method='dense'

        # Calculate differences
        if 'ECR' in df.columns:
            df['ECR_Diff'] = df['ECR'] - df['Custom_Rank']
        if 'AVG ADP' in df.columns:
            df['ADP_Diff'] = df['AVG ADP'] - df['Custom_Rank']

        return df

    def simulate_draft(self, my_pick: int, strategy: str = 'vbd') -
        """Simulate a draft and recommend picks.

        Args:
            my_pick: Your draft position (1-14)
            strategy: Draft strategy ('vbd', 'adp', 'custom')

        Returns:
            List of recommended picks
        """
        if self.scored_data is None:
            raise ValueError("Must calculate custom scoring first")
        num_teams = self.config['basic_settings']['teams']
        roster_size = self.config['roster']['total_size']

        # Simple draft simulation
        available_players = self.scored_data.copy()

        if strategy == 'vbd':
            sort_col = 'VBD_Custom'
        elif strategy == 'adp':
            sort_col = 'AVG ADP'
            available_players = available_players.sort_values(sort_
        else:
            sort_col = 'Custom_Points'

        if strategy != 'adp':
            available_players = available_players.sort_values(sort_

        recommendations = []
        my_roster = {'QB': 0, 'RB': 0, 'WR': 0, 'TE': 0, 'DEF': 0,
        max_roster = self.config['roster']['maximums']

        for round_num in range(1, min(6, roster_size + 1)):  # Firs
            # Calculate pick number
            if round_num % 2 == 1:  # Odd rounds
                pick_num = (round_num - 1) * num_teams + my_pick
            else:  # Even rounds (snake)
                pick_num = round_num * num_teams - my_pick + 1

            # Find best available player considering roster needs
            for _, player in available_players.iterrows():
                pos = player['Position']
                if my_roster[pos] < max_roster[pos]:
                    recommendations.append(f"Round {round_num} (Pic
                    my_roster[pos] += 1
                    available_players = available_players.drop(play
                    break

        return recommendations

print("FantasyFootballAnalyzer class defined successfully!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 class FantasyFootballAnalyzer:
      2 
      4     def __init__(self, csv_path: str, config_path: str):

Cell In[2], line 90, in FantasyFootballAnalyzer()
     87     print(f"✓ Data cleaned: {len(self.clean_data)} players across {df['Position'].nunique()} positions")
     88     print(f"  Position breakdown: {dict(df['Position'].value_counts())}")
---> 90 def calculate_custom_scoring(self, projections: Optional[Dict[str, Dict]] = None) -> pd.DataFrame:
     91     """Calculate custom fantasy points based on league scoring settings.
     92 
     93     Args:
   (...)     97         DataFrame with custom fantasy points calculated
     98     """
     99     df = self.clean_data.copy()

NameError: name 'Optional' is not defined
```

In [3]:
```
# Initialize the analyzer
csv_path = "/Users/ben/projects/fantasy-football-draft-spreadsheet/
config_path = "/Users/ben/projects/fantasy-football-draft-spreadshe

analyzer = FantasyFootballAnalyzer(csv_path, config_path)
print("\n✓ Fantasy Football Analyzer initialized successfully!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 5
      2 csv_path = "/Users/ben/projects/fantasy-football-draft-spreadsheet/CSG Fantasy Football Sheet - 2025 v13.01.csv"
      3 config_path = "/Users/ben/projects/fantasy-football-draft-spreadsheet/config/league-config.yaml"
----> 5 analyzer = FantasyFootballAnalyzer(csv_path, config_path)
      6 print("\n✓ Fantasy Football Analyzer initialized successfully!")

NameError: name 'FantasyFootballAnalyzer' is not defined
```

2. Data Exploration

In [4]:
```
# Explore the loaded data
print("Dataset Overview:")
print(f"Shape: {analyzer.clean_data.shape}")
print(f"\nColumns: {list(analyzer.clean_data.columns)}")
print(f"\nPosition Distribution:")
print(analyzer.clean_data['Position'].value_counts())
```

```
Dataset Overview:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[4], line 3
      1 # Explore the loaded data
      2 print("Dataset Overview:")
----> 3 print(f"Shape: {analyzer.clean_data.shape}")
      4 print(f"\nColumns: {list(analyzer.clean_data.columns)}")
      5 print(f"\nPosition Distribution:")

NameError: name 'analyzer' is not defined
```

In [5]:
```
# Display sample data
print("Sample of clean data:")
display_columns = ['Player', 'Position', 'Team', 'Age', 'ECR', 'AVG
available_display_cols = [col for col in display_columns if col in
analyzer.clean_data[available_display_cols].head(10)
```

```
Sample of clean data:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[5], line 4
      2 print("Sample of clean data:")
      3 display_columns = ['Player', 'Position', 'Team', 'Age', 'EC
R', 'AVG ADP', 'ECR Prj']
----> 4 available_display_cols = [col for col in display_columns if
col in analyzer.clean_data.columns]
      5 analyzer.clean_data[available_display_cols].head(10)

NameError: name 'analyzer' is not defined
```

In [6]:
```
# Basic statistics
numeric_cols = analyzer.clean_data.select_dtypes(include=[np.number
print("Basic Statistics for Numeric Columns:")
analyzer.clean_data[numeric_cols].describe()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[6], line 2
      1 # Basic statistics
----> 2 numeric_cols = analyzer.clean_data.select_dtypes(include=[n
p.number]).columns
      3 print("Basic Statistics for Numeric Columns:")
      4 analyzer.clean_data[numeric_cols].describe()

NameError: name 'analyzer' is not defined
```

3. Custom Scoring System

In [7]:
```
# Calculate custom scoring (using ECR projections as base)
scored_data = analyzer.calculate_custom_scoring()
print("Custom scoring calculated!")
print(f"\nTop 10 players by projected points:")
top_10 = scored_data.nlargest(10, 'Custom_Points')[['Player', 'Posi
top_10
```

```
---------------------------------------------------------------------------
--------
NameError                                 Traceback (most recent cal
l last)
Cell In[7], line 2
      1 # Calculate custom scoring (using ECR projections as base)
----> 2 scored_data = analyzer.calculate_custom_scoring()
      3 print("Custom scoring calculated!")
      4 print(f"\nTop 10 players by projected points:")

NameError: name 'analyzer' is not defined
```

In [8]:
```
# Display league scoring settings
print("League Scoring Settings:")
print("="*50)

scoring = analyzer.config['scoring']

print("\nPassing:")
for stat, points in scoring['passing'].items():
    print(f"  {stat}: {points} points")

print("\nRushing:")
for stat, points in scoring['rushing'].items():
    print(f"  {stat}: {points} points")

print("\nReceiving:")
for stat, points in scoring['receiving'].items():
    print(f"  {stat}: {points} points")

print("\nMiscellaneous:")
for stat, points in scoring['miscellaneous'].items():
    print(f"  {stat}: {points} points")
```

```
League Scoring Settings:
==================================================
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 5
      2 print("League Scoring Settings:")
      3 print("="*50)
----> 5 scoring = analyzer.config['scoring']
      7 print("\nPassing:")
      8 for stat, points in scoring['passing'].items():

NameError: name 'analyzer' is not defined
```

4. Position Analysis

In [9]:
```
# Analyze each position
positions = ['QB', 'RB', 'WR', 'TE']

for position in positions:
    print(f"\n{position} Rankings (Top 10):")
    print("="*40)

    pos_rankings = analyzer.get_position_rankings(position, 10)
    display_cols = ['Player', 'Team', 'Custom_Points', 'ECR', 'AVG 
    available_cols = [col for col in display_cols if col in pos_ran

    if not pos_rankings.empty:
        display(pos_rankings[available_cols].reset_index(drop=True)
    else:
        print(f"No data available for {position}")
```

```
QB Rankings (Top 10):
========================================
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 8
      5 print(f"\n{position} Rankings (Top 10):")
      6 print("="*40)
----> 8 pos_rankings = analyzer.get_position_rankings(position, 10)
      9 display_cols = ['Player', 'Team', 'Custom_Points', 'ECR', 'AVG ADP']
     10 available_cols = [col for col in display_cols if col in pos_rankings.columns]

NameError: name 'analyzer' is not defined
```

5. Value-Based Drafting

In [10]:
```
# Calculate VBD scores
vbd_data = analyzer.calculate_vbd()
```

```
print("Value-Based Drafting scores calculated!")

# Show replacement levels
print("\nReplacement Levels:")
for pos, level in analyzer.config['replacement_level'].items():
    print(f"{pos}: {level} players")

print("\nTop 20 players by VBD:")
top_vbd = vbd_data.nlargest(20, 'VBD_Custom')[['Player', 'Position'
top_vbd.reset_index(drop=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[10], line 2
      1 # Calculate VBD scores
----> 2 vbd_data = analyzer.calculate_vbd()
      3 print("Value-Based Drafting scores calculated!")
      5 # Show replacement levels

NameError: name 'analyzer' is not defined
```

In [11]:
```
# VBD by position
print("VBD Leaders by Position:")
print("="*50)

for position in positions:
    pos_data = vbd_data[vbd_data['Position'] == position].nlargest(
    if not pos_data.empty:
        print(f"\n{position} (Top 5):")
        for idx, player in pos_data.iterrows():
            print(f"  {player['Player']} ({player['Team']}): {playe
```

```
VBD Leaders by Position:
==================================================
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[11], line 6
      3 print("="*50)
      5 for position in positions:
----> 6     pos_data = vbd_data[vbd_data['Position'] == position].nl
argest(5, 'VBD_Custom')
      7     if not pos_data.empty:
      8         print(f"\n{position} (Top 5):")

NameError: name 'vbd_data' is not defined
```

6. Rankings Comparison

In [12]:
```
# Compare rankings
comparison_data = analyzer.compare_rankings()
```

```
print("Rankings Comparison (Top 30):")
print("Note: Positive differences mean player is ranked higher in o

comparison_cols = ['Player', 'Position', 'Custom_Rank', 'ECR', 'ECR
available_comparison_cols = [col for col in comparison_cols if col

top_comparison = comparison_data.nsmallest(30, 'Custom_Rank')[avail
top_comparison.reset_index(drop=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[12], line 2
      1 # Compare rankings
----> 2 comparison_data = analyzer.compare_rankings()
      4 print("Rankings Comparison (Top 30):")
      5 print("Note: Positive differences mean player is ranked high
er in our system")

NameError: name 'analyzer' is not defined
```

In [13]:
```
# Find the biggest value and reach players
if 'ECR_Diff' in comparison_data.columns:
    print("\nBiggest VALUES (players ranked much higher in our syst
    values = comparison_data.nlargest(10, 'ECR_Diff')[['Player', 'P
    display(values.reset_index(drop=True))

    print("\nBiggest REACHES (players ranked much lower in our syst
    reaches = comparison_data.nsmallest(10, 'ECR_Diff')[['Player',
    display(reaches.reset_index(drop=True))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[13], line 2
      1 # Find the biggest value and reach players
----> 2 if 'ECR_Diff' in comparison_data.columns:
      3     print("\nBiggest VALUES (players ranked much higher in o
ur system):")
      4     values = comparison_data.nlargest(10, 'ECR_Diff')[['Play
er', 'Position', 'Custom_Rank', 'ECR', 'ECR_Diff']]

NameError: name 'comparison_data' is not defined
```

7. Draft Simulation

In [14]:
```
# Draft simulation function
def interactive_draft_simulation(draft_position: int = 7):
    """Run an interactive draft simulation."""
    print(f"\nDraft Simulation - Your Position: {draft_position}")
    print("="*50)

    strategies = ['vbd', 'adp', 'custom']
    for strategy in strategies:
        print(f"\n{strategy.upper()} Strategy:")
        print("-" * 20)

        try:
            recommendations = analyzer.simulate_draft(draft_positio
            for rec in recommendations:
                print(rec)
        except Exception as e:
            print(f"Error in {strategy} simulation: {e}")

# Run draft simulation for different positions
draft_positions = [1, 7, 14]  # Early, middle, late picks

for pos in draft_positions:
    interactive_draft_simulation(pos)
```

```
Draft Simulation - Your Position: 1
==================================================

VBD Strategy:
--------------------
Error in vbd simulation: name 'analyzer' is not defined

ADP Strategy:
--------------------
Error in adp simulation: name 'analyzer' is not defined

CUSTOM Strategy:
--------------------
Error in custom simulation: name 'analyzer' is not defined

Draft Simulation - Your Position: 7
==================================================

VBD Strategy:
--------------------
Error in vbd simulation: name 'analyzer' is not defined

ADP Strategy:
--------------------
Error in adp simulation: name 'analyzer' is not defined

CUSTOM Strategy:
--------------------
Error in custom simulation: name 'analyzer' is not defined

Draft Simulation - Your Position: 14
==================================================

VBD Strategy:
--------------------
Error in vbd simulation: name 'analyzer' is not defined

ADP Strategy:
--------------------
Error in adp simulation: name 'analyzer' is not defined

CUSTOM Strategy:
--------------------
Error in custom simulation: name 'analyzer' is not defined
```

8. Visualizations

In [15]:
```python
# Position distribution visualization
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Fantasy Football Data Analysis', fontsize=16, fontwei

# Position counts
analyzer.clean_data['Position'].value_counts().plot(kind='bar', ax=
axes[0,0].set_title('Players by Position')
axes[0,0].set_xlabel('Position')
```

```
axes[0,0].set_ylabel('Count')
axes[0,0].tick_params(axis='x', rotation=45)

# Age distribution
if 'Age' in analyzer.clean_data.columns:
    analyzer.clean_data['Age'].hist(bins=20, ax=axes[0,1], color='l
    axes[0,1].set_title('Age Distribution')
    axes[0,1].set_xlabel('Age')
    axes[0,1].set_ylabel('Count')

# Projected points by position
if 'Custom_Points' in vbd_data.columns:
    position_points = vbd_data.groupby('Position')['Custom_Points']
    position_points.plot(kind='bar', ax=axes[1,0], color='lightgree
    axes[1,0].set_title('Average Projected Points by Position')
    axes[1,0].set_xlabel('Position')
    axes[1,0].set_ylabel('Average Points')
    axes[1,0].tick_params(axis='x', rotation=45)

# VBD distribution
if 'VBD_Custom' in vbd_data.columns:
    vbd_data['VBD_Custom'].hist(bins=30, ax=axes[1,1], color='gold'
    axes[1,1].set_title('VBD Score Distribution')
    axes[1,1].set_xlabel('VBD Score')
    axes[1,1].set_ylabel('Count')

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[15], line 2
      1 # Position distribution visualization
----> 2 fig, axes = plt.subplots(2, 2, figsize=(15, 10))
      3 fig.suptitle('Fantasy Football Data Analysis', fontsize=16,
fontweight='bold')
      5 # Position counts

NameError: name 'plt' is not defined
```

In [16]:

```
# Interactive plotly visualizations
def create_interactive_plots():
    """Create interactive plots using plotly."""

    # VBD vs ADP scatter plot
    if all(col in vbd_data.columns for col in ['VBD_Custom', 'AVG A
        fig1 = px.scatter(vbd_data.dropna(subset=['VBD_Custom', 'AV
                         x='AVG ADP',
                         y='VBD_Custom',
                         color='Position',
                         hover_data=['Player', 'Team'],
                         title='VBD Score vs Average ADP',
                         labels={'VBD_Custom': 'VBD Score', 'AVG AD
        fig1.show()
    # Position-wise box plots
    if 'Custom_Points' in vbd_data.columns:
        fig2 = px.box(vbd_data,
                     x='Position',
                     y='Custom_Points',
                     title='Projected Points Distribution by Positi
        fig2.show()

    # Top players by position
    top_by_pos = []
    for pos in positions:
        pos_top = analyzer.get_position_rankings(pos, 10)
        top_by_pos.append(pos_top)

    if top_by_pos:
        all_top = pd.concat(top_by_pos, ignore_index=True)

        if 'Custom_Points' in all_top.columns:
            fig3 = px.bar(all_top,
                         x='Player',
                         y='Custom_Points',
                         color='Position',
                         title='Top Players by Projected Points',
                         hover_data=['Team'])
            fig3.update_xaxes(tickangle=45)
            fig3.show()

create_interactive_plots()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Cell In[16], line 43
     40             fig3.update_xaxes(tickangle=45)
     41             fig3.show()
---> 43 create_interactive_plots()

Cell In[16], line 6, in create_interactive_plots()
      3 """Create interactive plots using plotly."""
      5 # VBD vs ADP scatter plot
----> 6 if all(col in vbd_data.columns for col in [                ,
        ]):
      7     fig1 = px.scatter(vbd_data.dropna(subset=['VBD_Custom',
'AVG ADP']),
      8                       x='AVG ADP',
      9                       y='VBD_Custom',
   (...)     12                       title='VBD Score vs Average AD
P',
     13                       labels={'VBD_Custom': 'VBD Score', 'AVG
ADP': 'Average ADP'})
     14     fig1.show()

Cell In[16], line 6, in <genexpr>(.0)
      3 """Create interactive plots using plotly."""
      5 # VBD vs ADP scatter plot
----> 6 if all(col in vbd_data.columns for col in ['VBD_Custom', 'AV
G ADP']):
      7     fig1 = px.scatter(vbd_data.dropna(subset=['VBD_Custom',
'AVG ADP']),
      8                       x='AVG ADP',
      9                       y='VBD_Custom',
   (...)     12                       title='VBD Score vs Average AD
P',
     13                       labels={'VBD_Custom': 'VBD Score', 'AVG
ADP': 'Average ADP'})
     14     fig1.show()

NameError: name 'vbd_data' is not defined
```

9. Advanced Analysis Functions

In [17]:
```python
def analyze_bye_weeks():
    """Analyze bye week distribution."""
    if 'Bye' in analyzer.clean_data.columns:
        print("Bye Week Analysis:")
        print("="*30)

        bye_analysis = analyzer.clean_data.groupby(['Bye', 'Positio
        print(bye_analysis)

        # Plot bye week distribution
        plt.figure(figsize=(12, 6))
        bye_analysis.plot(kind='bar', stacked=True)
        plt.title('Players by Bye Week and Position')
        plt.xlabel('Bye Week')
        plt.ylabel('Number of Players')
        plt.xticks(rotation=0)
        plt.legend(title='Position', bbox_to_anchor=(1.05, 1), loc=
        plt.tight_layout()
        plt.show()

def find_sleepers_and_busts():
    """Find potential sleepers and busts based on ranking differenc
    if all(col in comparison_data.columns for col in ['ECR_Diff', '
        # Sleepers: Low ADP but high in our rankings
        sleeper_threshold = 20  # Players drafted 20+ spots later t
        sleepers = comparison_data[
            (comparison_data['ADP_Diff'] > sleeper_threshold) &
            (comparison_data['Custom_Rank'] <= 100)
        ].sort_values('ADP_Diff', ascending=False)

        print("\nPotential SLEEPERS (High value, low ADP):")
        sleeper_cols = ['Player', 'Position', 'Custom_Rank', 'AVG A
        available_sleeper_cols = [col for col in sleeper_cols if co
        display(sleepers[available_sleeper_cols].head(10))

        # Busts: High ADP but low in our rankings
        bust_threshold = -20  # Players drafted 20+ spots earlier t
        busts = comparison_data[
            (comparison_data['ADP_Diff'] < bust_threshold) &
            (comparison_data['AVG ADP'] <= 100)
        ].sort_values('ADP_Diff', ascending=True)

        print("\nPotential BUSTS (Low value, high ADP):")
        bust_cols = ['Player', 'Position', 'Custom_Rank', 'AVG ADP'
        available_bust_cols = [col for col in bust_cols if col in b
        display(busts[available_bust_cols].head(10))

def create_cheat_sheet():
    """Create a draft cheat sheet."""
    print("\nDRAFT CHEAT SHEET")
    print("="*50)

    cheat_sheet = vbd_data.nlargest(50, 'VBD_Custom')[[
        'Player', 'Position', 'Team', 'Bye', 'Custom_Points', 'VBD_
    ]].copy()

    cheat_sheet['Draft_Rank'] = range(1, len(cheat_sheet) + 1)
    cheat_sheet = cheat_sheet[['Draft_Rank', 'Player', 'Position',

    available_cheat_cols = [col for col in cheat_sheet.columns if c
    return cheat_sheet[available_cheat_cols]

# Run advanced analyses
analyze_bye_weeks()
find_sleepers_and_busts()

print("\n" + "="*60)
```

```
cheat_sheet = create_cheat_sheet()
display(cheat_sheet)
```

```
---------------------------------------------------------------------------
--------
NameError                                 Traceback (most recent cal
l last)
Cell In[17], line 64
     61     return cheat_sheet[available_cheat_cols]
     63 # Run advanced analyses
---> 64 analyze_bye_weeks()
     65 find_sleepers_and_busts()
     67 print("\n" + "="*60)

Cell In[17], line 3, in analyze_bye_weeks()
      1 def analyze_bye_weeks():
      2     """Analyze bye week distribution."""
----> 3     if 'Bye' in analyzer.clean_data.columns:
      4         print("Bye Week Analysis:")
      5         print("="*30)

NameError: name 'analyzer' is not defined
```

10. Export and Save Results

In [18]:

```
def export_results():
    """Export analysis results to CSV files."""

    # Create exports directory
    export_dir = Path('/Users/ben/projects/fantasy-football-draft-s
    export_dir.mkdir(exist_ok=True)

    # Export cheat sheet
    cheat_sheet_path = export_dir / 'draft_cheat_sheet.csv'
    cheat_sheet.to_csv(cheat_sheet_path, index=False)
    print(f"✓ Cheat sheet exported to: {cheat_sheet_path}")

    # Export full analysis
    full_analysis_path = export_dir / 'full_player_analysis.csv'
    vbd_data.to_csv(full_analysis_path, index=False)
    print(f"✓ Full analysis exported to: {full_analysis_path}")

    # Export position rankings
    for position in positions:
        pos_rankings = analyzer.get_position_rankings(position, 30)
        if not pos_rankings.empty:
            pos_path = export_dir / f'{position.lower()}_rankings.c
            pos_rankings.to_csv(pos_path, index=False)
            print(f"✓ {position} rankings exported to: {pos_path}")

    print(f"\nAll exports completed in: {export_dir}")

# Export results
export_results()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[18], line 29
     26     print(f"\nAll exports completed in: {export_dir}")
     28 # Export results
---> 29 export_results()

Cell In[18], line 5, in export_results()
      2 """Export analysis results to CSV files."""
      4 # Create exports directory
----> 5 export_dir = Path('/Users/ben/projects/fantasy-football-draft-spreadsheet/exports')
      6 export_dir.mkdir(exist_ok=True)
      8 # Export cheat sheet

NameError: name 'Path' is not defined
```

Summary and Key Insights

This notebook provides a comprehensive fantasy football analysis tool that:

Key Features:

1. **Data Integration**: Loads and cleans player data from CSV files
2. **Custom Scoring**: Applies league-specific scoring rules from YAML configuration
3. **Value-Based Drafting**: Calculates VBD scores using configurable replacement levels
4. **Rankings Comparison**: Compares custom rankings with expert consensus and ADP
5. **Draft Simulation**: Provides draft recommendations based on different strategies
6. **Advanced Analytics**: Identifies sleepers, busts, and bye week considerations
7. **Visualizations**: Interactive charts for data exploration
8. **Export Capabilities**: Saves results as CSV files for external use

Next Steps:

1. **Update Projections**: Replace ECR projections with custom projections for more accurate scoring
2. **Tier Analysis**: Implement Boris Chen-style tier groupings
3. **Auction Values**: Add auction draft value calculations
4. **Trade Analysis**: Build tools for evaluating trades
5. **Weekly Updates**: Create functions to update player values throughout

the season

Usage:

- Modify the `config/league-config.yaml` file to match your league settings
- Update the CSV data source as new projections become available
- Run the notebook before your draft to generate updated rankings and cheat sheets
- Use the draft simulation to practice different strategies

Happy drafting! 🏈

In [ ]:

In [ ]: