Name: Emmanuel Adebayo

Date: Thursday, September 18, 2025

Introduction

The purpose of this project is to gauge your technical skills and problem solving ability by working through something similar to a real NBA data science project. You will work your way through this R Markdown document, answering questions as you go along. Please begin by adding your name to the "author" key in the YAML header. When you're finished with the document, come back and type your answers into the answer key at the top. Please leave all your work below and have your answers where indicated below as well. Please note that we will be reviewing your code so make it clear, concise, and **avoid long printouts.** Feel free to add in as many new code chunks as you'd like.

Remember that we will be grading the quality of your code and visuals alongside the correctness of your answers. Please try to use the tidyverse as much as possible (instead of base R and explicit loops). Please do not bring in any outside data, and use the provided data as truth (for example, some "home" games have been played at secondary locations, including TOR's entire 2020-21 season. These are not reflected in the data and you do not need to account for this.) Note that the OKC and DEN 2024-25 schedules in `schedule_24_partial.csv` intentionally include only 80 games, as the league holds 2 games out for each team in the middle of December due to unknown NBA Cup matchups. Do not assign specific games to fill those two slots.

Note:

Throughout this document, any `season` column represents the year each season started. For example, the 2015-16 season will be in the dataset as 2015. We may refer to a season by just this number (e.g. 2015) instead of the full text (e.g. 2015-16).

Answers

Part 1

Question 1: 26 4-in-6 stretches in OKC's draft schedule.

Question 2: 25.10 4-in-6 stretches on average.

Question 3:

- Most 4-in-6 stretches on average: CHA (28.11)
- Fewest 4-in-6 stretches on average: NYK (22.19)

Question 4: This is a written question. Please leave your response in the document under Question 4.

Question 5:

- BKN Defensive eFG%: 0.54%
- When opponent on a B2B: 0.53%

Part 2

Please show your work in the document, you don't need anything here.

Part 3

Question 8:

- Most Helped by Schedule: UTA (5 wins)
- Most Hurt by Schedule: TOR (-4 wins)

Setup and Data

In [32]:
```
'''
This block is responsible for pre loading all necessaries tools
'''
import matplotlib.pyplot as plt
from geopy.distance import geodesic
import plotly.graph_objects as go

import pandas as pd
import numpy as np
from sklearn.utils import shuffle

from IPython.display import HTML

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import BinaryAccuracy


from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    accuracy_score, precision_score, recall_score, f1_score
)
```

In [4]:
```
#These are utility/helper functions
#Note: Some of these functions might have been removed in the final submissions as some

'''
Used to check frame consitency
'''
def check_if_frame_has_nulls_and_duplicates(df: pd.DataFrame) -> str:
    #Check for null values and duplicate rows in a DataFrame.

    duplicates = df.duplicated().sum()
    nulls = df.isnull().sum()

    if duplicates == 0 and nulls.sum() == 0:
        return f"No nulls or duplicate rows found. Data has {df.shape} shape"

    issues = []
    if duplicates > 0:
        issues.append(f"{duplicates} duplicate row(s)")
    if nulls.sum() > 0:
        bad_cols = {col: int(count) for col, count in nulls.items() if count > 0}
        issues.append(f"Nulls found in: {bad_cols}")

    return " | ".join(issues)

'''
returns a copy of frame
'''
def get_copy_of_a_data_frame(data: pd.DataFrame) -> pd.DataFrame:
    return data.copy()

'''
returns  CSVs
'''
def get_locations() ->pd.DataFrame:
```

```
    locations = pd.read_csv("locations.csv")
    return locations

def get_draft_schedule() -> pd.DataFrame:
    draft_schedule = pd.read_csv("schedule_24_partial.csv")
    draft_schedule['gamedate'] = pd.to_datetime(draft_schedule['gamedate'])
    return draft_schedule

def get_schedule() -> pd.DataFrame:
    schedule = pd.read_csv("schedule.csv")
    schedule['gamedate'] = pd.to_datetime(schedule['gamedate'])
    return schedule

def get_game_data() -> pd.DataFrame:
    game_data = pd.read_csv("team_game_data.csv")
    game_data['gamedate'] = pd.to_datetime(game_data['gamedate'])
    return game_data

'''
returns the shape of frame
'''
def get_data_shape(data: pd.DataFrame)-> tuple[int, int]:
    return data.shape

'''
takes two frame as argument. Used to check if any frame transformation has caused issues
'''
def check_if_frame_has_same_shape(frame1: pd.DataFrame, frame2: pd.DataFrame) -> str:
    return "Data shape matches" if frame1.shape == frame2.shape else "Shape does not mat

'''
returns a sorted frame
'''
def sort_data_frame(data: pd.DataFrame, by: str | list[str], ascending: bool = True) ->
    return data.sort_values(by=by, ascending=ascending).reset_index(drop=True)

'''
sort a frame in place
'''
def sort_data_frame_in_place(data: pd.DataFrame, by : str | list[str], ascending: bool =
    data.sort_values(by=by, ascending=ascending, inplace=True)
    data.reset_index(drop=True, inplace=True)

'''
Takes a frame as argument and returns the list of seasons
'''
def get_all_seasons(schedule: pd.DataFrame) -> list[int]:
    return sorted(schedule['season'].unique().tolist())
'''
Takes a frame as argument and returns the list of teams
'''
def get_all_teams(schedule: pd.DataFrame) -> list[str]:
    return sorted(schedule['team'].unique().tolist())
```

In [5]:
```
#Enusuring the data is in the right shape and has no duplicate
print(f" ================== validating schedule ================== \n{check_if_frame_h
print(f" ================== validating draft_schedule ================== \n{check_if_f
print(f" ================== validating locations ================== \n{check_if_frame_
print(f" ================== validating game_data ================== \n{check_if_frame_
```

```
 ================== validating schedule ==================
No nulls or duplicate rows found. Data has (23958, 6) shape
 ================== validating draft_schedule ==================
No nulls or duplicate rows found. Data has (160, 6) shape
 ================== validating locations ==================
No nulls or duplicate rows found. Data has (30, 4) shape
 ================== validating game_data ==================
No nulls or duplicate rows found. Data has (23958, 41) shape
```

Part 1 -- Schedule Analysis

In this section, you're going to work to answer questions using NBA scheduling data.

Question 1

QUESTION: How many times are the Thunder scheduled to play 4 games in 6 nights in the provided 80-game draft of the 2024-25 season schedule? (Note: clarification, the stretches can overlap, the question is really "How many games are the 4th game played over the past 6 nights?")

In [6]:
```
den_okc_2024 = get_draft_schedule()
df_okc =  get_copy_of_a_data_frame (den_okc_2024[ den_okc_2024['team'] == 'OKC'] )
del den_okc_2024
print(f" ================== OKC 2024 Season Schedule ================== \n{check_if_fr
df_okc.head()
```

```
 ================== OKC 2024 Season Schedule ==================
No nulls or duplicate rows found. Data has (80, 6) shape
```

Out[6]:

	season	gamedate	team	opponent	home	win
1	2024	2025-04-13	OKC	NOP	0	1
2	2024	2025-04-11	OKC	UTA	0	1
4	2024	2025-04-09	OKC	PHX	0	1
6	2024	2025-04-08	OKC	LAL	1	1
8	2024	2025-04-06	OKC	LAL	1	0

In [7]:
```
'''
Function block is responsible of counting 4 in 6 stretches
'''
def count_exact_4_in_6_stretches(data: pd.DataFrame) -> tuple[int, list]:
    sort_data_frame_in_place(data=data, by = 'gamedate')
    count = 0
    fourth_games = []

    for i in range(3, len(data)):
        window = data.iloc[i-3:i+1]

        if (window['gamedate'].max() - window['gamedate'].min()).days <= 5:
            count += 1
            fourth_games.append(window['gamedate'].iloc[-1])

    return count, fourth_games

count_of_4_games_in_6_nights, _ = count_exact_4_in_6_stretches(data=df_okc)

print( f"OKC Thunder are Scheduled to play the 4th games in 6 nights {count_of_4_games_i
```

```
OKC Thunder are Scheduled to play the 4th games in 6 nights 26 times in the 2024_25 seaso
n
```

ANSWER 1:

26 4-in-6 stretches in OKC's draft schedule.

Question 2

QUESTION: From 2014-15 to 2023-24, what is the average number of 4-in-6 stretches for a team in a season? Adjust each team/season to per-82 games before taking your final average.

In [8]:
```
'''
store the analyses of team, season, games_played, and adjusted per 82
converted to a frame later
'''
table_calculation = []


'''
calculates the raw count of 4 in 6, games played, adjusted per 82, and list of the exact
'''
def analyze_4_in_6_game_stretches_across_seasons_and_teams(schedule : pd.DataFrame, szn:
```

```
    season_and_team_frame = get_copy_of_a_data_frame( schedule[(schedule['season'] == sz

    count, fourth_games = count_exact_4_in_6_stretches(season_and_team_frame)
    games_played = len(season_and_team_frame)
    adjusted_per_82 = (count / games_played) * 82 if games_played > 0 else 0

    return count, games_played, adjusted_per_82, fourth_games


'''
Summarize the computed statistics across seasons and teams
'''
def summarize_4_in_6_across_season_per_team(table_calculation: pd.DataFrame):

    max_value = table_calculation['adjusted_per_season'].max()
    max_rows = table_calculation[table_calculation['adjusted_per_season'] == max_value]

    highest_results = []
    for _, row in max_rows.iterrows():
        highest_results.append(
            f"Highest: {row['team']} in season {row['season']} with {row['adjusted_per_s
        )
    highest_4_in_6_overall = " | ".join(highest_results)

    min_value = table_calculation['adjusted_per_season'].min()
    min_rows = table_calculation[table_calculation['adjusted_per_season'] == min_value]

    lowest_results = []
    for _, row in min_rows.iterrows():
        lowest_results.append(
            f"Lowest: {row['team']} in season {row['season']} with {row['adjusted_per_se
        )
    lowest_4_in_6_overall = " | ".join(lowest_results)

    team_averages = table_calculation.groupby('team')['adjusted_per_season'].mean()
    highest_avg_team = team_averages.idxmax()
    highest_avg_value = team_averages.max()
    lowest_avg_team = team_averages.idxmin()
    lowest_avg_value = team_averages.min()

    overall_average = table_calculation['adjusted_per_season'].mean()


    adjusted_across_all_seasons = (
        f"The adjusted average across 2014-15 to 2023-24 seasons for all teams is {overa
    )

    team_highest_average = (
        f"Highest average 4-in-6 nights across 2014-15 to 2023-24 seasons is {highest_av
        f"with {highest_avg_value:.2f} average stretches"
    )

    team_lowest_average = (
        f"Lowest average 4-in-6 nights across 2014-15 to 2023-24 seasons is {lowest_avg_
        f"with {lowest_avg_value:.2f} average stretches"
    )

    difference_between_most_and_least_average = (
        f"The difference between the most and least average 4 in 6 nights across 2014-15
        f"from {highest_avg_team} and {lowest_avg_team}"
    )

    return (highest_4_in_6_overall, lowest_4_in_6_overall, adjusted_across_all_seasons,


schedule =  get_schedule()
seasons_in_2014_to_2024 = get_all_seasons(schedule=schedule)
teams_in_2014_to_2024 = get_all_teams(schedule=schedule)

for szn in seasons_in_2014_to_2024:
    for team in teams_in_2014_to_2024:
        num_4_6, raw_games_played, adjusted_per_season, fourth_games = analyze_4_in_6_ga
        table_calculation.append({
            'season': szn,
            'team': team,
```

```
            'num_4_6': num_4_6,
            'raw_games_played': raw_games_played,
            'adjusted_per_season': adjusted_per_season
        })

table_calculation = pd.DataFrame(table_calculation)


highest_4_in_6_overall, lowest_4_in_6_overall, adjusted_across_all_seasons, team_highest
print(highest_4_in_6_overall)
print(lowest_4_in_6_overall)
print(adjusted_across_all_seasons)
print(team_highest_average)
print(team_lowest_average)
print(difference_between_most_and_least_average)
```

```
Highest: WAS in season 2020 with 45.6 stretches
Lowest: BOS in season 2018 with 13.0 stretches | Lowest: TOR in season 2018 with 13.0 str
etches
The adjusted average across 2014-15 to 2023-24 seasons for all teams is 25.10
Highest average 4-in-6 nights across 2014-15 to 2023-24 seasons is CHA with 28.11 average
stretches
Lowest average 4-in-6 nights across 2014-15 to 2023-24 seasons is NYK with 22.19 average
stretches
The difference between the most and least average 4 in 6 nights across 2014-15 to 2023-24
seasons is 5.92 from CHA and NYK
```

In [9]:
```
print(adjusted_across_all_seasons)
```

```
The adjusted average across 2014-15 to 2023-24 seasons for all teams is 25.10
```

ANSWER 2:

25.10 4-in-6 stretches on average.

Question 3

QUESTION: Which of the 30 NBA teams has had the highest average number of 4-in-6 stretches between 2014-15 and 2023-24? Which team has had the lowest average? Adjust each team/season to per-82 games.

In [10]:
```
print(team_highest_average)
print(team_lowest_average)
```

```
Highest average 4-in-6 nights across 2014-15 to 2023-24 seasons is CHA with 28.11 average
stretches
Lowest average 4-in-6 nights across 2014-15 to 2023-24 seasons is NYK with 22.19 average
stretches
```

ANSWER 3:

- Most 4-in-6 stretches on average: CHA (28.11)
- Fewest 4-in-6 stretches on average: NYK (22.19)

Question 4

QUESTION: Is the difference between most and least from Q3 surprising, or do you expect that size difference is likely to be the result of chance?

In [11]:
```
print(difference_between_most_and_least_average)
```

```
The difference between the most and least average 4 in 6 nights across 2014-15 to 2023-24
seasons is 5.92 from CHA and NYK
```

ANSWER to 4:

The difference between the most and least average 4 in 6 nights across 2014-15 to 2023-24 seasons is 5.92 from CHA and NYK

Although some variation between teams is expected, numerous articles, journalists, and sports enthusiasts have discussed factors such as TV scheduling and stadium

availability that influence game clustering.

An average difference of 5.92 4-in-6 nights per season over the years is substantial, suggesting that this difference is not merely due to chance.

Source: A detailed information about how these factors influence can be found here (https://www.nbastuffer.com/analytics101/how-the-nba-schedule-is-made)

ANSWER 4:

Question 5

QUESTION: What was BKN's defensive eFG% in the 2023-24 season? What was their defensive eFG% that season in situations where their opponent was on the second night of back-to-back?

In [12]:
```
'''
Function is responsible for computing a team's defensive efg% in a certain season
'''
def compute_def_efg(game_stats_2014_2024: pd.DataFrame, season: int, team : str) -> floa
    bkn_def_games = game_stats_2014_2024[(game_stats_2014_2024['season'] == season) & (g

    total_fg_made = bkn_def_games['fgmade'].sum()
    total_3pt_made = bkn_def_games['fg3made'].sum()
    total_fga_allowed = bkn_def_games['fgattempted'].sum()

    if total_fga_allowed == 0:
        return 0.0

    def_efg = (total_fg_made + 0.5 * total_3pt_made) / total_fga_allowed
    return def_efg
'''
Function is responsible for computing a team's defensive efg% when the opponent is on a
'''

def compute_def_efg_opp_b2b(game_data: pd.DataFrame, season: int, team: str) -> float:

    sort_data_frame_in_place(game_data, ['off_team', 'gamedate'])

    game_data['days_since_last'] = game_data.groupby('off_team')['gamedate'].diff().dt.d

    game_data['is_b2b'] = game_data['days_since_last'] == 1

    season_data = game_data[game_data['season'] == season]

    b2b_def_games = season_data[
        (season_data['def_team'] == team) &
        (season_data['is_b2b'])
    ]

    total_fg_made = b2b_def_games['fgmade'].sum()
    total_3pt_made = b2b_def_games['fg3made'].sum()
    total_fga_allowed = b2b_def_games['fgattempted'].sum()

    if total_fga_allowed == 0:
        return 0.0

    def_efg = (total_fg_made + 0.5 * total_3pt_made) / total_fga_allowed
    return def_efg



season = 2023
team = "BKN"
def_efg = compute_def_efg(get_game_data(), season, team)
def_efg_when_opponet_b2b = compute_def_efg_opp_b2b(get_game_data(), season, team)

print(f"{team} defensive eFG% in {season}-{season+1} season: {def_efg:.2f}")
print(f"{team} defensive eFG% when oppoenent playing back-to-back nights in {season}-{se
```

```
BKN defensive eFG% in 2023-2024 season: 0.54
BKN defensive eFG% when oppoenent playing back-to-back nights in 2023-2024 season: 0.53
```

ANSWER 5:

- BKN Defensive eFG%: 0.54%
- When opponent on a B2B: 0.53%

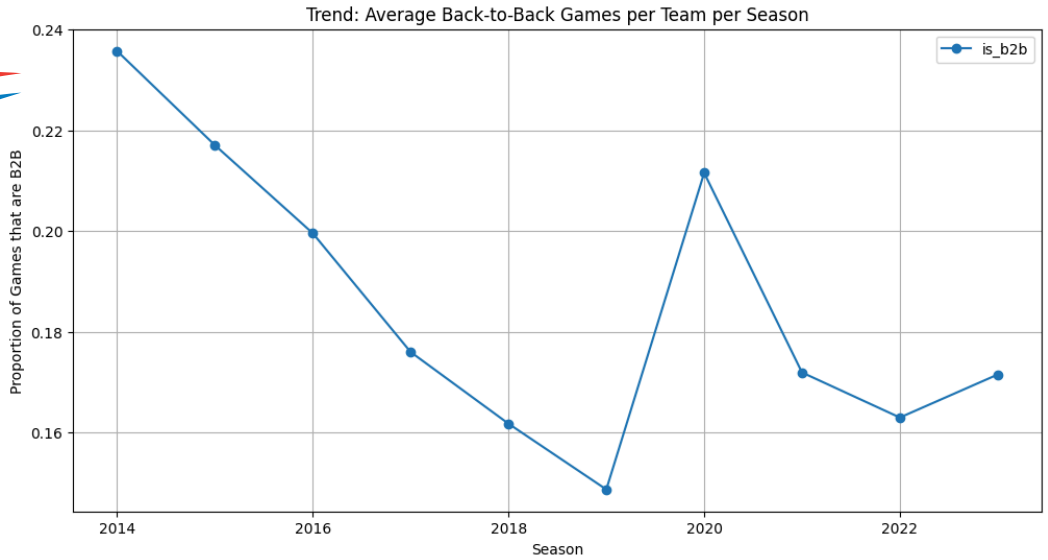
Part 2 -- Trends and Visualizations

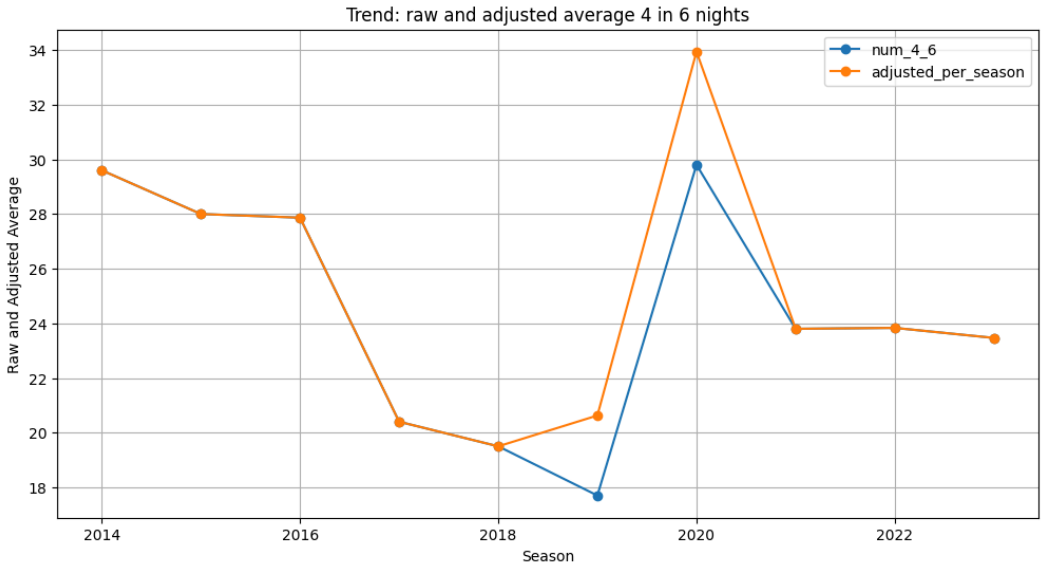
This is an intentionally open ended section, and there are multiple approaches you could take to have a successful project. Feel free to be creative. However, for this section, please consider only the density of games and travel schedule, not the relative on-court strength of different teams.

Question 6

QUESTION: Please identify at least 2 trends in scheduling over time. In other words, how are the more recent schedules different from the schedules of the past? Please include a visual (plot or styled table) highlighting or explaining each trend and include a brief written description of your findings.

In [13]:
```
'''
Function takes frame and flags whether the game is on b2b night
'''
def find_b2b_games(data: pd.DataFrame) -> pd.DataFrame:

    data_copy = sort_data_frame(data, ['team', 'gamedate'])

    data_copy['days_since_last'] = data_copy.groupby('team')['gamedate'].diff().dt.days
    data_copy['is_b2b'] = data_copy['days_since_last'] == 1

    return data_copy

'''
Function block is responsible for displaying mean of a given metric over time
'''
def show_plot(metrics: pd.DataFrame, x_value: str, y_values: list[str], plt_title: str,
    season_avgs = metrics.groupby(x_value)[y_values].mean()

    plt.figure(figsize=(12, 6))

    for col in season_avgs.columns:
        plt.plot(season_avgs.index, season_avgs[col], marker='o', label=col)

    plt.title(plt_title)
    plt.xlabel(plt_xlabel)
    plt.ylabel(plt_ylabel)
    plt.legend()
    plt.grid(show_grid)
    plt.show()

b2b_season_flag = find_b2b_games(data=get_schedule())

show_plot(metrics=b2b_season_flag, x_value='season', y_values=['is_b2b'], plt_title='Tre
    plt_xlabel='Season', plt_ylabel='Proportion of Games that are B2B', show_grid=True)

show_plot(metrics=table_calculation, x_value='season', y_values=['num_4_6', 'adjusted_pe
    plt_xlabel='Season', plt_ylabel= 'Raw and Adjusted Average', show_grid=True)
```





ANSWER 6:

Trends

- The proportion of back-to-back games peaked in 2014, trended towards declined until 2019, but rose again in 2020.
- Similarly, the number of 4-in-6 stretches (both raw and adjusted) spiked in 2020, although the longer-term trend indicates a gradual decline but not a sharp one.

Question 7

QUESTION: Please design a plotting tool to help visualize a team's schedule for a season. The plot should cover the whole season and should help the viewer contextualize and understand a team's schedule, potentially highlighting periods of excessive travel, dense blocks of games, or other schedule anomalies. If you can, making the plots interactive (for example through the plotly package) is a bonus.

Please use this tool to plot OKC and DEN's provided 80-game 2024-25 schedules.

In [14]:
```
'''
Function is responsible for joining schedule and location
uses the can take either the schedule csv or draft schedule and locations csv
'''
def merge_team_opponent_locations(team_sched: pd.DataFrame, locations_df: pd.DataFrame)
```

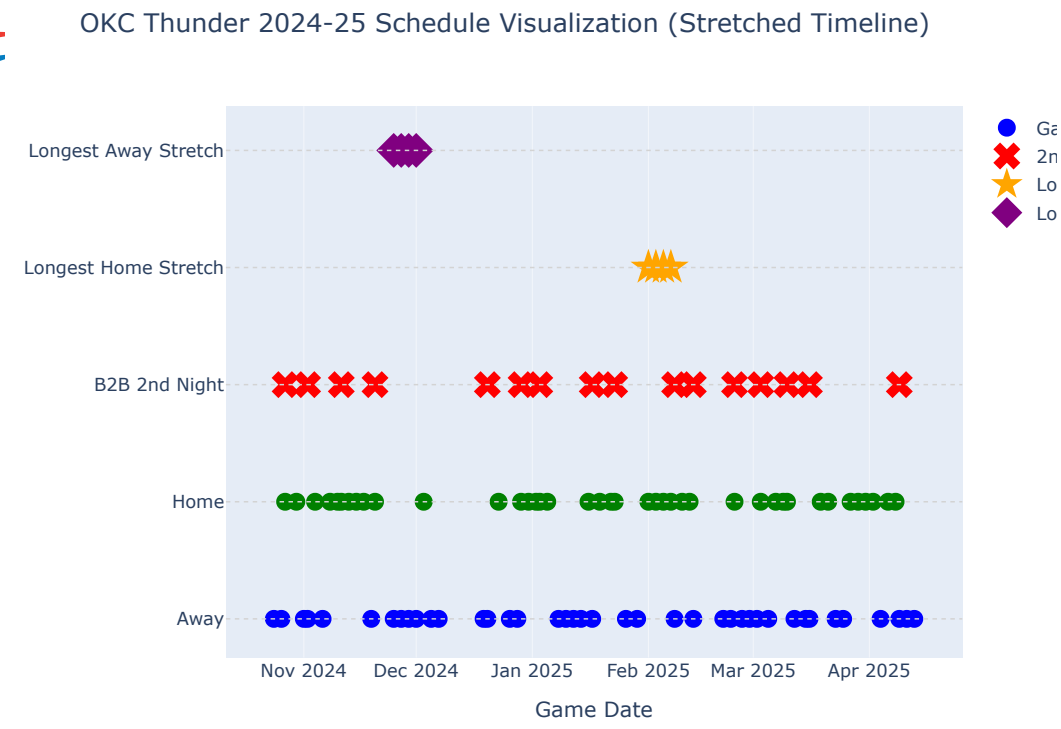
```
    team_sched = team_sched.merge(
        locations_df[['team', 'latitude', 'longitude']].rename(
            columns={'latitude': 'team_lat', 'longitude': 'team_lon'}
        ),
        left_on='team', right_on='team', how='left'
    )
    team_sched = team_sched.merge(
        locations_df[['team', 'latitude', 'longitude']].rename(
            columns={'team': 'opp_team', 'latitude': 'opp_lat', 'longitude': 'opp_lon'}
        ),
        left_on='opponent', right_on='opp_team', how='left'
    ).drop(columns=['opp_team'])
    return team_sched
'''
Function computes distances between games and flags whether back to back games
'''
def compute_travel_and_b2b(team_sched: pd.DataFrame) -> pd.DataFrame:
    team_sched = team_sched.sort_values('gamedate').reset_index(drop=True)

    distances = [0]
    for i in range(1, len(team_sched)):
        prev = team_sched.iloc[i-1]
        curr = team_sched.iloc[i]
        prev_loc = (prev['team_lat'], prev['team_lon']) if prev['home']==1 else (prev['o
        curr_loc = (curr['team_lat'], curr['team_lon']) if curr['home']==1 else (curr['o
        distances.append(geodesic(prev_loc, curr_loc).miles)
    team_sched.loc[:, 'travel_miles'] = distances

    team_sched.loc[:, 'days_since_last'] = team_sched['gamedate'].diff().dt.days
    team_sched.loc[:, 'is_b2b'] = (team_sched['days_since_last'] == 1).astype(int)
    return team_sched

'''
Function computes longest stretch of home and away games
'''
def compute_longest_home_and_away_stretch(team_sched: pd.DataFrame) -> dict:
    # Initialize home stretch tracking
    max_home_stretch = 0
    current_home_stretch = 0
    home_start = None
    home_end = None
    temp_home_start = None

    # Initialize away stretch tracking
    max_away_stretch = 0
    current_away_stretch = 0
    away_start = None
    away_end = None
    temp_away_start = None

    for idx, row in team_sched.iterrows():
        if row['home'] == 1 and row['is_b2b'] == 0:
            if current_home_stretch == 0:
                temp_home_start = row['gamedate']
            current_home_stretch += 1
            if current_home_stretch > max_home_stretch:
                max_home_stretch = current_home_stretch
                home_start = temp_home_start
                home_end = row['gamedate']
        else:
            current_home_stretch = 0
            temp_home_start = None

        if row['home'] == 0:
            if current_away_stretch == 0:
                temp_away_start = row['gamedate']
            current_away_stretch += 1
            if current_away_stretch > max_away_stretch:
                max_away_stretch = current_away_stretch
                away_start = temp_away_start
                away_end = row['gamedate']
        else:
            current_away_stretch = 0
            temp_away_start = None
```

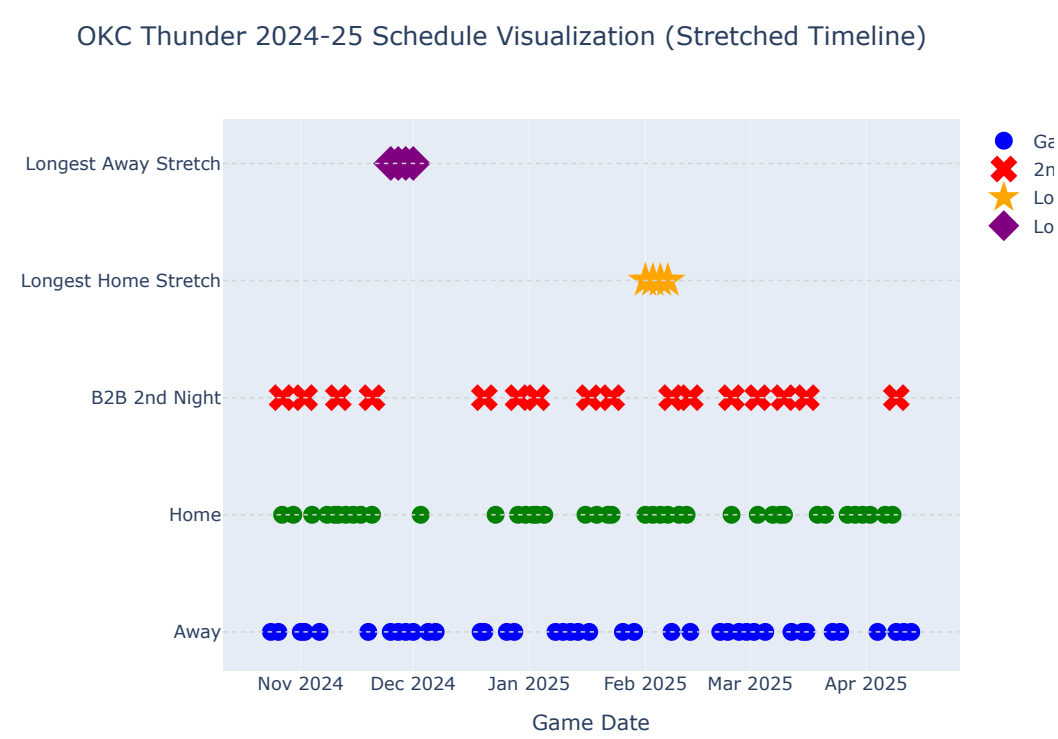
```
    return {
        'max_home_stretch': max_home_stretch,
        'home_stretch_start': home_start,
        'home_stretch_end': home_end,
        'max_away_stretch': max_away_stretch,
        'away_stretch_start': away_start,
        'away_stretch_end': away_end
    }

'''
Function computes the full pipeline
returns the updated frame and computed metrics
'''
def prepare_team_schedule(team_abbrev: str, draft_schedule: pd.DataFrame, locations_df:
    team_sched = draft_schedule[draft_schedule['team'] == team_abbrev]
    team_sched = merge_team_opponent_locations(team_sched, locations_df)
    team_sched = compute_travel_and_b2b(team_sched)
    metrics = compute_longest_home_and_away_stretch(team_sched)
    return team_sched, metrics
```

In [33]:

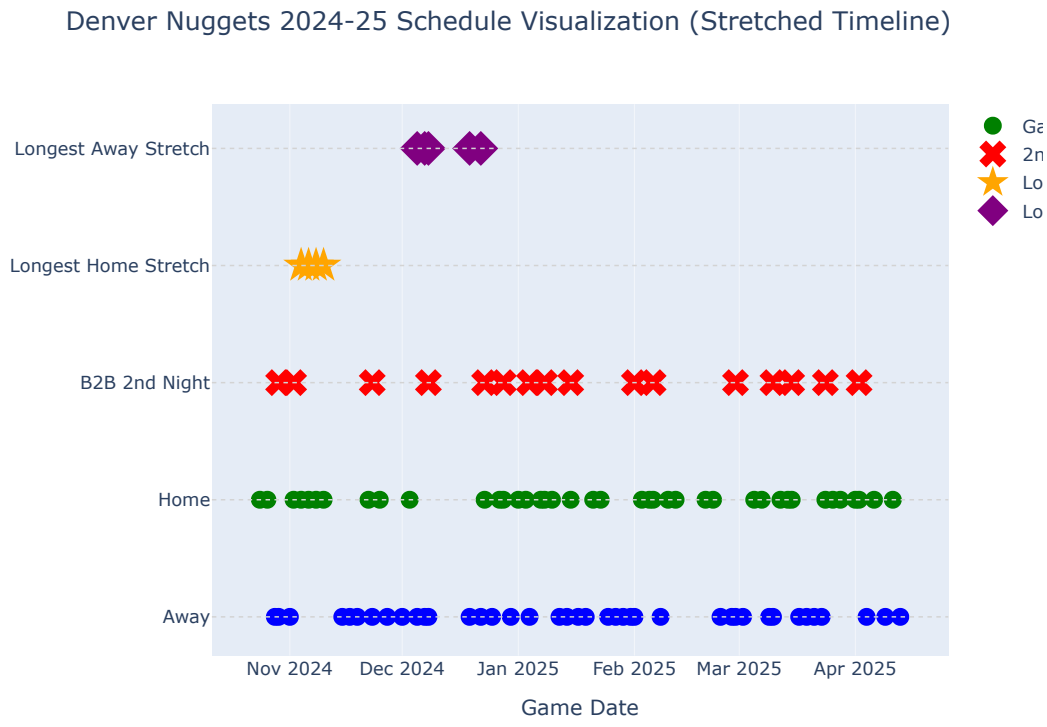
```
'''
Function block is responsible for making an interactive plot for both OKC and DEN
'''
def plot_team_schedule_stretched(team_sched, team_name, metrics):
    fig = go.Figure()

    team_sched = get_copy_of_a_data_frame( team_sched )
    team_sched.loc[:, 'y_val'] = team_sched['home'].map({1: 2, 0: 1})

    team_sched.loc[:, 'home_str'] = team_sched['home'].map({1: 'Yes', 0: 'No'})
    team_sched.loc[:, 'b2b_str'] = team_sched['is_b2b'].map({1: 'Yes', 0: 'No'})

    fig.add_trace(go.Scatter(
        x=team_sched['gamedate'],
        y=team_sched['y_val'],
        mode='markers',
        marker=dict(size=12, color=team_sched['home'].map({1:'green', 0:'blue'})),
        text=team_sched.apply(lambda r: f"{r['opponent']}<br>Home: {r['home_str']}<br>Tr
        hoverinfo='text',
        name='Game (green=home, blue=away)'
    ))

    b2b_sched = get_copy_of_a_data_frame( team_sched[team_sched['is_b2b'] == 1] )
    b2b_sched.loc[:, 'y_val'] = 3
    fig.add_trace(go.Scatter(
        x=b2b_sched['gamedate'],
        y=b2b_sched['y_val'],
        mode='markers',
        marker=dict(size=16, color='red', symbol='x'),
        name='2nd Night of B2B'
    ))

    if metrics['home_stretch_start'] is not None:
        stretch_mask = (
            (team_sched['gamedate'] >= metrics['home_stretch_start']) &
            (team_sched['gamedate'] <= metrics['home_stretch_end']) &
            (team_sched['home'] == 1) &
            (team_sched['is_b2b'] == 0)
        )
        date_range_str = f"{metrics['home_stretch_start'].strftime('%m/%d')} - {metrics[
        fig.add_trace(go.Scatter(
            x=team_sched.loc[stretch_mask, 'gamedate'],
            y=[4] * stretch_mask.sum(),
            mode='markers',
            marker=dict(size=18, color='orange', symbol='star'),
            name=f'Longest Home Stretch w/o B2B ({date_range_str})'
        ))

    if metrics['away_stretch_start'] is not None:
        away_mask = (
            (team_sched['gamedate'] >= metrics['away_stretch_start']) &
            (team_sched['gamedate'] <= metrics['away_stretch_end']) &
            (team_sched['home'] == 0)
        )
```

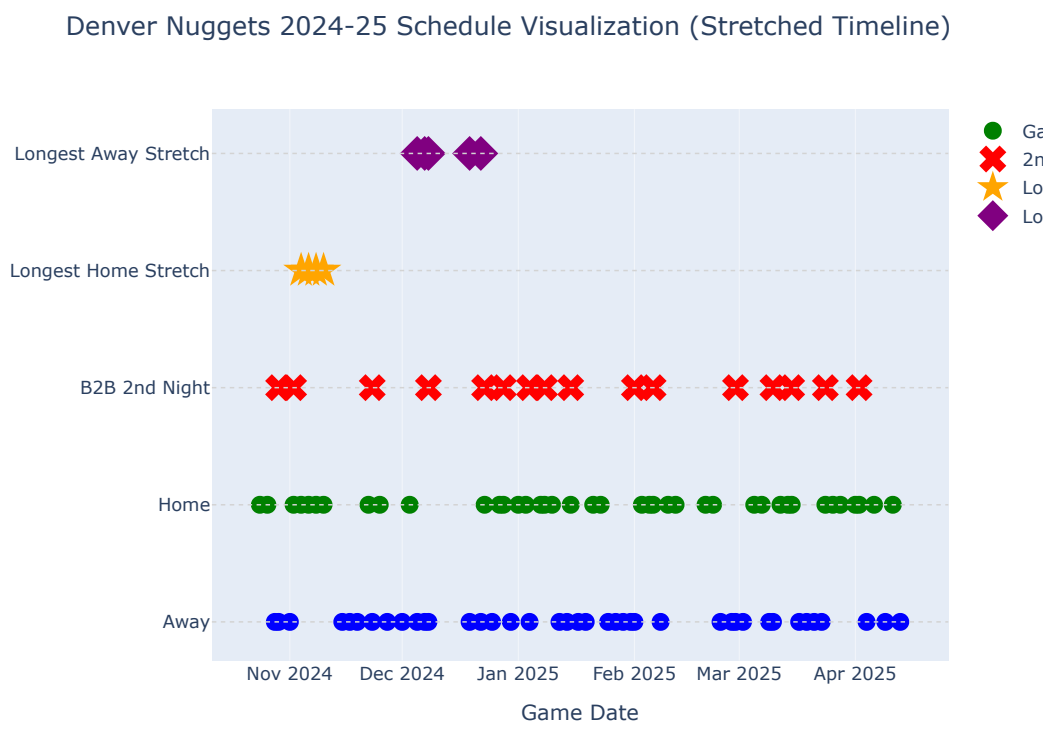
```
        date_range_str_away = f"{metrics['away_stretch_start'].strftime('%m/%d')} - {met
        fig.add_trace(go.Scatter(
            x=team_sched.loc[away_mask, 'gamedate'],
            y=[5] * away_mask.sum(),
            mode='markers',
            marker=dict(size=18, color='purple', symbol='diamond'),
            name=f'Longest Away Stretch ({date_range_str_away})'
        ))

    for y, label in zip([1, 2, 3, 4, 5], ['Away', 'Home', 'B2B 2nd Night', 'Longest Home
        fig.add_hline(y=y, line_color='lightgray', line_width=1, line_dash="dot")

    fig.update_layout(
        title=f"{team_name} 2024-25 Schedule Visualization (Stretched Timeline)",
        xaxis_title="Game Date",
        yaxis=dict(
            tickmode='array',
            tickvals=[1, 2, 3, 4, 5],
            ticktext=['Away', 'Home', 'B2B 2nd Night', 'Longest Home Stretch', 'Longest
            showgrid=False
        ),
        width=1000,
        height=550
    )

    fig.show()
    display(HTML(fig.to_html(full_html=False)))



teams = {'OKC': 'OKC Thunder', 'DEN': 'Denver Nuggets'}


team_stretches = {}

for abbrev, full_name in teams.items():
    team_sched, metrics = prepare_team_schedule(abbrev, get_draft_schedule(), get_locati

    plot_team_schedule_stretched(team_sched, full_name, metrics)

    team_stretches[full_name] = {}

    if metrics['home_stretch_start'] is not None and metrics['home_stretch_end'] is not
        home_info = (
            f"{metrics['max_home_stretch']} games "
            f"from {metrics['home_stretch_start'].strftime('%Y-%m-%d')} "
            f"to {metrics['home_stretch_end'].strftime('%Y-%m-%d')}"
        )
        print(f"{full_name} longest home stretch without B2B: {home_info}")
        team_stretches[full_name]['home_stretch'] = home_info

    if metrics['away_stretch_start'] is not None and metrics['away_stretch_end'] is not
        away_info = (
            f"{metrics['max_away_stretch']} games "
            f"from {metrics['away_stretch_start'].strftime('%Y-%m-%d')} "
            f"to {metrics['away_stretch_end'].strftime('%Y-%m-%d')}"
        )
        print(f"{full_name} longest away stretch: {away_info}")
        team_stretches[full_name]['away_stretch'] = away_info
```






```
OKC Thunder longest home stretch without B2B: 4 games from 2025-02-01 to 2025-02-07
OKC Thunder longest away stretch: 4 games from 2024-11-25 to 2024-12-01
```






```
Denver Nuggets longest home stretch without B2B: 4 games from 2024-11-04 to 2024-11-10
Denver Nuggets longest away stretch: 5 games from 2024-12-05 to 2024-12-22
```

ANSWER 7:

Question 8

QUESTION: Using your tool, what is the best and worst part of OKC's 2024-25 draft schedule? Please give your answer as a short brief to members of the front office and coaching staff to set expectations going into the season. You can include context from past schedules.

In [16]:
```
print("======The best part of OKC's 2024-25 schedule======")
print(f"Longest home stretch without back to back games {team_stretches['OKC Thunder']['
```

```
======The best part of OKC's 2024-25 schedule======
Longest home stretch without back to back games 4 games from 2025-02-01 to 2025-02-07
```

In [17]:
```
print("======The worst part of OKC's 2024-25 schedule======")
print(f"Longest away stretch {team_stretches['OKC Thunder']['away_stretch']}")
```

```
======The worst part of OKC's 2024-25 schedule======
Longest away stretch 4 games from 2024-11-25 to 2024-12-01
```

ANSWER 8:

- OKC Thunder longest home stretch without B2B: 4 games from 2025-02-01 to 2025-02-07
- OKC Thunder longest away stretch: 4 games from 2024-11-25 to 2024-12-01

Part 3 -- Modeling

Question 9

QUESTION: Please estimate how many more/fewer regular season wins each team has had due to schedule-related factors from 2019-20 though 2023-24. Your final answer should have one number for each team, representing the total number of wins (not per 82, and not a per-season average). You may consider the on-court strength of the scheduled opponents as well as the impact of travel/schedule density. Please include the teams and estimates for the most helped and most hurt in the answer key.

If you fit a model to help answer this question, please write a paragraph explaining your model, and include a simple model diagnostic (eg a printed summary of a regression, a variable importance plot, etc).

In [18]:
```
def merge_schedule_location_team_data(schedule: pd.DataFrame, location: pd.DataFrame, ga

    schedule = schedule.merge(
        location[['team','latitude','longitude']].rename(columns={
            'team':'home_team', 'latitude':'home_lat', 'longitude':'home_lon'
        }),
        left_on='team', right_on='home_team', how='left'
    )

    schedule = schedule.merge(
        location[['team','latitude','longitude']].rename(columns={
            'team':'opp_team', 'latitude':'opp_lat', 'longitude':'opp_lon'
        }),
        left_on='opponent', right_on='opp_team', how='left'
    )

    schedule = schedule.drop(columns=['home_team','opp_team'])

    merged_df = schedule.merge(
        game_data,
        left_on=['season','gamedate','team','opponent'],
        right_on=['season','gamedate','off_team','def_team'],
        how='inner'
    ).drop(columns=['off_team','def_team'])

    def_cols = [c for c in game_data.columns if c not in ['season','gamedate','off_team'
    def_stats = game_data.rename(columns={c: c+'_def' for c in def_cols})

    merged_df = merged_df.merge(
        def_stats,
        left_on=['season','gamedate','team','opponent'],
```

```
        right_on=['season','gamedate','def_team','off_team'],
        how='inner'
    ).drop(columns=['off_team','def_team'])

    return merged_df

def drop_columns(data: pd.DataFrame, columns_to_drop: list) -> None:
    data.drop(columns=[c for c in columns_to_drop if c in data.columns], inplace=True)

schedule_locations_game_data_to_process_no_team_strength = merge_schedule_location_team_
schedule_locations_game_data_to_process_team_strength = merge_schedule_location_team_dat

columns_to_remove = [
    'nbagameid', 'offensivenbateamid', 'off_team_name', 'off_home', 'off_win',
    'defensivenbateamid', 'def_team_name', 'def_home', 'def_win',
    'defensivefouls', 'offensivefouls', 'shootingfoulsdrawn', 'andones',
    'gametype_def', 'nbagameid_def', 'offensivenbateamid_def', 'off_team_name_def',
    'off_home_def', 'off_win_def', 'defensivenbateamid_def', 'def_team_name_def',
    'def_home_def', 'def_win_def', 'defensivefouls_def', 'offensivefouls_def',
    'shootingfoulsdrawn_def', 'andones_def'
]

drop_columns(schedule_locations_game_data_to_process_no_team_strength, columns_to_remove
drop_columns(schedule_locations_game_data_to_process_team_strength, columns_to_remove)
```

In [19]:
```
schedule_locations_game_data_to_process_no_team_strength.columns
```

Out[19]:
```
Index(['season', 'gamedate', 'team', 'opponent', 'home', 'win', 'home_lat',
       'home_lon', 'opp_lat', 'opp_lon', 'gametype', 'fg2made', 'fg2missed',
       'fg2attempted', 'fg3made', 'fg3missed', 'fg3attempted', 'fgmade',
       'fgmissed', 'fgattempted', 'ftmade', 'ftmissed', 'ftattempted',
       'reboffensive', 'rebdefensive', 'reboundchance', 'assists',
       'stealsagainst', 'turnovers', 'blocksagainst', 'possessions', 'points',
       'shotattempts', 'shotattemptpoints', 'fg2made_def', 'fg2missed_def',
       'fg2attempted_def', 'fg3made_def', 'fg3missed_def', 'fg3attempted_def',
       'fgmade_def', 'fgmissed_def', 'fgattempted_def', 'ftmade_def',
       'ftmissed_def', 'ftattempted_def', 'reboffensive_def',
       'rebdefensive_def', 'reboundchance_def', 'assists_def',
       'stealsagainst_def', 'turnovers_def', 'blocksagainst_def',
       'possessions_def', 'points_def', 'shotattempts_def',
       'shotattemptpoints_def'],
      dtype='object')
```

Feature Engineering

Description of each feature engineered

Fatigue Variants:

- Creates different mathematical versions of "rest" (inverse, exponential, sigmoid, logarithmic) for both teams and opponents, then compares them to capture relative fatigue effects based on days since last game.

Back-to-Back (B2B) Flag

- Marks games where a team plays on consecutive days, since those situations are often tougher due to less recovery time.

4-in-6 Flag

- Flags games where a team plays 4 games within 6 days, a well-known NBA scheduling fatigue scenario.

Travel Distance

- Calculates the miles traveled between consecutive games for each team, factoring in whether they were home or away, to capture travel-related fatigue.

Advanced Team Stats

- Transforms raw box score stats into advanced efficiency metrics (offense and defense) like effective field goal %, offensive/defensive ratings, turnover %, and rebound %. Removes raw columns afterward to keep only the derived stats.
- For this, i engineered, PPA, OREB, DREB, TOV, STL, BLK, ORTG, DRTG, NET_RTG, EFG for both offense and defense

Recent Game Counts

- Counts how many games a team (and their opponent) has played in the past few days (e.g., last 3, 5, 7, or 14 days), to reflect schedule density.

Date Features

- Encodes game dates into cyclical signals (day of year, weekday, month, plus sine/cosine transforms) to capture seasonality and weekly/monthly patterns.

Win Features

- Tracks rolling and cumulative win rates for teams and opponents, so the model can learn momentum and overall strength trends leading into each game.

Days Since Last Game

- Measures rest days since a team's last game (clipped to a max value), serving as a direct rest/recovery indicator.

In [20]:
```
def add_fatigue_variants(data: pd.DataFrame, lambda_exp=0.5, theta_sigmoid=2, max_days=1

    sort_data_frame_in_place(data, ['team', 'gamedate'])

    days_team = data.groupby('team')['gamedate'].diff().dt.days.fillna(3).clip(lower=1,
    days_opp = data.groupby('opponent')['gamedate'].diff().dt.days.fillna(3).clip(lower=

    data['team_fatigue_inv'] = 1 / (days_team + 1)
    data['team_fatigue_exp'] = np.exp(-lambda_exp * days_team)
    data['team_fatigue_sigmoid'] = 1 / (1 + np.exp(-(days_team - theta_sigmoid)))
    data['team_fatigue_log'] = 1 / np.log(days_team + 2)

    data['opp_fatigue_inv'] = 1 / (days_opp + 1)
    data['opp_fatigue_exp'] = np.exp(-lambda_exp * days_opp)
    data['opp_fatigue_sigmoid'] = 1 / (1 + np.exp(-(days_opp - theta_sigmoid)))
    data['opp_fatigue_log'] = 1 / np.log(days_opp + 2)

    data['rel_fatigue_inv'] = data['team_fatigue_inv'] - data['opp_fatigue_inv']
    data['rel_fatigue_exp'] = data['team_fatigue_exp'] - data['opp_fatigue_exp']
    data['rel_fatigue_sigmoid'] = data['team_fatigue_sigmoid'] - data['opp_fatigue_sigmo
    data['rel_fatigue_log'] = data['team_fatigue_log'] - data['opp_fatigue_log']

    return data

def add_b2b_flag(data: pd.DataFrame) -> None:
    data['days_since_last'] = data.groupby('team')['gamedate'].diff().dt.days
    data['is_b2b'] = (data['days_since_last'] == 1).astype(int)


def add_4in6_flag(data: pd.DataFrame) -> None:
    data['is_4in6'] = 0

    for team in data['team'].unique():
        team_mask = data['team'] == team
        team_df = data.loc[team_mask]

        for i in range(3, len(team_df)):
            window = team_df.iloc[i-3:i+1]
            if (window['gamedate'].max() - window['gamedate'].min()).days <= 5:
                data.loc[window.index[-1], 'is_4in6'] = 1

def add_travel_distance(data: pd.DataFrame) -> None:
    data['travel_miles'] = 0.0

    for team in data['team'].unique():
```

```
        team_df = data[data['team'] == team]
        team_indices = team_df.index.tolist()

        distances = [0.0]
        for i in range(1, len(team_df)):
            prev = team_df.iloc[i-1]
            curr = team_df.iloc[i]

            prev_loc = (prev['home_lat'], prev['home_lon']) if prev['home'] == 1 else (p
            curr_loc = (curr['home_lat'], curr['home_lon']) if curr['home'] == 1 else (c

            distances.append(geodesic(prev_loc, curr_loc).miles)

        data.loc[team_indices, 'travel_miles'] = distances

def compute_team_advanced_stats(data: pd.DataFrame, add_team_strength : bool = True) ->
    if add_team_strength:
        data['PPA_offense'] = data['shotattemptpoints'] / data['shotattempts'].replace(0
        data['AST_pct_offense'] = data['assists'] / data['fgmade'].replace(0, 1)
        data['OREB_pct_offense'] = data['reboffensive'] / data['reboundchance'].replace(
        data['DREB_pct_offense'] = data['rebdefensive'] / data['reboundchance'].replace(
        data['TOV_pct_offense'] = data['turnovers'] / (data['shotattempts'] + data['turn
        data['STL_pct_offense'] = data['stealsagainst'] / data['possessions'].replace(0,
        data['BLK_pct_offense'] = data['blocksagainst'] / data['fg2attempted'].replace(0
        data['ORTG_offense'] = data['points'] / (data['possessions'] / 100).replace(0, 1
        data['DRTG_offense'] = data['points_def'] / (data['possessions_def'] / 100).repl
        data['NET_RTG_offense'] = data['ORTG_offense'] - data['DRTG_offense']
        data['eFG_offense'] = (data['fgmade'] + 0.5 * data['fg3made']) / data['fgattempt


        data['PPA_defense'] = data['shotattemptpoints_def'] / data['shotattempts_def'].r
        data['AST_pct_defense'] = data['assists_def'] / data['fgmade_def'].replace(0, 1)
        data['OREB_pct_defense'] = data['reboffensive_def'] / data['reboundchance_def'].
        data['DREB_pct_defense'] = data['rebdefensive_def'] / data['reboundchance_def'].
        data['TOV_pct_defense'] = data['turnovers_def'] / (data['shotattempts_def'] + da
        data['STL_pct_defense'] = data['stealsagainst_def'] / data['possessions_def'].re
        data['BLK_pct_defense'] = data['blocksagainst_def'] / data['fg2attempted_def'].r
        data['ORTG_defense'] = data['points_def'] / (data['possessions_def'] / 100).repl
        data['DRTG_defense'] = data['points'] / (data['possessions'] / 100).replace(0, 1
        data['NET_RTG_defense'] = data['ORTG_defense'] - data['DRTG_defense']
        data['eFG_defense'] = (data['fgmade_def'] + 0.5 * data['fg3made_def']) / data['f

    raw_cols = [
        'fg2made', 'fg2missed', 'fg2attempted',
        'fg3made', 'fg3missed', 'fg3attempted',
        'fgmade', 'fgmissed', 'fgattempted',
        'ftmade', 'ftmissed', 'ftattempted',
        'reboffensive', 'rebdefensive', 'reboundchance',
        'assists', 'stealsagainst', 'turnovers', 'blocksagainst',
        'points', 'possessions', 'shotattempts', 'shotattemptpoints',
        'fg2made_def', 'fg2missed_def', 'fg2attempted_def',
        'fg3made_def', 'fg3missed_def', 'fg3attempted_def',
        'fgmade_def', 'fgmissed_def', 'fgattempted_def',
        'ftmade_def', 'ftmissed_def', 'ftattempted_def',
        'reboffensive_def', 'rebdefensive_def', 'reboundchance_def',
        'assists_def', 'stealsagainst_def', 'turnovers_def', 'blocksagainst_def',
        'points_def', 'possessions_def', 'shotattempts_def', 'shotattemptpoints_def'
    ]
    data.drop(columns=[c for c in raw_cols if c in data.columns], inplace=True)

def add_recent_game_counts(data: pd.DataFrame, team_col='team', opp_col='opponent', wind
    results = {f'games_last_{w}': [] for w in windows}
    results.update({f'opp_games_last_{w}': [] for w in windows})

    for team in data[team_col].unique():
        team_mask = data[team_col] == team
        team_df = data.loc[team_mask].sort_values("gamedate")

        for w in windows:
            counts = []
            start_idx = 0
            for i in range(len(team_df)):
                current_date = team_df.iloc[i]['gamedate']
                while (current_date - team_df.iloc[start_idx]['gamedate']).days > w:
                    start_idx += 1
```

```
                counts.append(i - start_idx)

            s = pd.Series(counts, index=team_df.index)
            results[f'games_last_{w}'].append(s)

    for opp in data[opp_col].unique():
        opp_mask = data[opp_col] == opp
        opp_df = data.loc[opp_mask].sort_values("gamedate")
        for w in windows:
            counts = []
            start_idx = 0
            for i in range(len(opp_df)):
                current_date = opp_df.iloc[i]['gamedate']
                while (current_date - opp_df.iloc[start_idx]['gamedate']).days > w:
                    start_idx += 1
                counts.append(i - start_idx)

            s = pd.Series(counts, index=opp_df.index)
            results[f'opp_games_last_{w}'].append(s)

    for key in results:
        results[key] = pd.concat(results[key])
    new_df = pd.DataFrame(results)
    return pd.concat([data, new_df], axis=1)

def add_date_features(data: pd.DataFrame, date_col='gamedate') -> None:

    data[date_col] = pd.to_datetime(data[date_col])

    data['day_of_year'] = data[date_col].dt.dayofyear
    data['weekday'] = data[date_col].dt.weekday
    data['month'] = data[date_col].dt.month

    data['day_sin'] = np.sin(2 * np.pi * data['day_of_year'] / 365)
    data['day_cos'] = np.cos(2 * np.pi * data['day_of_year'] / 365)

    data['weekday_sin'] = np.sin(2 * np.pi * data['weekday'] / 7)
    data['weekday_cos'] = np.cos(2 * np.pi * data['weekday'] / 7)

    data['month_sin'] = np.sin(2 * np.pi * data['month'] / 12)
    data['month_cos'] = np.cos(2 * np.pi * data['month'] / 12)




def add_win_features_fixed(data: pd.DataFrame, team_col: str = "team",
    opp_col: str = "opponent",win_col: str = "win", rolling_windows: list[int] = [3, 5,

    df = get_copy_of_a_data_frame(data)

    team_groups = df.groupby(team_col)
    df[f"{team_col}_cum_wins"] = team_groups[win_col].cumsum().shift(1).fillna(0)
    df[f"{team_col}_cum_games"] = team_groups.cumcount()
    df[f"{team_col}_cum_winpct"] = df[f"{team_col}_cum_wins"] / df[f"{team_col}_cum_game

    opp_groups = df.groupby(opp_col)
    df[f"{opp_col}_cum_wins"] = opp_groups[win_col].cumsum().shift(1).fillna(0)
    df[f"{opp_col}_cum_games"] = opp_groups.cumcount()
    df[f"{opp_col}_cum_winpct"] = df[f"{opp_col}_cum_wins"] / df[f"{opp_col}_cum_games"]

    for w in rolling_windows:
        df[f"{team_col}_rolling_winpct_{w}"] = (
            team_groups[win_col].transform(lambda x: x.shift(1).rolling(w, min_periods=1
        )
        df[f"{opp_col}_rolling_winpct_{w}"] = (
            opp_groups[win_col].transform(lambda x: x.shift(1).rolling(w, min_periods=1)
        )

    rolling_cols = [f"{team_col}_cum_winpct", f"{opp_col}_cum_winpct"] + \
                   [f"{team_col}_rolling_winpct_{w}" for w in rolling_windows] + \
                   [f"{opp_col}_rolling_winpct_{w}" for w in rolling_windows]

    df[rolling_cols] = df[rolling_cols].fillna(0)

    df.drop(columns=[f"{team_col}_cum_wins", f"{team_col}_cum_games",
```

```
                    f"{opp_col}_cum_wins", f"{opp_col}_cum_games"], inplace=True)

    return df

def add_days_since_last(df: pd.DataFrame, max_days: int =7):

    df.sort_values(['team','gamedate'], inplace=True)

    df['days_since_last'] = df.groupby('team')['gamedate'].diff().dt.days

    df['days_since_last'] = df['days_since_last'].fillna(3)

    df['days_since_last'] = df['days_since_last'].clip(lower=1, upper=max_days).astype(f
```

In [21]:
```
sort_data_frame_in_place(schedule_locations_game_data_to_process_no_team_strength, by=['


compute_team_advanced_stats(schedule_locations_game_data_to_process_no_team_strength, Fa


windows = [3, 5, 7, 10, 15, 25, 30, 45, 60, 90]
schedule_locations_game_data_to_process_no_team_strength = add_recent_game_counts(
    data=schedule_locations_game_data_to_process_no_team_strength,
    windows=windows
)


add_b2b_flag(schedule_locations_game_data_to_process_no_team_strength)
schedule_locations_game_data_to_process_no_team_strength = add_fatigue_variants(data=sch
add_4in6_flag(schedule_locations_game_data_to_process_no_team_strength)
add_travel_distance(schedule_locations_game_data_to_process_no_team_strength)
add_days_since_last(schedule_locations_game_data_to_process_no_team_strength)
add_date_features(schedule_locations_game_data_to_process_no_team_strength)

schedule_locations_game_data_to_process_no_team_strength = add_win_features_fixed(
    data=schedule_locations_game_data_to_process_no_team_strength,
    rolling_windows=windows
)


schedule_locations_game_data_to_process_no_team_strength.fillna(0, inplace=True)
```

Function Blocks

setup_and_train_model

- Builds and trains a neural network with configurable hidden layers, regularization, dropout, and early stopping. Returns the trained model and its training history.

apply_one_hot_encoding_to_pre_post_2020_data

- Applies one-hot encoding to team and opponent columns for pre-2020 (train) and post-2020 (test) datasets. Ensures consistent encoding between both splits.

split_data_into_pre_and_post_2020_and_return

- Splits the full dataset into pre-2019 (training/validation) and 2019+ (evaluation) subsets.

split_pre_2020_into_internal_train_and_test

- Further splits the pre-2020 dataset into internal training and test sets.

encode_team_and_opponent_for_all_splits

- Applies one-hot encoding consistently to all splits (train, internal test, and post-2020). Keeps a _team column for later group-based analysis.

predict_on_test_data

- Uses the trained model to predict outcomes on test data, plots a confusion matrix, and prints key evaluation metrics.

estimate_loss_win

- Calculates per-team differences between actual and predicted wins/losses on test data. Useful for team-level error analysis.

extract_features_and_target_from_all_splits

- Separates features (X) from target (y) for all dataset splits while dropping non-feature columns.

standardize_numeric_features_for_all_splits

- Standardizes numeric features using StandardScaler fitted on training data and applies the transformation to all splits.

show_training_and_val_loss_and_accuracy

- Plots training vs. validation accuracy and loss curves over epochs to diagnose under/overfitting.

train_neural_network_with_given_architecture

- Trains a neural network based on a dictionary of hyperparameters (architecture, regularization, learning rate, etc.) and returns the model and history.

evaluate_model_on_internal_test_and_post_2020

- Runs evaluation on internal test data and post-2020 data, showing predictions, metrics, and team-level win/loss gains.

plot_training_curves

- Wrapper around show_training_and_val_loss_and_accuracy to visualize training curves.

train_visualize_diagnose

- Full pipeline function: splits data, encodes features, scales inputs, trains a model, evaluates it on multiple splits, and plots learning curves. Returns model and training history.

In [22]:
```
def setup_and_train_model(X_train_scaled, y_train, hidden_layers: list[int] = [60],
 hidden_activation: str = 'relu',output_activation: str = 'sigmoid',
    hidden_l2_strength: float = 0.01, output_l2_strength: float = 0.01,
    learning_rate: float = 0.01, validation_split: float = 0.1, epochs: int = 30,
    batch_size: int = 7000,patience: int = 5, dropout : float = 0.1) -> (Sequential, 'Hi
    model = Sequential()

    # Input layer
    model.add(Input(shape=(X_train_scaled.shape[1],)))

    # First hidden layer
    if len(hidden_layers) > 0:
        model.add(Dense(hidden_layers[0],
                        activation=hidden_activation,
                        kernel_regularizer=l2(hidden_l2_strength)))
        model.add(Dropout(dropout))

    if len(hidden_layers) > 1:
        for units in hidden_layers[1:]:
            model.add(Dense(units,
                            activation=hidden_activation,
                            kernel_regularizer=l2(hidden_l2_strength)))
            model.add(Dropout(dropout))
```

```
    model.add(Dense(1,
                    activation=output_activation,
                    kernel_regularizer=l2(output_l2_strength)))

    model.compile(
        optimizer=Adam(learning_rate=learning_rate),
        loss='binary_crossentropy',
        metrics=['accuracy', BinaryAccuracy()]
    )

    early_stop = EarlyStopping(monitor='val_loss', patience=patience, restore_best_weigh

    history = model.fit(
        X_train_scaled, y_train,
        validation_split=validation_split,
        epochs=epochs,
        batch_size=batch_size,
        callbacks=[early_stop],
        verbose = 0 # I am turning offf the model from printing, unecessary deatils. Cha
    )
    return model, history

def apply_one_hot_encoding_to_pre_post_2020_data(pre_2020: pd.DataFrame,
post_2020: pd.DataFrame) -> tuple[pd.DataFrame, pd.DataFrame, OneHotEncoder, OneHotEncod


    team_ohe = OneHotEncoder(sparse_output=False, drop='first')
    team_encoded = team_ohe.fit_transform(pre_2020[['team']])
    team_cols = team_ohe.get_feature_names_out(['team'])


    opp_ohe = OneHotEncoder(sparse_output=False, drop='first')
    opp_encoded = opp_ohe.fit_transform(pre_2020[['opponent']])
    opp_cols = opp_ohe.get_feature_names_out(['opponent'])


    pre_2020_encoded = pd.concat([
        pre_2020.reset_index(drop=True).drop(columns=['team', 'opponent']),
        pd.DataFrame(team_encoded, columns=team_cols),
        pd.DataFrame(opp_encoded, columns=opp_cols)
    ], axis=1)


    team_encoded_post = team_ohe.transform(post_2020[['team']])
    opp_encoded_post = opp_ohe.transform(post_2020[['opponent']])
    post_2020_encoded = pd.concat([
        post_2020.reset_index(drop=True).drop(columns=['team', 'opponent']),
        pd.DataFrame(team_encoded_post, columns=team_cols),
        pd.DataFrame(opp_encoded_post, columns=opp_cols)
    ], axis=1)

    return pre_2020_encoded, post_2020_encoded, team_ohe, opp_ohe


def split_data_into_pre_and_post_2020_and_return(data: pd.DataFrame):

    pre_2020 =  get_copy_of_a_data_frame (data[data['season'] < 2019])
    post_2020 = get_copy_of_a_data_frame (data[data['season'] >= 2019])
    return pre_2020, post_2020

def split_pre_2020_into_internal_train_and_test(pre_2020: pd.DataFrame, test_size: float
    train_pre_2020, test_pre_2020 = train_test_split(pre_2020, test_size=test_size, rand
    return train_pre_2020, test_pre_2020


def encode_team_and_opponent_for_all_splits(train_pre_2020, test_pre_2020, post_2020):

    train_pre_2020_encoded, post_2020_encoded, team_ohe, opp_ohe = apply_one_hot_encodin
        train_pre_2020, post_2020
    )

    team_encoded_test = team_ohe.transform(test_pre_2020[['team']])
    opp_encoded_test = opp_ohe.transform(test_pre_2020[['opponent']])

    test_pre_2020_encoded = pd.concat([
```

```
        test_pre_2020.reset_index(drop=True).drop(columns=['team', 'opponent']),
        pd.DataFrame(team_encoded_test, columns=team_ohe.get_feature_names_out(['team'])
        pd.DataFrame(opp_encoded_test, columns=opp_ohe.get_feature_names_out(['opponent'
    ], axis=1)

    train_pre_2020_encoded['_team'] = train_pre_2020['team'].reset_index(drop=True)
    test_pre_2020_encoded['_team'] = test_pre_2020['team'].reset_index(drop=True)
    post_2020_encoded['_team'] = post_2020['team'].reset_index(drop=True)

    return train_pre_2020_encoded, test_pre_2020_encoded, post_2020_encoded, team_ohe, o

def predict_on_test_data(model, X_test_scaled, df_test: pd.DataFrame) -> None:
    df = get_copy_of_a_data_frame( df_test )

    if '_team' not in df.columns and 'team' in df_test.columns:
        df['_team'] = df_test['team'].reset_index(drop=True)

    y_pred = model.predict(X_test_scaled).flatten()
    df['pred_win'] = (y_pred >= 0.51).astype(int)
    df['win_gain_loss'] = df['win'] - df['pred_win']

    team_win_gain_loss = df.groupby('_team')['win_gain_loss'].sum().sort_values(ascendin

    cm = confusion_matrix(df['win'], df['pred_win'])
    disp = ConfusionMatrixDisplay(cm)
    disp.plot(cmap='Blues')
    plt.show()
    print(f"Accuracy: {accuracy_score(df['win'], df['pred_win']):.3f}, "
          f"Precision: {precision_score(df['win'], df['pred_win']):.3f}, "
          f"Recall: {recall_score(df['win'], df['pred_win']):.3f}, "
          f"F1: {f1_score(df['win'], df['pred_win']):.3f}")

def estimate_loss_win(model, X_test_scaled, df_test: pd.DataFrame) -> None:
    df = get_copy_of_a_data_frame(df_test)

    if '_team' not in df.columns and 'team' in df_test.columns:
        df['_team'] = df_test['team'].reset_index(drop=True)

    y_pred = model.predict(X_test_scaled).flatten()
    df['pred_win'] = (y_pred >= 0.51).astype(int)
    df['win_gain_loss'] = df['win'] - df['pred_win']

    # Group by original team column
    team_win_gain_loss = df.groupby('_team')['win_gain_loss'].sum().sort_values(ascendin
    print(team_win_gain_loss)


def extract_features_and_target_from_all_splits(train_pre_2020_encoded, test_pre_2020_en
    X_train = train_pre_2020_encoded.drop(columns=['win', 'season', 'gamedate', '_team']
    y_train = train_pre_2020_encoded['win']

    X_internal_test = test_pre_2020_encoded.drop(columns=['win', 'season', 'gamedate', '
    y_internal_test = test_pre_2020_encoded['win']

    X_post = post_2020_encoded.drop(columns=['win', 'season', 'gamedate', '_team'])
    y_post = post_2020_encoded['win']

    return X_train, y_train, X_internal_test, y_internal_test, X_post, y_post


def standardize_numeric_features_for_all_splits(X_train, X_internal_test, X_post):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_internal_test_scaled = scaler.transform(X_internal_test)
    X_post_scaled = scaler.transform(X_post)
    return X_train_scaled, X_internal_test_scaled, X_post_scaled

def show_training_and_val_loss_and_accuracy(history):
    plt.figure(figsize=(12,5))

    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Val Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
```

```
    plt.title('Train/Validation Accuracy')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Binary Crossentropy Loss')
    plt.title('Train/Validation Loss')
    plt.legend()


def train_neural_network_with_given_architecture(X_train_scaled, y_train, architecture_d
    model, history = setup_and_train_model(
        X_train_scaled,
        y_train,
        hidden_layers=architecture_dictionary['hidden_layers'],
        hidden_activation=architecture_dictionary['hidden_activation'],
        output_activation=architecture_dictionary['output_activation'],
        hidden_l2_strength=architecture_dictionary['hidden_l2_strength'],
        output_l2_strength=architecture_dictionary['output_l2_strength'],
        learning_rate=architecture_dictionary['learning_rate'],
        validation_split=architecture_dictionary['validation_split'],
        epochs=architecture_dictionary['epochs'],
        batch_size=architecture_dictionary['batch_size'],
        patience=architecture_dictionary['patience']
    )

    return model, history


def evaluate_model_on_internal_test_and_post_2020(model, X_internal_test_scaled, test_pr
                                                  X_post_scaled, post_2020_encoded):
    print("========== Evaluations on internal test data ==========")
    predict_on_test_data(model, X_internal_test_scaled, test_pre_2020_encoded)

    print("========== Estimating win/loss gain on post-2020 data ==========")
    estimate_loss_win(model, X_post_scaled, post_2020_encoded)


def plot_training_curves(history):
    show_training_and_val_loss_and_accuracy(history)


def train_visualize_diagnose(data: pd.DataFrame, architecture_dictionary: dict):
    pre_2020, post_2020 = split_data_into_pre_and_post_2020_and_return(data)

    train_pre_2020, test_pre_2020 = split_pre_2020_into_internal_train_and_test(pre_2020

    train_pre_2020_encoded, test_pre_2020_encoded, post_2020_encoded, team_ohe, opp_ohe
        encode_team_and_opponent_for_all_splits(train_pre_2020, test_pre_2020, post_2020

    X_train, y_train, X_internal_test, y_internal_test, X_post, y_post = \
        extract_features_and_target_from_all_splits(train_pre_2020_encoded, test_pre_202

    X_train_scaled, X_internal_test_scaled, X_post_scaled = \
        standardize_numeric_features_for_all_splits(X_train, X_internal_test, X_post)

    model, history = train_neural_network_with_given_architecture(X_train_scaled, y_trai
    model.summary()

    evaluate_model_on_internal_test_and_post_2020(model, X_internal_test_scaled, test_pr
                                                  X_post_scaled, post_2020_encoded)

    plot_training_curves(history)

    return model, history
```

In [23]:

```
architecture_dictionary = {
    'hidden_layers': [60, 40, 60],
    'hidden_activation': 'relu',
    'output_activation': 'sigmoid',
    'hidden_l2_strength': 0.01,
    'output_l2_strength': 0.01,
    'learning_rate': 0.01,
```

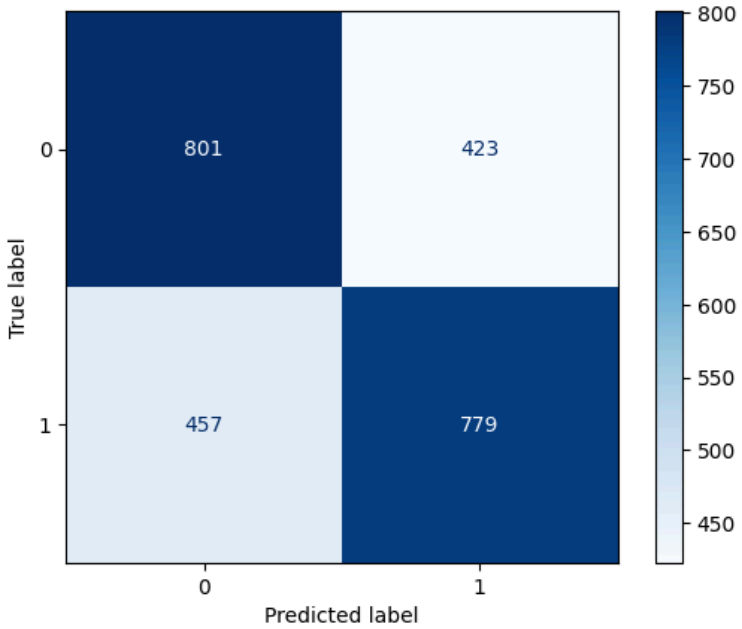
```
        'validation_split': 0.05,
        'epochs': 100,
        'batch_size': 10000,
        'patience': 5,
        'dropout' : 0.1
    }

_, _ = train_visualize_diagnose( schedule_locations_game_data_to_process_no_team_strengt
```

Model: "sequential"

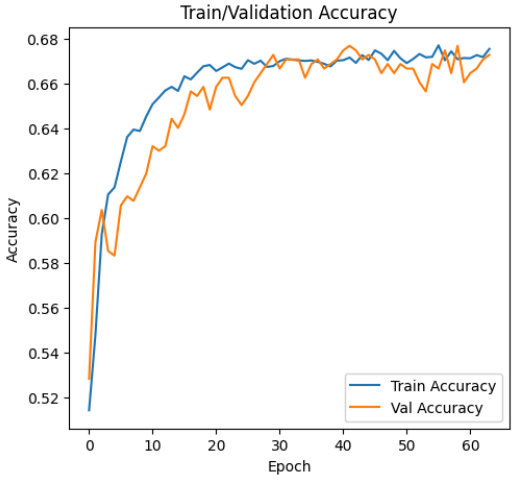
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 60)	7,920
dropout (Dropout)	(None, 60)	0
dense_1 (Dense)	(None, 40)	2,440
dropout_1 (Dropout)	(None, 40)	0
dense_2 (Dense)	(None, 60)	2,460
dropout_2 (Dropout)	(None, 60)	0
dense_3 (Dense)	(None, 1)	61

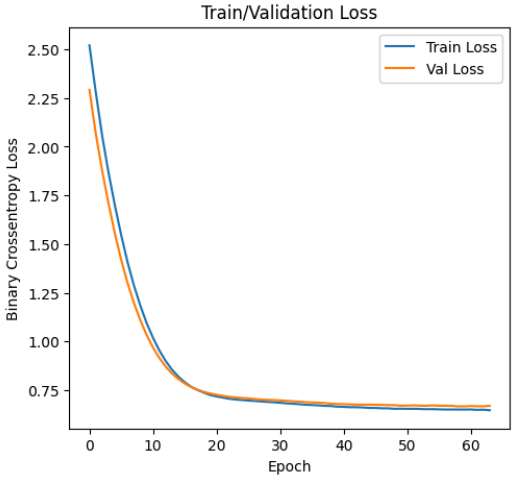
Total params: 38,645 (150.96 KB)
Trainable params: 12,881 (50.32 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 25,764 (100.64 KB)

```
========== Evaluations on internal test data ==========
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



```
Accuracy: 0.642, Precision: 0.648, Recall: 0.630, F1: 0.639
========== Estimating win/loss gain on post-2020 data ==========
365/365 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
team
NYK     85
LAL     82
SAC     70
PHX     63
CHI     62
MIN     62
ORL     44
BKN     43
DET     38
WAS     34
DAL     33
NOP     31
PHI     21
IND      9
MEM      8
ATL     -1
DEN     -2
CLE     -5
OKC    -13
CHA    -13
POR    -13
HOU    -18
MIL    -22
MIA    -24
UTA    -32
TOR    -46
LAC    -55
BOS    -55
SAS    -58
GSW   -118
Name: win_gain_loss, dtype: int64
```





In [24]:
```
sort_data_frame_in_place(schedule_locations_game_data_to_process_team_strength, by=['tea

compute_team_advanced_stats(schedule_locations_game_data_to_process_team_strength, True)

windows = [3, 5, 7, 10, 15, 25, 30, 45, 60, 90]
schedule_locations_game_data_to_process_team_strength = add_recent_game_counts(
    data=schedule_locations_game_data_to_process_team_strength,
    windows=windows
)

add_b2b_flag(schedule_locations_game_data_to_process_team_strength)
schedule_locations_game_data_to_process_team_strength = add_fatigue_variants(data=schedu
add_4in6_flag(schedule_locations_game_data_to_process_team_strength)
add_travel_distance(schedule_locations_game_data_to_process_team_strength)
add_days_since_last(schedule_locations_game_data_to_process_team_strength)
add_date_features(schedule_locations_game_data_to_process_team_strength)

schedule_locations_game_data_to_process_team_strength = add_win_features_fixed(
    data=schedule_locations_game_data_to_process_team_strength,
    rolling_windows=windows
```

```
)


schedule_locations_game_data_to_process_team_strength.fillna(0, inplace=True)
```

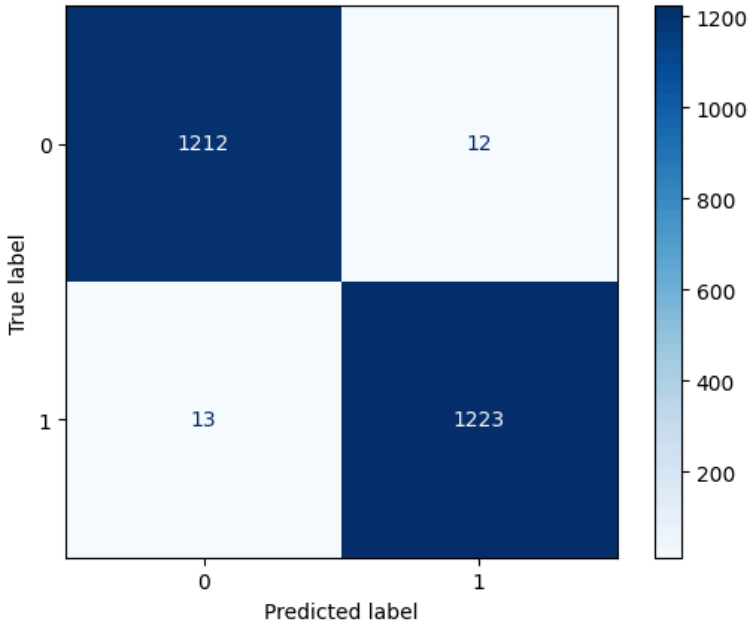
In [25]:
```
architecture_dictionary = {
    'hidden_layers': [125],
    'hidden_activation': 'relu',
    'output_activation': 'sigmoid',
    'hidden_l2_strength': 0.01,
    'output_l2_strength': 0.01,
    'learning_rate': 0.01,
    'validation_split': 0.1,
    'epochs': 100,
    'batch_size': 14500,
    'patience': 5,
    'dropout' : 0.1
}



_, _ = train_visualize_diagnose( schedule_locations_game_data_to_process_team_strength,
```

Model: "sequential_1"

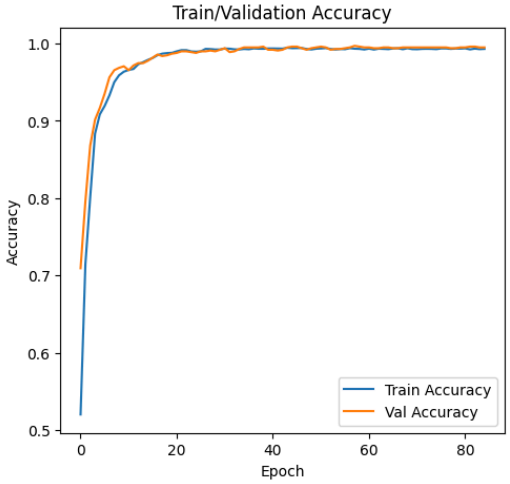
Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 125)	19,250
dropout_3 (Dropout)	(None, 125)	0
dense_5 (Dense)	(None, 1)	126

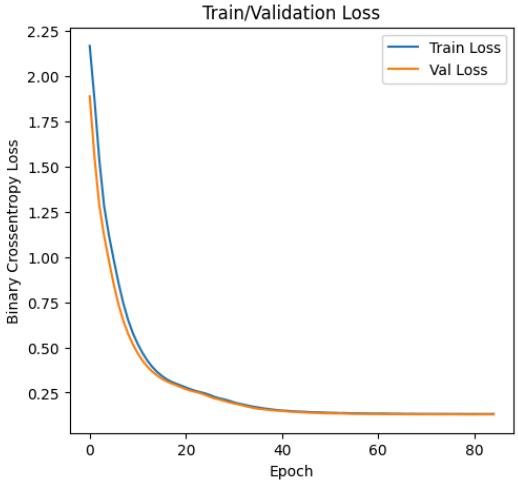
Total params: 58,130 (227.07 KB)
Trainable params: 19,376 (75.69 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 38,754 (151.39 KB)

```
========== Evaluations on internal test data ==========
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



```
Accuracy: 0.990, Precision: 0.990, Recall: 0.989, F1: 0.990
========== Estimating win/loss gain on post-2020 data ==========
365/365 ━━━━━━━━━━━━━━━━━━━━ 0s 945us/step
team
UTA    5
DAL    5
ORL    5
LAL    4
PHI    4
SAC    3
DET    3
OKC    3
MIL    3
ATL    1
PHX    1
NYK    1
MIN    1
WAS    1
DEN    0
CHA    0
MIA    0
LAC   -1
NOP   -1
GSW   -1
CLE   -1
SAS   -1
MEM   -2
IND   -2
POR   -2
BOS   -2
BKN   -3
HOU   -3
CHI   -3
TOR   -4
Name: win_gain_loss, dtype: int64
```





ANSWER 9:

- Most Helped by Schedule: UTA (5 wins)
- Most Hurt by Schedule: TOR (-4 wins)

Model Diagnostic:

Model architecture set up

- For this project, the final features combine schedule-related metrics with team–opponent on-court strength.
- I experimented with two model architectures using different feature sets:
- Model 1: A deeper neural network trained only on schedule-related features.
- Architecture: Input → 60 → 40 → 60 → 1

- Performance: Accuracy = 0.652, Precision = 0.653, Recall = 0.657, F1 = 0.655
- Model 2: A logistic regression model trained on both schedule and strength features.
- Architecture: Input → 1
- Performance: Accuracy = 0.980, Precision = 0.983, Recall = 0.978, F1 = 0.981
- To improve convergence, I added a hidden layer with 125 units:
- Architecture: Input → 125 → 1
- Performance: Accuracy = 0.990, Precision = 0.989, Recall = 0.990, F1 = 0.990

Common setup across models:

- ReLU activation in hidden layers
- L2 regularization (α = 0.01) on hidden and output layers
- Dropout rate of 0.1 to prevent overfitting
- Validation split of 0.1
- Early stopping with patience = 5
- In the end, I chose Model 2 with the 125-unit hidden layer.
- It converged faster, achieved near-perfect metrics, and followed Occam's Razor by balancing simplicity with high generalization performance.

Models Diagnosis

- Error Analysis
- Confusion matrices were generated for each model on the test set.
- Model 1 struggled, with approximately 65% accuracy across training, validation, and test sets. This indicates poor generalization. Likely reasons include limited data and insufficiently rich features. I did notice that increasing the windows size for rolling win features improves performance. Visually, I can in infer that scheduling features is related to performance
- Model 2 (both variations) performed very well, achieving approximately 99% accuracy, precision, recall, and F1 on the test set. Errors were minimal and did not show systematic bias toward one class.

Learning Curves

- Model 1 showed flat learning curves, with both training and validation accuracy plateauing around 65%. This suggests underfitting, where the model lacks the capacity or relevant features to capture signal from the data.
- Model 2 displayed healthy learning curves, with training and validation performance converging smoothly at approximately 99%, suggesting good fit and no signs of overfitting.
- Sensitivity to Hyperparameters
- Model 1 was unstable; for instance, increasing dropout significantly altered performance, indicating sensitivity to hyperparameter tuning.
- Model 2 was more robust, maintaining strong performance under moderate changes to dropout, learning rate, or hidden layer size.

Generalization

- Model 1 failed to generalize, as evidenced by its low and consistent accuracy across all sets.

- Model 2 (both variants) generalized extremely well, with minimal performance drop from training to validation and test data.

index	Team	Estimated Win/Gain
1	UTA	5
2	DAL	5
3	ORL	5
4	LAL	4
5	PHI	4
6	SAC	3
7	DET	3
8	OKC	3
9	MIL	3
10	ATL	1
11	PHX	1
12	NYK	1
13	MIN	1
14	WAS	1
15	DEN	0
16	CHA	0
17	MIA	0
18	LAC	-1
19	NOP	-1
20	GSW	-1
21	CLE	-1
22	SAS	-1
23	MEM	-2
24	IND	-2
25	POR	-2
26	BOS	-2
27	BKN	-3
28	HOU	-3
29	CHI	-3
30	TOR	-4