Movie Watching Habits - CS 210 Project

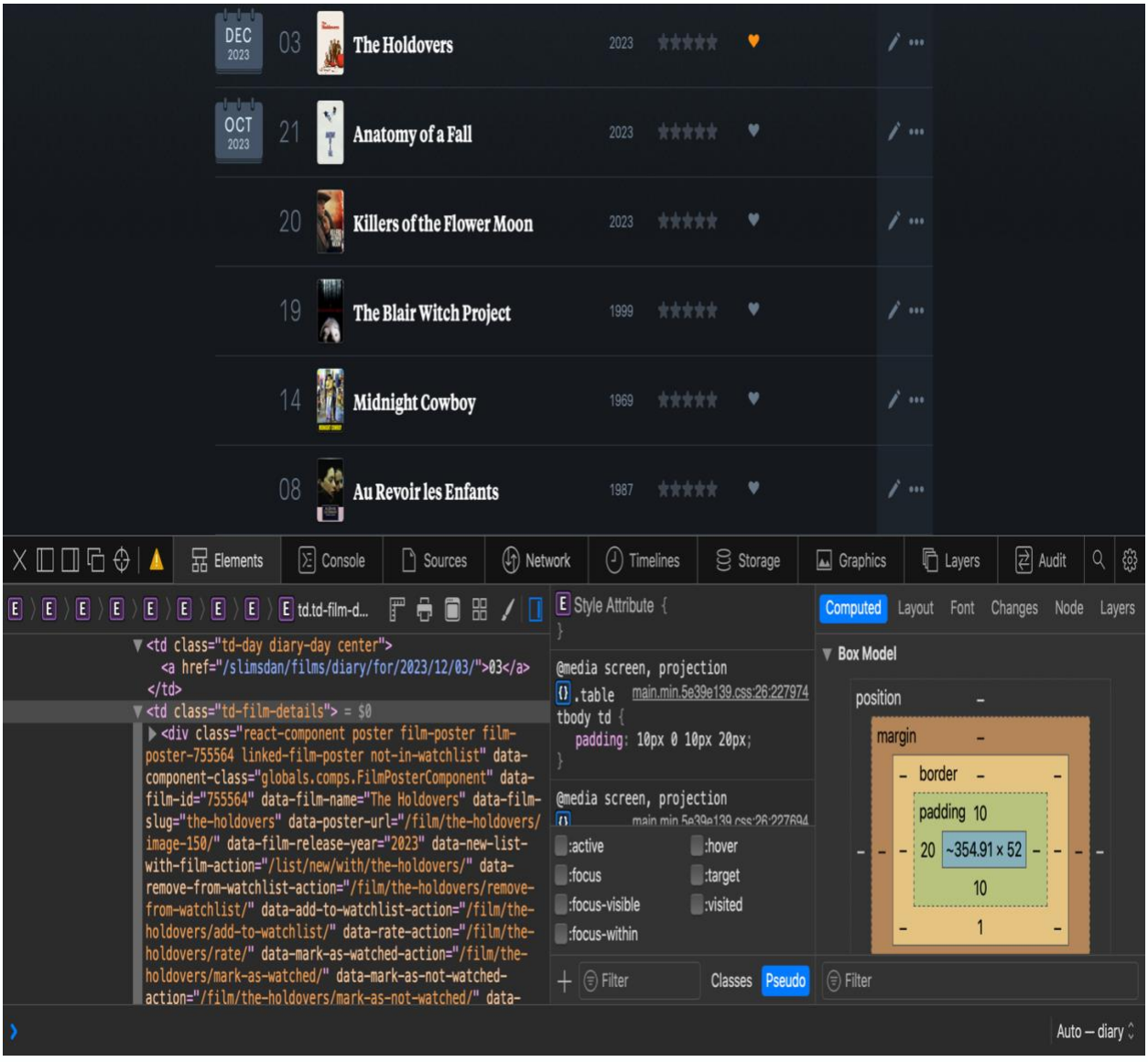
Selim Sıdan

INTRODUCTION

As a regular movie watcher, I've been aware of my movie watching habits changing drastically in different periods of the year, yet I wasn't sure about what was causing those changes and if there were some factors I wasn't even aware of. Luckily, I've been keeping track of all the movies I've watched since I started Sabancı University on Letterboxd, making the data of my diary in Letterboxd ideal to explore the effects of periods like fall/spring semesters and winter/summer breaks. So, I've added these periods and various important date ranges such as holidays in recurring locations to my dataset. My motivation for this project is to find the relationships between the number of movies watched, lengths of movies, time of the year and location while using this dataset. In order to do so I'll be using exploratory data analysis techniques such as correlation matrices, hypothesis testing, data visualization with matplotlib library and machine learning techniques such as linear regression, random forest with hyperparameter tuning to make accurate predictions.

DATA SOURCE & COLLECTION

In order to analyze my movie watching habits I've used the data from my movie diary on my Letterboxd account in which I've been keeping track of all the movies I've watched since I've started university. The data I've used covers the period from October 2021 to December 2023. I've scraped this data from my diary which consists of individual logs for each movie with their specific dates.



I’ve written a function to repeat this process of finding movie names and their dates for each log in each page of the diary. So, with this function I’ve collected these movies’ names and dates they have been watched.

```
In [2]: def extract_filtered_lines(url):
            response = requests.get(url)

            if response.status_code == 200:
                soup = BeautifulSoup(response.content, 'html.parser')

                lines_list = []

                links = soup.find_all('a')

                for link in links:
                    href = link.get('href')
                    lines_list.append(href)

                filtered_lines = [line for line in lines_list
                                  if line.startswith('/slimsdan/films/diary/for/')
                                  or line.startswith('/slimsdan/film/')]

                return filtered_lines

            else:
                print(f"Failed to fetch the webpage. Status code: {response.status_code}")
                return None

        urls = [
            'https://letterboxd.com/slimsdan/films/diary/page/1/',
            'https://letterboxd.com/slimsdan/films/diary/page/2/',
            'https://letterboxd.com/slimsdan/films/diary/page/3/',
            'https://letterboxd.com/slimsdan/films/diary/page/4/',
            'https://letterboxd.com/slimsdan/films/diary/page/5/'
        ]

        all_filtered_lines = []
        for url in urls:
            filtered_lines = extract_filtered_lines(url)
            if filtered_lines:
                all_filtered_lines.extend(filtered_lines)

        if len(all_filtered_lines) >= 20:
            all_filtered_lines = all_filtered_lines[:-20]
        else:
            print("Not enough lines to remove 20 lines.")
```

After the initial scraping, number of movies per day was not specific enough with the length of movies having a high variance because of both shorter feature length movies and short movies. So, I've scraped the movie lengths from individual pages of each movie and added them to the dataset as well.

Scraping and Adding Movie Lengths

In [24]:
```
base_url = 'https://letterboxd.com/film/'
def format_movie_url(movie_name):
    formatted_name = movie_name.lower().replace(' ', '-')
    return base_url + formatted_name + '/'
movies_and_urls = [(movie, format_movie_url(movie)) for movie in df['Movie']]
```

In [25]:
```
def scrape_movie_length(movie_url):
    response = requests.get(movie_url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        length_element = soup.find('p', class_='text-link text-footer')
        if length_element:
            length_text = length_element.get_text(strip=True)
            length = ''.join(filter(str.isdigit, length_text))
            return int(length) if length else None
    return None
```

Then, to make comparisons based on my location during holidays (Istanbul or Izmir) I've fed additional data for summer and winter breaks with date ranges specifying the location. Then, I've combined previous data with these new date ranges to check for relationships and possible effects they have on each other.

```
date_ranges = [
    ('2021-10-02', '2021-12-26'),
    ('2021-12-27', '2022-01-04'),
    ('2022-01-05', '2022-01-21'),
    ('2022-01-22', '2022-02-06'),
    ('2022-02-07', '2022-02-27'),
    ('2022-02-28', '2022-06-10'),
    ('2022-06-11', '2022-06-23'),
    ('2022-06-24', '2022-07-22'),
    ('2022-07-23', '2022-08-10'),
    ('2022-08-11', '2022-09-08'),
    ('2022-09-09', '2022-10-02'),
    ('2022-10-03', '2023-01-06'),
    ('2023-01-07', '2023-01-14'),
    ('2023-01-15', '2023-01-26'),
    ('2023-01-27', '2023-02-26'),
    ('2023-02-26', '2023-05-31'),
    ('2023-06-01', '2023-06-11'),
    ('2023-06-12', '2023-06-21'),
    ('2023-06-22', '2023-07-30'),
    ('2023-07-31', '2023-08-14'),
    ('2023-08-15', '2023-08-30'),
    ('2023-08-31', '2023-10-01'),
    ('2023-10-02', '2023-12-22')
]
```

```
custom_names = [
    "21/22 Fall Lessons",
    "21/22 Fall Finals",
    "22 Winter Break (Istanbul)",
    "22 Winter Break (Izmir)",
    "22 Winter Break (Istanbul)",
    "21/22 Spring Lessons",
    "21/22 Spring Finals",
    "22 Summer Break (Izmir)",
    "22 Summer Break (Istanbul)",
    "22 Summer Break (Izmir)",
    "22 Summer Break (Istanbul)",
    "22/23 Fall Lessons",
    "22/23 Fall Finals",
    "23 Winter Break (Izmir)",
    "22 Winter Break (Istanbul)",
    "22/23 Spring Lessons",
    "22/23 Spring Finals",
    "23 Summer Break (Istanbul)",
    "23 Summer Break (Izmir)",
    "23 Summer Break (Istanbul)",
    "23 Summer Break (Izmir)",
    "23 Summer Break (Istanbul)",
    "23/24 Fall Lessons",
]
```

DATA ANALYSIS & FINDINGS

Preprocessing

a) Data Transformation and Integration

I've created a basic DataFrame of movies' names and their dates using the scraped data from my diary.

Dataframe of Movie Names and Dates

In [3]:
```
movie_data = []
current_date = None

for line in all_filtered_lines:
    if line.startswith('/slimsdan/films/diary/for/'):
        match = re.search(r'\d{4}/\d{2}/\d{2}', line)
        if match:
            current_date = match.group()
    elif line.startswith('/slimsdan/film/'):
        parts = line.split('/')
        movie_name = next((p for p in parts[3:] if p and not p.isdigit()), None)
        if movie_name:
            movie_name = movie_name.replace('-', ' ')
            movie_data.append((movie_name, current_date))


df = pd.DataFrame(movie_data, columns=['Movie', 'Watch Date'])

df.head()
```

	Movie	Watch Date
0	the holdovers	2023/12/03
1	anatomy of a fall	2023/10/21
2	killers of the flower moon	2023/10/20
3	the blair witch project	2023/10/19
4	midnight cowboy	2023/10/14

Then I've generated another DataFrame using the dates I've provided for specific periods. This one was consisting of number of movies watched in total and per day for these specific periods of time.

Movies per Day for Specific Date Ranges

In [23]:

```
df['Watch Date'] = pd.to_datetime(df['Watch Date'])
date_ranges = [
    ('2021-10-02', '2021-12-26'),
    ('2021-12-27', '2022-01-04'),
    ('2022-01-05', '2022-01-21'),
    ('2022-01-22', '2022-02-06'),
    ('2022-02-07', '2022-02-27'),
    ('2022-02-28', '2022-06-10'),
    ('2022-06-11', '2022-06-23'),
    ('2022-06-24', '2022-07-22'),
    ('2022-07-23', '2022-08-10'),
    ('2022-08-11', '2022-09-08'),
    ('2022-09-09', '2022-10-02'),
    ('2022-10-03', '2023-01-06'),
    ('2023-01-07', '2023-01-14'),
    ('2023-01-15', '2023-01-26'),
    ('2023-01-27', '2023-02-26'),
    ('2023-02-26', '2023-05-31'),
    ('2023-06-01', '2023-06-11'),
    ('2023-06-12', '2023-06-21'),
    ('2023-06-22', '2023-07-30'),
    ('2023-07-31', '2023-08-14'),
    ('2023-08-15', '2023-08-30'),
    ('2023-08-31', '2023-10-01'),
    ('2023-10-02', '2023-12-22')
]
results = []
for start_date, end_date in date_ranges:
    filtered_data = df.loc[(df['Watch Date'] >= start_date) & (df['Watch Date'] <= end_date)]
    total_movies_watched = len(filtered_data)

    num_days = (pd.to_datetime(end_date) - pd.to_datetime(start_date)).days + 1
    movies_per_day = total_movies_watched / num_days if num_days != 0 else 0

    results.append({
        'Date Range': f'{start_date} to {end_date}',
        'Total Movies Watched': total_movies_watched,
        'Movies Watched per Day': movies_per_day
    })

results_df = pd.DataFrame(results)
print(results_df)
results_df.head()
```

	Date Range	Total Movies Watched	Movies Watched per Day
0	2021-10-02 to 2021-12-26	46	0.534884
1	2021-12-27 to 2022-01-04	0	0.000000
2	2022-01-05 to 2022-01-21	18	1.058824
3	2022-01-22 to 2022-02-06	13	0.812500
4	2022-02-07 to 2022-02-27	6	0.285714

Then, with the scraped data of each movie from their individual pages I've gathered their lengths as I've explained previously. So, with this new data I've created a third DataFrame and finally I've integrated them to create a final merged one.

In [24]:
```
base_url = 'https://letterboxd.com/film/'
def format_movie_url(movie_name):
    formatted_name = movie_name.lower().replace(' ', '-')
    return base_url + formatted_name + '/'
movies_and_urls = [(movie, format_movie_url(movie)) for movie in df['Movie']]
```

In [25]:
```
def scrape_movie_length(movie_url):
    response = requests.get(movie_url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        length_element = soup.find('p', class_='text-link text-footer')
        if length_element:
            length_text = length_element.get_text(strip=True)
            length = ''.join(filter(str.isdigit, length_text))
            return int(length) if length else None
    return None
movie_lengths = []
number = 1

for movie, url in movies_and_urls:
    length = scrape_movie_length(url)

    movie_lengths.append((movie, length))

df_movie_lengths = pd.DataFrame(movie_lengths, columns=['Movie', 'Length_Minutes'])
df_movie_lengths.head()
```

	Movie	Length_Minutes
0	the holdovers	133
1	anatomy of a fall	152
2	killers of the flower moon	206
3	the blair witch project	81
4	midnight cowboy	113

In [26]:
```
merged_df = pd.merge(df, df_movie_lengths, on='Movie', how='inner')
movie_counts = merged_df.groupby('Movie').size()
merged_df.head()
```

	Movie	Watch Date	Length_Minutes
0	the holdovers	2023-12-03	133
1	anatomy of a fall	2023-10-21	152
2	killers of the flower moon	2023-10-20	206
3	the blair witch project	2023-10-19	81
4	midnight cowboy	2023-10-14	113

b) Data Imputation

The data I've scraped and collected was only consisting of days in which a movie has been watched and the days without a movie were problematic as they were giving NaN values. In order to make the data continuous for each day and period I've replaced days without a movie watched to have 0 for both minutes and number of movies for that day. So, in the end I had data consisting of continuous time series both for days and periods.

```
In [35]: merged_df['Watch Date'] = pd.to_datetime(merged_df['Watch Date'])

         resampled_data = merged_df.resample('7D', on='Watch Date')['Length_Minutes'].sum()
         num_movies_per_week = merged_df.resample('7D', on='Watch Date').size()

         avg_length_per_week = resampled_data / num_movies_per_week

         # turn NaN values into 0 as they are the days and periods without a movie watched
         avg_length_per_week = avg_length_per_week.fillna(0)

         avg_length_list = avg_length_per_week.tolist()

         total_minutes_per_week = merged_df.resample('D', on='Watch Date')['Length_Minutes'].sum().resam

         num_days_per_week = merged_df.resample('D', on='Watch Date').size().resample('7D').count()

         minutes_per_day_per_week = total_minutes_per_week / num_days_per_week

         minutes_per_day_list = minutes_per_day_per_week.tolist()
```

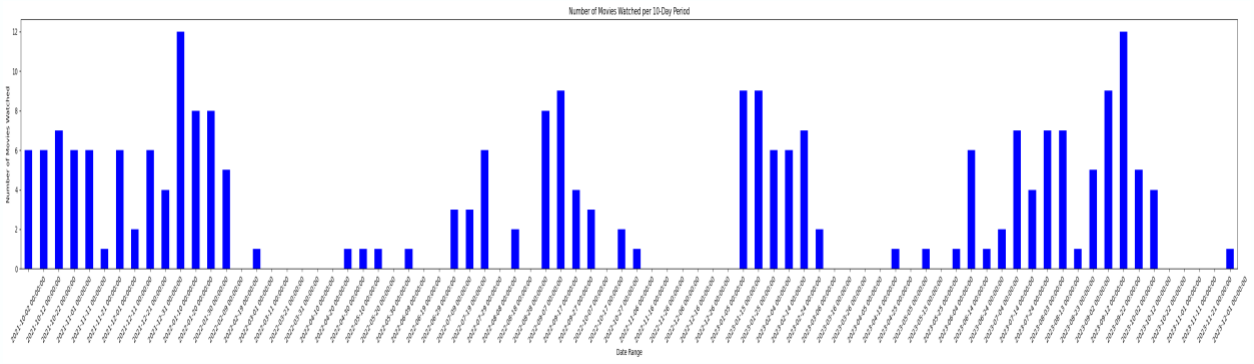
Exploratory Data Analysis

a) Bar Charts

Using the DataFrames, I've created several bar charts to see the changes over time whether it is days or specific periods. I've initially used the number of movies watched per day to evaluate my routines and created a bar chart of number of movies watched for 10-day periods. Even though certain periods could easily be detected as some had 0 or close to 0 it was not specific enough and too crowded.

Total Number of Movies Watched per 10-Day Periods

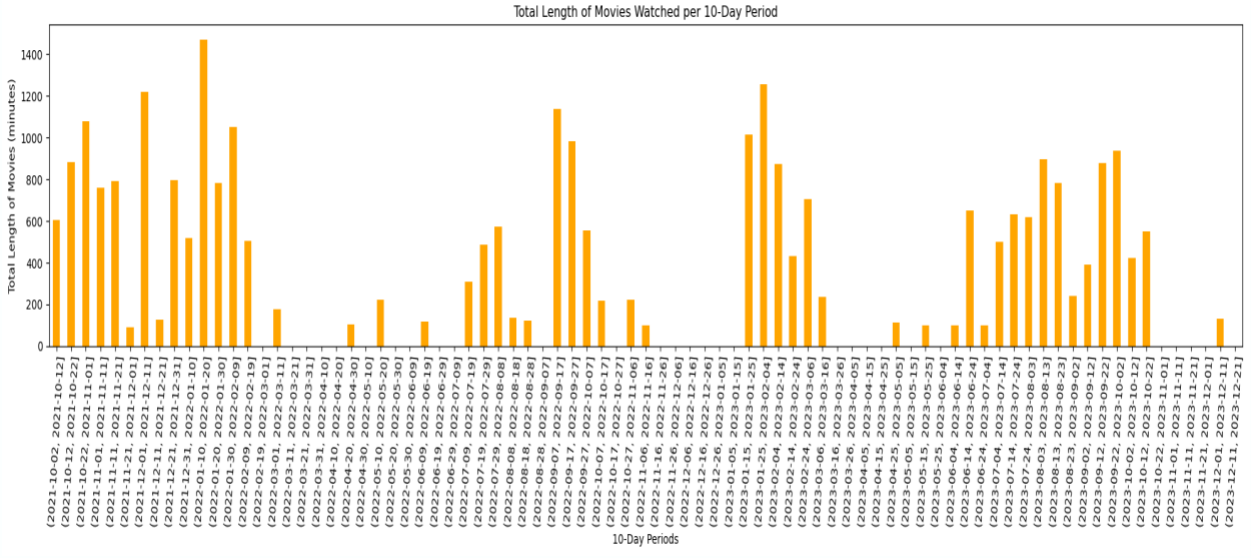
```
In [27]: df['Watch Date'] = pd.to_datetime(df['Watch Date'])

         start_date = '2021-10-02'
         end_date = '2023-12-22'
         filtered_data = df.loc[(df['Watch Date'] >= start_date) & (df['Watch Date'] <= end_date)]

         movies_per_day = filtered_data.groupby('Watch Date').size()

         movies_per_week = movies_per_day.resample('10D').sum()

         plt.figure(figsize=(40, 6))
         movies_per_week.plot(kind='bar', color='blue')
         plt.title('Number of Movies Watched per 10-Day Period')
         plt.xlabel('Date Range')
         plt.ylabel('Number of Movies Watched')
         plt.xticks(rotation=45)
         plt.tight_layout()
         plt.show()
```



Then, I've used the lengths of movies rather than number of them to make the chart more precise as the lengths can vary from 12 minutes to 228 minutes.

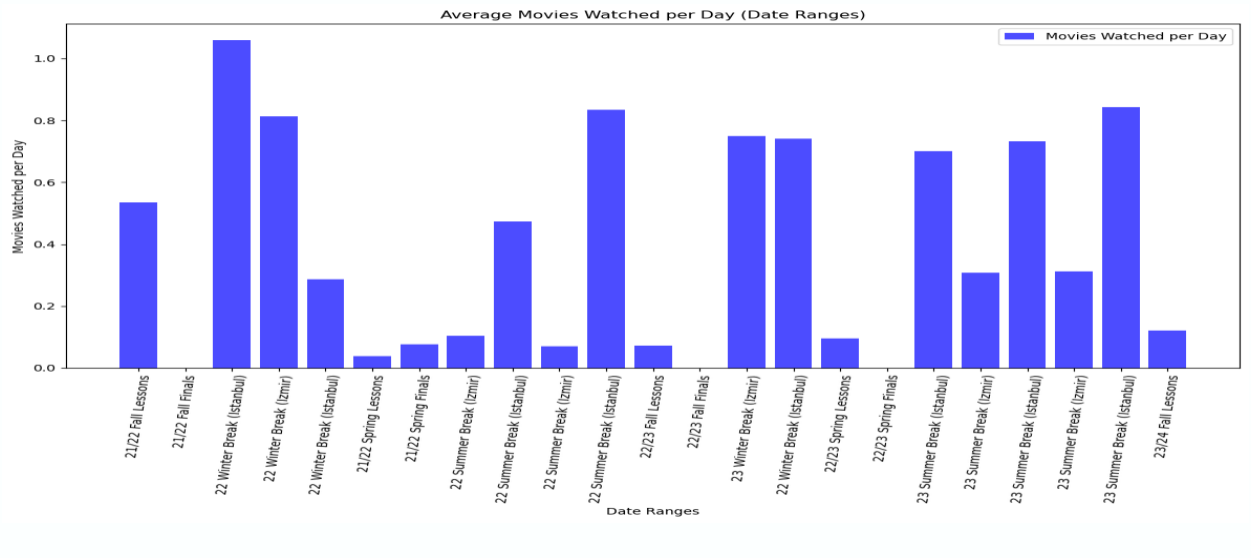
Total Minutes Watched for 10-Day Periods

```
In [28]: merged_df['Watch Date'] = pd.to_datetime(merged_df['Watch Date'])

         start_date = '2021-10-02'
         end_date = '2023-12-22'

         date_range = pd.date_range(start=start_date, end=end_date, freq='10D')

         length_per_period = merged_df.groupby(pd.cut(merged_df['Watch Date'], bins=date_range))['Length_Minut

         plt.figure(figsize=(20, 6))
         length_per_period.plot(kind='bar', color='orange')
         plt.title('Total Length of Movies Watched per 10-Day Period')
         plt.xlabel('10-Day Periods')
         plt.ylabel('Total Length of Movies (minutes)')
         plt.xticks(rotation=85)
         plt.tight_layout()
         plt.show()
```

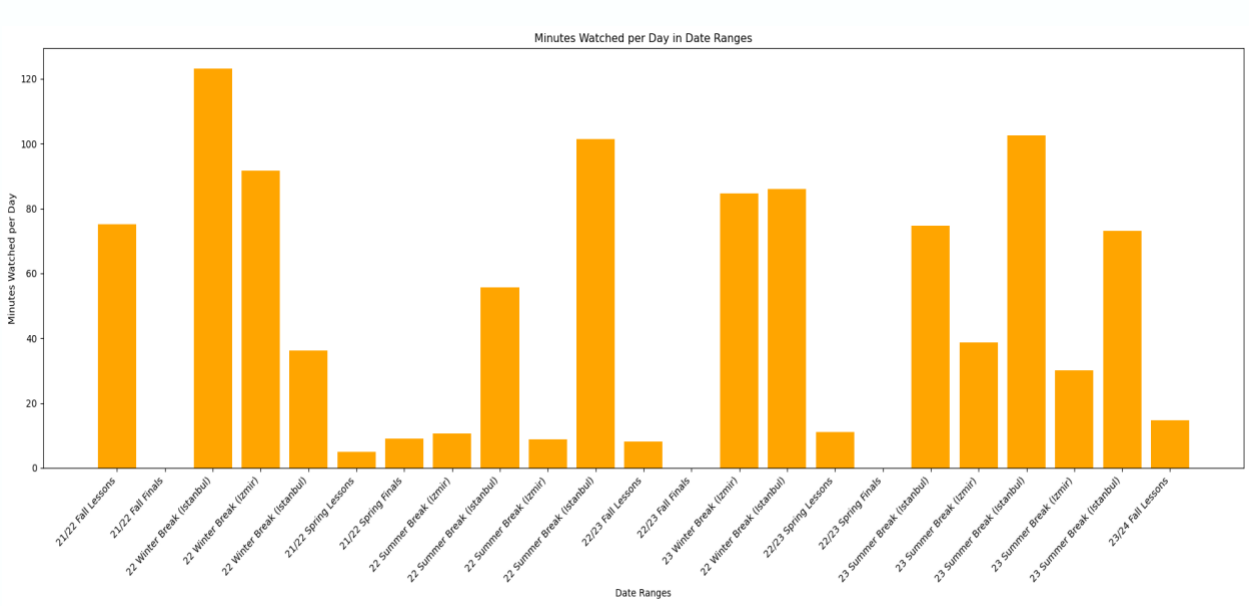


I've created another 2 charts based on the specific periods I've provided. I've named these date ranges indicating their period and location. Looking at these charts, the significant difference between semesters and breaks became obvious as for semesters it was always very close to 0. There was also remarkable increase in both winter and summer breaks. In these bar charts I've once again used number of movies and minutes of movies watched per day.

```
In [29]: custom_names = [
             "21/22 Fall Lessons",
             "21/22 Fall Finals",
             "22 Winter Break (Istanbul)",
             "22 Winter Break (Izmir)",
             "22 Winter Break (Istanbul)",
             "21/22 Spring Lessons",
             "21/22 Spring Finals",
             "22 Summer Break (Izmir)",
             "22 Summer Break (Istanbul)",
             "22 Summer Break (Izmir)",
             "22 Summer Break (Istanbul)",
             "22/23 Fall Lessons",
             "22/23 Fall Finals",
             "23 Winter Break (Izmir)",
             "22 Winter Break (Istanbul)",
             "22/23 Spring Lessons",
             "22/23 Spring Finals",
             "23 Summer Break (Istanbul)",
             "23 Summer Break (Izmir)",
             "23 Summer Break (Istanbul)",
             "23 Summer Break (Izmir)",
             "23 Summer Break (Istanbul)",
             "23/24 Fall Lessons",
         ]

         plt.figure(figsize=(12, 8))
         plt.bar(range(len(results_df)), results_df['Movies Watched per Day'], color='blue', alpha=0.7, la
         plt.title('Movies Watched per Day in Date Ranges')
         plt.xlabel('Date Ranges')
         plt.ylabel('Movies Watched per Day')
         plt.xticks(range(len(results_df)), custom_names, rotation=84)
         plt.legend()
         plt.tight_layout()
         plt.show()
```

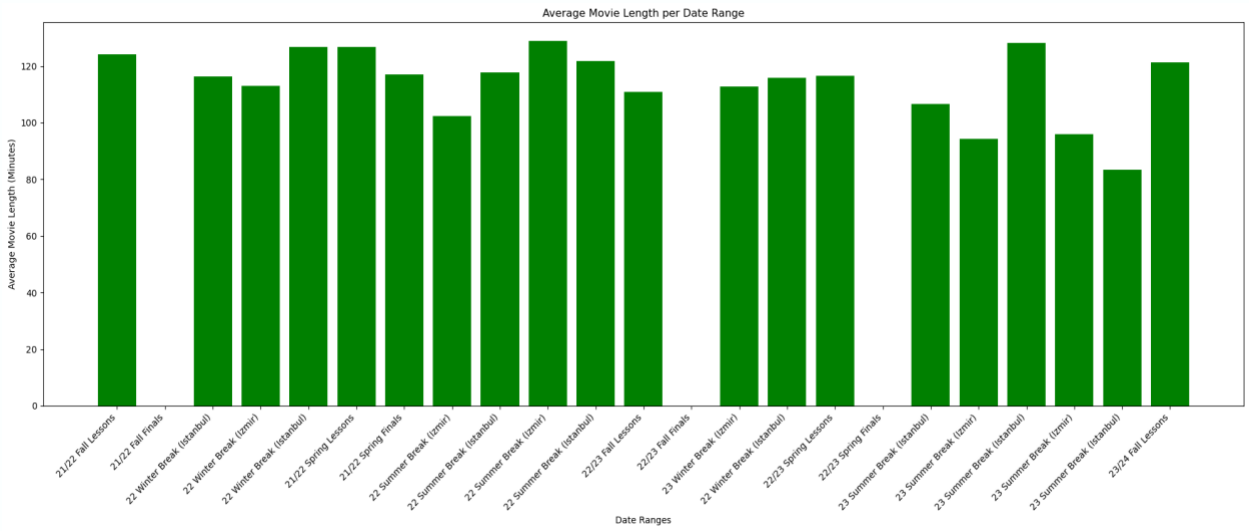


In [30]:
```
minutes_per_day_per_range = []

for start_date, end_date in date_ranges:
    start = pd.to_datetime(start_date)
    end = pd.to_datetime(end_date)

    filtered_data = merged_df.loc[(merged_df['Watch Date'] >= start) & (merged_df['Watch Date'] <= end)

    total_minutes = filtered_data['Length_Minutes'].sum()
    days_count = (end - start).days + 1

    if days_count > 0:
        minutes_per_day = total_minutes / days_count
    else:
        minutes_per_day = 0

    minutes_per_day_per_range.append(minutes_per_day)

plt.figure(figsize=(20, 8))
plt.bar(range(len(date_ranges)), minutes_per_day_per_range, color='orange')
plt.title('Minutes Watched per Day in Date Ranges')
plt.xlabel('Date Ranges')
plt.ylabel('Minutes Watched per Day')
plt.xticks(range(len(date_ranges)), custom_names, rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Finally, I've created a bar chart based on the average lengths of movies for each specific date range. I was hoping to see whether there was a relationship between minutes watched per day and the average movie lengths, yet this chart was not enough to draw any conclusions.

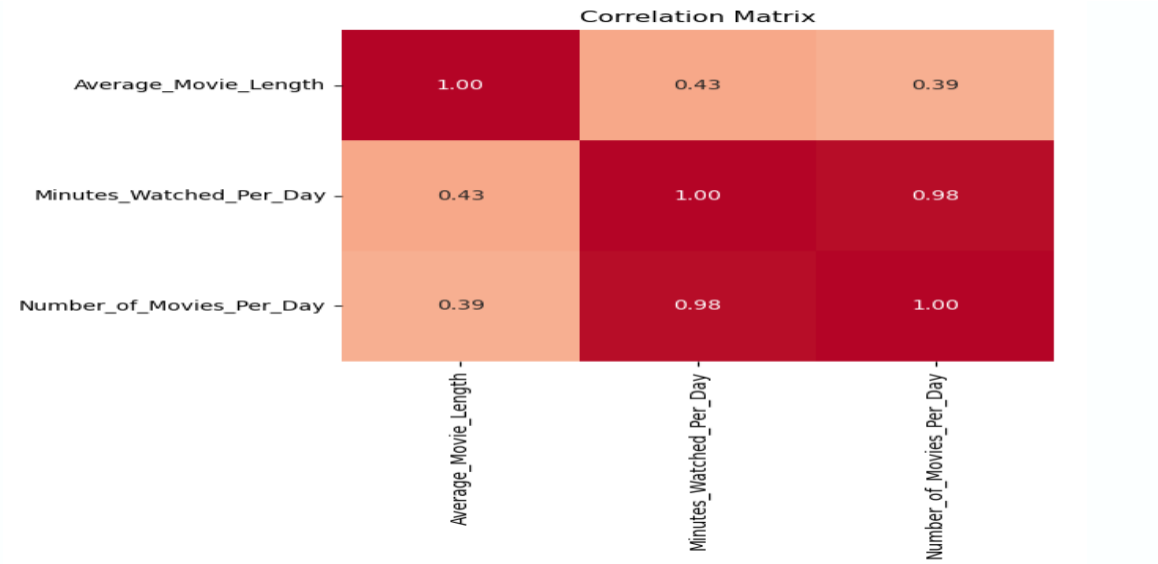
```
In [31]: avg_length_per_range = []

         for i in range(len(date_ranges)):
             start_date, end_date = date_ranges[i]
             start = pd.to_datetime(start_date)
             end = pd.to_datetime(end_date)

             filtered_data = merged_df.loc[(merged_df['Watch Date'] >= start) & (merged_df['Watch Date'] <= end)

             total_minutes = filtered_data['Length_Minutes'].sum()
             total_movies = filtered_data.shape[0]

             if total_movies > 0:
                 avg_length = total_minutes / total_movies
             else:
                 avg_length = 0

             avg_length_per_range.append(avg_length)

         plt.figure(figsize=(20, 8))
         plt.bar(range(len(date_ranges)), avg_length_per_range, color='green')
         plt.title('Average Movie Length per Date Range')
         plt.xlabel('Date Ranges')
         plt.ylabel('Average Movie Length (Minutes)')
         plt.xticks(range(len(date_ranges)), custom_names, rotation=45, ha='right')
         plt.tight_layout()
         plt.show()
```



b) Correlation Matrix

Continuing from the last bar chart I've created, in order to analyze the relationship between average length of movies and number of movies watched I've generated a correlation matrix. According to the matrix, correlation between average movie length and number of movies is 0.39 and for minutes watched it is 0.43 thus implying that both correlations are moderate.

```
In [33]: movies_per_day = results_df['Movies Watched per Day']

         data = {
             'Average_Movie_Length': avg_length_per_range,
             'Minutes_Watched_Per_Day': minutes_per_day_per_range,
             'Number_of_Movies_Per_Day': movies_per_day
         }

         df = pd.DataFrame(data, columns=['Average_Movie_Length', 'Minutes_Watched_Per_Day', 'Number_of_Movies

         correlation_matrix = df.corr()

         print("Correlation Matrix:")
         print(correlation_matrix)

         corr_matrix = df.corr()

         plt.figure(figsize=(8, 6))
         sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
         plt.title('Correlation Matrix')
         plt.tight_layout()
         plt.show()
```



Hypothesis Test #1

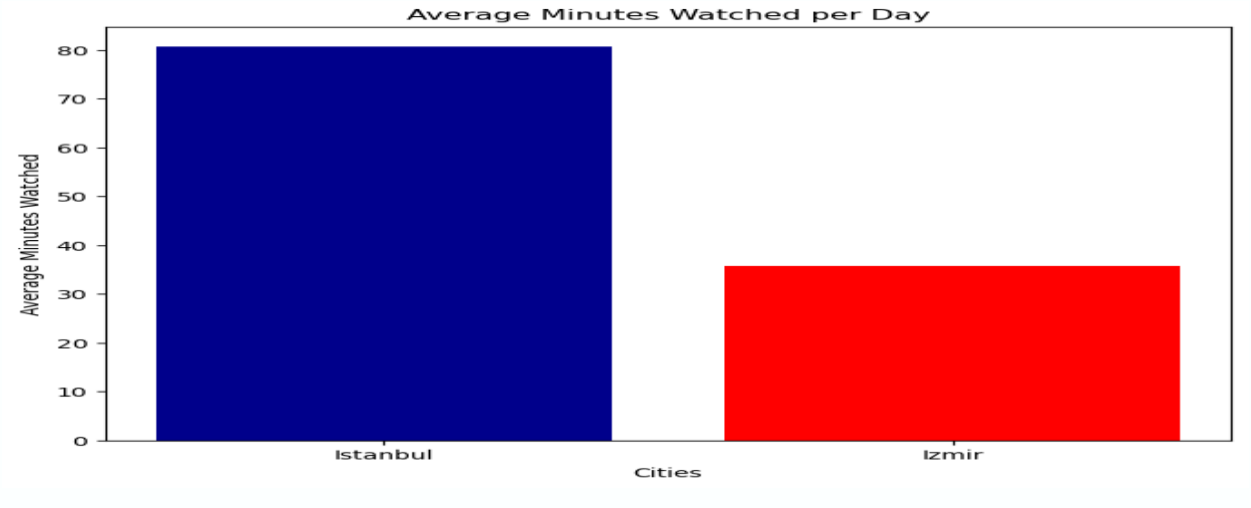
Null Hypothesis: Location (Istanbul, Izmir) has no effect on minutes watched per day.

I have specified the date ranges in Istanbul and Izmir for the hypothesis testing with the significance value = 0.05 as usual. According to the calculations p-value = 1.4e-7 and since it is smaller than 0.05 my null hypothesis is rejected. This indicates that location has a significant impact on minutes watched per day.

```
istanbul_days, izmir_days = [], []
istanbul_minutes, izmir_minutes = [], []

for (start_date, end_date), name in zip(date_ranges, custom_names):
    start = datetime.strptime(start_date, "%Y-%m-%d")
    end = datetime.strptime(end_date, "%Y-%m-%d")

    total_days = (end - start).days + 1

    if name.endswith("(Istanbul)"):
        istanbul_days.extend(pd.date_range(start=start, end=end))
        istanbul_filtered_data = merged_df.loc[(merged_df['Watch Date'] >= start) & (merged_df['Watch Date'] <= end)]
        istanbul_daily_minutes = istanbul_filtered_data.groupby('Watch Date')['Length_Minutes'].sum().reindex(pd.date_range(start=start, end=end), fill_value=0).tolist()
        istanbul_minutes.extend(istanbul_daily_minutes)
    elif name.endswith("(Izmir)"):
        izmir_days.extend(pd.date_range(start=start, end=end))
        izmir_filtered_data = merged_df.loc[(merged_df['Watch Date'] >= start) & (merged_df['Watch Date'] <= end)]
        izmir_daily_minutes = izmir_filtered_data.groupby('Watch Date')['Length_Minutes'].sum().reindex(pd.date_range(start=start, end=end), fill_value=0).tolist()
        izmir_minutes.extend(izmir_daily_minutes)

overall_minutes = istanbul_minutes + izmir_minutes

from scipy.stats import ttest_ind

t_statistic, p_value = ttest_ind(istanbul_minutes, izmir_minutes)

significance = 0.05

print("P-value is ", p_value)

if p_value < significance:
    print("Reject Null Hypothesis: Location has a significant effect on minutes watched.")
else:
    print("Accept Null Hypothesis: Location has no significant effect on minutes watched.")
```

I've also created a bar chart based on this comparison of Istanbul vs Izmir and the significant impact of location can also be understood from this chart.

```
In [32]: total_days_istanbul = 0
         total_days_izmir = 0

         for (start_date, end_date), name in zip(date_ranges, custom_names):
             start = datetime.strptime(start_date, "%Y-%m-%d")
             end = datetime.strptime(end_date, "%Y-%m-%d")

             total_days = (end - start).days + 1

             if name.endswith("(Istanbul)"):
                 total_days_istanbul += total_days
             elif name.endswith("(Izmir)"):
                 total_days_izmir += total_days

         total_minutes_per_range = []
         total_minutes_istanbul = 0
         total_minutes_izmir = 0

         for start_date, end_date in date_ranges:
             start = pd.to_datetime(start_date)
             end = pd.to_datetime(end_date)

             filtered_data = merged_df.loc[(merged_df['Watch Date'] >= start) & (merged_df['Watch Date'] <= end)]

             total_minutes = filtered_data['Length_Minutes'].sum()
             total_minutes_per_range.append(total_minutes)

             custom_name = custom_names[date_ranges.index((start_date, end_date))]
             if custom_name.endswith('(Istanbul)'):
                 total_minutes_istanbul += total_minutes
             elif custom_name.endswith('(Izmir)'):
                 total_minutes_izmir += total_minutes

         avg_minutes_istanbul = total_minutes_istanbul / total_days_istanbul
         avg_minutes_izmir = total_minutes_izmir / total_days_izmir

         categories = ['Istanbul', 'Izmir']
         avg_minutes = [avg_minutes_istanbul, avg_minutes_izmir]

         plt.figure(figsize=(8, 6))
         plt.bar(categories, avg_minutes, color=['darkblue', 'red'])
         plt.title('Average Minutes Watched per Day')
         plt.xlabel('Cities')
         plt.ylabel('Average Minutes Watched')
         plt.show()
```



Hypothesis Test #2

Null Hypothesis: Minutes watched per day has no effect on average lengths of movies (excluding periods with 0 movies watched).

I've tried to find the relationship between avg. lengths of movies and minutes watched per day before with correlation matrix and I've decided to test it again. However, this time I've only tested for the days and periods in which at least a movie has been seen. The reason for this exception was to find the true effect one has on another since 0 movies or minutes watched will obviously mean that avg. movie length for that period will also have to be 0. Yet this is a direct result caused by lack of information for that period, not an indication of their true relationship. So, I've set the significance value as 0.05 and the p-value resulted as 0.067. Since it wasn't less than the significance value, null hypothesis couldn't be rejected thus indicating that there is no significant relationship between minutes watched per day and avg. lengths of movies.

In [35]:
```
merged_df['Watch Date'] = pd.to_datetime(merged_df['Watch Date'])

resampled_data = merged_df.resample('7D', on='Watch Date')['Length_Minutes'].sum()
num_movies_per_week = merged_df.resample('7D', on='Watch Date').size()

avg_length_per_week = resampled_data / num_movies_per_week

# turn NaN values into 0 as they are the days and periods without a movie watched
avg_length_per_week = avg_length_per_week.fillna(0)
avg_length_list = avg_length_per_week.tolist()

total_minutes_per_week = merged_df.resample('D', on='Watch Date')['Length_Minutes'].sum().resample('7D').sum()
num_days_per_week = merged_df.resample('D', on='Watch Date').size().resample('7D').count()

minutes_per_day_per_week = total_minutes_per_week / num_days_per_week
minutes_per_day_list = minutes_per_day_per_week.tolist()
```

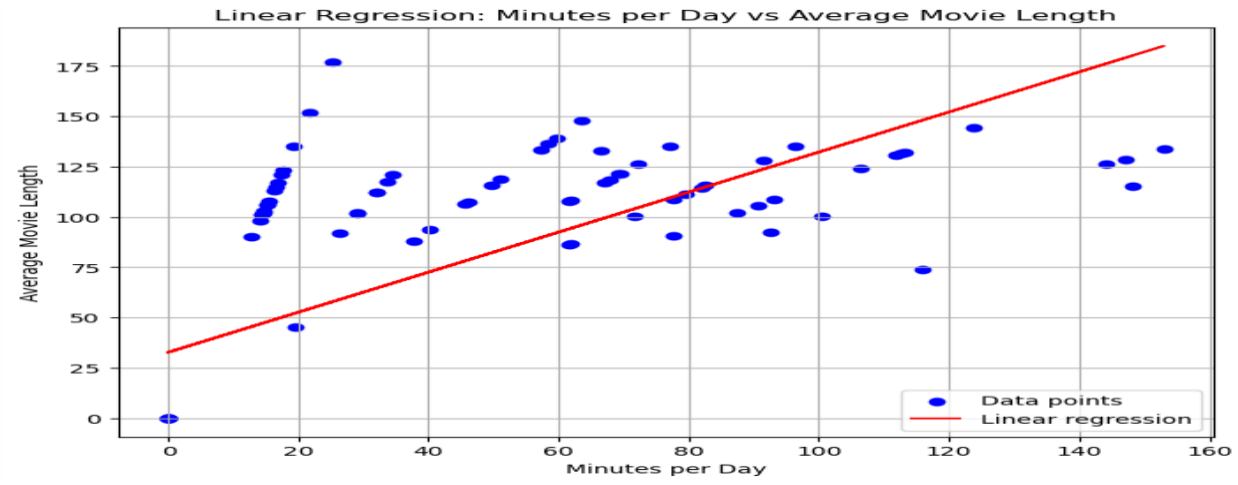
In [36]:
```
from scipy.stats import pearsonr

non_zero_minutes = [m for m in minutes_per_day_list if m != 0.0]
non_zero_lengths = [l for l in avg_length_list if l != 0.0]

corr, p_value = pearsonr(non_zero_minutes, non_zero_lengths)
alpha = 0.05

print(f"P-value: {p_value}")
if p_value < alpha:
    print("There is a significant relationship between non-zero Minutes watched per day and average length of movies.")
else:
    print("There is no significant relationship between non-zero Minutes watched per day and average length of movies.")
```

Machine Learning

a) Linear Regression

Continuing from the Hypothesis #2, I've created a linear regression model for the variables minutes per day and average movie length. As it can be seen below the linear regression model isn't very accurate and the results support this as well. R^2 score is 0.52 indicating a moderate accuracy and the Mean Squared Error is 1606 which is way too big for an accurate prediction.

```
In [37]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import r2_score, mean_squared_error

         X = np.array(minutes_per_day_list).reshape(-1, 1)
         y = np.array(avg_length_list)

         model = LinearRegression()

         model.fit(X, y)
         y_pred = model.predict(X)
         plt.figure(figsize=(8, 6))
         plt.scatter(X, y, color='blue', label='Data points')
         plt.plot(X, y_pred, color='red', label='Linear regression')
         plt.title('Linear Regression: Minutes per Day vs Average Movie Length')
         plt.xlabel('Minutes per Day')
         plt.ylabel('Average Movie Length')
         plt.legend()
         plt.grid(True)
         plt.show()

         r2 = r2_score(y, y_pred)
         mse = mean_squared_error(y, y_pred)
         print(f"R^2 score: {r2}")
         print(f"Mean Squared Error: {mse}")
```



b) Random Forest & Hyperparameter Tuning

In order to have an improved prediction I have used hyperparameter tuning and Random Forest in this step. I have performed grid search with cross-validation and have gotten the best hyperparameters from the grid search. Then, I have trained the model with these best hyperparameters: max_depth: 5, min_samples_leaf: 1, min_samples_split: 5, n_estimators: 200. Finally, I have created the random forest model, and the results have improved considerably with R^2 score 0.84 and Mean Squared Error 461.

In [38]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV


param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


forest = RandomForestRegressor(random_state=42)


grid_search = GridSearchCV(estimator=forest, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)


best_params = grid_search.best_params_


best_forest = RandomForestRegressor(**best_params, random_state=42)
best_forest.fit(X_train, y_train)


y_pred_best_forest = best_forest.predict(X_test)


mse_best = mean_squared_error(y_test, y_pred_best_forest)
r2_best = r2_score(y_test, y_pred_best_forest)

print("Best Hyperparameters:", best_params)
print(f"Mean Squared Error (MSE) with Best Hyperparameters: {mse_best}")
print(f"R-squared (R2) Score with Best Hyperparameters: {r2_best}")
```

Findings

1) The city I'm in during holidays, Istanbul or Izmir, has a significant impact on the number of movies I watch according to both the hypothesis test #1 and the comparative bar chart.

2) Average lengths of movies I watch are not affected by the average minutes of movies I watch per day. Before I began this project, I believed that there was a correlation, yet it has been proven both by the hypothesis test #2 and the correlation matrix I've presented above with a correlation rate of 0.42 that there is no significant relationship.

3) Linear regression model hasn't been able to make a good prediction for the relationship between average lengths of movies and minutes of movies watched per day. It can be clearly seen through the model and with the low R2 and high MSE scores. However, with the hyperparameter tuning using grid search with cross-validation random forest has been able to create a more accurate predictive model with significantly higher R2 and lower MSE scores.

4) The time of the year clearly affects the number of movies I watch as the bar charts both for individual days and for specific periods show drastic fluctuations. There was no need for additional analysis for this part as the differences between breaks and semesters were obvious as I almost never watch movies during semesters.

CONCLUSION

a) Summary

In this project I've scraped data from my movie diary on Letterboxd and those movies' individual pages to find relationships related to my movie watching habits. I've first created DataFrames, and bar charts based on them. From these I've observed that semesters and breaks have a significant effect on the number of movies I watch. Then, with hypothesis testing and comparative bar chart it became evident that I watch considerably more movies per day in Istanbul than Izmir. After that, through correlation matrix and hypothesis testing I've found that avg. length of movies and number of movies I watch don't have a serious relationship. Finally, I've used machine learning methods linear regression and random forest. Linear regression model hasn't resulted in an accurate model, whereas random forest model with hyperparameter tuning resulted in a superior predictive model that is accurate according to R2 and MSE scores.

b) Limitations & Future Work

A notable limitation was the unknown length of movie credits as they can vary from 5 to 15 minutes. They impact the minutes watched even though they aren't actually watched. However, it wasn't possible to make any assumptions since there was no data available for the movies I've watched. Ideally, they would've been disregarded from the minutes watched but the high variance and their occasional inclusion in the beginning or middle rather than the end of movies made it problematic. Another limitation was the restriction to my personal data and movie watching routines as this was an individual project. So, there was only certain factors that I could work with. If I was able to work with routines of other

students and their data, there could've been more interesting results. I could've analyzed other factors affecting number of movies watched such as courses taken, majors and GPAs. This is something that I want to revisit in the future to improve my project with more students and data. I would also like to further enhance it with the Goodreads data in a similar manner. Finally, I also hope to enrich my project in the future with different machine learning techniques after taking the ML course.