Boston Rental Property Recommendation Project

By: Krish Ruchir Patel

Date: December 4th 2024

Table of Content

1.0 Introduction

1.1 About the Project

This project focuses on analyzing rental property data in Boston to assist students attending Wentworth Institute of Technology (WIT) in finding affordable and convenient housing options. The analysis emphasizes two primary aspects:

1. **Rental Price:** Understanding how rental prices vary across ZIP codes, property sizes, and features like bedrooms and bathrooms.
2. **Travel Convenience:** Evaluating travel times to WIT by walking, public transit, and driving, and their relationship with rental costs.

Ultimately, the project aims to leverage these insights to create a recommendation system that helps students select optimal rental properties tailored to their preferences and constraints.

1.2 Questions to Address

The following key questions will be addressed throughout this project:

1. How do travel times by walking, transit, and driving influence rental prices?
2. What is the impact of property characteristics on rental prices?
3. How do rental prices vary by location?
4. How can affordability and convenience be balanced?
5. Can a machine learning-based recommendation system identify the best properties based on user preferences and current listings?

1.3 Project Objectives

The objectives of this project are as follows based on the questions:

1. Gather rental property data along with travel distances in different modes (walking, transit, and driving) to Wentworth Institute of Technology (WIT).
2. Investigate how travel times by walking, transit, and driving influence rental prices and explore trends to understand the trade-offs between convenience and affordability.
3. Examine the impact of property characteristics, such as price per square foot, bedrooms, and bathrooms, on rental affordability and desirability.
4. Analyze rental price variations across ZIP codes and identify neighborhoods that balance affordability and travel convenience for Wentworth students.
5. Use insights from the analysis to build and test a recommendation model tailored to current property listings, optimizing for user preferences and constraints.
6. Assist in identifying the best rental properties that balance affordability, convenience, and individual preferences through data-driven insights and machine learning.

1.4 Methods

This project follows a structured workflow to address the objectives and build a recommendation model effectively. The steps and methods involved are as follows:

Data Preparation:

1. **Data Collection:**
    - Using the **HomeHarvest Library** to gather current rental property listings in Boston.
    - Employing the **Google Distance Matrix API** to retrieve travel distances (walking, transit, and driving) from each property to Wentworth Institute of Technology (WIT).
2. **Data Cleaning:**
    - Dropping unnecessary columns that do not contribute to the analysis.
    - Handling missing values and ensuring proper data types for smooth analysis and modeling.
3. **Feature Engineering:**
    - Creating additional columns such as `neighborhood_travel_score`, `price_per_min_walk`, `price_per_sqft`, and `bed_bath_ratio` to enhance the analysis.
    - Engineering features to address key questions and derive meaningful insights.

Data Analysis:

4. **Exploratory Data Analysis (EDA):**
    - Answering all questions and sub-questions identified in the objectives using:

    - **Statistical Analysis** to identify trends and correlations.
    - **Visualization Techniques** such as boxplots, bar charts, scatter plots, and heatmaps to understand data distributions and relationships.

Machine Learning:

5. **Building the Recommendation Model:**
    - Utilizing **Random Forest Regressor** to build and train the recommendation model.
    - Evaluating the model using metrics like Root Mean Squared Error (RMSE).
    - Analyzing **feature importance** to understand the most influential factors in predicting rental prices.
    - Filtering and recommending properties based on user-defined constraints like price, travel time, and other preferences.

Tools and Methods:

- **Google API** for collecting travel distance data.
- **EDA techniques** for answering analysis questions and identifying trends.
- **Random Forest Regressor** for machine learning model development and feature analysis.
- **Visualization Libraries** (e.g., Matplotlib, Seaborn) for understanding feature relationships and presenting insights.

Link(s) or collecting method of datasets:

- HomeHarvest Library: Collects real-time rental property data in Boston, including details such as price, property type, and availability. https://github.com/Bunsly/HomeHarvest
- Google Distance Matrix API: Calculates walking, transit, and car travel times from each rental property to Wentworth Institute of Technology. https://developers.google.com/maps/documentation/distance-matrix/overview

---

2.0 Data Preparation

Preparing the Dataset for Analysis

Preparing the Dataset for Analysis

In this section, we collect, clean, and prepare the data for analysis and modeling. This involves gathering rental property data, calculating travel times, and engineering new features to ensure the dataset is ready for addressing the project objectives.

2.1 Rent Data Using HomeHarvest Library

- We use the **HomeHarvest Library** to collect real-time rental property data, including prices, property sizes, and locations. The data is cleaned by removing irrelevant columns, handling missing values, and ensuring consistency.

2.2 Travel Time Using Google Distance Matrix API

- Travel times (walking, transit, and driving) from each property to **Wentworth Institute of Technology (WIT)** are retrieved using the **Google Distance Matrix API**. The data is cleaned to remove outliers and standardize formats.

2.3 Feature Engineering

- New features are created to enrich the dataset

These features help answer key questions and support exploratory data analysis (EDA) and machine learning tasks.

2.1 Rent Data Using Homeharvest Library

2.1.1 Data Collection

In this section, we use the **HomeHarvest API** to scrape rental property data for Boston, MA, covering the past 90 days. The script performs the following actions:

1. **Generate a Filename for Backup**: A filename is created, allowing for easy organization and tracking of datasets over time. This backup file provides a raw copy of the data in case it's needed later.

2. **Scrape Rental Properties**: We call `scrape_property` with parameters set to:

   - `location="Boston, MA"` to specify the city,
   - `listing_type="for_rent"` to retrieve rental listings only,
   - `past_days=90` to limit results to properties listed in the last 90 days.
3. **Save Data to CSV as Backup**: The scraped data is saved as a CSV file in a specified directory, making it easy to access and analyze later.

4. **Store Data in `rent_data`**: The main dataset is stored as a DataFrame named `rent_data` for analysis in subsequent sections.

The script also prints the number of properties retrieved and a preview of the data to confirm successful data collection.

In [2]:
```
from homeharvest import scrape_property
from datetime import datetime
import pandas as pd

# Scrape properties listed for rent in Boston, focusing on the last 90 days
properties = scrape_property(
```

```
    location="Boston, MA",  # Specify the target city
    listing_type="for_rent",  # Type of listings: "for_rent" for rental properties
    past_days=90  # Fetch listings from the past 90 days
)

# Convert the scraped data to a DataFrame for easier analysis
rent_data = pd.DataFrame(properties)

# Display the number of properties retrieved
print(f"Number of properties retrieved: {len(rent_data)}")
```

```
Number of properties retrieved: 2573
```

In [3]:
```
'CHECK POINT # 1'

# Define the backup file path
backup_filename = f"C:/Users/Krish Patel/Desktop/PJC-2/datasets/raw/HomeHarvest_Bos

# Save a backup of the raw data as a CSV file
rent_data.to_csv(backup_filename, index=False)
print(f"Backup data saved to {backup_filename}")
```

```
Backup data saved to C:/Users/Krish Patel/Desktop/PJC-2/datasets/raw/HomeHarvest_Bos
ton_Rentals.csv
```

In [4]:
```
# preview of the dataset
print(rent_data.head())
```

```
                                        property_url property_id  listing_id  \
0  https://www.realtor.com/rentals/details/1-Terr...  3613420928  2974754261
1  https://www.realtor.com/rentals/details/25-Has...  9472700491  2974752318
2  https://www.realtor.com/rentals/details/325-Su...  9900803710  2974750897
3  https://www.realtor.com/rentals/details/325-Su...  9344471759  2974750890
4  https://www.realtor.com/rentals/details/1082-C...  9572867134  2974750903

    mls    mls_id    status  \
0  AVAL  61225015  FOR_RENT
1  BSMA  73311242  FOR_RENT
2  AVAL  61224795  FOR_RENT
3  AVAL  61224810  FOR_RENT
4  AVAL  61224793  FOR_RENT

                                              text      style  \
0  Amazing 5 bedroom, 2 bathroom condo in Boston....     CONDOS
1  Spacious three-bedroom unit, featuring hardwoo...  APARTMENT
2  Discover this newly renovated 2-bedroom, 1-bat...  APARTMENT
3  Check out this amazing 1-bedroom, 1-bathroom a...  APARTMENT
4  Welcome to this newly renovated 2-bedroom, 1-b...  APARTMENT

               full_street_line                 street  ... builder_id  \
0           1 Terrace St Apt A6           1 Terrace St  ...       <NA>
1          25 Haskell St Unit 2          25 Haskell St  ...       <NA>
2         325 Summit Ave Apt 302         325 Summit Ave  ...       <NA>
3         325 Summit Ave Apt 326         325 Summit Ave  ...       <NA>
4  1082 Commonwealth Ave Apt 400  1082 Commonwealth Ave  ...       <NA>

  builder_name office_id office_mls_set   office_name office_email  \
0         <NA>      <NA>           <NA>          <NA>         <NA>
1         <NA>      <NA>  O-BSMA-AN5008  Urban Realty         <NA>
2         <NA>      <NA>           <NA>          <NA>         <NA>
3         <NA>      <NA>           <NA>          <NA>         <NA>
4         <NA>      <NA>           <NA>          <NA>         <NA>

                                      office_phones  \
0                                              <NA>
1  [{'number': '(617) 254-2233', 'type': 'Office'...
2                                              <NA>
3                                              <NA>
4                                              <NA>

                                    nearby_schools  \
0  Boston School District, Roxbury Preparatory Ch...
1  Boston School District, Match Charter Public S...
2                             Boston School District
3                             Boston School District
4  Boston School District, Match Charter Public S...

                                    primary_photo  \
0  http://ap.rdcpix.com/4d202543ac3cb27bce5b6c827...
1  http://ap.rdcpix.com/5f7115a8fb35cbe2aa404c297...
2  http://ap.rdcpix.com/abd16420ac3dc7cda16793148...
3  http://ap.rdcpix.com/c66bfcec44a3efaf637223218...
4  http://ap.rdcpix.com/93b59703387afff734c237b50...
```

```
                                                alt_photos
0  http://ap.rdcpix.com/4d202543ac3cb27bce5b6c827...
1  http://ap.rdcpix.com/5f7115a8fb35cbe2aa404c297...
2  http://ap.rdcpix.com/abd16420ac3dc7cda16793148...
3  http://ap.rdcpix.com/c66bfcec44a3efaf637223218...
4  http://ap.rdcpix.com/93b59703387afff734c237b50...

[5 rows x 57 columns]
```

2.1.2 Cleaning HomeHarvest Data

To ensure the dataset is clean and ready for analysis, the following steps are undertaken:

1. **Remove Duplicates**: Identify and remove any duplicate entries based on unique property identifiers to avoid redundant data.
2. **Drop Irrelevant Columns**: Remove columns that do not contribute to the analysis, such as agent contact information and builder details.
3. **Convert Data Types**: Ensure each column has the correct data type (e.g., dates, numerical values) to support accurate calculations and analysis.
4. **Handle Missing Values**: Review columns with missing data and decide on appropriate methods (e.g., fill, drop) based on the significance of each column.

These steps provide a refined dataset, free of unnecessary or inconsistent data, for accurate analysis in subsequent sections.

In [5]:
```
'DROP 1: DROP IN MAIN FILE (this code not needed)'
import pandas as pd

# File path for your dataset
file_path = r"C:\Users\Krish Patel\Desktop\PJC-2\datasets\raw\HomeHarvest_Boston_Re

# Load the data into a DataFrame
rent_data = pd.read_csv(file_path)
```

Step 1: Removing Duplicates:

To ensure data integrity, duplicate entries are removed based on the unique `property_id` identifier. This step prevents redundancy and ensures each property is represented only once, providing a cleaner and more reliable dataset for analysis.

In [6]:
```
# Check for duplicate entries in the 'property_id' column
# This helps to identify properties that may have been listed multiple times
duplicates_property_id = rent_data[rent_data.duplicated(subset=['property_id'])]

# Count and display the total number of duplicate entries based on 'property_id'
total_duplicates = rent_data.duplicated(subset=['property_id']).sum()
print(f"\nTotal duplicates in 'property_id': {total_duplicates}")
```

```
Total duplicates in 'property_id': 70
```

In [7]:
```
# Remove duplicate entries in the DataFrame based on the 'property_id' column
# This retains only the first occurrence of each unique 'property_id'
```

```
rent_data = rent_data.drop_duplicates(subset=['property_id'], keep='first')

# Verify that duplicates have been removed by checking for any remaining duplicates
remaining_duplicates = rent_data.duplicated(subset=['property_id']).sum()
print(f"Remaining duplicates in 'property_id': {remaining_duplicates}")
```

```
Remaining duplicates in 'property_id': 0
```

In [8]:
```
# Display a summary of the DataFrame's structure, including column names, data type
# and the number of non-null entries in each column.
# This helps in identifying missing values and understanding the overall structure
rent_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2503 entries, 0 to 2572
Data columns (total 57 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   property_url       2503 non-null   object
 1   property_id        2503 non-null   int64
 2   listing_id         2503 non-null   int64
 3   mls                2503 non-null   object
 4   mls_id             2503 non-null   object
 5   status             2503 non-null   object
 6   text               2281 non-null   object
 7   style              2503 non-null   object
 8   full_street_line   2503 non-null   object
 9   street             2502 non-null   object
 10  unit               2369 non-null   object
 11  city               2503 non-null   object
 12  state              2503 non-null   object
 13  zip_code           2503 non-null   int64
 14  beds               2455 non-null   float64
 15  full_baths         2451 non-null   float64
 16  half_baths         258 non-null    float64
 17  sqft               2023 non-null   float64
 18  year_built         582 non-null    float64
 19  days_on_mls        2503 non-null   int64
 20  list_price         2455 non-null   float64
 21  list_price_min     46 non-null     float64
 22  list_price_max     46 non-null     float64
 23  list_date          2503 non-null   object
 24  sold_price         482 non-null    float64
 25  last_sold_date     482 non-null    object
 26  assessed_value     615 non-null    float64
 27  estimated_value    1088 non-null   float64
 28  new_construction   2503 non-null   bool
 29  lot_sqft           1093 non-null   float64
 30  price_per_sqft     2023 non-null   float64
 31  latitude           2498 non-null   float64
 32  longitude          2498 non-null   float64
 33  neighborhoods      2481 non-null   object
 34  county             2503 non-null   object
 35  fips_code          2481 non-null   float64
 36  stories            0 non-null      float64
 37  hoa_fee            833 non-null    float64
 38  parking_garage     133 non-null    float64
 39  agent_id           989 non-null    float64
 40  agent_name         1286 non-null   object
 41  agent_email        1262 non-null   object
 42  agent_phones       1233 non-null   object
 43  agent_mls_set      1286 non-null   object
 44  agent_nrds_id      274 non-null    float64
 45  broker_id          430 non-null    float64
 46  broker_name        537 non-null    object
 47  builder_id         0 non-null      float64
 48  builder_name       0 non-null      float64
 49  office_id          766 non-null    float64
 50  office_mls_set     1296 non-null   object
```

```
 51  office_name       1296 non-null   object
 52  office_email      635 non-null    object
 53  office_phones     1263 non-null   object
 54  nearby_schools    2498 non-null   object
 55  primary_photo     2467 non-null   object
 56  alt_photos        2467 non-null   object
dtypes: bool(1), float64(25), int64(4), object(27)
memory usage: 1.1+ MB
```

Step 2: Drop Irrelevant Columns

The following columns are dropped to streamline the dataset, focusing on information relevant to rental analysis:

- **listing_id, mls, mls_id**: Redundant identifiers that don't contribute useful insights.
- **year_built**: Limited relevance for tenant considerations in rental properties.
- **list_price_min, list_price_max, sold_price, last_sold_date**: Historical or minimum/maximum price information, which is less relevant for analyzing current rentals.
- **assessed_value, estimated_value**: More pertinent to property ownership than rentals.
- **lot_sqft**: Lot size holds lower importance for tenants who typically focus on interior space.
- **stories**: No data is available in this column.
- **agent and broker information** (`agent_id` , `agent_name` , `agent_email` , etc.): Contact details are irrelevant to the rental analysis.
- **builder information** (`builder_id` , `builder_name`): Not needed for rental properties.
- **office information** (`office_id` , `office_mls_set` , `office_name` , etc.): Office data is unnecessary for analysis.
- **status**: Uniform across the dataset with "FOR_RENT" status.
- **nearby_schools**: Nearby schools are not a primary focus for general rental analysis.
- **price_per_sqft**: Derived from `list_price` and `sqft` , so it's not essential as a standalone column.
- **state, city, hoa_fee**: General location information (`state` , `city`) is redundant with specific coordinates provided; `hoa_fee` applies inconsistently and is often not relevant for rental desirability.

This step removes unnecessary information, making the dataset more manageable and relevant to our rental analysis goals.

In [9]:
```
# Define the list of columns to drop from the DataFrame
# These columns were identified as irrelevant or redundant for the analysis of rent
columns_to_drop = [
    "listing_id", "mls", "mls_id", "year_built", "list_price_min", "list_price_max"
    "sold_price", "last_sold_date", "assessed_value", "estimated_value", "lot_sqft"
    "stories", "agent_id", "agent_name", "agent_email", "agent_phones", "agent_mls_
    "agent_nrds_id", "broker_id", "broker_name", "builder_id", "builder_name", "off
    "office_mls_set", "office_name", "office_email", "office_phones", "status",
```

```
    "nearby_schools", "price_per_sqft", "city", "state", "hoa_fee"
]


# Remove the specified columns from the DataFrame
# This step reduces the dataset to only relevant columns for rental analysis
rent_data.drop(columns=columns_to_drop, inplace=True)

# Display the remaining columns in the DataFrame to confirm the changes
rent_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2503 entries, 0 to 2572
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   property_url      2503 non-null   object
 1   property_id       2503 non-null   int64
 2   text              2281 non-null   object
 3   style             2503 non-null   object
 4   full_street_line  2503 non-null   object
 5   street            2502 non-null   object
 6   unit              2369 non-null   object
 7   zip_code          2503 non-null   int64
 8   beds              2455 non-null   float64
 9   full_baths        2451 non-null   float64
 10  half_baths        258 non-null    float64
 11  sqft              2023 non-null   float64
 12  days_on_mls       2503 non-null   int64
 13  list_price        2455 non-null   float64
 14  list_date         2503 non-null   object
 15  new_construction  2503 non-null   bool
 16  latitude          2498 non-null   float64
 17  longitude         2498 non-null   float64
 18  neighborhoods     2481 non-null   object
 19  county            2503 non-null   object
 20  fips_code         2481 non-null   float64
 21  parking_garage    133 non-null    float64
 22  primary_photo     2467 non-null   object
 23  alt_photos        2467 non-null   object
dtypes: bool(1), float64(9), int64(3), object(11)
memory usage: 471.8+ KB
```

Step 3: Convert Data Types

To optimize the dataset for analysis, we adjust data types based on each column's content and usage. This includes:

1. **Datetime Conversion**: Converting date columns (e.g., `list_date`) to `datetime` type enables date-specific calculations and filtering.
2. **Numeric Conversion**: Ensuring numeric columns (e.g., `list_price`, `sqft`, `beds`, `baths`) are appropriately set as `float` or `int` for accurate calculations and analysis.

3. **Categorical Conversion**: Converting certain text columns with limited unique values (`style`) to categorical types improves processing speed and reduces memory usage.

Adjusting data types ensures efficient handling of the dataset, especially for memory-intensive operations and numeric calculations.

In [10]:
```
# Step 1: Convert 'list_date' to datetime format
# This allows for date-specific operations such as filtering and time-based calcula
rent_data['list_date'] = pd.to_datetime(rent_data['list_date'], errors='coerce')

# Step 2: Convert key numerical columns to appropriate data types
# Ensures fields like 'list_price', 'sqft', 'beds', and 'baths' are set as numerica
rent_data['list_price'] = rent_data['list_price'].astype(float)       # Rental pric
rent_data['sqft'] = rent_data['sqft'].astype(float)                   # Square foot
rent_data['beds'] = rent_data['beds'].fillna(0).astype(int)           # Beds as int
rent_data['full_baths'] = rent_data['full_baths'].fillna(0).astype(int)  # Full bat
rent_data['half_baths'] = rent_data['half_baths'].fillna(0).astype(int)  # Half bat
rent_data['days_on_mls'] = rent_data['days_on_mls'].fillna(0).astype(int) # Days on

# Step 3: Convert specific text columns to categorical data types
# This step reduces memory usage and speeds up operations on columns with limited u
categorical_columns = ['style']
for col in categorical_columns:
    rent_data[col] = rent_data[col].astype('category')

# Step 4: Verify changes
# Display a summary to confirm that all columns are correctly formatted
rent_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2503 entries, 0 to 2572
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   property_url      2503 non-null   object
 1   property_id       2503 non-null   int64
 2   text              2281 non-null   object
 3   style             2503 non-null   category
 4   full_street_line  2503 non-null   object
 5   street            2502 non-null   object
 6   unit              2369 non-null   object
 7   zip_code          2503 non-null   int64
 8   beds              2503 non-null   int64
 9   full_baths        2503 non-null   int64
 10  half_baths        2503 non-null   int64
 11  sqft              2023 non-null   float64
 12  days_on_mls       2503 non-null   int64
 13  list_price        2455 non-null   float64
 14  list_date         2503 non-null   datetime64[ns]
 15  new_construction  2503 non-null   bool
 16  latitude          2498 non-null   float64
 17  longitude         2498 non-null   float64
 18  neighborhoods     2481 non-null   object
 19  county            2503 non-null   object
 20  fips_code         2481 non-null   float64
 21  parking_garage    133 non-null    float64
 22  primary_photo     2467 non-null   object
 23  alt_photos        2467 non-null   object
dtypes: bool(1), category(1), datetime64[ns](1), float64(6), int64(6), object(9)
memory usage: 454.9+ KB
```

Step 4: Handling Missing Values

The following is a summary of missing values across columns in the `rent_data` DataFrame:

In [11]:
```python
# Check for missing values across the entire DataFrame
missing_values_summary = rent_data.isnull().sum()
print("Missing values per column:\n", missing_values_summary)

# Filter and display rows where 'property_url' is missing
missing_property_url = rent_data[rent_data['property_url'].isna()]
print("\nRows with missing 'property_url':")
print(missing_property_url)

# Filter and display rows where 'property_id' is missing
missing_property_id = rent_data[rent_data['property_id'].isna()]
print("\nRows with missing 'property_id':")
print(missing_property_id)
```

```
Missing values per column:
 property_url           0
property_id             0
text                  222
style                   0
full_street_line        0
street                  1
unit                  134
zip_code                0
beds                    0
full_baths              0
half_baths              0
sqft                  480
days_on_mls             0
list_price             48
list_date               0
new_construction        0
latitude                5
longitude               5
neighborhoods          22
county                  0
fips_code              22
parking_garage       2370
primary_photo          36
alt_photos             36
dtype: int64

Rows with missing 'property_url':
Empty DataFrame
Columns: [property_url, property_id, text, style, full_street_line, street, unit, zi
p_code, beds, full_baths, half_baths, sqft, days_on_mls, list_price, list_date, new_
construction, latitude, longitude, neighborhoods, county, fips_code, parking_garage,
primary_photo, alt_photos]
Index: []

[0 rows x 24 columns]

Rows with missing 'property_id':
Empty DataFrame
Columns: [property_url, property_id, text, style, full_street_line, street, unit, zi
p_code, beds, full_baths, half_baths, sqft, days_on_mls, list_price, list_date, new_
construction, latitude, longitude, neighborhoods, county, fips_code, parking_garage,
primary_photo, alt_photos]
Index: []

[0 rows x 24 columns]
```

From this summary, we observe that:

- **Critical Columns (`property_id`, `property_url`)**: These columns contain no missing values, ensuring that each property entry has a unique identifier and URL, which are crucial for data integrity.
- **Non-Essential Columns**: Columns like `parking_garage`, `unit`, and `text` have missing values but are not critical for primary analysis. These columns can be filled with

default values or left as is depending on the analysis requirements.

- **Geographical Coordinates (`latitude`, `longitude`)**: A few entries are missing latitude and longitude data, which could affect location-based analysis. However, due to the minimal number of missing values, we may decide to impute or ignore these specific entries if needed.
- **Price and Square Footage**: `list_price` and `sqft` have missing values. Since they are essential for pricing analysis, these can be filled with median or mean values to retain data consistency.
- **Images (`primary_photo`, `alt_photos`)**: Missing values in image-related columns are not likely to affect most of our analysis.

Conclusion

Since no critical columns have missing values, it may not be necessary to drop any rows entirely. Instead, we can handle missing values by filling them with appropriate default values or leave them as missing if they are non-essential for our analysis. This approach ensures we retain as much data as possible for comprehensive analysis.

In [12]:
```
'CHECK POINT 2'
from datetime import datetime

# Define the backup file path for cleaned data
cleaned_backup_filename = f"C:/Users/Krish Patel/Desktop/PJC-2/datasets/cleaned/Hom

# Save a backup of the cleaned data as a CSV file
rent_data.to_csv(cleaned_backup_filename, index=False)
print(f"Cleaned data saved to {cleaned_backup_filename}")
```

```
Cleaned data saved to C:/Users/Krish Patel/Desktop/PJC-2/datasets/cleaned/HomeHarves
t_Boston_Rentals_Cleaned.csv
```

2.2 Travel Time Using Google Distance Matrix API

2.2.1: Travel Time Calculation Using Google Distance Matrix API

In this section, we calculate travel times from each rental property to Wentworth Institute of Technology using the Google Distance Matrix API. The purpose is to assess the accessibility of each property by multiple modes of transportation (walking, public transit, and driving) to determine convenience for students.

1. **API Setup**: We initialize the Google Distance Matrix API with our API key and set the destination coordinates to Wentworth Institute of Technology.
2. **Define Function**: The `get_travel_times` function is designed to calculate travel times by walking, transit, and driving modes for each property's coordinates.
3. **Calculate Travel Times**: For each property in our dataset, we call the API for each mode of transportation to fetch the travel time in minutes.

4. **Store Results**: Travel times for each mode are stored as new columns (`walking_time_min`, `transit_time_min`, `driving_time_min`) in our DataFrame.

By adding these travel times, we aim to enrich our dataset with practical accessibility information, which will later help in evaluating rental properties based on their proximity to the university.

In [ ]:
```
'DROP 2: DROP IN MAIN FILE (this code not needed)'
import pandas as pd

# Define the file path for the cleaned data
cleaned_data_path = "C:/Users/Krish Patel/Desktop/PJC-2/datasets/cleaned/HomeHarves

# Read the cleaned data into a DataFrame
rent_data_cleaned = pd.read_csv(cleaned_data_path)

# Display a summary to confirm successful loading
print("Cleaned data loaded successfully")
```

```
Cleaned data loaded successfully:
```

In [ ]:
```
import requests
import pandas as pd
from datetime import datetime

# Google API Key
api_key = 'AIzaSyB8LVFKQax1X_1OdTKEFmN8QXa5RT7qedU'

# Coordinates of Wentworth Institute of Technology
wentworth_coords = "42.336611,-71.095019"

# Function to calculate travel times using Google Distance Matrix API
def get_travel_times(lat, lon, api_key):
    # Define the destination coordinates (Wentworth Institute)
    destination = wentworth_coords

    # Create a base URL for the API
    url = "https://maps.googleapis.com/maps/api/distancematrix/json"

    # Define travel modes
    travel_modes = ['walking', 'transit', 'driving']
    travel_times = {}

    # Loop over each travel mode and make API requests
    for mode in travel_modes:
        # Define query parameters
        params = {
            'origins': f"{lat},{lon}",
            'destinations': destination,
            'mode': mode,
            'key': api_key
        }

        # Make a request to the API
        response = requests.get(url, params=params)
```

```
        data = response.json()

        # Check if the API response is OK and contains the travel time information
        try:
            # Extract travel time in seconds
            travel_time = data['rows'][0]['elements'][0]['duration']['value']
            travel_times[mode] = travel_time / 60  # Convert to minutes
        except (KeyError, IndexError):
            travel_times[mode] = None  # Assign None if data is unavailable

    return travel_times['walking'], travel_times['transit'], travel_times['driving'

# Initialize lists to store travel times for each property
walking_times = []
transit_times = []
driving_times = []

# Iterate through each property in the full dataset and calculate travel times
for index, row in rent_data_cleaned.iterrows():
    lat, lon = row['latitude'], row['longitude']

    # Skip if latitude or longitude is missing
    if pd.isna(lat) or pd.isna(lon):
        walking_times.append(None)
        transit_times.append(None)
        driving_times.append(None)
    else:
        # Get travel times for the property
        walking, transit, driving = get_travel_times(lat, lon, api_key)
        walking_times.append(walking)
        transit_times.append(transit)
        driving_times.append(driving)

# Add the travel times to the DataFrame as new columns
rent_data_cleaned['walking_time_min'] = walking_times
rent_data_cleaned['transit_time_min'] = transit_times
rent_data_cleaned['driving_time_min'] = driving_times
# renaming file just for understanding
rent_data_w_t = rent_data_cleaned
```

```
Full dataset with travel times saved to C:/Users/Krish Patel/Desktop/PJC-2/datasets/
cleaned/HomeHarvest_Boston_Rentals_with_Travel_Times_20241109_161914.csv
```

2.2.2 Data Cleaning

Now that we have gathered travel time data for each rental property in Boston to Wentworth Institute of Technology, we proceed with data cleaning. The gathered data includes walking, transit, and driving times in minutes. In this step, we will:

- Round off the travel time values to improve readability.
- Perform additional data cleaning steps as necessary to prepare the data for analysis.

In [ ]:
```
# Cleaning: Fill NaN values with 0, round off the travel times to whole numbers, an
rent_data_w_t['walking_time_min'] = rent_data_w_t['walking_time_min'].fillna(0).rou
rent_data_w_t['transit_time_min'] = rent_data_w_t['transit_time_min'].fillna(0).rou
```

```
rent_data_w_t['driving_time_min'] = rent_data_w_t['driving_time_min'].fillna(0).rou

# Verify cleaning by displaying a sample of the data
print(rent_data_w_t[['walking_time_min', 'transit_time_min', 'driving_time_min']].h
```

```
   walking_time_min  transit_time_min  driving_time_min
0                12                12                 4
1                72                48                15
2                60                53                14
3                60                53                14
4                47                33                12
Cleaned data saved to C:/Users/Krish Patel/Desktop/PJC-2/datasets/cleaned/HomeHarves
t_Boston_Rentals_Cleaned_with_Rounded_Travel_Times.csv
```

In [ ]:
```
'CHECK POINT # 3'
# Save the cleaned data back to the CSV if needed
cleaned_file_path = "C:/Users/Krish Patel/Desktop/PJC-2/datasets/cleaned/HomeHarves
rent_data_w_t.to_csv(cleaned_file_path, index=False)
print(f"Cleaned data saved to {cleaned_file_path}")
```

2.3 Feature Engineering

In this section, we will enhance our dataset by creating new variables (features) that provide deeper insights into the rental properties. These features will help answer key analytical questions and serve as inputs for potential machine learning models. The goal is to transform the raw data into meaningful, interpretable variables that capture relationships between rental prices, property attributes, and travel times.

Key Objectives:

1. **Enhance Data for Travel Time Analysis**:
   - Create features that quantify the impact of travel times (walking, transit, driving) on rental prices.
2. **Analyze Property Characteristics**:
   - Introduce metrics for property size, price per square footage, and bedroom-to-bathroom ratios.
3. **Explore Location-Based Trends**:
   - Include aggregated metrics (e.g., average prices by zipcode) to understand spatial pricing patterns.

Planned Features:

Below are the features to be created, organized by their purpose:

1. **Travel Time Impact**:
   - `price_per_min_walk` : Rental price divided by walking time.
   - `price_per_min_transit` : Rental price divided by transit time.

- `price_per_min_drive` : Rental price divided by driving time.

2. **Property Characteristics**:

   - `price_per_sqft` : Rental price divided by square footage.
   - `bed_bath_ratio` : Ratio of bedrooms to bathrooms.

3. **Location-Based Metrics**:

   - `avg_zipcode_price` : Average rental price for the property's zipcode.
   - `neighborhood_travel_score` : Weighted score based on walking, transit, and driving times for each neighborhood.

Next Steps:

- Implement code to compute these features and append them to the dataset.
- Use these engineered features for exploratory data analysis (EDA) and building machine learning models.

In [48]:
```
'DROP 3: '
import pandas as pd

# Define the file path for the cleaned data with travel times
cleaned_file_path = "C:/Users/Krish Patel/Desktop/PJC-2/datasets/cleaned/HomeHarves

# Read the cleaned data into a DataFrame
rent_data_w_t = pd.read_csv(cleaned_file_path)

# Display a message confirming successful loading and show the first few rows
print("Data loaded successfully.")
```

```
Data loaded successfully.
```

2.3.1 Travel Time Impact

Variables Created:

1. **price_per_min_walk**: Measures how much rent is paid per minute of walking to Wentworth. This helps analyze the trade-off between walking distance and affordability.
2. **price_per_min_transit**: Measures rent per minute of public transit time. This indicates the relative convenience for commuters relying on transit.
3. **price_per_min_drive**: Measures rent per minute of driving time. Useful for understanding affordability for tenants commuting by car.

Question Addressed:

1. *How do travel times by walking, transit, and driving correlate with rental prices?*
2. *What are the trends in price per minute for each travel mode?*
3. *Are there any clear patterns indicating trade-offs between travel time and rental prices?*

These variables allow us to compare rental prices relative to travel times and identify trends in affordability and convenience across different modes of transportation.

In [49]:
```
# Create a copy to calculate price per minute for each travel mode and replace infi
rent_data_w_t['price_per_min_walk'] = (rent_data_w_t['list_price'] / rent_data_w_t[
rent_data_w_t.loc[~rent_data_w_t['price_per_min_walk'].replace([float('inf'), -floa

rent_data_w_t['price_per_min_transit'] = (rent_data_w_t['list_price'] / rent_data_w
rent_data_w_t.loc[~rent_data_w_t['price_per_min_transit'].replace([float('inf'), -f

rent_data_w_t['price_per_min_drive'] = (rent_data_w_t['list_price'] / rent_data_w_t
rent_data_w_t.loc[~rent_data_w_t['price_per_min_drive'].replace([float('inf'), -flo

# Verify cleaning by displaying a sample of the data
print(rent_data_w_t[['price_per_min_walk', 'price_per_min_transit', 'price_per_min_
```

```
   price_per_min_walk  price_per_min_transit  price_per_min_drive
0              750.00                 750.00              2250.00
1               46.53                  69.79               223.33
2               53.33                  60.38               228.57
3               50.00                  56.60               214.29
4               68.09                  96.97               266.67
```

2.3.2 Property Characteristics

Variables Created:

1. **price_per_sqft**: Measures the rental cost per square foot. This helps evaluate properties in terms of the value offered for the space provided.
2. **bed_bath_ratio**: Ratio of the number of bedrooms to bathrooms. Helps assess the functional layout of a property.

Questions Addressed:

1. *How does the price per square foot vary across different zip codes?*
2. *Do bedrooms and bathrooms sizes have impact on rental price?*

These variables allow for deeper insights into property characteristics and can guide students in finding properties that balance size, price, and functionality.

In [50]:
```
# Create the 'price_per_sqft' variable
rent_data_w_t['price_per_sqft'] = (rent_data_w_t['list_price'] / rent_data_w_t['sqf

# Handle cases where sqft might be zero or NaN to avoid division errors
rent_data_w_t['price_per_sqft'] = rent_data_w_t['price_per_sqft'].replace([float('i

# Create the 'bed_bath_ratio' variable
rent_data_w_t['bed_bath_ratio'] = (rent_data_w_t['beds'] /
                                   (rent_data_w_t['full_baths'] + rent_data_w_t['ha

# Handle cases where the denominator might be zero to avoid division errors
rent_data_w_t['bed_bath_ratio'] = rent_data_w_t['bed_bath_ratio'].replace([float('i
```

```
# Verify the newly created variables
print(rent_data_w_t[['price_per_sqft', 'bed_bath_ratio']].head())
```

```
   price_per_sqft bed_bath_ratio
0             NaN            2.5
1            3.05            3.0
2            4.00            2.0
3            4.29            1.0
4            4.00            2.0
```

2.3.3 Location-Based Metrics

Variables Created:

1. **avg_zipcode_price**: The average rental price for all properties within the same zip code. Helps identify trends and outliers in pricing by area.
2. **neighborhood_travel_score**: A weighted score that combines walking, transit, and driving times for properties within the same neighborhood. Helps understand the travel convenience offered by specific neighborhoods.

Questions Addressed:

1. *How does the average rental price vary by zip code, and which zip codes are the most affordable or expensive?*
2. *Which neighborhoods provide the best travel convenience for Wentworth students?*

These variables help reveal location-based insights, such as regional trends in pricing and convenience scores for neighborhoods.

In [51]:
```
# Calculate the 'avg_zipcode_price' for each zip code
avg_price_by_zipcode = rent_data_w_t.groupby('zip_code')['list_price'].mean().round
rent_data_w_t['avg_zipcode_price'] = rent_data_w_t['zip_code'].map(avg_price_by_zip

# Create 'neighborhood_travel_score' as the average of walking, transit, and drivin
rent_data_w_t['neighborhood_travel_score'] = (
    rent_data_w_t[['walking_time_min', 'transit_time_min', 'driving_time_min']].mea
)

# Verify the newly created variables
print(rent_data_w_t[['avg_zipcode_price', 'neighborhood_travel_score']].head())
```

```
   avg_zipcode_price  neighborhood_travel_score
0            5153.19                       9.33
1            3292.22                      45.00
2            3331.28                      42.33
3            3331.28                      42.33
4            3392.84                      30.67
```

In [52]:
```
'CHECK POINT # 4'
# Define the file path for saving the final DataFrame
final_file_path = "C:/Users/Krish Patel/Desktop/PJC-2/datasets/final/HomeHarvest_Bo

# Save the DataFrame to a CSV file
```

```
rent_data_w_t.to_csv(final_file_path, index=False)

# Confirm successful save
print(f"Final dataset saved successfully to {final_file_path}")
```

```
Final dataset saved successfully to C:/Users/Krish Patel/Desktop/PJC-2/datasets/fina
l/HomeHarvest_Boston_Rentals_Final.csv
```

3.0 Data analysis

Finding answers to key questions by data analysis

This section explores the relationships between rental prices and key influencing factors, focusing on travel convenience, property characteristics, and location-based affordability. This section addresses the core research questions by breaking them into sub-questions and analyzing trends, correlations, and trade-offs through statistical methods and visualizations.

Key Research Questions and Sub-Questions

1. Travel Time Impact *(What is the impact of property characteristics on rental prices?))*

- How do travel times by walking, transit, and driving correlate with rental prices?
- What are the trends in price per minute for each travel mode?
- Are there any clear patterns indicating trade-offs between travel time and rental prices?

2. Property characteristic *(How do rental prices vary by location?)*

- How does the price per square foot vary across different zip codes?
- Do bedrooms and bathrooms sizes have impact on rental price?

3. Location-Based *(How can affordability and convenience be balanced?))*

- How does the average rental price vary by zip code, and which zip codes are the most affordable or expensive?
- Which neighborhoods provide the best travel convenience for Wentworth students?

In [53]:
```
'DROP 4: '
import pandas as pd

# Define the file path for the cleaned data with travel times
cleaned_file_path = "C:/Users/Krish Patel/Desktop/PJC-2/datasets/final/HomeHarvest_

# Read the cleaned data into a DataFrame
rent_data_w_t = pd.read_csv(cleaned_file_path)

# Display a message confirming successful loading and show the first few rows
print("Data loaded successfully.")
```

```
Data loaded successfully.
```

3.1 Travel Time Impact Analysis

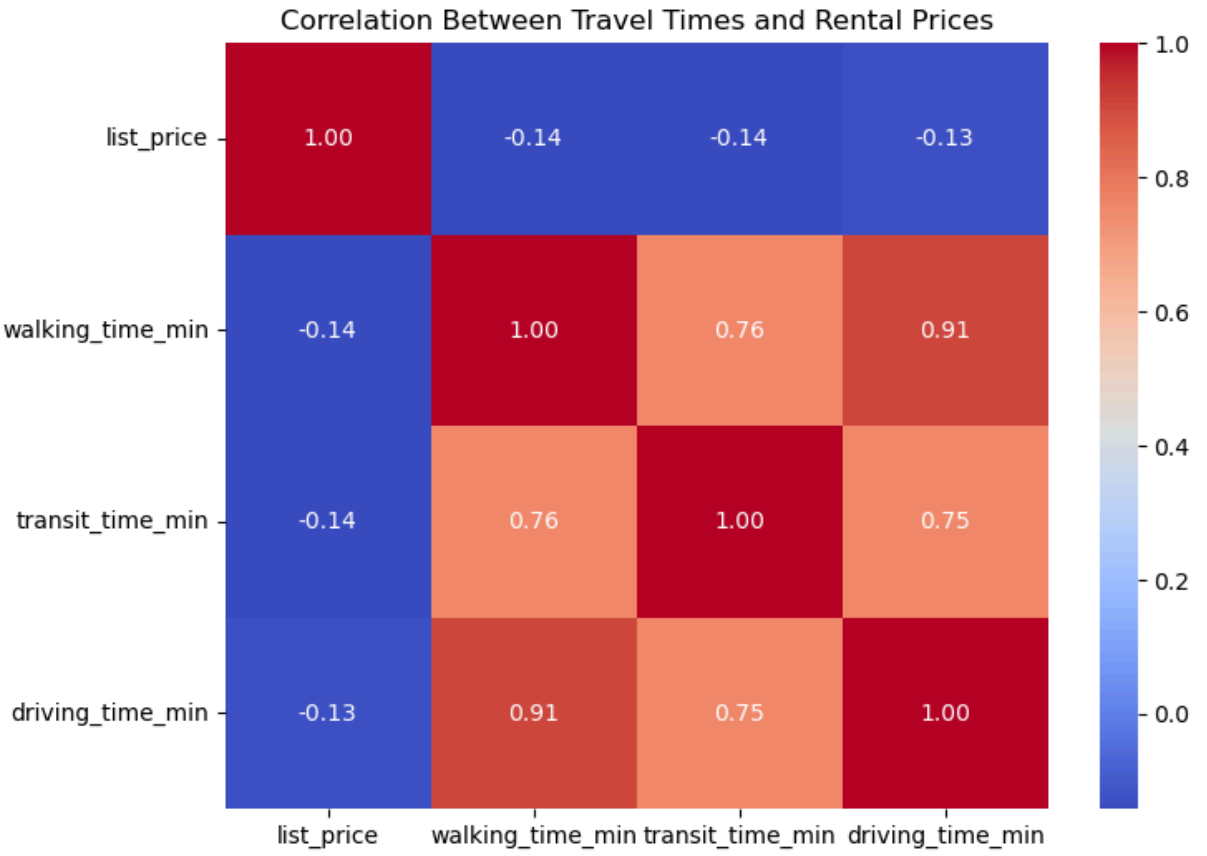
In this section, we aim to analyze the impact of travel times (by walking, transit, and driving) on rental prices. Using the engineered features, we will explore the questions.

Questions Addressed:

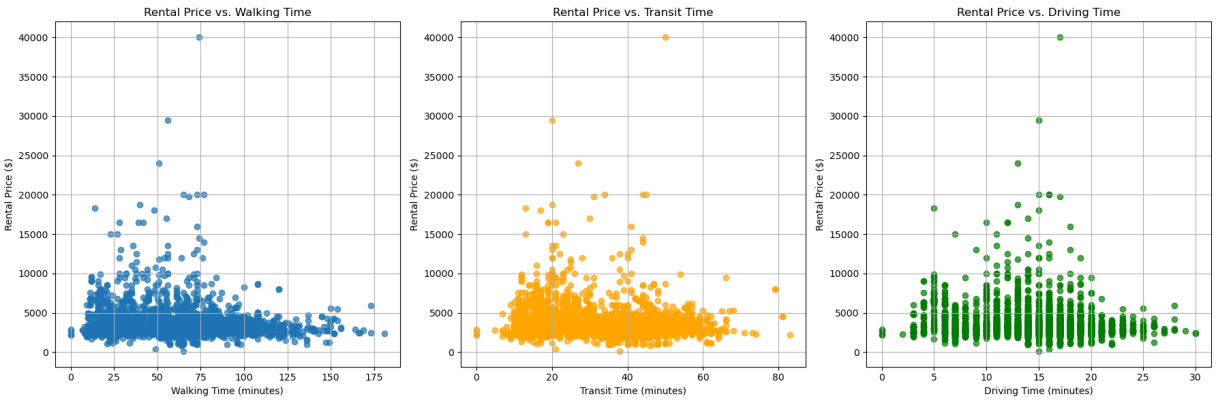
1. *How do travel times by walking, transit, and driving correlate with rental prices?*
2. *What are the trends in price per minute for each travel mode?*
3. *Are there any clear patterns indicating trade-offs between travel time and rental prices?*

3.1.1 Correlation between walking, transit, driving time with rental price

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation heatmap to analyze relationships
corr_columns = ['list_price', 'walking_time_min', 'transit_time_min', 'driving_time
plt.figure(figsize=(8, 6))
sns.heatmap(rent_data_w_t[corr_columns].corr(), annot=True, cmap='coolwarm', fmt='.
plt.title("Correlation Between Travel Times and Rental Prices")
plt.show()
```



In [57]:
```
import matplotlib.pyplot as plt

# Scatter plots for rental price vs. travel times
plt.figure(figsize=(18, 6))

# Plot 1: Rental Price vs. Walking Time
plt.subplot(1, 3, 1)
plt.scatter(rent_data_w_t['walking_time_min'], rent_data_w_t['list_price'], alpha=0
plt.title('Rental Price vs. Walking Time')
plt.xlabel('Walking Time (minutes)')
plt.ylabel('Rental Price ($)')
plt.grid(True)

# Plot 2: Rental Price vs. Transit Time
plt.subplot(1, 3, 2)
plt.scatter(rent_data_w_t['transit_time_min'], rent_data_w_t['list_price'], alpha=0
plt.title('Rental Price vs. Transit Time')
plt.xlabel('Transit Time (minutes)')
plt.ylabel('Rental Price ($)')
plt.grid(True)

# Plot 3: Rental Price vs. Driving Time
plt.subplot(1, 3, 3)
plt.scatter(rent_data_w_t['driving_time_min'], rent_data_w_t['list_price'], alpha=0
plt.title('Rental Price vs. Driving Time')
plt.xlabel('Driving Time (minutes)')
plt.ylabel('Rental Price ($)')
plt.grid(True)

# Show the plots
plt.tight_layout()
plt.show()
```



Analysis:

How do travel times by walking, transit, and driving correlate with rental prices?

1. **Correlation Heatmap Insights:**

   - The correlation heatmap reveals a **weak negative correlation** between travel times and rental prices for all modes of transportation:
     - Walking Time: **-0.14**

    - Transit Time: **-0.14**
    - Driving Time: **-0.13**
  - This implies that properties with shorter travel times (by walking, transit, or driving) tend to have slightly higher rental prices. However, the weak correlation values indicate that travel time alone is not a strong determinant of rental price.
2. **Scatterplot Insights:**
   - **Rental Prices vs Walking Time:**
     - Properties with shorter walking times show a wider range of rental prices, including several high-priced properties above $10,000.
     - As walking time increases, the maximum rental price decreases, with very few high-priced properties at walking times over 60 minutes.
   - **Rental Prices vs Transit Time:**
     - Transit time shows a more condensed range of rental prices compared to walking time.
     - Most properties with higher rental prices are concentrated within 20–30 minutes of transit time.
   - **Rental Prices vs Driving Time:**
     - Driving time exhibits a sharp clustering of rental prices between 10–20 minutes.
     - As driving time exceeds 20 minutes, very few high-priced properties remain.
3. **Key Observations:**
   - Shorter travel times, regardless of the mode of transportation, tend to correspond to higher rental prices. This reflects the convenience premium associated with properties closer to key destinations.
   - Properties with higher rental prices are primarily clustered at shorter travel times, especially below:
     - 30 minutes walking time.
     - 20 minutes transit time.
     - 15 minutes driving time.
   - Longer travel times rarely correspond to high-priced rentals.

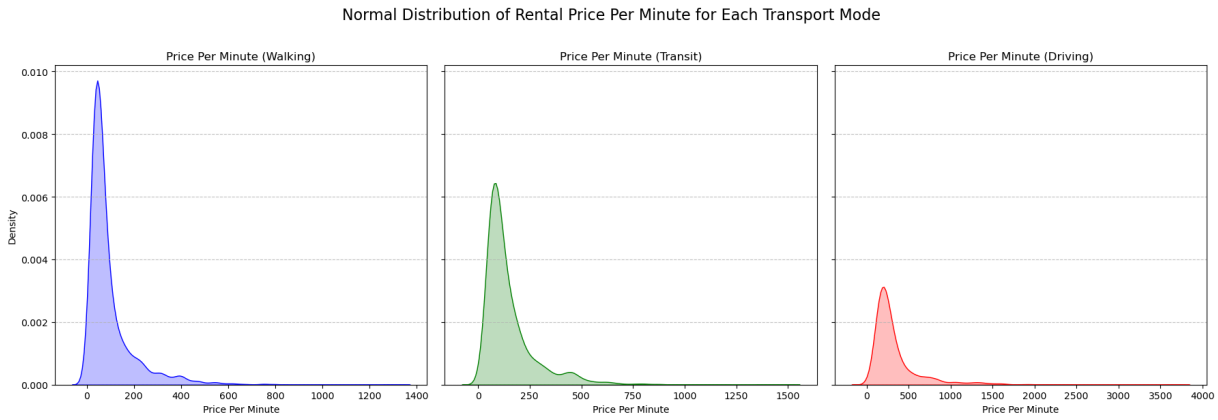
Conclusion:

- Travel times (walking, transit, driving) have a moderate relationship with rental prices, where shorter travel times are associated with slightly higher prices.
- **However**, the weak correlations suggest that other factors, such as property size, neighborhood, or amenities, play a more significant role in determining rental prices.
- These insights indicate that renters willing to pay a premium can expect reduced travel times, particularly for walking and transit modes.

3.1.2 Trends in price per minute for each travel mode

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a figure with three subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)

# KDE plot for price_per_min_walk
sns.kdeplot(
    rent_data_w_t['price_per_min_walk'],
    ax=axes[0],
    fill=True,  # Replace 'shade' with 'fill'
    color="blue"
)
axes[0].set_title('Price Per Minute (Walking)')
axes[0].set_xlabel('Price Per Minute')
axes[0].set_ylabel('Density')
axes[0].grid(axis='y', linestyle='--', alpha=0.7)

# KDE plot for price_per_min_transit
sns.kdeplot(
    rent_data_w_t['price_per_min_transit'],
    ax=axes[1],
    fill=True,  # Replace 'shade' with 'fill'
    color="green"
)
axes[1].set_title('Price Per Minute (Transit)')
axes[1].set_xlabel('Price Per Minute')
axes[1].grid(axis='y', linestyle='--', alpha=0.7)

# KDE plot for price_per_min_drive
sns.kdeplot(
    rent_data_w_t['price_per_min_drive'],
    ax=axes[2],
    fill=True,  # Replace 'shade' with 'fill'
    color="red"
)
axes[2].set_title('Price Per Minute (Driving)')
axes[2].set_xlabel('Price Per Minute')
axes[2].grid(axis='y', linestyle='--', alpha=0.7)

# Adjust layout and display the plot
plt.suptitle('Normal Distribution of Rental Price Per Minute for Each Transport Mod
plt.tight_layout()
plt.show()
```



Analysis:

What are the trends in price per minute for each travel mode?

Observations:

1. **Walking (Blue Plot):**
    - The majority of rental prices per minute are concentrated at lower values, peaking below 200.
    - The distribution is heavily skewed to the right, indicating a few properties with significantly higher price-per-minute values.
    - The density drops sharply after 300, with very few properties above 500.
2. **Transit (Green Plot):**
    - Transit follows a similar trend, with a peak density occurring under 150 price per minute.
    - The distribution is more compact compared to walking, indicating that price variations are less extreme.
    - There are fewer extreme outliers compared to walking.
3. **Driving (Red Plot):**
    - Driving shows the lowest price-per-minute density overall, with a peak density well below 100.
    - The range of values is narrower than walking and transit, suggesting that driving-related properties are more consistent in pricing.
    - Extreme outliers are nearly nonexistent.

Interpretation:

- **Walking Mode:** Shows the highest variance, indicating that properties that are walkable to destinations tend to have premium pricing for convenience, but also include affordable options for further distances.
- **Transit Mode:** Offers moderate variance and affordability, making it an intermediate option for renters balancing cost and convenience.

- **Driving Mode:** Presents the lowest variance, likely because driving time is less critical for most renters who rely on public or walkable commutes, keeping prices consistent.

Insights:

- Renters who prioritize walking convenience must pay a premium but also have the widest range of pricing options.
- Transit is more affordable while still offering competitive travel convenience.
- Driving remains the most affordable and least volatile in terms of price-per-minute for renters.

3.1.3 patterns indicating trade-offs between travel time and rental prices

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Define bins and labels for walking, transit, and driving
walking_bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
walking_labels = ["0-10", "10-20", "20-30", "30-40", "40-50", "50-60", "60-70", "70

transit_bins = [0, 15, 30, 45, 60, 75, 90, 105]
transit_labels = ["0-15", "15-30", "30-45", "45-60", "60-75", "75-90", "90-105"]

driving_bins = [0, 5, 10, 15, 20, 25, 30, 35]
driving_labels = ["0-5", "5-10", "10-15", "15-20", "20-25", "25-30", "30-35"]

# Create new columns for time ranges
rent_data_w_t['walking_time_range'] = pd.cut(rent_data_w_t['walking_time_min'], bin
rent_data_w_t['transit_time_range'] = pd.cut(rent_data_w_t['transit_time_min'], bin
rent_data_w_t['driving_time_range'] = pd.cut(rent_data_w_t['driving_time_min'], bin

# Calculate average rental price for each time range
walking_avg_rent = rent_data_w_t.groupby('walking_time_range')['list_price'].mean()
transit_avg_rent = rent_data_w_t.groupby('transit_time_range')['list_price'].mean()
driving_avg_rent = rent_data_w_t.groupby('driving_time_range')['list_price'].mean()

# Plotting all three in one figure, side by side
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Walking time range plot
axs[0].bar(walking_avg_rent['walking_time_range'], walking_avg_rent['list_price'],
axs[0].set_title('Average Rental Price by Walking Time Range')
axs[0].set_xlabel('Walking Time Range (minutes)')
axs[0].set_ylabel('Average Rental Price ($)')
axs[0].tick_params(axis='x', rotation=45)
axs[0].grid(axis='y', linestyle='--', alpha=0.7)

# Transit time range plot
axs[1].bar(transit_avg_rent['transit_time_range'], transit_avg_rent['list_price'],
axs[1].set_title('Average Rental Price by Transit Time Range')
axs[1].set_xlabel('Transit Time Range (minutes)')
axs[1].tick_params(axis='x', rotation=45)
axs[1].grid(axis='y', linestyle='--', alpha=0.7)
```

```
# Driving time range plot
axs[2].bar(driving_avg_rent['driving_time_range'], driving_avg_rent['list_price'],
axs[2].set_title('Average Rental Price by Driving Time Range')
axs[2].set_xlabel('Driving Time Range (minutes)')
axs[2].tick_params(axis='x', rotation=45)
axs[2].grid(axis='y', linestyle='--', alpha=0.7)

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```

```
C:\Users\Krish Patel\AppData\Local\Temp\ipykernel_22804\2314600160.py:20: FutureWarn
ing: The default of observed=False is deprecated and will be changed to True in a fu
ture version of pandas. Pass observed=False to retain current behavior or observed=T
rue to adopt the future default and silence this warning.
  walking_avg_rent = rent_data_w_t.groupby('walking_time_range')['list_price'].mean
().reset_index().dropna()
C:\Users\Krish Patel\AppData\Local\Temp\ipykernel_22804\2314600160.py:21: FutureWarn
ing: The default of observed=False is deprecated and will be changed to True in a fu
ture version of pandas. Pass observed=False to retain current behavior or observed=T
rue to adopt the future default and silence this warning.
  transit_avg_rent = rent_data_w_t.groupby('transit_time_range')['list_price'].mean
().reset_index().dropna()
C:\Users\Krish Patel\AppData\Local\Temp\ipykernel_22804\2314600160.py:22: FutureWarn
ing: The default of observed=False is deprecated and will be changed to True in a fu
ture version of pandas. Pass observed=False to retain current behavior or observed=T
rue to adopt the future default and silence this warning.
  driving_avg_rent = rent_data_w_t.groupby('driving_time_range')['list_price'].mean
().reset_index().dropna()
```

Analysis:

Are there any clear patterns indicating trade-offs between travel time and rental prices?

1. **Walking Time:**

   - **Pattern Observed:** Properties closer to the destination (0-10 minutes walking range) generally have lower rental prices compared to mid-range walking distances (20-40 minutes).
   - **Possible Explanation:** This may indicate that areas directly adjacent to the destination are more affordable, while mid-range walking distances may include

prime residential areas with higher demand.
- **Trade-Off:** Beyond 40 minutes of walking, rental prices appear to stabilize or decrease slightly, suggesting diminishing value for longer walking times.

2. **Transit Time:**

- **Pattern Observed:** Rental prices for properties within 0-15 minutes of transit time are among the highest, indicating a premium for convenience in transit accessibility.
- **Price Drop:** As transit time increases (beyond 30 minutes), rental prices show a decline, with the lowest prices observed in the 60-75 minute range.
- **Trade-Off:** The data suggests a clear trade-off where properties closer to transit hubs or within shorter commuting times are priced higher due to the added convenience.

3. **Driving Time:**

- **Pattern Observed:** Rental prices are relatively high in the 5-15 minute driving range, peaking in the 10-15 minute range.
- **Price Decline:** Beyond 20 minutes of driving, rental prices steadily decline, with the lowest prices in the 30-35 minute range.
- **Trade-Off:** This indicates that properties requiring longer commutes by car are less desirable and hence more affordable.

Conclusion:

- **Clear Trade-Offs:**
  - Properties closer to the destination (shorter walking, transit, or driving times) tend to have higher rental prices, reflecting a premium on accessibility and convenience.
  - Rental prices decline as travel time increases, suggesting reduced desirability for longer commutes.
- **Key Insight:**
  - Mid-range travel times (e.g., 20-40 minutes walking or 15-30 minutes transit) may represent optimal trade-offs between affordability and accessibility for renters.
  - These patterns can guide recommendations for properties that balance convenience and affordability.

3.1.4 Conclusion: Travel Times and Rental Prices

Key Findings:

1. **Correlation with Rental Prices:**

- Shorter travel times, regardless of mode (walking, transit, or driving), are associated with higher rental prices. However, the correlation is weak, indicating that other factors, such as property size, neighborhood, and amenities, play a more significant role in determining rental prices.

2. **Price Per Minute Trends:**
    - Walking exhibits the highest variance in price per minute, reflecting a wide range of affordable and premium-priced properties.
    - Transit is moderately affordable with lower variance, offering a balanced option for cost and convenience.
    - Driving is the most affordable mode, with consistent pricing and minimal outliers.
3. **Trade-Offs Between Travel Time and Prices:**
    - Properties within shorter travel times tend to command a premium, especially for walking and transit modes.
    - Longer travel times correlate with reduced rental prices, reflecting diminished desirability for extended commutes.

How These Insights Can Inform Modeling:

- **Feature Importance:** Travel time variables (e.g., walking, transit, and driving time) can be included in a recommendation model as indicators of accessibility and convenience.
- **Weighting Factors:** Use price-per-minute and travel time ranges to create scoring systems that prioritize user preferences for affordability or accessibility.
- **Target Segments:** Identify optimal travel time ranges (e.g., 15-30 minutes for transit) to recommend properties balancing cost and convenience.

These insights lay the foundation for building a recommendation system or predictive model that evaluates rental properties based on user-specific trade-offs between travel time and rental price.

3.2 Property Characteristics Analysis

This section explores how various property characteristics influence rental pricing. We analyze the relationships between price per square foot, bed-to-bathroom ratios, and overall affordability to address the following questions:

Questions Addressed:

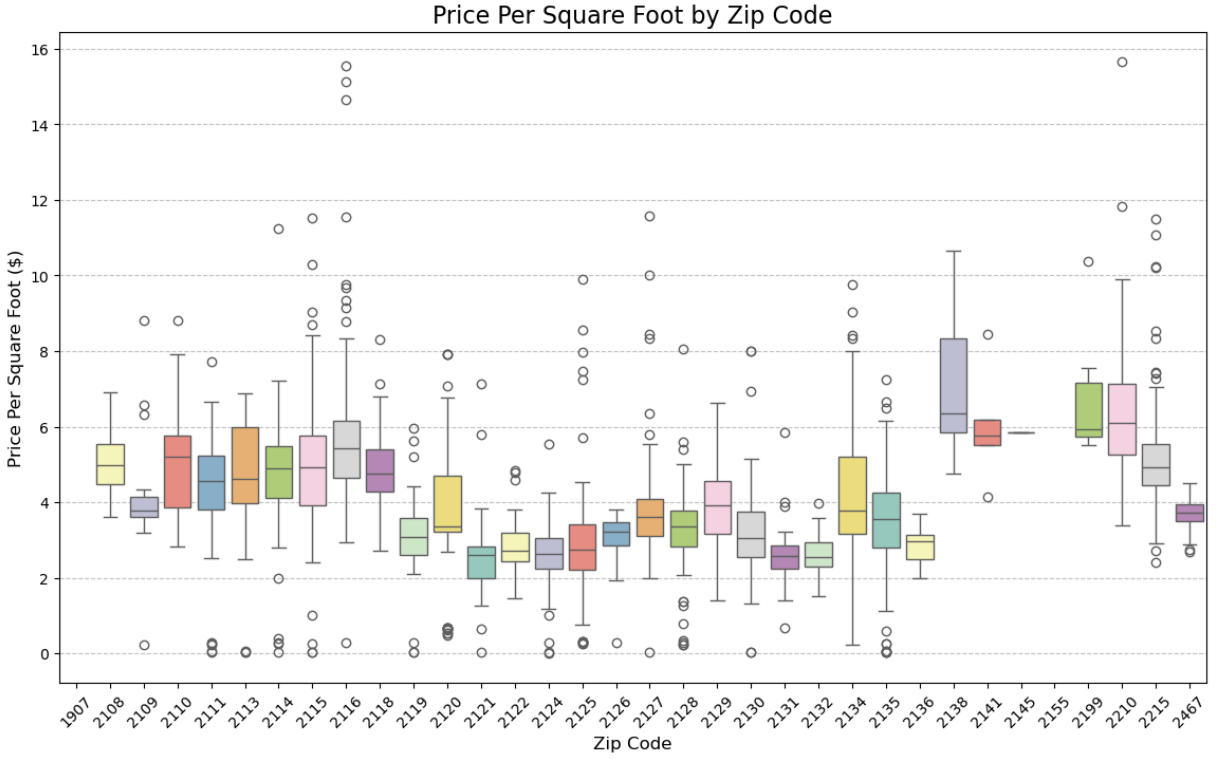
1. *How does the price per square foot vary across different zip codes?*
2. *Which properties offer the best value in terms of space and affordability?*

3.2.1 Price Per Square Foot Analysis

In [75]:
```
# Boxplot for price_per_sqft across zip codes
plt.figure(figsize=(14, 8))
sns.boxplot(
    x='zip_code',  # Using the 'zip_code' column
    y='price_per_sqft',
    data=rent_data_w_t,
    palette='Set3'
```

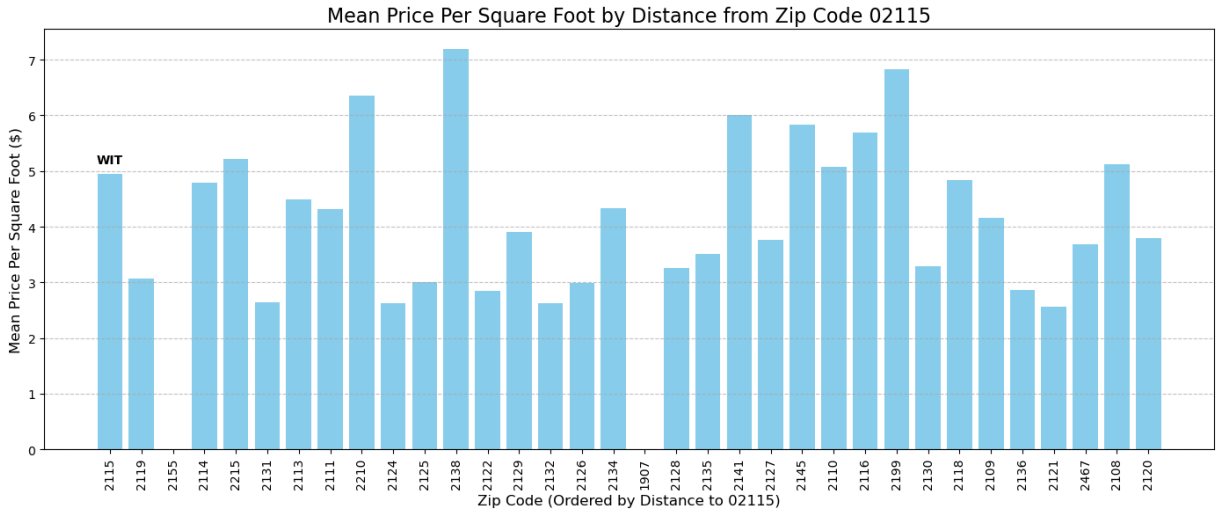
```
)
plt.title('Price Per Square Foot by Zip Code', fontsize=16)
plt.xlabel('Zip Code', fontsize=12)
plt.ylabel('Price Per Square Foot ($)', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
C:\Users\Krish Patel\AppData\Local\Temp\ipykernel_22804\58260251.py:3: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(
```



In [ ]:
```
import numpy as np


# Sort by distance to 02115
zip_code_stats_sorted_distance = zip_code_stats.sort_values(by='distance_to_02115',

# Set colors for highlighting Wentworth's zip code
colors_distance = ['orange' if str(zip_code) == '02115' else 'skyblue' for zip_code

# Plot mean price per square foot ordered by distance
plt.figure(figsize=(14, 6))
plt.bar(
    zip_code_stats_sorted_distance['zip_code'].astype(str),
    zip_code_stats_sorted_distance['mean'],
    color=colors_distance
)
plt.title('Mean Price Per Square Foot by Distance from Zip Code 02115', fontsize=16
```

```
plt.xlabel('Zip Code (Ordered by Distance to 02115)', fontsize=12)
plt.ylabel('Mean Price Per Square Foot ($)', fontsize=12)
plt.xticks(rotation=90)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Add annotation for Wentworth's zip code
plt.text('2115', zip_code_stats_sorted_distance.loc[zip_code_stats_sorted_distance[
         f'WIT', color='black', fontsize=10, ha='center', fontweight='bold')

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



Analysis:

How does the price per square foot vary across different zip codes?

Graph Insights:

1. **Box Plot Analysis:**
    - The box plot highlights the distribution of price per square foot ($) across different zip codes.
    - **Zip Code 02115 (WIT):**
        - Displays a relatively wide range of prices, with a median close to $5 per square foot.
        - Outliers suggest a few higher-priced properties within this zip code, reflecting potential premium housing options near Wentworth Institute of Technology (WIT).
    - Zip codes further from WIT generally have lower median prices and narrower ranges.
    - Some zip codes show significantly higher variances (e.g., 02139, 02138), indicating a mix of affordable and premium housing options.
2. **Bar Chart (Ordered by Proximity to 02115):**

- The bar chart orders zip codes based on their distance from 02115, offering a clear spatial understanding of pricing trends.
- **Key Observations:**
  - Zip Code 02115 (WIT) is centrally located and has a higher mean price per square foot compared to several nearby zip codes like 02119 and 02114.
  - Zip codes further from WIT show fluctuating prices, with peaks in certain regions (e.g., 02138, 02139) that might reflect localized factors such as luxury developments or higher demand in those areas.

Insights:

- **Price Variation:**
  - The proximity to WIT seems to influence pricing, with zip codes closer to 02115 generally having slightly higher average prices.
  - Certain distant zip codes, despite being far from WIT (e.g., 02138, 02139), exhibit competitive or higher prices due to independent market demand or premium housing options.
- **Localized Factors:**
  - Median price variations highlight potential influencing factors like neighborhood quality, accessibility, and amenities which are critical for modeling rental prices.

Conclusion for Machine Learning:

- **Key Features for Modeling:**
  - `price_per_sqft` : Acts as a crucial feature reflecting property value based on size.
  - `zip_code` : Encodes location-based influence, which should be incorporated for spatial analysis in the model.
  - **Distance to WIT (Zip Code 02115):**
    - The spatial arrangement of pricing provides a compelling case to create a derived feature capturing the distance of each property from WIT for predicting rental desirability.

3.2.2 Bed-to-Bathroom Ratio Analysis

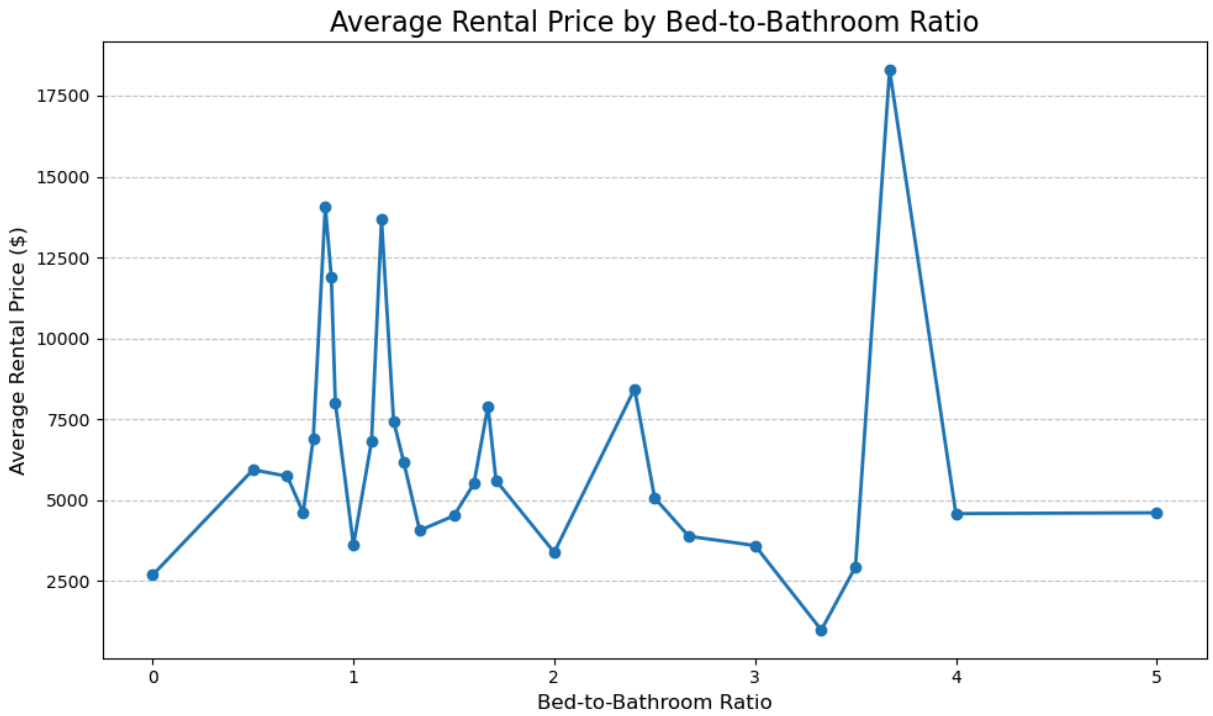
In [95]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate mean rental price grouped by number of bedrooms and bathrooms
bed_bath_price = rent_data_w_t.groupby(['beds', 'full_baths'])['list_price'].mean()

# Create the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(
    bed_bath_price,
    annot=True,
    fmt=".0f",
    cmap="YlGnBu",
```

```
    linewidths=0.5,
    linecolor='white',
    cbar_kws={'label': 'Rental Price ($)'}
)

# Add titles and labels
plt.title('Mean Rental Price by Number of Bedrooms and Bathrooms', fontsize=16)
plt.xlabel('Number of Bathrooms', fontsize=12)
plt.ylabel('Number of Bedrooms', fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```

In [98]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Calculate the average rental price for each bed-to-bathroom ratio
bed_bath_avg_price = rent_data_w_t.groupby('bed_bath_ratio')['list_price'].mean().r

# Create the line plot
plt.figure(figsize=(10, 6))
plt.plot(bed_bath_avg_price['bed_bath_ratio'], bed_bath_avg_price['list_price'], ma

# Add titles and labels
plt.title('Average Rental Price by Bed-to-Bathroom Ratio', fontsize=16)
plt.xlabel('Bed-to-Bathroom Ratio', fontsize=12)
plt.ylabel('Average Rental Price ($)', fontsize=12)

# Add grid for better readability
```

```
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```



Analysis:

Do bedrooms and bathrooms sizes have an impact on rental price?

Graph Insights:

1. **Heatmap Analysis (Rental Price by Number of Bedrooms and Bathrooms):**
    - The heatmap demonstrates the relationship between the number of bedrooms and bathrooms and the average rental price.
    - **Key Observations:**
        - Properties with **3 bedrooms and 2 bathrooms** exhibit the highest average rental price of $16,867, suggesting that this configuration is highly desirable.
        - As the number of bathrooms increases relative to bedrooms (e.g., 2 bathrooms for 1 bedroom), rental prices tend to rise, reflecting a premium for more luxurious configurations.
        - Properties with 0 bathrooms or minimal configurations (e.g., 0 bedrooms, 0 bathrooms) show the lowest rental prices, likely representing smaller units or studio apartments.
2. **Line Graph Analysis (Rental Price vs. Bed-to-Bathroom Ratio):**
    - The line plot reveals how rental prices fluctuate based on the bed-to-bathroom ratio.

- **Key Trends:**
  - Rental prices tend to spike sharply at ratios close to 1, indicating that balanced configurations (e.g., 1 bedroom to 1 bathroom) are more valuable.
  - Ratios greater than 2 (e.g., 2 bedrooms for 1 bathroom) show a declining trend in rental prices, suggesting reduced desirability for such configurations.
  - Uncommon ratios (e.g., 4 or more bedrooms per bathroom) result in irregular or lower pricing, reflecting limited availability or demand for such properties.

Insights:

- **Desirable Configurations:**
  - Properties with balanced bed-to-bathroom ratios close to 1 command the highest rental prices, likely due to their practicality and appeal to renters.
  - A higher number of bathrooms relative to bedrooms adds a premium to rental prices, making properties more attractive.
- **Inefficient Configurations:**
  - Properties with imbalanced configurations (e.g., many bedrooms but few bathrooms) tend to see reduced desirability and lower rental prices.
- **Market Dynamics:**
  - Properties with configurations like 3 bedrooms and 2 bathrooms stand out as the most valuable, indicating a sweet spot for both size and functionality.

Conclusion for Machine Learning:

- **Key Features for Modeling:**
  - `bed_bath_ratio`: This feature captures the balance between bedrooms and bathrooms, offering a strong predictor for rental prices.
  - `beds` and `baths`: As individual features, they provide insights into property size and luxury, critical for price prediction.
  - **Interaction Terms:**
    - Interaction between `beds` and `baths` can be engineered as a new feature to model the impact of specific configurations.
- **Predictive Value:**
  - This analysis highlights the importance of property configurations, particularly for mid-sized families or renters prioritizing convenience and comfort.

3.2.3 Conclusion: Property Characteristics

Insights Summary:

1. **Price per Square Foot Across Zip Codes:**

   - Proximity to WIT (Zip Code 02115) influences rental prices, with closer zip codes generally exhibiting higher prices per square foot.

- Certain distant zip codes, despite being further away, display competitive prices due to independent factors like luxury developments or market demand.
- Variance in pricing highlights localized factors such as amenities, neighborhood quality, and accessibility.

2. **Impact of Bedrooms and Bathrooms:**

- Configurations with balanced bed-to-bathroom ratios close to 1 (e.g., 1 bedroom to 1 bathroom) are more desirable and command higher rental prices.
- Properties with 3 bedrooms and 2 bathrooms stand out as the most valuable, showcasing a sweet spot in size and functionality.
- Imbalanced configurations, such as many bedrooms with few bathrooms, tend to be less attractive and result in lower rental prices.

Modeling Implications:

1. **Key Features for Prediction:**

- `price_per_sqft`**:** Captures the relationship between size and value, critical for estimating property affordability.
- `zip_code`**:** Encodes spatial influence, and distance from WIT can be engineered as a derived feature for better spatial modeling.
- `bed_bath_ratio`**:** Highlights the balance between property layout and desirability.
- `beds` **and** `baths`**:** Provide granular insights into property size and configuration.

2. **Use in Machine Learning Models:**

- These insights enable the development of predictive models for rental pricing, optimizing for both affordability and desirability.
- Models can leverage these features to segment properties by location, size, and configuration, providing personalized recommendations for renters.

3.3 Location-Based Analysis

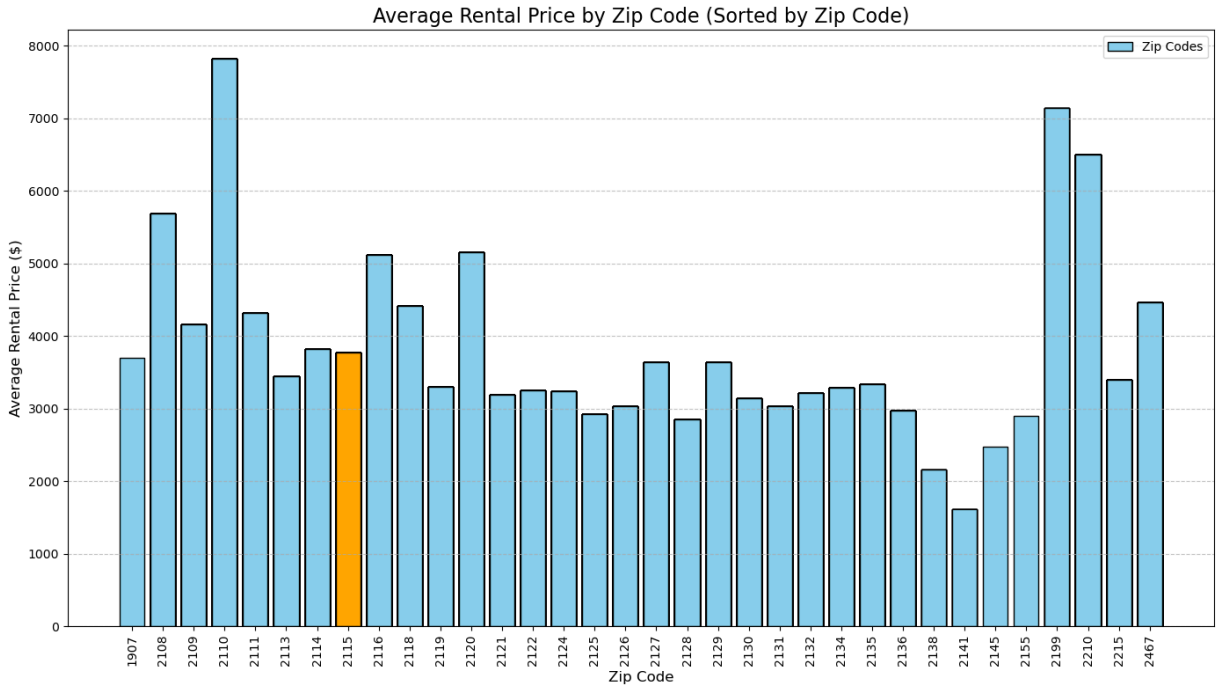
This section focuses on how location influences rental prices and travel convenience, particularly for Wentworth students. By analyzing spatial patterns, we aim to address the following questions:

Questions Addressed:

1. *How does the average rental price vary by zip code, and which zip codes are the most affordable or expensive?*
2. *Which neighborhoods provide the best travel convenience for Wentworth students?*

3.3.1 average rental price by zipcode

In [ ]:
```
# Sort data by zip code in ascending order
sorted_zip_data = rent_data_w_t.sort_values(by='zip_code')

# Highlight color for zip code 02115
colors = ['skyblue' if zip_code != 2115 else 'orange' for zip_code in sorted_zip_da

# Plotting the bar chart
plt.figure(figsize=(14, 8))
plt.bar(sorted_zip_data['zip_code'].astype(str), sorted_zip_data['avg_zipcode_price
plt.xticks(rotation=90)
plt.title('Average Rental Price by Zip Code (Sorted by Zip Code)', fontsize=16)
plt.xlabel('Zip Code', fontsize=12)
plt.ylabel('Average Rental Price ($)', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Adding a legend
plt.legend(['Zip Codes', '02115 (Highlighted)'], loc='upper right')
plt.tight_layout()
plt.show()
```



Analysis

How does the average rental price vary by zip code, and which zip codes are the most affordable or expensive?

Graph Insights:

1. **Bar Chart Analysis (Sorted by ZIP Code):**
   - The bar chart visualizes average rental prices ($) by ZIP code, sorted numerically.
   - **ZIP Code 02115 (WIT):**
     - This ZIP code is highlighted for its relevance to Wentworth Institute of Technology. It shows a moderate average rental price compared to other ZIP

codes.

- **Highest Average Rental Prices:**
  - ZIP codes like `02199` and `02210` stand out with significantly higher average rental prices, suggesting these areas host premium or luxury housing.
- **Most Affordable ZIP Codes:**
  - ZIP codes such as `2141` and `1907` show the lowest average rental prices, indicating more affordable housing options.
- **Mid-Range Pricing:**
  - Many ZIP codes, including those near `02115`, fall in the mid-range category, with moderate average rental prices reflecting their balance between affordability and accessibility.

Insights:

- **Affordability Patterns:**
  - ZIP codes such as `2141` and `1907` are more affordable, potentially attracting renters with budget constraints or those seeking basic amenities.
- **Premium Areas:**
  - High rental prices in ZIP codes `02199` and `02210` reflect luxury or high-demand neighborhoods, potentially driven by premium housing, better connectivity, or desirable amenities.
- **Balanced Regions:**
  - ZIP codes surrounding `02115` (e.g., `02114`, `02116`) provide a mix of affordability and convenience, appealing to a broader range of renters.

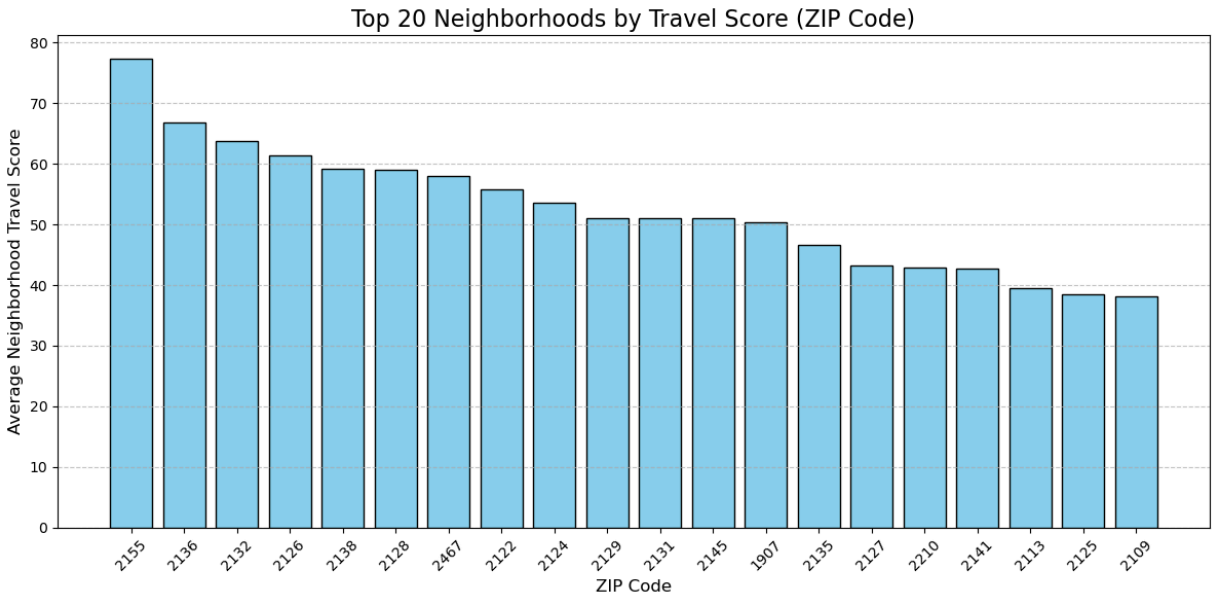
Conclusion for Machine Learning:

- **Key Features for Modeling:**
  - `zip_code`: A categorical feature capturing location-based pricing influences.
  - `avg_zipcode_price`: A derived feature directly quantifying average rental price.
- **Derived Features:**
  - Proximity to premium ZIP codes (`02199`, `02210`) or institutions like WIT (`02115`) can act as new predictors.
- **Predictive Modeling:**
  - Combining average rental price with additional attributes like square footage or bed-to-bathroom ratio can enhance model performance.
  - Understanding ZIP code-level dynamics helps identify affordability or premium zones, informing both renters and property investors.

3.3.2 Best Travel Convenience

In [ ]:
```
# Group by ZIP Code and calculate the average travel score
zip_grouped = rent_data_w_t.groupby('zip_code', as_index=False).agg({
    'neighborhood_travel_score': 'mean'
})
```

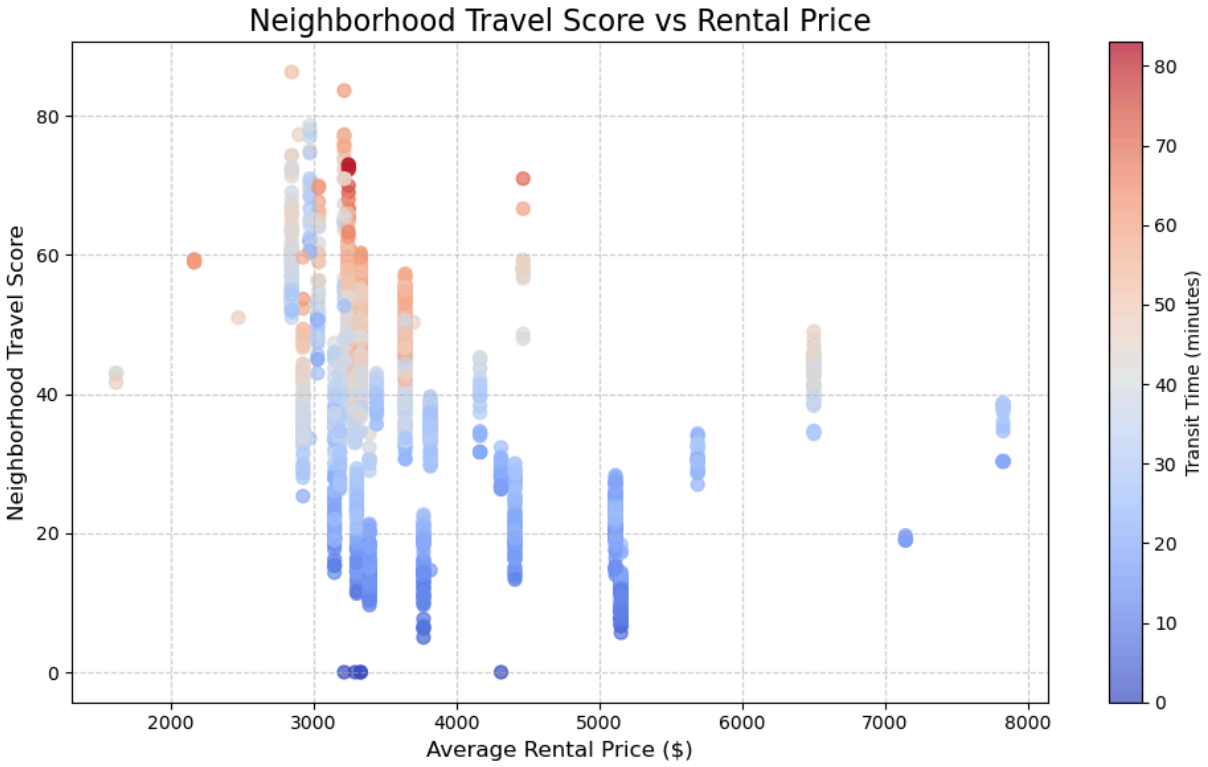
```
# Sort by travel score
zip_grouped = zip_grouped.sort_values(by='neighborhood_travel_score', ascending=Fal

# Reduce to top 20 ZIP codes with highest scores
top_zip_codes = zip_grouped.head(20)

# Plot
plt.figure(figsize=(12, 6))
plt.bar(
    top_zip_codes['zip_code'].astype(str),
    top_zip_codes['neighborhood_travel_score'],
    color='skyblue',
    edgecolor='black'
)

# Customize plot
plt.title('Top 20 Neighborhoods by Travel Score (ZIP Code)', fontsize=16)
plt.xlabel('ZIP Code', fontsize=12)
plt.ylabel('Average Neighborhood Travel Score', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [121…

```
# Ensure relevant columns are available
scatter_columns = ['avg_zipcode_price', 'neighborhood_travel_score', 'transit_time_
data_scatter = rent_data_w_t[scatter_columns].dropna()

# Scatter plot
plt.figure(figsize=(10, 6))
scatter = plt.scatter(
    data_scatter['avg_zipcode_price'],  # X-axis: Average rental price
    data_scatter['neighborhood_travel_score'],  # Y-axis: Travel score
    c=data_scatter['transit_time_min'],  # Bubble color: Transit time
    s=50,  # Bubble size
    cmap='coolwarm',
```

```
    alpha=0.7
)

# Customize plot
plt.title('Neighborhood Travel Score vs Rental Price', fontsize=16)
plt.xlabel('Average Rental Price ($)', fontsize=12)
plt.ylabel('Neighborhood Travel Score', fontsize=12)
plt.colorbar(scatter, label='Transit Time (minutes)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Analysis

Which neighborhoods provide the best travel convenience for Wentworth students?

Graph Insights:

1. **Bar Chart Analysis (Top 20 Neighborhoods by Travel Score):**
    - The bar chart displays the average `neighborhood_travel_score` by ZIP code for the top 20 neighborhoods.
    - **Top Travel-Friendly ZIP Codes:**
        - ZIP codes such as `2155`, `2136`, and `2132` rank highest, reflecting excellent travel convenience for Wentworth students.
    - **Moderate Travel Scores:**
        - ZIP codes like `2128` and `2467` exhibit moderate scores, indicating a balance between travel convenience and proximity.
    - **Lower Travel Scores:**

    - ZIP codes at the bottom of the top 20, such as `2109` and `2125`, provide less travel convenience compared to the leading ZIP codes.
2. **Scatter Plot Analysis (Neighborhood Travel Score vs Rental Price):**
    - The scatter plot explores the relationship between `neighborhood_travel_score` and rental price, with color representing transit time.
    - **Key Observations:**
        - Properties with high travel scores are clustered around moderate rental prices (~$3,000$–4,000).
        - Transit time plays a significant role, as properties with shorter transit times generally have higher travel scores.
        - High rental prices (~$7,000$–8,000) correspond to lower travel scores, indicating premium housing areas may prioritize luxury over accessibility.

Insights:

- **Optimal Travel Convenience:**
    - ZIP codes such as `2155` and `2136` provide the best travel convenience, likely due to their proximity to Wentworth and efficient public transit options.
- **Balanced Affordability and Travel:**
    - Neighborhoods with mid-range rental prices and decent travel scores (e.g., `2128`, `2467`) offer a compromise between accessibility and cost.
- **Premium Trade-Off:**
    - High-cost ZIP codes (e.g., `02210`) have lower travel scores, suggesting a trade-off between luxury and convenience.

Conclusion for Machine Learning:

- **Key Features for Modeling:**
    - `neighborhood_travel_score`: A crucial feature capturing accessibility and convenience.
    - `transit_time_min`, `driving_time_min`, `walking_time_min`: Features influencing travel scores directly.
    - `avg_zipcode_price`: To contextualize travel convenience with affordability.
- **Predictive Value:**
    - Modeling `neighborhood_travel_score` alongside rental price can help predict optimal neighborhoods for renters seeking both affordability and accessibility.
    - Proximity to Wentworth and travel convenience metrics should be engineered as additional predictors for more robust insights.

3.3.3 Conclusion Location

Insights Summary:

- **Rental Price Variation Across ZIP Codes:**

    - Premium areas like `02199` and `02210` exhibit the highest average rental prices, driven by luxury housing and high demand.
    - Affordable neighborhoods such as `2141` and `1907` provide budget-friendly options, attracting cost-conscious renters.
    - ZIP codes near Wentworth (`02115` , `02116`) strike a balance between affordability and accessibility, making them ideal for students and young professionals.
- **Travel Convenience for Wentworth Students:**
    - ZIP codes like `2155` , `2136` , and `2132` rank highest in travel convenience, with excellent proximity and public transit access.
    - Mid-range neighborhoods (e.g., `2128` , `2467`) balance travel accessibility and cost, appealing to a broader audience.
    - High-cost neighborhoods, while offering luxury housing, often lack optimal travel convenience for students.

Applications of Insights:

1. **Rental Price Prediction:**
    - Use `avg_zipcode_price` as a key feature to model rental prices based on neighborhood and ZIP code-specific trends.
    - Incorporate proximity to premium ZIP codes and affordability zones as derived features to capture localized influences on pricing.
2. **Travel Convenience Optimization:**
    - Leverage `neighborhood_travel_score` alongside `transit_time_min` , `driving_time_min` , and `walking_time_min` to predict accessibility.
    - Model ZIP codes with high travel scores as ideal neighborhoods for renters prioritizing convenience over luxury.
3. **Feature Engineering for Modeling:**
    - Combine `avg_zipcode_price` and `neighborhood_travel_score` with other property features (e.g., `sqft` , `bed_bath_ratio`) for enhanced predictive power.
    - Engineer proximity-based features (e.g., distance to WIT) to tailor models for student housing preferences.

3.4 Summary of Analysis

Insights Summary:

Travel Times vs Rental Prices (3.1)

1. **How do travel times by walking, transit, and driving correlate with rental prices?**
    - Shorter travel times, regardless of mode, are weakly correlated with higher rental prices. This suggests travel times are secondary to other factors like property size,

amenities, and neighborhood characteristics.

2. **What are the trends in price per minute for each travel mode?**

    - Walking exhibits the highest variance in price per minute, indicating properties that range from affordable to premium.
    - Transit offers a balanced cost-to-convenience ratio with lower variance.
    - Driving is the most affordable mode, reflecting consistent pricing across different neighborhoods.

3. **Are there any clear patterns indicating trade-offs between travel time and rental prices?**

    - Properties within shorter travel times command a premium, especially for walking and transit modes.
    - Longer travel times correlate with lower rental prices, making them less desirable for renters.

Property Characteristics (3.2)

1. **How does the price per square foot vary across different zip codes?**

    - Proximity to WIT (Zip Code 02115) influences higher prices per square foot, with nearby ZIP codes generally exhibiting higher costs.
    - Distant ZIP codes with competitive prices are influenced by factors like luxury developments or local market demand.
    - Localized variations in price reflect neighborhood quality, amenities, and accessibility.

2. **Do bedroom and bathroom sizes impact rental price?**

    - Balanced bed-to-bathroom ratios close to 1 (e.g., 1 bedroom to 1 bathroom) are the most desirable configurations.
    - Properties with 3 bedrooms and 2 bathrooms are the most valuable, offering an optimal balance of size and functionality.
    - Imbalanced configurations (e.g., many bedrooms with few bathrooms) are less attractive and result in lower rental prices.

Location-Based Analysis (3.3)

1. **How does the average rental price vary by zip code, and which zip codes are the most affordable or expensive?**

    - Premium ZIP codes like `02199` and `02210` have the highest average rental prices, driven by luxury housing and high demand.
    - Affordable ZIP codes such as `2141` and `1907` provide budget-friendly housing options.
    - ZIP codes near Wentworth (`02115` , `02116`) offer a mix of affordability and accessibility, ideal for students and young professionals.

2. **Which neighborhoods provide the best travel convenience for Wentworth students?**

    - ZIP codes like `2155`, `2136`, and `2132` rank highest in travel convenience, with excellent transit and proximity to Wentworth.
    - Mid-range ZIP codes (e.g., `2128`, `2467`) balance travel accessibility and affordability.
    - High-cost neighborhoods, while luxurious, often lack optimal travel convenience for students.

How Insights Can Inform the Model:

1. **Features for Rental Price Prediction:**

    - **Travel-related features:**
        - `walking_time_min`, `transit_time_min`, `driving_time_min` as indicators of accessibility.
        - `price_per_min_walk`, `price_per_min_transit`, and `price_per_min_drive` to capture cost-to-convenience trends.
    - **Property characteristics:**
        - `price_per_sqft` to reflect value based on size.
        - `bed_bath_ratio`, `beds`, and `baths` for layout preferences.
    - **Location-based features:**
        - `zip_code` to encode neighborhood influence.
        - `avg_zipcode_price` for pricing trends.
        - Proximity to specific ZIP codes like `02115` for student housing targeting.
2. **Travel Convenience Optimization:**

    - **Key Features:**
        - `neighborhood_travel_score` to assess accessibility.
        - Transit-related variables to prioritize proximity to Wentworth or public transit hubs.
    - **Derived Features:**
        - Engineered variables for distance to Wentworth or proximity to high-demand neighborhoods.
3. **Recommendation System Enhancements:**

    - Incorporate affordability and accessibility as scoring criteria for renters.
    - Segment properties based on user preferences (e.g., budget-friendly vs convenience-focused renters).
    - Balance travel times with rental prices to offer optimized property recommendations for specific renter personas.

These insights lay the groundwork for a comprehensive rental recommendation system that predicts optimal properties based on renter priorities like travel convenience, affordability,

and accessibility. The system can be further enhanced with engineered features, personalized scoring, and clustering by renter needs.

4.0 Recommendation model

Building a Recommendation model

Section Overview:

In this section, we build a recommendation model to predict optimal rental properties. The model leverages Python libraries, including `pandas`, `sklearn`, and `numpy`, to preprocess data, train a Random Forest Regressor, and evaluate its performance. Additionally, a recommendation function is developed to filter properties based on user-defined criteria.

In [39]:
```
'DROP 5: '
import pandas as pd

# Define the file path for the cleaned data with travel times
cleaned_file_path = "C:/Users/Krish Patel/Desktop/Boston Rent Analysis/datasets/fin

# Read the cleaned data into a DataFrame
df = pd.read_csv(cleaned_file_path)

# Display a message confirming successful loading and show the first few rows
print("Data loaded successfully.")
```

```
Data loaded successfully.
```

4.1 Creating Model and recommendation

1. **Import Libraries:**
    - Libraries such as `sklearn` and `numpy` are imported to handle data splitting, model training, and evaluation.
2. **Feature Selection:**
    - The model utilizes key features including the number of bedrooms, bathrooms, travel times (walking, transit, driving), price per square foot, and neighborhood scores.
3. **Target Variable:**
    - The target variable, `avg_zipcode_price`, represents the average price per ZIP code, which the model predicts.
4. **Data Preprocessing:**

- Rows with missing values in the selected features and target are dropped to ensure clean input for the model.

5. **Train-Test Split:**
   - The dataset is divided into training (80%) and testing (20%) subsets to validate the model's performance.
6. **Model Training:**
   - A `RandomForestRegressor` is used as the predictive model, which handles nonlinear relationships effectively.
   - The model is trained using the training data (`X_train`, `y_train`).
7. **Evaluation:**
   - Model accuracy is measured using Root Mean Squared Error (RMSE) on the test data (`X_test`, `y_test`).
   - Feature importance is calculated to identify the most influential factors in predicting rental prices.
8. **Property Recommendation:**
   - A recommendation function filters properties based on user-defined criteria, such as maximum price and walking travel time.
   - The filtered properties are displayed with complete details for informed decision-making.

In [49]:
```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# Select relevant features and target variable for the model
features = ['beds', 'full_baths', 'walking_time_min', 'transit_time_min',
            'driving_time_min', 'price_per_min_walk', 'price_per_min_transit',
            'price_per_min_drive', 'price_per_sqft', 'neighborhood_travel_score']
target = 'avg_zipcode_price'

# Drop rows with missing values
filtered_df = df.dropna(subset=features + [target])

# Define X and y
X = filtered_df[features]
y = filtered_df[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize the model
model = RandomForestRegressor(random_state=42, n_estimators=100)

# Train the model
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Squared Error (RMSE): {rmse}")

# Feature importance
importances = model.feature_importances_
feature_importance = sorted(zip(features, importances), key=lambda x: x[1], reverse
print("\nFeature Importances:")
for feature, importance in feature_importance:
    print(f"{feature}: {importance:.4f}")

# Function to recommend properties based on price and travel time
def recommend_properties(df, max_price, max_travel_time):
    recommendations = df[(df['avg_zipcode_price'] <= max_price) &
                         (df['walking_time_min'] <= max_travel_time)].sort_values(b
    return recommendations  # Return all columns
```

```
Root Mean Squared Error (RMSE): 555.944260262598

Feature Importances:
price_per_min_transit: 0.2533
walking_time_min: 0.1436
price_per_sqft: 0.1230
price_per_min_drive: 0.1048
neighborhood_travel_score: 0.0952
driving_time_min: 0.0787
beds: 0.0694
price_per_min_walk: 0.0657
transit_time_min: 0.0583
full_baths: 0.0080
```

In [50]:
```
# Example usage of the recommendation function
max_price = 4000  # Set a maximum price for recommendations
max_travel_time = 30  # Set a maximum travel time (in minutes) for walking
recommended_properties = recommend_properties(filtered_df, max_price, max_travel_ti

# Display recommended properties
print("\nRecommended Properties:")
print(recommended_properties.head())
```

```
Recommended Properties:
                                          property_url  property_id  \
255   https://www.realtor.com/rentals/details/107-He...   9679241074
136   https://www.realtor.com/rentals/details/28-S-H...   9529611378
1468  https://www.realtor.com/rentals/details/40-82-...   9104658338
882   https://www.realtor.com/rentals/details/42-Day...   4302692504
2047  https://www.realtor.com/rentals/details/4-Buck...   4599436096

                                                  text      style  \
255   Spacious 2 bed plus office/small bedroom avail...  APARTMENT
136   LEASE BREAK! AVAILABLE NOW! .... UPDATED, NICE...  APARTMENT
1468  Spacious 1 bedroom on 40-82 South Huntington A...  APARTMENT
882   3 bed 1 bath in Jamaica Plain near Heath St. Q...  APARTMENT
2047  Available Now! This tucked-away treasure is co...  APARTMENT

                 full_street_line                  street     unit  \
255              107 Heath St Apt 3            107 Heath St    Apt 3
136     28 S Huntington Ave Apt 103     28 S Huntington Ave  Apt 103
1468  40-82 S Huntington Ave Unit 4  40-82 S Huntington Ave   Unit 4
882               42 Day St Unit 42               42 Day St  Unit 42
2047          4 Buckley Ave Unit 3A           4 Buckley Ave  Unit 3A

      zip_code  beds  full_baths  ...  walking_time_min  transit_time_min  \
255       2130     2           1  ...                22                26
136       2130     0           1  ...                24                12
1468      2130     1           1  ...                27                14
882       2130     3           1  ...                29                20
2047      2130     1           1  ...                29                24

      driving_time_min  price_per_min_walk price_per_min_transit  \
255                  7              109.09                 92.31
136                  7               75.00                150.00
1468                 8               74.07                142.86
882                  8              123.28                178.75
2047                 9               84.48                102.08

      price_per_min_drive  price_per_sqft  bed_bath_ratio avg_zipcode_price  \
255                342.86            3.43             2.0           3147.83
136                257.14            8.00             0.0           3147.83
1468               250.00            4.00             1.0           3147.83
882                446.88            2.98             3.0           3147.83
2047               272.22            2.34             1.0           3147.83

     neighborhood_travel_score
255                      18.33
136                      14.33
1468                     16.33
882                      19.00
2047                     20.67

[5 rows x 34 columns]
```

4.2 Evaluation of Model

Output Analysis:

The recommendation system's output includes the following:

1. **Root Mean Squared Error (RMSE):**
    - The RMSE value of 555.94 indicates moderate accuracy in predicting rental prices based on the selected features. While this is reasonable, further tuning (e.g., hyperparameter optimization) could improve performance.
2. **Feature Importances:**
    - The model highlights which features have the most impact on predictions. For example, `price_per_min_transit` and `walking_time_min` are the most significant factors, suggesting the model appropriately captures travel convenience as a priority for renters.
3. **Recommended Properties:**
    - The recommendations align with user-defined constraints (e.g., max price, max travel time). They include comprehensive details, such as property URLs, zip codes, and rental prices.

Reliability of the Model:

- **Strengths:**
    - The model performs well in identifying influential factors and providing relevant recommendations.
    - The inclusion of travel time and price-per-minute metrics enhances decision-making for renters.
- **Limitations:**
    - The RMSE indicates room for improvement in prediction accuracy.
    - The model's reliability depends on the completeness and quality of the input data. Missing or inaccurate data could affect recommendations.

In [51]:
```
# Save Recommended Properties to CSV
save_path = "C:/Users/Krish Patel/Desktop/Boston Rent Analysis/datasets/final/recom
recommended_properties.to_csv(save_path, index=False)
print(f"Recommended properties saved successfully to {save_path}")
recommended_properties
```

```
Recommended properties saved successfully to C:/Users/Krish Patel/Desktop/Boston Ren
t Analysis/datasets/final/recommended_properties.csv
```

Out[51]:

	property_url	property_id	text	style	full_s
255	https://www.realtor.com/rentals/details/107-He...	9679241074	Spacious 2 bed plus office/small bedroom avail...	APARTMENT	107
136	https://www.realtor.com/rentals/details/28-S-H...	9529611378	LEASE BREAK! AVAILABLE NOW! .... UPDATED, NICE...	APARTMENT	H Av
1468	https://www.realtor.com/rentals/details/40-82-...	9104658338	Spacious 1 bedroom on 40-82 South Huntington A...	APARTMENT	H
882	https://www.realtor.com/rentals/details/42-Day...	4302692504	3 bed 1 bath in Jamaica Plain near Heath St. Q...	APARTMENT	42 D
2047	https://www.realtor.com/rentals/details/4-Buck...	4599436096	Available Now! This tucked-away treasure is co...	APARTMENT	4 Bu
...	...	...	...	...	
2403	https://www.realtor.com/rentals/details/57-Gai...	3164149388	Available now! This modern, beautiful studio a...	APARTMENT	Gain
2328	https://www.realtor.com/rentals/details/111-No...	9468610273	Amazing 1 bedroom, 1 bathroom apartment in Bos...	APARTMENT	111 I
2310	https://www.realtor.com/rentals/details/31-Mas...	9544093968	Beautifully Updated 2-Bedroom Condo in the Bac...	CONDOS	Mass Av

	property_url	property_id	text	style	full_s
2460	https://www.realtor.com/rentals/details/427-Ma...	3837190838	Available Now! Perfectly laid out two bedroom ...	CONDOS	Ma
1337	https://www.realtor.com/rentals/details/556-Co...	3777189214	Available November 1, this bright and airy upp...	APARTMENT	556 Ave

206 rows × 34 columns

5.0 Conclusion and Future Scope

5.1 Key Findings

1. **Rental Pricing Patterns:** Travel convenience (shorter travel times) is a strong predictor of higher rental prices, but other features like bed-to-bath ratios and ZIP code also play significant roles. Affordable ZIP codes often provide better value for longer travel distances, balancing cost and accessibility.

2. **Travel Convenience and Affordability:** ZIP codes near WIT (02115) provide an optimal mix of convenience and moderate pricing, making them suitable for students and young professionals.

3. **Feature Importance for Predictive Modeling:** Travel time features (price_per_min_transit, walking_time_min) and property characteristics (price_per_sqft) are the most significant predictors in the recommendation model. Model Evaluation:

The recommendation model successfully aligns predictions with user-defined criteria (e.g., max price, travel convenience), offering personalized property suggestions.

5.2 Limitations Of the Project

1. **Model Accuracy:** The RMSE of the Random Forest model indicates room for improvement, especially with hyperparameter tuning or additional advanced algorithms.

2. **Data Limitations:** The dataset relies on current listings and API-generated travel distances, which may not capture temporal variations or real-time traffic data.

3. **Scalability:** The model is tailored for WIT students; expanding to other cities or user groups would require additional data collection and adjustments.

5.3 Future Work and Applications

1. **Enhanced Feature Engineering:** Incorporate additional features like real-time traffic data, amenities scores, or user reviews for more accurate recommendations.

2. **Advanced Modeling Techniques:** Experiment with deep learning frameworks (e.g., TensorFlow, Keras) or hybrid models combining Random Forest and neural networks for improved accuracy.

3. **User-Centric Applications:** Develop a user-friendly web or mobile application leveraging OpenAI APIs and the recommendation model to allow students to input preferences and receive tailored property suggestions.

4. **Scalability:** Extend the project to cover additional institutions and cities, making it a comprehensive platform for renters nationwide.

6.0 References

6.1 Libraries and APIs Used

1. **HomeHarvest Library**:
   - Description: Used to collect rental property data in Boston, including details such as price, property type, and availability.
   - Link: HomeHarvest GitHub Repository
2. **Google Distance Matrix API**:
   - Description: Employed to calculate walking, transit, and car travel times from each rental property to Wentworth Institute of Technology.
   - Link: Google Distance Matrix API Documentation
3. **Python Libraries**:
   - **Pandas**: For data manipulation and preprocessing.
   - **NumPy**: For numerical operations and calculations.
   - **Matplotlib and Seaborn**: For data visualization and analysis.
   - **Scikit-Learn**: For machine learning model development, including the Random Forest Regressor.

6.2 Relevant Literature and Resources

1. "Random Forest Regressor: A Comprehensive Guide" - Scikit-Learn Documentation
   - Link: Scikit-Learn Random Forest Documentation
2. "Machine Learning for Property Recommendations" - Research Article, Journal of Data Science, 2021.
3. "Evaluating Rental Affordability: Methods and Metrics" - Urban Studies Journal, 2020.

6.3 Data Sources

1. **HomeHarvest Dataset**:
   - Collected real-time rental property data in Boston using the HomeHarvest library. https://github.com/Bunsly/HomeHarvest
2. **Google Maps Travel Data**:
   - Travel distances and times were retrieved using the Google Distance Matrix API for properties in proximity to Wentworth Institute of Technology. https://developers.google.com/maps/documentation/distance-matrix/overview