Configuration Manual

MSc Research Project
Programme Name

Yihui Zhang
Student ID: x23193824

School of Computing
National College of Ireland

Supervisor: Dr Anu Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Yihui Zhang
Student ID:	x23193824
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr Anu Sahni
Submission Due Date:	24/04/2025
Project Title:	An Integrated Hedonic Pricing and Predictive Modelling Approach: Comparing ML and DL for Dublin's Rental Market
Word Count:	1489
Page Count:	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Yihui Zhang
Date:	23rd April 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Yihui Zhang
x23193824

1 Introduction

This configuration report aims to provide a comprehensive overview of the working process for the research project titled " An Integrated Hedonic Pricing and Predictive Modelling Approach: Comparing ML and DL for Dublin's Rental Market."

The manual is structured as follows: Section 2 outlines the project's environment setup and configurations. Section 3 describes the dataset used in the research and details the preprocessing steps applied. The last section 4 presents the implementation process, including the data analysis workflow and the development of predictive models.

2 Environment Configuration

2.1 Hardware

The hardware employed in this research contained an M1 chip MacBook Pro and 16 GB of RAM.

Figure 1: Hardware

2.2 Software

The software used in this project consisted of Python 3, Jupyter Notebook, Google Maps API, and PostgreSQL. The Python libraries used in this project are summarised in Table

1.

Table 1: Python Libraries

Library Name	Usage
daftlistings	web scrape listings data from Daft.ie
pandas	ETL, and analysing dataframes
json	handle Geojson file
shapely	handle geometric objects
geopy	calculate distance between two coordinates
numpy	numerical operations
matplotlib	data visualisation
seaborn	data visualisation
sklearn	splits data, perform randomised hyperparameter search, feature scaling, evaluation metrics
tensorflow	building and training deep learning models

3 Datasets and Preprocessing

The datasets used in this project have 3 categories: neighbourhood, location, and internal factors.

3.1 Neighbourhood

The first category, the neighbourhood category, contains 4 datasets: the school dataset, the hospital dataset, the shopping centres, and the enterprise parks. School datasets, including primary schools and post-primary schools, were downloaded from the website [1]. Similarly, the hospital locations dataset was downloaded from [2]. The enterprise parks and shopping centres datasets were not found, and therefore gathered using the Google Maps API. The figure 2 shows the collection process.

[1] `https://www.gov.ie/en/collection/63363b-data-on-individual-schools/`
[2] `https://www.geohive.ie/datasets/feb34881088341bbbf80d86af6a4f333_0/about`

```python
import requests
import json
import pandas as pd
import time

API_KEY = "AIzaSyAEzaxILK7Yg6HO5fhCQLKKRdsEw-9EPM0"
QUERY = "shopping central in county dublin"
#QUERY = "enterprise park in county dublin"
URL = f"https://maps.googleapis.com/maps/api/place/textsearch/json?query={QUERY}&key={API_KEY}"

response = requests.get(URL)
data = response.json()

#information into a DataFrame
stations = []
def fetch_places(url):
    while url:
        response = requests.get(url)
        data = response.json()

        # Extract station details
        for place in data.get("results", []):
            stations.append({
                "Name": place["name"],
                "Latitude": place["geometry"]["location"]["lat"],
                "Longitude": place["geometry"]["location"]["lng"]
            })

        # Check if there's a next page
        next_page_token = data.get("next_page_token")
        if next_page_token:
            url = f"https://maps.googleapis.com/maps/api/place/nearbysearch/json?pagetoken={next_page_token}&key={API_KEY}"
            time.sleep(2)  # google requires a short delay before using the next_page_token
        else:
            url = None

fetch_places(URL)
df = pd.DataFrame(stations)

df.to_csv("shopping_centres_dublin.csv", index=False, encoding="utf-8")
#df.to_csv("enterprise_park_dublin.csv", index=False, encoding="utf-8")
```

Figure 2: Shopping Centres and Enterprise Parks Data Gathering

The other two datasets were filtered to contain only the facilities in Dublin. The code is displayed in the figure 3.

```python
## Dublin Eircode start
Dublin_eircode = ('D', 'K32', 'A41', 'A94', 'A42', 'A96', 'K78', 'K45', 'K36', 'A45', 'K56', 'K34', 'K67')
```

```python
## only keep the Dublin data
df_hospital = df_hospital[df_hospital['Eircode'].str.startswith(Dublin_eircode, na=False)]
```

```python
df_hospital = df_hospital.reset_index()
```

Figure 3: Filtered Data

3.2 Location

The location datasets used in the research were transportation data, including Luas and train stops. The dataset was manually collected by the author. The preprocessing process includes merging two datasets and deleting duplicate stops.

3.3 Internal Factors

The main dataset contains the key internal factors of a property were web scraped from the website Daft.ie (Figure 4).

```python
import daftlistings
import pandas as pd
from daftlistings import Daft, listing, Location, SearchType, PropertyType

property_types = [
    PropertyType.APARTMENT, PropertyType.BUNGALOW, PropertyType.DETACHED_HOUSE,
    PropertyType.DUPLEX, PropertyType.END_OF_TERRACE_HOUSE, PropertyType.HOUSE,
    PropertyType.SEMI_DETACHED_HOUSE, PropertyType.STUDIO_APARTMENT, PropertyType.TERRACED_HOUSE,
    PropertyType.TOWNHOUSE
]

daft = Daft()
daft.set_location(Location.DUBLIN)
daft.set_search_type(SearchType.RESIDENTIAL_RENT)
daft.set_property_type(PropertyType.APARTMENT)

for property_type in property_types:
    print(f"Property type: {property_type.name}")
    print("=" * 50)

    daft = Daft()
    daft.set_location(Location.DUBLIN)
    daft.set_search_type(SearchType.RESIDENTIAL_RENT)
    daft.set_property_type(property_type)

    listings = daft.search()

    for listing in listings:
        print("Title:", getattr(listing, 'title', 'Not available'))
        print("Price:", getattr(listing, 'price', 'Not available'))
        print("Bathrooms:", getattr(listing, 'bathrooms', 'Not available'))

        try:
            print("Bedrooms:", listing.bedrooms)
        except KeyError:
            print("Bedrooms: Not available")

        try:
            print("BER Rating:", listing.ber)
        except KeyError:
            print("BER Rating: Not available")

        print("Latitude:", getattr(listing, 'latitude', 'Not available'))
        print("Longitude:", getattr(listing, 'longitude', 'Not available'))

        print("Publish Date:", getattr(listing, 'publish_date', 'Not available'))
        print("\n" + "-" * 40 + "\n")

    print("\n" + "=" * 50 + "\n")
```

Figure 4: Web-scarping Listings

The preprocessing of the main data contains 6 steps:

Preprocessing Step 1: Merge the different months' listings into one big listing (Figure 5).

```python
import pandas as pd
import re
```

```python
csv_files = ["/Users/yihuizhang/Desktop/thesis/data/old_rental_data/rental_data_2024_Nov.csv",
             "/Users/yihuizhang/Desktop/thesis/data/old_rental_data/rental_data_2024_Dec.csv",
             "/Users/yihuizhang/Desktop/thesis/data/old_rental_data/rental_data_2025_Jan.csv",
             "/Users/yihuizhang/Desktop/thesis/data/old_rental_data/rental_data_2025_Feb.csv",
            "/Users/yihuizhang/Desktop/thesis/data/old_rental_data/rental_data_2025_March.csv",
            "/Users/yihuizhang/Desktop/thesis/data/rental_data_2025_April.csv"]
```

```python
dfs = [pd.read_csv(file) for file in csv_files]
```

```python
combined_df = pd.concat(dfs, ignore_index=True)
```

Figure 5: Preprocessing Step 1

Preprocessing Step 2: Remove duplicate listings, convert weekly rental price to monthly, and extract numbers (Figure 6).

```python
import pandas as pd

## weekly rental price to monthly
def extract_price(price):
    if isinstance(price, str):
        # numeric value
        match = re.search(r"\d+", price.replace(",", ""))
        if match:
            value = int(match.group())  # convert extracted number to integer
            # If it's per week, convert to monthly
            if "per week" in price.lower():
                return round((30 / 7) * value)  # Convert weekly price to monthly
            return value  # Otherwise, return the monthly price
    return None  # Return None if no valid number found

# Extract only numeric values from the "Price" column
combined_df["Price"] = combined_df["Price"].astype(str).apply(extract_price)

## extract numbers
def extract_number(column):
    if isinstance(column, str):
        match = re.search(r"\d+", column)  # Extract the first number
        if match:
            return int(match.group())  # Convert to integer
    return None  # Return None if no number found

combined_df["Bedrooms"] = combined_df["Bedrooms"].astype(str).apply(extract_number)
combined_df["Bathrooms"] = combined_df["Bathrooms"].astype(str).apply(extract_number)

## remove duplicates
new_combined_df = combined_df.drop_duplicates(subset=[col for col in combined_df.columns if col != "Publish_Date"])
sorted_new_combined_df = new_combined_df.sort_values(by="Address", ignore_index = True)
sorted_new_combined_df.to_csv("/Users/yihuizhang/Desktop/thesis/data/rantal_data_2024_nov_to_2025_April.csv")
```

Figure 6: Preprocessing Step 2

Preprocessing Step 3: Group data into 12 regions based on the latest electoral zone boundary [3] (Figure 7). Dummy variables were generated to represent each region (Figure 8).

```python
import json
import pandas as pd
from shapely.geometry import shape, Point

file = "/Users/yihuizhang/Desktop/thesis/data/rantal_data_2024_nov_to_2025_April.csv"
df = pd.read_csv(file)

# GeoJSON file
with open("/Users/yihuizhang/Desktop/thesis/data/National_Electoral_Boundaries_2023.geojson") as f:
    geodata = json.load(f)

## define the function to assign garda station to each address
def assign_zones(df):
    longitude = df['Longitude']
    latitude = df['Latitude']

    for feature in geodata['features']:
        # Convert GeoJSON geometry to Shapely shape
        polygon = shape(feature['geometry'])
        # Check if property is within the polygon
        if polygon.contains(Point(longitude, latitude)):
            return feature['properties']['ENG_NAME_VALUE']  # Name of the assigned Garda division

    return None

df['zone'] = df.apply(assign_zones, axis=1)
```

Figure 7: Preprocessing Step 3.1

```python
df_cleaned = pd.get_dummies(df_cleaned, columns = ['zone'], drop_first=True)
```

```python
df_cleaned.columns
```

Figure 8: Preprocessing Step 3.2

Preprocessing Step 4: Convert BER and property type to numerical levels (Figure 9).

[3] https://data-osi.opendata.arcgis.com/datasets/a37ad6a3a6ff47e4a5a0ff313b418448_0/explore

```python
## BER ranges from A1 to G
ber_mapping = {
    "A1": 1, "A2": 2, "A3": 3, "B1": 4, "B2": 5, "B3": 6,
    "C1": 7, "C2": 8, "C3": 9, "D1": 10, "D2": 11,"E1": 12,
    "E2": 13, "F": 14, "G": 15
}

df["BER_encoded"] = df["BER"].map(ber_mapping)

housetype_mapping = {
    "HOUSE": 1, "APARTMENT": 2, "STUDIO_APARTMENT": 3
}

df["property_type_encoded"] = df["Property_Type"].map(housetype_mapping)
```

Figure 9: Preprocessing Step 4

Preprocessing Step 5: Handle missing values (Figure 10).

```python
df.isnull().sum()

## missing value in zone
df[df["zone"].isnull()].index
df.loc[[639, 2081, 3481, 3482, 3483, 3484, 3485, 3486],"zone"] = "Dublin Fingal East (3)"
df.loc[[4427, 4428],"zone"] = "Dublin Bay South (4)"

## missing value in bedroom
df[df["Bedrooms"].isnull()]
df.loc[1282,"Bedrooms"] = 2
df.loc[1905,"Bedrooms"] = 2
df.loc[2280,"Bedrooms"] = 1
df.loc[2290,"Bedrooms"] = 4
df.loc[3161,"Bedrooms"] = 2
df.loc[3230,"Bedrooms"] = 1
df.loc[3445,"Bedrooms"] = 2
df.loc[3562,"Bedrooms"] = 2
df.loc[3581,"Bedrooms"] = 3

## missing value in bathroom
df_bathroom_null = df[df["Bathrooms"].isnull()]
df_bathroom_null[df_bathroom_null["Property_Type"] == "HOUSE"].index
df.loc[[2483, 2485, 3059, 3060, 3065, 3066, 3069, 3123, 3267, 3764, 3778,3863, 3864,
       3878, 4133, 4365,4818], "Bathrooms"] = 1
df.loc[[3861,3868,3876], "Bathrooms"] = 2
df.loc[4388,"Bathrooms"] = 1
df.loc[4399,"Bathrooms"] = 1
df.loc[4816,"Bathrooms"] = 1

df_bathroom_null[df_bathroom_null["Property_Type"] == "APARTMENT"]
df.loc[[3537, 3538, 3672, 3674, 3675, 3676], "Bathrooms"] = 1
df.loc[3759, "Bathrooms"] = 2
df.loc[3766, "Bathrooms"] = 3
df.loc[4394, "Bathrooms"] = 3
df.loc[4788, "Bathrooms"] = 3

index_studio_null_bathroom = df_bathroom_null[df_bathroom_null["Property_Type"] == "STUDIO_APARTMENT"].index
df.loc[index_studio_null_bathroom, "Bathrooms"] = 1

## missing value in BER
df["BER"].unique()
df.loc[df[df["BER"] == "A1A3"].index, "BER_encoded"] = 2
df.loc[df[df["BER"] == "A2A3"].index, "BER_encoded"] = 2
df['BER_encoded'] = df['BER_encoded'].fillna(df['BER_encoded'].median())
```

Figure 10: Preprocessing Step 5

Preprocessing Step 6: Find the nearest facilities for each property and calculate the distances in km (Figure 11).

```python
### function to find the shortest distance for each property
def find_nearest_facilities(property_lat, property_lon, facility_df):
    property_location = (property_lat, property_lon)

    nearest_facility = min(
        facility_df.itertuples(index=False),
        key=lambda row: geodesic(property_location, (row.Latitude, row.Longitude)).km
    )

    # geodesic distance
    nearest_distance = geodesic(property_location, (nearest_facility.Latitude, nearest_facility.Longitude)).km

    return nearest_facility.Name, nearest_distance

# example of applying function
df[['nearest_enter_park', 'nearest_enter_park_distance_km']] = df.apply(
    lambda row: find_nearest_facilities(row['Latitude'], row['Longitude'], df_jobs),
    axis=1,
    result_type='expand'
)
```

Figure 11: Preprocessing Step 6

4 Implementations

4.1 Data Analytics

There are 8 steps in data analytics: Data Analytics Step 1 & 2: Generate a distribution plot and box plot of rental prices, as well as distribution plots for distances to nearby facilities (Figure 12).

```
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 10), sharex='col')

# Histogram
sns.histplot(df["Price"], bins=500, color="black", ax=axes[0,0])
axes[0, 0].set_xlabel("Price")
axes[0, 0].set_ylabel("Count")
axes[0, 0].set_title("Distribution of Rental Price")

sns.histplot(df["nearest_shop_central_distance_km"], bins=500, color="black", ax=axes[0,1])
axes[0, 1].set_title("Distribution of Distance to Shopping Centre")
axes[0, 1].set_ylabel("Count")
axes[0, 1].set_xlabel("Distance to Shopping Centre (km)")

sns.histplot(df["nearest_hospital_distance_km"], bins=500, color="black", ax=axes[0,2])
axes[0, 2].set_title("Distribution of Distance to Hospital")
axes[0, 2].set_ylabel("Count")
axes[0, 2].set_xlabel("Distance to Hospital (km)")

sns.histplot(df["nearest_enter_park_distance_km"], bins=500, color="black", ax=axes[0,3])
axes[0, 3].set_title("Distribution of Distance to Enterprise Park")
axes[0, 3].set_ylabel("Count")
axes[0, 3].set_xlabel("Distance to Enterprise Park (km)")

# Boxplot
sns.boxplot(x=df["Price"], color="skyblue", ax=axes[1,0])
axes[1, 0].set_xlabel("Price")
axes[1, 0].set_title("Boxplot of Rental Price")

sns.histplot(df["nearest_primary_school_km"], bins=500, color="black", ax=axes[1,1])
axes[1, 1].set_title("Distribution of Distance to Primary School")
axes[1, 1].set_ylabel("Count")
axes[1, 1].set_xlabel("Distance to Primary School (km)")

sns.histplot(df["nearest_post_primary_school_km"], bins=500, color="black", ax=axes[1,2])
axes[1, 2].set_title("Distribution of Distance to Post Primary School")
axes[1, 2].set_ylabel("Count")
axes[1, 2].set_xlabel("Distance to Post Primary School (km)")

sns.histplot(df["nearest_traffic_stop_km"], bins=500, color="black", ax=axes[1,3])
axes[1, 3].set_title("Distribution of Distance to Traffic Stops")
axes[1, 3].set_ylabel("Count")
axes[1, 3].set_xlabel("Distance to Traffic Stops (km)")

# Adjust layout
plt.tight_layout()
plt.show()
```

Figure 12: Data Analytics Step 1 & 2

Data Analytics Step 3: Statistics of rental prices (Figure 13).

```python
## over all properties
Q1 = df["Price"].quantile(0.25)  # First quartile (25%)
Q3 = df["Price"].quantile(0.75)  # Third quartile (75%)
IQR = Q3 - Q1  # Interquartile range
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

mean_all = np.mean(df['Price']).round(3)
minimum_all = df['Price'].min().round(3)
maximum_all = df['Price'].max().round(3)
std_all = np.std(df['Price']).round(3)
median_all = np.median(df['Price']).round(3)

print(minimum_all)
print(mean_all)
print(median_all)
print(maximum_all)
print(std_all)

## properties in different zones
Q1_zone = df.groupby('zone')['Price'].quantile(0.25)  # First quartile (25%)
Q3_zone = df.groupby('zone')['Price'].quantile(0.75)  # Third quartile (75%)
IQR_zone = Q3 - Q1  # Interquartile range

# Define outliers (values outside 1.5 * IQR)
lower_bound_zone = Q1_zone - 1.5 * IQR_zone
upper_bound_zone = Q3_zone + 1.5 * IQR_zone

mean_zone = df.groupby('zone')['Price'].mean().round(1)
minimum_zone = df.groupby('zone')['Price'].min().round(1)
maximum_zone = df.groupby('zone')['Price'].max().round(1)
std_zone = df.groupby('zone')['Price'].std().round(1)
median_zone = df.groupby('zone')['Price'].median().round(1)

rental_stats = pd.DataFrame({
    'mean_price': mean_zone,
    'std_price': std_zone,
    'lower_bound': lower_bound_zone,
    'Q1': Q1_zone,
    'median_price': median_zone,
    'Q3': Q3_zone,
    'upper_bound': upper_bound_zone

}).reset_index()

rental_stats = rental_stats.sort_values(by='mean_price', ascending=False)

print(rental_stats)
```

Figure 13: Data Analytics Step 3

Data Analytics Step 4: Heatmaps of mean and median rental price across different regions (Figure 14).

```python
dublin_gdf = gpd.read_file("/Users/yihuizhang/Desktop/thesis/data/Dublin_Electoral_Boundaries.geojson")
dublin_gdf_merge = dublin_gdf.merge(rental_stats, left_on="ENG_NAME_VALUE", right_on="zone", how="left")

import matplotlib.pyplot as plt

# Create a figure with two subplots (one row, two plots)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(24, 12), sharex=True, sharey=True)

# Plot mean rental prices heatmap
dublin_gdf_merge.plot(column='mean_price', cmap="Blues", linewidth=0.1, edgecolor='black',
                      legend=True, ax=axes[0], missing_kwds={"color": "lightgrey", "label": "No Data"})
axes[0].set_title("Heatmap of Mean Rental Prices in Dublin", fontsize=14)
axes[0].set_xticks([])
axes[0].set_yticks([])

# Plot median rental prices heatmap
dublin_gdf_merge.plot(column='median_price', cmap="Blues", linewidth=0.1, edgecolor='black',
                      legend=True, ax=axes[1], missing_kwds={"color": "lightgrey", "label": "No Data"})
axes[1].set_title("Heatmap of Median Rental Prices in Dublin", fontsize=14)
axes[1].set_xticks([])
axes[1].set_yticks([])

# Adjust layout for better spacing
plt.tight_layout()

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

Figure 14: Data Analytics Step 4

Data Analytics Step 5: Statistical significance (Figure 15) of rental price between properties within and beyond 15-minute walking distance (1.25 km).

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import shapiro, mannwhitneyu, ttest_ind

def compare_price_significance(df, distance_threshold=1.25, distance_col="nearest_traffic_stop_km", price_col="Price", title="Traffic"):

    # split data based on the threshold
    df_within = df[df[distance_col] <= distance_threshold]
    df_outside = df.drop(df_within.index)

    # check normality using Shapiro-Wilk test
    stat_within, p_within = shapiro(df_within[price_col])
    stat_outside, p_outside = shapiro(df_outside[price_col])

    normal_within = p_within >= 0.05
    normal_outside = p_outside >= 0.05

    print(f"If {title} within {distance_threshold} km is normally distributed: {normal_within}")
    print(f"If {title} outside {distance_threshold} km is normally distributed: {normal_outside}")
    print("----")

    results = {
        "Normality Test": {
            "Within Threshold": normal_within,
            "Outside Threshold": normal_outside
        }
    }

    # choose appropriate statistical test
    if normal_within and normal_outside:
        # Perform t-test if data is normally distributed
        t_stat, p_value = ttest_ind(df_within[price_col], df_outside[price_col], equal_var=False)
        print(f"T-test result: T-statistic = {t_stat}, P-value = {p_value}")
        test_type = "T-test"
    else:
        # perform Mann-Whitney U test if data is not normally distributed
        u_stat, p_value = mannwhitneyu(df_within[price_col], df_outside[price_col], alternative='two-sided')
        print(f"Mann-Whitney U test result: U-statistic = {u_stat}, P-value = {p_value}")
        test_type = "Mann-Whitney U test"

    # interpret significance
    significance = p_value < 0.05
    print(f"The difference is {'statistically significant' if significance else 'NOT statistically significant'}.")

    results["Test"] = {
        "Type": test_type,
        "Statistic": t_stat if test_type == "T-test" else u_stat,
        "P-value": p_value,
        "Significant": significance
    }

    return results
```

```python
### traffic
compare_price_significance(df, 1.25, "nearest_traffic_stop_km", "Price", "Traffic")
```

Figure 15: Data Analytics Step 5

Data Analytics Step 6: Box plots of rental prices of different internal factors (Figure 16).

```python
property_type_labels = {1: "House", 2: "Apartment", 3: "Studio"}
ber_type_labels = {
    1: "A1", 2: "A2", 3: "A3", 4: "B1", 5: "B2", 6: "B3",
    7: "C1", 8: "C2", 9: "C3", 10: "D1", 11: "D2",
    12: "E1", 13: "E2", 14: "F", 15: "G"
}

# create a figure with 2 rows and 2 columns
fig, axes = plt.subplots(2, 2, figsize=(18, 12))

# boxplot 1: Property Type
sns.boxplot(x='property_type_encoded', y='Price', data=df, color="skyblue", ax=axes[0, 0])
axes[0, 0].set_title("By Property Type")
axes[0, 0].set_xlabel("Property Type")
axes[0, 0].set_ylabel("Price")
legend_text = "Property Type:\n" + "\n".join([f"{key}: {value}" for key, value in property_type_labels.items()])
axes[0, 0].text(2.6, df["Price"].max()*1.03, legend_text, fontsize=10, verticalalignment='top', bbox=dict(facecolor='white', alpha=0.5))

# boxplot 2: BER
sns.boxplot(x='BER_encoded', y='Price', data=df, color="skyblue", ax=axes[0, 1])
axes[0, 1].set_title("By BER")
axes[0, 1].set_xlabel("BER")
axes[0, 1].set_ylabel("")
legend_text = "BER Type:\n" + "\n".join([f"{key}: {value}" for key, value in ber_type_labels.items()])
axes[0, 1].text(15, df["Price"].max()*1.03, legend_text, fontsize=10, verticalalignment='top', bbox=dict(facecolor='white', alpha=0.5))

# boxplot 3: Bathrooms
sns.boxplot(x='Bathrooms', y='Price', data=df, color="skyblue", ax=axes[1, 0])
axes[1, 0].set_title("By Number of Bathrooms")
axes[1, 0].set_xlabel("Number of Bathrooms")
axes[1, 0].set_ylabel("Price")

# boxplot 4: Bedrooms
sns.boxplot(x='Bedrooms', y='Price', data=df, color="skyblue", ax=axes[1, 1])
axes[1, 1].set_title("By Number of Bedrooms")
axes[1, 1].set_xlabel("Number of Bedrooms")
axes[1, 1].set_ylabel("")

# layout
plt.suptitle("Boxplots of Property Prices by Different Features", fontsize=18)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Figure 16: Data Analytics Step 6

Data Analytics Step 7: Relationship between time and rental prices (Figure 17).

```python
df_all_month_mean = df_date_sorted.groupby('date_month')['Price'].mean()
df_all_month_median = df_date_sorted.groupby('date_month')['Price'].median()
```

```python
df_all_mean = df_all_month_mean.reset_index()
df_all_mean.columns = ['date', 'Average Price']

df_all_median = df_all_month_median.reset_index()
df_all_median.columns = ['date', 'Median Price']
```

```python
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Example plots (Replace with your actual data)
axes[0].plot(df_all_mean["date"], df_all_mean["Average Price"],alpha=0.8, color = 'black')
axes[0].set_title("Date vs Average Price")
axes[0].set_ylabel("Average Price")
axes[0].set_xlabel("Date")

axes[1].plot(df_all_median["date"], df_all_median["Median Price"],alpha=0.8, color = 'black')
axes[1].set_title("Date vs Median Price")
axes[1].set_ylabel("Median Price")
axes[1].set_xlabel("Date")

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

Figure 17: Data Analytics Step 7

Data Analytics Step 8: The Rent-to-Income Ratio (RIR) was calculated by dividing the median income by the median rent across various household types [4].

4.2 Log Transformation

Due to the right skew of the rental price and distance to facilities distributions, the log transformation was performed on these variables.

[4]https://www.cso.ie/en/releasesandpublications/ep/p-silc/surveyonincomeandlivingconditionssilc2024 householdincome/

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/cleaned_data.csv")

df['Price_log'] = np.log1p(df['Price'])
df['enter_park_distance_km_log'] = np.log1p(df['nearest_enter_park_distance_km'])
df['shop_central_distance_km_log'] = np.log1p(df['nearest_shop_central_distance_km'])
df['hospital_distance_km_log'] = np.log1p(df['nearest_hospital_distance_km'])
df['traffic_stop_km_log'] = np.log1p(df['nearest_traffic_stop_km'])
df['primary_school_km_log'] = np.log1p(df['nearest_primary_school_km'])
df['post_primary_school_km_log'] = np.log1p(df['nearest_post_primary_school_km'])
```

Figure 18: Log Transformation

4.3 Training and Testing Split

The outliers were first removed, the key variables were selected, and the dataset was divided into 80% training and 20% testing sets.

```python
Q1 = df["Price"].quantile(0.25)  # First quartile (25%)
Q3 = df["Price"].quantile(0.75)  # Third quartile (75%)
IQR = Q3 - Q1  # Interquartile range

# Define outliers (values outside 1.5 * IQR)
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Count outliers
outliers = df[(df["Price"] < lower_bound) | (df["Price"] > upper_bound)]
print(f"Number of outliers: {outliers.shape[0]}")
```

```
Number of outliers: 283
```

```python
df_non_outliers = df.drop(outliers.index, axis=0)
```

```python
X = df_non_outliers[['Bedrooms', 'Bathrooms', 'Latitude', 'Longitude',
       'BER_encoded', 'property_type_encoded',
       'count_1.25_enter_park', 'count_1.25_shop_central',
       'count_1.25_hospital', 'count_1.25_traffic',
       'count_1.25_primary_school', 'count_1.25_post_primary_school',
       'count_1.25_facilities','date_numeric',
       'zone_Dublin Bay South (4)', 'zone_Dublin Central (4)',
       'zone_Dublin Fingal East (3)', 'zone_Dublin Fingal West (3)',
       'zone_Dublin Mid-West (5)', 'zone_Dublin North-West (3)',
       'zone_Dublin Rathdown (4)', 'zone_Dublin South-Central (4)',
       'zone_Dublin South-West (5)', 'zone_Dublin West (5)',
       'zone_Dún Laoghaire (4)','enter_park_distance_km_log',
       'shop_central_distance_km_log', 'hospital_distance_km_log',
       'traffic_stop_km_log', 'primary_school_km_log',
       'post_primary_school_km_log']]
y = df_non_outliers["Price_log"]
```

```python
# Split data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
X_train.to_csv("/Users/yihuizhang/Desktop/thesis/data/x_train.csv", index=False)
X_test.to_csv("/Users/yihuizhang/Desktop/thesis/data/x_test.csv", index=False)
y_train.to_csv("/Users/yihuizhang/Desktop/thesis/data/y_train.csv", index=False)
y_test.to_csv("/Users/yihuizhang/Desktop/thesis/data/y_test.csv", index=False)
```

Figure 19: Training & Testing Split

4.4 Model 1 Random Forest Implementation

The grid search was first performed on the random forest model to find the optimised parameters (Figure 20).

```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    'n_estimators': [100, 300, 500], ## number of trees
    'max_depth': [10, 20, None], ## maximum depth of each tree
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Initialize Random Forest Regressor
rf = RandomForestRegressor(random_state=42)

# Perform Grid Search
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", grid_search.best_params_)
```

Figure 20: Random Forest - Grid Search

The rental prices were predicted by using these parameters (Figure 21), the feature importance histogram was plotted to evaluate the key factors (Figure 22), and the metrics MAE, MSE, $RMSE$, R^2, and $\%RMSE$, $\%MAE$ were used to check the model's performance (Figure 23).

```python
best_model_rf = RandomForestRegressor(
    max_depth=None,
    max_features='sqrt',
    min_samples_leaf=1,
    min_samples_split=2,
    n_estimators=500,
    random_state=42
)
best_model_rf.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_features='sqrt', n_estimators=500, random_state=42)
```

Figure 21: Parameters of Random Forest

```python
import matplotlib.pyplot as plt

# Get feature importance scores
importance = best_model_rf.feature_importances_
feature_names = X.columns

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_names, importance, color="skyblue")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance")
plt.show()
```

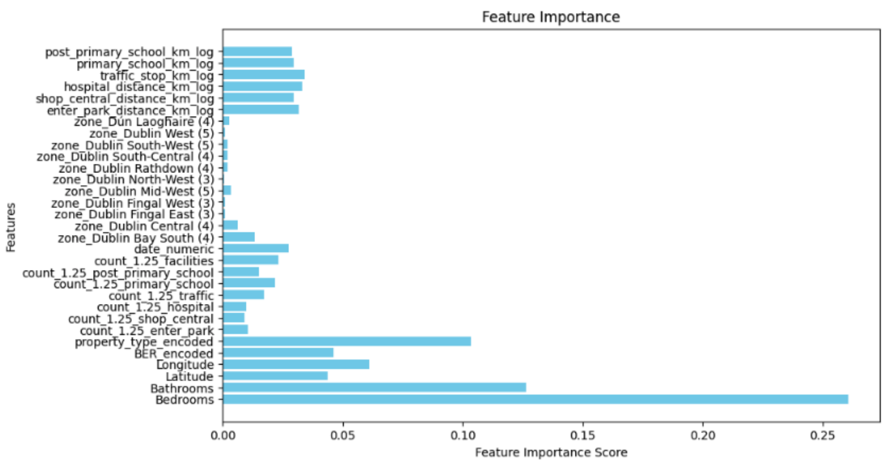


Figure 22: Feature Importance

```
y_pred = best_model_rf.predict(X_test)
```

```
y_pred_original = np.expm1(y_pred)
y_test_original = np.expm1(y_test).values
```

```
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 240.10
Mean Squared Error (MSE): 116752.79
Root Mean Squared Error (RMSE): 341.69
R-squared (R²): 0.77
%rmse: 13.96
%mae: 10.61
```

Figure 23: Random Forest - Metrics

4.5 Model 2 XGBoost Implementation

To reduce the running time, the Random Search algorithm was performed on XGBoost to determine the best parameters. As a result, the best parameters are shown in Figure 24.

```
from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
```

```
# Define parameter grid
param_grid = {
    'n_estimators': [100, 300, 500],   # Number of trees
    'learning_rate': [0.01, 0.05, 0.1, 0.2],  # Step size shrinkage
    'max_depth': [3, 5, 7, 9],  # Maximum tree depth
    'subsample': [0.6, 0.8, 1.0],  # Row sampling
    'colsample_bytree': [0.6, 0.8, 1.0],  # Feature sampling
    'gamma': [0, 0.1, 0.2, 0.3],  # Minimum loss reduction required to split
    'reg_alpha': [0, 0.01, 0.1, 1],  # L1 regularization (lasso)
    'reg_lambda': [1, 1.5, 2, 5],  # L2 regularization (ridge)
}

# Initialize XGBoost model
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Use RandomizedSearchCV for faster tuning
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_grid,
    n_iter=100,  # Number of random combinations to try
    scoring='r2',
    cv=5,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1,
    random_state=42
)

# Fit the model
random_search.fit(X_train, y_train)

# Best parameters
print("Best Parameters:", random_search.best_params_)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'subsample': 1.0, 'reg_lambda': 5, 'reg_alpha': 0.1, 'n_estimators': 500, 'max_depth': 7, 'learning_rate': 0.05, 'gamma':
0, 'colsample_bytree': 0.6}
```

Figure 24: XGBoost - Random Search

The metric results are shown in Figure 25.

```
best_xgb = XGBRegressor(
    objective='reg:squarederror',
    subsample=1.0,
    reg_lambda=5,
    reg_alpha=0.1,
    n_estimators=500,
    max_depth=7,
    learning_rate=0.05,
    gamma=0,
    colsample_bytree=0.6,
    random_state=42
)

best_xgb.fit(X_train, y_train)
y_pred = best_xgb.predict(X_test)
```

```
y_pred_original = np.expm1(y_pred)
y_test_original = np.expm1(y_test).values
```

```
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 233.82
Mean Squared Error (MSE): 108867.02
Root Mean Squared Error (RMSE): 329.95
R-squared (R²): 0.78
%rmse: 13.48
%mae: 10.45
```

Figure 25: XGBoost - Metrics

4.6 Model 3 SVR Implementation

The hyperparameter tuning was performed on SVR implementation shown in Figure 26.

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
X_train = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/x_train.csv")
X_test = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/x_test.csv")
y_train = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/y_train.csv")
y_test = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/y_test.csv")
```

Hyperparameter Tuning

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# SVR is sensitive to feature scaling, so we use a pipeline with StandardScaler
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svr', SVR())
])

# Define parameter grid for SVR
param_grid = {
    'svr__kernel': ['linear', 'rbf', 'poly'],
    'svr__C': [0.1, 1, 10, 100],
    'svr__epsilon': [0.1, 0.2, 0.5],
    'svr__gamma': ['scale', 'auto']  # Only applicable to 'rbf' and 'poly'
}

# Initialize GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='r2', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

```
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
```

```python
print("Best Parameters:", grid_search.best_estimator_)
```

```
Best Parameters: Pipeline(steps=[('scaler', StandardScaler()), ('svr', SVR(C=1, gamma='auto'))])
```

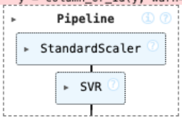
Figure 26: SVR - Hypermaramenter Tuning

The metric results are shown in Figure 27.

```python
best_svr_model = grid_search.best_estimator_

# Fit the best svr model (optional, already fitted during grid search)
best_svr_model.fit(X_train, y_train)
```

```
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/utils/validation.py:1300: DataConversionWarning: A column
-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```




```python
y_pred_original = np.expm1(y_pred)
y_test_original = np.expm1(y_test).values
```

```python
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 554.26
Mean Squared Error (MSE): 505483.76
Root Mean Squared Error (RMSE): 710.97
R-squared (R²): -0.00
%rmse: 29.06
%mae: 24.33
```

Figure 27: SVR - Metrics

4.7 Model 4 LightGBM Implementation

The hyperparameter tuning was performed on LightGBM implementation shown in Figure 28.

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
X_train = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/x_train.csv")
X_test = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/x_test.csv")
y_train = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/y_train.csv")
y_test = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/y_test.csv")
```

Hyperparameter Tuning

```python
from sklearn.model_selection import RandomizedSearchCV
from lightgbm import LGBMRegressor
# define parameter distributions for LightGBM
param_dist = {
    'n_estimators': [100, 300, 500],'learning_rate': [0.01, 0.05, 0.1], 'max_depth': [-1, 10, 20],
    'num_leaves': [31, 50, 70], 'min_child_samples': [5, 10, 20], 'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0]
}
# initialize LightGBM Regressor
lgbm = LGBMRegressor(random_state=42)

# Use RandomizedSearchCV
random_search = RandomizedSearchCV(
    lgbm,
    param_distributions=param_dist,
    n_iter=25,
    scoring='r2',
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001549 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2469
[LightGBM] [Info] Number of data points in the train set: 2992, number of used features: 31
[LightGBM] [Info] Start training from score 7.739275
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```python
print("Best Parameters:", random_search.best_params_)
```

```
Best Parameters: {'subsample': 0.7, 'num_leaves': 50, 'n_estimators': 500, 'min_child_samples': 5, 'max_depth': -1, 'learning_rate': 0.05, 'c
olsample_bytree': 0.8}
```

Figure 28: LightGBM - Hypermarameter Tuning

The metric results are shown in Figure 29.

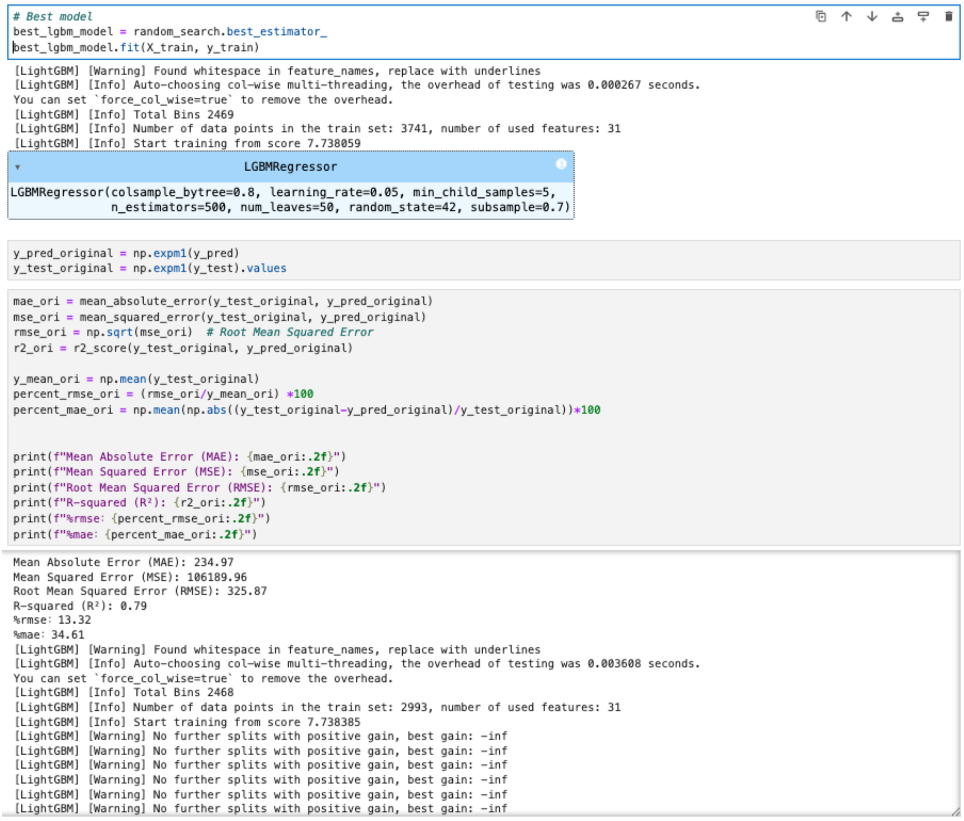


Figure 29: LightGBM - Metrics

4.8 Model 5 GBR Implementation

The hyperparameter tuning was performed on GBR implementation shown in Figure 30.

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
X_train = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/x_train.csv")
X_test = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/x_test.csv")
y_train = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/y_train.csv")
y_test = pd.read_csv("/Users/yihuizhang/Desktop/thesis/data/y_test.csv")
```

Hyperparameter Tuning

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingRegressor
# Define parameter distributions for Gradient Boosting Regressor
param_dist = {
    'n_estimators': [100, 300, 500],'learning_rate': [0.01, 0.05, 0.1],'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],'min_samples_leaf': [1, 2, 4],'subsample': [0.8, 1.0]
}
# initialize GBR
gbr = GradientBoostingRegressor(random_state=42)
# RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=gbr,
    param_distributions=param_dist,
    n_iter=25,  # Reduce this to speed up or increase for more thorough search
    cv=5,
    scoring='r2',
    verbose=2,
    random_state=42,
    n_jobs=-1
)
# Fit the model
random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
```

```python
print("Best Parameters:", random_search.best_params_)
```

```
Best Parameters: {'subsample': 0.8, 'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_depth': 5, 'learning_rate': 0.0
5}
```

Figure 30: GBR - Hypermarameter Tuning

The metric results are shown in Figure 31.

```python
# Best model
best_gbr_model = random_search.best_estimator_
best_gbr_model.fit(X_train, y_train)
```

```
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:668: DataConversionWarning: A column-vect
or y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
```

```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.05, max_depth=5, min_samples_leaf=4,
                          min_samples_split=5, n_estimators=500,
                          random_state=42, subsample=0.8)
```

```python
y_pred_original = np.expm1(y_pred)
y_test_original = np.expm1(y_test).values
```

```python
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 235.80
Mean Squared Error (MSE): 106750.21
Root Mean Squared Error (RMSE): 326.73
R-squared (R²): 0.79
%rmse: 13.35
%mae: 34.50
```

Figure 31: GBR - Metrics

4.9 Model 6 RNNS Implementation

The first step in deep learning implementation was to compare the performance between 3 selected DNN architectures.

The first architecture is shown in Figure 32, and the results are shown in Figure 33.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
model = Sequential([
    Dense(40, input_shape=(X_train_scaled.shape[1],), activation='relu'),
    # Dropout(0.2),
    Dense(20, activation='relu'),
    Dense(10, activation='relu'),
    Dense(1)
])
```

```
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `i
nput_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the mo
del instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])
```

```
# Callbacks
early_stopping = EarlyStopping(
    patience=5,
    restore_best_weights=True,
    monitor='val_loss')

lr_reduction = ReduceLROnPlateau(
    patience=10,
    factor=0.5,
    min_lr=1e-6,
    monitor='val_loss')
```

```
# Train the model with validation split
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.2,
    epochs=40,
    batch_size=16,
    callbacks=[early_stopping, lr_reduction],
    verbose=1
)
```

Figure 32: Architecture 1

```
y_pred = model.predict(X_test_scaled).flatten()
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 747us/step
```

```
### original value
y_test_original = np.expm1(y_test).values
y_pred_original = np.expm1(y_pred)
```

```
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 378.91
Mean Squared Error (MSE): 318218.50
Root Mean Squared Error (RMSE): 564.11
R-squared (R²): 0.37
%rmse: 23.05
%mae: 39.35
```

Figure 33: Architecture 1 - results

The second architecture is shown in Figure 34, and the results are shown in Figure 35.

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
model = Sequential([
    Dense(128, input_shape=(X_train_scaled.shape[1],), activation='relu'),
    # Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])
```

```
/Users/yihuizhang/anaconda3/envs/pythonProject/lib/python3.11/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `i
nput_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the mo
del instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```python
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])
```

```python
# Callbacks
early_stopping = EarlyStopping(
    patience=5,
    restore_best_weights=True,
    monitor='val_loss')

lr_reduction = ReduceLROnPlateau(
    patience=10,
    factor=0.5,
    min_lr=1e-6,
    monitor='val_loss')
```

```python
# Train the model with validation split
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.2,
    epochs=300,
    batch_size=64,
    callbacks=[early_stopping, lr_reduction],
    verbose=1
)
```

Figure 34: Architecture 2

```python
y_pred = model.predict(X_test_scaled).flatten()
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 810us/step
```

```python
### original value
y_test_original = np.expm1(y_test).values
y_pred_original = np.expm1(y_pred)
```

```python
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 353.32
Mean Squared Error (MSE): 265824.42
Root Mean Squared Error (RMSE): 515.58
R-squared (R²): 0.47
%rmse: 21.07
%mae: 36.68
```

Figure 35: Architecture 2 - results

The third architecture is shown in Figure 36, and the results are displayed in Figure 37.

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
def create_model(learning_rate=0.001, dropout_rate=0.2, activation = 'relu'):

    model = Sequential([
        Dense(100, input_shape=(X_train_scaled.shape[1],), activation=activation),
        Dropout(dropout_rate),
        Dense(75, activation=activation),
        Dropout(dropout_rate),
        Dense(50, activation=activation),
        Dense(1)
    ])

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

    return model
```

```python
regressor = KerasRegressor(model=create_model, verbose=0)
```

```python
regressor.get_params().keys()
```

```
dict_keys(['model', 'build_fn', 'warm_start', 'random_state', 'optimizer', 'loss', 'metrics', 'batch_size', 'validation_batch_size', 'verbos
e', 'callbacks', 'validation_split', 'shuffle', 'run_eagerly', 'epochs'])
```

```python
param_dist = {
    'batch_size': [10, 20], ## ok
    'epochs': [50, 75, 100], ## ok
    'model__learning_rate': [1e-2, 1e-3, 1e-4],
    'model__dropout_rate': [0.1, 0.2],
    'model__activation': ['relu', 'sigmoid']
}
```

```python
random_search = RandomizedSearchCV(
    estimator=regressor,
    param_distributions=param_dist,
    n_iter=50, ## number of combinations start from 30
    cv=3, # three-fold cross-validation
    verbose=2, ## detailed infor
    random_state=42,
    scoring='neg_mean_squared_error'  # Recommended for regression
)
```

```python
random_search_result = random_search.fit(X_train_scaled, y_train)
```

Figure 36: Architecture 3

```python
best_model = random_search_result.best_estimator_
y_pred = best_model.predict(X_test_scaled)
```

```python
### original value
y_test_original = np.expm1(y_test).values
y_pred_original = np.expm1(y_pred)
```

```python
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 279.55
Mean Squared Error (MSE): 141446.98
Root Mean Squared Error (RMSE): 376.09
R-squared (R²): 0.72
%rmse: 15.37
%mae: 13.10
```

Figure 37: Architecture 3 - results

The results indicate that the third architecture outperformed the others. Therefore, the hyperparameter tuning was performed on the third model to determine the optimised parameters (Figure 38), and the results are shown in Figure 39.

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
def create_model(learning_rate=0.001, dropout_rate1=0.2, dropout_rate2=0.2,
                 neurons1=100, neurons2=75, neurons3=50):

    model = Sequential([
        Dense(neurons1, input_shape=(X_train_scaled.shape[1],), activation='relu'),
        Dropout(dropout_rate1),
        Dense(neurons2, activation='relu'),
        Dropout(dropout_rate2),
        Dense(neurons3, activation='relu'),
        Dense(1)
    ])

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

    return model
```

```python
regressor = KerasRegressor(model=create_model, verbose=0)
```

```python
param_dist = {
    'batch_size': [10, 20, 30, 40], ## ok
    'epochs': [100, 50, 75, 125], ## ok
    'model__learning_rate': [1e-2, 1e-3, 1e-4],
    'model__dropout_rate1': [0.1, 0.2, 0.5, 0],
    'model__dropout_rate2': [0.1, 0.2, 0.5, 0],
    'model__neurons1': [100],
    'model__neurons2': [75],
    'model__neurons3': [50]
}
```

```python
random_search = RandomizedSearchCV(
    estimator=regressor,
    param_distributions=param_dist,
    n_iter=100, ## number of combinations start from 30 to 50 to 100.
    cv=3, # three-fold cross-validation
    verbose=2, ## detailed infor
    random_state=42,
    scoring='neg_mean_squared_error'  # Recommended for regression
)
```

```python
random_search_result = random_search.fit(X_train_scaled, y_train)
```

Figure 38: Architecture 3 - Best Parameters

```python
best_model = random_search_result.best_estimator_
y_pred = best_model.predict(X_test_scaled)
```

```python
### original value
y_test_original = np.expm1(y_test).values
y_pred_original = np.expm1(y_pred)
```

```python
mae_ori = mean_absolute_error(y_test_original, y_pred_original)
mse_ori = mean_squared_error(y_test_original, y_pred_original)
rmse_ori = np.sqrt(mse_ori)  # Root Mean Squared Error
r2_ori = r2_score(y_test_original, y_pred_original)

y_mean_ori = np.mean(y_test_original)
percent_rmse_ori = (rmse_ori/y_mean_ori) *100
percent_mae_ori = np.mean(np.abs((y_test_original-y_pred_original)/y_test_original))*100


print(f"Mean Absolute Error (MAE): {mae_ori:.2f}")
print(f"Mean Squared Error (MSE): {mse_ori:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_ori:.2f}")
print(f"R-squared (R²): {r2_ori:.2f}")
print(f"%rmse: {percent_rmse_ori:.2f}")
print(f"%mae: {percent_mae_ori:.2f}")
```

```
Mean Absolute Error (MAE): 266.15
Mean Squared Error (MSE): 132662.30
Root Mean Squared Error (RMSE): 364.23
R-squared (R²): 0.74
%rmse: 14.89
%mae: 12.16
```

Figure 39: Architecture 3 - Best Results