36106_25AU-AT1_25589351_experiment_0

March 29, 2025

1 Experiment Notebook

1.1 0. Setup Environment

1.1.1 0.a Install Mandatory Packages

Do not modify this code before running it

[ ]:
```
# Do not modify this code

import os
import sys
from pathlib import Path

COURSE = "36106"
ASSIGNMENT = "AT1"
DATA = "data"

asgmt_path = f"{COURSE}/assignment/{ASSIGNMENT}"
root_path = "./"

print("###### Install required Python packages ######")
! pip install -r https://raw.githubusercontent.com/aso-uts/labs_datasets/main/
  ↪36106-mlaa/requirements.txt

if os.getenv("COLAB_RELEASE_TAG"):

    from google.colab import drive
    from pathlib import Path

    print("\n###### Connect to personal Google Drive ######")
    gdrive_path = "/content/gdrive"
    drive.mount(gdrive_path)
    root_path = f"{gdrive_path}/MyDrive/"

print("\n###### Setting up folders ######")
folder_path = Path(f"{root_path}/{asgmt_path}/") / DATA
```

```
folder_path.mkdir(parents=True, exist_ok=True)
print(f"\nYou can now save your data files in: {folder_path}")

if os.getenv("COLAB_RELEASE_TAG"):
    %cd {folder_path}
```

```
###### Install required Python packages ######
Requirement already satisfied: pandas==2.2.2 in /usr/local/lib/python3.11/dist-
packages (from -r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 1)) (2.2.2)
Requirement already satisfied: scikit-learn==1.6.1 in
/usr/local/lib/python3.11/dist-packages (from -r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 2)) (1.6.1)
Requirement already satisfied: altair==5.5.0 in /usr/local/lib/python3.11/dist-
packages (from -r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (5.5.0)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-
packages (from pandas==2.2.2->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 1)) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas==2.2.2->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 1)) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas==2.2.2->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 1)) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas==2.2.2->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 1)) (2025.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn==1.6.1->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 2)) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn==1.6.1->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 2)) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn==1.6.1->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 2)) (3.6.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages
(from altair==5.5.0->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (3.1.6)
Requirement already satisfied: jsonschema>=3.0 in
/usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (4.23.0)
```

```
Requirement already satisfied: narwhals>=1.14.2 in
/usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (1.31.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from altair==5.5.0->-r https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (24.2)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (4.12.2)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-
packages (from jsonschema>=3.0->altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-
packages (from jsonschema>=3.0->altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (0.23.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas==2.2.2->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 1)) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->altair==5.5.0->-r
https://raw.githubusercontent.com/aso-
uts/labs_datasets/main/36106-mlaa/requirements.txt (line 3)) (3.0.2)

###### Connect to personal Google Drive ######
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).

###### Setting up folders ######

You can now save your data files in:
/content/gdrive/MyDrive/36106/assignment/AT1/data
/content/gdrive/MyDrive/36106/assignment/AT1/data
```

1.1.2 0.b Disable Warnings Messages

Do not modify this code before running it

[ ]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

1.1.3 0.c Install Additional Packages

If you are using additional packages, you need to install them here using the command:
`! pip install <package_name>`

[1]:
```
# <Student to fill this section>
!apt-get update > /dev/null 2>&1
!apt-get install -y texlive texlive-xetex texlive-latex-extra pandoc > /dev/
 ↪null 2>&1
```

1.1.4 0.d Import Packages

[2]:
```
import ipywidgets as widgets
import pandas as pd
import numpy as np
import altair as alt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

1.2 A. Project Description

[3]:
```
# @title Student Information
wgt_student_name = widgets.Text(
    value="Fatemeh Elyasifar",
    placeholder='<student to fill this section>',
    description='Student Name:',
    style={'description_width': 'initial'},
    disabled=False
)

wgt_student_id = widgets.Text(
    value="25589351",
    placeholder='<student to fill this section>',
    description='Student Id:',
    style={'description_width': 'initial'},
    disabled=False
)
```

```
widgets.HBox([wgt_student_name, wgt_student_id])
```

```
HBox(children=(Text(value='Fatemeh Elyasifar', description='Student Name:',␣
 ↪placeholder='<student to fill this…
```

[4]:
```
print("Student Name:", wgt_student_name.value)
print("Student Id:", wgt_student_id.value)
```

```
Student Name: Fatemeh Elyasifar
Student Id: 25589351
```

[5]:
```
# @title Experiment ID

wgt_experiment_id = widgets.BoundedIntText(
    value="0",
    min=0,
    max=3,
    step=1,
    description='Experiment ID:',
    style={'description_width': 'initial'},
    disabled=False
)
wgt_experiment_id
```

```
BoundedIntText(value=0, description='Experiment ID:', max=3,␣
 ↪style=DescriptionStyle(description_width='initial…
```

[6]:
```
print("Experiment ID:", wgt_experiment_id.value)
```

```
Experiment ID: 0
```

[7]:
```
# @title Business Objective

wgt_business_objective = widgets.Textarea(
    value="The main objective is to develop a machine learning model that␣
 ↪accurately predicts rental prices specifically for affordable properties in␣
 ↪Australia, excluding luxury homes. This model aims to assist real estate␣
 ↪agencies, property investors, and tenants in making informed decisions based␣
 ↪on the features of affordable housing and market trends. The key success␣
 ↪metric is RMSE, with the goal of achieving an RMSE score of less than 16 on␣
 ↪the validation set, which quantifies the prediction error. Additionally,␣
 ↪feature tuning will be necessary to optimise the model's performance and␣
 ↪ensure accurate predictions based on the most relevant property␣
 ↪characteristics.",
    placeholder='<student to fill this section>',
    description='Business Objective:',
    disabled=False,
```

```
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_business_objective
```

```
Textarea(value='The main objective is to develop a machine learning model that␣
 ↪accurately predicts rental pric…
```

[8]:
```
print("Business Objective:", wgt_business_objective.value)
```

```
Business Objective: The main objective is to develop a machine learning model
that accurately predicts rental prices specifically for affordable properties in
Australia, excluding luxury homes. This model aims to assist real estate
agencies, property investors, and tenants in making informed decisions based on
the features of affordable housing and market trends. The key success metric is
RMSE, with the goal of achieving an RMSE score of less than 16 on the validation
set, which quantifies the prediction error. Additionally, feature tuning will be
necessary to optimise the model's performance and ensure accurate predictions
based on the most relevant property characteristics.
```

1.3 B. Experiment Description

[10]:
```
# @title Experiment Hypothesis

wgt_experiment_hypothesis = widgets.Textarea(
    value="Rental prices can be accurately predicted using a regression model␣
 ↪based on key property features such as location, size, number of bedrooms,␣
 ↪and proximity to amenities. Additionally, location is expected to be the␣
 ↪most influential factor in determining rental price variations. H (Null␣
 ↪Hypothesis): Property attributes such as the number of bedrooms, floor area,␣
 ↪location (suburb), and furnishing status have no significant impact on␣
 ↪rental prices.",
    placeholder='<student to fill this section>',
    description='Experiment Hypothesis:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_experiment_hypothesis
```

```
Textarea(value='Rental prices can be accurately predicted using a regression␣
 ↪model based on key property featu…
```

[11]:
```
print("Experiment Hypothesis:", wgt_experiment_hypothesis.value)
```

```
Experiment Hypothesis: Rental prices can be accurately predicted using a
```

regression model based on key property features such as location, size, number of bedrooms, and proximity to amenities. Additionally, location is expected to be the most influential factor in determining rental price variations. H (Null Hypothesis): Property attributes such as the number of bedrooms, floor area, location (suburb), and furnishing status have no significant impact on rental prices.

[13]:
```
# @title Experiment Expectations

wgt_experiment_expectations = widgets.Textarea(
    value="Identify missing values and outliers among features. Conduct␣
 ↪necessary data transformations and engineering to improve data quality.␣
 ↪Train and evaluate a baseline model using DummyRegressor as a simple␣
 ↪benchmark. Establish a baseline RMSE score that will serve as a reference␣
 ↪for subsequent experiments.",
    placeholder='<student to fill this section>',
    description='Experiment Expectations:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_experiment_expectations
```

```
Textarea(value='Identify missing values and outliers among features. Conduct␣
 ↪necessary data transformations an…
```

[14]:
```
print("Experiment Expectations:", wgt_experiment_expectations.value)
```

```
Experiment Expectations: Identify missing values and outliers among features.
Conduct necessary data transformations and engineering to improve data quality.
Train and evaluate a baseline model using DummyRegressor as a simple benchmark.
Establish a baseline RMSE score that will serve as a reference for subsequent
experiments.
```

1.4 C. Data Understanding

1.4.1 C.1 Load Datasets

Do not change this code

[ ]:
```
# Load training data
training_df = pd.read_csv(folder_path / "rental_training.csv")
```

[ ]:
```
# Load validation data
validation_df = pd.read_csv(folder_path / "rental_validation.csv")
```

[ ]:
```
# Load testing data
testing_df = pd.read_csv(folder_path / "rental_testing.csv")
```

1.4.2 C.2 Explore Training Set

You can add more cells in this section

[ ]:
```
# <Student to fill this section>
training_df.head()
```

[ ]:

	advertised_date	number_of_bedrooms	rent	floor_area	level	\
0	2022-05-18	2	568.0	1100	Ground out of 2	
1	2022-05-13	2	581.0	800	1 out of 3	
2	2022-05-16	2	577.0	1000	1 out of 3	
3	2022-05-09	2	565.0	850	1 out of 2	
4	2022-04-29	2	564.0	600	Ground out of 1	

	suburb	furnished	tenancy_preference	number_of_bathrooms	\
0	Canberra	Unfurnished	Bachelors/Family	2	
1	Canberra	Semi-Furnished	Bachelors/Family	1	
2	Canberra	Semi-Furnished	Bachelors/Family	1	
3	Canberra	Unfurnished	Bachelors	1	
4	Canberra	Unfurnished	Bachelors/Family	2	

	point_of_contact	secondary_address	building_number	street_name	\
0	Contact Owner	02/	1	Mcdowell Edge	
1	Contact Owner	667/	6	Lewis Parkway	
2	Contact Owner	859/	459	Daniel Copse	
3	Contact Owner	Flat 54	482	Young Walkway	
4	Contact Owner	Unit 75	838	Michael Port	

	street_suffix	prefix	first_name	last_name	gender	phone_number	\
0	Driveway	Mr.	Robert	Jones	m	(08) 8174 5701	
1	Viaduct	Mrs.	Lisa	Mcknight	f	(08).5553.7944	
2	Meander	NaN	Annette	Lester	u	(03).6394.3934	
3	Firetrail	Mrs.	Emma	Hill	f	+61836311377	
4	Esplanade	Miss	Ariana	Richardson	f	+61 409 341 340	

	email
0	georgelopez@example.org
1	robertdorsey@example.net
2	rodriguezkaren@example.net
3	johnsonjeremy@example.com
4	sbrown@example.net

[ ]:
```
training_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 3434 entries, 0 to 3433
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   advertised_date       3434 non-null   object
 1   number_of_bedrooms    3434 non-null   int64
 2   rent                  3434 non-null   float64
 3   floor_area            3434 non-null   int64
 4   level                 3434 non-null   object
 5   suburb                3434 non-null   object
 6   furnished             3434 non-null   object
 7   tenancy_preference    3434 non-null   object
 8   number_of_bathrooms   3434 non-null   int64
 9   point_of_contact      3434 non-null   object
 10  secondary_address     3434 non-null   object
 11  building_number       3434 non-null   int64
 12  street_name           3434 non-null   object
 13  street_suffix         3434 non-null   object
 14  prefix                2274 non-null   object
 15  first_name            3434 non-null   object
 16  last_name             3433 non-null   object
 17  gender                3434 non-null   object
 18  phone_number          3434 non-null   object
 19  email                 3434 non-null   object
dtypes: float64(1), int64(4), object(15)
memory usage: 536.7+ KB
```

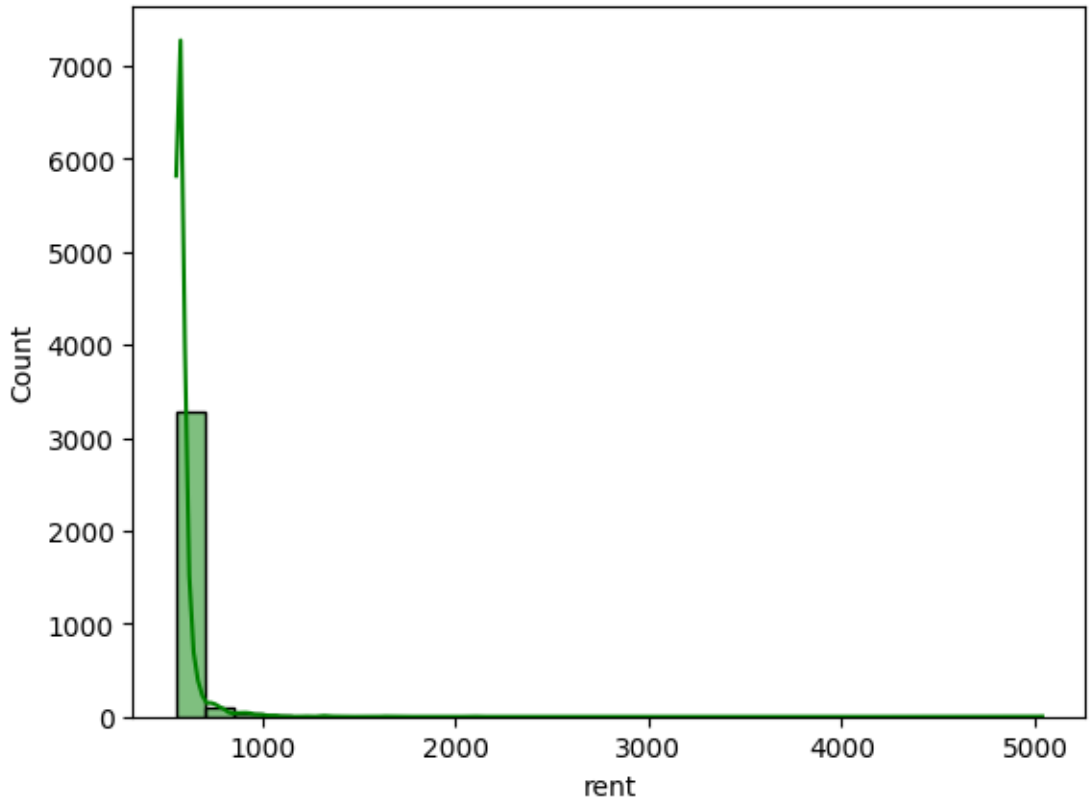
[ ]:
```
training_df.describe()
```

[ ]:
```
       number_of_bedrooms         rent   floor_area  number_of_bathrooms  \
count         3434.000000  3434.000000  3434.000000          3434.000000
mean             2.022423   595.080664   919.708794             1.881188
std              0.813388   105.380805   588.741127             0.850203
min              1.000000   557.000000    20.000000             1.000000
25%              1.000000   567.000000   550.000000             1.000000
50%              2.000000   574.000000   800.000000             2.000000
75%              2.000000   590.000000  1186.000000             2.000000
max              6.000000  5037.000000  8000.000000            10.000000

       building_number
count      3434.000000
mean        189.853815
std         284.860733
min           0.000000
25%           7.000000
50%          46.000000
75%         268.750000
```

```
max          998.000000
```

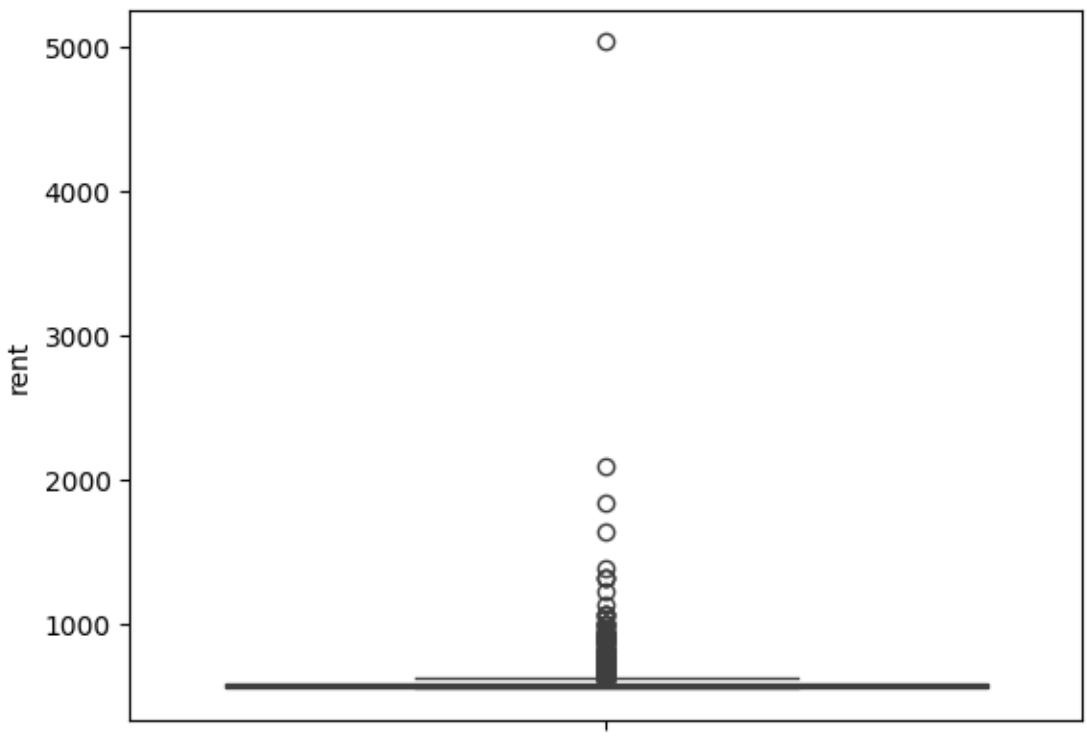
[ ]:
```
sns.histplot(training_df['rent'], bins=30, kde=True, color="green")
```

[ ]:
```
<Axes: xlabel='rent', ylabel='Count'>
```



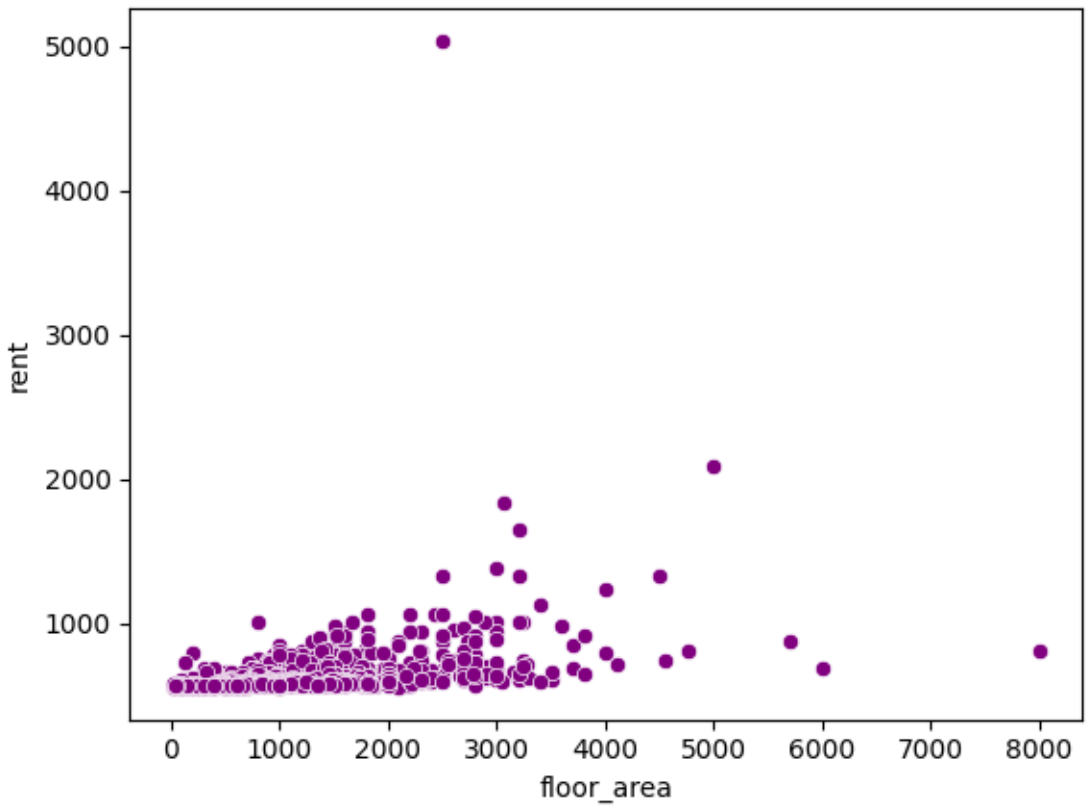
[ ]:
```
sns.boxplot(y=training_df['rent'])
```

[ ]:
```
<Axes: ylabel='rent'>
```



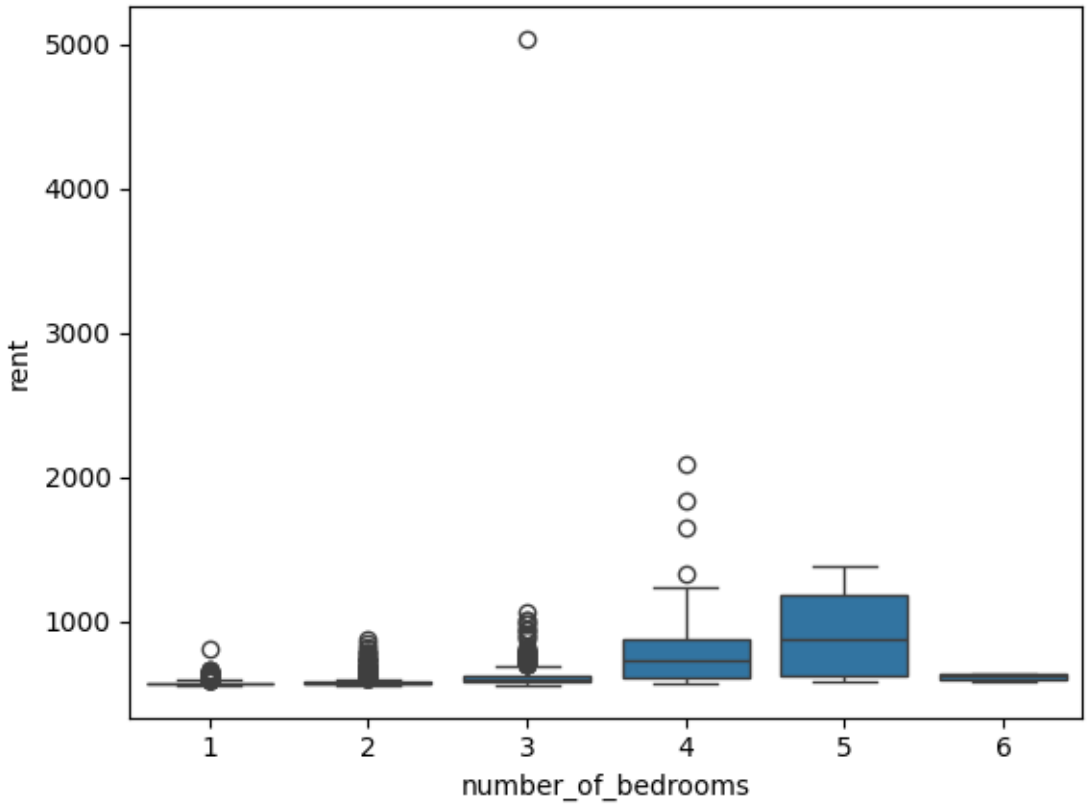
[ ]:
```
sns.scatterplot(x=training_df['floor_area'], y=training_df['rent'],␣
↪color="purple")
```

[ ]:
```
<Axes: xlabel='floor_area', ylabel='rent'>
```



[ ]:
```
sns.boxplot(x=training_df['number_of_bedrooms'], y=training_df['rent'])
```

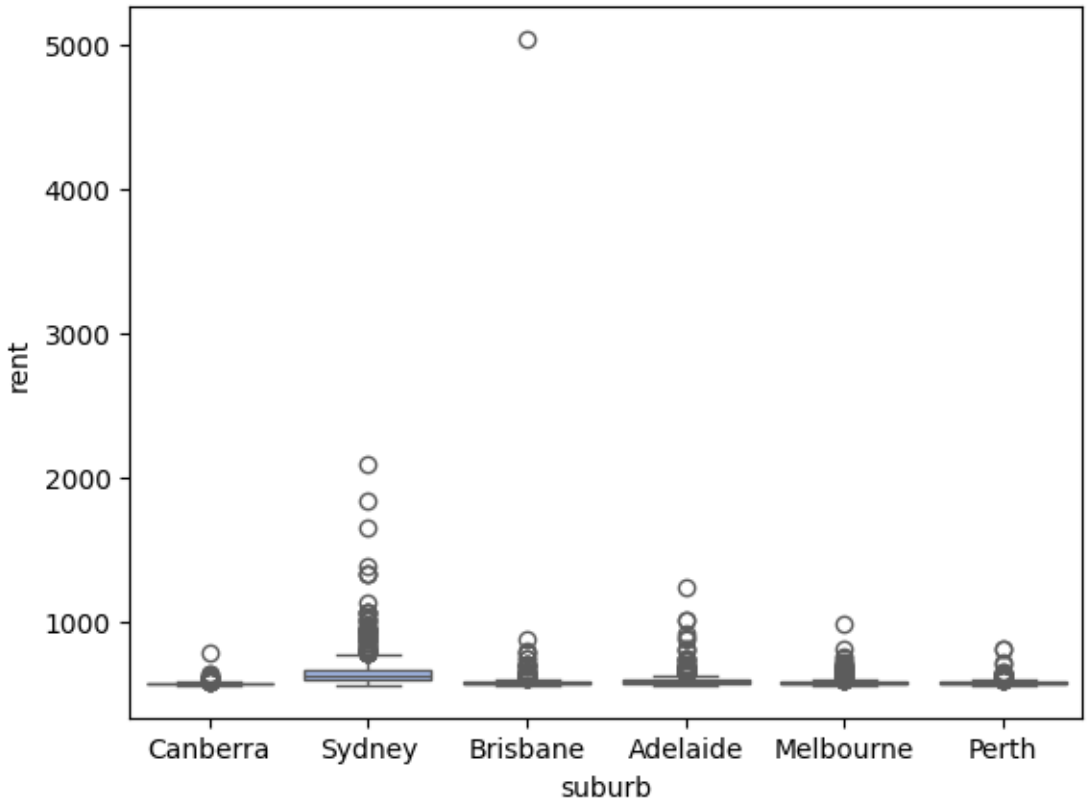
[ ]:
```
<Axes: xlabel='number_of_bedrooms', ylabel='rent'>
```



[ ]:
```
sns.boxplot(x=training_df['suburb'], y=training_df['rent'], palette="coolwarm")
```

```
<ipython-input-13-c6f3a7ebee97>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=training_df['suburb'], y=training_df['rent'],
palette="coolwarm")
```

[ ]:
```
<Axes: xlabel='suburb', ylabel='rent'>
```



[15]:
```
# @title Training Set Insights

wgt_eda_training_set_insights = widgets.Textarea(
    value="The dataset contains 20 features, including rental prices. A few␣
 ↪missing values are present in some columns, but no missing values are␣
 ↪detected in numerical columns. The rental costs are likely segmented into␣
 ↪ranges, showing their distribution. The highest recorded rent is 5,037,␣
 ↪while the minimum is 557. There is a relationship between floor area and␣
 ↪rent, suggesting that larger properties tend to have higher rental prices.␣
 ↪Sydney appears to have greater variation in rental prices, with houses␣
 ↪having four or five bedrooms generally commanding higher rents, while those␣
 ↪with one to three bedrooms tend to be more affordable.",
    placeholder='<student to fill this section>',
    description='Training Set Insights:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_eda_training_set_insights
```

```
Textarea(value='The dataset contains 20 features, including rental prices. A few␣
↪missing values are present in…
```

[16]:
```
print("Training Set Insights:", wgt_eda_training_set_insights.value)
```

```
Training Set Insights: The dataset contains 20 features, including rental
prices. A few missing values are present in some columns, but no missing values
are detected in numerical columns. The rental costs are likely segmented into
ranges, showing their distribution. The highest recorded rent is 5,037, while
the minimum is 557. There is a relationship between floor area and rent,
suggesting that larger properties tend to have higher rental prices. Sydney
appears to have greater variation in rental prices, with houses having four or
five bedrooms generally commanding higher rents, while those with one to three
bedrooms tend to be more affordable.
```

1.4.3 C.3 Explore Validation Set

You can add more cells in this section

[ ]:
```
# <Student to fill this section>
validation_df.head()
```

[ ]:
```
  advertised_date  number_of_bedrooms   rent  floor_area  \
0      2022-06-13                   2  571.0         560
1      2022-06-04                   2  683.0         750
2      2022-04-29                   3  574.0         950
3      2022-05-18                   1  565.0         500
4      2022-04-28                   2  565.0         600

                      level     suburb       furnished tenancy_preference  \
0           Ground out of 1  Melbourne  Semi-Furnished             Family
1  Upper Basement out of 30     Sydney    Unfurnished   Bachelors/Family
2           Ground out of 3   Adelaide    Unfurnished   Bachelors/Family
3                2 out of 2     Sydney  Semi-Furnished          Bachelors
4                2 out of 3   Brisbane  Semi-Furnished   Bachelors/Family

   number_of_bathrooms point_of_contact secondary_address  building_number  \
0                    2    Contact Owner          Level 1                1
1                    2    Contact Agent               1/               31
2                    2    Contact Owner          Unit 37               89
3                    1    Contact Owner               16/              82
4                    2    Contact Owner          Flat 64                9

       street_name street_suffix prefix first_name last_name gender  \
0   Baldwin Towers       Footway    NaN        Jay    Glover      u
1   Cox Fire Track       Lookout    Dr.   Danielle      Tran      f
2  Davidson Ground          Part    NaN     Ashley   Pacheco      u
3  Fitzpatrick Key       Heights    NaN   Victoire     Weber      u
```

```
4     Heidi Access          Mews   Mrs.      Kerry      Koch      f

     phone_number                      email
0    (03)08687820   brettkennedy@example.net
1  (03)-0313-6072         dana35@example.net
2    08-9358-6662       justin89@example.org
3  (02).9817.8199  pruittmichael@example.net
4       4124.0210    hansendiana@example.com
```

[ ]:
```
validation_df.info()
```

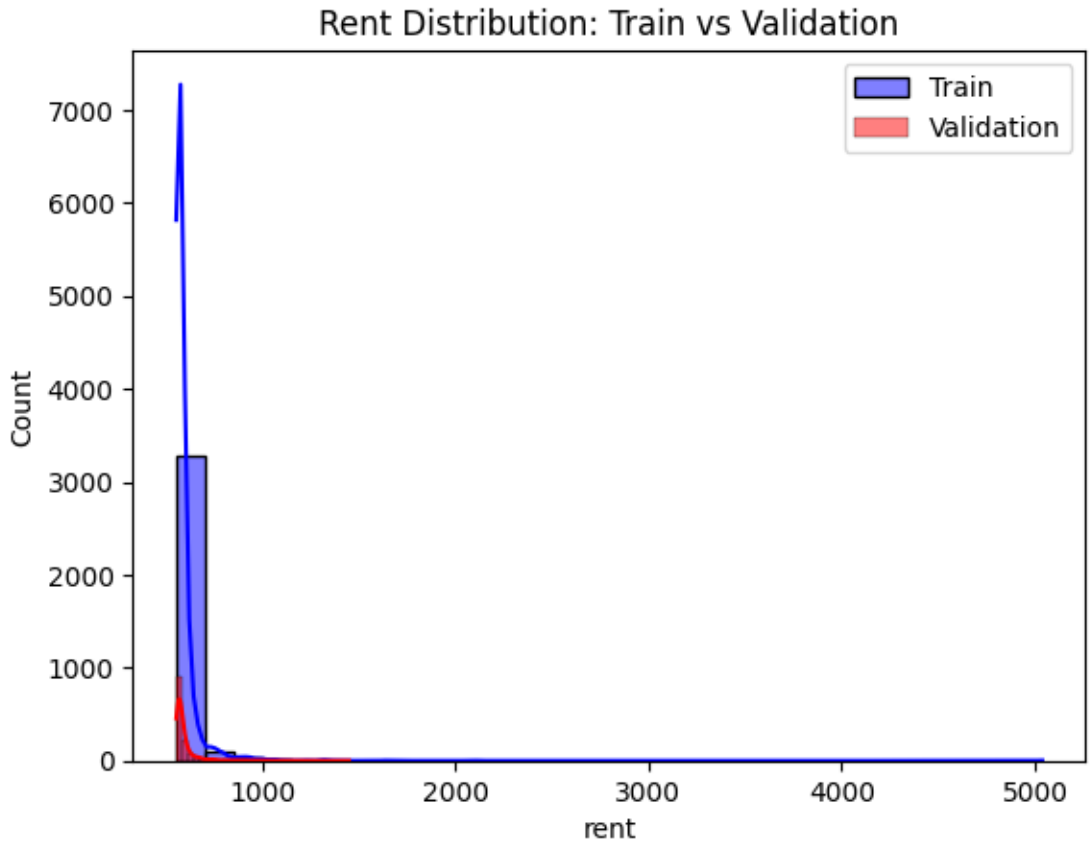
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1320 entries, 0 to 1319
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   advertised_date      1320 non-null   object
 1   number_of_bedrooms   1320 non-null   int64
 2   rent                 1320 non-null   float64
 3   floor_area           1320 non-null   int64
 4   level                1320 non-null   object
 5   suburb               1320 non-null   object
 6   furnished            1320 non-null   object
 7   tenancy_preference   1320 non-null   object
 8   number_of_bathrooms  1320 non-null   int64
 9   point_of_contact     1320 non-null   object
 10  secondary_address    1320 non-null   object
 11  building_number      1320 non-null   int64
 12  street_name          1320 non-null   object
 13  street_suffix        1320 non-null   object
 14  prefix               855 non-null    object
 15  first_name           1320 non-null   object
 16  last_name            1319 non-null   object
 17  gender               1320 non-null   object
 18  phone_number         1320 non-null   object
 19  email                1320 non-null   object
dtypes: float64(1), int64(4), object(15)
memory usage: 206.4+ KB
```

[ ]:
```
validation_df.describe()
```

[ ]:
```
       number_of_bedrooms         rent   floor_area  number_of_bathrooms  \
count         1320.000000  1320.000000  1320.000000          1320.000000
mean             2.091667   596.413636   959.723485             1.946212
std              0.819543    67.974174   645.170039             0.879891
min              1.000000   557.000000    10.000000             1.000000
25%              2.000000   568.000000   550.000000             1.000000
```

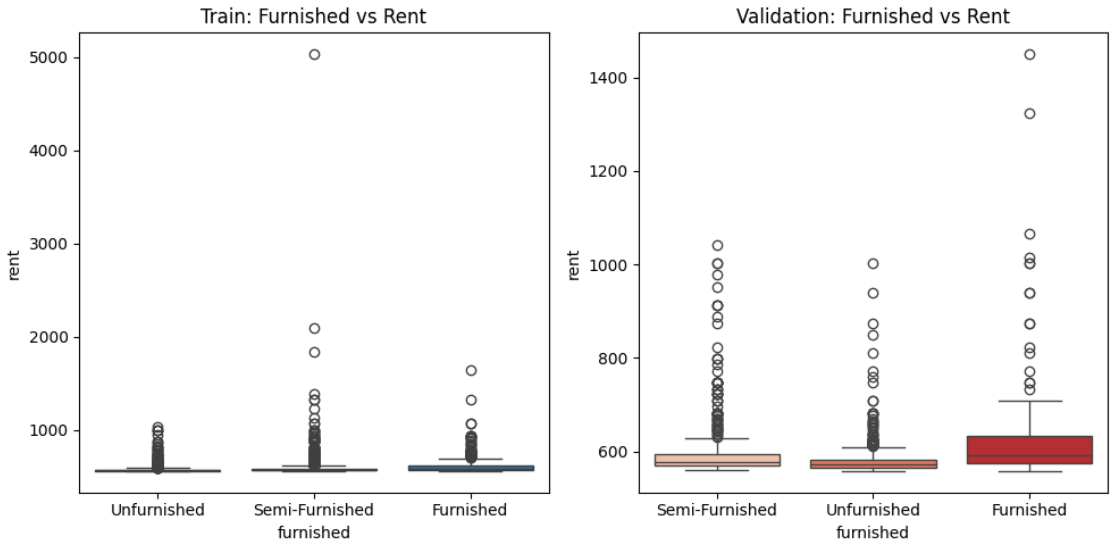
```
50%              2.000000   574.000000   825.500000             2.000000
75%              3.000000   594.000000  1200.000000             2.000000
max              6.000000  1451.000000  6000.000000             7.000000

       building_number
count      1320.000000
mean        189.813636
std         283.228988
min           0.000000
25%           6.000000
50%          49.000000
75%         270.000000
max         996.000000
```

[ ]:
```
sns.histplot(training_df['rent'], bins=30, kde=True, color="blue",␣
 ↪label="Train", alpha=0.5)
sns.histplot(validation_df['rent'], bins=30, kde=True, color="red",␣
 ↪label="Validation", alpha=0.5)
plt.legend()
plt.title("Rent Distribution: Train vs Validation")
plt.show()
```

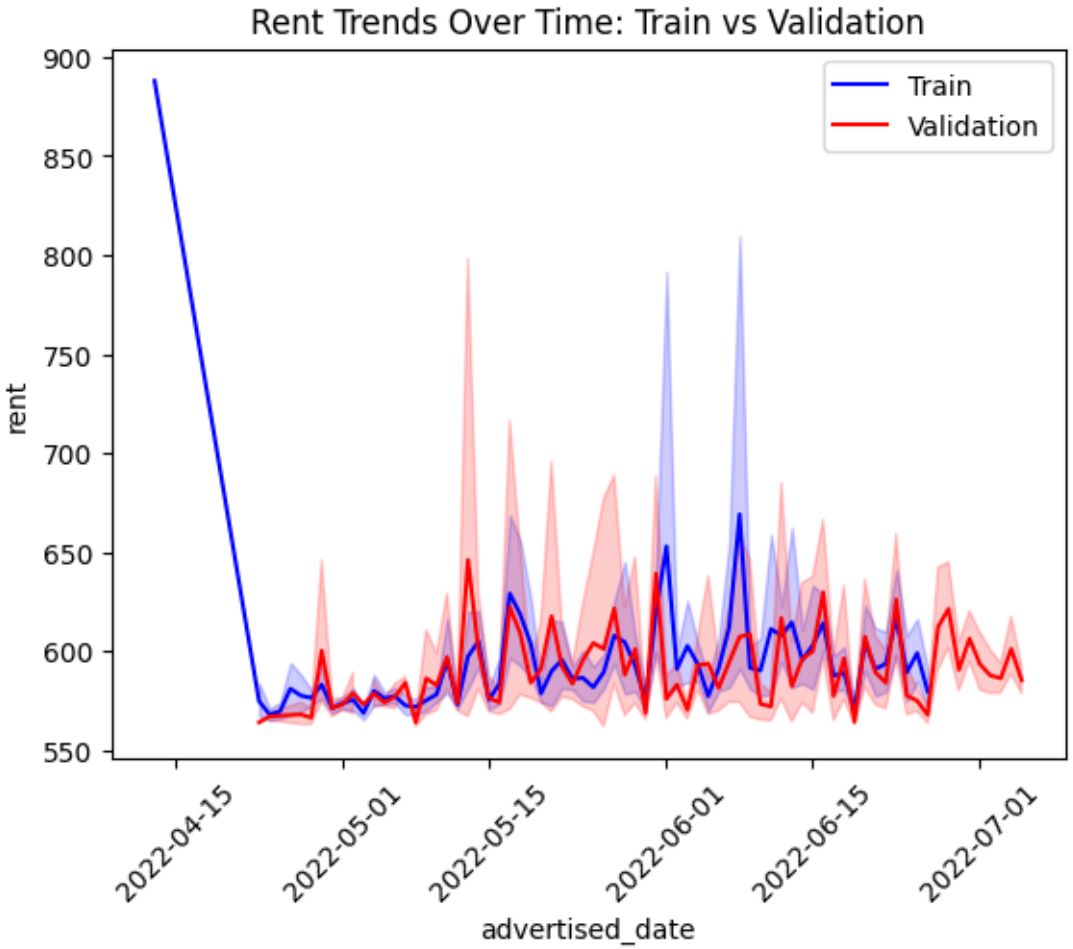


[ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

sns.boxplot(x=training_df['furnished'], y=training_df['rent'], palette="Blues",␣
 ↪ax=axes[0])
axes[0].set_title("Train: Furnished vs Rent")

sns.boxplot(x=validation_df['furnished'], y=validation_df['rent'],␣
 ↪palette="Reds", ax=axes[1])
axes[1].set_title("Validation: Furnished vs Rent")

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

```
<ipython-input-18-4b842f1a6dfb>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=training_df['furnished'], y=training_df['rent'],
palette="Blues", ax=axes[0])
<ipython-input-18-4b842f1a6dfb>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x=validation_df['furnished'], y=validation_df['rent'],
palette="Reds", ax=axes[1])
```



[ ]:
```
training_df['advertised_date'] = pd.to_datetime(training_df['advertised_date'])
validation_df['advertised_date'] = pd.
 ↪to_datetime(validation_df['advertised_date'])

sns.lineplot(x=training_df['advertised_date'], y=training_df['rent'],␣
 ↪label="Train", color="blue")
sns.lineplot(x=validation_df['advertised_date'], y=validation_df['rent'],␣
 ↪label="Validation", color="red")
plt.title("Rent Trends Over Time: Train vs Validation")
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



[ ]:
```
# @title Validation Set Insights

wgt_eda_validation_set_insights = widgets.Textarea(
    value=None,
    placeholder='<student to fill this section>',
    description='Validation Set Insights:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_eda_validation_set_insights
```

```
Textarea(value='', description='Validation Set Insights:',␣
 ↪layout=Layout(height='100%', width='auto'), placeho…
```

1.4.4 C.4 Explore Testing Set

You can add more cells in this section

[ ]:
```
# <Student to fill this section>
testing_df.head()
```

[ ]:

	advertised_date	number_of_bedrooms	rent	floor_area	level	suburb	furnished	tenancy_preference	number_of_bathrooms	point_of_contact	secondary_address	building_number	street_name	street_suffix	prefix	first_name	last_name	gender
0	2022-06-13	2	571.0	560	Ground out of 1	Melbourne	Semi-Furnished	Family	2	Contact Owner	Level 1	1	Baldwin Towers	Footway	NaN	Jay	Glover	u
1	2022-06-04	2	683.0	750	Upper Basement out of 30	Sydney	Unfurnished	Bachelors/Family	2	Contact Agent	1/	31	Cox Fire Track	Lookout	Dr.	Danielle	Tran	f
2	2022-04-29	3	574.0	950	Ground out of 3	Adelaide	Unfurnished	Bachelors/Family	2	Contact Owner	Unit 37	89	Davidson Ground	Part	NaN	Ashley	Pacheco	u
3	2022-05-18	1	565.0	500	2 out of 2	Sydney	Semi-Furnished	Bachelors	1	Contact Owner	16/	82	Fitzpatrick Key	Heights	NaN	Victoire	Weber	u
4	2022-04-28	2	565.0	600	2 out of 3	Brisbane	Semi-Furnished	Bachelors/Family	2	Contact Owner	Flat 64	9						

```
4     Heidi Access          Mews   Mrs.      Kerry      Koch      f

     phone_number                      email
0    (03)08687820   brettkennedy@example.net
1  (03)-0313-6072         dana35@example.net
2    08-9358-6662       justin89@example.org
3  (02).9817.8199  pruittmichael@example.net
4       4124.0210    hansendiana@example.com
```

[ ]:
```
testing_df.info()
```

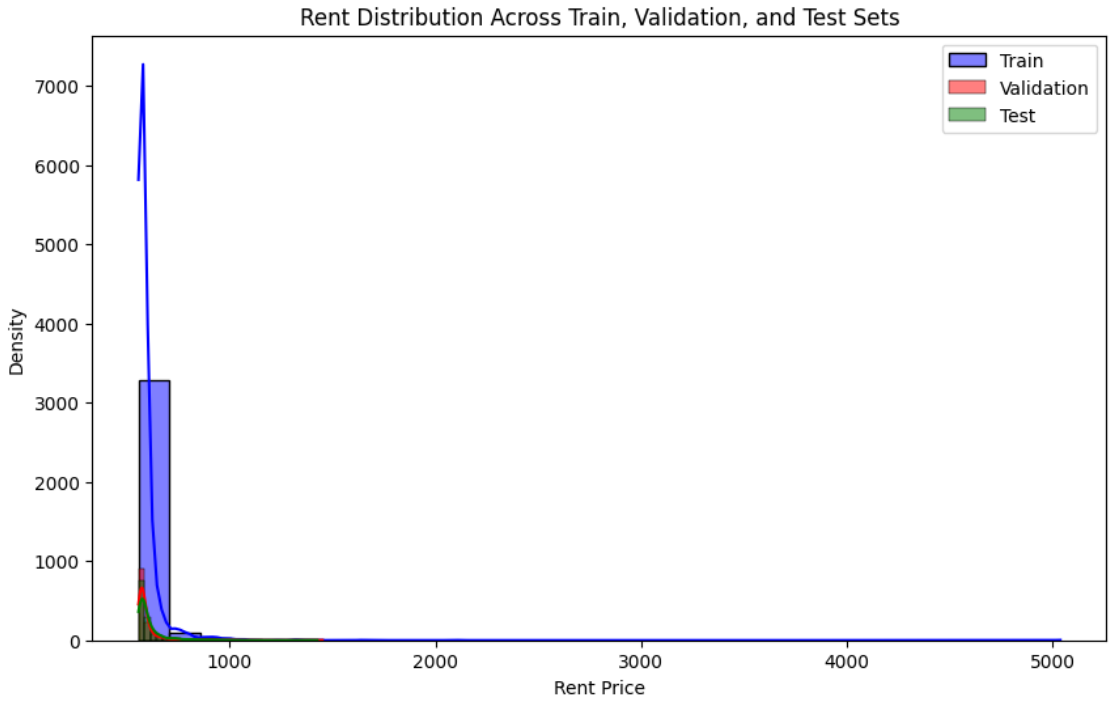
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1364 entries, 0 to 1363
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   advertised_date      1364 non-null   object
 1   number_of_bedrooms   1364 non-null   int64
 2   rent                 1364 non-null   float64
 3   floor_area           1364 non-null   int64
 4   level                1364 non-null   object
 5   suburb               1364 non-null   object
 6   furnished            1364 non-null   object
 7   tenancy_preference   1364 non-null   object
 8   number_of_bathrooms  1364 non-null   int64
 9   point_of_contact     1364 non-null   object
 10  secondary_address    1364 non-null   object
 11  building_number      1364 non-null   int64
 12  street_name          1364 non-null   object
 13  street_suffix        1364 non-null   object
 14  prefix               877 non-null    object
 15  first_name           1364 non-null   object
 16  last_name            1364 non-null   object
 17  gender               1364 non-null   object
 18  phone_number         1364 non-null   object
 19  email                1364 non-null   object
dtypes: float64(1), int64(4), object(15)
memory usage: 213.3+ KB
```

[ ]:
```
testing_df.describe()
```

[ ]:
```
       number_of_bedrooms         rent   floor_area  number_of_bathrooms  \
count         1364.000000  1364.000000  1364.000000          1364.000000
mean             2.184751   609.290323  1054.319648             2.098974
std              0.845966    79.660648   691.094588             0.928729
min              1.000000   557.000000    25.000000             1.000000
25%              2.000000   571.000000   600.000000             1.000000
```

```
50%              2.000000   581.000000   900.000000             2.000000
75%              3.000000   613.000000  1300.000000             3.000000
max              6.000000  1426.000000  7000.000000             7.000000

       building_number
count      1364.000000
mean        194.462610
std         290.294334
min           0.000000
25%           6.000000
50%          46.000000
75%         274.500000
max         996.000000
```

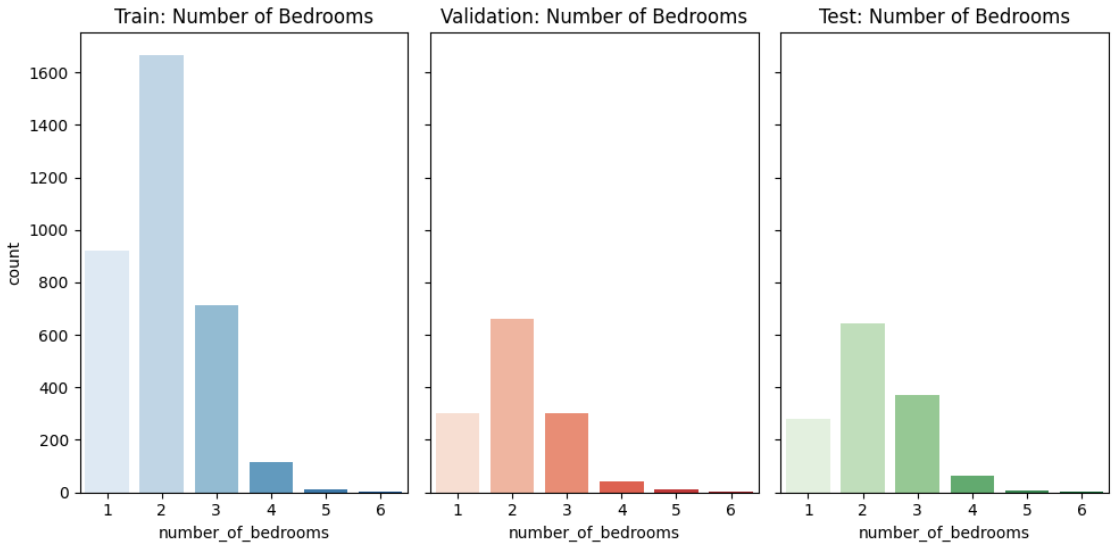
[ ]:
```
plt.figure(figsize=(10, 6))

sns.histplot(training_df['rent'], bins=30, kde=True, color="blue",␣
 ↪label="Train", alpha=0.5)
sns.histplot(validation_df['rent'], bins=30, kde=True, color="red",␣
 ↪label="Validation", alpha=0.5)
sns.histplot(testing_df['rent'], bins=30, kde=True, color="green",␣
 ↪label="Test", alpha=0.5)

plt.legend()
plt.title("Rent Distribution Across Train, Validation, and Test Sets")
plt.xlabel("Rent Price")
plt.ylabel("Density")
plt.show()
```



[ ]:
```
fig, axes = plt.subplots(1, 3, figsize=(10, 5), sharey=True)

sns.countplot(x=training_df['number_of_bedrooms'], palette="Blues", ax=axes[0])
axes[0].set_title("Train: Number of Bedrooms")

sns.countplot(x=validation_df['number_of_bedrooms'], palette="Reds", ax=axes[1])
axes[1].set_title("Validation: Number of Bedrooms")

sns.countplot(x=testing_df['number_of_bedrooms'], palette="Greens", ax=axes[2])
axes[2].set_title("Test: Number of Bedrooms")

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

```
<ipython-input-24-c25ef976dd3d>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=training_df['number_of_bedrooms'], palette="Blues",
ax=axes[0])
<ipython-input-24-c25ef976dd3d>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=validation_df['number_of_bedrooms'], palette="Reds",
ax=axes[1])
<ipython-input-24-c25ef976dd3d>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=testing_df['number_of_bedrooms'], palette="Greens",
ax=axes[2])
```



[17]:
```
# @title Testing Set Insights

wgt_eda_testing_set_insights = widgets.Textarea(
    value="Similar to the training and validation sets, the testing dataset has␣
 ↪20 features with 1,364 rows, closely matching the validation set size.␣
 ↪Missing values are present in the Prefix column, but no missing values were␣
 ↪found in numerical columns. The Train dataset (blue) has the highest density␣
 ↪and dominates in sample size, while the Validation (red) and Test (green)␣
 ↪datasets follow similar distributions but contain fewer observations. All␣
 ↪datasets exhibit similar distribution patterns, indicating a well-aligned␣
 ↪data split. Additionally, the distribution of bedrooms is skewed toward␣
 ↪2-bedroom properties, which are the most common, followed by 3-bedroom and␣
 ↪1-bedroom properties. This imbalance could introduce bias, potentially␣
 ↪leading to poorer model performance for properties with more or fewer␣
 ↪bedrooms.",
```

```
    placeholder='<student to fill this section>',
    description='Testing Set Insights:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_eda_testing_set_insights
```

```
Textarea(value='Similar to the training and validation sets, the testing dataset␣
 ↪has 20 features with 1,364 ro…
```

[18]:
```
print("Testing Set Insights:", wgt_eda_testing_set_insights.value)
```

```
Testing Set Insights: Similar to the training and validation sets, the testing
dataset has 20 features with 1,364 rows, closely matching the validation set
size. Missing values are present in the Prefix column, but no missing values
were found in numerical columns. The Train dataset (blue) has the highest
density and dominates in sample size, while the Validation (red) and Test
(green) datasets follow similar distributions but contain fewer observations.
All datasets exhibit similar distribution patterns, indicating a well-aligned
data split. Additionally, the distribution of bedrooms is skewed toward
2-bedroom properties, which are the most common, followed by 3-bedroom and
1-bedroom properties. This imbalance could introduce bias, potentially leading
to poorer model performance for properties with more or fewer bedrooms.
```

1.4.5 C.5 Explore Target Variable

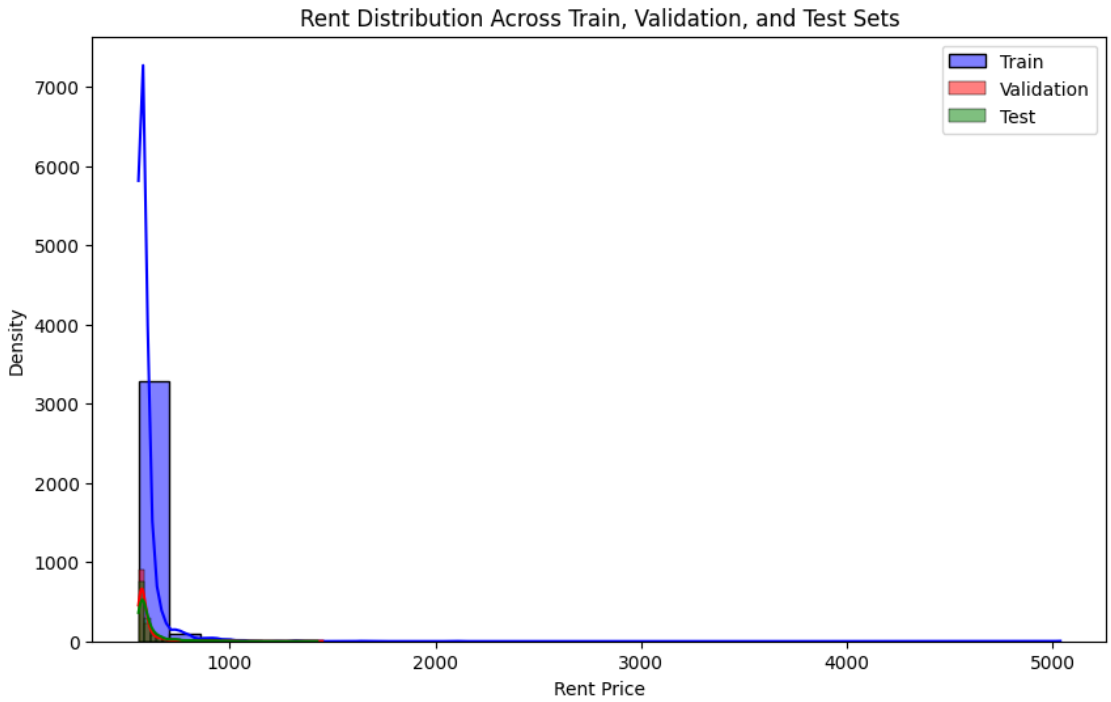
Save the name of column used as the target variable and call it `target_name`

You can add more cells in this section

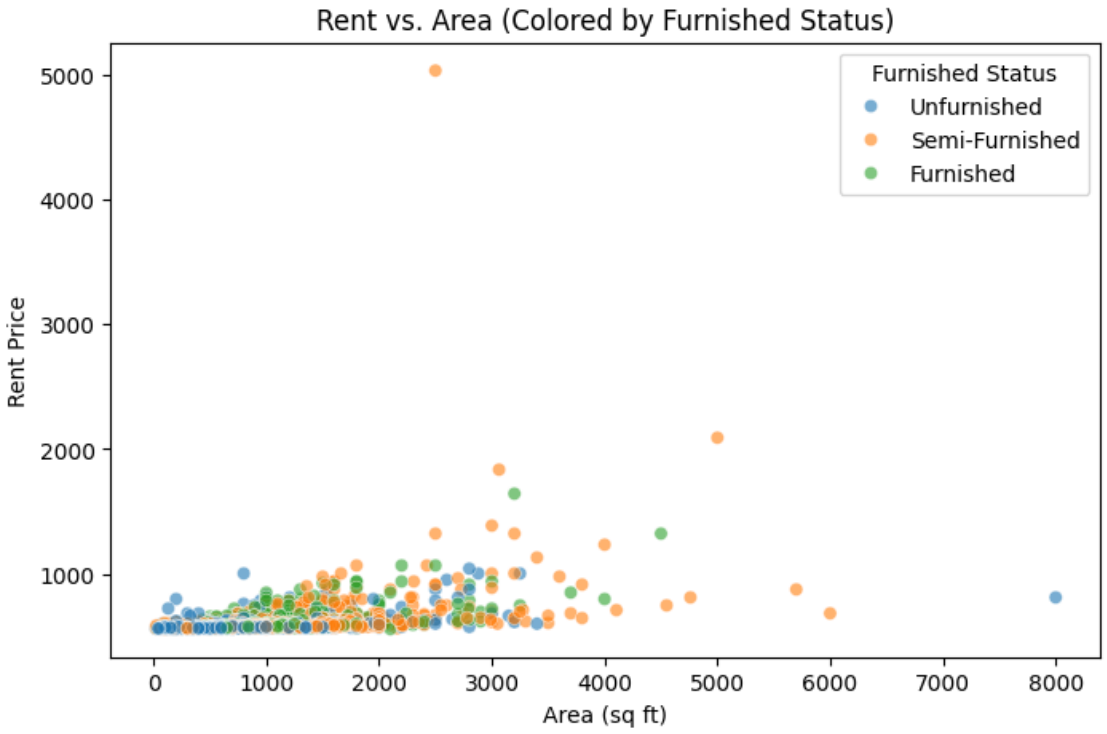
[ ]:
```
# <Student to fill this section>

target_name = 'rent'
```

[ ]:
```
plt.figure(figsize=(10, 6))

sns.histplot(training_df[target_name], bins=30, kde=True, color="blue",␣
 ↪label="Train", alpha=0.5)
sns.histplot(validation_df[target_name], bins=30, kde=True, color="red",␣
 ↪label="Validation", alpha=0.5)
sns.histplot(testing_df[target_name], bins=30, kde=True, color="green",␣
 ↪label="Test", alpha=0.5)

plt.legend()
plt.title("Rent Distribution Across Train, Validation, and Test Sets")
plt.xlabel("Rent Price")
plt.ylabel("Density")
```

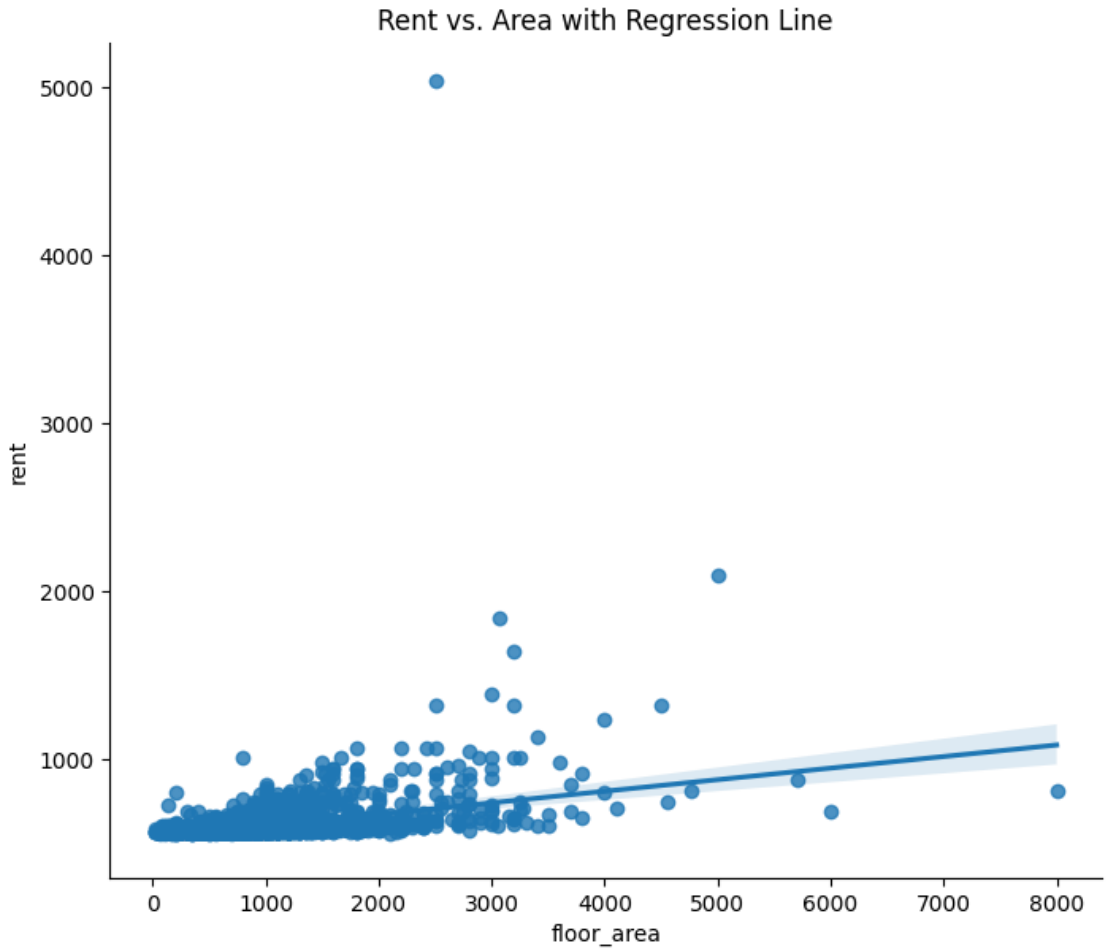
```
plt.show()
```



[ ]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x=training_df['floor_area'], y=training_df[target_name],␣
↪hue=training_df['furnished'], alpha=0.6)
plt.xlabel('Area (sq ft)')
plt.ylabel('Rent Price')
plt.title('Rent vs. Area (Colored by Furnished Status)')
plt.legend(title='Furnished Status')
plt.show()
```



[ ]:
```
sns.lmplot(x='floor_area', y=target_name, data=training_df, height=6, aspect=1.
↪2)
plt.title('Rent vs. Area with Regression Line')
plt.show()
```



[19]:
```
# @title Target Variable Insights

wgt_eda_target_variable_insights = widgets.Textarea(
    value="The target variable shows a highly right-skewed distribution in all␣
 ↪three datasets. The furnished status exhibits a positive correlation with␣
 ↪the target variable. Furnished houses tend to have more affordable rental␣
 ↪prices, while semi-furnished houses show greater price dispersion.␣
 ↪Additionally, there is a moderate positive relationship between floor area␣
 ↪and rental prices, with larger floor areas generally associated with higher␣
 ↪rental prices.",
    placeholder='<student to fill this section>',
    description='Target Variable Insights:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_eda_target_variable_insights
```

```
Textarea(value='The target variable shows a highly right-skewed distribution in␣
↪all three datasets. The furnis…
```

[20]:
```
print("Target Variable Insights:", wgt_eda_target_variable_insights.value)
```

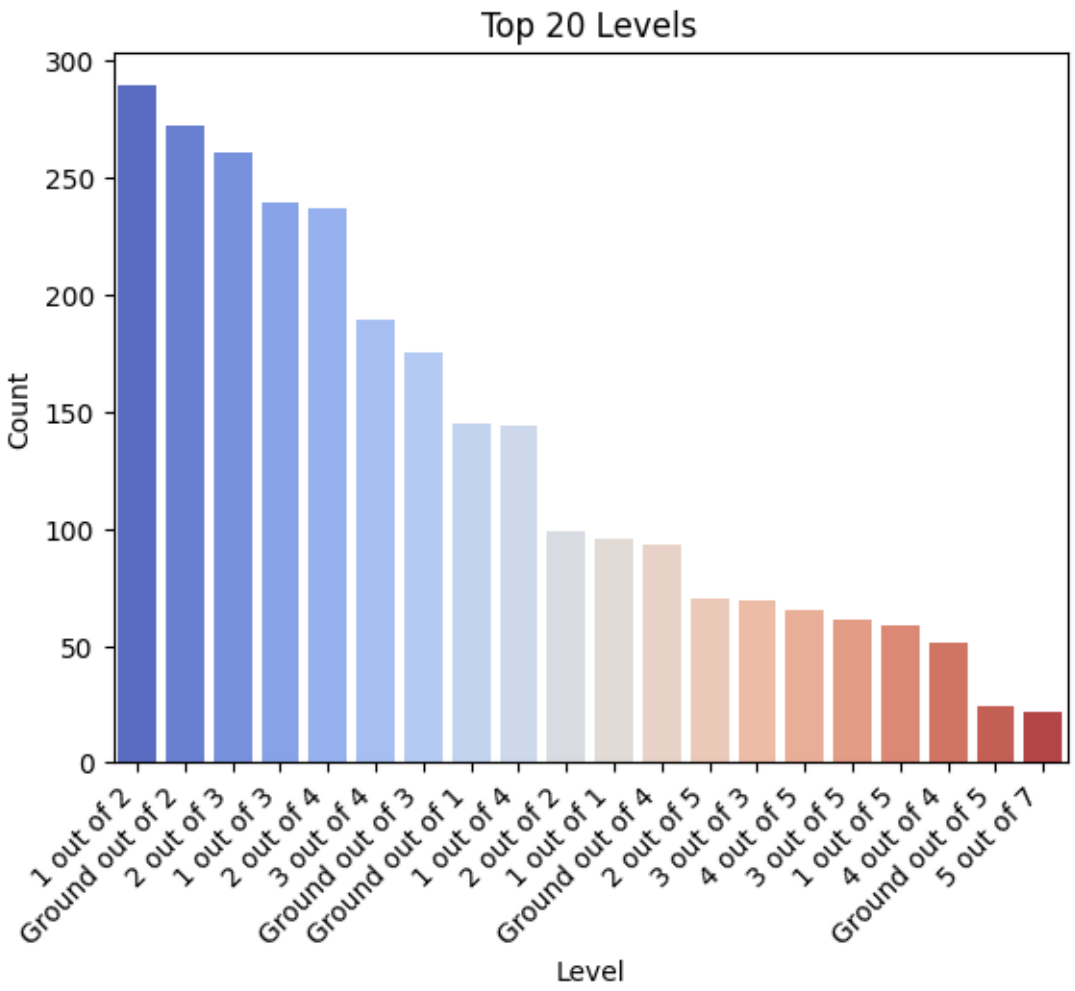
```
Target Variable Insights: The target variable shows a highly right-skewed
distribution in all three datasets. The furnished status exhibits a positive
correlation with the target variable. Furnished houses tend to have more
affordable rental prices, while semi-furnished houses show greater price
dispersion. Additionally, there is a moderate positive relationship between
floor area and rental prices, with larger floor areas generally associated with
higher rental prices.
```

1.4.6 C.6 Explore Feature of Interest

You can add more cells in this section

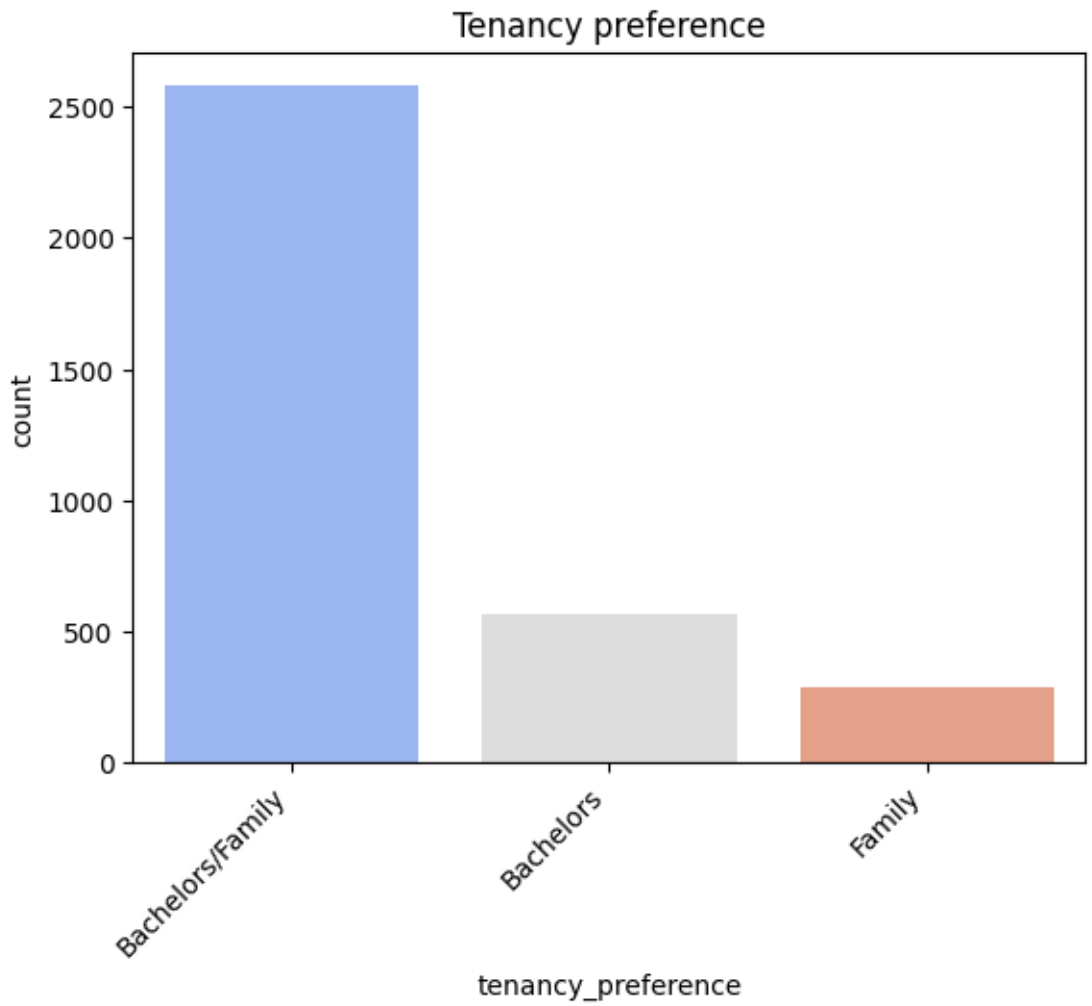
[ ]:
```
# <Student to fill this section>

top_levels = training_df['level'].value_counts().nlargest(20)
sns.barplot(x=top_levels.index, y=top_levels.values, palette="coolwarm")
plt.xlabel('Level')
plt.ylabel('Count')
plt.title('Top 20 Levels')
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
<ipython-input-29-7c8f6bf3a284>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_levels.index, y=top_levels.values, palette="coolwarm")
```



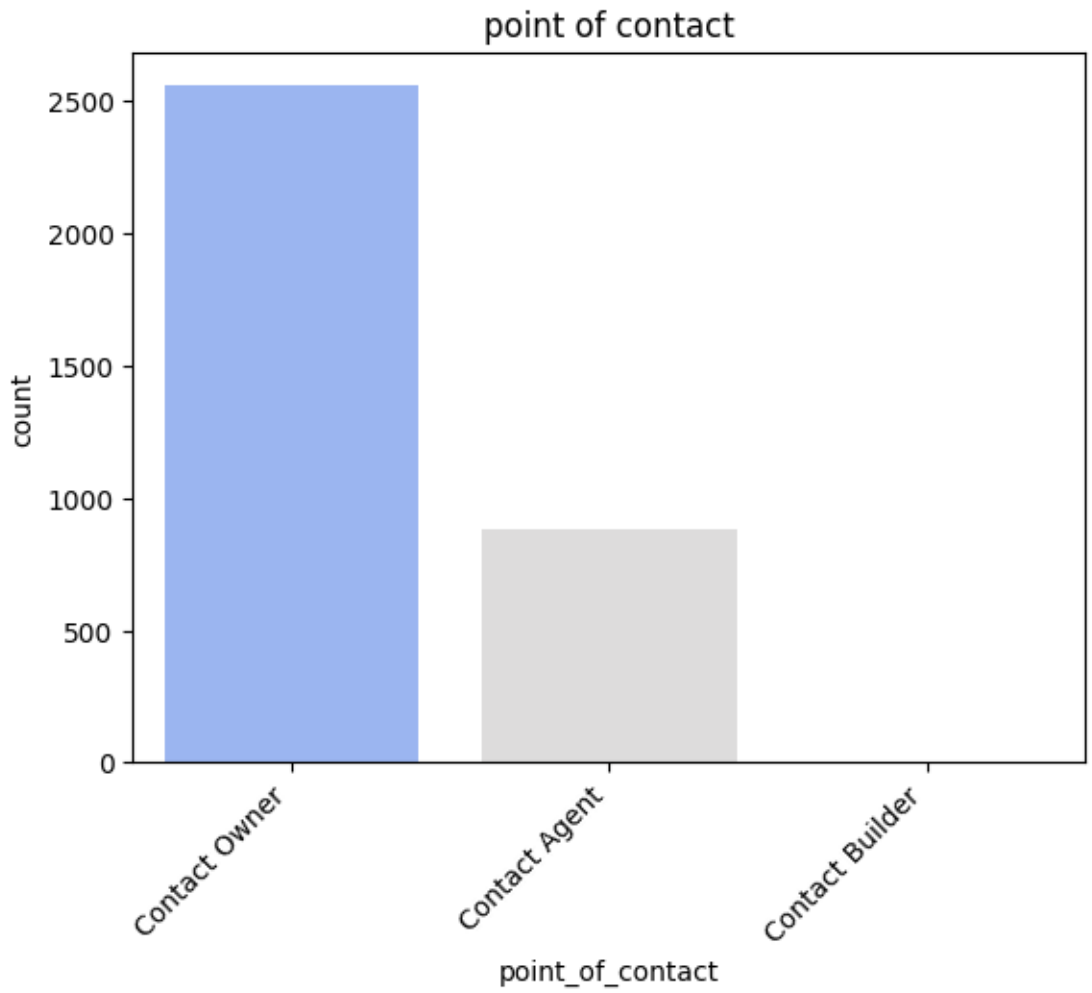
[ ]:
```
sns.countplot(x=training_df['tenancy_preference'], palette="coolwarm")
plt.title('Tenancy preference')
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
<ipython-input-30-c10296f5d189>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=training_df['tenancy_preference'], palette="coolwarm")
```



[ ]:
```
sns.countplot(x=training_df['point_of_contact'], palette="coolwarm")
plt.title('point of contact')
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
<ipython-input-31-d929dd96fbcc>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=training_df['point_of_contact'], palette="coolwarm")
```



[21]:
```
# @title Feature Insights

wgt_eda_feature_insights = widgets.Textarea(
    value="The majority of accommodations are in level 1 out of 2. Generally,␣
 ↪low-level buildings are more prevalent in the testing set. Additionally,␣
 ↪there are three possible tenancy preferences, with bachelors/families having␣
 ↪the highest preference. Lastly, most houses offer the option to contact the␣
 ↪owner in case of renting.",
    placeholder='<student to fill this section>',
    description='Feature Insights:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_eda_feature_insights
```

```
Textarea(value='The majority of accommodations are in level 1 out of 2.␣
↪Generally, low-level buildings are mor…
```

[22]:
```
print("Feature Insights:", wgt_eda_feature_insights.value)
```

```
Feature Insights: The majority of accommodations are in level 1 out of 2.
Generally, low-level buildings are more prevalent in the testing set.
Additionally, there are three possible tenancy preferences, with
bachelors/families having the highest preference. Lastly, most houses offer the
option to contact the owner in case of renting.
```

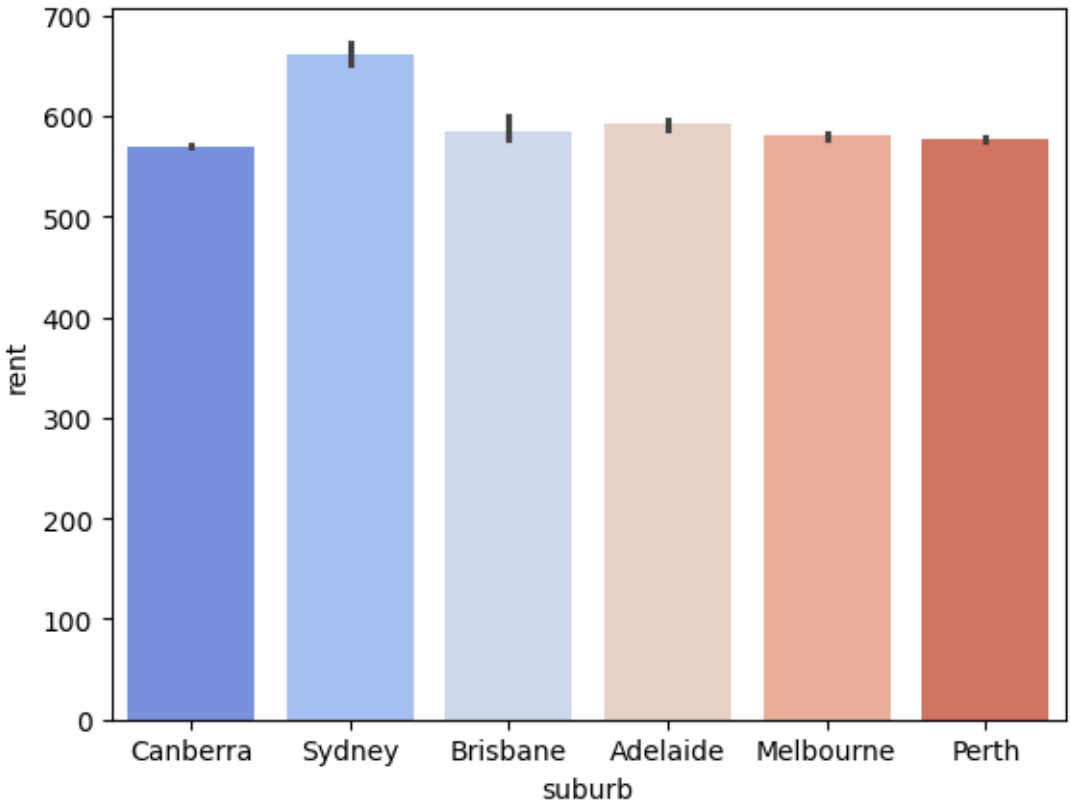
1.5 D. Feature Selection

1.5.1 D.1 Approach 1

[ ]:
```
# <Student to fill this section>
sns.barplot(x=training_df['suburb'], y=training_df['rent'], palette="coolwarm")
```

```
<ipython-input-32-133616612b57>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=training_df['suburb'], y=training_df['rent'],
palette="coolwarm")
```

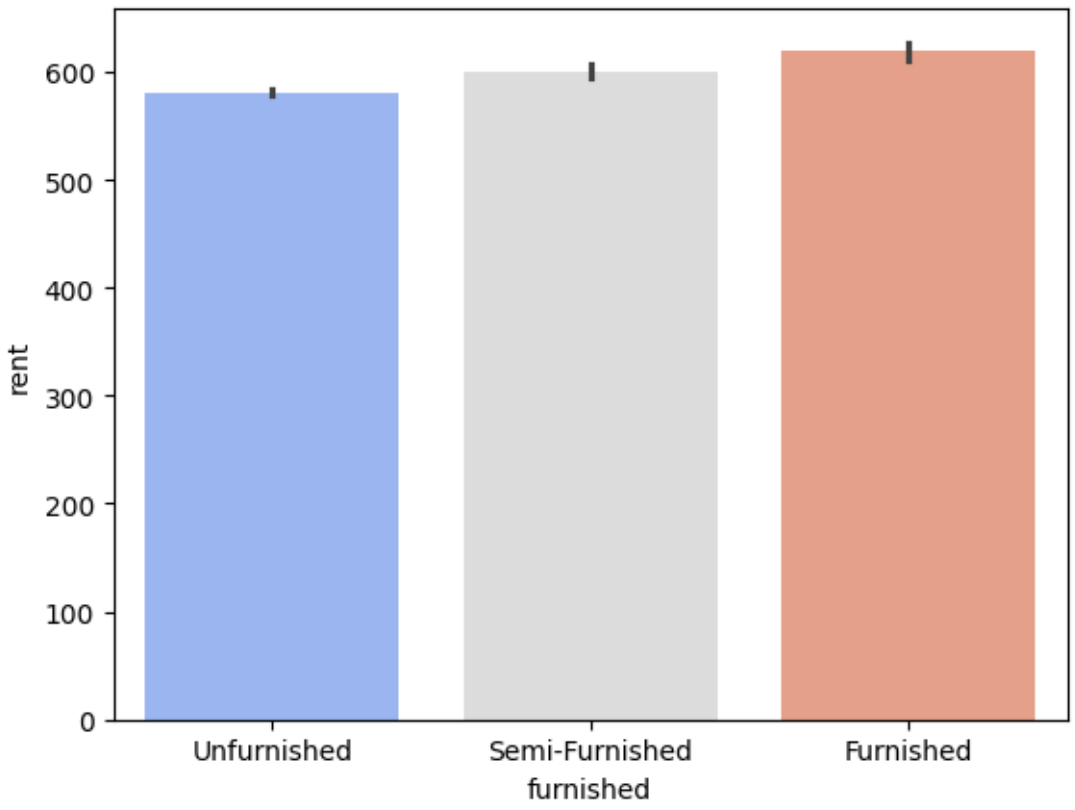
[ ]:
```
<Axes: xlabel='suburb', ylabel='rent'>
```



[ ]:
```
sns.barplot(x=training_df['furnished'], y=training_df['rent'],␣
↪palette="coolwarm")
```

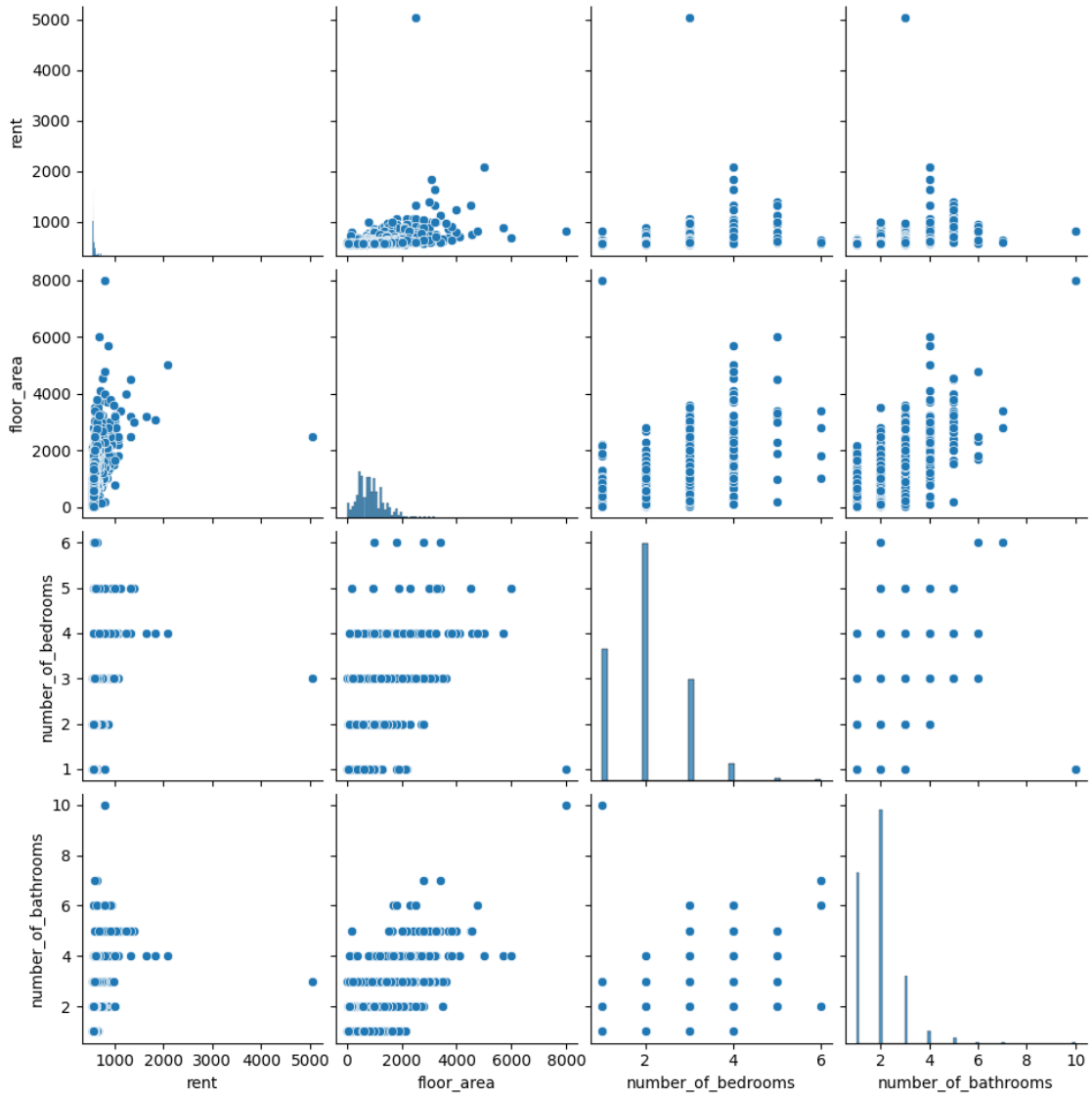
```
<ipython-input-33-8459c0047291>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=training_df['furnished'], y=training_df['rent'],
palette="coolwarm")
```

[ ]:
```
<Axes: xlabel='furnished', ylabel='rent'>
```



[ ]:
```
sns.pairplot(training_df[['rent', 'floor_area', 'number_of_bedrooms',␣
↪'number_of_bathrooms']])
```

[ ]: <seaborn.axisgrid.PairGrid at 0x7d375c1d3c50>



[23]:
```
# @title Feature Selection 1 Insights

wgt_feat_selection_1_insights = widgets.Textarea(
    value="Rent prices vary across cities such as Canberra, Sydney, Brisbane,␣
 ↪Adelaide, Melbourne, and Perth, highlighting regional variations in the␣
 ↪market. They also differ based on furnished status, suggesting a slight␣
 ↪positive correlation. Additionally, houses with 4 bedrooms or bathrooms tend␣
 ↪to have higher rental prices. Naturally, houses with more bedrooms also␣
 ↪appear to have more bathrooms. These features seem to be relevant to rental␣
 ↪prices and may have a positive impact on the prediction.",
    placeholder='<student to fill this section>',
    description='Feature Selection 1:',
    disabled=False,
```

```
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_feat_selection_1_insights
```

```
Textarea(value='Rent prices vary across cities such as Canberra, Sydney,␣
↪Brisbane, Adelaide, Melbourne, and Pe…
```

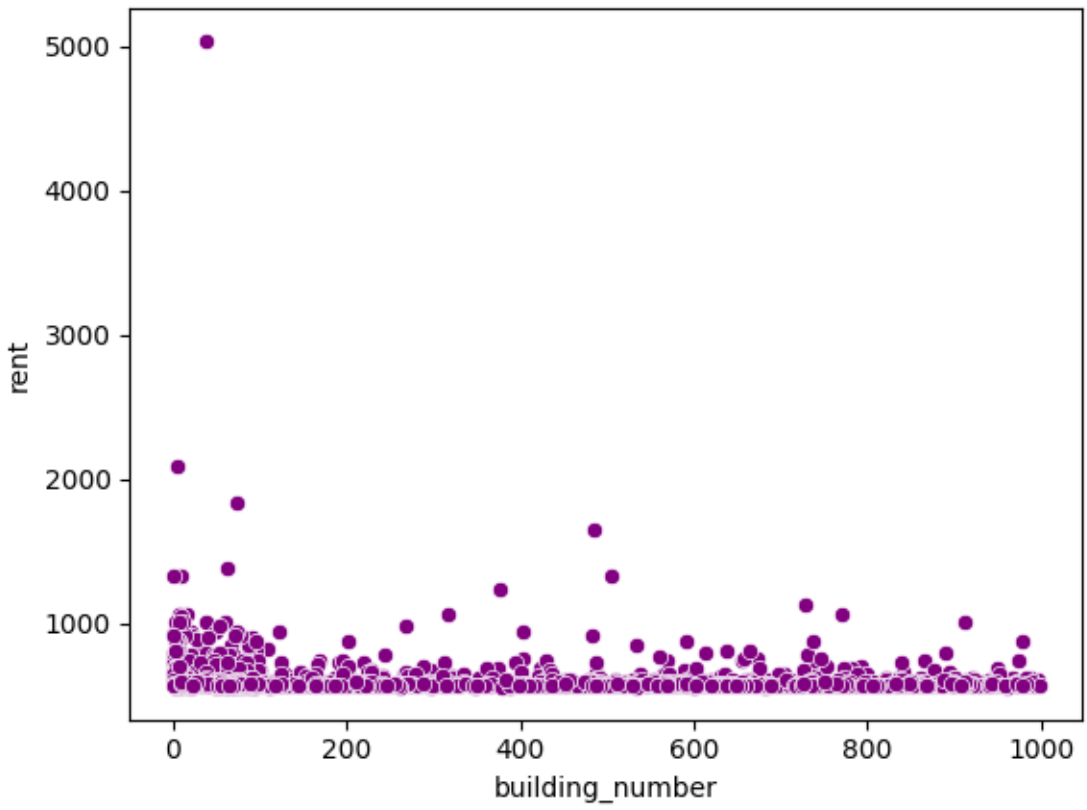
[24]:
```
print("Feature Selection 1:", wgt_feat_selection_1_insights.value)
```

```
Feature Selection 1: Rent prices vary across cities such as Canberra, Sydney,
Brisbane, Adelaide, Melbourne, and Perth, highlighting regional variations in
the market. They also differ based on furnished status, suggesting a slight
positive correlation. Additionally, houses with 4 bedrooms or bathrooms tend to
have higher rental prices. Naturally, houses with more bedrooms also appear to
have more bathrooms. These features seem to be relevant to rental prices and may
have a positive impact on the prediction.
```

1.5.2 D.2 Approach 2

[ ]:
```
# <Student to fill this section>
sns.scatterplot(x=training_df['building_number'], y=training_df['rent'],␣
↪color="purple")
```

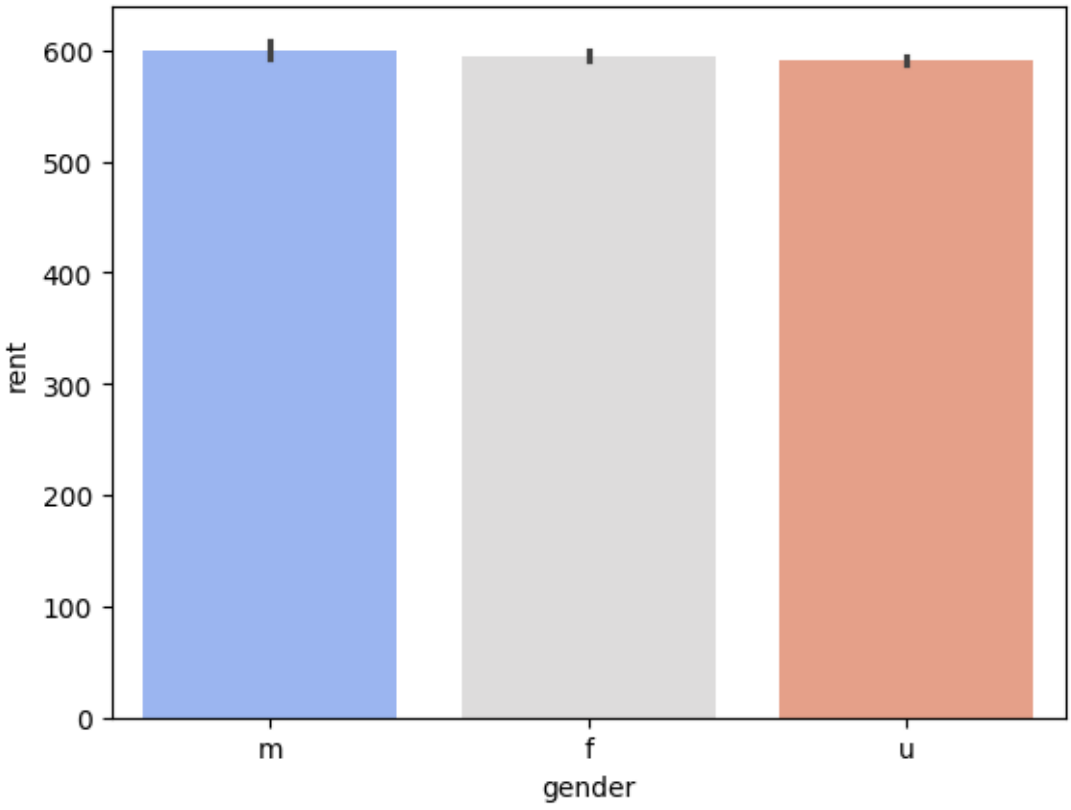
[ ]:
```
<Axes: xlabel='building_number', ylabel='rent'>
```



[ ]:
```
sns.barplot(x=training_df['gender'], y=training_df['rent'], palette="coolwarm")
```

```
<ipython-input-36-f96265192fb4>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=training_df['gender'], y=training_df['rent'],
palette="coolwarm")
```

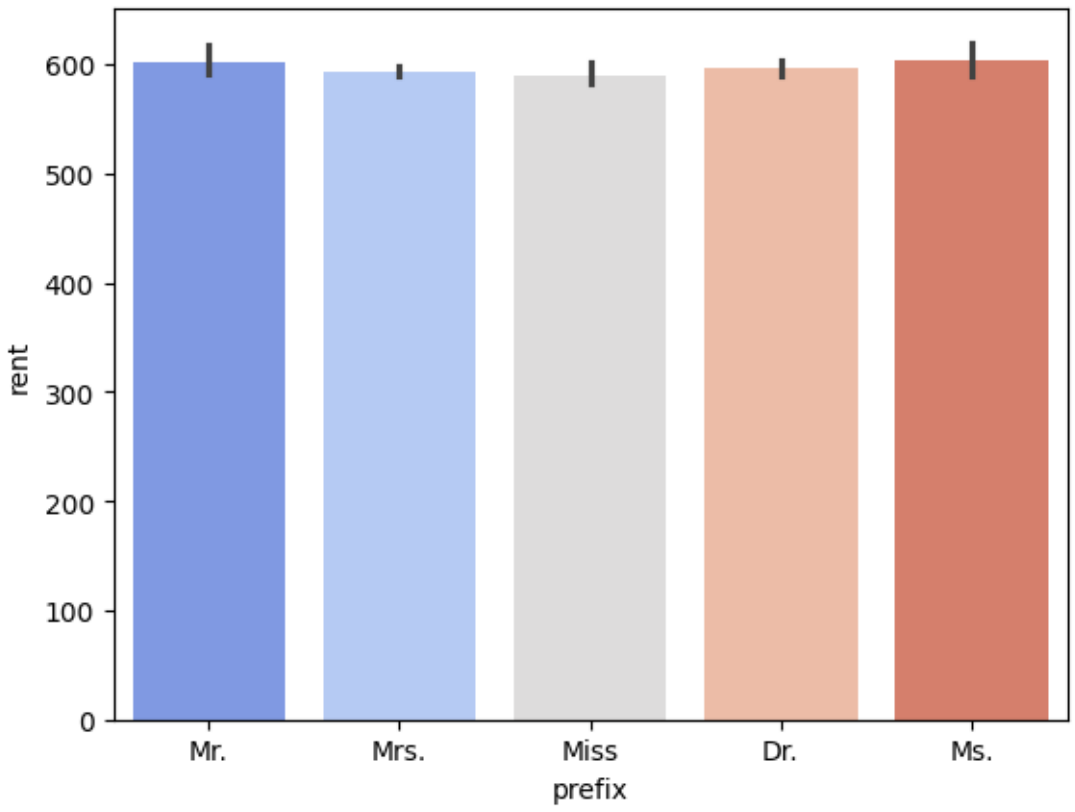
[ ]:
```
<Axes: xlabel='gender', ylabel='rent'>
```



[ ]:
```
sns.barplot(x=training_df['prefix'], y=training_df['rent'], palette="coolwarm")
```

```
<ipython-input-37-048444da1c53>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=training_df['prefix'], y=training_df['rent'],
palette="coolwarm")
```

[ ]:
```
<Axes: xlabel='prefix', ylabel='rent'>
```



[25]:
```
# @title Feature Selection 2 Insights

wgt_feat_selection_2_insights = widgets.Textarea(
    value="Features like building number, agent's gender, or the prefix of the␣
 ↪agent do not show a significant impact on rental prices. Keeping these␣
 ↪features may be redundant and could potentially confuse the model,␣
 ↪disrupting the prediction process. Therefore, it is better to remove them as␣
 ↪they are not necessary.",
    placeholder='<student to fill this section>',
    description='Feature Selection 2:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_feat_selection_2_insights
```

```
Textarea(value="Features like building number, agent's gender, or the prefix of␣
 ↪the agent do not show a signif…
```

[26]:
```
print("Feature Selection 2:", wgt_feat_selection_2_insights.value)
```

```
Feature Selection 2: Features like building number, agent's gender, or the
prefix of the agent do not show a significant impact on rental prices. Keeping
these features may be redundant and could potentially confuse the model,
disrupting the prediction process. Therefore, it is better to remove them as
they are not necessary.
```

1.6 D.3 Final Selection of Features

Save the names of selected features into a list called `features_list`

[ ]:
```
# <Student to fill this section>

features_list =␣
 ↪['advertised_date','number_of_bedrooms','rent','floor_area','level','suburb','furnished','t
```

[27]:
```
# @title Feature Selection Explanation

wgt_feat_selection_explanation = widgets.Textarea(
    value="The selected features for predicting rent have a direct influence on␣
 ↪rental prices. For example, a higher number of bedrooms or a larger floor␣
 ↪area generally leads to higher rent. On the other hand, some features not␣
 ↪included in the selection may be redundant for rent prediction. For instance␣
 ↪personal details such as first name, last name, gender, phone number, and␣
 ↪email are unrelated to rental pricing.",
    placeholder='<student to fill this section>',
    description='Feature Selection Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_feat_selection_explanation
```

```
Textarea(value='The selected features for predicting rent have a direct␣
 ↪influence on rental prices. For exampl…
```

[28]:
```
print("Feature Selection Explanation:", wgt_feat_selection_explanation.value)
```

```
Feature Selection Explanation: The selected features for predicting rent have a
direct influence on rental prices. For example, a higher number of bedrooms or a
larger floor area generally leads to higher rent. On the other hand, some
features not included in the selection may be redundant for rent prediction. For
instance personal details such as first name, last name, gender, phone number,
and email are unrelated to rental pricing.
```

1.7 E. Data Cleaning

1.7.1 E.1 Copy Datasets

Create copies of the datasets and called them `training_df_clean`, `validation_df_clean` and `testing_df_clean`

Do not change this code

[ ]:
```
# Create copy of datasets

training_df_clean = training_df[features_list].copy()
validation_df_clean = validation_df[features_list].copy()
testing_df_clean = testing_df[features_list].copy()
```

1.7.2 E.2 Fixing "Data Types"

Provide some explanations on why you believe it is important to fix this issue and its impacts

You can add more cells in this section

[ ]:
```
# <Student to fill this section>
training_df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3434 entries, 0 to 3433
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   advertised_date      3434 non-null   datetime64[ns]
 1   number_of_bedrooms   3434 non-null   int64
 2   rent                 3434 non-null   float64
 3   floor_area           3434 non-null   int64
 4   level                3434 non-null   object
 5   suburb               3434 non-null   object
 6   furnished            3434 non-null   object
 7   tenancy_preference   3434 non-null   object
 8   number_of_bathrooms  3434 non-null   int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(4)
memory usage: 241.6+ KB
```

[ ]:
```
validation_df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1320 entries, 0 to 1319
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   advertised_date      1320 non-null   datetime64[ns]
```

```
 1   number_of_bedrooms   1320 non-null   int64
 2   rent                 1320 non-null   float64
 3   floor_area           1320 non-null   int64
 4   level                1320 non-null   object
 5   suburb               1320 non-null   object
 6   furnished            1320 non-null   object
 7   tenancy_preference   1320 non-null   object
 8   number_of_bathrooms  1320 non-null   int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(4)
memory usage: 92.9+ KB
```

[ ]:
```
testing_df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1364 entries, 0 to 1363
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   advertised_date      1364 non-null   object
 1   number_of_bedrooms   1364 non-null   int64
 2   rent                 1364 non-null   float64
 3   floor_area           1364 non-null   int64
 4   level                1364 non-null   object
 5   suburb               1364 non-null   object
 6   furnished            1364 non-null   object
 7   tenancy_preference   1364 non-null   object
 8   number_of_bathrooms  1364 non-null   int64
dtypes: float64(1), int64(3), object(5)
memory usage: 96.0+ KB
```

[ ]:
```
training_df_clean[['level','suburb','furnished','tenancy_preference']] =␣
 ↪training_df_clean[['level','suburb','furnished','tenancy_preference']].
 ↪astype('string')
validation_df_clean[['level','suburb','furnished','tenancy_preference']] =␣
 ↪validation_df_clean[['level','suburb','furnished','tenancy_preference']].
 ↪astype('string')
testing_df_clean[['level','suburb','furnished','tenancy_preference']] =␣
 ↪testing_df_clean[['level','suburb','furnished','tenancy_preference']].
 ↪astype('string')
```

[ ]:
```
training_df_clean['advertised_date'] = pd.
 ↪to_datetime(training_df_clean['advertised_date'], format='%Y-%m-%d')
validation_df_clean['advertised_date'] = pd.
 ↪to_datetime(validation_df_clean['advertised_date'], format='%Y-%m-%d')
testing_df_clean['advertised_date'] = pd.
 ↪to_datetime(testing_df_clean['advertised_date'], format='%Y-%m-%d')
```

[ ]:
```
training_df_clean.dtypes
```

[ ]:
```
advertised_date        datetime64[ns]
number_of_bedrooms              int64
rent                          float64
floor_area                      int64
level                  string[python]
suburb                 string[python]
furnished              string[python]
tenancy_preference     string[python]
number_of_bathrooms             int64
dtype: object
```

[ ]:
```
validation_df_clean.dtypes
```

[ ]:
```
advertised_date        datetime64[ns]
number_of_bedrooms              int64
rent                          float64
floor_area                      int64
level                  string[python]
suburb                 string[python]
furnished              string[python]
tenancy_preference     string[python]
number_of_bathrooms             int64
dtype: object
```

[ ]:
```
testing_df_clean.dtypes
```

[ ]:
```
advertised_date        datetime64[ns]
number_of_bedrooms              int64
rent                          float64
floor_area                      int64
level                  string[python]
suburb                 string[python]
furnished              string[python]
tenancy_preference     string[python]
number_of_bathrooms             int64
dtype: object
```

[29]:
```
# @title Data Cleaning 1 Explanation

wgt_data_cleaning_1_explanation = widgets.Textarea(
```

```
    value="Converting the data type from 'object' to 'string' ensures that␣
 ↪text-based columns are properly recognised as categorical data, improving␣
 ↪both data processing efficiency and compatibility with machine learning␣
 ↪algorithms. Additionally, the advertised_date column should be converted to␣
 ↪datetime, as required for future feature engineering. This conversion␣
 ↪enables better handling of time-related operations, avoiding potential␣
 ↪issues when performing string operations or working with dates.",
    placeholder='<student to fill this section>',
    description='Data Cleaning 1 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_data_cleaning_1_explanation
```

```
Textarea(value="Converting the data type from 'object' to 'string' ensures that␣
 ↪text-based columns are properl…
```

[30]:
```
print("Data Cleaning 1 Explanation:", wgt_data_cleaning_1_explanation.value)
```

```
Data Cleaning 1 Explanation: Converting the data type from 'object' to 'string'
ensures that text-based columns are properly recognised as categorical data,
improving both data processing efficiency and compatibility with machine
learning algorithms. Additionally, the advertised_date column should be
converted to datetime, as required for future feature engineering. This
conversion enables better handling of time-related operations, avoiding
potential issues when performing string operations or working with dates.
```

1.7.3 E.3 Fixing "Missing Values & Duplicates"

Provide some explanations on why you believe it is important to fix this issue and its impacts

You can add more cells in this section

[ ]:
```
# <Student to fill this section>
print(training_df_clean.isna().sum())
print("..............................")
print(validation_df_clean.isna().sum())
print("..............................")
print(testing_df_clean.isna().sum())
```

```
advertised_date        0
number_of_bedrooms     0
rent                   0
floor_area             0
level                  0
suburb                 0
furnished              0
```

```
tenancy_preference     0
number_of_bathrooms    0
dtype: int64
…
advertised_date        0
number_of_bedrooms     0
rent                   0
floor_area             0
level                  0
suburb                 0
furnished              0
tenancy_preference     0
number_of_bathrooms    0
dtype: int64
…
advertised_date        0
number_of_bedrooms     0
rent                   0
floor_area             0
level                  0
suburb                 0
furnished              0
tenancy_preference     0
number_of_bathrooms    0
dtype: int64
```

[ ]:
```
duplicate_count = training_df_clean.duplicated().sum()
duplicate_count1 = validation_df_clean.duplicated().sum()
duplicate_count2 = testing_df_clean.duplicated().sum()

print("training_df_clean duplicates: ", duplicate_count)
print("validation_df_clean duplicates: ", duplicate_count)
print("testing_df_clean duplicates: ", duplicate_count)
```

```
training_df_clean duplicates:  6
validation_df_clean duplicates:  6
testing_df_clean duplicates:  6
```

[ ]:
```
training_df_clean.drop_duplicates(inplace=True)
validation_df.drop_duplicates(inplace=True)
testing_df_clean.drop_duplicates(inplace=True)
```

[ ]:
```
duplicate_count = training_df_clean.duplicated().sum()
duplicate_count1 = validation_df_clean.duplicated().sum()
duplicate_count2 = testing_df_clean.duplicated().sum()

print("training_df_clean duplicates: ", duplicate_count)
```

```
print("validation_df_clean duplicates: ", duplicate_count)
print("testing_df_clean duplicates: ", duplicate_count)
```

```
training_df_clean duplicates:  0
validation_df_clean duplicates:  0
testing_df_clean duplicates:  0
```

[31]:
```
# @title Data Cleaning 2 Explanation

wgt_data_cleaning_2_explanation = widgets.Textarea(
    value="There were no missing values in the dataset, ensuring that no␣
 ↪imputation of missing data was necessary during preprocessing. On the other␣
 ↪hand, duplicates were identified and removed to ensure data consistency and␣
 ↪accuracy, preventing them from skewing the analysis or model predictions.␣
 ↪Removing duplicates helps improve the quality of the dataset by ensuring␣
 ↪that each observation is unique and representative.",
    placeholder='<student to fill this section>',
    description='Data Cleaning 1 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_data_cleaning_2_explanation
```

```
Textarea(value='There were no missing values in the dataset, ensuring that no␣
 ↪imputation of missing data was n…
```

[32]:
```
print("Data Cleaning 2 Explanation:", wgt_data_cleaning_2_explanation.value)
```

```
Data Cleaning 2 Explanation: There were no missing values in the dataset,
ensuring that no imputation of missing data was necessary during preprocessing.
On the other hand, duplicates were identified and removed to ensure data
consistency and accuracy, preventing them from skewing the analysis or model
predictions. Removing duplicates helps improve the quality of the dataset by
ensuring that each observation is unique and representative.
```

1.7.4 E.4 Fixing "Outliers"

Provide some explanations on why you believe it is important to fix this issue and its impacts

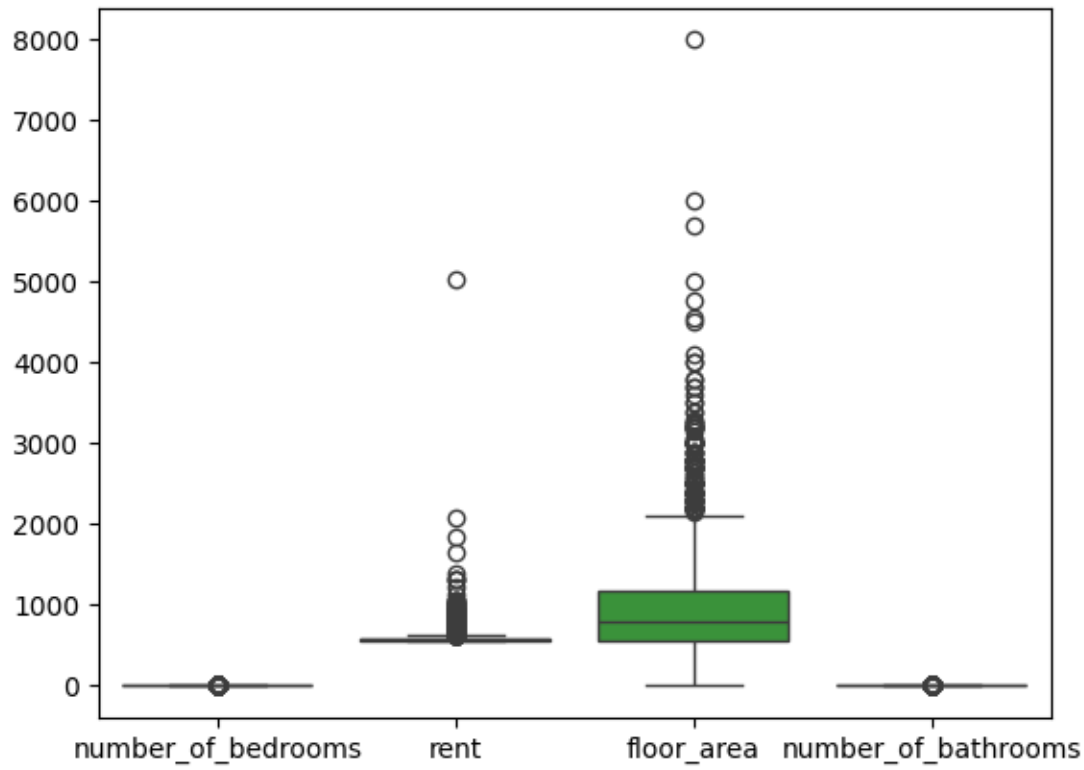
You can add more cells in this section

[ ]:
```
# <Student to fill this section>

Q1 = training_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.25)
Q3 = training_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.75)
```

```
IQR = Q3 - Q1

# Define outliers
outliers_iqr = training_df_clean[((training_df_clean[['number_of_bedrooms',␣
 ↪'rent', 'floor_area', 'number_of_bathrooms']] < (Q1 - 1.5 * IQR)) |
                                  (training_df_clean[['number_of_bedrooms',␣
 ↪'rent', 'floor_area', 'number_of_bathrooms']] > (Q3 + 1.5 * IQR))).
 ↪any(axis=1)]

print(outliers_iqr)
```

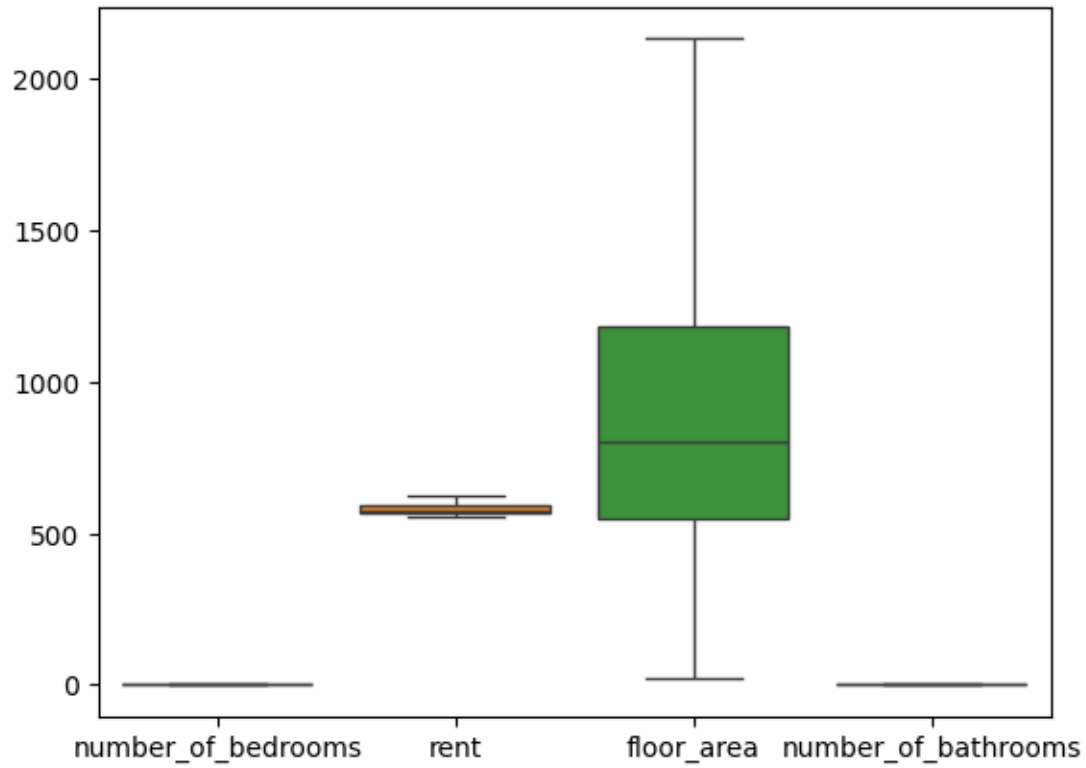
```
     advertised_date  number_of_bedrooms   rent  floor_area            level  \
73        2022-06-21                   6  581.0        1000        1 out of 1
85        2022-04-29                   4  578.0        1200        1 out of 2
88        2022-06-13                   4  600.0        1600        1 out of 5
90        2022-06-16                   2  786.0         950   Ground out of 1
141       2022-04-30                   4  606.0        1300        1 out of 2
…                …                   …     …           …                …
3368      2022-06-20                   4  613.0        2300        4 out of 5
3389      2022-06-18                   3  632.0        2170        4 out of 5
3412      2022-06-17                   3  600.0        2500        2 out of 2
3413      2022-04-30                   4  576.0        1000        1 out of 3
3415      2022-04-29                   4  696.0        3250     12 out of 17

        suburb       furnished tenancy_preference  number_of_bathrooms
73    Canberra  Semi-Furnished   Bachelors/Family                    2
85    Canberra     Unfurnished             Family                    2
88    Canberra     Unfurnished   Bachelors/Family                    3
90    Canberra     Unfurnished   Bachelors/Family                    2
141   Canberra     Unfurnished   Bachelors/Family                    3
…          …              …                  …                    …
3368     Perth  Semi-Furnished   Bachelors/Family                    4
3389     Perth  Semi-Furnished   Bachelors/Family                    3
3412     Perth     Unfurnished   Bachelors/Family                    2
3413     Perth  Semi-Furnished   Bachelors/Family                    2
3415     Perth  Semi-Furnished   Bachelors/Family                    5

[466 rows x 9 columns]
```

[ ]:
```
# Create box plot for each numeric feature
sns.boxplot(data=training_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']])
plt.show()
```



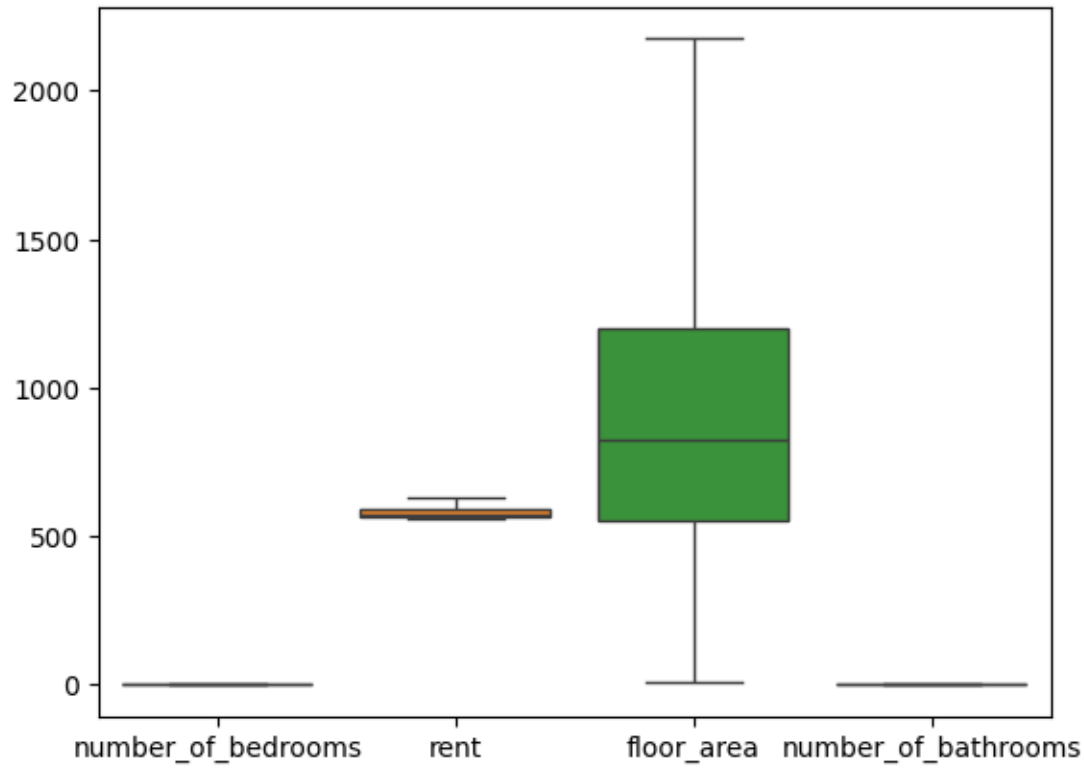
[ ]:
```
Q1 = training_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.25)
Q3 = training_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Apply clipping
training_df_clean['number_of_bedrooms'] =␣
 ↪training_df_clean['number_of_bedrooms'].
 ↪clip(lower=lower_bound['number_of_bedrooms'],␣
 ↪upper=upper_bound['number_of_bedrooms'])
training_df_clean['rent'] = training_df_clean['rent'].
 ↪clip(lower=lower_bound['rent'], upper=upper_bound['rent'])
training_df_clean['floor_area'] = training_df_clean['floor_area'].
 ↪clip(lower=lower_bound['floor_area'], upper=upper_bound['floor_area'])
training_df_clean['number_of_bathrooms'] =␣
 ↪training_df_clean['number_of_bathrooms'].
 ↪clip(lower=lower_bound['number_of_bathrooms'],␣
 ↪upper=upper_bound['number_of_bathrooms'])
```

```
# Create box plot for each numeric feature
sns.boxplot(data=training_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']])
plt.show()
```



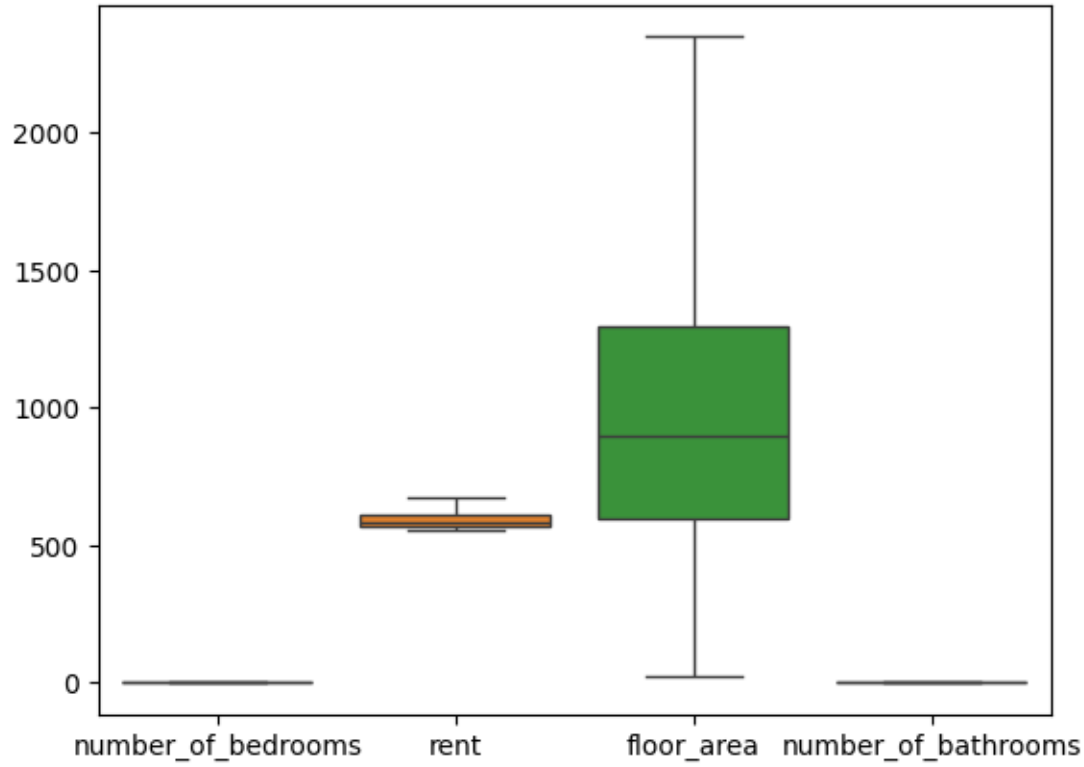
[ ]:
```
Q1 = validation_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.25)
Q3 = validation_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Apply clipping
validation_df_clean['number_of_bedrooms'] =␣
 ↪validation_df_clean['number_of_bedrooms'].
 ↪clip(lower=lower_bound['number_of_bedrooms'],␣
 ↪upper=upper_bound['number_of_bedrooms'])
validation_df_clean['rent'] = validation_df_clean['rent'].
 ↪clip(lower=lower_bound['rent'], upper=upper_bound['rent'])
```

```
validation_df_clean['floor_area'] = validation_df_clean['floor_area'].
 ↪clip(lower=lower_bound['floor_area'], upper=upper_bound['floor_area'])
validation_df_clean['number_of_bathrooms'] =␣
 ↪validation_df_clean['number_of_bathrooms'].
 ↪clip(lower=lower_bound['number_of_bathrooms'],␣
 ↪upper=upper_bound['number_of_bathrooms'])

# Create box plot for each numeric feature
sns.boxplot(data=validation_df_clean[['number_of_bedrooms', 'rent',␣
 ↪'floor_area', 'number_of_bathrooms']])
plt.show()
```



[ ]:
```
Q1 = testing_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.25)
Q3 = testing_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
# Apply clipping
testing_df_clean['number_of_bedrooms'] = testing_df_clean['number_of_bedrooms'].
 ↪clip(lower=lower_bound['number_of_bedrooms'],␣
 ↪upper=upper_bound['number_of_bedrooms'])
testing_df_clean['rent'] = testing_df_clean['rent'].
 ↪clip(lower=lower_bound['rent'], upper=upper_bound['rent'])
testing_df_clean['floor_area'] = testing_df_clean['floor_area'].
 ↪clip(lower=lower_bound['floor_area'], upper=upper_bound['floor_area'])
testing_df_clean['number_of_bathrooms'] =␣
 ↪testing_df_clean['number_of_bathrooms'].
 ↪clip(lower=lower_bound['number_of_bathrooms'],␣
 ↪upper=upper_bound['number_of_bathrooms'])

# Create box plot for each numeric feature
sns.boxplot(data=testing_df_clean[['number_of_bedrooms', 'rent', 'floor_area',␣
 ↪'number_of_bathrooms']])
plt.show()
```



[33]:
```
# @title Data Cleaning 3 Explanation

wgt_data_cleaning_3_explanation = widgets.Textarea(
```

```
    value="The Interquartile Range (IQR) method is used to handle outliers␣
 ↪because it is a robust statistical approach that helps identify and limit␣
 ↪extreme values without being overly sensitive to the data distribution.␣
 ↪Clipping outliers assigns any data points below the lower bound (Q1) to the␣
 ↪lower bound, and any data points above the upper bound (Q3) to the upper␣
 ↪bound. This reduces the effect of extreme outliers without removing them␣
 ↪entirely. The IQR method is more robust and resistant to the influence of␣
 ↪skewed data.",
    placeholder='<student to fill this section>',
    description='Data Cleaning 3 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_data_cleaning_3_explanation
```

```
Textarea(value='The Interquartile Range (IQR) method is used to handle outliers␣
 ↪because it is a robust statist…
```

[34]:
```
print("Data Cleaning 3 Explanation:", wgt_data_cleaning_3_explanation.value)
```

```
Data Cleaning 3 Explanation: The Interquartile Range (IQR) method is used to
handle outliers because it is a robust statistical approach that helps identify
and limit extreme values without being overly sensitive to the data
distribution. Clipping outliers assigns any data points below the lower bound
(Q1) to the lower bound, and any data points above the upper bound (Q3) to the
upper bound. This reduces the effect of extreme outliers without removing them
entirely. The IQR method is more robust and resistant to the influence of skewed
data.
```

1.8 F. Feature Engineering

1.8.1 F.1 Copy Datasets

Create copies of the datasets and called them `training_df_eng`, `validation_df_eng` and `testing_df_eng`

Do not change this code

[ ]:
```
# Create copy of datasets

training_df_eng = training_df_clean.copy()
validation_df_eng = validation_df_clean.copy()
testing_df_eng = testing_df_clean.copy()
```

1.8.2 F.2 New Feature "month"

Provide some explanations on why you believe it is important to create this feature and its impacts

[ ]:
```
# <Student to fill this section>
training_df_eng['month'] = training_df_eng['advertised_date'].dt.month
validation_df_eng['month'] = validation_df_eng['advertised_date'].dt.month
testing_df_eng['month'] = testing_df_eng['advertised_date'].dt.month
```

[ ]:
```
training_df_eng.drop('advertised_date', axis=1, inplace=True)
validation_df_eng.drop('advertised_date', axis=1, inplace=True)
testing_df_eng.drop('advertised_date', axis=1, inplace=True)
```

[38]:
```
# @title Feature Engineering 1 Explanation

wgt_feature_engineering_1_explanation = widgets.Textarea(
    value="The new feature Month was created to capture potential seasonal␣
 ↪variations in rent prices. Some months may experience higher rents due to␣
 ↪increased demand, such as during peak rental seasons or holidays.",
    placeholder='<student to fill this section>',
    description='Feature Engineering 1 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_feature_engineering_1_explanation
```

```
Textarea(value='The new feature Month was created to capture potential seasonal␣
 ↪variations in rent prices. Som…
```

[39]:
```
print("Feature Engineering 1 Explanation:",␣
 ↪wgt_feature_engineering_1_explanation.value)
```

```
Feature Engineering 1 Explanation: The new feature Month was created to capture
potential seasonal variations in rent prices. Some months may experience higher
rents due to increased demand, such as during peak rental seasons or holidays.
```

[ ]:
```
training_df_eng.head()
```

[ ]:

	number_of_bedrooms	rent	floor_area	level	suburb	furnished	tenancy_preference	number_of_bathrooms	month
0	2.0	568.0	1100	Ground out of 2	Canberra				
1	2.0	581.0	800	1 out of 3	Canberra				
2	2.0	577.0	1000	1 out of 3	Canberra				
3	2.0	565.0	850	1 out of 2	Canberra				
4	2.0	564.0	600	Ground out of 1	Canberra				

```
0     Unfurnished   Bachelors/Family                  2.0      5
1  Semi-Furnished   Bachelors/Family                  1.0      5
2  Semi-Furnished   Bachelors/Family                  1.0      5
3     Unfurnished          Bachelors                  1.0      5
4     Unfurnished   Bachelors/Family                  2.0      4
```

1.8.3 F.3 New Feature "level_numerator"

Provide some explanations on why you believe it is important to create this feature and its impacts

[ ]:
```
# <Student to fill this section>
training_df_eng['level'].value_counts()
```

[ ]:
```
level
1 out of 2          289
Ground out of 2     272
2 out of 3          261
1 out of 3          239
2 out of 4          235
                   ...
5 out of 34           1
4 out of 31           1
4 out of 26           1
1 out of 35           1
12 out of 17          1
Name: count, Length: 334, dtype: Int64
```

[ ]:
```
training_df_eng[['level_numerator', 'level_denominator']] =␣
 ↪training_df_eng['level'].str.split(' out of ', expand=True)

training_df_eng['level_numerator'] = training_df_eng['level_numerator'].
 ↪replace('Ground', '0')
training_df_eng['level_numerator'] = training_df_eng['level_numerator'].
 ↪replace('Lower Basement', '-2')
training_df_eng['level_numerator'] = training_df_eng['level_numerator'].
 ↪replace('Upper Basement', '-1')
training_df_eng['level_denominator'] = training_df_eng['level_denominator'].
 ↪fillna('1')

training_df_eng['level_numerator'] = training_df_eng['level_numerator'].
 ↪astype(int)
training_df_eng['level_denominator'] = training_df_eng['level_denominator'].
 ↪astype(int)
```

[ ]:
```
validation_df_eng[['level_numerator', 'level_denominator']] =␣
 ↪validation_df_eng['level'].str.split(' out of ', expand=True)
```

```
validation_df_eng['level_numerator'] = validation_df_eng['level_numerator'].
 ↪replace('Ground', '0')
validation_df_eng['level_numerator'] = validation_df_eng['level_numerator'].
 ↪replace('Lower Basement', '-2')
validation_df_eng['level_numerator'] = validation_df_eng['level_numerator'].
 ↪replace('Upper Basement', '-1')
validation_df_eng['level_denominator'] = validation_df_eng['level_denominator'].
 ↪fillna('1')

validation_df_eng['level_numerator'] = validation_df_eng['level_numerator'].
 ↪astype(int)
validation_df_eng['level_denominator'] = validation_df_eng['level_denominator'].
 ↪astype(int)
```

[ ]:
```
testing_df_eng[['level_numerator', 'level_denominator']] =␣
 ↪testing_df_eng['level'].str.split(' out of ', expand=True)

testing_df_eng['level_numerator'] = testing_df_eng['level_numerator'].
 ↪replace('Ground', '0')
testing_df_eng['level_numerator'] = testing_df_eng['level_numerator'].
 ↪replace('Lower Basement', '-2')
testing_df_eng['level_numerator'] = testing_df_eng['level_numerator'].
 ↪replace('Upper Basement', '-1')
testing_df_eng['level_denominator'] = testing_df_eng['level_denominator'].
 ↪fillna('1')

testing_df_eng['level_numerator'] = testing_df_eng['level_numerator'].
 ↪astype(int)
testing_df_eng['level_denominator'] = testing_df_eng['level_denominator'].
 ↪astype(int)
```

[47]:
```
# @title Feature Engineering 2 Explanation

wgt_feature_engineering_2_explanation = widgets.Textarea(
    value="Importance of the level_numerator Feature: The level_numeratoe is␣
 ↪valuable as an indicator of the specific floor level to preserve the␣
 ↪original data and its significance. Impacts on the Model: The␣
 ↪level_numerator refines the model's ability to capture floor-level␣
 ↪variations, leading to more accurate rent predictions.",
    placeholder='<student to fill this section>',
    description='Feature Engineering 2 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
```

```
wgt_feature_engineering_2_explanation
```

```
Textarea(value="Importance of the level_numerator Feature: The level_numeratoe␣
 ↪is valuable as an indicator of …
```

[48]:
```
print("Feature Engineering 2 Explanation:",␣
 ↪wgt_feature_engineering_2_explanation.value)
```

```
Feature Engineering 2 Explanation: Importance of the level_numerator Feature:
The level_numeratoe is valuable as an indicator of the specific floor level to
preserve the original data and its significance. Impacts on the Model: The
level_numerator refines the model's ability to capture floor-level variations,
leading to more accurate rent predictions.
```

1.8.4 F.4 New Feature "level_ratio"

Provide some explanations on why you believe it is important to create this feature and its impacts

[ ]:
```
# <Student to fill this section>
training_df_eng['level_ratio'] = training_df_eng['level_numerator'] /␣
 ↪training_df_eng['level_denominator']

training_df_eng.drop(['level_denominator', 'level'], axis=1, inplace=True)
```

[ ]:
```
validation_df_eng['level_ratio'] = validation_df_eng['level_numerator'] /␣
 ↪validation_df_eng['level_denominator']

validation_df_eng.drop(['level_denominator', 'level'], axis=1, inplace=True)
```

[ ]:
```
testing_df_eng['level_ratio'] = testing_df_eng['level_numerator'] /␣
 ↪testing_df_eng['level_denominator']

testing_df_eng.drop(['level_denominator', 'level'], axis=1, inplace=True)
```

[45]:
```
# @title Feature Engineering 3 Explanation

wgt_feature_engineering_3_explanation = widgets.Textarea(
    value="Importance of the level_ratio Feature: The level_ratio is valuable␣
 ↪as it provides a continuous measure of a property's floor position relative␣
 ↪to the total number of floors. This helps the model capture how floor␣
 ↪location impacts rent, with higher floors often commanding higher prices due␣
 ↪to factors like views. Impacts on the Model: The level_ratio improves the␣
 ↪model by simplifying the feature set, combining the floor number and total␣
 ↪floors into one variable. This reduces complexity, avoids redundancy, and␣
 ↪allows the model to better capture patterns in rent prices, enhancing␣
 ↪predictive accuracy.",
```

```
    placeholder='<student to fill this section>',
    description='Feature Engineering 3 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_feature_engineering_3_explanation
```

```
Textarea(value='Importance of the level_ratio Feature: The level_ratio is␣
↪valuable as it provides a continuous…
```

[46]:
```
print("Feature Engineering 3 Explanation:",␣
↪wgt_feature_engineering_3_explanation.value)
```

```
Feature Engineering 3 Explanation: Importance of the level_ratio Feature: The
level_ratio is valuable as it provides a continuous measure of a property's
floor position relative to the total number of floors. This helps the model
capture how floor location impacts rent, with higher floors often commanding
higher prices due to factors like views. Impacts on the Model: The level_ratio
improves the model by simplifying the feature set, combining the floor number
and total floors into one variable. This reduces complexity, avoids redundancy,
and allows the model to better capture patterns in rent prices, enhancing
predictive accuracy.
```

1.9 G. Data Preparation for Modeling

1.9.1 G.1 Copy Datasets

Create copies of the datasets and split them into X and y

Do not change this code

[ ]:
```
# Create copy of datasets

X_train = training_df_eng.copy()
X_val = validation_df_eng.copy()
X_test = testing_df_eng.copy()

y_train = X_train.pop(target_name)
y_val = X_val.pop(target_name)
y_test = X_test.pop(target_name)
```

1.9.2 G.2 Data Transformation

Provide some explanations on why you believe it is important to perform this data transformation and its impacts

[ ]:
```
# <Student to fill this section>

# One-Hot Encoding for 'tenancy_preference' and 'suburb'
df_dummies = pd.get_dummies(X_train[['tenancy_preference', 'suburb']])
df_dummies = df_dummies.astype(int)
X_train = pd.concat([X_train, df_dummies], axis=1)
X_train.drop(['tenancy_preference', 'suburb'], axis=1, inplace=True)
```

[ ]:
```
# One-Hot Encoding for 'tenancy_preference' and 'suburb'
df_dummies = pd.get_dummies(X_val[['tenancy_preference', 'suburb']])
df_dummies = df_dummies.astype(int)

X_val = pd.concat([X_val, df_dummies], axis=1)
X_val.drop(['tenancy_preference', 'suburb'], axis=1, inplace=True)
```

[ ]:
```
# One-Hot Encoding for 'tenancy_preference' and 'suburb'
df_dummies = pd.get_dummies(X_test[['tenancy_preference', 'suburb']])
df_dummies = df_dummies.astype(int)

X_test = pd.concat([X_test, df_dummies], axis=1)
X_test.drop(['tenancy_preference', 'suburb'], axis=1, inplace=True)
```

[49]:
```
# @title Data Preparation 1 Explanation

wgt_data_preparation_1_explanation = widgets.Textarea(
    value="Proper encoding allows the model to identify patterns and make␣
 ↪better predictions by correctly interpreting categorical relationships.␣
 ↪Without feature encoding, categorical data might be ignored or␣
 ↪misinterpreted, resulting in poorer model performance. So, get_dummies() is␣
 ↪a crucial step to transform categorical variables into a form that enhances␣
 ↪model training, ensures accurate representation of relationships, and␣
 ↪ultimately improves model performance.",
    placeholder='<student to fill this section>',
    description='Data Preparation 1 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_data_preparation_1_explanation
```

```
Textarea(value='Proper encoding allows the model to identify patterns and make␣
 ↪better predictions by correctly…
```

[50]:
```
print("Data Preparation 1 Explanation:", wgt_data_preparation_1_explanation.
 ↪value)
```

```
Data Preparation 1 Explanation: Proper encoding allows the model to identify
```

patterns and make better predictions by correctly interpreting categorical relationships. Without feature encoding, categorical data might be ignored or misinterpreted, resulting in poorer model performance. So, get_dummies() is a crucial step to transform categorical variables into a form that enhances model training, ensures accurate representation of relationships, and ultimately improves model performance.

1.9.3 G.3 Data Transformation

Provide some explanations on why you believe it is important to perform this data transformation and its impacts

[ ]:
```
# <Student to fill this section>

furnished_mapping = {
    'Furnished': 2,
    'Semi-Furnished': 1,
    'Unfurnished': 0
}

# Apply the mapping to the 'furnished' column
X_train['furnished_encoded'] = X_train['furnished'].map(furnished_mapping)

X_train.drop(['furnished'], axis=1, inplace=True)
```

[ ]:
```
# Apply the mapping to the 'furnished' column
X_val['furnished_encoded'] = X_val['furnished'].map(furnished_mapping)

X_val.drop(['furnished'], axis=1, inplace=True)
```

[ ]:
```
# Apply the mapping to the 'furnished' column
X_test['furnished_encoded'] = X_test['furnished'].map(furnished_mapping)

X_test.drop(['furnished'], axis=1, inplace=True)
```

[52]:
```
# @title Data Preparation 2 Explanation

wgt_data_preparation_2_explanation = widgets.Textarea(
    value="Mapping is essential for transforming ordinal categorical variables␣
 ↪into a structured numerical format while preserving their inherent order. In␣
 ↪the furnished column: Unfurnished → 0, Semi-Furnished → 1, Furnished → 2.␣
 ↪This approach enables the model to recognise the increasing degree of␣
 ↪furnishing, leading to improved predictive accuracy.Mapping is essential for␣
 ↪transforming ordinal categorical variables into a structured numerical␣
 ↪format while preserving their inherent order. In the furnished column:␣
 ↪Unfurnished → 0, Semi-Furnished → 1, Furnished → 2. This approach enables␣
 ↪the model to recognise the increasing degree of furnishing, leading to␣
 ↪improved predictive accuracy.",
```

```
    placeholder='<student to fill this section>',
    description='Data Preparation 2 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_data_preparation_2_explanation
```

```
Textarea(value='Mapping is essential for transforming ordinal categorical␣
 ↪variables into a structured numerica…
```

[53]:
```
print("Data Preparation 2 Explanation:", wgt_data_preparation_2_explanation.
 ↪value)
```

```
Data Preparation 2 Explanation: Mapping is essential for transforming ordinal
categorical variables into a structured numerical format while preserving their
inherent order. In the furnished column: Unfurnished → 0, Semi-Furnished → 1,
Furnished → 2. This approach enables the model to recognise the increasing
degree of furnishing, leading to improved predictive accuracy.Mapping is
essential for transforming ordinal categorical variables into a structured
numerical format while preserving their inherent order. In the furnished column:
Unfurnished → 0, Semi-Furnished → 1, Furnished → 2. This approach enables the
model to recognise the increasing degree of furnishing, leading to improved
predictive accuracy.
```

1.9.4 G.4 Data Transformation

> Provide some explanations on why you believe it is important to perform this data transformation and its impacts

[ ]:
```
standard_scaler = StandardScaler()

column_names = X_train.columns.tolist()

X_train = pd.DataFrame(standard_scaler.fit_transform(X_train),␣
 ↪columns=column_names)
X_val = pd.DataFrame(standard_scaler.transform(X_val), columns=column_names)
X_test = pd.DataFrame(standard_scaler.transform(X_test), columns=column_names)
```

[54]:
```
# @title Data Preparation 3 Explanation

wgt_data_preparation_3_explanation = widgets.Textarea(
    value="Importance: Standardising features ensures all variables are on the␣
 ↪same scale, preventing large features from dominating the model's learning␣
 ↪process. Impact: It improves model performance by allowing the algorithm to␣
 ↪treat all features equally, leading to better accuracy and faster␣
 ↪convergence, especially in regression models.",
    placeholder='<student to fill this section>',
```

```
    description='Data Preparation 3 Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_data_preparation_3_explanation
```

```
Textarea(value='Importance: Standardising features ensures all variables are on␣
↪the same scale, preventing lar…
```

[55]:
```
print("Data Preparation 3 Explanation:", wgt_data_preparation_3_explanation.
↪value)
```

```
Data Preparation 3 Explanation: Importance: Standardising features ensures all
variables are on the same scale, preventing large features from dominating the
model's learning process. Impact: It improves model performance by allowing the
algorithm to treat all features equally, leading to better accuracy and faster
convergence, especially in regression models.
```

1.10 H. Save Datasets

Do not change this code

[ ]:
```
# Save training set

X_train.to_csv(folder_path / 'X_train.csv', index=False)
y_train.to_csv(folder_path / 'y_train.csv', index=False)
```

[ ]:
```
# Save validation set

X_val.to_csv(folder_path / 'X_val.csv', index=False)
y_val.to_csv(folder_path / 'y_val.csv', index=False)
```

[ ]:
```
# Save testing set

X_test.to_csv(folder_path / 'X_test.csv', index=False)
y_test.to_csv(folder_path / 'y_test.csv', index=False)
```

1.11 I. Assess Baseline Model

1.11.1 I.1 Generate Predictions with Baseline Model

[ ]:
```
# <Student to fill this section>
from sklearn.dummy import DummyRegressor

base_reg = DummyRegressor(strategy='mean')
base_reg.fit(X_train, y_train)
y_train_preds = base_reg.predict(X_train)
```

1.11.2 I.2 Selection of Performance Metrics

Provide some explanations on why you believe the performance metrics you chose is appropriate

[ ]:
```
# <Student to fill this section>

mse = mean_squared_error(y_train, y_train_preds)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_train, y_train_preds)
r2 = r2_score(y_train, y_train_preds)

print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"R2: {r2}")
```

```
MSE: 399.6714953999529
RMSE: 19.99178569812994
MAE: 16.079371746710784
R2: 0.0
```

[56]:
```
# @title Performance Metrics Explanation

wgt_perf_metrics_explanation = widgets.Textarea(
    value="The selected performance metrics (MSE, RMSE, MAE, and R² Score) are␣
 ↪well-suited for evaluating the baseline regression model: Mean Squared Error␣
 ↪(MSE): Measures the average squared difference between actual and predicted␣
 ↪values. It penalises larger errors more, making it useful for identifying␣
 ↪models with significant deviations. Root Mean Squared Error (RMSE): The␣
 ↪square root of MSE, which provides an error measure in the same unit as the␣
 ↪target variable, making it more interpretable. Mean Absolute Error (MAE):␣
 ↪Represents the average absolute difference between actual and predicted␣
 ↪values. Unlike MSE, it treats all errors equally and is less sensitive to␣
 ↪outliers. These metrics provide a balanced evaluation of prediction␣
 ↪accuracy, sensitivity to outliers, and model fit, making them appropriate␣
 ↪for assessing the Dummy Regressor's baseline performance.",
```

```
    placeholder='<student to fill this section>',
    description='Performance Metrics Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_perf_metrics_explanation
```

```
Textarea(value="The selected performance metrics (MSE, RMSE, MAE, and R² Score)␣
↪are well-suited for evaluating…
```

[57]:
```
print("Performance Metrics Explanation:", wgt_perf_metrics_explanation.value)
```

```
Performance Metrics Explanation: The selected performance metrics (MSE, RMSE,
MAE, and R² Score) are well-suited for evaluating the baseline regression model:
Mean Squared Error (MSE): Measures the average squared difference between actual
and predicted values. It penalises larger errors more, making it useful for
identifying models with significant deviations. Root Mean Squared Error (RMSE):
The square root of MSE, which provides an error measure in the same unit as the
target variable, making it more interpretable. Mean Absolute Error (MAE):
Represents the average absolute difference between actual and predicted values.
Unlike MSE, it treats all errors equally and is less sensitive to outliers.
These metrics provide a balanced evaluation of prediction accuracy, sensitivity
to outliers, and model fit, making them appropriate for assessing the Dummy
Regressor's baseline performance.
```

1.11.3 I.3 Baseline Model Performance

Provide some explanations on model performance

[ ]:
```
# <Student to fill this section>
(y_train_preds - y_train).sum()
```

[ ]:
```
np.float64(-1.255102688446641e-10)
```

[58]:
```
# @title Performance Metrics Explanation

wgt_model_performance_explanation = widgets.Textarea(
    value="The result is very close to zero, meaning the baseline model's␣
↪predictions neither consistently overestimate nor underestimate the actual␣
↪values. Since the Dummy Regressor predicts the mean of y_train, the sum of␣
↪differences should ideally be zero. This confirms that the baseline model␣
↪serves as a simple benchmark, and any improved model should aim for lower␣
↪RMSE and MAE with a higher R² score.",
    placeholder='<student to fill this section>',
    description='Model Performance Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
```

```
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_model_performance_explanation
```

```
Textarea(value="The result is very close to zero, meaning the baseline model's␣
 ↪predictions neither consistentl…
```

[59]:
```
print("Model Performance Explanation:", wgt_model_performance_explanation.value)
```

```
Model Performance Explanation: The result is very close to zero, meaning the
baseline model's predictions neither consistently overestimate nor underestimate
the actual values. Since the Dummy Regressor predicts the mean of y_train, the
sum of differences should ideally be zero. This confirms that the baseline model
serves as a simple benchmark, and any improved model should aim for lower RMSE
and MAE with a higher $R^2$ score.
```

[ ]:
```
# Clear metadata for all experiment notebooks
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
 ↪36106_25AU-AT1_25589351_experiment_0.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
 ↪36106_25AU-AT1_25589351_experiment_1.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
 ↪36106_25AU-AT1_25589351_experiment_2.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
 ↪36106_25AU-AT1_25589351_experiment_3.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
```

```
--inplace

# Convert all notebooks to PDF
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
  ↪36106_25AU-AT1_25589351_experiment_0.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
  ↪36106_25AU-AT1_25589351_experiment_1.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
  ↪36106_25AU-AT1_25589351_experiment_2.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
  ↪36106_25AU-AT1_25589351_experiment_3.ipynb" --to pdf
```