challenge_2

August 22, 2023

[ ]:
```
# The purpose of the exercise is to build a model to estimate the real estate␣
 ↪prices of old apartments in Paris. The final goal is to have a model which,␣
 ↪given a point in Paris (geographical coordinates), can return an estimate,␣
 ↪as precise as possible, of the price per m2. We suggest you implement a␣
 ↪nearest neighbor method.


# To do this, you have two freely accessible databases available:

# The list of real estate transactions carried out on the entire French␣
 ↪territory since 2014: https://www.data.gouv.fr/en/datasets/r/
 ↪90a98de0-f562-4328-aa16-fe0dd1dca60f You will have to work a little on these␣
 ↪data and clean them. To help you, you will find documentation here: https://
 ↪www.data.gouv.fr/en/datasets/r/d573456c-76eb-4276-b91c-e6b9c89d6656
# The list of cadastral parcels in Paris, which should allow you to associate␣
 ↪geographical coordinates to each transaction: https://cadastre.data.gouv.fr/
 ↪data/etalab-cadastre/2021-04-01/shp/departements/75/
 ↪cadastre-75-parcelles-shp.zip


# the following links above to download the needed data into the␣
 ↪challenge_files folder:
```

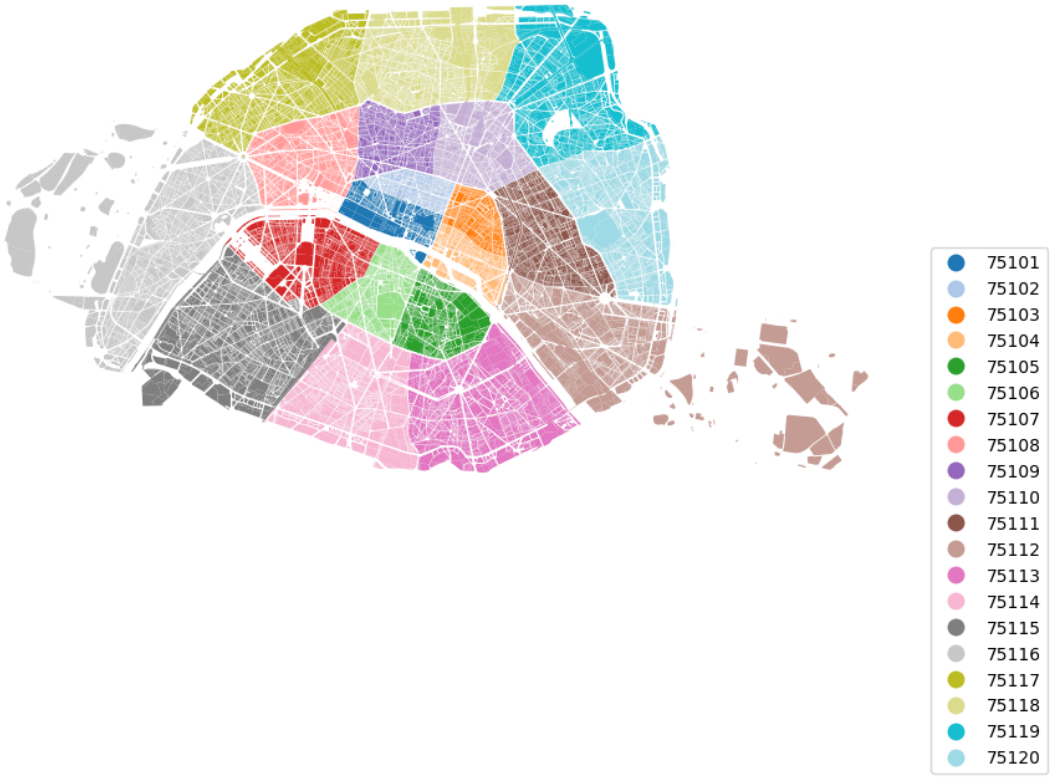
[ ]:
```
# %% importing needed packages.
import pandas as pd
import numpy as np
from numpy.random import seed
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt
import mapclassify as mc
from shapely.ops import unary_union
import seaborn as sns

color = sns.color_palette()
get_ipython().run_line_magic("matplotlib", "inline")
import plotly.offline as py
```

```
py.init_notebook_mode(connected=True)
import plotly.express as px

from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
import category_encoders as ce
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
```

[ ]:
```
# %% loading in the datasets provided and doing some data wrangling.
df = pd.read_table(
    "challenge_files/valeursfoncieres-2020.txt", sep="|", low_memory=False
)
print("Initial dataframe has the shape:", df.shape)
df.drop_duplicates(keep="last", inplace=True)
print("after dropping duplicates, dataframe has the shape:", df.shape)
df.isnull().mean().mul(100).sort_values(ascending=False)
df = df.loc[:, df.isnull().mean() < 1]
num_col = [
    "Surface Carrez du 1er lot",
    "Surface Carrez du 2eme lot",
    "Surface Carrez du 3eme lot",
    "Surface Carrez du 4eme lot",
    "Surface Carrez du 5eme lot",
    "Valeur fonciere",
]
for col in num_col:
    df[col] = df[col].str.replace(",", ".")
    df[col] = df[col].astype(float)

# filtered the data for only Paris related transaction
df = df.loc[(df["Code postal"] > 74999) & (df["Code postal"] < 75991)]
df["Code postal"] = df["Code postal"].astype(int)

df["Code INSEE"] = df["Code departement"].astype(str) + df["Code commune"].
 ↪astype(str)

# created a unique id to merge with the GIS data
df["id"] = (
    df["Code INSEE"].astype(str)
    + "000"
    + df["Section"]
    + "000"
    + df["No plan"].astype(str)
)
```

```
gis_df = gpd.read_file("challenge_files/parcelles.shp")
print("GIS dataframe as the shape:", gis_df.shape)
gis_df.commune = gis_df.commune.astype(int)

# plotting the map of Paris where each district has it's own colour.
fig, ax = plt.subplots(1, figsize=(10, 10))
gis_df.plot(cmap="tab20", column="commune", ax=ax, categorical=True,␣
 ↪legend=True)
leg = ax.get_legend()
leg.set_bbox_to_anchor((1.15, 0.5))
ax.set_axis_off()
plt.show()
```

```
Initial dataframe has the shape: (3514698, 43)
after dropping duplicates, dataframe has the shape: (3371077, 43)
GIS dataframe as the shape: (77720, 9)
```



[ ]:
```
# %% merged the two dataset and found the centroid of each location, the␣
 ↪centroids was used to plot the transaction history across the map of Paris.
```

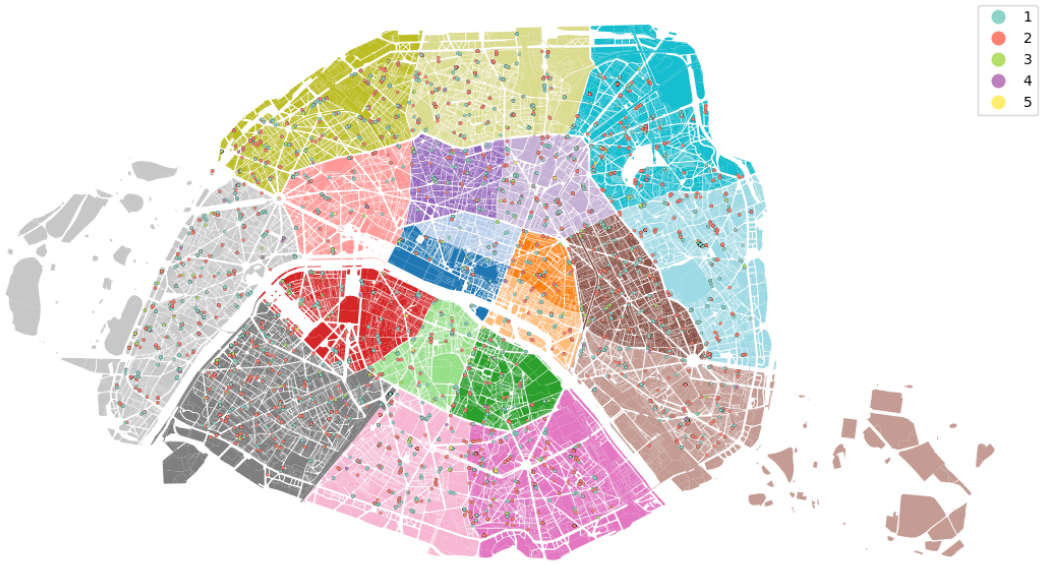
```
# The rational for subseting the data and only working with the rows who's␣
 ↪coordinates map to the gis data is because the aim is to build a model that␣
 ↪predicts price per m2 given coordinates and rows that do not map the gis␣
 ↪data are effectively outside of the scope of this project.
```

[ ]:
```
df2 = gis_df.merge(df, how="right", on="id")
df2 = gpd.GeoDataFrame(df2)

df3 = df2.to_crs(2154)
df2["lon"] = df3.centroid.x
df2["lat"] = df3.centroid.y
print("Merged dataframe between gis df and tabular df as the shape:", df2.shape)
gdf = gpd.GeoDataFrame(df2.copy(), geometry=gpd.points_from_xy(df2.lon, df2.
 ↪lat))

# aggregated the area to one column
gdf["Total Surface Carrez"] = np.nansum(
    gdf[
        [
            "Surface Carrez du 1er lot",
            "Surface Carrez du 2eme lot",
            "Surface Carrez du 3eme lot",
            "Surface Carrez du 4eme lot",
            "Surface Carrez du 5eme lot",
            "Surface terrain",
            "Surface reelle bati",
        ]
    ],
    axis=1,
)

# this done to avoid a zero division error
gdf = gdf.loc[
    (~gdf["Valeur fonciere"].isna())
    & (~gdf.lat.isna())
    & (gdf["Total Surface Carrez"] != 0)
]

gdf.drop(columns=["section", "created", "updated", "numero"], inplace=True)

gdf["price_per_m2"] = gdf["Valeur fonciere"] / gdf["Total Surface Carrez"]

gdf = gdf.loc[:, gdf.isnull().mean() < 1]

# this is dictionary used to check which district a specific coordinate belongs␣
 ↪to.
mul_pl_lst = {}
```

```
for i in list(gdf.commune.unique()):
    polygons = [gis_df.geometry.loc[gis_df["commune"] == i].unique()]
    mul_pl_lst[i] = gpd.GeoSeries(unary_union(polygons[0]))

print("Cleaned dataset as the shape:", gdf.shape)
```

```
Merged dataframe between gis df and tabular df as the shape: (52271, 47)
Cleaned dataset as the shape: (5086, 42)
```

[ ]:
```
# %% this a map of Paris with each point on the map representing a different␣
 ↪type of property.

gdf = gdf.loc[(gdf["Nombre de lots"].isin([1, 2, 3, 4, 5]))]

fig, ax = plt.subplots(1, figsize=(14, 14))
gis_df.plot(cmap="tab20", column="commune", ax=ax, categorical=True,␣
 ↪legend=True)
gdf.plot(
    ax=ax,
    cmap="Set3",
    column="Nombre de lots",
    markersize=5,
    edgecolors="black",
    linewidth=0.1,
    categorical=True,
    legend=True,
)
leg = ax.get_legend()
leg.set_bbox_to_anchor((1, 1))
ax.set_axis_off()
plt.show()

# checked to see if the data spans several years; after filtering and cleaning␣
 ↪we have data only from 2020.
gdf["Date mutation"].str[-4:].sort_values().unique()

gdf_a = gdf.loc[gdf["Code type local"] == 2]
gdf_h = gdf.loc[gdf["Code type local"] == 1]
gdf_c = gdf.loc[gdf["Code type local"] == 4]
gdf_d = gdf.loc[gdf["Code type local"] == 3]
```



[ ]:
```
# %% Performed some EDA to get a better idea of the data characteristics and␣
 ↪distribution.

# Found that most of the transaction in our data is Apartment property. On␣
 ↪average industrial properties cost more per square meter than Apartments or␣
 ↪houses.

# District 15 has the most number of Apartment transaction.
# Majority of the transaction are from district 15 and 16.
```

[ ]:
```
tmp_df = (
    gdf_a.groupby(["Commune"])
    .agg(property_value=("Valeur fonciere", "mean"))
    .sort_values("property_value", ascending=False)
    .reset_index()
)

fig = px.bar(
    tmp_df,
    x="Commune",
    y="property_value",
    title="Mean Property Value in Paris' 20 Districts",
    labels={"property_value": "Mean Property Value"},
    text_auto=".2s",
    color="Commune",
    width=700,
    height=600,
```

```
)
fig.show()

tmp_df = (
    gdf_a.groupby(["Commune", "Nombre de lots"])
    .agg(property_value=("Valeur fonciere", "mean"))
    .sort_values("property_value", ascending=False)
    .reset_index()
)

fig = px.bar(
    tmp_df,
    x="Commune",
    y="property_value",
    title="Mean Property Value in Paris' 20 Districts",
    labels={"property_value": "Mean Property Value"},
    text_auto=".2s",
    color="Nombre de lots",
    barmode="stack",
    width=900,
    height=600,
)
fig.show()

tmp_df = (
    gdf_a.groupby(["Commune"])
    .agg(property_value=("price_per_m2", "mean"))
    .sort_values("property_value", ascending=False)
    .reset_index()
)

fig = px.bar(
    tmp_df,
    x="Commune",
    y="property_value",
    title="Mean Property Value per M^2 in Paris' 20 Districts",
    labels={"property_value": "Mean Property Value per M^2"},
    text_auto=".2s",
    color="Commune",
    barmode="stack",
    width=900,
    height=600,
)
fig.show()

tmp_df = (
    gdf_a.groupby(["Commune", "Nombre de lots"])
```

```
    .agg(property_value=("price_per_m2", "mean"))
    .sort_values("property_value", ascending=False)
    .reset_index()
)

fig = px.bar(
    tmp_df,
    x="Commune",
    y="property_value",
    title="Mean Property Value per M^2 in Paris' 20 Districts",
    labels={"property_value": "Mean Property Value per M^2"},
    text_auto=".2s",
    color="Nombre de lots",
    barmode="stack",
    width=900,
    height=600,
)
fig.show()


tmp_df = (
    gdf_a.groupby(["Type local"])
    .agg(property_count=("Type local", "count"))
    .sort_values("property_count", ascending=False)
    .reset_index()
)

fig = px.bar(
    tmp_df,
    x="Type local",
    y="property_count",
    title="Property Distribution in Paris' 20 Districts",
    labels={"property_count": "Property Count"},
    text_auto=".2s",
    color="Type local",
    width=900,
    height=600,
)
fig.show()

tmp_df = (
    gdf_a.groupby(["Commune"])
    .agg(property_count=("Commune", "count"))
    .sort_values("property_count", ascending=False)
    .reset_index()
)
fig = px.bar(
```

```
    tmp_df,
    x="Commune",
    y="property_count",
    title="Property Distribution in Paris' 20 Districts",
    labels={"property_count": "Property Count"},
    text_auto=".2s",
    width=900,
    height=600,
)
fig.show()
tmp_df = (
    gdf_a.groupby(["Commune", "Nombre de lots"])
    .agg(property_count=("Nombre de lots", "count"))
    .sort_values("property_count", ascending=False)
    .reset_index()
)

fig = px.bar(
    tmp_df,
    x="Commune",
    y="property_count",
    title="Property Distribution in Paris' 20 Districts",
    labels={"property_count": "Property Count"},
    text_auto=".2s",
    color="Nombre de lots",
    barmode="stack",
    width=900,
    height=600,
)
fig.show()
```

[ ]:
```
# %% KNN model build first created the model data using location (lat, lon,␣
 ↪commune) has my only predictor variables; the response variable being␣
 ↪price_per_m2 (price/m2)
cols = [
    "lon",
    "lat",
    "price_per_m2",
]

df_mod = gdf_a[cols].copy()
df_x = df_mod.iloc[:, :-1]

df_y = df_mod["price_per_m2"]

# split the data into train and test (75/25) split and used stratified sampling␣
 ↪to ensure a fair representation of each district within the train set.
```

```
X_train, X_test, y_train, y_test = train_test_split(
    df_x,
    df_y,
    random_state=0,
    test_size=0.20,
)

X_train_sc = X_train
X_test_sc = X_test

# I set seed for reproducibility sake and performed a grid search for the best␣
 ↪hyperparameters for our KNN model.


seed(123)
knnreg = KNeighborsRegressor()
k = np.arange(1, 51)
gr_val = {"n_neighbors": k, "weights": ["uniform", "distance"]}
gr_reg_acc = GridSearchCV(knnreg, param_grid=gr_val, cv=5)
gr_reg_acc.fit(X_train_sc, y_train)

print(f"Grid best parameter (max. accuracy): {gr_reg_acc.best_params_}")
print(f"Grid best score (accuracy): {gr_reg_acc.best_score_}")
print(f"Training R-Squared score: {gr_reg_acc.score(X_train_sc, y_train)}")
print(f"R-squared test score: {gr_reg_acc.score(X_test_sc, y_test)}")
```

```
Grid best parameter (max. accuracy): {'n_neighbors': 12, 'weights': 'uniform'}
Grid best score (accuracy): 0.37445036131644993
Training R-Squared score: 0.3854463424906124
R-squared test score: 0.30512969217699093
```

[ ]:
```
# binning the total area into 9 bins
# Define bin edges and labels
bin_edges = [
    0,
    50,
    100,
    150,
    200,
    250,
    300,
    350,
    500,
    600,
    800,
    1000,
]  # Binning ranges
```

```
bins = list(zip(bin_edges, bin_edges[1:]))
gdf_a["binned_area"] = 0
for i, j in bins:
    gdf_a.loc[
        (gdf_a["Total Surface Carrez"] > i) & (gdf_a["Total Surface Carrez"] <=␣
 ↪j),
        "binned_area",
    ] = j

districts = sorted(list(gdf_a["commune"].sort_values().unique()))
gdf_a["binned_price"] = 0
for dis in districts:
    for i, j in bins:
        gdf_a.loc[
            (gdf_a["Total Surface Carrez"] > i)
            & (gdf_a["Total Surface Carrez"] <= j)
            & (gdf_a["commune"] == dis),
            "binned_price",
        ] = np.mean(
            gdf_a.loc[
                (gdf_a["Total Surface Carrez"] > i)
                & (gdf_a["Total Surface Carrez"] <= j)
                & (gdf_a["commune"] == dis)
            ]["Valeur fonciere"]
        )

gdf_a["binned_price_m2"] = gdf_a["binned_price"] / gdf_a["binned_area"]
```

```
c:\Users\Tolu\miniconda3\envs\geo_pred\lib\site-
packages\geopandas\geodataframe.py:1538: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

c:\Users\Tolu\miniconda3\envs\geo_pred\lib\site-
packages\geopandas\geodataframe.py:1538: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
c:\Users\Tolu\miniconda3\envs\geo_pred\lib\site-
packages\geopandas\geodataframe.py:1538: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[ ]:
```python
# %% scoring the binned price per area
cols = [
    "lon",
    "lat",
    "binned_price_m2",
]

df_mod = gdf_a[cols].copy()
df_x = df_mod.iloc[:, :-1]
df_y = df_mod["binned_price_m2"]

X_train, X_test, y_train, y_test = train_test_split(
    df_x,
    df_y,
    random_state=0,
    test_size=0.20,
)

X_train_sc = X_train
X_test_sc = X_test

seed(123)
knnreg = KNeighborsRegressor()
k = np.arange(1, 51)
gr_val = {"n_neighbors": k, "weights": ["uniform", "distance"]}
gr_reg_acc = GridSearchCV(knnreg, param_grid=gr_val, cv=5)
gr_reg_acc.fit(X_train_sc, y_train)

print(f"Grid best parameter (max. accuracy): {gr_reg_acc.best_params_}")
print(f"Grid best score (accuracy): {gr_reg_acc.best_score_}")
print(f"Training R-Squared score: {gr_reg_acc.score(X_train_sc, y_train)}")
print(f"R-squared test score: {gr_reg_acc.score(X_test_sc, y_test)}")
```

```
Grid best parameter (max. accuracy): {'n_neighbors': 23, 'weights': 'uniform'}
Grid best score (accuracy): 0.5149746661922995
Training R-Squared score: 0.5554420366838108
R-squared test score: 0.5229879013557727
```

[ ]:
```
# %% scaled lat and lon with different distance metric and binned price per␣
 ↪area predictor
cols = [
    "lon",
    "lat",
    "binned_price_m2",
]

df_mod = gdf_a[cols].copy()
df_x = df_mod.iloc[:, :-1]
df_y = df_mod["binned_price_m2"]

X_train, X_test, y_train, y_test = train_test_split(
    df_x,
    df_y,
    random_state=0,
    test_size=0.20,
)

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

seed(123)
knnreg = KNeighborsRegressor()
k = np.arange(1, 51)
gr_val = {
    "n_neighbors": k,
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "haversine", "chebyshev", "manhattan", "minkowski"],
}
gr_reg_acc = GridSearchCV(knnreg, param_grid=gr_val, cv=5)
gr_reg_acc.fit(X_train_scaled, y_train)

print(f"Grid best parameter (max. accuracy): {gr_reg_acc.best_params_}")
print(f"Grid best score (accuracy): {gr_reg_acc.best_score_}")
print(f"Training R-Squared score: {gr_reg_acc.score(X_train_sc, y_train)}")
print(f"R-squared test score: {gr_reg_acc.score(X_test_scaled, y_test)}")
# Calculate Mean Squared Error (MSE) and R-squared
y_pred = gr_reg_acc.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

############################### scaled the lat and lon and tried different␣
 ↪distance metrics
cols = [
    "lon",
```

```
    "lat",
    "price_per_m2",
]

df_mod = gdf_a[cols].copy()
df_x = df_mod.iloc[:, :-1]
df_y = df_mod["price_per_m2"]

X_train, X_test, y_train, y_test = train_test_split(
    df_x,
    df_y,
    random_state=0,
    test_size=0.20,
)

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

seed(123)
knnreg = KNeighborsRegressor()
k = np.arange(1, 51)
gr_val = {
    "n_neighbors": k,
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "haversine", "chebyshev", "manhattan", "minkowski"],
}
gr_reg_acc = GridSearchCV(knnreg, param_grid=gr_val, cv=5)
gr_reg_acc.fit(X_train_scaled, y_train)

print(f"Grid best parameter (max. accuracy): {gr_reg_acc.best_params_}")
print(f"Grid best score (accuracy): {gr_reg_acc.best_score_}")
print(f"Training R-Squared score: {gr_reg_acc.score(X_train_sc, y_train)}")
print(f"R-squared test score: {gr_reg_acc.score(X_test_scaled, y_test)}")
# Calculate Mean Squared Error (MSE) and R-squared
y_pred = gr_reg_acc.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Grid best parameter (max. accuracy): {'metric': 'haversine', 'n_neighbors': 17,
'weights': 'uniform'}
Grid best score (accuracy): 0.513932382701431
Training R-Squared score: -0.36723622814958223
R-squared test score: 0.45779175195435695
Mean Squared Error: 31495976.75329382
```

```
c:\Users\Tolu\miniconda3\envs\geo_pred\lib\site-packages\sklearn\base.py:457:
UserWarning:
```

```
X has feature names, but KNeighborsRegressor was fitted without feature names


Grid best parameter (max. accuracy): {'metric': 'manhattan', 'n_neighbors': 10,
'weights': 'uniform'}
Grid best score (accuracy): 0.4008521836038527
Training R-Squared score: -0.036358050858926294
R-squared test score: 0.3269561453627424
Mean Squared Error: 784783622.6232053

c:\Users\Tolu\miniconda3\envs\geo_pred\lib\site-packages\sklearn\base.py:457:
UserWarning:

X has feature names, but KNeighborsRegressor was fitted without feature names
```

[ ]:
```
# %%
# Using binning we saw improvements from the initial model performance from:
# Grid best parameter (max. accuracy): {'n_neighbors': 12, 'weights': 'uniform'}
# Grid best score (accuracy): 0.37445036131644993
# Training R-Squared score: 0.385446342490612 4
# R-squared test score: 0.30512969217699093

# to:
# Grid best parameter (max. accuracy): {'n_neighbors': 23, 'weights': 'uniform'}
# Grid best score (accuracy): 0.5149746661922995
# Training R-Squared score: 0.5554420366838108
# R-squared test score: 0.5229879013557727

# overall performance is not ideal, however, this is an improvement from the␣
 ↪initial model.



# Next approach was to scale the model and try a different distance metric,␣
 ↪however, that did not improve the model and in fact the performance worsen.␣
 ↪Overall binning the total area and taking the average price per district␣
 ↪seemed to yield the best solution.

#
# Clearly the location features alone are not enough to determine the price per␣
 ↪square meter. I considered features like number of rooms and property type␣
 ↪however those only degraded the model further.
#
# Ways to improve it, I believe the location data maybe too granular. I think␣
 ↪given more time, and further data wrangling I could aggregate the data to a␣
 ↪district level and use district as my location variable rather than␣
 ↪coordinates.
```

```
#
# There could be presents of outliers that is degrading the model, for example␣
 ↪district 2, district with the largest mean property, has a mean property␣
 ↪value larger than the next 3 districts. Yet it only accounts for 53␣
 ↪transactions of out over 5k rows.


# Another reason for this poor model performance is that we are trying to␣
 ↪predict an exact price. I believe a better approach is to find the average␣
 ↪price per binned m2.
```

[ ]:
```
# %% This function effectively takes in lat and lon coordinates and predicts␣
 ↪the binned price per m2.
def price_per_m2(
    lon,
    lat,
):
    temp = pd.DataFrame({"lon": lon, "lat": lat}, index=[0])
    features = [[lon, lat]]
    print(
        f"Based on the coordinates {features[0]}, the approximate price per␣
 ↪square meter in that area is:  {gr_reg_acc.predict(temp)}"
    )


price_per_m2(652455.920961, 6.862873e06)
```

```
Based on the coordinates [652455.920961, 6862873.0], the approximate price per
square meter in that area is:  [4565.63022523]

c:\Users\Tolu\miniconda3\envs\geo_pred\lib\site-packages\sklearn\base.py:457:
UserWarning:

X has feature names, but KNeighborsRegressor was fitted without feature names
```

[ ]:
```

```