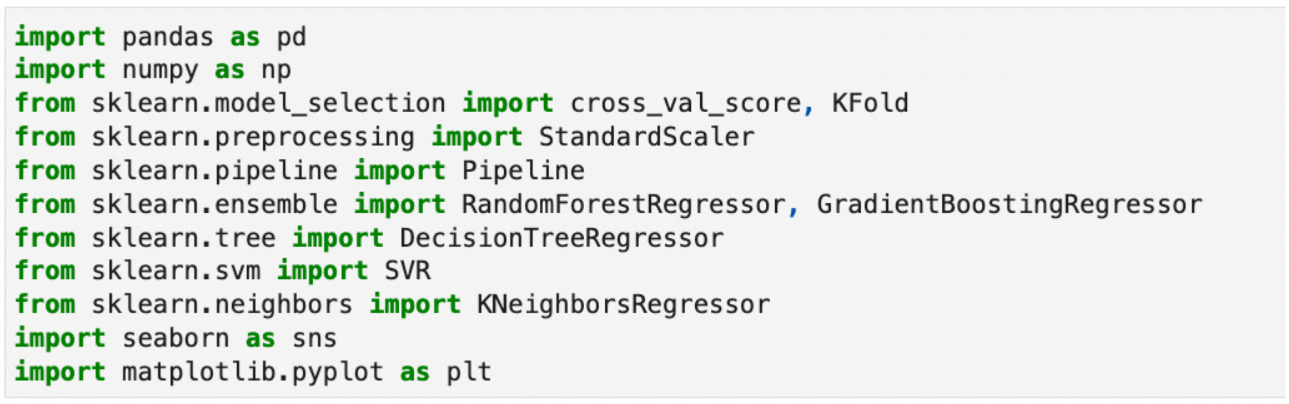
Data Preprocessing Documentation

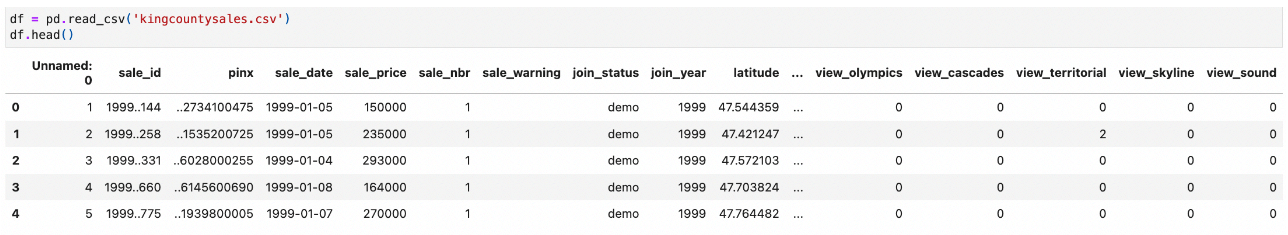
Objective

The goal of this preprocessing pipeline was to prepare a large real estate dataset (≈560k rows) for machine learning regression tasks by cleaning, transforming, and reducing features while preserving predictive power.

Data Cleaning



```python
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
import seaborn as sns
import matplotlib.pyplot as plt
```

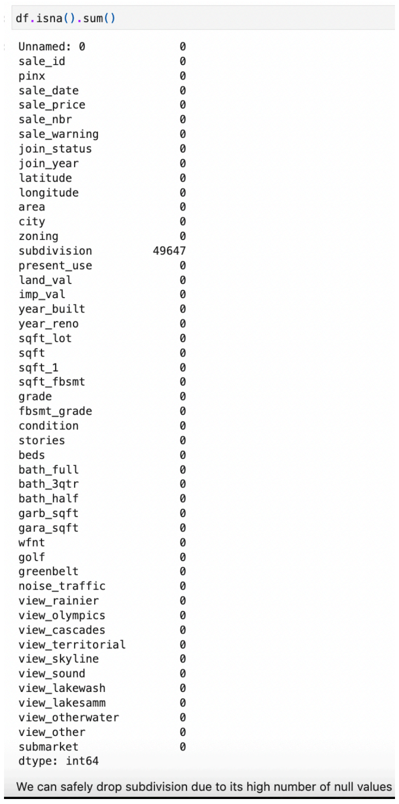


```python
df = pd.read_csv('kingcountysales.csv')
df.head()
```

	Unnamed: 0	sale_id	pinx	sale_date	sale_price	sale_nbr	sale_warning	join_status	join_year	latitude	...	view_olympics	view_cascades	view_territorial	view_skyline	view_sound
0	1	1999..144	..2734100475	1999-01-05	150000	1		demo	1999	47.544359	...	0	0	0	0	0
1	2	1999..258	..1535200725	1999-01-05	235000	1		demo	1999	47.421247	...	0	0	2	0	0
2	3	1999..331	..6028000255	1999-01-04	293000	1		demo	1999	47.572103	...	0	0	0	0	0
3	4	1999..660	..6145600690	1999-01-08	164000	1		demo	1999	47.703824	...	0	0	0	0	0
4	5	1999..775	..1939800005	1999-01-07	270000	1		demo	1999	47.764482	...	0	0	0	0	0

Above Image: Imports and Data loading snippet

Checked for null values:



```python
df.isna().sum()
```

```
Unnamed: 0            0
sale_id               0
pinx                  0
sale_date             0
sale_price            0
sale_nbr              0
sale_warning          0
join_status           0
join_year             0
latitude              0
longitude             0
area                  0
city                  0
zoning                0
subdivision       49647
present_use           0
land_val              0
imp_val               0
year_built            0
year_reno             0
sqft_lot              0
sqft                  0
sqft_1                0
sqft_fbsmt            0
grade                 0
fbsmt_grade           0
condition             0
stories               0
beds                  0
bath_full             0
bath_3qtr             0
bath_half             0
garb_sqft             0
gara_sqft             0
wfnt                  0
golf                  0
greenbelt             0
noise_traffic         0
view_rainier          0
view_olympics         0
view_cascades         0
view_territorial      0
view_skyline          0
view_sound            0
view_lakewash         0
view_lakesamm         0
view_otherwater       0
view_other            0
submarket             0
dtype: int64
```

We can safely drop subdivision due to its high number of null values

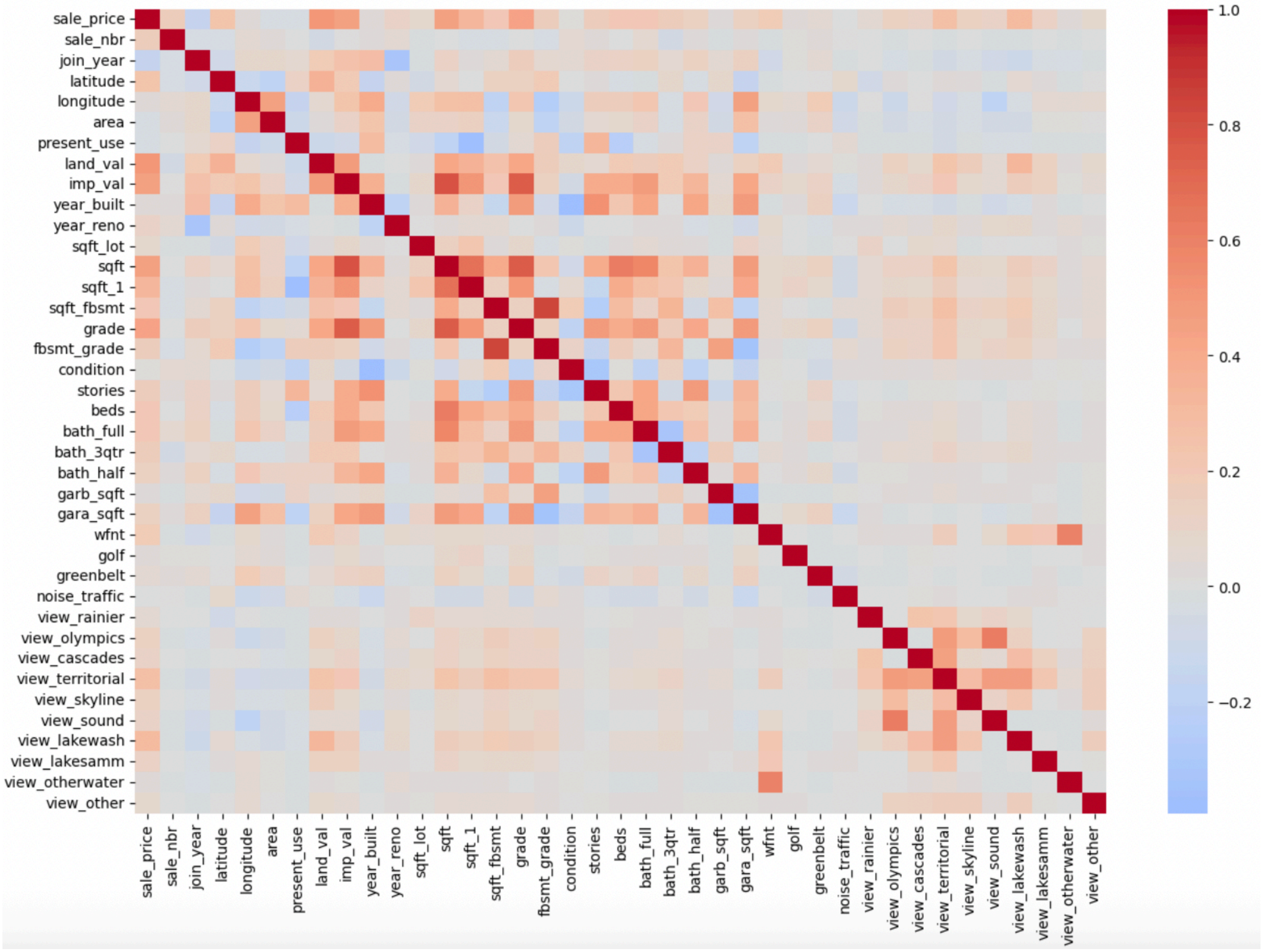
Dropped irrelevant columns:
Removed Unnamed: 0, subdivision, for its high number of null values.

Feature Selection & Reduction

Used correlation to figure out multi corrlinearity within the dataset:

```
corr_matrix = df.corr(numeric_only=True)

plt.figure(figsize=(15,10))
sns.heatmap(corr_matrix, cmap="coolwarm", center=0)
plt.show()
```



The proceeded to drop those with high correlation relation , >= 0.8 and less than 1.

```
high_corr = corr_matrix[(corr_matrix >= 0.8) & (corr_matrix <1.0)]
print(high_corr.dropna(how="all", axis=0).dropna(how="all", axis=1))
```

```
             sqft_fbsmt  fbsmt_grade
sqft_fbsmt          NaN     0.831937
fbsmt_grade    0.831937          NaN
```

```
corr_matrix = df.corr(numeric_only=True).abs()

upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

to_drop = [column for column in upper.columns if any(upper[column] >= 0.8)]

print("Columns to drop due to high correlation (>0.6):")
print(to_drop)
df = df.drop(columns=to_drop)
print(f"Shape before: {df.shape}, after dropping: {df_reduced.shape}")
```

```
Columns to drop due to high correlation (>0.6):
['fbsmt_grade']
Shape before: (560219, 46), after dropping: (560219, 40)
```

Feature Transformation

Figured out skewness of the dataset to log transform highly skewed values
Target transformation:
1. Converted target variable sale_price → sale_price_log using np.log1p() to stabilize variance and reduce skewness.
2.Numeric feature transformation
3.Applied log transformation to skewed numeric variables to approximate normality.
4.Dropped longitude-log because of high values of null values.

```
skewed_cols = df.select_dtypes(include=np.number).columns
skew_values = df[skewed_cols].skew()
high_skew_cols = skew_values[abs(skew_values) > 0.5].index.tolist()

print("Columns with skewness beyond ±0.5:")
print(high_skew_cols)

for col in high_skew_cols:
    df[col + "_log"] = np.log1p(df[col])

df = df.drop(columns=high_skew_cols)

print("Shape after transforming and dropping skewed columns:", df.shape)
```

Dropping irrelevant Columns

```
corr_with_target = df.corr(numeric_only = True)['sale_price_log'].sort_values(ascending=False)
print(corr_with_target)
```

```
sale_price_log          1.000000
grade_log               0.517509
sqft_log                0.489694
latitude                0.331384
land_val_log            0.293215
sqft_1_log              0.274055
stories                 0.261577
sale_nbr_log            0.256255
bath_full_log           0.248442
beds_log                0.246922
view_territorial_log    0.246631
view_lakewash_log       0.223811
imp_val_log             0.219523
bath_3qtr_log           0.183970
bath_half               0.172055
sqft_fbsmt_log          0.158464
view_olympics_log       0.141324
wfnt_log                0.128259
view_sound_log          0.120097
view_skyline_log        0.114431
year_reno_log           0.109963
sqft_lot_log            0.109034
view_cascades_log       0.108603
view_lakesamm_log       0.104893
year_built_log          0.097130
greenbelt_log           0.073889
view_other_log          0.072121
gara_sqft_log           0.066246
view_rainier_log        0.046074
golf_log                0.045608
view_otherwater_log     0.036360
garb_sqft_log           0.022216
condition_log          -0.002574
present_use_log        -0.003072
noise_traffic_log      -0.012530
area                   -0.019054
join_year_log          -0.115564
longitude_log                NaN
Name: sale_price_log, dtype: float64
```

```
irrelevant_cols = corr_with_target[abs(corr_with_target) < 0.1].index.tolist()
print(irrelevant_cols)
df = df.drop(columns=irrelevant_cols)
```

```
['year_built_log', 'greenbelt_log', 'view_other_log', 'gara_sqft_log', 'view_rainier_log', 'golf_log', 'view_otherwater_log', 'garb_sqft_log', 'condition_log', 'present_use_log', 'noise_traffic_log', 'area']
```

Finded correlated columns in respect to the target and then proceed to drop columns who had correlation smaller than 0.1

Final Dataset

The final dataset was saved as a separate csv file to further compare the difference in accuracy between linear and tree models and the impact of the data processing. The columns were reduced to 33.

```
df.to_csv('processed_data.csv', index=False)
```

```
df.shape
```

```
(560219, 33)
```