5_FeatureEngineering

February 13, 2026

1 Ironkaggle

[31]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
import plotly.express as px
from sklearn.linear_model import LinearRegression, Ridge, Lasso
import statsmodels.api as sm
```

1.0.1 Load Data - Show basics

[2]:
```
dataset_csv= pd.read_csv("king_ country_ houses_aa.csv")
dataset = dataset_csv.copy()
```

2 Functions used for Analysis

[3]:
```
### Functions - Taken from Anne's Utils.py

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import (r2_score,
                             mean_absolute_error,
                             mean_absolute_percentage_error,
                             mean_squared_error,
```

```
                                        root_mean_squared_error)


from sklearn.base import BaseEstimator
from typing import SupportsFloat, Union, Dict

# Define types for clearer code
ArrayLike = Union[pd.DataFrame, pd.Series, np.ndarray]


def adjusted_r2(y_true: ArrayLike, y_pred: ArrayLike, X: ArrayLike) -> float:
    """
    Calculate Adjusted R2, handling both Arrays and DataFrames.
    """
    r2 = r2_score(y_true, y_pred)
    n = len(y_true)

    # Handle X shape safely (whether it's a DataFrame or Numpy array)
    if hasattr(X, 'shape'):
        p = X.shape[1] if len(X.shape) > 1 else 1
    else:
        p = 1 # Fallback for 1D lists

    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

def create_metrics_df():
    """Create the dataframe to store the metrics

    Returns:
        pd.DataFrame: The empty metrics DataFrame.
    """
    columns = ["Model","Split", "R2", "Adjusted_R2", "MAE", "RMSE",␣
 ↪"MAPE","Comments"]
    metrics_df = pd.DataFrame(columns=columns)
    return metrics_df

def _get_metrics(
    trained_model: BaseEstimator,
    X: ArrayLike,
    y: ArrayLike,
    split: str = "train",
    comments: str = "Baseline model"
) -> Dict[str, Union[str, float]]:
    """
    Internal function to calculate metrics for a single split.

    Args:
```

```python
        trained_model: A fitted sklearn model.
        X: Feature matrix (DataFrame or Numpy Array).
        y: Target vector (Series or Numpy Array).
        split: Label for the data split (e.g., 'Train', 'Test').
        comments: User notes.

    Returns:
        dict: A dictionary containing all calculated metrics.
    """
    # Generate predictions
    y_pred = trained_model.predict(X)

    # Calculate metrics
    new_row = {
        "Model": trained_model.__class__.__name__,
        "Split": split,
        "R2": np.round(r2_score(y, y_pred), 4),
        "Adjusted_R2": np.round(adjusted_r2(y, y_pred, X), 4),
        "MAE": np.round(mean_absolute_error(y, y_pred), 4),
        "MAPE": np.round(mean_absolute_percentage_error(y, y_pred), 4),
        "RMSE": np.round(root_mean_squared_error(y, y_pred), 4),
        "Comments": comments
    }

    return new_row

def add_new_metrics(
    metrics_df: pd.DataFrame,
    trained_model: BaseEstimator,
    X: ArrayLike,
    y: ArrayLike,
    split: str = "train",
    comments: str = "Baseline model"
) -> pd.DataFrame:
    """
    Calculates metrics and appends them to the tracking DataFrame.

    Args:
        metrics_df: The existing DataFrame to update.
        trained_model: A fitted sklearn model.
        X: Feature matrix.
        y: Target vector.
        split: "Train" or "Test".
        comments: Notes about this run.

    Returns:
        pd.DataFrame: The updated DataFrame with the new row.
```

```
    """

    # Get the metrics dictionary
    new_row_dict = _get_metrics(trained_model, X, y, split, comments)

    # Create a DataFrame from the new row
    new_row_df = pd.DataFrame([new_row_dict])

    # Concatenate and RETURN the result
    if metrics_df.empty:
        return new_row_df
    else:
        updated_df = pd.concat([metrics_df, new_row_df], ignore_index=True)

    return updated_df

def generate_heatmap(X):
    # 1. Calculate correlation
    corr = np.round(np.abs(X.corr()), 2)

    # 2. Create the mask (True for upper triangle and diagonal)
    mask = np.zeros_like(corr, dtype=bool)
    mask[np.triu_indices_from(mask)] = True

    # 3. Slice the matrix and mask to remove the completely empty row/column
    #    Row 0 is fully masked (hidden), Column -1 is fully masked (hidden)
    corr_sliced = corr.iloc[1:, :-1]
    mask_sliced = mask[1:, :-1]

    f, ax = plt.subplots(figsize=(16, 16))

    # 4. Plot using the sliced data
    #    Note: annot=True is safer than annot=corr when slicing
    sns.heatmap(
        corr_sliced,
        mask=mask_sliced,
        annot=True,           # Use True to automatically label values
        square=True,
        linewidths=.5,
        vmax=1
    )
    plt.show()


    from sklearn.metrics import r2_score, mean_squared_error,␣
 ↪mean_absolute_error
```

```
def get_r_squared(y_train, y_test, y_pred_train, y_pred_test):
    """Get the r2 for train and test and print the values.

    Args:
        y_train (_type_): Ground Truth target values for the train.
        y_test (_type_): Ground Truth target values for the test.
        y_pred_train (_type_): Predicted values for the train.
        y_pred_test (_type_): Predicted values for the test.

    Returns:
        tuple: tuple containing the r2 for the train and the test in that order.
    """
    r2_train = r2_score(y_train, y_pred_train)
    r2_test = r2_score(y_test, y_pred_test)

    print_metrics("R2", r2_train, r2_test)
    return (r2_train, r2_test)

def get_mse(y_train, y_test, y_pred_train, y_pred_test):
    """Get the mse for train and test and print the values.

    Args:
        y_train (_type_): Ground Truth target values for the train.
        y_test (_type_): Ground Truth target values for the test.
        y_pred_train (_type_): Predicted values for the train.
        y_pred_test (_type_): Predicted values for the test.

    Returns:
        tuple: tuple containing the mse for the train and the test in that␣
 ↪order.
    """
    mse_train = mean_squared_error(y_train, y_pred_train)
    mse_test = mean_squared_error(y_test, y_pred_test)

    print_metrics("MSE", mse_train, mse_test)
    return (mse_train, mse_test)

def get_mae(y_train, y_test, y_pred_train, y_pred_test):
    """Get the mae for train and test and print the values.

    Args:
        y_train (_type_): Ground Truth target values for the train.
        y_test (_type_): Ground Truth target values for the test.
        y_pred_train (_type_): Predicted values for the train.
        y_pred_test (_type_): Predicted values for the test.

    Returns:
```

```
        tuple: tuple containing the mae for the train and the test in that␣
↪order.
    """
    mae_train = mean_absolute_error(y_train, y_pred_train)
    mae_test = mean_absolute_error(y_test, y_pred_test)

    print_metrics("MSE", mae_train, mae_test)
    return (mae_train, mae_test)

def print_metrics(metric_name:str, train_score:SupportsFloat, test_score:
↪SupportsFloat):
    string = f"""
    {metric_name} score:
    train | {train_score}
    test  | {test_score}
    """

    print(string)
```

3 Basic Data Inspection

[4]:
```
# Basic Data Inspection

# Check Shape of the Dataset
print(f"Dataset Shape : {dataset.shape[0]} rows x {dataset.shape[1]}")

# Check Column Names of the Dataset
print(f"\nColumn Names : {dataset.columns.tolist()}")

# Temp Dataset
dataset_alt = dataset.copy()

# Missing Values Check
print("\nCheck for possible Null Values:")
missingVal = dataset.isnull().sum()
print(missingVal)
if missingVal.sum() == 0:
    print("NO Missing Values found")


# Check for duplicates
print(f"\nChecking for Dupicates :")
duplicatesVal = dataset.duplicated().sum()
if duplicatesVal == 0:
    print(f"No Duplicates found")
```

```
Dataset Shape : 21613 rows x 21
```

```
Column Names : ['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
'sqft_living15', 'sqft_lot15']

Check for possible Null Values:
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
NO Missing Values found

Checking for Dupicates :
No Duplicates found
```

4 Feature Engineering

Proposal

- Convert Date Colummn to Pandas Data-Time series and extract 3 new columns : day,month and year.
- Create 2 new columns called : Year_Since_Renovation and Was_Renovated(booolean)

[5]:
```
# Convert Date Column to Day,month and Year
dataset_alt.date = pd.to_datetime(dataset_alt.date)
dataset_alt["year_sold"] = dataset_alt.date.dt.year
dataset_alt["month_sold"] = dataset_alt.date.dt.month
dataset_alt["day_sold"] = dataset_alt.date.dt.day
```

```python
# Selected features for FEATURE ENGINEERING
dataset_alt['yr_built'] = pd.to_numeric(dataset_alt['yr_built'],␣
↪errors='coerce')
dataset_alt['yr_renovated'] = pd.to_numeric(dataset_alt['yr_renovated'],␣
↪errors='coerce')
dataset_alt['year_sold'] = pd.to_numeric(dataset_alt['year_sold'],␣
↪errors='coerce')

# Add Columns : Age at Sale, Year since renovation, Was Renovated(bool) to the␣
↪Dataset
dataset_alt["Age_at_Sale"] = dataset_alt["year_sold"] - dataset_alt["yr_built"]

dataset_alt["Year_since_Renovation"] = dataset_alt["year_sold"] -␣
↪dataset_alt["yr_renovated"]

dataset_alt.loc[dataset_alt["yr_renovated"] == 0, "Year_since_Renovation"] =␣
↪dataset_alt["year_sold"] - dataset_alt["yr_built"]

print(f"\nColumn Names : {dataset_alt.columns.tolist()}")

dataset_alt["was_renovated"] = (dataset_alt["yr_renovated"] > 0) &␣
↪(dataset_alt["yr_renovated"] > dataset_alt["yr_built"])

print(f"\nData Type of columns :\n{dataset_alt.dtypes}")

Cleaned_Dataset = dataset_alt.drop(columns = ["date", "id", "yr_built",␣
↪"year_sold", "month_sold" ]) # yr_built, year_sold
Cleaned_Dataset.info()
print(f"\nColumn Names : {Cleaned_Dataset.columns.tolist()}")
Cleaned_Dataset.to_csv("king_country_houses_fe.csv", index=False)
```

```
Column Names : ['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
'sqft_living15', 'sqft_lot15', 'year_sold', 'month_sold', 'day_sold',
'Age_at_Sale', 'Year_since_Renovation']

Data Type of columns :
id                                int64
date                     datetime64[ns]
price                           float64
bedrooms                          int64
bathrooms                       float64
sqft_living                       int64
sqft_lot                          int64
floors                          float64
```

```
waterfront                    int64
view                          int64
condition                     int64
grade                         int64
sqft_above                    int64
sqft_basement                 int64
yr_built                      int64
yr_renovated                  int64
zipcode                       int64
lat                         float64
long                        float64
sqft_living15                 int64
sqft_lot15                    int64
year_sold                     int32
month_sold                    int32
day_sold                      int32
Age_at_Sale                   int64
Year_since_Renovation         int64
was_renovated                  bool
dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   price                  21613 non-null  float64
 1   bedrooms               21613 non-null  int64
 2   bathrooms              21613 non-null  float64
 3   sqft_living            21613 non-null  int64
 4   sqft_lot               21613 non-null  int64
 5   floors                 21613 non-null  float64
 6   waterfront             21613 non-null  int64
 7   view                   21613 non-null  int64
 8   condition              21613 non-null  int64
 9   grade                  21613 non-null  int64
 10  sqft_above             21613 non-null  int64
 11  sqft_basement          21613 non-null  int64
 12  yr_renovated           21613 non-null  int64
 13  zipcode                21613 non-null  int64
 14  lat                    21613 non-null  float64
 15  long                   21613 non-null  float64
 16  sqft_living15          21613 non-null  int64
 17  sqft_lot15             21613 non-null  int64
 18  day_sold               21613 non-null  int32
 19  Age_at_Sale            21613 non-null  int64
 20  Year_since_Renovation  21613 non-null  int64
 21  was_renovated          21613 non-null  bool
dtypes: bool(1), float64(5), int32(1), int64(15)
```

```
memory usage: 3.4 MB

Column Names : ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
'sqft_basement', 'yr_renovated', 'zipcode', 'lat', 'long', 'sqft_living15',
'sqft_lot15', 'day_sold', 'Age_at_Sale', 'Year_since_Renovation',
'was_renovated']
```

5 Split Data for train_test_split

- Here we split the Data :
  - X - Holds the Cleaned Dataset without PRICE
  - Y - Holds the Cleaned Dataset with only PRICE

[6]:
```
X = Cleaned_Dataset.drop(columns = ["price"])
y = Cleaned_Dataset["price"]
seed = 13

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=seed)

print(f"X_train Training Samples: {X_train.shape[0]}")
print(f"X_test Samples: {X_test.shape[0]}")
print(f"Y_train Training Samples: {y_train.shape[0]}")
print(f"Y_test Training Samples: {y_test.shape[0]}")


print(f"Split: {len(X_train)} train ({len(X_train)/len(Cleaned_Dataset):.1%}) |␣
 ↪"
      f"{len(X_test)} test ({len(X_test)/len(Cleaned_Dataset):.1%})")
```

```
X_train Training Samples: 17290
X_test Samples: 4323
Y_train Training Samples: 17290
Y_test Training Samples: 4323
Split: 17290 train (80.0%) | 4323 test (20.0%)
```

6 Correlation

```
- Here a simple correlation without controlling any parameters is done.
- And consquently a Heatmap
```

[7]:
```
check_correlation = Cleaned_Dataset.corr()
print("\nCorrelation Analysis :")
print(check_correlation.round(3))

# Plot correlation Heatmap
```

```python
plt.figure(figsize=(15,15))
sns.heatmap(check_correlation,annot=True, cmap='coolwarm',center=0,fmt='.
 ↪2f',square=True,linewidths=1)
plt.title("Housing Dataset Correlation Heatmap", fontsize=14,fontweight='bold')
plt.tight_layout()
plt.show()
plt.figure()
```

```
Correlation Analysis :
                       price  bedrooms  bathrooms  sqft_living  sqft_lot  \
price                  1.000     0.308      0.525        0.702     0.090
bedrooms               0.308     1.000      0.516        0.577     0.032
bathrooms              0.525     0.516      1.000        0.755     0.088
sqft_living            0.702     0.577      0.755        1.000     0.173
sqft_lot               0.090     0.032      0.088        0.173     1.000
floors                 0.257     0.175      0.501        0.354    -0.005
waterfront             0.266    -0.007      0.064        0.104     0.022
view                   0.397     0.080      0.188        0.285     0.075
condition              0.036     0.028     -0.125       -0.059    -0.009
grade                  0.667     0.357      0.665        0.763     0.114
sqft_above             0.606     0.478      0.685        0.877     0.184
sqft_basement          0.324     0.303      0.284        0.435     0.015
yr_renovated           0.126     0.019      0.051        0.055     0.008
zipcode               -0.053    -0.153     -0.204       -0.199    -0.130
lat                    0.307    -0.009      0.025        0.053    -0.086
long                   0.022     0.129      0.223        0.240     0.230
sqft_living15          0.585     0.392      0.569        0.756     0.145
sqft_lot15             0.082     0.029      0.087        0.183     0.719
day_sold              -0.015    -0.008     -0.005       -0.007     0.001
Age_at_Sale           -0.054    -0.154     -0.506       -0.318    -0.053
Year_since_Renovation -0.106    -0.166     -0.537       -0.344    -0.053
was_renovated          0.126     0.019      0.050        0.055     0.008

                       floors  waterfront   view  condition  grade  …  \
price                   0.257       0.266  0.397      0.036  0.667  …
bedrooms                0.175      -0.007  0.080      0.028  0.357  …
bathrooms               0.501       0.064  0.188     -0.125  0.665  …
sqft_living             0.354       0.104  0.285     -0.059  0.763  …
sqft_lot               -0.005       0.022  0.075     -0.009  0.114  …
floors                  1.000       0.024  0.029     -0.264  0.458  …
waterfront              0.024       1.000  0.402      0.017  0.083  …
view                    0.029       0.402  1.000      0.046  0.251  …
condition              -0.264       0.017  0.046      1.000 -0.145  …
grade                   0.458       0.083  0.251     -0.145  1.000  …
sqft_above              0.524       0.072  0.168     -0.158  0.756  …
sqft_basement          -0.246       0.081  0.277      0.174  0.168  …
```

yr_renovated	0.006	0.093	0.104	-0.061	0.014	…
zipcode	-0.059	0.030	0.085	0.003	-0.185	…
lat	0.050	-0.014	0.006	-0.015	0.114	…
long	0.125	-0.042	-0.078	-0.107	0.198	…
sqft_living15	0.280	0.086	0.280	-0.093	0.713	…
sqft_lot15	-0.011	0.031	0.073	-0.003	0.119	…
day_sold	-0.007	0.011	0.011	-0.005	-0.012	…
Age_at_Sale	-0.490	0.026	0.053	0.361	-0.447	…
Year_since_Renovation	-0.506	0.000	0.018	0.396	-0.461	…
was_renovated	0.006	0.093	0.104	-0.060	0.014	…

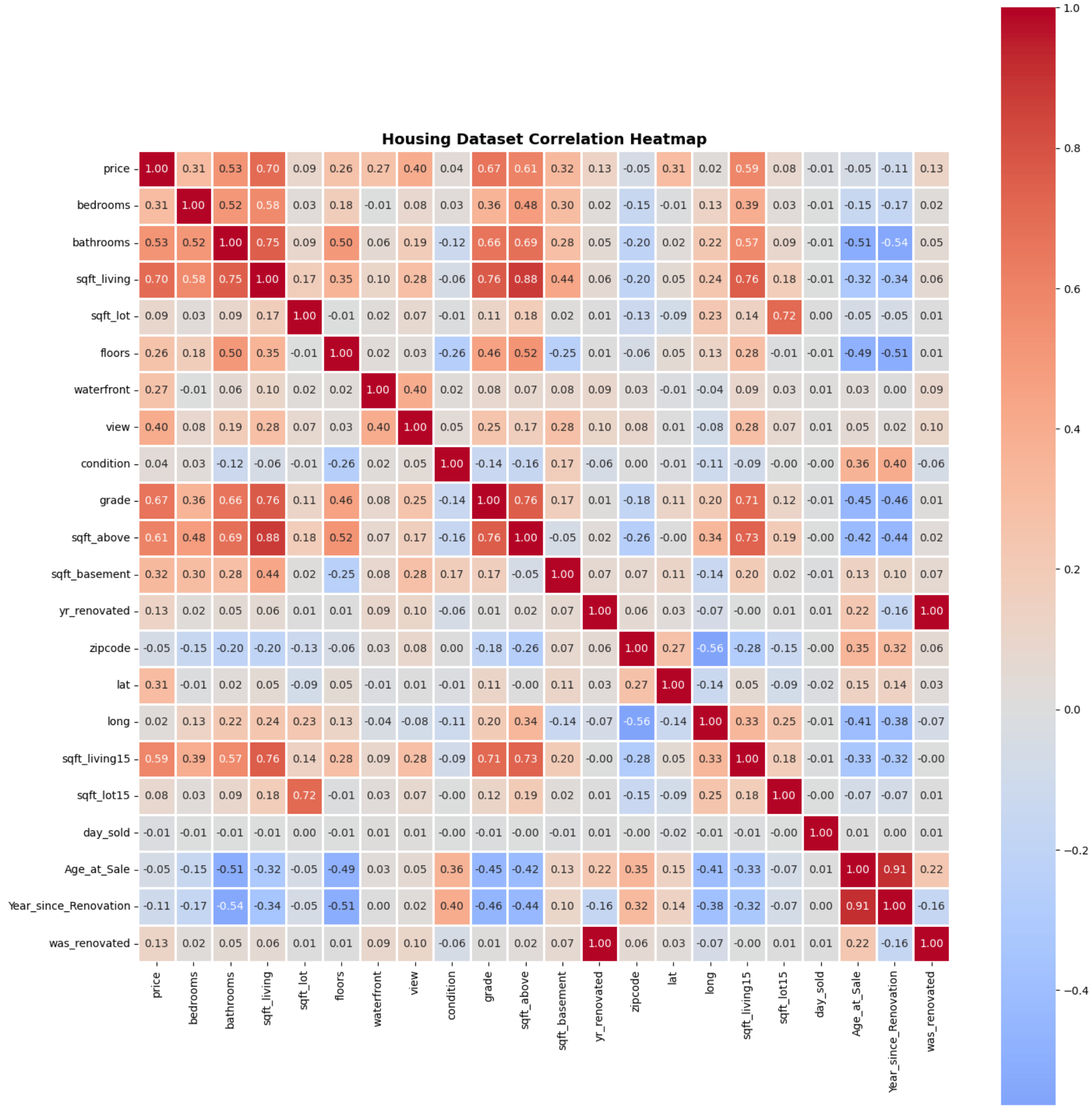
	yr_renovated	zipcode	lat	long	sqft_living15	\
price	0.126	-0.053	0.307	0.022	0.585	
bedrooms	0.019	-0.153	-0.009	0.129	0.392	
bathrooms	0.051	-0.204	0.025	0.223	0.569	
sqft_living	0.055	-0.199	0.053	0.240	0.756	
sqft_lot	0.008	-0.130	-0.086	0.230	0.145	
floors	0.006	-0.059	0.050	0.125	0.280	
waterfront	0.093	0.030	-0.014	-0.042	0.086	
view	0.104	0.085	0.006	-0.078	0.280	
condition	-0.061	0.003	-0.015	-0.107	-0.093	
grade	0.014	-0.185	0.114	0.198	0.713	
sqft_above	0.023	-0.261	-0.001	0.344	0.732	
sqft_basement	0.071	0.075	0.111	-0.145	0.200	
yr_renovated	1.000	0.064	0.029	-0.068	-0.003	
zipcode	0.064	1.000	0.267	-0.564	-0.279	
lat	0.029	0.267	1.000	-0.136	0.049	
long	-0.068	-0.564	-0.136	1.000	0.335	
sqft_living15	-0.003	-0.279	0.049	0.335	1.000	
sqft_lot15	0.008	-0.147	-0.086	0.254	0.183	
day_sold	0.008	-0.003	-0.016	-0.007	-0.009	
Age_at_Sale	0.224	0.347	0.148	-0.409	-0.327	
Year_since_Renovation	-0.165	0.321	0.135	-0.383	-0.325	
was_renovated	1.000	0.064	0.029	-0.068	-0.003	

	sqft_lot15	day_sold	Age_at_Sale	\
price	0.082	-0.015	-0.054	
bedrooms	0.029	-0.008	-0.154	
bathrooms	0.087	-0.005	-0.506	
sqft_living	0.183	-0.007	-0.318	
sqft_lot	0.719	0.001	-0.053	
floors	-0.011	-0.007	-0.490	
waterfront	0.031	0.011	0.026	
view	0.073	0.011	0.053	
condition	-0.003	-0.005	0.361	
grade	0.119	-0.012	-0.447	
sqft_above	0.194	-0.002	-0.424	
sqft_basement	0.017	-0.010	0.133	

yr_renovated	0.008	0.008	0.224
zipcode	-0.147	-0.003	0.347
lat	-0.086	-0.016	0.148
long	0.254	-0.007	-0.409
sqft_living15	0.183	-0.009	-0.327
sqft_lot15	1.000	-0.003	-0.071
day_sold	-0.003	1.000	0.006
Age_at_Sale	-0.071	0.006	1.000
Year_since_Renovation	-0.070	0.003	0.910
was_renovated	0.008	0.008	0.225

	Year_since_Renovation	was_renovated
price	-0.106	0.126
bedrooms	-0.166	0.019
bathrooms	-0.537	0.050
sqft_living	-0.344	0.055
sqft_lot	-0.053	0.008
floors	-0.506	0.006
waterfront	0.000	0.093
view	0.018	0.104
condition	0.396	-0.060
grade	-0.461	0.014
sqft_above	-0.436	0.023
sqft_basement	0.102	0.071
yr_renovated	-0.165	1.000
zipcode	0.321	0.064
lat	0.135	0.029
long	-0.383	-0.068
sqft_living15	-0.325	-0.003
sqft_lot15	-0.070	0.008
day_sold	0.003	0.008
Age_at_Sale	0.910	0.225
Year_since_Renovation	1.000	-0.164
was_renovated	-0.164	1.000

[22 rows x 22 columns]

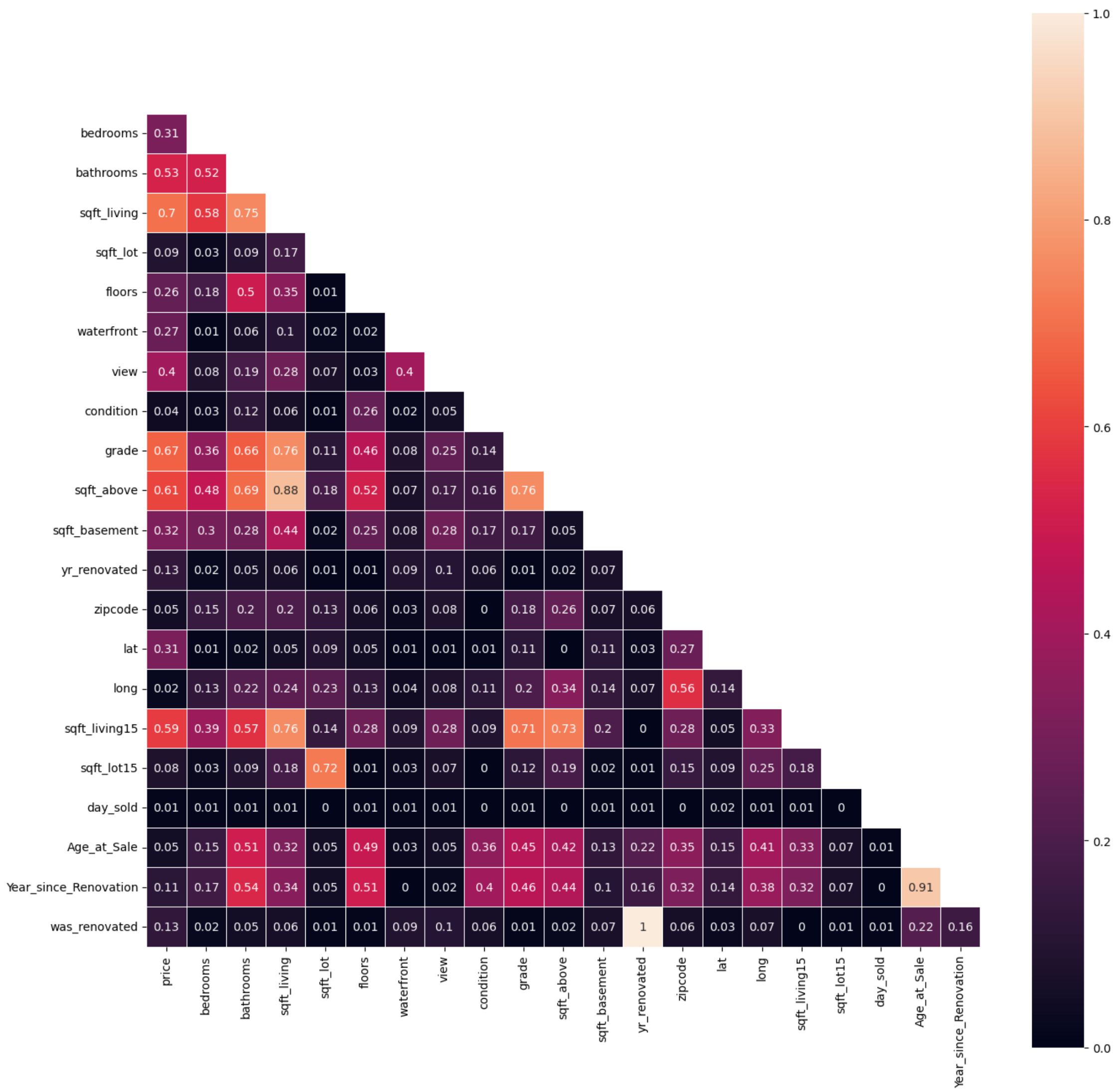


[7]: <Figure size 640x480 with 0 Axes>

<Figure size 640x480 with 0 Axes>

7 Parameterized Heatmap

[8]:
```
generate_heatmap(Cleaned_Dataset)
```



8 Feature Importance

- Lasso Regularization

[9]:
```
std_scaler = StandardScaler()

X_train_standard = std_scaler.fit_transform(X_train)
X_test_standard = std_scaler.fit_transform(X_test)

print(X_train_standard.min(), X_train_standard.max())
```

```
-3.956684808851382 39.35388067380475
```

[10]:
```
features = X_train.columns
features
```

[10]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'day_sold', 'Age_at_Sale',
       'Year_since_Renovation', 'was_renovated'],
      dtype='object')
```

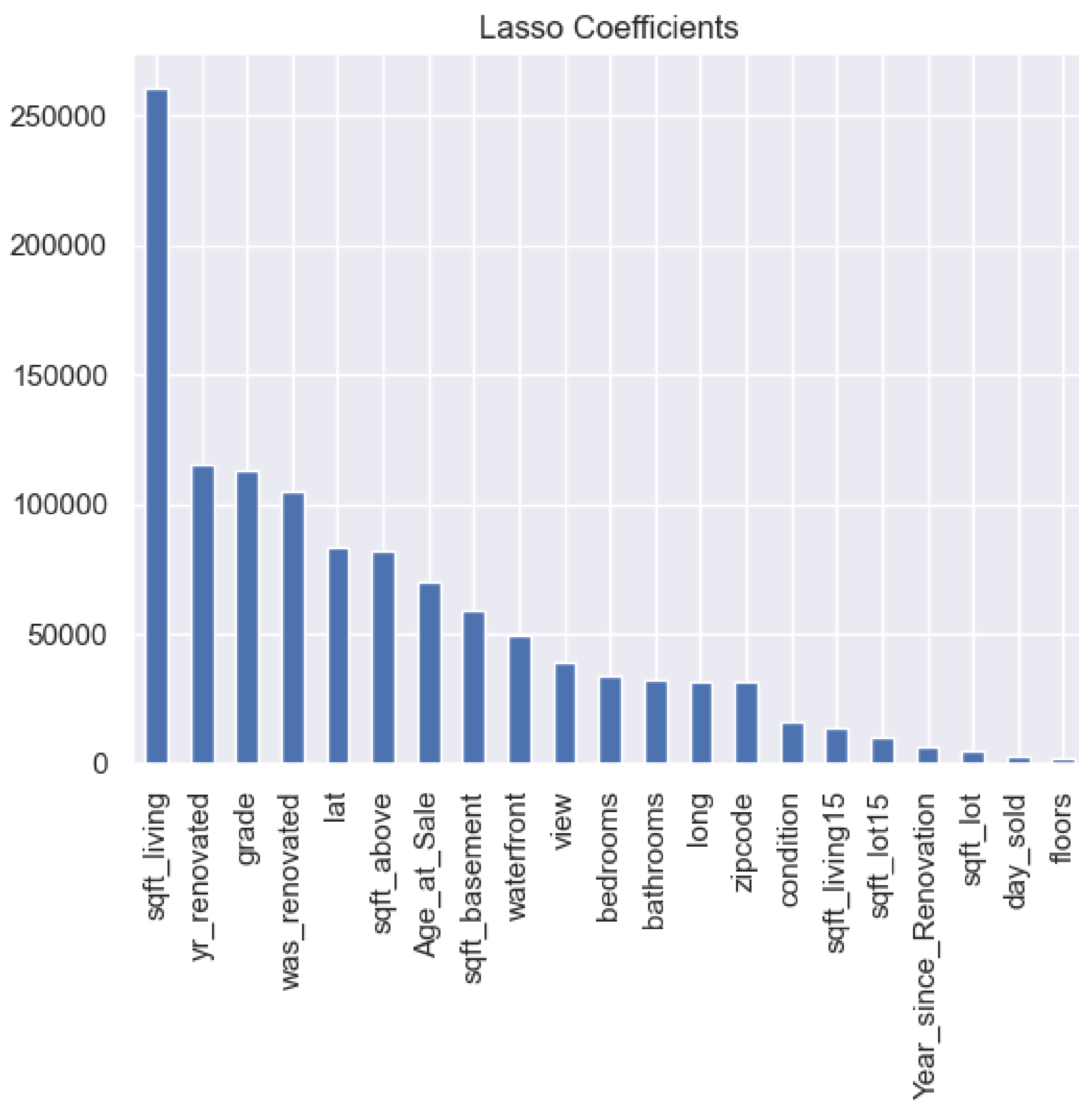
[11]:
```
from IPython.display import clear_output

lasso_regressor = Lasso(random_state=seed)

lasso_regressor.fit(X_train_standard, y_train)

# Remove warning
clear_output()
```

[12]:
```
# Regression coefficients

coefs_lasso = pd.Series(np.abs(lasso_regressor.coef_), features).
 ↪sort_values(ascending=False)

coefs_lasso
```

[12]:
```
sqft_living              260667.308014
yr_renovated             115539.976929
grade                    113464.284916
was_renovated            105055.743718
lat                       83899.034194
sqft_above                81996.071094
Age_at_Sale               70493.906203
sqft_basement             59474.251516
waterfront                49347.059323
view                      39356.052812
bedrooms                  33745.986217
bathrooms                 32306.386441
long                      31980.674767
zipcode                   31606.104587
condition                 16509.503473
sqft_living15             14330.914670
sqft_lot15                10419.853206
Year_since_Renovation      6825.039435
sqft_lot                   4926.447704
day_sold                   3153.359687
floors                     2301.221136
```

```
dtype: float64
```

[13]:
```
sns.set_theme()
coefs_lasso.plot(kind='bar', title='Lasso Coefficients')
fig1 = plt.savefig("Graphs/Lasso_Coefficients.png")
```



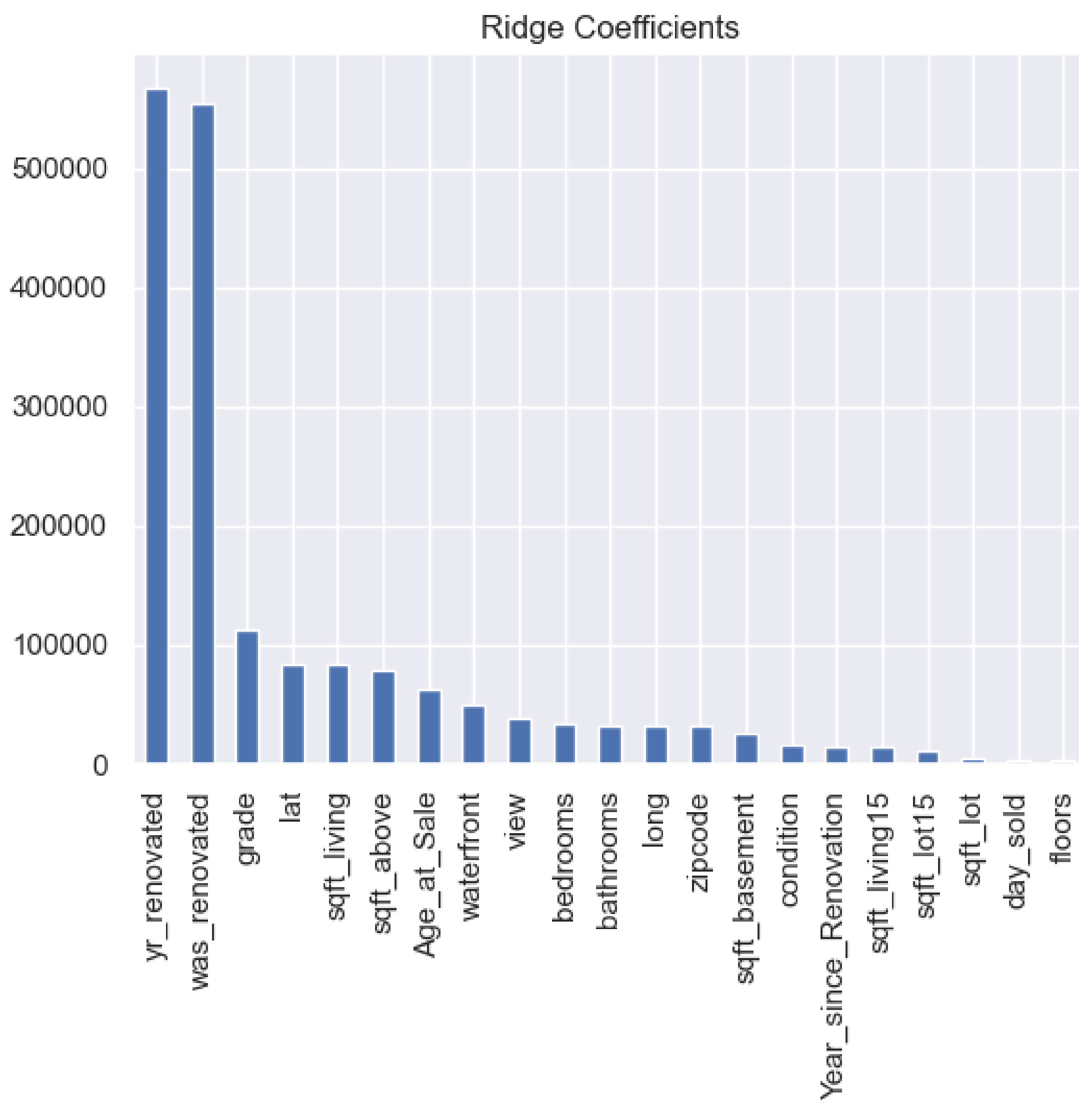
9 Ridge (L2 Regularization)

- Reduces coefficients magnitudes for correlated features. Should give us an idea of mul

[14]:
```
ridge_regressor = Ridge(random_state=seed)
ridge_regressor.fit(X_train_standard, y_train)
```

```python
# Regression coefficients

coefs_ridge = pd.Series(np.abs(ridge_regressor.coef_), features).
 ↪sort_values(ascending=False)
```

[15]:
```python
sns.set_theme()
coefs_ridge.plot(kind='bar', title='Ridge Coefficients')
fig1 = plt.savefig("Graphs/Ridge_Coefficients.png")
```



10 Lasso vs Ridge

[16]:
```python
# combined

feature_importance = pd.DataFrame({"lasso": coefs_lasso, "ridge": coefs_ridge})
```

[17]:
```python
feature_importance = feature_importance.reset_index()
feature_importance
```

[17]:

	index	lasso	ridge
0	Age_at_Sale	70493.906203	62758.474492
1	Year_since_Renovation	6825.039435	14832.195034
2	bathrooms	32306.386441	32258.641807
3	bedrooms	33745.986217	33822.267903
4	condition	16509.503473	16613.195141
5	day_sold	3153.359687	3174.078409
6	floors	2301.221136	2587.054029
7	grade	113464.284916	113284.414558
8	lat	83899.034194	83883.757531
9	long	31980.674767	31943.753755
10	sqft_above	81996.071094	78480.622929
11	sqft_basement	59474.251516	25424.127466
12	sqft_living	260667.308014	83030.526816
13	sqft_living15	14330.914670	14424.743626
14	sqft_lot	4926.447704	4972.626029
15	sqft_lot15	10419.853206	10457.284205
16	view	39356.052812	39373.650718
17	was_renovated	105055.743718	553980.792177
18	waterfront	49347.059323	49480.985886
19	yr_renovated	115539.976929	567539.527477
20	zipcode	31606.104587	31661.929745

[18]:
```python
import pandas as pd
import plotly.express as px

# Fix formating of the dataframe for plotting
df_melted = feature_importance.melt(id_vars='index',
                     value_vars=['lasso', 'ridge'],
                     var_name='Model',
                     value_name='Coefficient')

# Create a interactive bar plot
fig = px.bar(df_melted.sort_values("Coefficient", ascending=False),
             x='index',
             y='Coefficient',
             color='Model',
             barmode='group',        # side-by-side, not stacked
```

```
            color_discrete_map={
                'lasso': '#4c72b0',
                'ridge': '#dd8452'},
             title='Feature Importance: Lasso vs Ridge')

# 3. Rotate x-axis labels
fig.update_layout(
    xaxis_tickangle=-45,
    xaxis_title='Features',
    yaxis_title='Absolute Coefficient Value'
)

fig.show()
fig.write_image("Graphs/Lasso_Ridge_Coefficients.png")
```

11 Linear Regression

- Given Numeriacal Dataset we perform a Linear Regression and start creating metrics

[19]:
```
from sklearn.linear_model import LinearRegression

# Create Estimator
LinReg = LinearRegression()

# Perform Fitting
LinReg.fit(X_train, y_train)

# Make Predictions
y_train_predict = LinReg.predict(X_train)
y_test_predict = LinReg.predict(X_test)

print("Model trained successfully!")
print(f"Training predictions shape: {y_train_predict.shape}")
print(f"Testing predictions shape: {y_test_predict.shape}")
print(f"\nFirst 5 training predictions: {y_train_predict[:5]}")
print(f"First 5 actual training values: {y_train.iloc[:5].values}")

print(f"Intercept: {LinReg.intercept_:.2f}")
print(f"Number of coefficients: {len(LinReg.coef_)}")
print(f"\nTop3 Influential features:")
feature_importance = sorted(zip(X_train.columns,LinReg.coef_),key=lambda x:
 ↪abs(x[1]), reverse=True)[:3]
for feat, coef in feature_importance:
    print(f"{feat}:  {coef:.4f}")

metrics_df = create_metrics_df()
```

```
metrics_df = add_new_metrics(metrics_df,LinReg, X_train, y_train,␣
↪split="train", comments="Linear Regression FE Metrics - train - dropped␣
↪features: date, id, yr_built, year_sold, month_sold")
metrics_df = add_new_metrics(metrics_df,LinReg, X_test, y_test, split="test",␣
↪comments="Linear Regression FE Metrics - test - dropped features: date, id,␣
↪yr_built, year_sold, month_sold")

metrics_df
```

```
Model trained successfully!
Training predictions shape: (17290,)
Testing predictions shape: (4323,)

First 5 training predictions: [247834.7561139  728804.13253798 810711.97302496
216848.11429534
  90564.97897474]
First 5 actual training values: [284950. 625000. 838400. 282000. 218000.]
Intercept: 1355520.65
Number of coefficients: 21

Top3 Influential features:
was_renovated:  -9156171.3570
lat:  604623.1733
waterfront:  583972.7739
```

[19]:

```
              Model  Split      R2  Adjusted_R2          MAE    MAPE  \
0  LinearRegression  train  0.6977       0.6973  125914.8975  0.2557
1  LinearRegression   test  0.7165       0.7151  125994.7484  0.2590

          RMSE                                         Comments
0  202891.3616  Linear Regression FE Metrics - train - dropped…
1  191439.9182  Linear Regression FE Metrics - test - dropped …
```

12 Mean Sq Error, Mean Absolute Error, Root Mean Squared Error

[20]:

```
get_mse(y_train, y_test, y_train_predict, y_test_predict)

get_r_squared(y_train, y_test, y_train_predict, y_test_predict)

get_mae(y_train, y_test, y_train_predict, y_test_predict)
```

```
MSE score:
train | 41164904608.63687
test  | 36649242289.43816
```

```
R2 score:
train | 0.697664220549328
test  | 0.716453297645963


MSE score:
train | 125914.89749025984
test  | 125994.74837969066
```

[20]: (125914.89749025984, 125994.74837969066)

13 Random Forest - Regression

- Random Forrest Regressor is applied, again without controlling any parameters.

[21]:
```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

rf_regressor = RandomForestRegressor(n_estimators=110,random_state=seed)

rf_regressor.fit(X_train, y_train)
```

[21]: RandomForestRegressor(n_estimators=110, random_state=13)

13.0.1 Predictions

[22]:
```python
y_pred = rf_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")

single_data = X_test.iloc[0].values.reshape(1, -1)
predicted_value = rf_regressor.predict(single_data)
print(f"Predicted Value: {predicted_value[0]:.2f}")
print(f"Actual Value: {y_test.iloc[0]:.2f}")
```

```
metrics_df = add_new_metrics(metrics_df, rf_regressor, X_train, y_train,␣
↪split="train", comments="Random Forest Train FE Metrics - train - dropped␣
↪features: date, id, yr_built, year_sold, month_sold")
metrics_df = add_new_metrics(metrics_df,rf_regressor, X_test, y_test,␣
↪split="test", comments="Random Forest Test FE Metrics - test - dropped␣
↪features: date, id, yr_built, year_sold, month_sold")

metrics_df
```

```
Mean Squared Error: 13500260163.85
R-squared Score: 0.90
Predicted Value: 290463.59
Actual Value: 324900.00
```

```
c:\Users\KaoticCharma\anaconda3\Lib\site-
packages\sklearn\utils\validation.py:2691: UserWarning:

X does not have valid feature names, but RandomForestRegressor was fitted with
feature names
```

[22]:

```
                   Model  Split      R2  Adjusted_R2          MAE    MAPE  \
0       LinearRegression  train  0.6977       0.6973  125914.8975  0.2557
1       LinearRegression   test  0.7165       0.7151  125994.7484  0.2590
2  RandomForestRegressor  train  0.9820       0.9820   25980.2304  0.0486
3  RandomForestRegressor   test  0.8956       0.8950   67606.7385  0.1280

          RMSE                                          Comments
0  202891.3616  Linear Regression FE Metrics - train - dropped…
1  191439.9182  Linear Regression FE Metrics - test - dropped …
2   49538.9365  Random Forest Train FE Metrics - train - dropp…
3  116190.6199  Random Forest Test FE Metrics - test - dropped…
```

14 Random Forest - Regression with Feature Importance

- Random Forrest Regressor is applied, again without controlling any parameters.

[23]:

```
from sklearn.ensemble import RandomForestRegressor

rf_regressor = RandomForestRegressor(random_state=seed)#default values +␣
↪random_state = 13
rf_regressor.fit(X_train, y_train)
```

[23]: RandomForestRegressor(random_state=13)

[24]:

```
y_pred = rf_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
```

```python
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")

single_data = X_test.iloc[0].values.reshape(1, -1)
predicted_value = rf_regressor.predict(single_data)
print(f"Predicted Value: {predicted_value[0]:.2f}")
print(f"Actual Value: {y_test.iloc[0]:.2f}")
```

```
Mean Squared Error: 13619426178.18
R-squared Score: 0.89
Predicted Value: 289293.45
Actual Value: 324900.00
```

```
c:\Users\KaoticCharma\anaconda3\Lib\site-
packages\sklearn\utils\validation.py:2691: UserWarning:

X does not have valid feature names, but RandomForestRegressor was fitted with
feature names
```

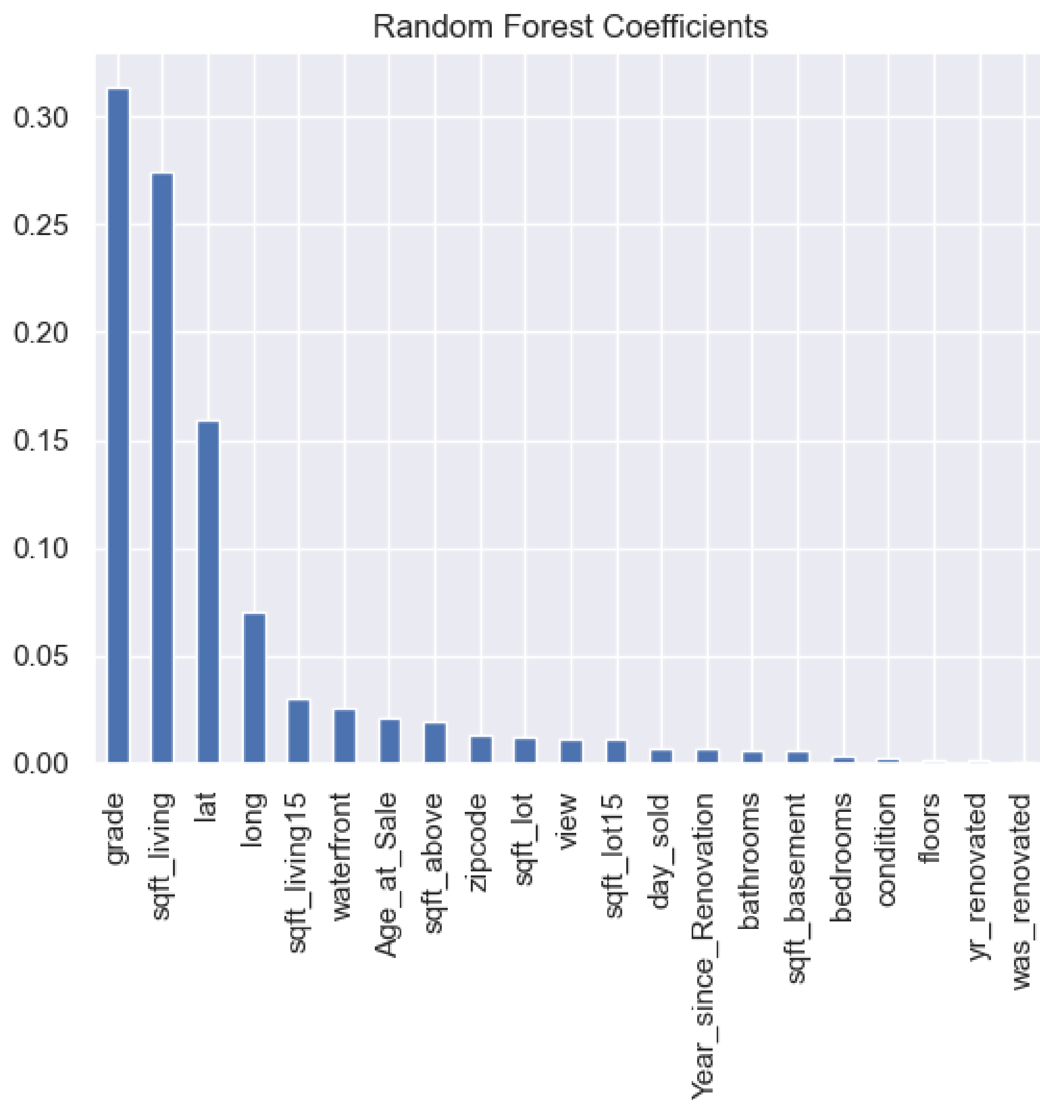
[25]:
```python
features = X_train.columns
coefs_rf = pd.Series(np.abs(rf_regressor.feature_importances_), features).
 ↪sort_values(ascending=False)

metrics_df = add_new_metrics(metrics_df, rf_regressor, X_train, y_train,␣
 ↪split="train", comments="Random Forest Train FE + FI  Metrics - train -␣
 ↪dropped features: date, id, yr_built, year_sold, month_sold")
metrics_df = add_new_metrics(metrics_df,rf_regressor, X_test, y_test,␣
 ↪split="test", comments="Random Forest Test FE + FI Metrics - test - dropped␣
 ↪features: date, id, yr_built, year_sold, month_sold")

coefs_rf
```

[25]:
```
grade            0.313535
sqft_living      0.274619
lat              0.159416
long             0.070064
sqft_living15    0.030581
waterfront       0.025367
Age_at_Sale      0.020887
sqft_above       0.019248
zipcode          0.012981
sqft_lot         0.012404
view             0.011900
sqft_lot15       0.011859
day_sold         0.007335
```

```
Year_since_Renovation    0.007207
bathrooms                0.006546
sqft_basement            0.006109
bedrooms                 0.003223
condition                0.002707
floors                   0.001969
yr_renovated             0.001513
was_renovated            0.000530
dtype: float64
```

[26]:

```python
coefs_rf.plot(kind='bar', title='Random Forest Coefficients')
fig1 = plt.savefig("Graphs/RandomForest_Coefficients.png")
```



15 XGBoost

[27]:
```
xgb_regressor = xgb.XGBRegressor(n_estimators=100, random_state=seed)

xgb_regressor.fit(X_train, y_train)

metrics_df = add_new_metrics(metrics_df, xgb_regressor, X_train, y_train,␣
 ↪split="train", comments="XGBoost Train FE Metrics - train - dropped features␣
 ↪- date, id, yr_built, year_sold, month_sold")
metrics_df = add_new_metrics(metrics_df, xgb_regressor, X_test, y_test,␣
 ↪split="test", comments="XGBoost Test FE Metrics - test - dropped features -␣
 ↪date, id, yr_built, year_sold, month_sold")

metrics_df

metrics_df.to_csv("ModelMetrics_FE.csv", index=False)

metrics_df
```

[27]:
```
                   Model  Split      R2  Adjusted_R2          MAE    MAPE  \
0       LinearRegression  train  0.6977       0.6973  125914.8975  0.2557
1       LinearRegression   test  0.7165       0.7151  125994.7484  0.2590
2  RandomForestRegressor  train  0.9820       0.9820   25980.2304  0.0486
3  RandomForestRegressor   test  0.8956       0.8950   67606.7385  0.1280
4  RandomForestRegressor  train  0.9819       0.9819   26031.4643  0.0487
5  RandomForestRegressor   test  0.8946       0.8941   67781.7366  0.1283
6           XGBRegressor  train  0.9761       0.9761   40791.8177  0.0903
7           XGBRegressor   test  0.9038       0.9033   66773.7091  0.1271

          RMSE                                           Comments
0  202891.3616  Linear Regression FE Metrics - train - dropped…
1  191439.9182  Linear Regression FE Metrics - test - dropped …
2   49538.9365  Random Forest Train FE Metrics - train - dropp…
3  116190.6199  Random Forest Test FE Metrics - test - dropped…
4   49598.7780  Random Forest Train FE + FI  Metrics - train -…
5  116702.2972  Random Forest Test FE + FI Metrics - test - dr…
6   57067.4036  XGBoost Train FE Metrics - train - dropped fea…
7  111527.5852  XGBoost Test FE Metrics - test - dropped featu…
```

16 XGBoost with Feature Importance

[28]:
```
import xgboost as xgb

xgb_clf = xgb.XGBRegressor(seed = seed)
xgb_clf.fit(X_train, y_train)
```

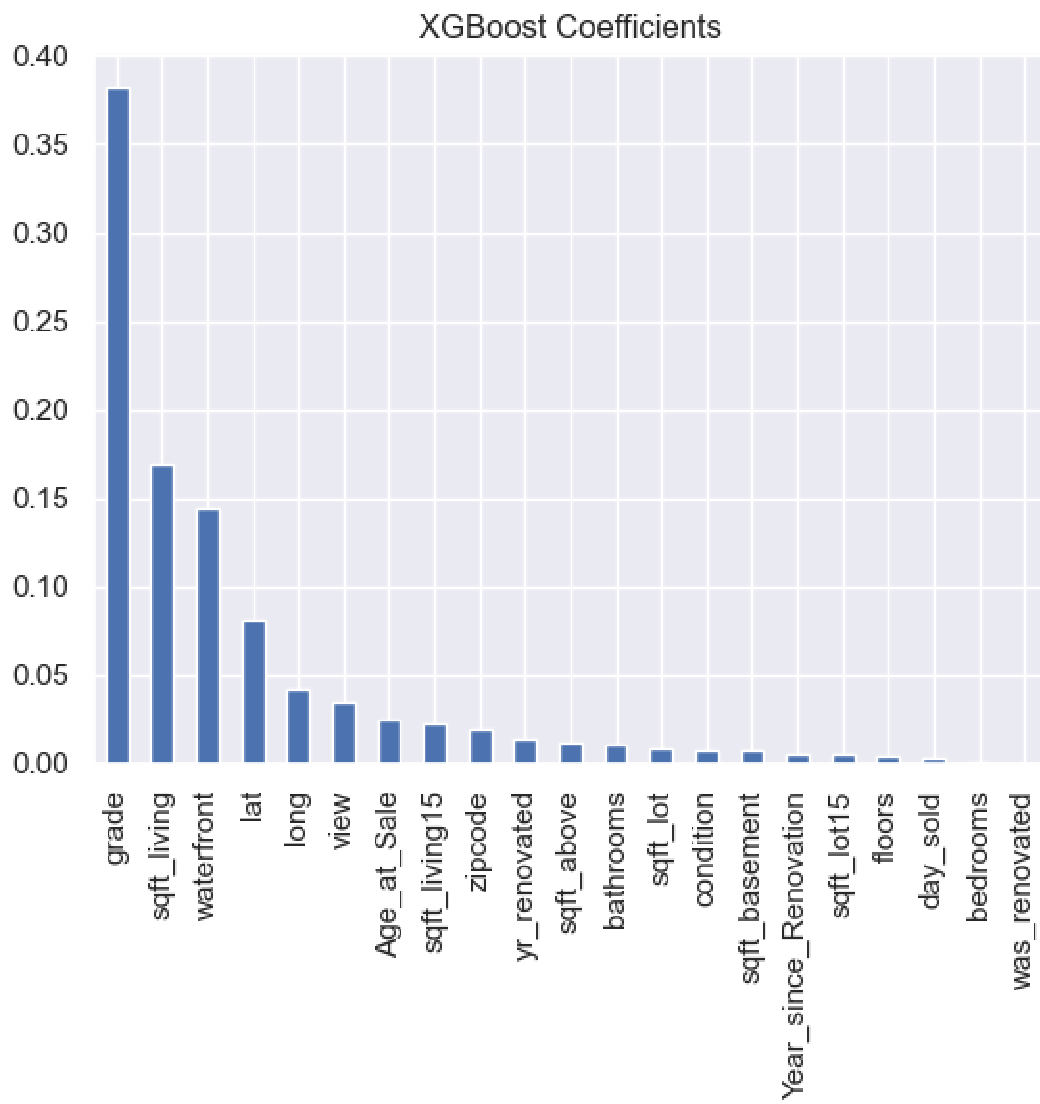
[28]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, …)
```

[29]:
```
features = X_train.columns
coefs_xgb = pd.Series(np.abs(xgb_clf.feature_importances_), features).
 ↪sort_values(ascending=False)
metrics_df = add_new_metrics(metrics_df, xgb_clf, X_train, y_train,␣
 ↪split="train", comments="XGBoost Train FE + FI Metrics - train - dropped␣
 ↪features: date, id, yr_built, year_sold, month_sold")
metrics_df = add_new_metrics(metrics_df, xgb_clf, X_test, y_test, split="test",␣
 ↪comments="XGBoost Test FE + FI Metrics - test - dropped features: date, id,␣
 ↪yr_built, year_sold, month_sold")

coefs_xgb
```

[29]:
```
grade                   0.382075
sqft_living             0.169310
waterfront              0.144378
lat                     0.081549
long                    0.042398
view                    0.034126
Age_at_Sale             0.024606
sqft_living15           0.022307
zipcode                 0.019849
yr_renovated            0.013571
sqft_above              0.012126
bathrooms               0.010843
sqft_lot                0.008284
condition               0.007685
sqft_basement           0.007208
Year_since_Renovation   0.005419
sqft_lot15              0.005056
floors                  0.004667
day_sold                0.003014
bedrooms                0.001529
was_renovated           0.000000
dtype: float32
```

[30]:
```python
coefs_xgb.plot(kind='bar', title='XGBoost Coefficients')
fig1 = plt.savefig("Graphs/XGBoost_Coefficients.png")
metrics_df
metrics_df.to_csv("ModelMetrics_FE.csv", index=False)
metrics_df
```

[30]:
```
                   Model  Split      R2  Adjusted_R2          MAE    MAPE  \
0       LinearRegression  train  0.6977       0.6973  125914.8975  0.2557
1       LinearRegression   test  0.7165       0.7151  125994.7484  0.2590
2  RandomForestRegressor  train  0.9820       0.9820   25980.2304  0.0486
3  RandomForestRegressor   test  0.8956       0.8950   67606.7385  0.1280
4  RandomForestRegressor  train  0.9819       0.9819   26031.4643  0.0487
5  RandomForestRegressor   test  0.8946       0.8941   67781.7366  0.1283
6           XGBRegressor  train  0.9761       0.9761   40791.8177  0.0903
7           XGBRegressor   test  0.9038       0.9033   66773.7091  0.1271
8           XGBRegressor  train  0.9761       0.9761   40791.8177  0.0903
9           XGBRegressor   test  0.9038       0.9033   66773.7091  0.1271

          RMSE                                           Comments
0  202891.3616  Linear Regression FE Metrics - train - dropped…
1  191439.9182  Linear Regression FE Metrics - test - dropped …
2   49538.9365  Random Forest Train FE Metrics - train - dropp…
3  116190.6199  Random Forest Test FE Metrics - test - dropped…
4   49598.7780  Random Forest Train FE + FI  Metrics - train -…
5  116702.2972  Random Forest Test FE + FI Metrics - test - dr…
6   57067.4036  XGBoost Train FE Metrics - train - dropped fea…
7  111527.5852  XGBoost Test FE Metrics - test - dropped featu…
8   57067.4036  XGBoost Train FE + FI Metrics - train - droppe…
9  111527.5852  XGBoost Test FE + FI Metrics - test - dropped …
```


XGBoost Coefficients
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
grade
sqft_living
waterfront
lat
long
view
Age_at_Sale
sqft_living15
zipcode
yr_renovated
sqft_above
bathrooms
sqft_lot
condition
sqft_basement
Year_since_Renovation
sqft_lot15
floors
day_sold
bedrooms
was_renovated