2_Feature_Importance

February 14, 2026

1 Feature Importance

1.1 Imports

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Models & Normalization
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb

# Evaluation
import statsmodels.api as sm

# Extra
from utils import *
```

[2]:
```
# Exporting plotly plots to pdf
import plotly.io as pio

# Configuration:

# Set to False for interactive zooming/hovering (Exploration)
# Set to True for static images (PDF Export)
EXPORT_MODE = True

if EXPORT_MODE:
    # Forces all charts to be static images (requires 'pip install -U kaleido')
    pio.renderers.default = "png" # Use "png" if svg gives you trouble
    print("  EXPORT_MODE is ON. Charts will be static images.")
else:
    # Default interactive plotly
```

```
    pio.renderers.default = "notebook_connected"
    print(" Interactive mode. Charts will include zoom/hover.")
```

```
EXPORT_MODE is ON. Charts will be static images.
```

1.2 Open the Clean data

[3]:
```
file_path = "Data/v0_cleaned_house_sales.csv"

df_clean = pd.read_csv(file_path)
df_clean.head()
```

[3]:
```
      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  \
0  221900.0         3       1.00         1180      5650     1.0           0
1  538000.0         3       2.25         2570      7242     2.0           0
2  180000.0         2       1.00          770     10000     1.0           0
3  604000.0         4       3.00         1960      5000     1.0           0
4  510000.0         3       2.00         1680      8080     1.0           0

   view  condition  grade  …  yr_built  yr_renovated  zipcode      lat  \
0     0          3      7  …      1955             0    98178  47.5112
1     0          3      7  …      1951          1991    98125  47.7210
2     0          3      6  …      1933             0    98028  47.7379
3     0          5      7  …      1965             0    98136  47.5208
4     0          3      8  …      1987             0    98074  47.6168

      long  sqft_living15  sqft_lot15  year_sold  month_sold  day_sold
0 -122.257           1340        5650       2014          10        13
1 -122.319           1690        7639       2014          12         9
2 -122.233           2720        8062       2015           2        25
3 -122.393           1360        5000       2014          12         9
4 -122.045           1800        7503       2015           2        18

[5 rows x 22 columns]
```

[4]:
```
# Same seed for all random states
seed = 13

# The price is the target variable
y = df_clean["price"]

# All other variables are the features for the baseline model
X = df_clean.drop("price", axis=1)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=seed)
```

1.3 Lasso (L1 regularization)

Should reduce less important features to 0. Applying it to find features we can drop.

[5]:
```
# standardization

std_scaler = StandardScaler()

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

[6]:
```
print(X_train_standard.min(), X_train_standard.max())
```

```
-3.956684808851382 39.35388067380475
```

The max value being so extreme even after standardization points to a distribution with many outliers.

[7]:
```
features = X_train.columns
features
```

[7]:
```
Index(['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'],
      dtype='str')
```

[8]:
```
from IPython.display import clear_output

lasso_regressor = Lasso(random_state=seed)

lasso_regressor.fit(X_train_standard, y_train)

# Remove warning
clear_output()
```

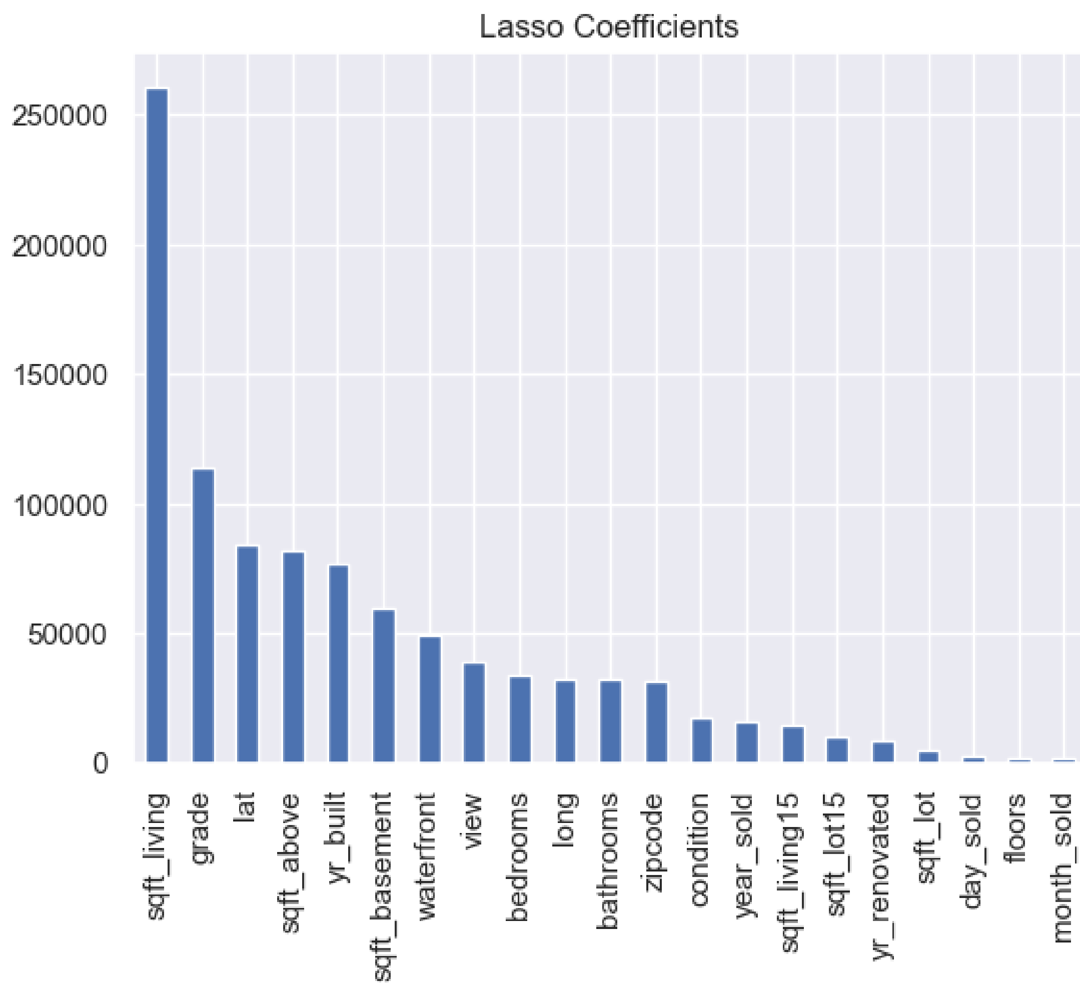
[9]:
```
# Regression coefficients

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

coefs_lasso
```

[9]:
```
sqft_living      260890.945338
grade            113621.446101
lat               84316.523244
sqft_above        81991.026779
yr_built          76475.889246
sqft_basement     59472.818333
```

```
waterfront        49274.007710
view              39200.831428
bedrooms          33789.239856
long              32117.139742
bathrooms         32082.314821
zipcode           31555.515044
condition         17557.032206
year_sold         16266.798818
sqft_living15     14460.312554
sqft_lot15        10391.559985
yr_renovated       8500.402589
sqft_lot           4863.151159
day_sold           2879.776176
floors             2293.208022
month_sold         1979.637736
dtype: float64
```

[10]:
```python
sns.set_theme()
coefs_lasso.plot(kind='bar', title='Lasso Coefficients')
```

[10]:
```
<Axes: title={'center': 'Lasso Coefficients'}>
```



The lasso regressor did not identify any irrelevant parameters that we could currently drop.

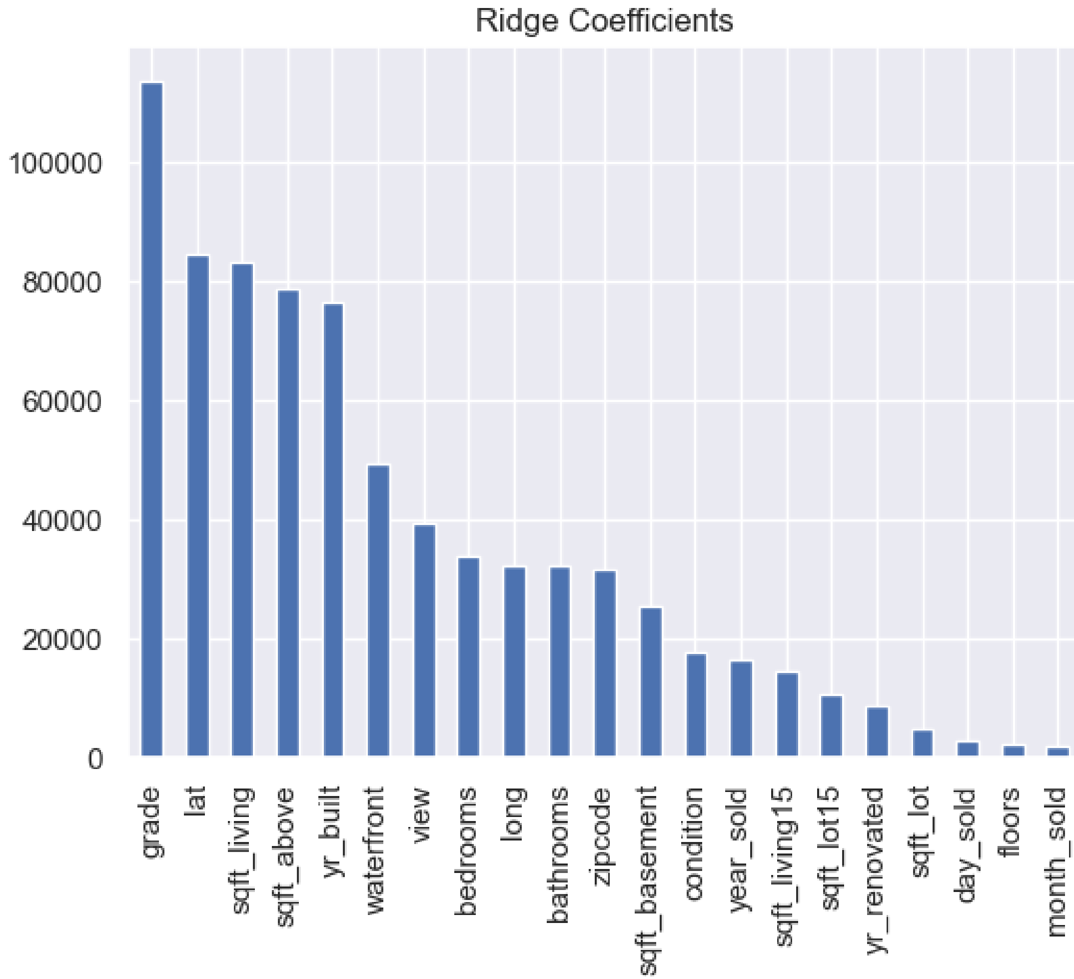
1.4 Ridge (L2 regularization)

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

[11]:
```
ridge_regressor = Ridge(random_state=seed)
ridge_regressor.fit(X_train_standard, y_train)


# Regression coefficients

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

[12]:
```
sns.set_theme()
coefs_ridge.plot(kind='bar', title='Ridge Coefficients')
```

[12]: <Axes: title={'center': 'Ridge Coefficients'}>



1.4.1 Lasso vs. Ridge

[13]:
```
# combined

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

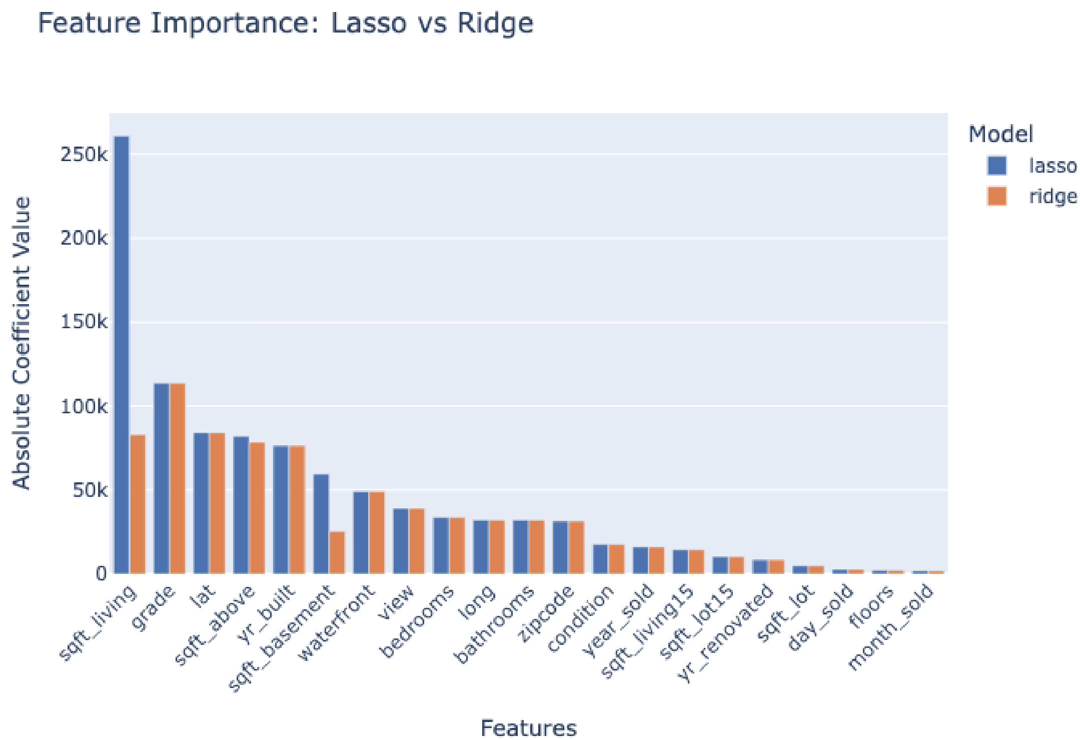
[14]:
```
feature_importance = feature_importance.reset_index()
feature_importance
```

[14]:

	index	lasso	ridge
0	bathrooms	32082.314821	32087.628497
1	bedrooms	33789.239856	33783.663620
2	condition	17557.032206	17559.541392
3	day_sold	2879.776176	2880.548680

```
4           floors    2293.208022    2295.681048
5            grade  113621.446101  113606.366537
6              lat   84316.523244   84314.151501
7             long   32117.139742   32118.058335
8       month_sold    1979.637736    1982.151021
9       sqft_above   81991.026779   78520.819072
10   sqft_basement   59472.818333   25543.707561
11     sqft_living  260890.945338   83124.018496
12   sqft_living15   14460.312554   14478.214453
13        sqft_lot    4863.151159    4866.064328
14      sqft_lot15   10391.559985   10392.616045
15            view   39200.831428   39200.967653
16      waterfront   49274.007710   49272.716356
17       year_sold   16266.798818   16268.574981
18        yr_built   76475.889246   76466.727470
19    yr_renovated    8500.402589    8503.739225
20         zipcode   31555.515044   31551.106899
```

[15]:
```
# 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()
```



1.5 XGBoost

[16]:
```
xgb_clf = xgb.XGBRegressor(seed = seed)
xgb_clf.fit(X_train, y_train)
```

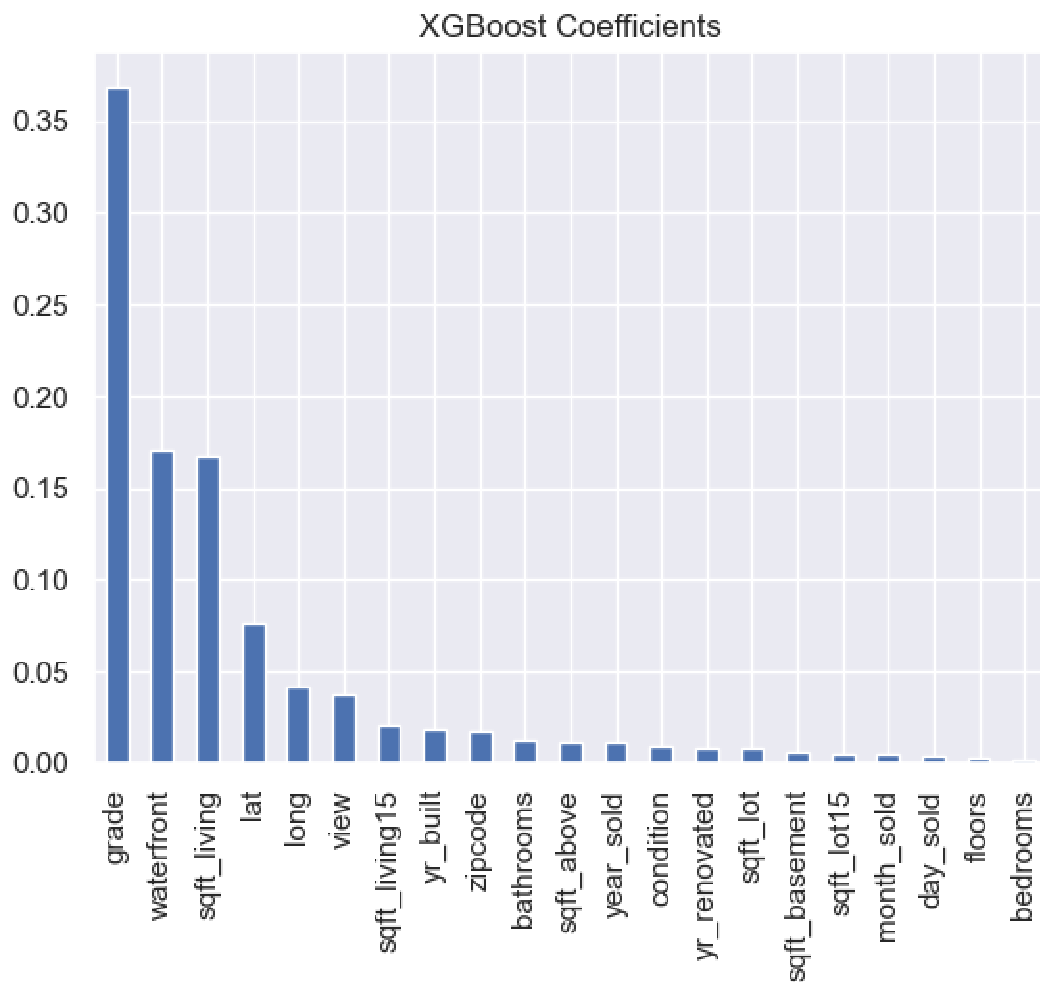
[16]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, 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, …)
```

[17]:
```
features = X_train.columns
coefs_xgb = pd.Series(np.abs(xgb_clf.feature_importances_), features).
 ↪sort_values(ascending=False)

coefs_xgb
```

[17]:
```
grade            0.368874
waterfront       0.170580
sqft_living      0.167270
lat              0.075877
long             0.041144
view             0.037144
sqft_living15    0.020389
yr_built         0.018146
zipcode          0.016965
bathrooms        0.012077
sqft_above       0.011518
year_sold        0.011033
condition        0.009306
yr_renovated     0.008090
sqft_lot         0.007944
sqft_basement    0.005541
sqft_lot15       0.005373
month_sold       0.004566
day_sold         0.003449
floors           0.003055
bedrooms         0.001659
dtype: float32
```

[18]:
```
coefs_xgb.plot(kind='bar', title='XGBoost Coefficients')
```

[18]:
```
<Axes: title={'center': 'XGBoost Coefficients'}>
```



1.6 Random Forest

[19]:
```
# most common hyperparameters or the default ones

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

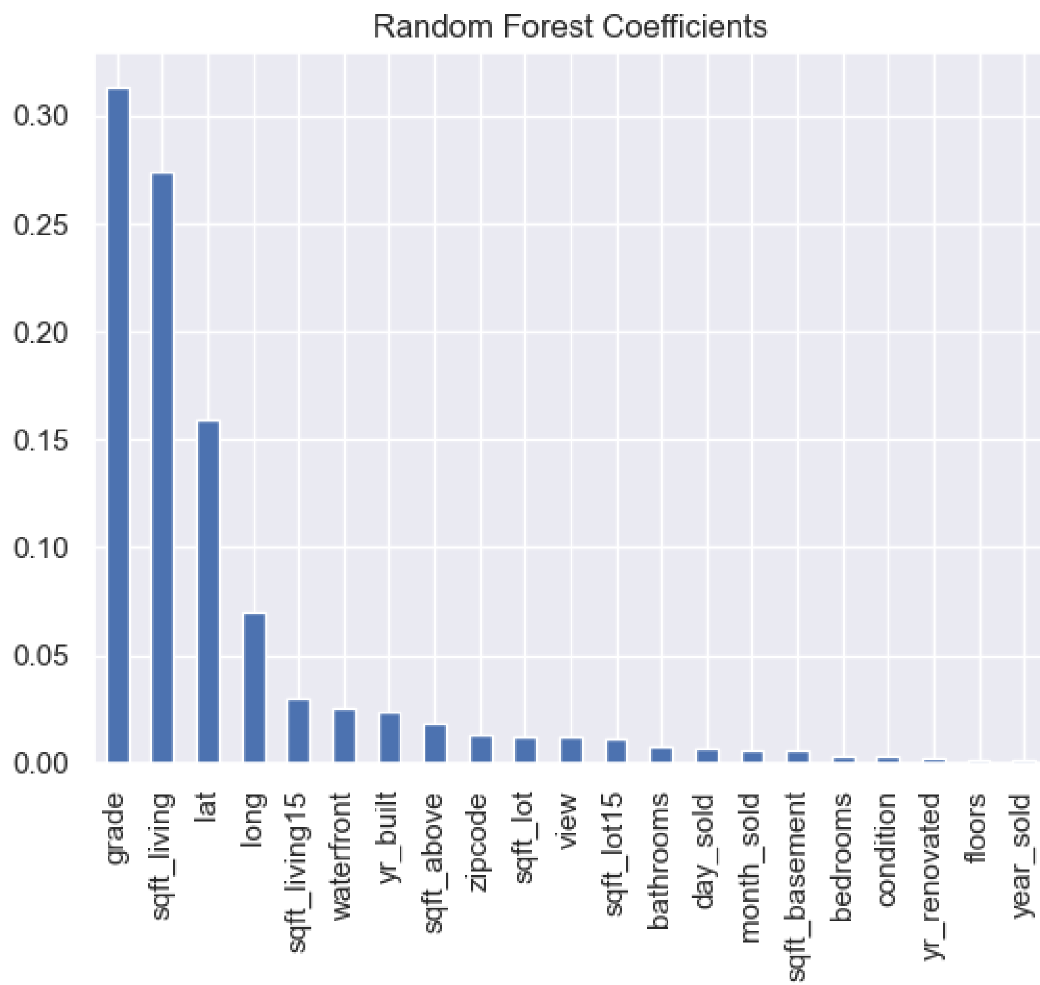
[19]: RandomForestRegressor(random_state=13)

[20]:
```
features = X_train.columns
coefs_rf = pd.Series(np.abs(rf_regressor.feature_importances_), features).
↪sort_values(ascending=False)

coefs_rf
```

[20]:
```
grade            0.313353
sqft_living      0.274236
lat              0.159146
long             0.069911
sqft_living15    0.030356
waterfront       0.025279
yr_built         0.023713
sqft_above       0.018558
zipcode          0.012953
sqft_lot         0.012211
view             0.011875
sqft_lot15       0.011449
bathrooms        0.007467
day_sold         0.006681
month_sold       0.006077
sqft_basement    0.005584
bedrooms         0.002960
condition        0.002917
yr_renovated     0.002241
floors           0.001909
year_sold        0.001126
dtype: float64
```

[21]:
```
coefs_rf.plot(kind='bar', title='Random Forest Coefficients')
```

[21]:
```
<Axes: title={'center': 'Random Forest Coefficients'}>
```



1.6.1 Random Forest vs XGBoost

[22]:
```
feature_importance = pd.DataFrame({"rf": coefs_rf, "xgb": coefs_xgb})
feature_importance = feature_importance.reset_index()
feature_importance
```

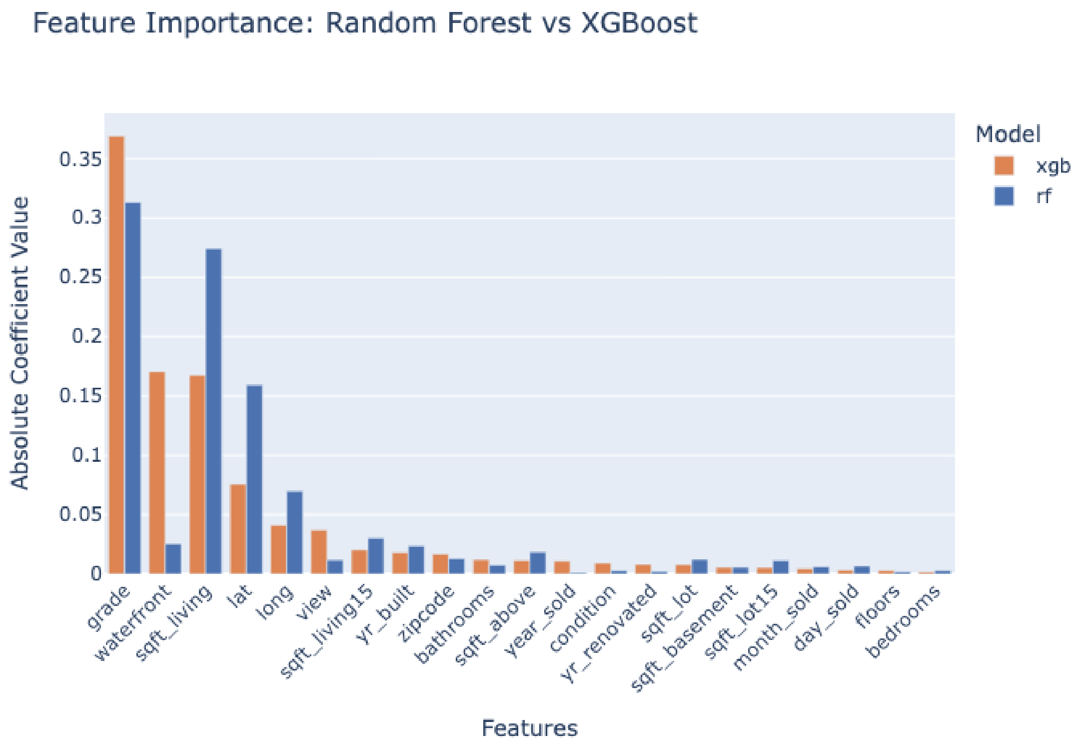
[22]:

	index	rf	xgb
0	bathrooms	0.007467	0.012077
1	bedrooms	0.002960	0.001659
2	condition	0.002917	0.009306
3	day_sold	0.006681	0.003449
4	floors	0.001909	0.003055
5	grade	0.313353	0.368874
6	lat	0.159146	0.075877
7	long	0.069911	0.041144
8	month_sold	0.006077	0.004566

```
9      sqft_above  0.018558  0.011518
10  sqft_basement  0.005584  0.005541
11    sqft_living  0.274236  0.167270
12  sqft_living15  0.030356  0.020389
13       sqft_lot  0.012211  0.007944
14     sqft_lot15  0.011449  0.005373
15           view  0.011875  0.037144
16     waterfront  0.025279  0.170580
17      year_sold  0.001126  0.011033
18       yr_built  0.023713  0.018146
19   yr_renovated  0.002241  0.008090
20        zipcode  0.012953  0.016965
```

[23]:

```python
# Fix formating of the dataframe for plotting
df_melted = feature_importance.melt(id_vars='index',
                     value_vars=['rf', 'xgb'],
                     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={
                 'rf': '#4c72b0',
                 'xgb': '#dd8452'},
             title='Feature Importance: Random Forest vs XGBoost')

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

fig.show()
```



1.6.2 Insights

The two ensemble algorithms, Random Forest and XGBoost, generally agree about their top features, even though they have varying degrees of importance.

[ ]:

```

```