Airbnb listings file modeling - Initial probing

This file establishes the baseline metrics for each of the models chosen by running them through the data straight after cleaning.

The effects of additional processing of the data are explored later in subsequent modeling and EDA files.

In [151]:
```
# Put these at the top of every notebook, to get automatic reloading and inl
from IPython.core.display import display, HTML
import pandas as pd
import warnings
import ast
warnings.filterwarnings('ignore')

%reload_ext autoreload
%autoreload 1
%matplotlib inline

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [152]:
```
import os
import seaborn as sns
import pandas as pd
import math
import sklearn.model_selection as cv

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegresso
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import Lasso, Ridge

from sklearn.metrics import mean_squared_error as MSE

from imblearn.over_sampling import SMOTE

from Utils.UtilsGeoViz import *
from Utils.UtilsViz import *
from Utils.DataUtils import *

RANDOM_SEED = 42
```

In [153]:
```
sns.set(font_scale=2, style="whitegrid")
```

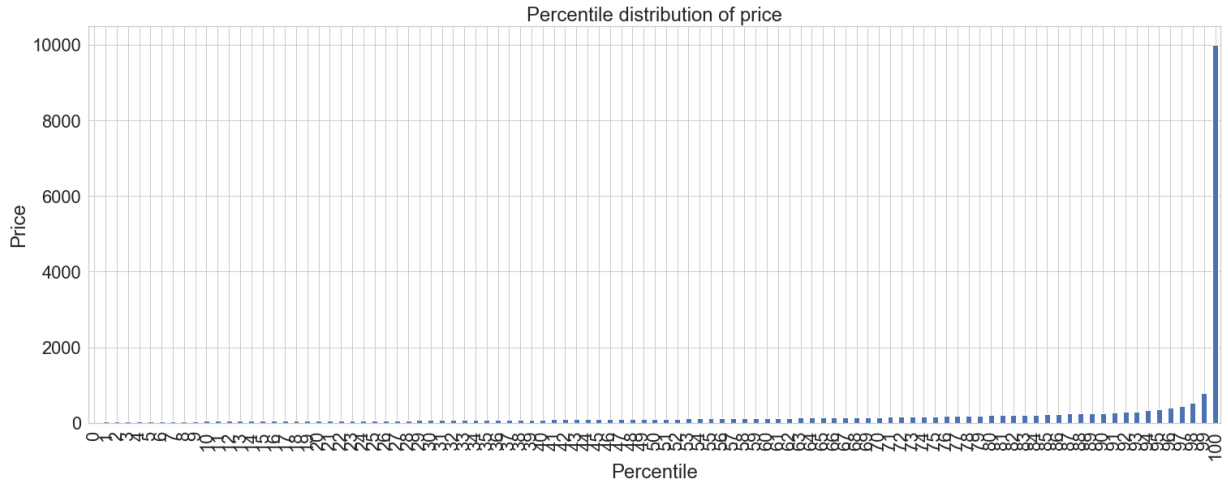
In [154]:
```
data_path = os.path.join(os.getcwd(), "../data/cleaned_listings_with_outlier
listings = pd.read_csv(data_path, index_col="id")
display(listings.shape)
```

```
(50220, 65)
```

Plot the distribution

In [155]:
```
percentiles = list(range(0,101, 1))
price_percentile = {}
for p in percentiles:
    price_percentile[p] = np.percentile(listings['price'].values, p)

price_percentile = pd.DataFrame.from_dict(price_percentile, orient='index')
price_percentile.plot(kind='bar', figsize=(25,9), grid=True, legend=False)
plt.title("Percentile distribution of price")
plt.xlabel("Percentile")
plt.ylabel("Price")
```

Out[155]:
```
Text(0, 0.5, 'Price')
```



In [156]:
```
f, ax = plt.subplots(1,1,figsize=(25,9))
g = sns.distplot(a=listings["price"], kde=False)
t = g.set_title("price distribution")
```

Round the prices to multiples of 5

In [157]:
```
def roundto(row, base=5):
    return int(base * round(float(row) / base))
listings["price"] = listings["price"].apply(roundto)
```

Train Test Split

In [158]:
```
ycol = ["price"]
xcol = [i for i in listings.columns if i not in ycol]

x = listings[xcol].values
y = listings[ycol].values

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, ra
```

Prediction

In [159]:
```
# Quick placeholder variable
rmse_dict = {}
rmse_m_dict = {}
```

a. Ridge Regression

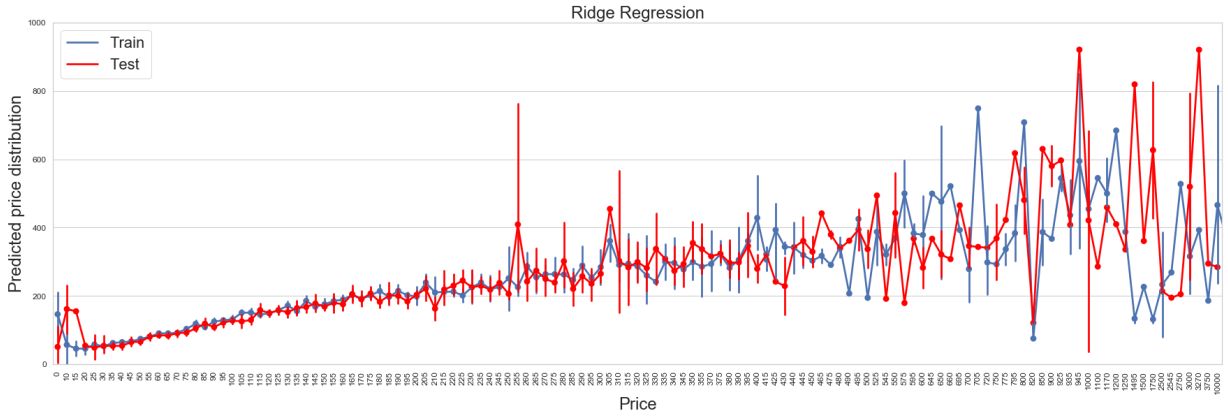
In [160]:
```
ridge = Ridge()
ridge.fit(X=x_train, y=y_train)
y_pred_train = ridge.predict(X=x_train)
y_pred_test = ridge.predict(X=x_test)

mse_train = MSE(y_train, y_pred_train)
mse_test = MSE(y_test, y_pred_test)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)
rmse_dict["Ridge"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE: {:.2f}".format(rmse_train))
print("Test set RMSE: {:.2f}".format(rmse_test))
```

```
Train set RMSE: 189.67
Test set RMSE: 223.51
```

In [161]:
```
th = price_percentile.iloc[80, :].values[0]

tmpdf = pd.DataFrame({"y_train":y_train.ravel(), "y_pred_train":y_pred_train
tmpdf = tmpdf[tmpdf["y_train"] <= th]

tmpdf2 = pd.DataFrame({"y_test":y_test.ravel(), "y_pred_test":y_pred_test.ra
tmpdf2 = tmpdf2[tmpdf2["y_test"] <= th]

mse_train = MSE(tmpdf["y_train"].values, tmpdf["y_pred_train"].values)
mse_test = MSE(tmpdf2["y_test"].values, tmpdf2["y_pred_test"].values)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)

rmse_m_dict["Ridge"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE Scaled: {:.2f}".format(rmse_train))
print("Test set RMSE Scaled: {:.2f}".format(rmse_test))
```

```
Train set RMSE Scaled: 62.22
Test set RMSE Scaled: 61.92
```

In [162]:
```
f, ax = plt.subplots(1,1, figsize=(30, 9), sharex=False)
# g = sns.pointplot(x=np.unique(y_train), y=np.unique(y_train), color="light
g = sns.pointplot(x=y_train.ravel()[::5], y=y_pred_train.ravel()[::5], ax=ax
g = sns.pointplot(x=y_test.ravel()[::5], y=y_pred_test.ravel()[::5], ax=ax,
t = g.set_xlabel("Price")
t = g.set_ylabel("Predicted price distribution")
t = g.set_xticklabels(g.get_xticklabels(), rotation=90)
t = g.set_title("Ridge Regression")
yl = g.set_ylim(0, 1000)
l = ax.legend(handles=ax.lines[::len(np.unique(y_train))+1], labels=["Train"
t = g.tick_params(labelsize=12)

```



In [164]:
```
# Use cross-validation on Train data

CV_scores = - cv.cross_val_score(ridge, x_train, y_train, scoring='neg_mean_

# Compute the 10-folds CV
CV = CV_scores.mean()**(1/2)

# Print Train CV accuracy
print('Cross Validation RMSE: {:.2f}'.format(CV))
```

```
Cross Validation RMSE: 190.35
```

b. Lasso Regression

In [165]:
```
lasso = Lasso()
lasso.fit(X=x_train, y=y_train)
y_pred_train = lasso.predict(X=x_train)
y_pred_test = lasso.predict(X=x_test)

mse_train = MSE(y_train, y_pred_train)
mse_test = MSE(y_test, y_pred_test)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)
rmse_dict["lasso"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE: {:.2f}".format(rmse_train))
print("Test set RMSE: {:.2f}".format(rmse_test))
```

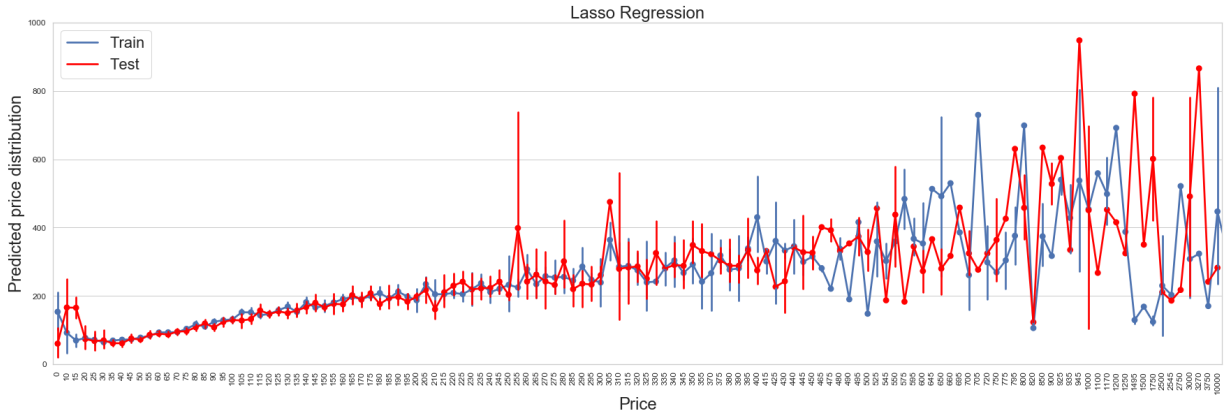
```
Train set RMSE: 190.65
Test set RMSE: 224.78
```

In [166]:
```
th = price_percentile.iloc[80, :].values[0]

tmpdf = pd.DataFrame({"y_train":y_train.ravel(), "y_pred_train":y_pred_train
tmpdf = tmpdf[tmpdf["y_train"] <= th]

tmpdf2 = pd.DataFrame({"y_test":y_test.ravel(), "y_pred_test":y_pred_test.ra
tmpdf2 = tmpdf2[tmpdf2["y_test"] <= th]

mse_train = MSE(tmpdf["y_train"].values, tmpdf["y_pred_train"].values)
mse_test = MSE(tmpdf2["y_test"].values, tmpdf2["y_pred_test"].values)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)

rmse_m_dict["lasso"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE Scaled: {:.2f}".format(rmse_train))
print("Test set RMSE Scaled: {:.2f}".format(rmse_test))
```

```
Train set RMSE Scaled: 61.02
Test set RMSE Scaled: 61.24
```

In [167]:

```
f, ax = plt.subplots(1,1, figsize=(30, 9), sharex=False)
# g = sns.pointplot(x=np.unique(y_train), y=np.unique(y_train), color="light
g = sns.pointplot(x=y_train.ravel()[::5], y=y_pred_train.ravel()[::5], ax=ax
g = sns.pointplot(x=y_test.ravel()[::5], y=y_pred_test.ravel()[::5], ax=ax,
t = g.set_xlabel("Price")
t = g.set_ylabel("Predicted price distribution")
t = g.set_xticklabels(g.get_xticklabels(), rotation=90)
t = g.set_title("Lasso Regression")
yl = g.set_ylim(0, 1000)
l = ax.legend(handles=ax.lines[::len(np.unique(y_train))+1], labels=["Train"
t = g.tick_params(labelsize=12)

```



In [169]:
```
# Use cross-validation on Train data

CV_scores = - cv.cross_val_score(lasso, x_train, y_train, scoring='neg_mean_

# Compute the 10-folds CV
CV = CV_scores.mean()**(1/2)

# Print Train CV accuracy
print('Cross Validation RMSE: {:.2f}'.format(CV))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-169-dabc9a0c5edd> in <module>
      1 # Use cross-validation on Train data
      2
----> 3 CV_scores = - cv.cross_val_score(lasso, x_train, y_train, scoring='neg_
mean_squared_error', cv=10, n_jobs=6)
      4
      5 # Compute the 10-folds CV

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\site-packages\s
klearn\model_selection\_validation.py in cross_val_score(estimator, X, y, group
s, scoring, cv, n_jobs, verbose, fit_params, pre_dispatch, error_score)
    400                                 fit_params=fit_params,
    401                                 pre_dispatch=pre_dispatch,
--> 402                                 error_score=error_score)
    403     return cv_results['test_score']
    404

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\site-packages\s
klearn\model_selection\_validation.py in cross_validate(estimator, X, y, group
s, scoring, cv, n_jobs, verbose, fit_params, pre_dispatch, return_train_score,
 return_estimator, error_score)
    238             return_times=True, return_estimator=return_estimator,
    239             error_score=error_score)
--> 240         for train, test in cv.split(X, y, groups))
    241
    242     zipped_scores = list(zip(*scores))

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\site-packages\s
klearn\externals\joblib\parallel.py in __call__(self, iterable)
    928
    929             with self._backend.retrieval_context():
--> 930                 self.retrieve()
    931             # Make sure that we get a last message telling us we are do
ne
    932             elapsed_time = time.time() - self._start_time

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\site-packages\s
klearn\externals\joblib\parallel.py in retrieve(self)
    831             try:
    832                 if getattr(self._backend, 'supports_timeout', False):
--> 833                     self._output.extend(job.get(timeout=self.timeout))
    834                 else:
    835                     self._output.extend(job.get())

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\site-packages\s
```

```
klearn\externals\joblib\_parallel_backends.py in wrap_future_result(future, tim
eout)
    519         AsyncResults.get from multiprocessing."""
    520         try:
--> 521             return future.result(timeout=timeout)
    522         except LokyTimeoutError:
    523             raise TimeoutError()

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\concurrent\futu
res\_base.py in result(self, timeout)
    425                 return self.__get_result()
    426
--> 427             self._condition.wait(timeout)
    428
    429             if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

c:\users\sriharis\appdata\local\programs\python\python37-32\lib\threading.py in
wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterrup
t)
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:

KeyboardInterrupt:
```

c. Decision Trees

In [ ]:

```
dtr = DecisionTreeRegressor()
dtr.fit(X=x_train, y=y_train)
y_pred_train = dtr.predict(X=x_train)
y_pred_test = dtr.predict(X=x_test)

mse_train = MSE(y_train, y_pred_train)
mse_test = MSE(y_test, y_pred_test)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)
rmse_dict["dt"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE: {:.2f}".format(rmse_train))
print("Test set RMSE: {:.2f}".format(rmse_test))
```

In [ ]:
```
th = price_percentile.iloc[80, :].values[0]

tmpdf = pd.DataFrame({"y_train":y_train.ravel(), "y_pred_train":y_pred_train
tmpdf = tmpdf[tmpdf["y_train"] <= th]

tmpdf2 = pd.DataFrame({"y_test":y_test.ravel(), "y_pred_test":y_pred_test.ra
tmpdf2 = tmpdf2[tmpdf2["y_test"] <= th]

mse_train = MSE(tmpdf["y_train"].values, tmpdf["y_pred_train"].values)
mse_test = MSE(tmpdf2["y_test"].values, tmpdf2["y_pred_test"].values)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)

rmse_m_dict["dt"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE Scaled: {:.2f}".format(rmse_train))
print("Test set RMSE Scaled: {:.2f}".format(rmse_test))
```

In [ ]:
```
f, ax = plt.subplots(1,1, figsize=(30, 9), sharex=False)
# g = sns.pointplot(x=np.unique(y_train), y=np.unique(y_train), color="light
g = sns.pointplot(x=y_train.ravel()[::5], y=y_pred_train.ravel()[::5], ax=ax
g = sns.pointplot(x=y_test.ravel()[::5], y=y_pred_test.ravel()[::5], ax=ax,
t = g.set_xlabel("Price")
t = g.set_ylabel("Predicted price distribution")
t = g.set_xticklabels(g.get_xticklabels(), rotation=90)
t = g.set_title("Decision Trees")
yl = g.set_ylim(0, 3000)
l = ax.legend(handles=ax.lines[::len(np.unique(y_train))+1], labels=["Train"
t = g.tick_params(labelsize=12)

```

Feature importance

In [ ]:
```
x_final = listings.drop(['price'], axis=1)

imp = pd.DataFrame({'Input Variable': x_final.columns, 'Importance (%)': -np
top20 = imp.iloc[:20,:]

f, ax = plt.subplots(1,1,figsize=(20,9))
g = sns.barplot(x="Importance (%)", y="Input Variable", data=top20, ax=ax)
t = g.set_xlabel("Importance (%)")
t = g.set_ylabel("Input Variable")
t = g.set_title("Feature Importance")
```

In [ ]:
```
# Use cross-validation on Train data

CV_scores = - cv.cross_val_score(dtr, x_train, y_train, scoring='neg_mean_sq

# Compute the 10-folds CV
CV = CV_scores.mean()**(1/2)

# Print Train CV accuracy
print('Cross Validation RMSE: {:.2f}'.format(CV))
```

d. Random forests

In [ ]:
```
rfr = RandomForestRegressor()
rfr.fit(X=x_train, y=y_train)
y_pred_train = rfr.predict(X=x_train)
y_pred_test = rfr.predict(X=x_test)

mse_train = MSE(y_train, y_pred_train)
mse_test = MSE(y_test, y_pred_test)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)
rmse_dict["rf"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE: {:.2f}".format(rmse_train))
print("Test set RMSE: {:.2f}".format(rmse_test))
```

In [ ]:
```
th = price_percentile.iloc[80, :].values[0]

tmpdf = pd.DataFrame({"y_train":y_train.ravel(), "y_pred_train":y_pred_train
tmpdf = tmpdf[tmpdf["y_train"] <= th]

tmpdf2 = pd.DataFrame({"y_test":y_test.ravel(), "y_pred_test":y_pred_test.ra
tmpdf2 = tmpdf2[tmpdf2["y_test"] <= th]

mse_train = MSE(tmpdf["y_train"].values, tmpdf["y_pred_train"].values)
mse_test = MSE(tmpdf2["y_test"].values, tmpdf2["y_pred_test"].values)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)

rmse_m_dict["rf"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE Scaled: {:.2f}".format(rmse_train))
print("Test set RMSE Scaled: {:.2f}".format(rmse_test))
```

In [ ]:
```
f, ax = plt.subplots(1,1, figsize=(30, 9), sharex=False)
# g = sns.pointplot(x=np.unique(y_train), y=np.unique(y_train), color="light
g = sns.pointplot(x=y_train.ravel()[::5], y=y_pred_train.ravel()[::5], ax=ax
g = sns.pointplot(x=y_test.ravel()[::5], y=y_pred_test.ravel()[::5], ax=ax,
t = g.set_xlabel("Price")
t = g.set_ylabel("Predicted price distribution")
t = g.set_xticklabels(g.get_xticklabels(), rotation=90)
t = g.set_title("Random Forest")
yl = g.set_ylim(0, 2000)
l = ax.legend(handles=ax.lines[::len(np.unique(y_train))+1], labels=["Train"
t = g.tick_params(labelsize=12)

```

Feature importance

In [ ]:
```
x_final = listings.drop(['price'], axis=1)

imp = pd.DataFrame({'Input Variable': x_final.columns, 'Importance (%)': -np
top20 = imp.iloc[:20,:]

f, ax = plt.subplots(1,1,figsize=(20,9))
g = sns.barplot(x="Importance (%)", y="Input Variable", data=top20, ax=ax)
t = g.set_xlabel("Importance (%)")
t = g.set_ylabel("Input Variable")
t = g.set_title("Feature Importance")
```

In [ ]:
```
# Use cross-validation on Train data

CV_scores = - cv.cross_val_score(rfr, x_train, y_train, scoring='neg_mean_sq

# Compute the 10-folds CV
CV = CV_scores.mean()**(1/2)

# Print Train CV accuracy
print('Cross Validation RMSE: {:.2f}'.format(CV))
```

e. Gradient Boosting regression

In [ ]:
```
gbr = GradientBoostingRegressor()
gbr.fit(X=x_train, y=y_train)
y_pred_train = gbr.predict(X=x_train)
y_pred_test = gbr.predict(X=x_test)

mse_train = MSE(y_train, y_pred_train)
mse_test = MSE(y_test, y_pred_test)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)
rmse_dict["gbr"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE: {:.2f}".format(rmse_train))
print("Test set RMSE: {:.2f}".format(rmse_test))
```

In [ ]:
```
th = price_percentile.iloc[80, :].values[0]

tmpdf = pd.DataFrame({"y_train":y_train.ravel(), "y_pred_train":y_pred_train
tmpdf = tmpdf[tmpdf["y_train"] <= th]

tmpdf2 = pd.DataFrame({"y_test":y_test.ravel(), "y_pred_test":y_pred_test.ra
tmpdf2 = tmpdf2[tmpdf2["y_test"] <= th]

mse_train = MSE(tmpdf["y_train"].values, tmpdf["y_pred_train"].values)
mse_test = MSE(tmpdf2["y_test"].values, tmpdf2["y_pred_test"].values)

rmse_train = mse_train**(1/2)
rmse_test = mse_test**(1/2)

rmse_m_dict["gbr"] = {"Train":rmse_train,"Test":rmse_test}

print("Train set RMSE Scaled: {:.2f}".format(rmse_train))
print("Test set RMSE Scaled: {:.2f}".format(rmse_test))
```

In [ ]:
```
f, ax = plt.subplots(1,1, figsize=(30, 9), sharex=False)
# g = sns.pointplot(x=np.unique(y_train), y=np.unique(y_train), color="light
g = sns.pointplot(x=y_train.ravel()[::5], y=y_pred_train.ravel()[::5], ax=ax
g = sns.pointplot(x=y_test.ravel()[::5], y=y_pred_test.ravel()[::5], ax=ax,
t = g.set_xlabel("Price")
t = g.set_ylabel("Predicted price distribution")
t = g.set_xticklabels(g.get_xticklabels(), rotation=90)
t = g.set_title("Gradient Boosting Regression")
yl = g.set_ylim(0, 2500)
l = ax.legend(handles=ax.lines[::len(np.unique(y_train))+1], labels=["Train"
t = g.tick_params(labelsize=12)

```

Feature importance

In [ ]:
```
x_final = listings.drop(['price'], axis=1)

imp = pd.DataFrame({'Input Variable': x_final.columns, 'Importance (%)': -np
top20 = imp.iloc[:20,:]

f, ax = plt.subplots(1,1,figsize=(20,9))
g = sns.barplot(x="Importance (%)", y="Input Variable", data=top20, ax=ax)
t = g.set_xlabel("Importance (%)")
t = g.set_ylabel("Input Variable")
t = g.set_title("Feature Importance")
```

In [ ]:
```
# Use cross-validation on Train data

CV_scores = - cv.cross_val_score(gbr, x_train, y_train, scoring='neg_mean_sq

# Compute the 10-folds CV
CV = CV_scores.mean()**(1/2)

# Print Train CV accuracy
print('Cross Validation RMSE: {:.2f}'.format(CV))
```

Final dataframe to plot the RMSEs

In [ ]:
```
rmse_df = pd.DataFrame(rmse_dict)
rmse_df = rmse_df.T.reset_index(drop=False)
rmse_df.columns = ["Regressor", "Test RMSE", "Train RMSE"]
rmse_df
```

In [ ]:
```
rmse_m_df = pd.DataFrame(rmse_m_dict)
rmse_m_df = rmse_m_df.T.reset_index(drop=False)
rmse_m_df.columns = ["Regressor", "Test RMSE", "Train RMSE"]
rmse_m_df
```

In [ ]:

```
def plot_bar(data, x, y, ax, hue=None, title="", xlabel="", ylabel="",
             xrot=0, yrot=0, highlight_max_min=True,
             plot_percentiles=[], plot_mean=True,
             point_plot=False, annot=True, legend=False):
    if highlight_max_min:
        clrs = []
        for v in data[y].values:
            if v < data[y].max():
                if v > data[y].min():
                    clrs.append('lightblue')
                else:
                    clrs.append('lightgreen')
            else:
                clrs.append('darksalmon')
        g = sns.barplot(x=x, y=y, data=data, ax=ax, palette=clrs)
    else:
        g = sns.barplot(x=x, y=y, data=data, ax=ax, hue=hue)

    if len(plot_percentiles) > 0:
        for p in plot_percentiles:
            v = np.percentile(data[y].values, p)
            plt.axhline(v, 1, 0, color='grey').set_linestyle("--")

    if plot_mean:
        v = data[y].mean()
        plt.axhline(v, 1, 0, color='k').set_linestyle("--")

    if point_plot:
        g1 = sns.pointplot(x=x, y=y, data=data, ax=ax, color="darkblue")
    if xrot != 0:
        g.set_xticklabels(rotation=xrot, labels=g.get_xticklabels())
    if yrot != 0:
        g.set_yticklabels(rotation=yrot, labels=g.get_yticklabels())
    if annot:
        # Add labels to the plot
        style = dict(size=12, color='darkblue')
        s1 = data[y].values
        counter = 0
        for idx, row in data.iterrows():
            rx, ry = row[x], row[y]
            if type('str') == type(idx):
                ax.text(counter, ry, str(np.round(ry, 2)),
                    **style, va="bottom", ha='right')
            else:
                ax.text(idx*0.99, ry, str(np.round(s1[idx], 2)),
                        **style, va="bottom", ha='right')
            counter += 1
    g.set(xlabel=xlabel, ylabel=ylabel, title=title)
    ax.set_ylim([0, data[y].max() * 1.2])
    if legend:
        ax.legend(handles=ax.lines[::len(data) + 1], labels=[y])

```

In [ ]:
```
f, ax = plt.subplots(1,2,figsize=(20,8), sharey=False)
# g = sns.barplot(x="Regressor", y="Train RMSE", data=rmse_df, ax=ax[0])
# t = g.set(title="Train RMSE", ylabel="RMSE", xlabel="")
# g = sns.barplot(x="Regressor", y="Test RMSE", data=rmse_df, ax=ax[1])
# t = g.set(title="Test RMSE", ylabel="RMSE", xlabel="")

g = plot_bar(x="Regressor", y="Train RMSE", data=rmse_df, ax=ax[0], plot_mea
g = plot_bar(x="Regressor", y="Test RMSE", data=rmse_df, ax=ax[1], plot_mean

```

In [ ]:
```
f, ax = plt.subplots(1,2,figsize=(20,8), sharey=False)
# g = sns.barplot(x="Regressor", y="Train RMSE", data=rmse_m_df, ax=ax[0])
# t = g.set(title="Train RMSE", ylabel="RMSE", xlabel="")
# g = sns.barplot(x="Regressor", y="Test RMSE", data=rmse_m_df, ax=ax[1])
# t = g.set(title="Test RMSE", ylabel="RMSE", xlabel="")

g = plot_bar(x="Regressor", y="Train RMSE", data=rmse_m_df, ax=ax[0], plot_m
g = plot_bar(x="Regressor", y="Test RMSE", data=rmse_m_df, ax=ax[1], plot_me
```

In [ ]:
```

```