#Setup

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold,
RandomizedSearchCV, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, make_scorer

sns.set(style="whitegrid", font_scale=1.1)

df = pd.read_excel("New York Housing Data.xlsx")
df.shape
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 4801,\n  \"fields\":
[\n    {\n      \"column\": \"BROKERTITLE\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\":
1036,\n        \"samples\": [\n          \"Brokered by LINCOLN V
WALTERS II REALTY CORP\",\n          \"Brokered by East Coast REALTORS
Inc\",\n          \"Brokered by Welhome Realty Inc\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TYPE\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 13,\n
\"samples\": [\n          \"Mobile house for sale\",\n
\"Pending\",\n          \"Condo for sale\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PRICE\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 31355246,\n
\"min\": 2494,\n        \"max\": 2147483647,\n
\"num_unique_values\": 1274,\n        \"samples\": [\n
49500,\n          755000,\n          699998\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BEDS\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 50,\n        \"num_unique_values\": 27,\n        \"samples\":
[\n          12,\n          14,\n          10\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BATH\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.946961630597847,\n
\"min\": 0.0,\n        \"max\": 50.0,\n        \"num_unique_values\":
22,\n        \"samples\": [\n          2.0,\n          13.0,\n
```

```
8.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PROPERTYSQFT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2377.1408935597533,\n        \"min\":
230.0,\n        \"max\": 65535.0,\n        \"num_unique_values\":
1445,\n        \"samples\": [\n          2472.0,\n          1445.0,\n
1868.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ADDRESS\",\n      \"properties\": {\n        \"dtype\": \"string\",\
n        \"num_unique_values\": 4582,\n        \"samples\": [\n
\"3671 Hudson Manor Ter Apt 14J\",\n          \"150 W End Ave Apt
5H\",\n          \"220 E 65th St Apt 20B\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"STATE\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\":
308,\n        \"samples\": [\n          \"Jackson Heights, NY
11370\",\n          \"Brooklyn, NY 11230\",\n          \"Manhattan, NY
10010\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MAIN_ADDRESS\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 4583,\n
\"samples\": [\n          \"111 Sunset BlvdBronx, NY 10473\",\n
\"21-16 35th St Unit 1BAstoria, NY 11105\",\n          \"104-02 223rd
StQueens Village, NY 11429\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ADMINISTRATIVE_AREA_LEVEL_2\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 29,\
n        \"samples\": [\n          11237,\n          \"Queens
County\",\n          11417\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"LOCALITY\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 11,\n
\"samples\": [\n          \"Queens County\",\n          \"New York\",\
n          \"Queens\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SUBLOCALITY\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 21,\n
\"samples\": [\n          \"Manhattan\",\n          \"Fort
Hamilton\",\n          \"Riverdale\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"STREET_NAME\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 174,\n        \"samples\": [\n          \"John
Street\",\n          \"Shore Road\",\n          \"Mill Basin\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"LONG_NAME\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2731,\n        \"samples\": [\n
\"205th Place\",\n          \"83-52\",\n          1530\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"FORMATTED_ADDRESS\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4550,\n        \"samples\": [\n          \"4410
Cayuga Ave Bsmt B3, Bronx, NY 10471, USA\",\n          \"270-10 Grand
Central Pkwy #25k, Little Neck, NY 11362, USA\",\n          \"165
Cromwell Ave #2b, Staten Island, NY 10304, USA\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"LATITUDE\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.08767556707715027,\n        \"min\": 40.4995462,\n        \"max\":
40.9127295,\n        \"num_unique_values\": 4196,\n
\"samples\": [\n          40.782238,\n          40.6180243,\n
40.7121557\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"LONGITUDE\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.10108248150173928,\n        \"min\": -
74.2530332,\n        \"max\": -73.70245,\n
\"num_unique_values\": 4118,\n        \"samples\": [\n          -
73.9071739,\n          -74.017865,\n          -74.0802854\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


##IQR Filter

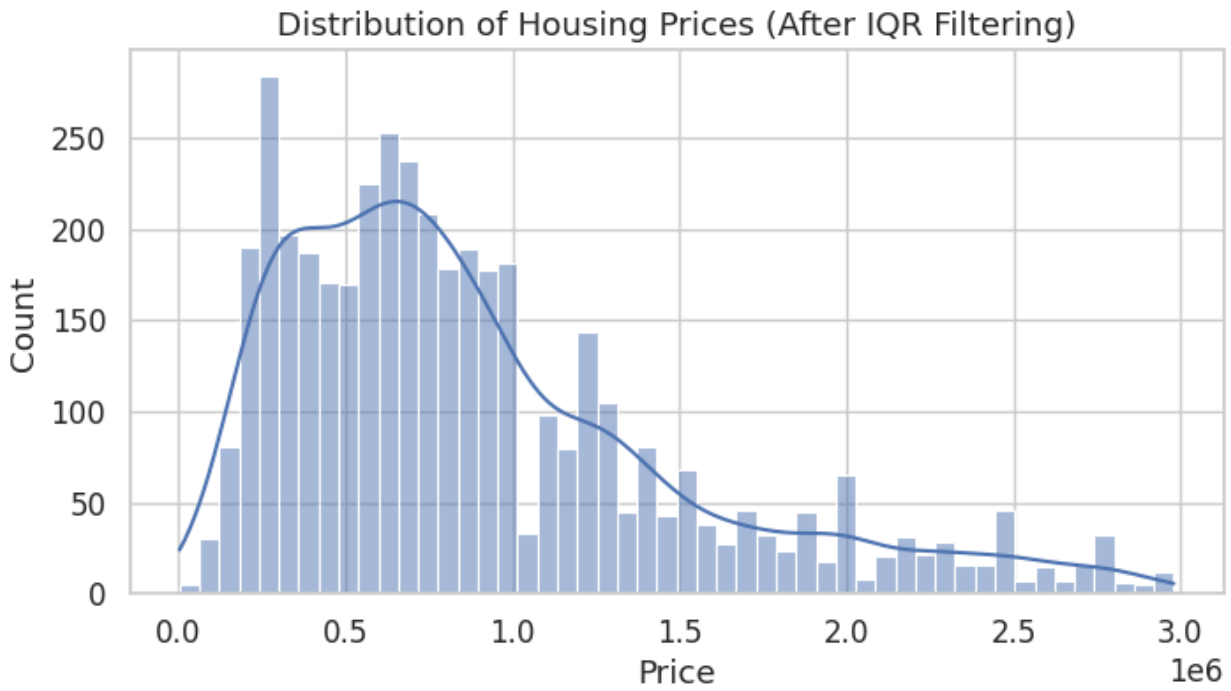
```
Q1 = df['PRICE'].quantile(0.25)
Q3 = df['PRICE'].quantile(0.75)
IQR = Q3 - Q1
upper_limit = Q3 + 1.5 * IQR

print(f"Q1: {Q1:,.0f}")
print(f"Q3: {Q3:,.0f}")
print(f"IQR: {IQR:,.0f}")
print(f"Upper limit (Q3 + 1.5*IQR): {upper_limit:,.0f}")

outliers = df[df['PRICE'] > upper_limit]
print(f"\nNumber of high-price outliers: {len(outliers)}")
print(f"Percentage of dataset: {len(outliers) / len(df) * 100:.2f}%")

df_iqr = df[df['PRICE'] <= upper_limit].copy()
print("\nAfter IQR filter:", df_iqr.shape)
```

```
Q1: 499,000
Q3: 1,495,000
IQR: 996,000
Upper limit (Q3 + 1.5*IQR): 2,989,000

Number of high-price outliers: 559
Percentage of dataset: 11.64%
```

```
After IQR filter: (4242, 17)
```

##Plots

```
plt.figure(figsize=(10,4))
sns.boxplot(x=df_iqr['PRICE'])
plt.title("Boxplot of Housing Prices (After IQR Filtering)")
plt.xlabel("Price")
plt.show()

plt.figure(figsize=(8,4))
sns.histplot(df_iqr['PRICE'], bins=50, kde=True)
plt.title("Distribution of Housing Prices (After IQR Filtering)")
plt.xlabel("Price")
plt.ylabel("Count")
plt.show()
```



##Clean (Property Type)

```
df_iqr['TYPE'].value_counts()
```

```
TYPE
Co-op for sale                 1347
House for sale                  946
Condo for sale                  728
Multi-family home for sale      673
Pending                         229
Townhouse for sale              152
Contingent                       87
Land for sale                    44
For sale                         15
Foreclosure                      13
Condop for sale                   5
Coming Soon                       2
Mobile house for sale             1
Name: count, dtype: int64
```

```
def clean_property_type(x):
    x = str(x).lower().strip()

    #keep
    type_patterns = [
        ("condo", "Condo"),
```

```
        ("co-op", "Co-op"),
        ("multi", "Multi-family"),
        ("townhouse", "Townhouse"),
        ("house", "House"),
    ]
    for pattern, label in type_patterns:
        if pattern in x:
            return label

    #remove
    removal_keywords = [
        "land",
        "foreclosure",
        "mobile",
        "pending",
        "contingent",
        "coming",
    ]
    if any(word in x for word in removal_keywords) or x == "for sale":
        return None
    return "Other"


df_iqr["TYPE_CLEAN"] = df_iqr["TYPE"].apply(clean_property_type)
df_clean = df_iqr.dropna(subset=["TYPE_CLEAN"]).copy()

print("After cleaning TYPE:", df_clean.shape)
df_clean["TYPE_CLEAN"].value_counts()
```

```
After cleaning TYPE: (3852, 18)
```

```
TYPE_CLEAN
Co-op           1347
House            947
Condo            733
Multi-family     673
Townhouse        152
Name: count, dtype: int64
```

##Borough

```
df_iqr['SUBLOCALITY'].value_counts()
```

```
SUBLOCALITY
New York            811
Queens County       667
Kings County        643
Queens              555
Richmond County     472
Brooklyn            450
```

```
Bronx County         295
The Bronx            184
New York County       77
Staten Island         59
Manhattan             12
Riverdale              4
Flushing               4
Coney Island           3
East Bronx             1
Brooklyn Heights       1
Jackson Heights        1
Rego Park              1
Fort Hamilton          1
Snyder Avenue          1
Name: count, dtype: int64
```

```python
def map_borough(x):
    x = str(x).strip().lower()

    borough_keywords = {
        "Manhattan": ["manhattan", "new york county", "new york"],
        "Brooklyn": [
            "brooklyn", "kings county", "dumbo", "coney island", "fort
hamilton",
            "brooklyn heights", "bushwick", "williamsburg", "snyder
avenue"
        ],
        "Queens": [
            "queens", "queens county", "flushing", "rego park",
"jackson heights",
            "astoria", "long island city"
        ],
        "Bronx": [
            "bronx", "bronx county", "the bronx", "riverdale", "east
bronx", "mott haven"
        ],
        "Staten Island": ["staten island", "richmond county"]
    }

    # Loop through dictionary instead of multiple if blocks
    for borough, keywords in borough_keywords.items():
        if any(k in x for k in keywords):
            return borough

    return "Other"


df_clean['BOROUGH'] = df_clean['SUBLOCALITY'].apply(map_borough)
df_clean['BOROUGH'].value_counts()
```

```
BOROUGH
Queens           1148
Brooklyn         1002
Manhattan         827
Bronx             459
Staten Island     416
Name: count, dtype: int64
```

##Drop Unused/High Cardinality

```
columns_to_drop = [
    'STATE',
    'MAIN_ADDRESS',
    'ADMINISTRATIVE_AREA_LEVEL_2',
    'LOCALITY',
    'STREET_NAME',
    'LONG_NAME',
    'FORMATTED_ADDRESS',
    'TYPE',           # before clean type
    'SUBLOCALITY',    # changed (BOROUGH)
    'BROKERTITLE'     # HighCard
]

df_clean = df_clean.drop(columns=[c for c in columns_to_drop if c in
df_clean.columns])

print("Columns after drop:")
print(df_clean.columns)
print("Shape after cleaning:", df_clean.shape)
```

```
Columns after drop:
Index(['PRICE', 'BEDS', 'BATH', 'PROPERTYSQFT', 'ADDRESS', 'LATITUDE',
       'LONGITUDE', 'TYPE_CLEAN', 'BOROUGH'],
      dtype='object')
Shape after cleaning: (3852, 9)
```

##One-Hot Encoding

```
categorical_cols = ['TYPE_CLEAN', 'BOROUGH']
df_encoded = pd.get_dummies(df_clean, columns=categorical_cols,
drop_first=True, dtype=int)

print(df_encoded.columns)
print('\n')
print("Encoded shape:", df_encoded.shape)
df_encoded.head()
```

```
Index(['PRICE', 'BEDS', 'BATH', 'PROPERTYSQFT', 'ADDRESS', 'LATITUDE',
       'LONGITUDE', 'TYPE_CLEAN_Condo', 'TYPE_CLEAN_House',
```

```
       'TYPE_CLEAN_Multi-family', 'TYPE_CLEAN_Townhouse',
'BOROUGH_Brooklyn',
       'BOROUGH_Manhattan', 'BOROUGH_Queens', 'BOROUGH_Staten
Island'],
      dtype='object')


Encoded shape: (3852, 15)
```


```
{"summary":"{\n  \"name\": \"df_encoded\",\n  \"rows\": 3852,\n
\"fields\": [\n    {\n      \"column\": \"PRICE\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
613000,\n        \"min\": 49500,\n        \"max\": 2980000,\n
\"num_unique_values\": 954,\n        \"samples\": [\n
1369000,\n          2300000,\n          399888\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BEDS\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 24,\n        \"num_unique_values\": 20,\n        \"samples\":
[\n          2,\n          24,\n          20\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BATH\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.3263755819232537,\n
\"min\": 1.0,\n        \"max\": 24.0,\n        \"num_unique_values\":
14,\n        \"samples\": [\n          7.0,\n          10.0,\n
2.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PROPERTYSQFT\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1033.590066439373,\n        \"min\":
250.0,\n        \"max\": 21000.0,\n        \"num_unique_values\":
1120,\n        \"samples\": [\n          3427.0,\n          1320.0,\n
415.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ADDRESS\",\n      \"properties\": {\n        \"dtype\": \"string\",\
n        \"num_unique_values\": 3673,\n        \"samples\": [\n
\"1250 Ocean Pkwy Apt 3J\",\n          \"90 E End Ave Apt 6B\",\n
\"136 Beach 117 St Unit 5J\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"LATITUDE\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.08968643193478688,\n        \"min\":
40.4997983,\n        \"max\": 40.9127295,\n
\"num_unique_values\": 3371,\n        \"samples\": [\n
40.7267155,\n          40.7192438,\n          40.7856236\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"LONGITUDE\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.10250502538882184,\n        \"min\": -74.2530332,\n        \"max\":
-73.70245,\n        \"num_unique_values\": 3319,\n        \"samples\":
[\n          -73.985324,\n          -73.8961042,\n          -
73.8794549\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"TYPE_CLEAN_Condo\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"TYPE_CLEAN_House\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"TYPE_CLEAN_Multi-family\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"TYPE_CLEAN_Townhouse\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BOROUGH_Brooklyn\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BOROUGH_Manhattan\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BOROUGH_Queens\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"BOROUGH_Staten Island\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_encoded"}
```


#Train/Test Setup

```
y = df_encoded['PRICE']
X = df_encoded.drop(columns=['PRICE'])
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

```
Train shape: (3081, 14)
Test shape: (771, 14)
```

##Scaling(Only for Linear Regression)

```
numeric_cols = ['BEDS', 'BATH', 'PROPERTYSQFT', 'LATITUDE',
'LONGITUDE']

scaler = StandardScaler()
scaler.fit(X_train[numeric_cols])

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[numeric_cols] = scaler.transform(X_train[numeric_cols])
X_test_scaled[numeric_cols] = scaler.transform(X_test[numeric_cols])

X_train_lr = X_train_scaled.drop(columns=['ADDRESS'])
X_test_lr = X_test_scaled.drop(columns=['ADDRESS'])
```

#Untuned Models

##Training

Linear Regression

```
lr_model = LinearRegression()
lr_model.fit(X_train_lr, y_train)
```

```
LinearRegression()
```

Randomforest Untuned

```
X_train_rf = X_train.drop(columns=['ADDRESS'])
X_test_rf = X_test.drop(columns=['ADDRESS'])

rf_model = RandomForestRegressor(
    random_state=42
)
rf_model.fit(X_train_rf, y_train)
```

```
RandomForestRegressor(random_state=42)
```

Gradient Boosting Untuned

```
gb_model = GradientBoostingRegressor(
    random_state=42
)

gb_model.fit(X_train_rf, y_train)
```

```
GradientBoostingRegressor(random_state=42)
```

XGBBoost Untuned

```
from xgboost import XGBRegressor
xgb_model = XGBRegressor(
    random_state=42,
    n_estimators = 100,
    verbosity = 0
  ,
)

xgb_model.fit(X_train_rf, y_train)
```

```
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=100,
             n_jobs=None, num_parallel_tree=None, ...)
```

LightGBM Untuned

```
from lightgbm import LGBMRegressor

lgb_model = LGBMRegressor(
    random_state=42,
    verbose=-1
)
```

```
lgb_model.fit(X_train_rf, y_train)
```

```
LGBMRegressor(random_state=42, verbose=-1)
```

CatBoost Untuned

```
#!pip install catboost
from catboost import CatBoostRegressor

cat_model = CatBoostRegressor(
    random_seed=42,
    verbose=False
)

cat_model.fit(X_train_rf, y_train)
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp312-cp312-
manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in
/usr/local/lib/python3.12/dist-packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.12/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in
/usr/local/lib/python3.12/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in
/usr/local/lib/python3.12/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in
/usr/local/lib/python3.12/dist-packages (from catboost) (1.16.3)
Requirement already satisfied: plotly in
/usr/local/lib/python3.12/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-
packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(1.3.3)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
```

```
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(25.0)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(3.2.5)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.12/dist-packages (from plotly->catboost)
(9.1.2)
Downloading catboost-1.2.8-cp312-cp312-manylinux2014_x86_64.whl (99.2
MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 6.8 MB/s eta
0:00:00

<catboost.core.CatBoostRegressor at 0x799189f52090>
```

##Cross Validation

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor

from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor

# Scorers
PRIMARY_SCORING = "neg_root_mean_squared_error"

#CV Define
kf = KFold(n_splits=5, shuffle=True, random_state=42)

def evaluate_cv(model, X, y, name):
    # Primary
    rmse_scores = -cross_val_score(model, X, y, cv=kf,
scoring="neg_root_mean_squared_error")

    # Secondary
    mae_scores  = -cross_val_score(model, X, y, cv=kf,
scoring="neg_mean_absolute_error")
```

```
    r2_scores   =  cross_val_score(model, X, y, cv=kf, scoring="r2")

    print(f"\n{name} CROSS-VALIDATION RESULTS (Primary Metric =
RMSE)")
    print("-------------------------------------------------------")
    print(f"RMSE: {rmse_scores.mean():,.2f} (+/-
{rmse_scores.std():,.2f})")
    print(f"MAE:  {mae_scores.mean():,.2f} (+/-
{mae_scores.std():,.2f})")
    print(f"R²:   {r2_scores.mean():.3f} (+/- {r2_scores.std():.3f})")


lr_cv  = LinearRegression()

rf_cv  = RandomForestRegressor(random_state=42)

gb_cv  = GradientBoostingRegressor(random_state=42)

xgb_cv = XGBRegressor(
    random_state=42,
    n_estimators=100,
    verbosity=0
)

lgb_cv = LGBMRegressor(
    random_state=42,
    verbose=-1
)

cat_cv = CatBoostRegressor(
    random_seed=42,
    verbose=False
)

# Prepare X for Linear Regression (scaled and 'ADDRESS' dropped)
X_for_lr_scaled = X.copy()
X_for_lr_scaled[numeric_cols] = scaler.transform(X[numeric_cols])
X_for_lr_scaled = X_for_lr_scaled.drop(columns=['ADDRESS'])

# Prepare X for Tree-based models (unscaled and 'ADDRESS' dropped)
X_for_trees = X.drop(columns=['ADDRESS'])

#RUN
evaluate_cv(lr_cv,  X_for_lr_scaled, y, "Linear Regression")
evaluate_cv(rf_cv,  X_for_trees,     y, "Random Forest")
evaluate_cv(gb_cv,  X_for_trees,     y, "Gradient Boosting")
evaluate_cv(xgb_cv, X_for_trees,     y, "XGBoost")
evaluate_cv(lgb_cv, X_for_trees,     y, "LightGBM")
evaluate_cv(cat_cv, X_for_trees,     y, "CatBoost")
```

```
Linear Regression CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 424,660.83 (+/- 8,903.74)
MAE:  297,587.55 (+/- 6,325.46)
R²:   0.519 (+/- 0.020)

Random Forest CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 308,634.01 (+/- 11,454.93)
MAE:  196,958.03 (+/- 6,067.62)
R²:   0.745 (+/- 0.023)

Gradient Boosting CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 330,132.25 (+/- 10,375.19)
MAE:  224,448.16 (+/- 5,005.27)
R²:   0.709 (+/- 0.015)

XGBoost CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 303,563.69 (+/- 15,097.33)
MAE:  197,072.28 (+/- 5,932.37)
R²:   0.753 (+/- 0.028)

LightGBM CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 302,181.83 (+/- 13,015.67)
MAE:  195,514.86 (+/- 5,598.02)
R²:   0.756 (+/- 0.024)

CatBoost CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 293,003.99 (+/- 8,207.33)
MAE:  191,177.59 (+/- 2,693.08)
R²:   0.771 (+/- 0.015)
```

#Untuned Models on Test(RMSE,MAE, R^2)

```python
def evaluate_on_test(name, y_true, y_pred):
    # Compute MSE first, then take square root for RMSE
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    print(f"{name} TEST RESULTS")
    print(f"RMSE: ${rmse:,.2f}")
    print(f"MAE:  ${mae:,.2f}")
```

```
    print(f"R²:   {r2:.3f}")
    return rmse, mae, r2

# LINEAR REGRESSION (scaled)
y_pred_lr = lr_model.predict(X_test_lr)
evaluate_on_test("Linear Regression (Untuned)", y_test, y_pred_lr)
print('\n')
# TREE-BASED MODELS (unscaled, drop ADDRESS)
X_test_rf = X_test.drop(columns=['ADDRESS'])
print('\n')
# Random Forest
y_pred_rf = rf_model.predict(X_test_rf)
evaluate_on_test("Random Forest (Untuned)", y_test, y_pred_rf)
print('\n')
# Gradient Boosting
y_pred_gb = gb_model.predict(X_test_rf)
evaluate_on_test("Gradient Boosting (Untuned)", y_test, y_pred_gb)
print('\n')
# XGBoost
y_pred_xgb = xgb_model.predict(X_test_rf)
evaluate_on_test("XGBoost (Untuned)", y_test, y_pred_xgb)
print('\n')
# LightGBM
y_pred_lgb = lgb_model.predict(X_test_rf)
evaluate_on_test("LightGBM (Untuned)", y_test, y_pred_lgb)
print('\n')
# CatBoost
y_pred_cat = cat_model.predict(X_test_rf)
evaluate_on_test("CatBoost (Untuned)", y_test, y_pred_cat)
```

```
Linear Regression (Untuned) TEST RESULTS
RMSE: $409,783.72
MAE:  $294,896.30
R²:   0.516
```

```
Random Forest (Untuned) TEST RESULTS
RMSE: $327,738.17
MAE:  $203,821.92
R²:   0.691
```

```
Gradient Boosting (Untuned) TEST RESULTS
RMSE: $325,824.63
MAE:  $220,779.92
R²:   0.694
```

```
XGBoost (Untuned) TEST RESULTS
RMSE: $318,387.73
MAE:  $202,538.00
R²:   0.708


LightGBM (Untuned) TEST RESULTS
RMSE: $307,857.73
MAE:  $195,476.37
R²:   0.727


CatBoost (Untuned) TEST RESULTS
RMSE: $290,146.42
MAE:  $186,206.86
R²:   0.757

(np.float64(290146.42017798603), 186206.86170152063,
0.7574325571163537)
```

#Tuned Models

##Random Forest

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import randint

#Perameters
rf_random_grid = {
    "n_estimators": randint(200, 800),
    "max_depth": randint(5, 30),                # depth range
    "min_samples_split": randint(2, 20),
    "min_samples_leaf": randint(1, 10),
    "max_features": ["sqrt", "log2", None]      # None for full
features
}

# Base Random Forest
rf_base = RandomForestRegressor(
    random_state=42,
    n_jobs=-1
)

# RandomizedSearchCV RF
rf_random_search = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=rf_random_grid,
    n_iter=50,                    #random parameter sets to try
    cv=3,
```

```
    scoring="neg_root_mean_squared_error",  #primary metric
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Fit tuned model
rf_random_search.fit(X_train_rf, y_train)

print("Best Random Forest Parameters (RandomizedSearchCV):")
print(rf_random_search.best_params_)

# Extract
rf_tuned = rf_random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best Random Forest Parameters (RandomizedSearchCV):
{'max_depth': 19, 'max_features': 'sqrt', 'min_samples_leaf': 1,
'min_samples_split': 8, 'n_estimators': 720}
```

##Gradient Boosting Tuned

```
from scipy.stats import uniform, randint
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV


# Parameters
gb_random_grid = {
    "n_estimators": randint(200, 800),          # boosting stages
    "learning_rate": uniform(0.01, 0.2),
    "max_depth": randint(2, 6),                 # depth
    "min_samples_split": randint(2, 30),
    "min_samples_leaf": randint(1, 20),
    "subsample": uniform(0.6, 0.4)
}

# Base GBM model
gb_base = GradientBoostingRegressor(
    random_state=42
)

# RandomizedSearchCV GBM
gb_random_search = RandomizedSearchCV(
    estimator=gb_base,
    param_distributions=gb_random_grid,
    n_iter=50,                    # number of random parameter sets
to try
    cv=3,
    scoring="neg_root_mean_squared_error",  # primary metric
```

```
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# tuned GBM
gb_random_search.fit(X_train_rf, y_train)

print("Best Gradient Boosting Parameters (RandomizedSearchCV):")
print(gb_random_search.best_params_)

# 5. Extract
gb_tuned = gb_random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best Gradient Boosting Parameters (RandomizedSearchCV):
{'learning_rate': np.float64(0.045422135881409795), 'max_depth': 5,
'min_samples_leaf': 14, 'min_samples_split': 22, 'n_estimators': 717,
'subsample': np.float64(0.7480634801021777)}
```

##XGBoost Tuned

```
from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Parameters
xgb_random_grid = {
    "n_estimators": randint(200, 800),          # number of boosting
rounds
    "learning_rate": uniform(0.01, 0.2),        # ~0.01–0.21
    "max_depth": randint(3, 10),                # tree depth
    "subsample": uniform(0.6, 0.4),             # 0.6–1.0
    "colsample_bytree": uniform(0.6, 0.4),      # 0.6–1.0
    "min_child_weight": randint(1, 10),         # minimum sum Hessian
in a child
    "gamma": uniform(0, 1.0)                    # minimum loss
reduction
}

# Base XGBoost model
xgb_base = XGBRegressor(
    objective="reg:squarederror",
    random_state=42,
    n_jobs=-1
)

# RandomizedSearchCV XGBoost
xgb_random_search = RandomizedSearchCV(
    estimator=xgb_base,
```

```
    param_distributions=xgb_random_grid,
    n_iter=50,
    cv=3,
    scoring="neg_root_mean_squared_error",
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Tuned XGBoost
xgb_random_search.fit(X_train_rf, y_train)

print("Best XGBoost Parameters (RandomizedSearchCV):")
print(xgb_random_search.best_params_)

# Extract
xgb_tuned = xgb_random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best XGBoost Parameters (RandomizedSearchCV):
{'colsample_bytree': np.float64(0.842571623863836), 'gamma':
np.float64(0.009197051616629648), 'learning_rate':
np.float64(0.030294308573206426), 'max_depth': 5, 'min_child_weight':
9, 'n_estimators': 537, 'subsample': np.float64(0.7942455014344907)}
```

##LightGBM

```
from lightgbm import LGBMRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Parameters
lgb_random_grid = {
    "n_estimators": randint(200, 800),           # boosting rounds
    "learning_rate": uniform(0.01, 0.2),
    "num_leaves": randint(20, 80),               # leaves in each tree
    "max_depth": [-1, 5, 7, 9, 11, 13],          # -1 = no limit, or
shallow trees
    "subsample": uniform(0.6, 0.4),
    "colsample_bytree": uniform(0.6, 0.4),
    "min_child_samples": randint(5, 50)          # min data in leaf
}

# Base LightGBM model
lgb_base = LGBMRegressor(
    random_state=42,
    n_jobs=-1,
    verbose=-1
)
```

```
# RandomizedSearchCV for LightGBM
lgb_random_search = RandomizedSearchCV(
    estimator=lgb_base,
    param_distributions=lgb_random_grid,
    n_iter=50,
    cv=3,
    scoring="neg_root_mean_squared_error",
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Tuned LightGBM
lgb_random_search.fit(X_train_rf, y_train)

print("Best LightGBM Parameters (RandomizedSearchCV):")
print(lgb_random_search.best_params_)

# 5. Extract
lgb_tuned = lgb_random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best LightGBM Parameters (RandomizedSearchCV):
{'colsample_bytree': np.float64(0.6914200087189198), 'learning_rate':
np.float64(0.04499098541918724), 'max_depth': -1, 'min_child_samples':
6, 'n_estimators': 503, 'num_leaves': 31, 'subsample':
np.float64(0.8233173814428391)}
```

##Catboost

```
from catboost import CatBoostRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Parameters
cat_random_grid = {
    "iterations": randint(300, 1200),        # boosting iterations
    "learning_rate": uniform(0.01, 0.2),
    "depth": randint(3, 10),                 # tree depth
    "l2_leaf_reg": uniform(1.0, 9.0),
    "subsample": uniform(0.6, 0.4)
}

# Base CatBoost model
cat_base = CatBoostRegressor(
    loss_function="RMSE",
    random_seed=42,
    verbose=False
)
```

```
# RandomizedSearchCV for CatBoost
cat_random_search = RandomizedSearchCV(
    estimator=cat_base,
    param_distributions=cat_random_grid,
    n_iter=50,
    cv=3,
    scoring="neg_root_mean_squared_error",
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Tuned CatBoost
cat_random_search.fit(X_train_rf, y_train)

print("Best CatBoost Parameters (RandomizedSearchCV):")
print(cat_random_search.best_params_)


# Extract
cat_tuned = cat_random_search.best_estimator_
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
Best CatBoost Parameters (RandomizedSearchCV):
{'depth': 6, 'iterations': 613, 'l2_leaf_reg':
np.float64(5.72280788469014), 'learning_rate':
np.float64(0.09638900372842316), 'subsample':
np.float64(0.7164916560792167)}
```

##Cross Validation(Tuned)

```
print("CROSS-VALIDATION: TUNED MODELS")

# Tuned Random Forest
evaluate_cv(rf_tuned, X_for_trees, y, "Random Forest (Tuned)")

# Tuned Gradient Boosting
evaluate_cv(gb_tuned, X_for_trees, y, "Gradient Boosting (Tuned)")

# Tuned XGBoost
evaluate_cv(xgb_tuned, X_for_trees, y, "XGBoost (Tuned)")

# Tuned LightGBM
evaluate_cv(lgb_tuned, X_for_trees, y, "LightGBM (Tuned)")

# Tuned CatBoost
evaluate_cv(cat_tuned, X_for_trees, y, "CatBoost (Tuned)")
```

```
CROSS-VALIDATION: TUNED MODELS
```

```
Random Forest (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 305,916.03 (+/- 10,735.22)
MAE:  199,622.40 (+/- 5,489.61)
R²:   0.750 (+/- 0.018)

Gradient Boosting (Tuned) CROSS-VALIDATION RESULTS (Primary Metric =
RMSE)
------------------------------------------------------------
RMSE: 298,840.48 (+/- 12,132.82)
MAE:  194,831.61 (+/- 4,251.37)
R²:   0.761 (+/- 0.018)

XGBoost (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 297,885.41 (+/- 7,900.60)
MAE:  194,137.23 (+/- 2,114.59)
R²:   0.763 (+/- 0.015)

LightGBM (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 293,741.52 (+/- 8,999.53)
MAE:  189,924.80 (+/- 1,335.02)
R²:   0.769 (+/- 0.016)

CatBoost (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 293,548.48 (+/- 6,301.28)
MAE:  190,843.81 (+/- 3,150.99)
R²:   0.770 (+/- 0.012)
```

#Tuned Models on Test

```
y_pred_rf_tuned = rf_tuned.predict(X_test_rf)
rf_rmse_tuned, rf_mae_tuned, rf_r2_tuned = evaluate_on_test(
    "Random Forest (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_rf_tuned
)
print('\n')
y_pred_gb_tuned = gb_tuned.predict(X_test_rf)
gb_rmse_tuned, gb_mae_tuned, gb_r2_tuned = evaluate_on_test(
    "Gradient Boosting (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_gb_tuned
)
print('\n')
y_pred_xgb_tuned = xgb_tuned.predict(X_test_rf)
xgb_rmse_tuned, xgb_mae_tuned, xgb_r2_tuned = evaluate_on_test(
```

```
    "XGBoost (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_xgb_tuned
)
print('\n')
y_pred_lgb_tuned = lgb_tuned.predict(X_test_rf)
lgb_rmse_tuned, lgb_mae_tuned, lgb_r2_tuned = evaluate_on_test(
    "LightGBM (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_lgb_tuned
)
print('\n')
y_pred_cat_tuned = cat_tuned.predict(X_test_rf)
cat_rmse_tuned, cat_mae_tuned, cat_r2_tuned = evaluate_on_test(
    "CatBoost (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_cat_tuned
)
```

```
Random Forest (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $310,950.25
MAE:  $199,835.95
R²:   0.721


Gradient Boosting (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $301,422.78
MAE:  $193,841.36
R²:   0.738


XGBoost (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $301,230.60
MAE:  $193,928.98
R²:   0.739


LightGBM (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $294,226.55
MAE:  $188,888.44
R²:   0.751


CatBoost (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $289,750.89
MAE:  $186,327.52
R²:   0.758
```

results

#Final Results

##Final Cross Validation

```
evaluate_cv(lr_cv,  X_for_lr_scaled, y, "Linear Regression")
evaluate_cv(rf_cv,  X_for_trees,     y, "Random Forest")
evaluate_cv(rf_tuned, X_for_trees, y, "Random Forest (Tuned)")
evaluate_cv(gb_cv,  X_for_trees,     y, "Gradient Boosting")
evaluate_cv(gb_tuned, X_for_trees, y, "Gradient Boosting (Tuned)")
evaluate_cv(xgb_cv, X_for_trees,     y, "XGBoost")
evaluate_cv(xgb_tuned, X_for_trees, y, "XGBoost (Tuned)")
evaluate_cv(lgb_cv, X_for_trees,     y, "LightGBM")
evaluate_cv(lgb_tuned, X_for_trees, y, "LightGBM (Tuned)")
evaluate_cv(cat_cv, X_for_trees,     y, "CatBoost")
evaluate_cv(cat_tuned, X_for_trees, y, "CatBoost (Tuned)")


Linear Regression CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 424,660.83 (+/- 8,903.74)
MAE:  297,587.55 (+/- 6,325.46)
R²:   0.519 (+/- 0.020)

Random Forest CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 308,634.01 (+/- 11,454.93)
MAE:  196,958.03 (+/- 6,067.62)
R²:   0.745 (+/- 0.023)

Random Forest (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 305,916.03 (+/- 10,735.22)
MAE:  199,622.40 (+/- 5,489.61)
R²:   0.750 (+/- 0.018)

Gradient Boosting CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 330,132.25 (+/- 10,375.19)
MAE:  224,448.16 (+/- 5,005.27)
R²:   0.709 (+/- 0.015)

Gradient Boosting (Tuned) CROSS-VALIDATION RESULTS (Primary Metric =
RMSE)
------------------------------------------------------------
RMSE: 298,840.48 (+/- 12,132.82)
MAE:  194,831.61 (+/- 4,251.37)
R²:   0.761 (+/- 0.018)

XGBoost CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 303,563.69 (+/- 15,097.33)
MAE:  197,072.28 (+/- 5,932.37)
```

```
R²:   0.753 (+/- 0.028)

XGBoost (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 297,885.41 (+/- 7,900.60)
MAE:  194,137.23 (+/- 2,114.59)
R²:   0.763 (+/- 0.015)

LightGBM CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 302,181.83 (+/- 13,015.67)
MAE:  195,514.86 (+/- 5,598.02)
R²:   0.756 (+/- 0.024)

LightGBM (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 293,741.52 (+/- 8,999.53)
MAE:  189,924.80 (+/- 1,335.02)
R²:   0.769 (+/- 0.016)

CatBoost CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 293,003.99 (+/- 8,207.33)
MAE:  191,177.59 (+/- 2,693.08)
R²:   0.771 (+/- 0.015)

CatBoost (Tuned) CROSS-VALIDATION RESULTS (Primary Metric = RMSE)
------------------------------------------------------------
RMSE: 293,548.48 (+/- 6,301.28)
MAE:  190,843.81 (+/- 3,150.99)
R²:   0.770 (+/- 0.012)
```

##Final Test

```
evaluate_on_test("Linear Regression (Untuned)", y_test, y_pred_lr)
print('\n')
evaluate_on_test("Random Forest (Untuned)", y_test, y_pred_rf)
print('\n')
rf_rmse_tuned, rf_mae_tuned, rf_r2_tuned = evaluate_on_test(
    "Random Forest (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_rf_tuned
)
print('\n')
evaluate_on_test("Gradient Boosting (Untuned)", y_test, y_pred_gb)
print('\n')
gb_rmse_tuned, gb_mae_tuned, gb_r2_tuned = evaluate_on_test(
    "Gradient Boosting (Tuned via RandomizedSearchCV)",
    y_test,
```

```
    y_pred_gb_tuned
)
print('\n')
evaluate_on_test("XGBoost (Untuned)", y_test, y_pred_xgb)
print('\n')
xgb_rmse_tuned, xgb_mae_tuned, xgb_r2_tuned = evaluate_on_test(
    "XGBoost (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_xgb_tuned
)
print('\n')
evaluate_on_test("LightGBM (Untuned)", y_test, y_pred_lgb)
print('\n')
lgb_rmse_tuned, lgb_mae_tuned, lgb_r2_tuned = evaluate_on_test(
    "LightGBM (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_lgb_tuned
)
print('\n')
evaluate_on_test("CatBoost (Untuned)", y_test, y_pred_cat)
print('\n')
cat_rmse_tuned, cat_mae_tuned, cat_r2_tuned = evaluate_on_test(
    "CatBoost (Tuned via RandomizedSearchCV)",
    y_test,
    y_pred_cat_tuned
)
```

```
Linear Regression (Untuned) TEST RESULTS
RMSE: $409,783.72
MAE:  $294,896.30
R²:   0.516


Random Forest (Untuned) TEST RESULTS
RMSE: $327,738.17
MAE:  $203,821.92
R²:   0.691


Random Forest (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $310,950.25
MAE:  $199,835.95
R²:   0.721


Gradient Boosting (Untuned) TEST RESULTS
RMSE: $325,824.63
MAE:  $220,779.92
R²:   0.694
```

```
Gradient Boosting (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $301,422.78
MAE:  $193,841.36
R²:   0.738


XGBoost (Untuned) TEST RESULTS
RMSE: $318,387.73
MAE:  $202,538.00
R²:   0.708


XGBoost (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $301,230.60
MAE:  $193,928.98
R²:   0.739


LightGBM (Untuned) TEST RESULTS
RMSE: $307,857.73
MAE:  $195,476.37
R²:   0.727


LightGBM (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $294,226.55
MAE:  $188,888.44
R²:   0.751


CatBoost (Untuned) TEST RESULTS
RMSE: $290,146.42
MAE:  $186,206.86
R²:   0.757


CatBoost (Tuned via RandomizedSearchCV) TEST RESULTS
RMSE: $289,750.89
MAE:  $186,327.52
R²:   0.758
```

##Helpers + Range

```python
def plot_residual_hist_fixed(y_true, y_pred, title,
                             x_min, x_max, y_max):
    residuals = y_true - y_pred

    plt.figure(figsize=(8,5))
    sns.histplot(residuals, bins=40, kde=True)
```

```
    plt.axvline(0, color='red', linestyle='--')

    plt.xlim(x_min, x_max)
    plt.ylim(0, y_max)

    plt.xlabel("Residual (Actual - Predicted)")
    plt.ylabel("Frequency")
    plt.title(title)
    plt.tight_layout()
    plt.show()

def plot_actual_vs_predicted_fixed(y_true, y_pred, title,
                                   min_val, max_val, color="blue"):
    plt.figure(figsize=(6,6))
    plt.scatter(y_true, y_pred, alpha=0.4, color=color)

    plt.plot([min_val, max_val], [min_val, max_val],
             color='red', linewidth=2)

    plt.xlim(min_val, max_val)
    plt.ylim(min_val, max_val)

    plt.xlabel("Actual Price")
    plt.ylabel("Predicted Price")
    plt.title(title)
    plt.tight_layout()
    plt.show()

def plot_residual_scatter_fixed(y_true, y_pred, title,
                                pred_min, pred_max,
                                resid_min, resid_max,
                                color="green"):
    residuals = y_true - y_pred

    plt.figure(figsize=(6,6))
    plt.scatter(y_pred, residuals, alpha=0.4, color=color)
    plt.axhline(0, color='red', linestyle='--')

    plt.xlim(pred_min, pred_max)
    plt.ylim(resid_min, resid_max)

    plt.xlabel("Predicted Price")
    plt.ylabel("Residual (Actual - Predicted)")
    plt.title(title)
    plt.tight_layout()
    plt.show()

import numpy as np

# 1. All predictions
all_preds = [
```

```
    y_pred_lr, y_pred_rf, y_pred_rf_tuned, y_pred_gb, y_pred_gb_tuned,
    y_pred_xgb, y_pred_xgb_tuned, y_pred_lgb, y_pred_lgb_tuned,
    y_pred_cat, y_pred_cat_tuned
]

# 2. Residual range
all_residuals = np.concatenate([y_test - p for p in all_preds])
global_res_min, global_res_max = all_residuals.min(),
all_residuals.max()

# 3. Histogram max height
global_hist_y_max = max(np.histogram(y_test - p, bins=40)[0].max() for
p in all_preds)

# 4. Actual vs Predicted price range
global_price_min = min(y_test.min(), *(p.min() for p in all_preds))
global_price_max = max(y_test.max(), *(p.max() for p in all_preds))

# 5. Predicted & residual axis ranges
global_pred_min  = global_price_min
global_pred_max  = global_price_max
global_resid_min = global_res_min
global_resid_max = global_res_max

preds = {
    "Linear Regression (Untuned)": y_pred_lr,
    "Random Forest (Untuned)": y_pred_rf,
    "Random Forest (Tuned)": y_pred_rf_tuned,
    "Gradient Boosting (Untuned)": y_pred_gb,
    "Gradient Boosting (Tuned)": y_pred_gb_tuned,
    "XGBoost (Untuned)": y_pred_xgb,
    "XGBoost (Tuned)": y_pred_xgb_tuned,
    "LightGBM (Untuned)": y_pred_lgb,
    "LightGBM (Tuned)": y_pred_lgb_tuned,
    "CatBoost (Untuned)": y_pred_cat,
    "CatBoost (Tuned)": y_pred_cat_tuned
}

colors = {
    "Linear Regression (Untuned)": "goldenrod",
    "Random Forest (Untuned)": "green",
    "Random Forest (Tuned)": "seagreen",
    "Gradient Boosting (Untuned)": "orange",
    "Gradient Boosting (Tuned)": "darkorange",
    "XGBoost (Untuned)": "royalblue",
    "XGBoost (Tuned)": "navy",
    "LightGBM (Untuned)": "purple",
    "LightGBM (Tuned)": "indigo",
    "CatBoost (Untuned)": "deeppink",
    "CatBoost (Tuned)": "red"
```

```
}
```

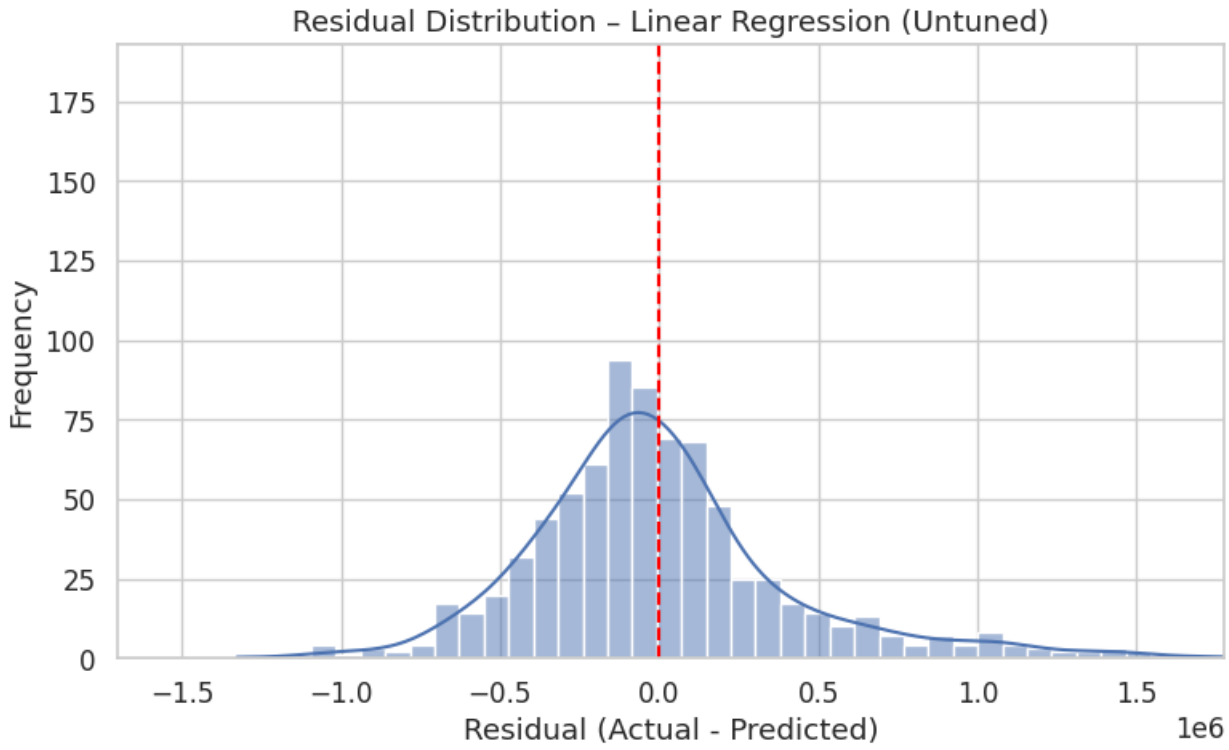
###Actual VS Predicted Fixed

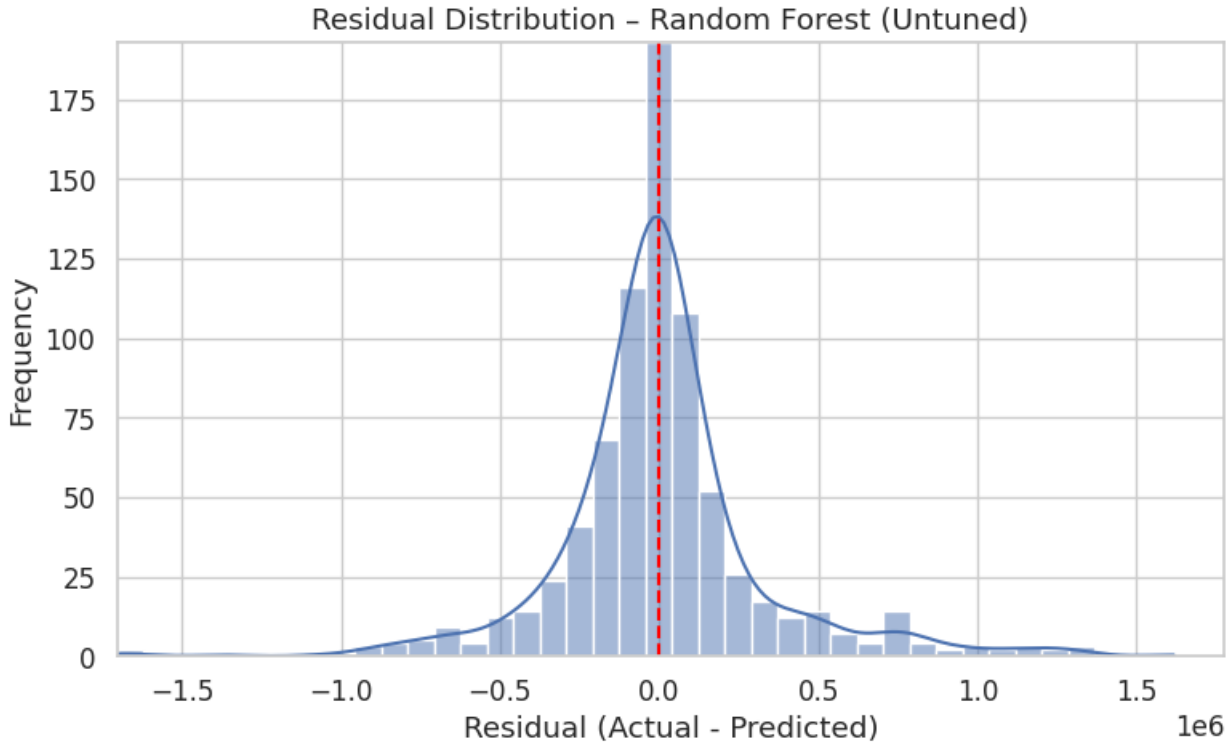
```
all_preds = list(preds.values())

global_price_min = min(y_test.min(), *(pred.min() for pred in
all_preds))
global_price_max = max(y_test.max(), *(pred.max() for pred in
all_preds))
```

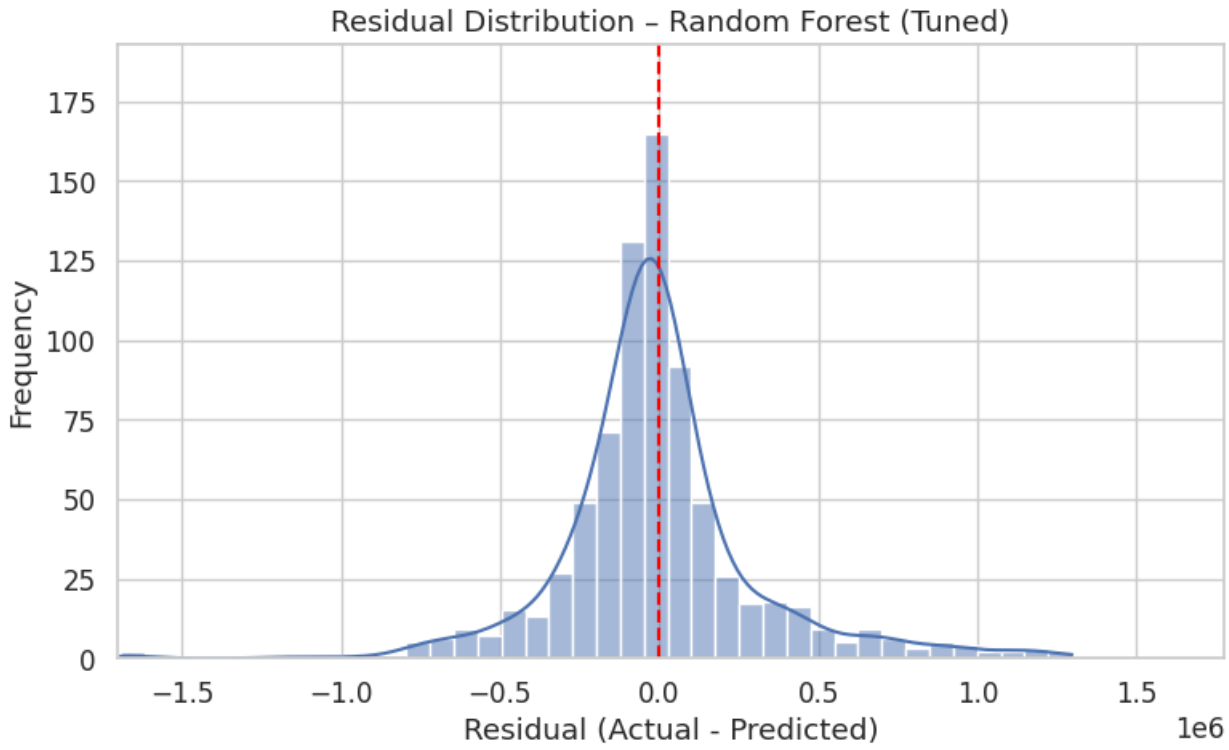
#Histogram plots

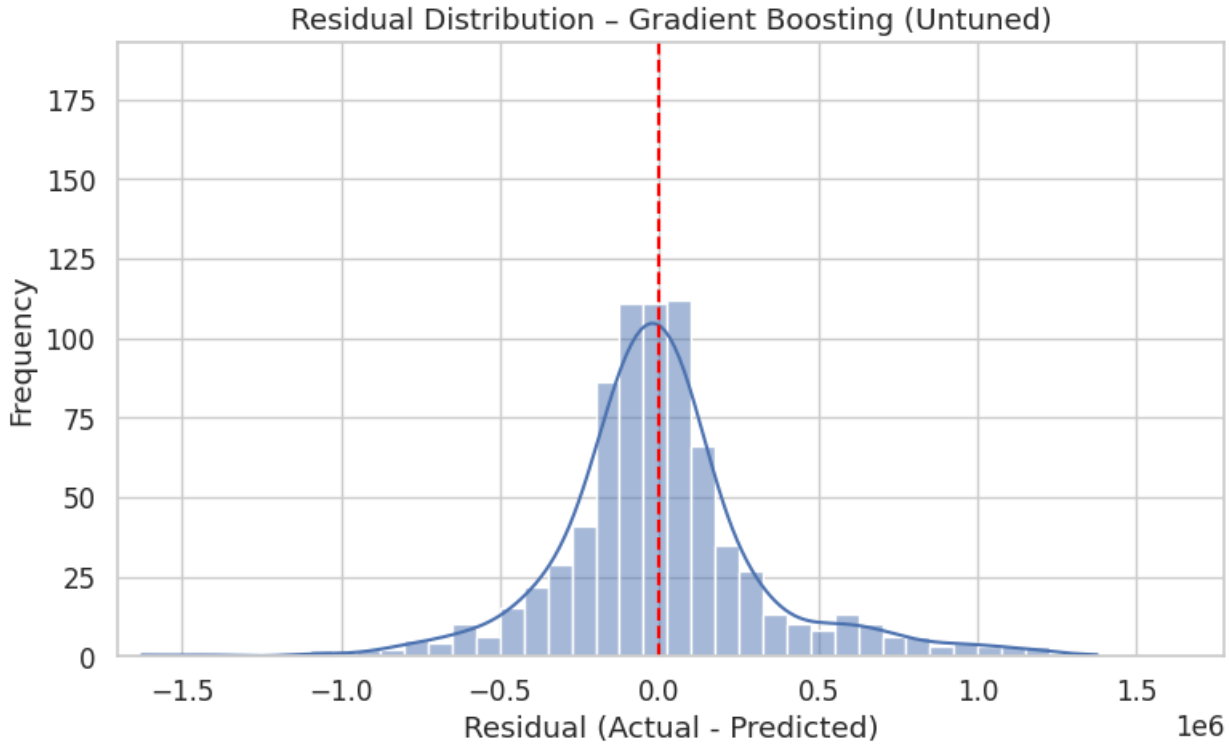
```
# Linear Regression (Untuned)
plot_residual_hist_fixed(
    y_test, y_pred_lr,
    "Residual Distribution – Linear Regression (Untuned)",
    global_res_min, global_res_max, global_hist_y_max
)

# Random Forest
plot_residual_hist_fixed(
    y_test, y_pred_rf,
    "Residual Distribution – Random Forest (Untuned)",
    global_res_min, global_res_max, global_hist_y_max
)

plot_residual_hist_fixed(
    y_test, y_pred_rf_tuned,
    "Residual Distribution – Random Forest (Tuned)",
    global_res_min, global_res_max, global_hist_y_max
)

# Gradient Boosting
plot_residual_hist_fixed(
    y_test, y_pred_gb,
    "Residual Distribution – Gradient Boosting (Untuned)",
    global_res_min, global_res_max, global_hist_y_max
)

plot_residual_hist_fixed(
    y_test, y_pred_gb_tuned,
    "Residual Distribution – Gradient Boosting (Tuned)",
    global_res_min, global_res_max, global_hist_y_max
)

# XGBoost
plot_residual_hist_fixed(
```

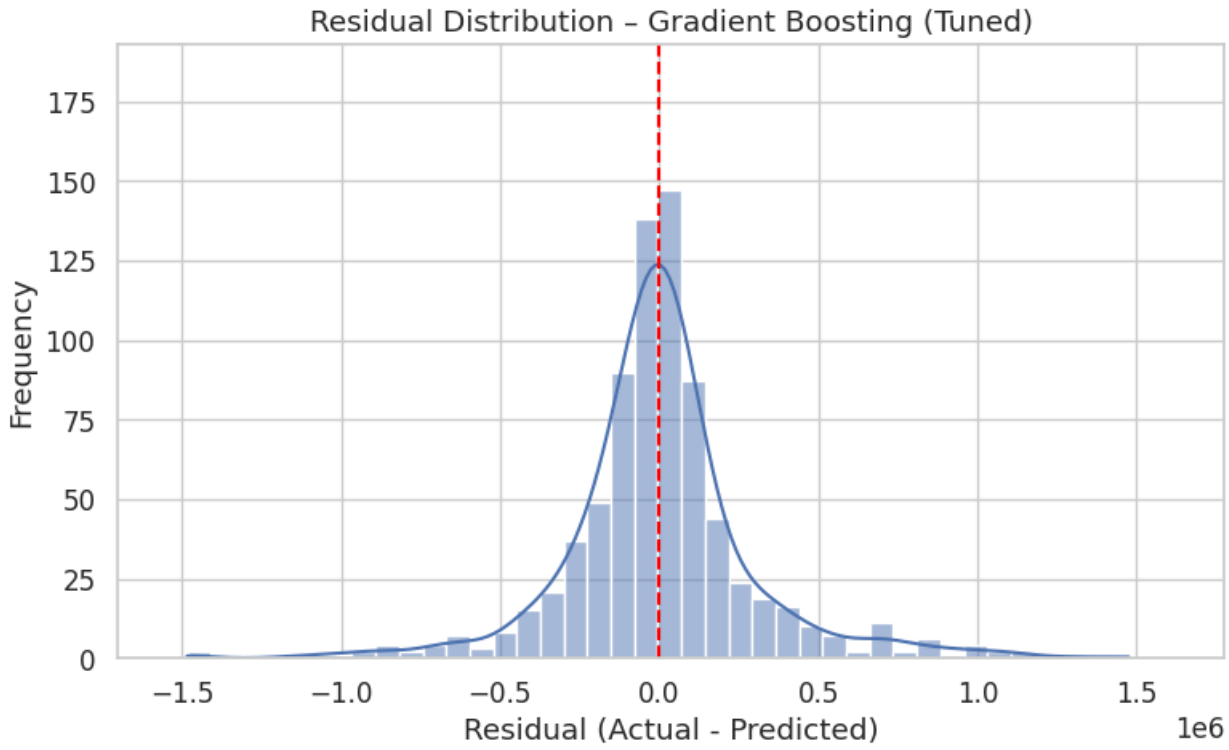
```
    y_test, y_pred_xgb,
    "Residual Distribution – XGBoost (Untuned)",
    global_res_min, global_res_max, global_hist_y_max
)

plot_residual_hist_fixed(
    y_test, y_pred_xgb_tuned,
    "Residual Distribution – XGBoost (Tuned)",
    global_res_min, global_res_max, global_hist_y_max
)

# LightGBM
plot_residual_hist_fixed(
    y_test, y_pred_lgb,
    "Residual Distribution – LightGBM (Untuned)",
    global_res_min, global_res_max, global_hist_y_max
)

plot_residual_hist_fixed(
    y_test, y_pred_lgb_tuned,
    "Residual Distribution – LightGBM (Tuned)",
    global_res_min, global_res_max, global_hist_y_max
)

# CatBoost
plot_residual_hist_fixed(
    y_test, y_pred_cat,
    "Residual Distribution – CatBoost (Untuned)",
    global_res_min, global_res_max, global_hist_y_max
)

plot_residual_hist_fixed(
    y_test, y_pred_cat_tuned,
    "Residual Distribution – CatBoost (Tuned)",
    global_res_min, global_res_max, global_hist_y_max
)
```

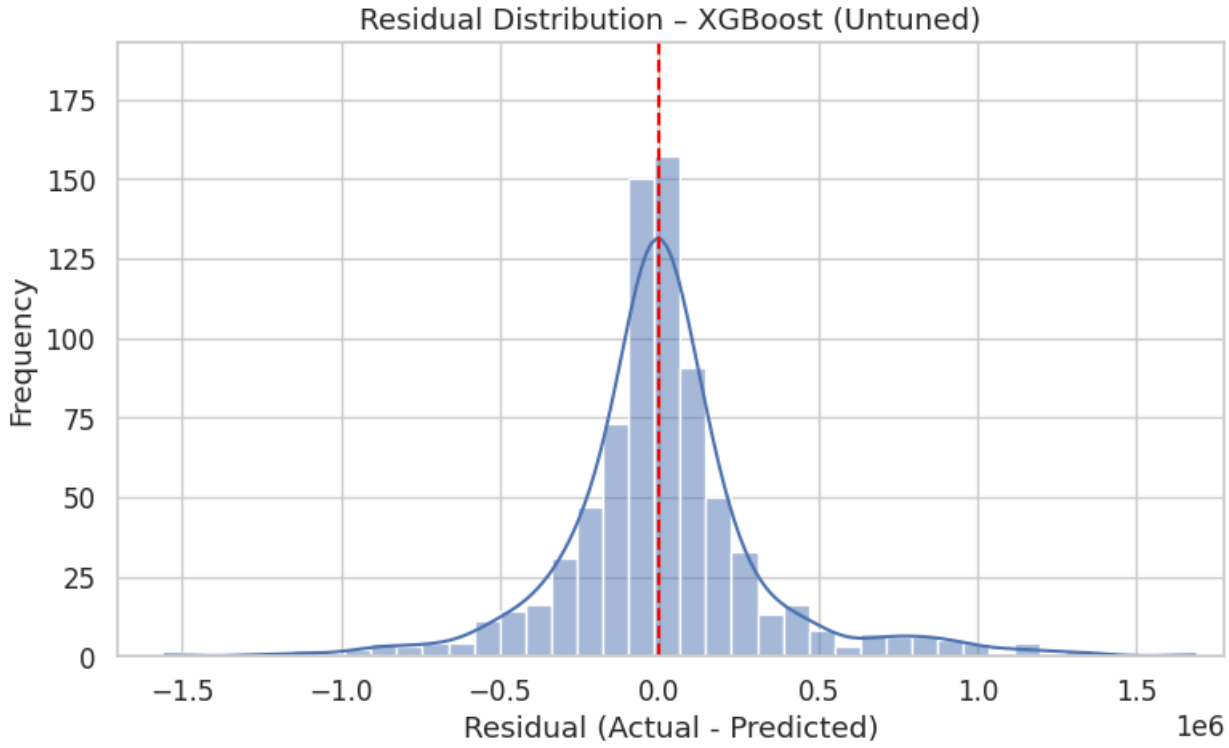

Residual Distribution – Linear Regression (Untuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

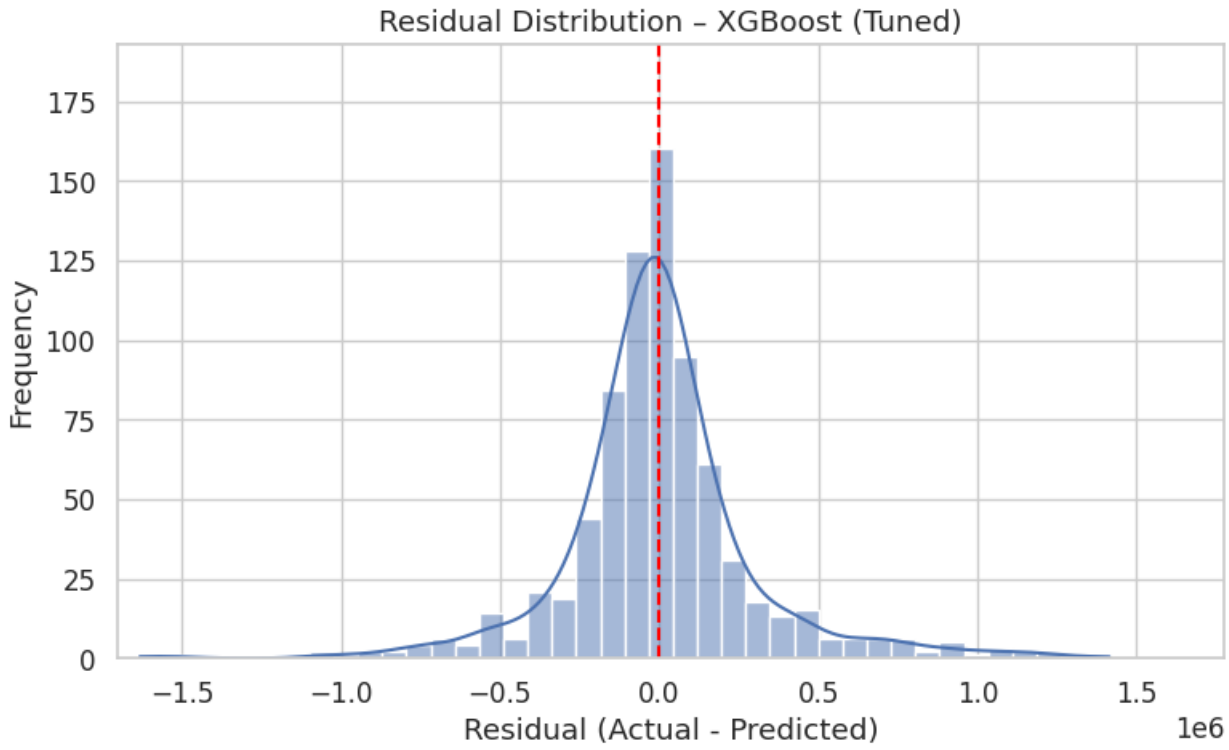

Residual Distribution – Random Forest (Untuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

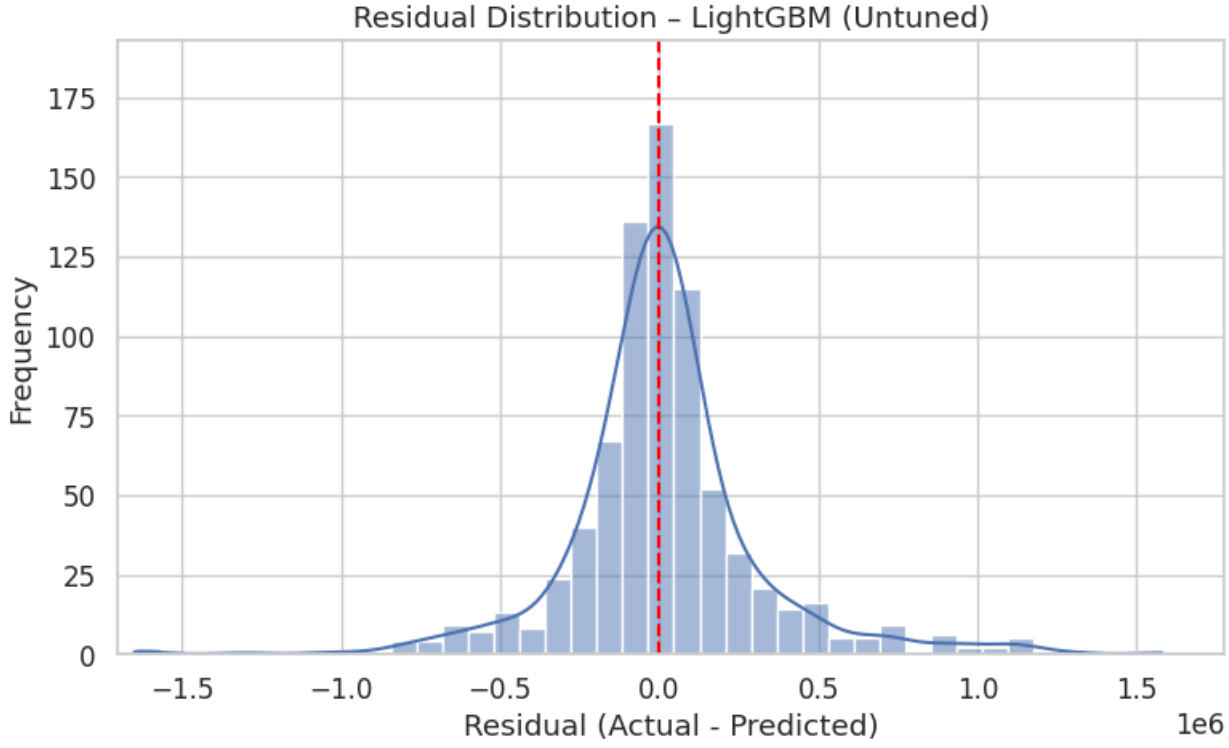

Residual Distribution – Random Forest (Tuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

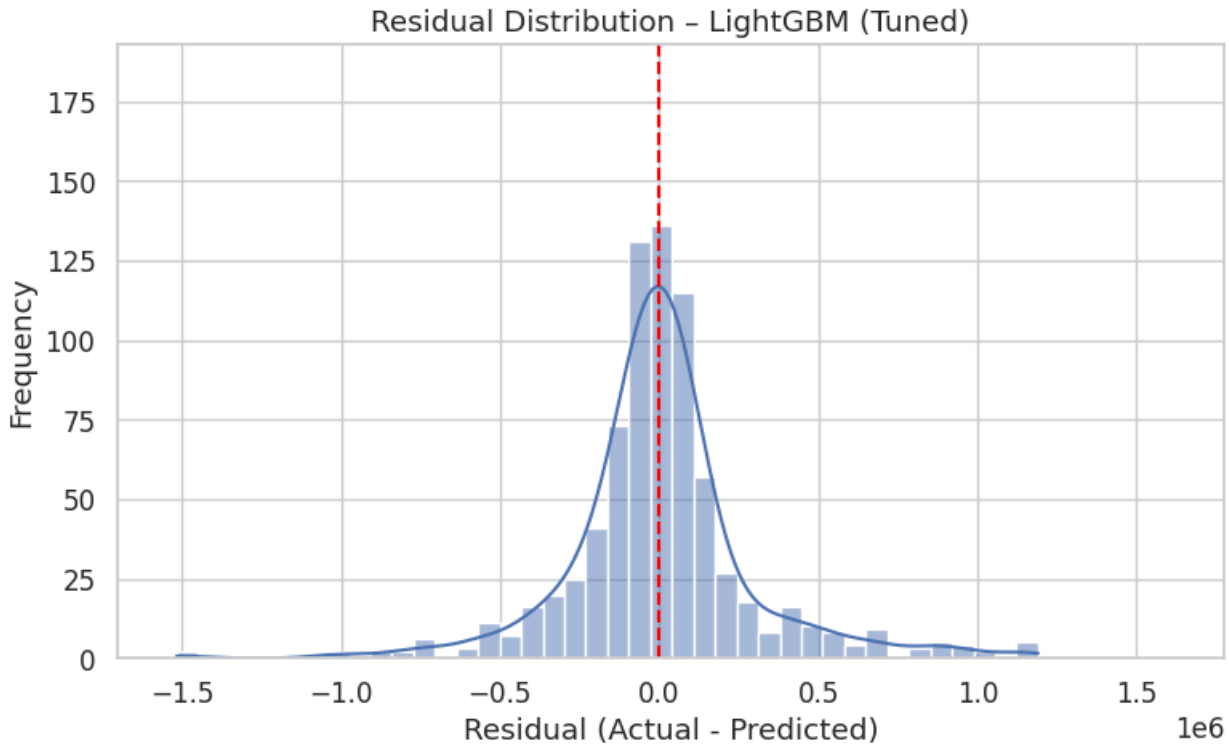

Residual Distribution – Gradient Boosting (Untuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

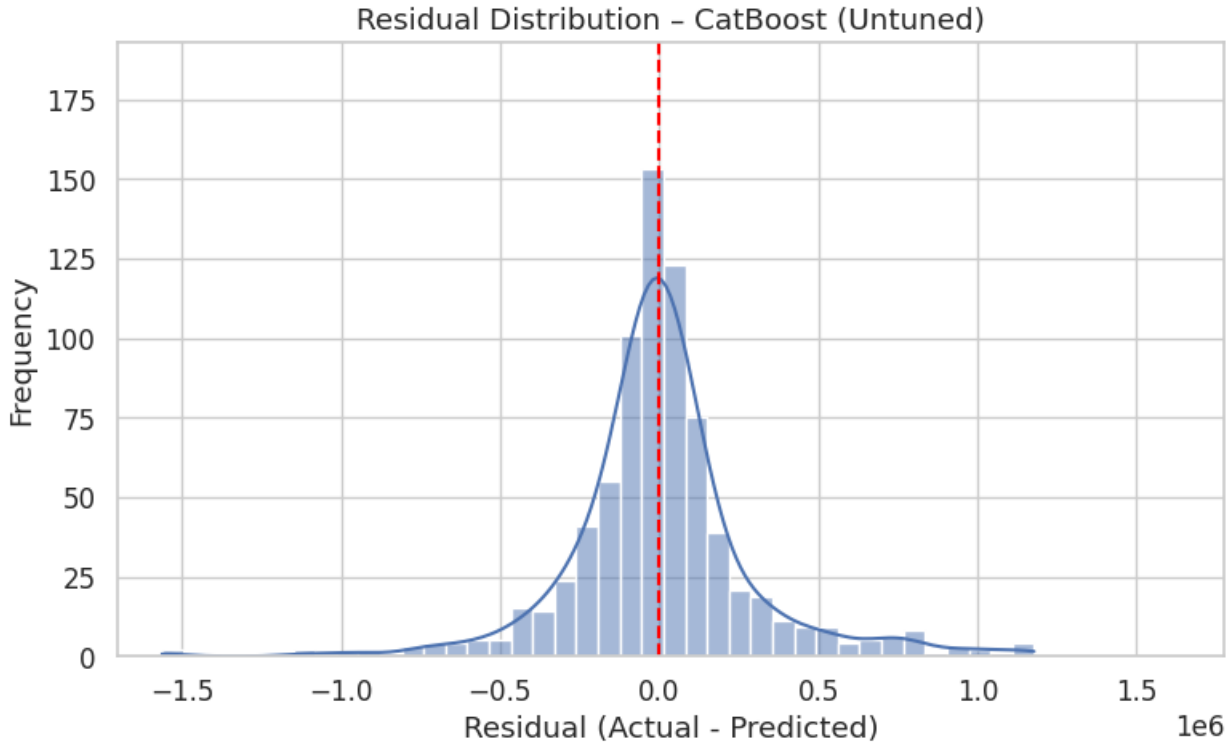

Residual Distribution – Gradient Boosting (Tuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

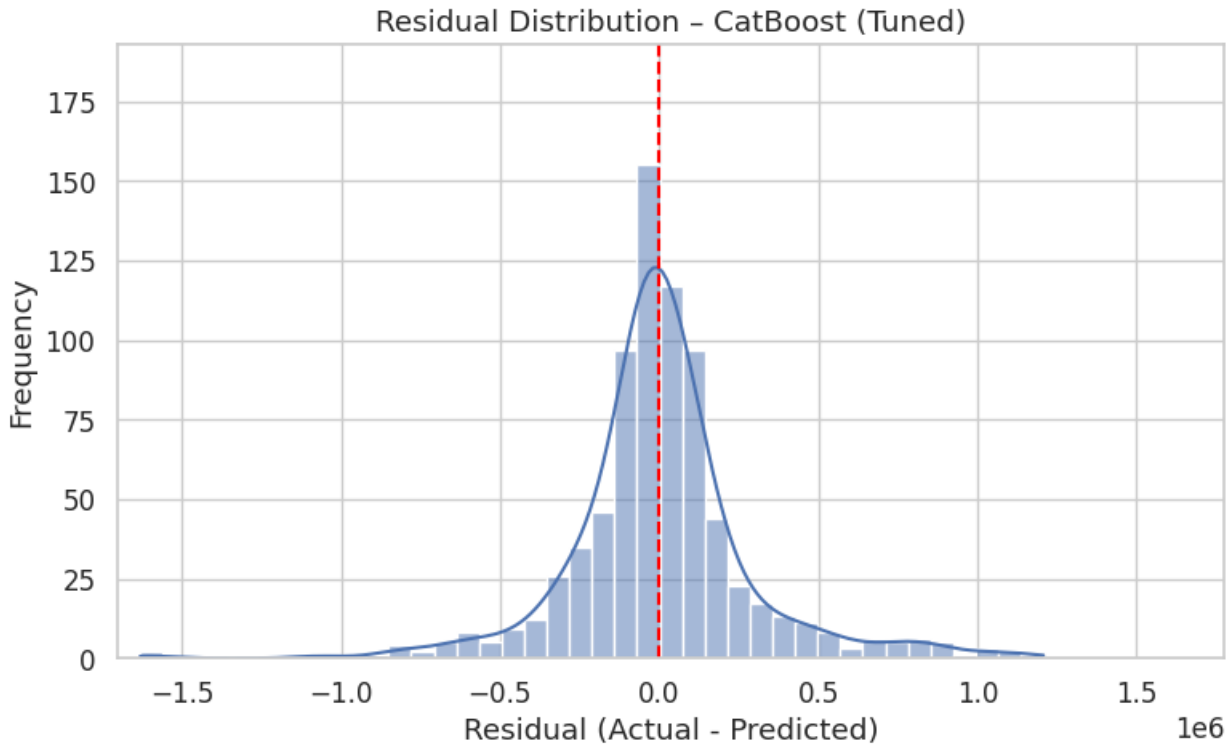

Residual Distribution – XGBoost (Untuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6


Residual Distribution – XGBoost (Tuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

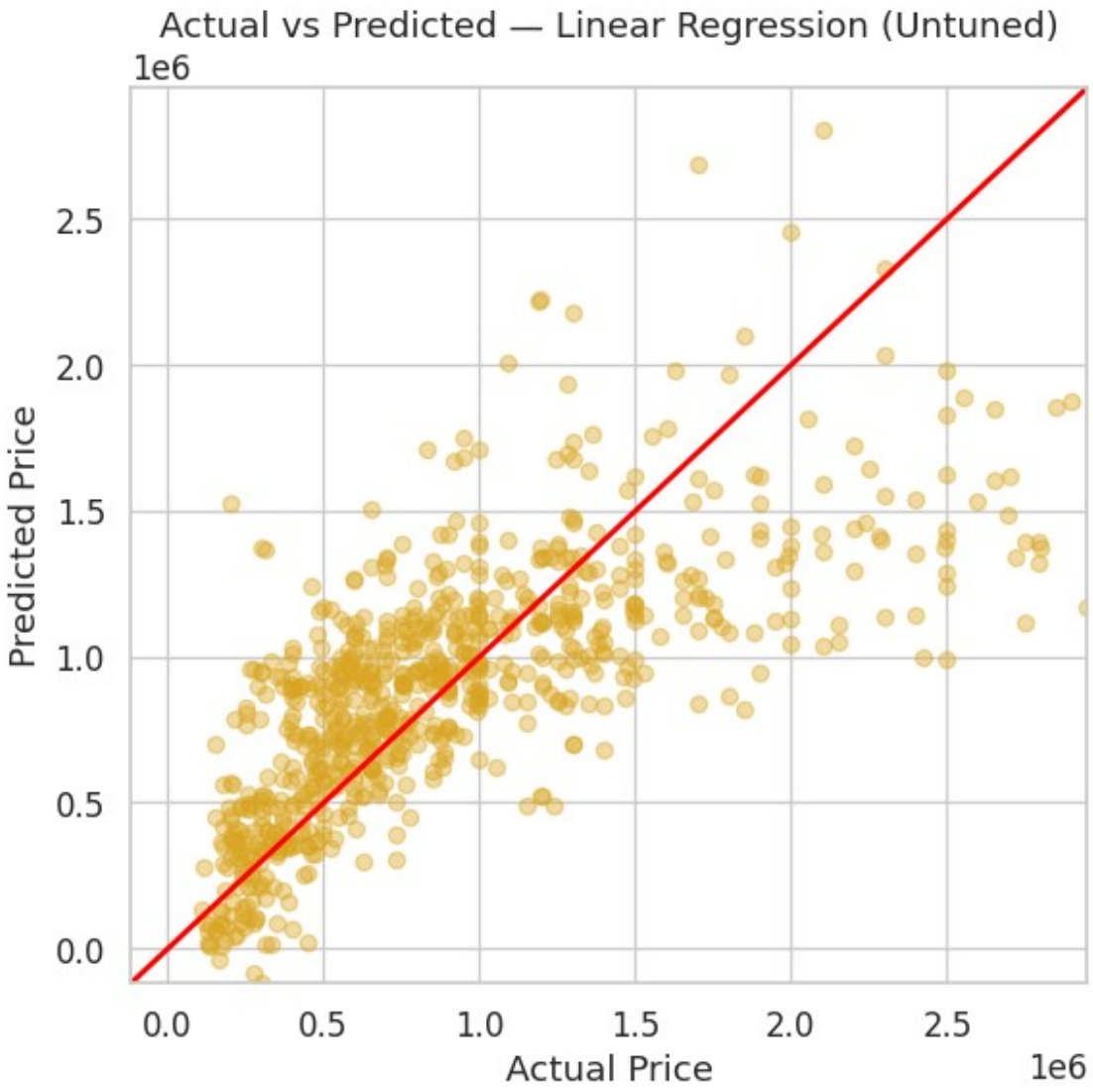

Residual Distribution – LightGBM (Untuned)
Frequency
175
150
125
100
75
50
25
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

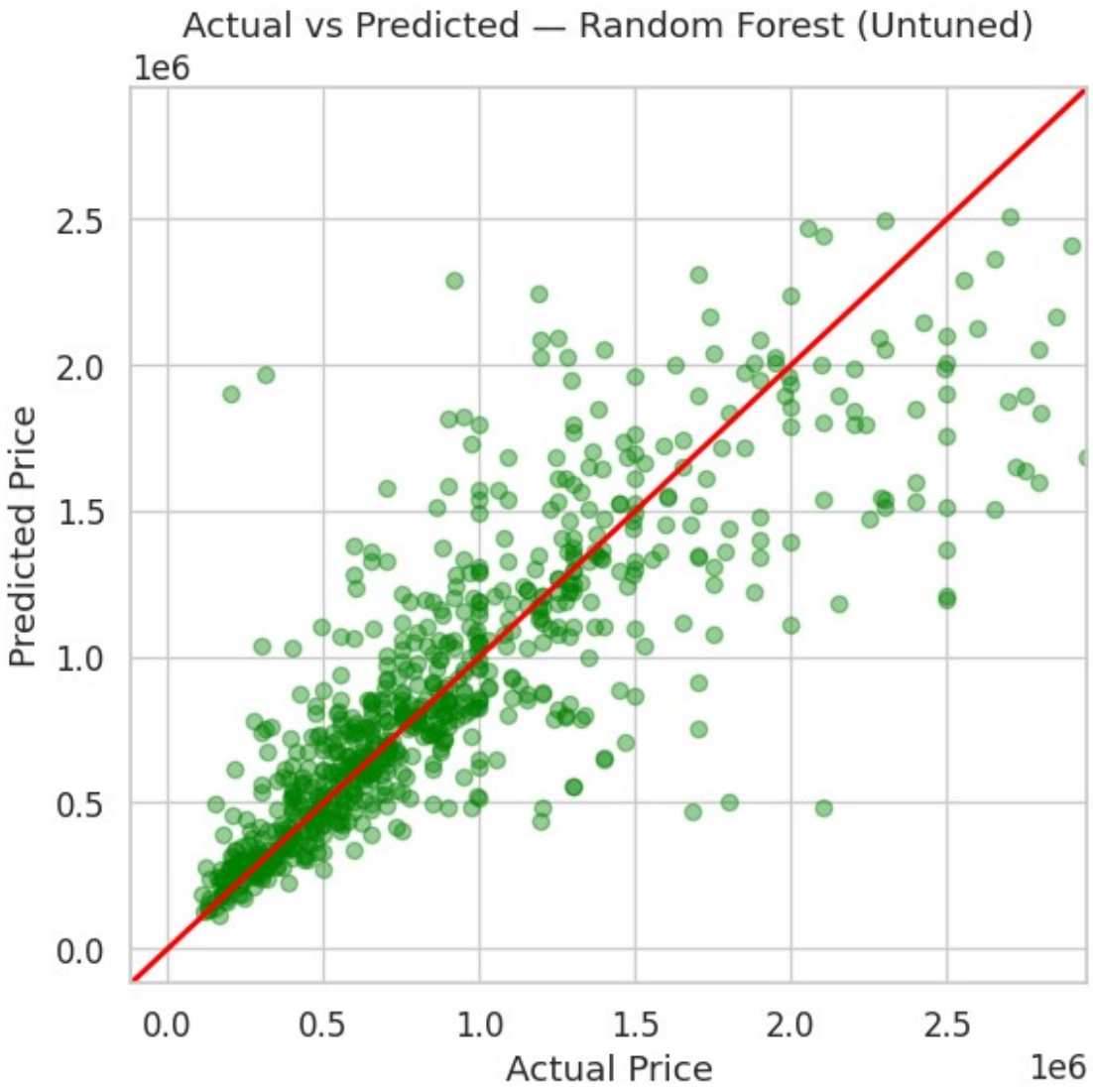

Residual Distribution – LightGBM (Tuned)
Frequency
0
25
50
75
100
125
150
175
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

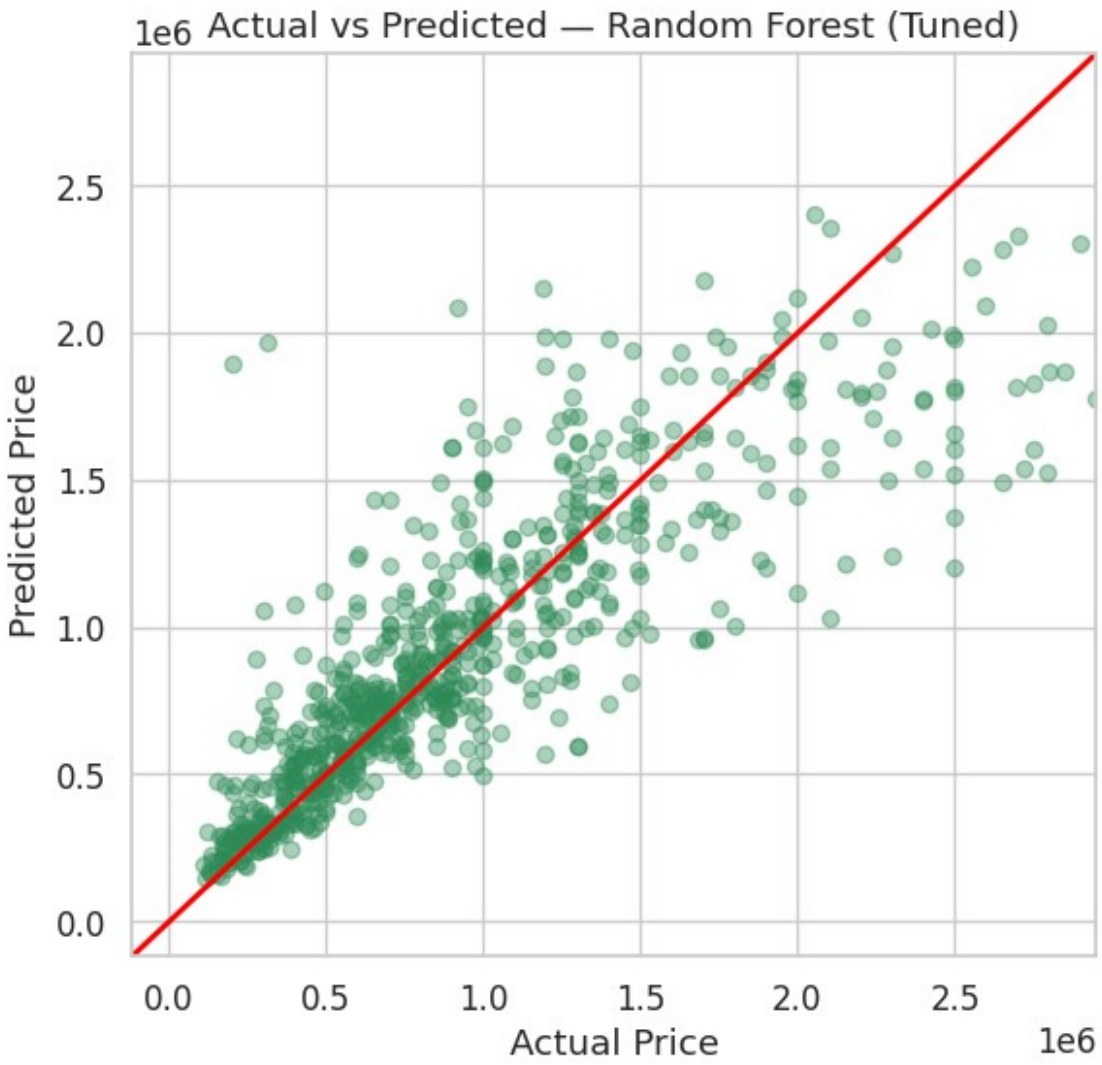

Residual Distribution – CatBoost (Untuned)
Frequency
0
25
50
75
100
125
150
175
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Residual (Actual - Predicted)
1e6

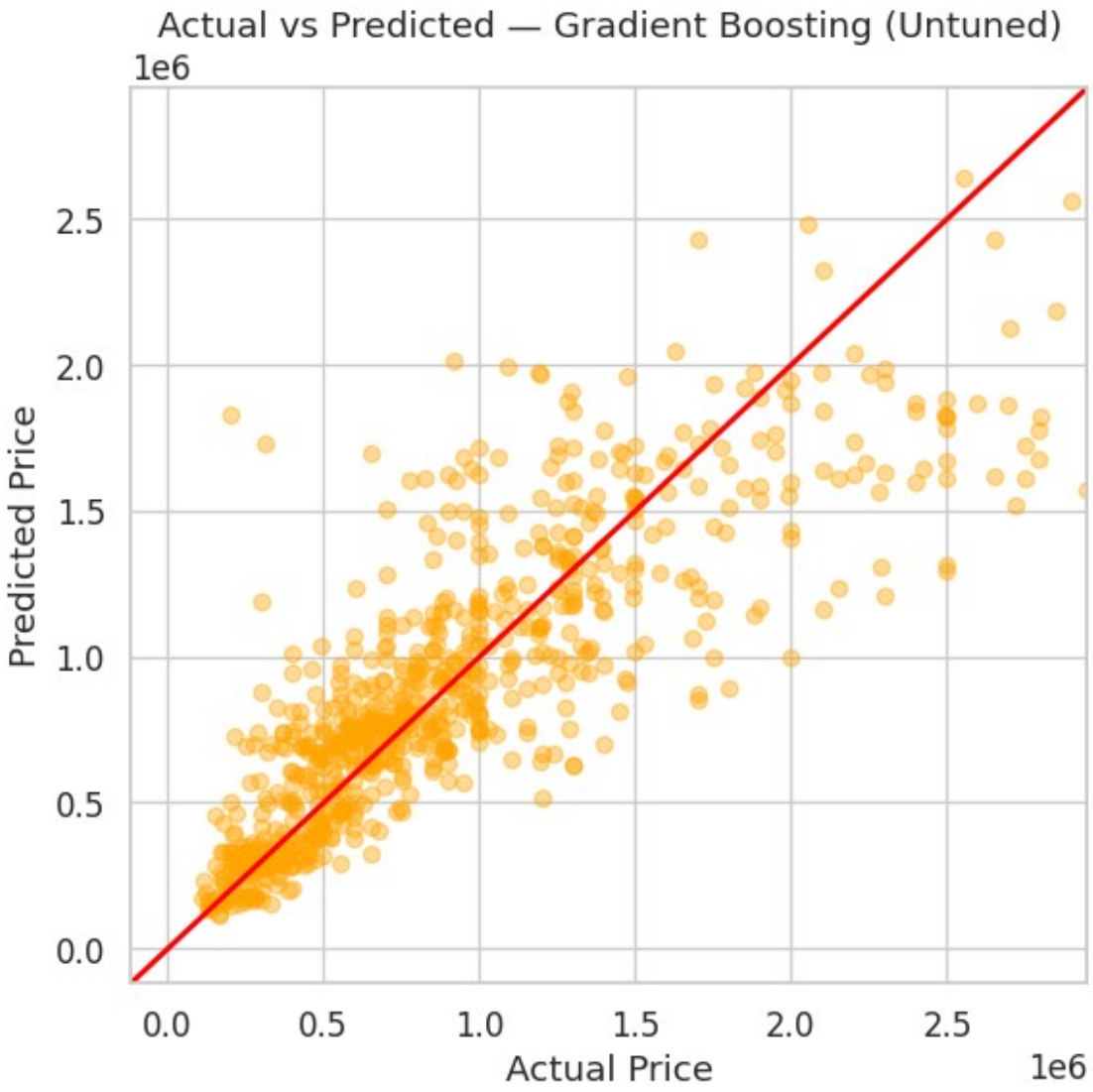


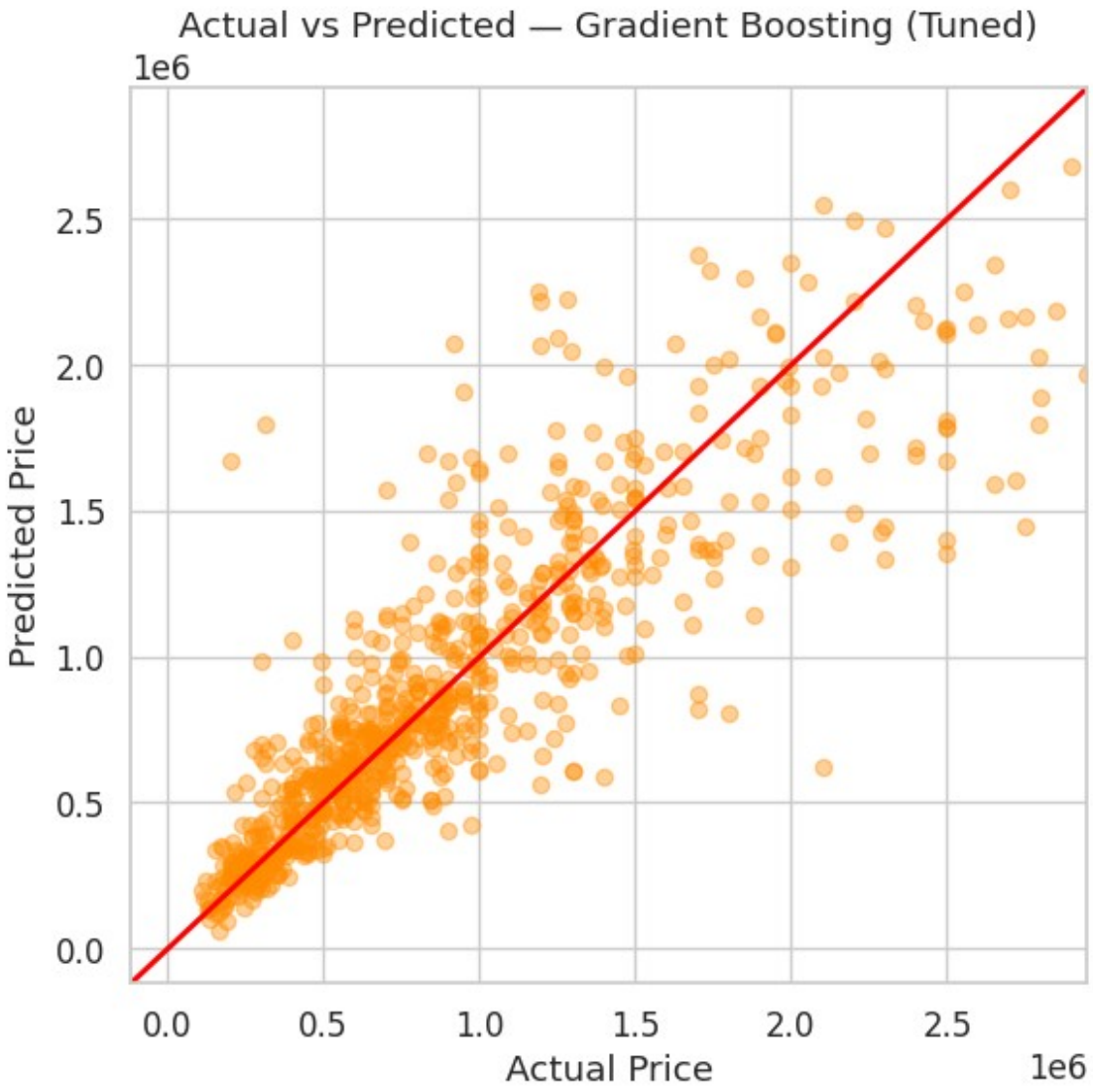

```
for name, yhat in preds.items():
    plot_actual_vs_predicted_fixed(
        y_test, yhat,
        f"Actual vs Predicted — {name}",
        global_price_min, global_price_max,
        colors.get(name, "black")
    )
```

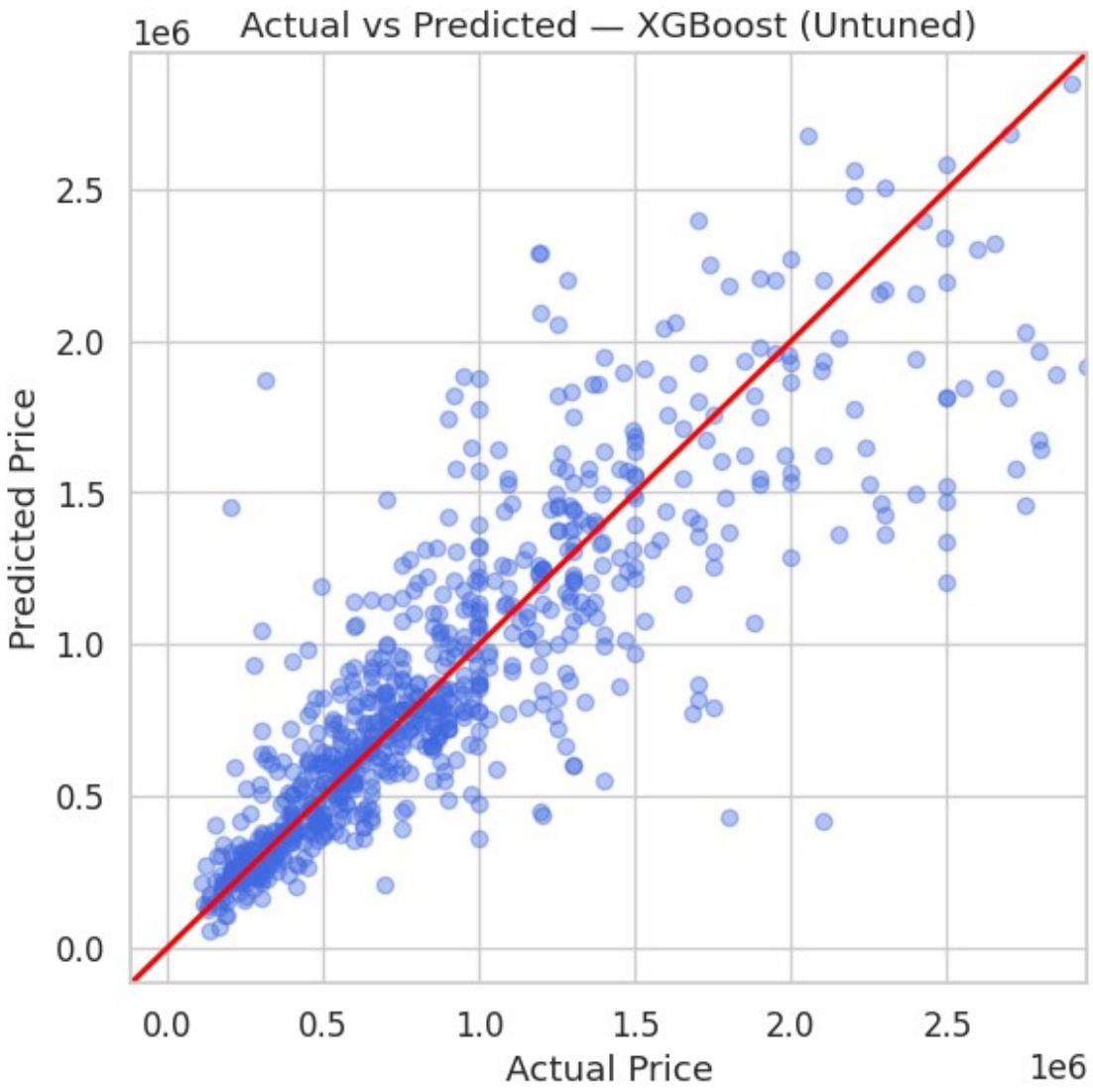

Actual vs Predicted — Linear Regression (Untuned)
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

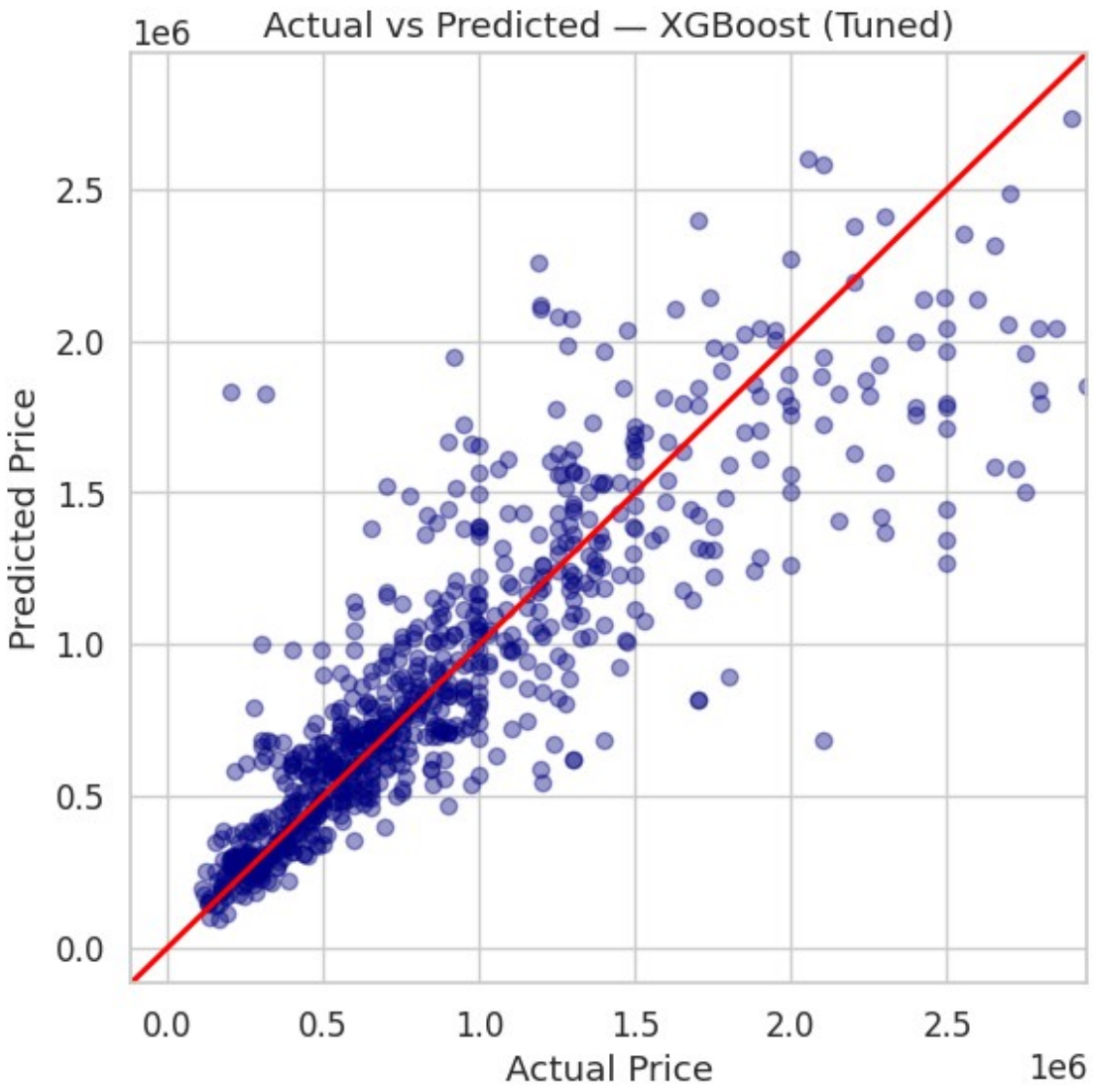

Actual vs Predicted — Random Forest (Untuned)
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

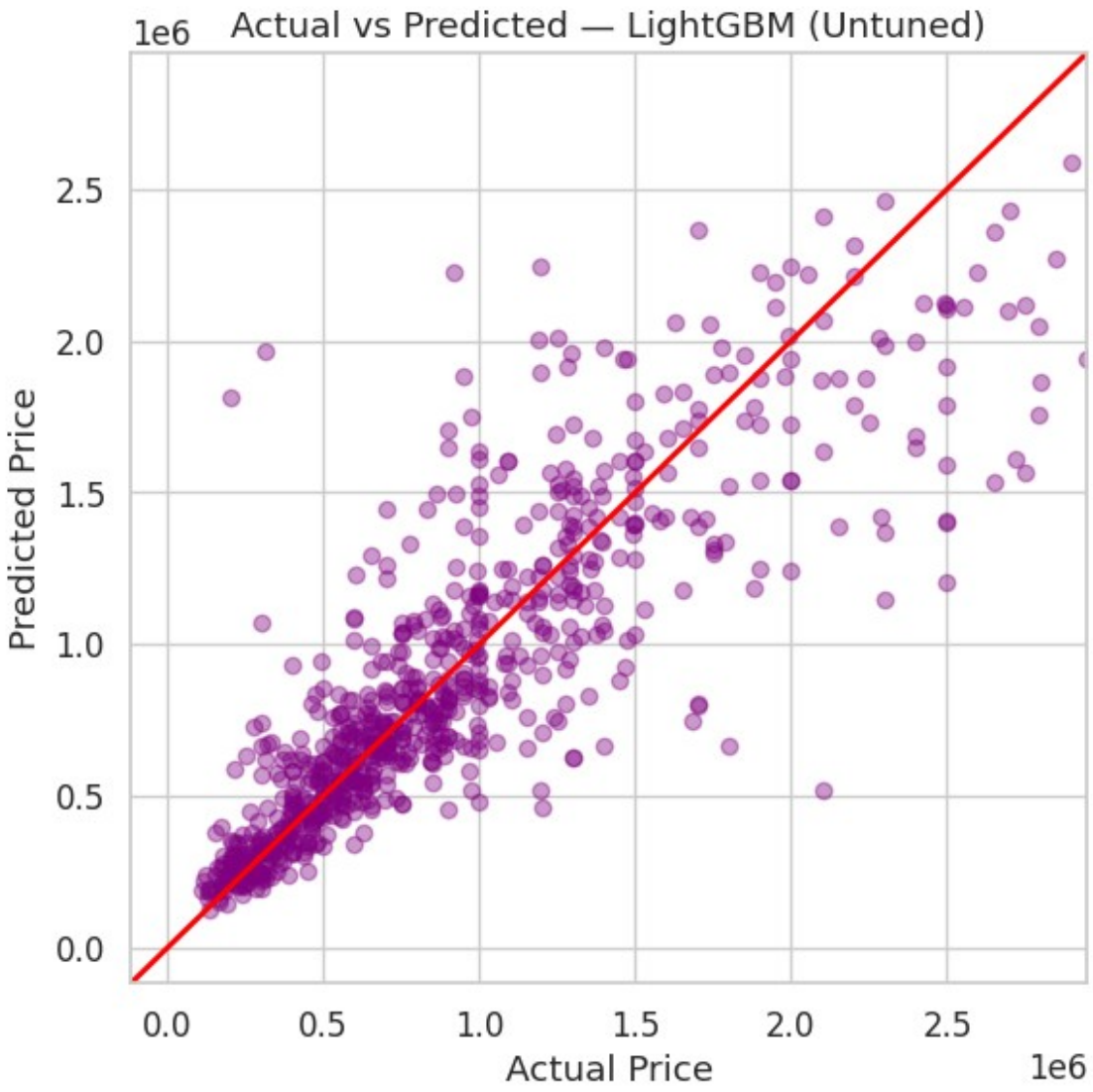

Actual vs Predicted — Random Forest (Tuned)
1e6
Predicted Price
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

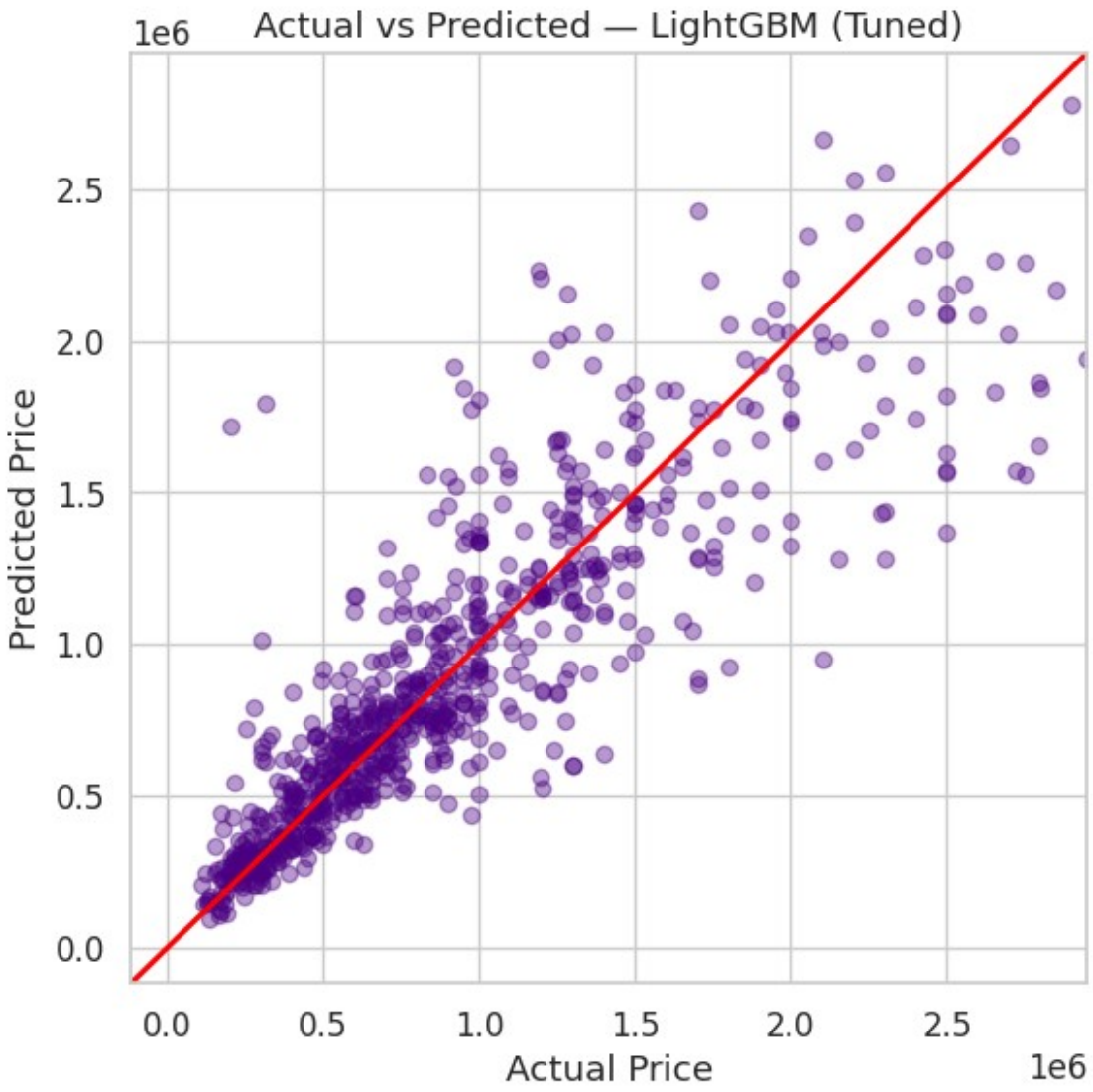

Actual vs Predicted — Gradient Boosting (Untuned)
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

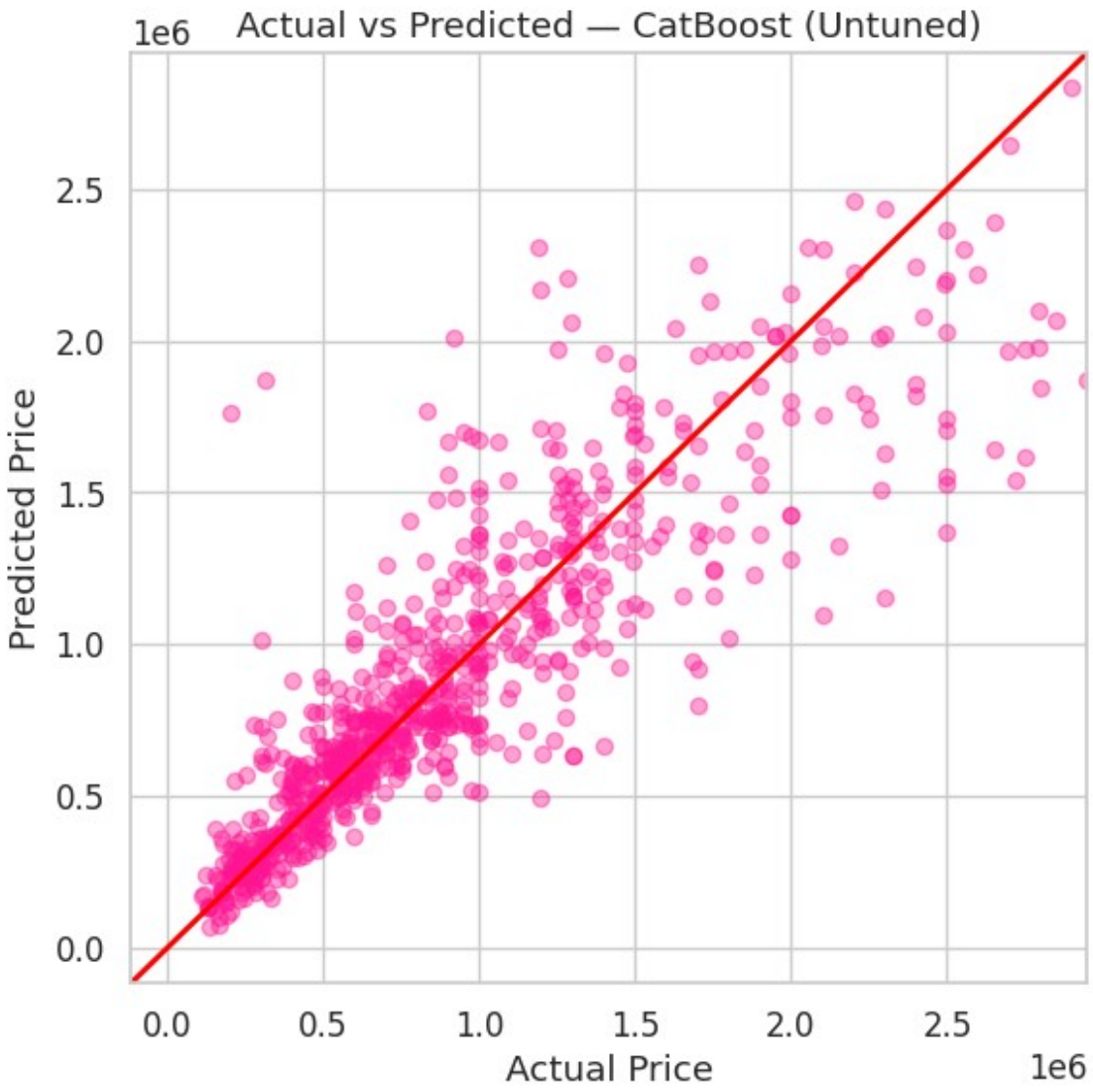

Actual vs Predicted — Gradient Boosting (Tuned)
1e6
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

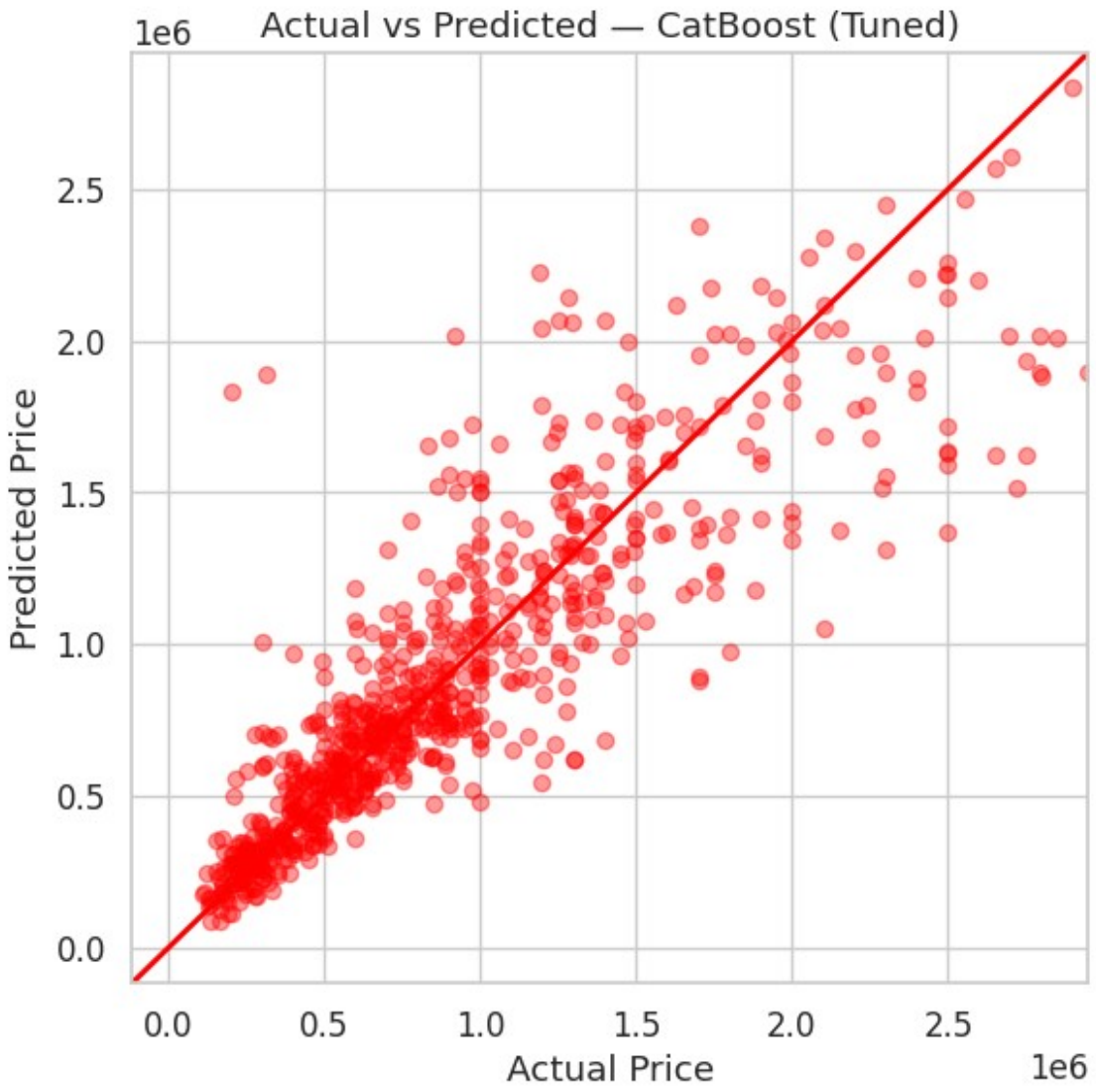

Actual vs Predicted — XGBoost (Untuned)
1e6
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6


Actual vs Predicted — XGBoost (Tuned)
1e6
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

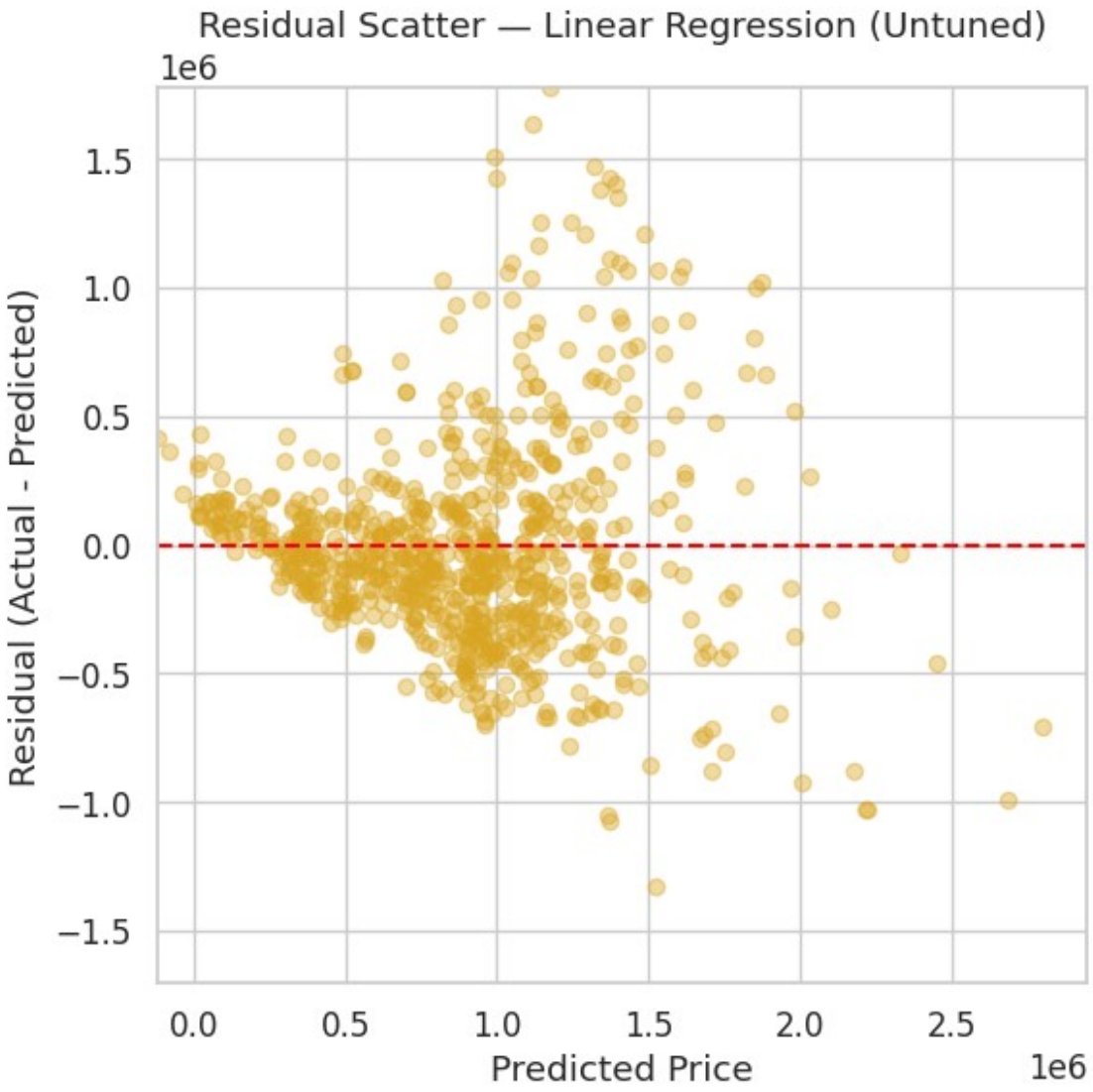

Actual vs Predicted — LightGBM (Untuned)
1e6
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

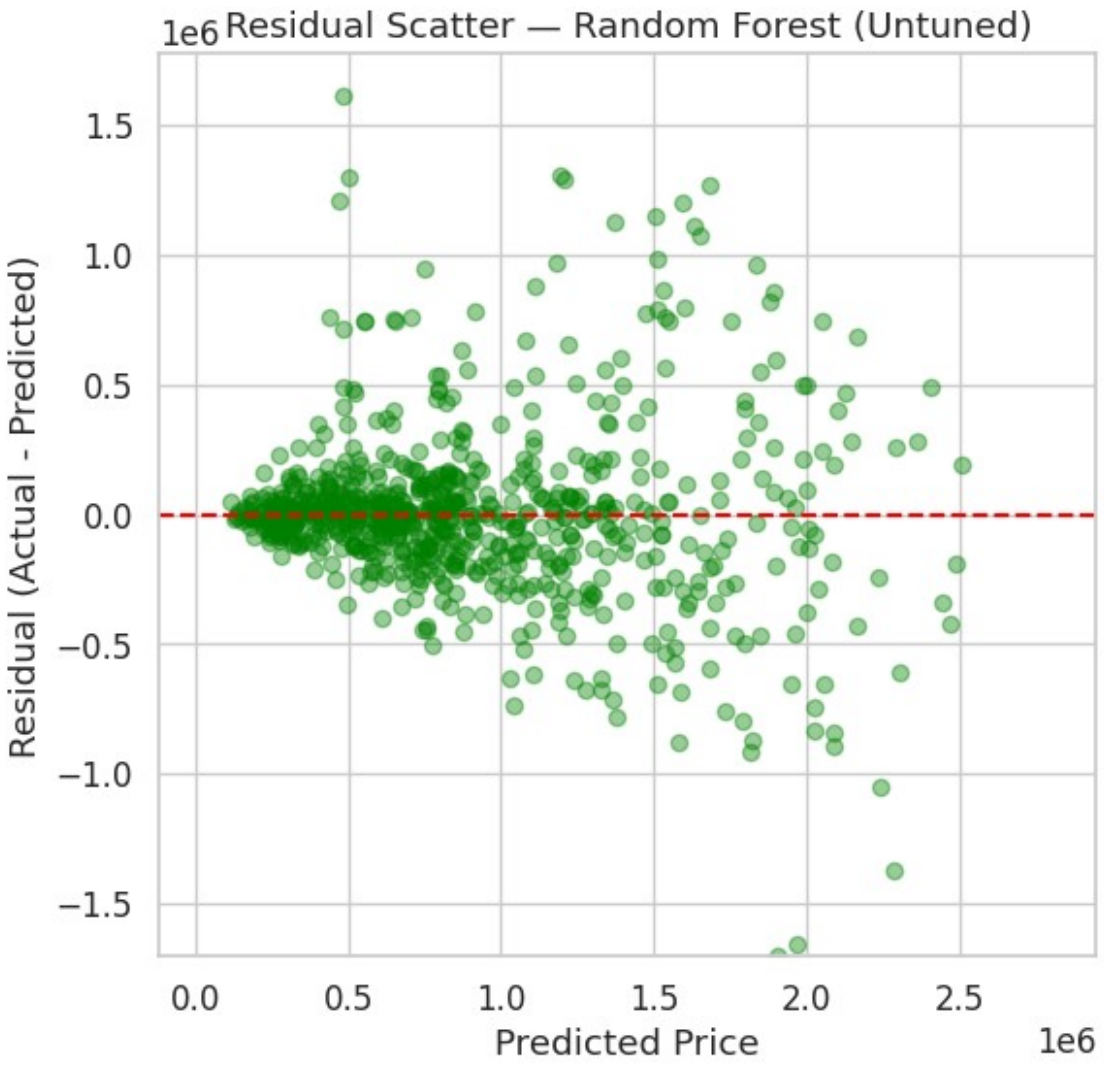

Actual vs Predicted — LightGBM (Tuned)
1e6
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

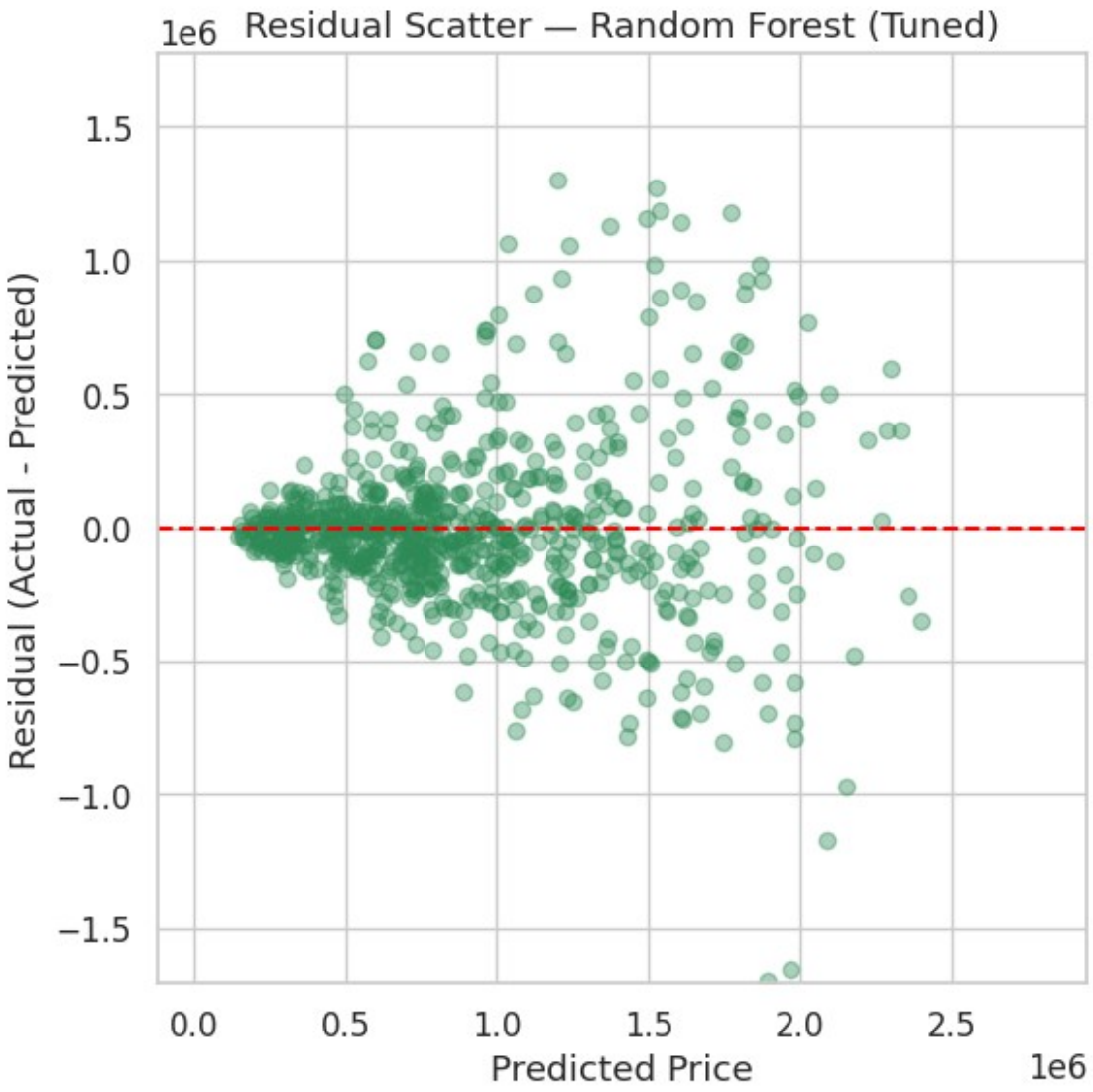

Actual vs Predicted — CatBoost (Untuned)
1e6
Predicted Price
0.0
0.5
1.0
1.5
2.0
2.5
Actual Price
1e6

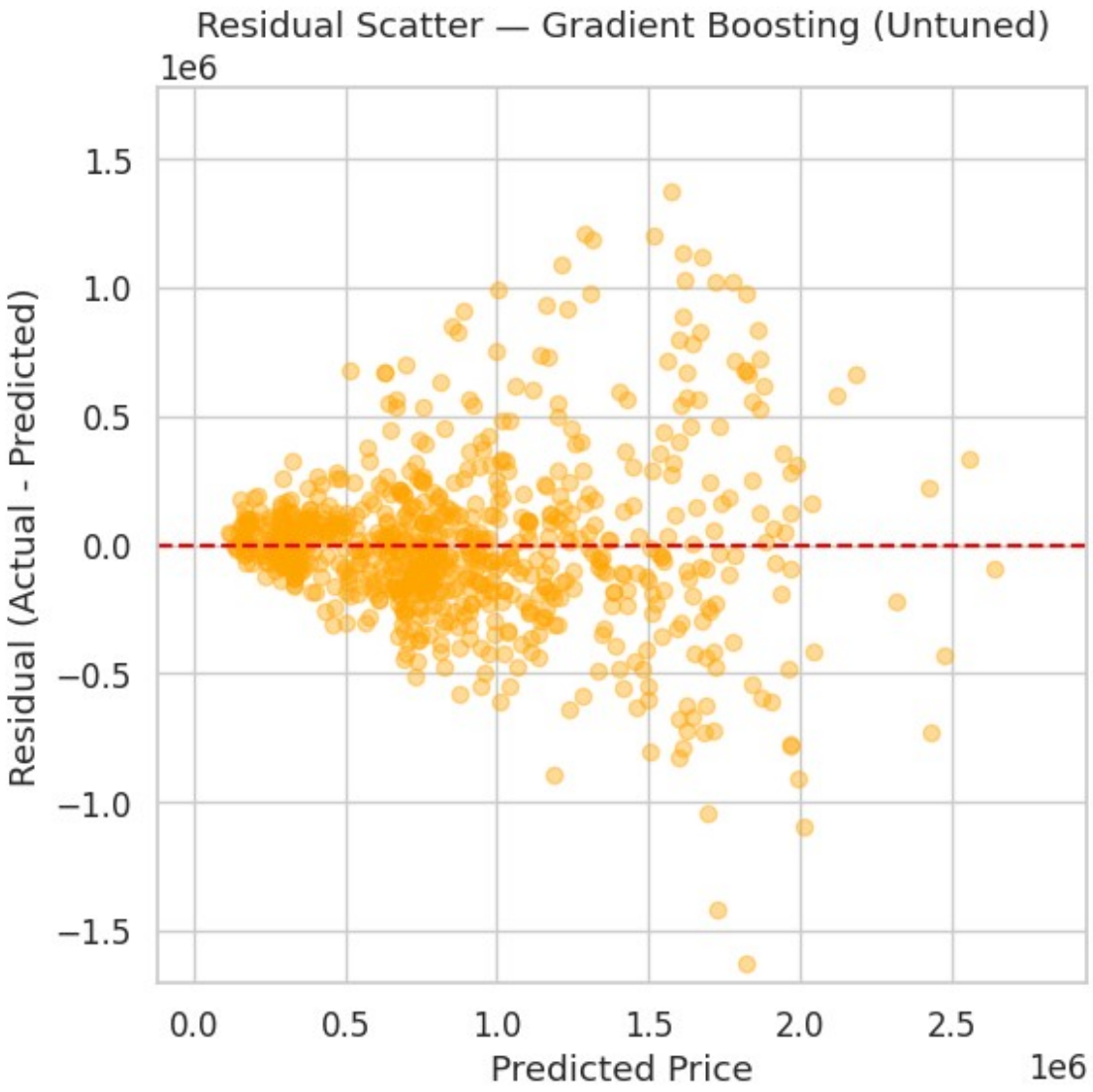


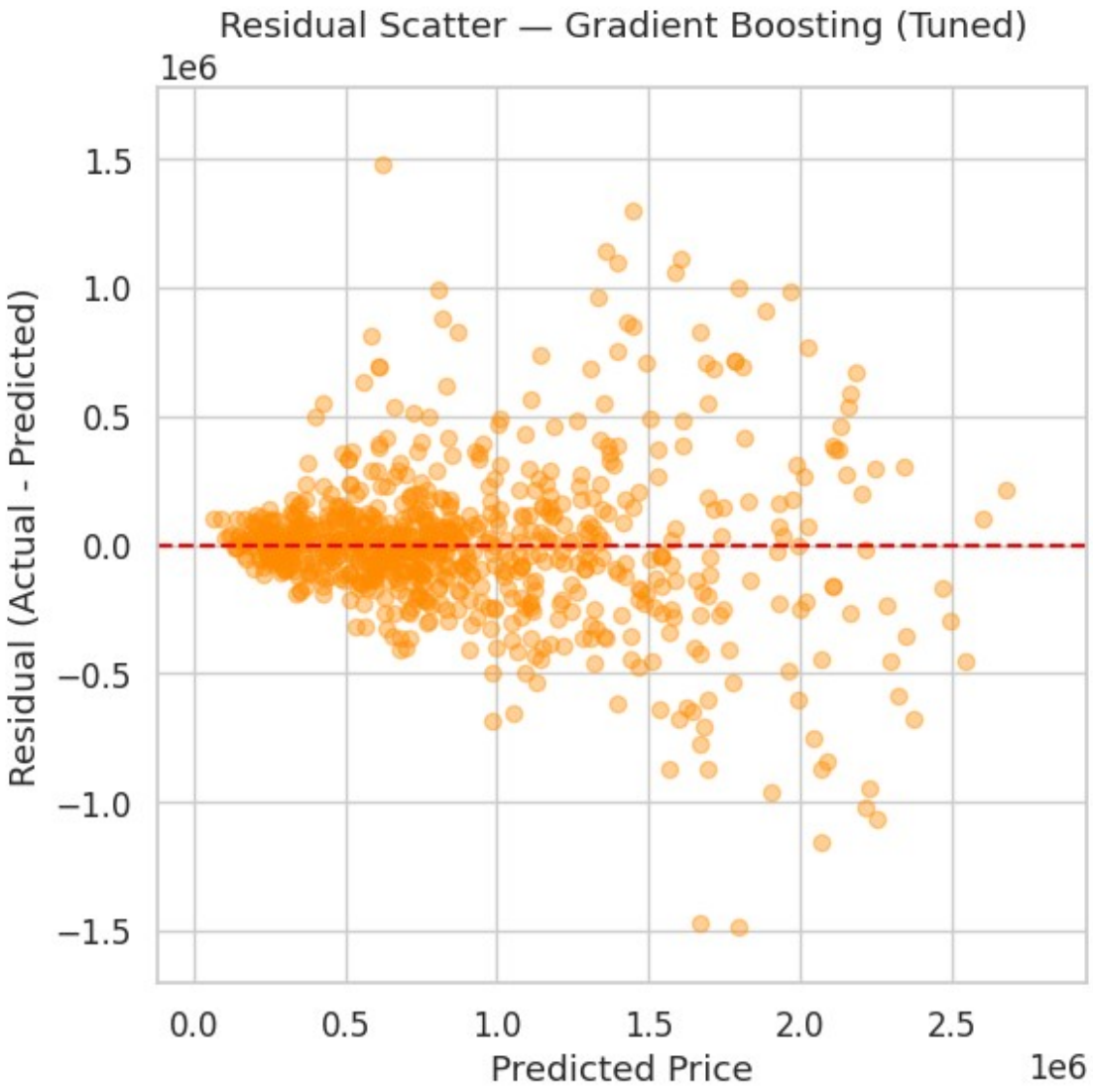

```
for name, yhat in preds.items():
    plot_residual_scatter_fixed(
        y_test, yhat,
        f"Residual Scatter — {name}",
        global_pred_min, global_pred_max,
        global_resid_min, global_resid_max,
        colors.get(name, "black")
    )
```

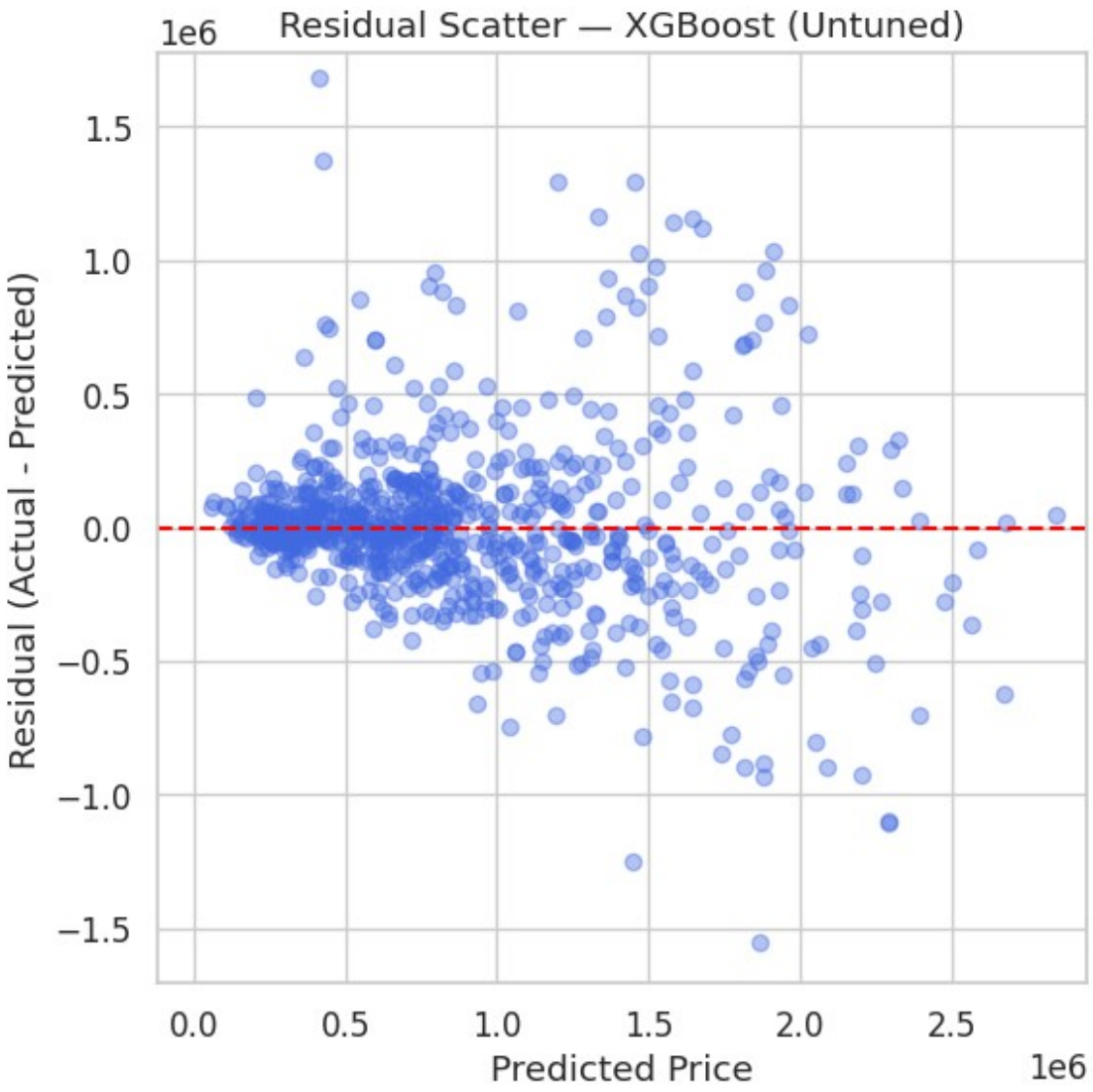

Residual Scatter — Linear Regression (Untuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6

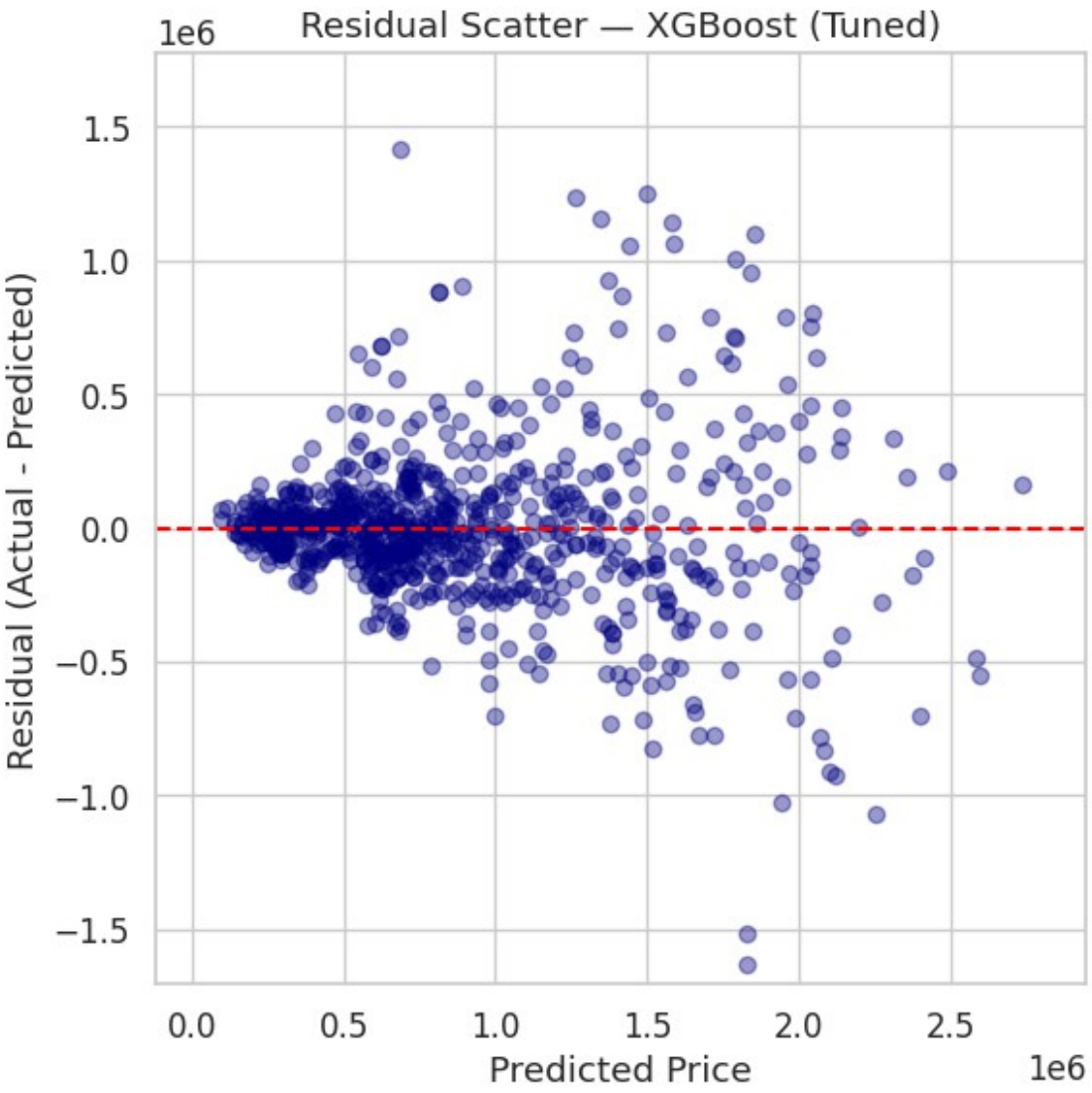

Residual Scatter — Random Forest (Untuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
1e6
Predicted Price

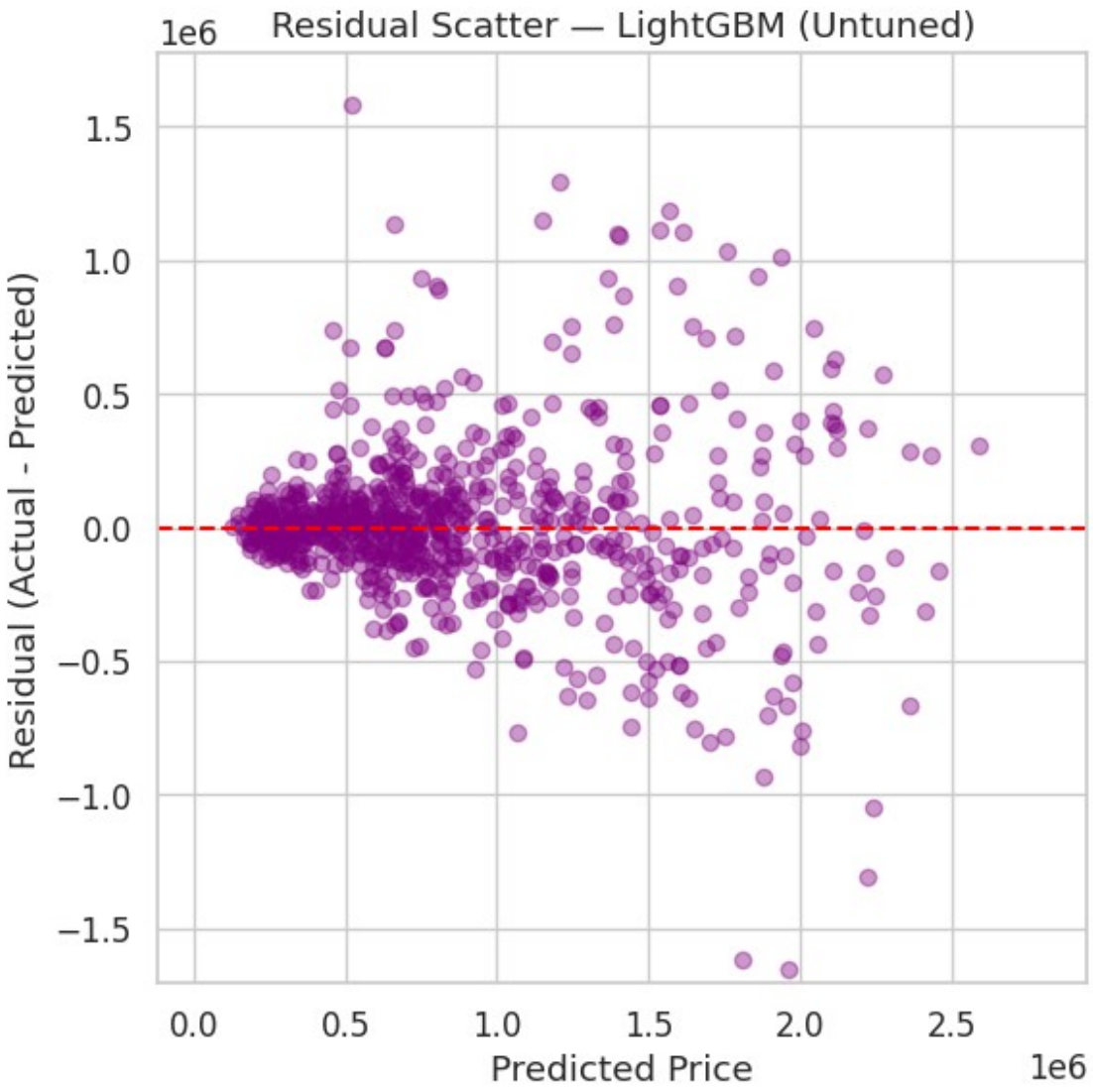

Residual Scatter — Random Forest (Tuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6

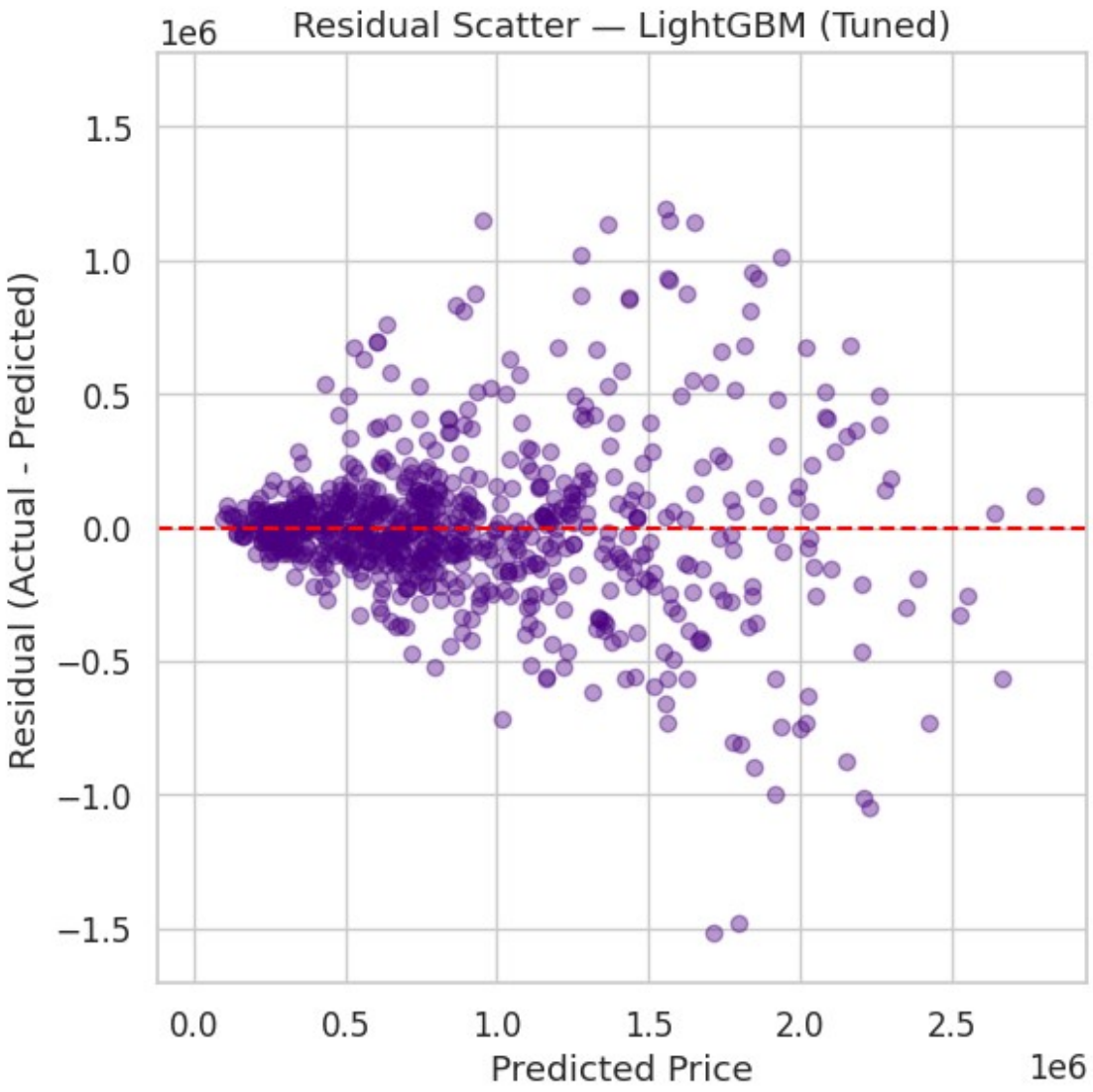

Residual Scatter — Gradient Boosting (Untuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6

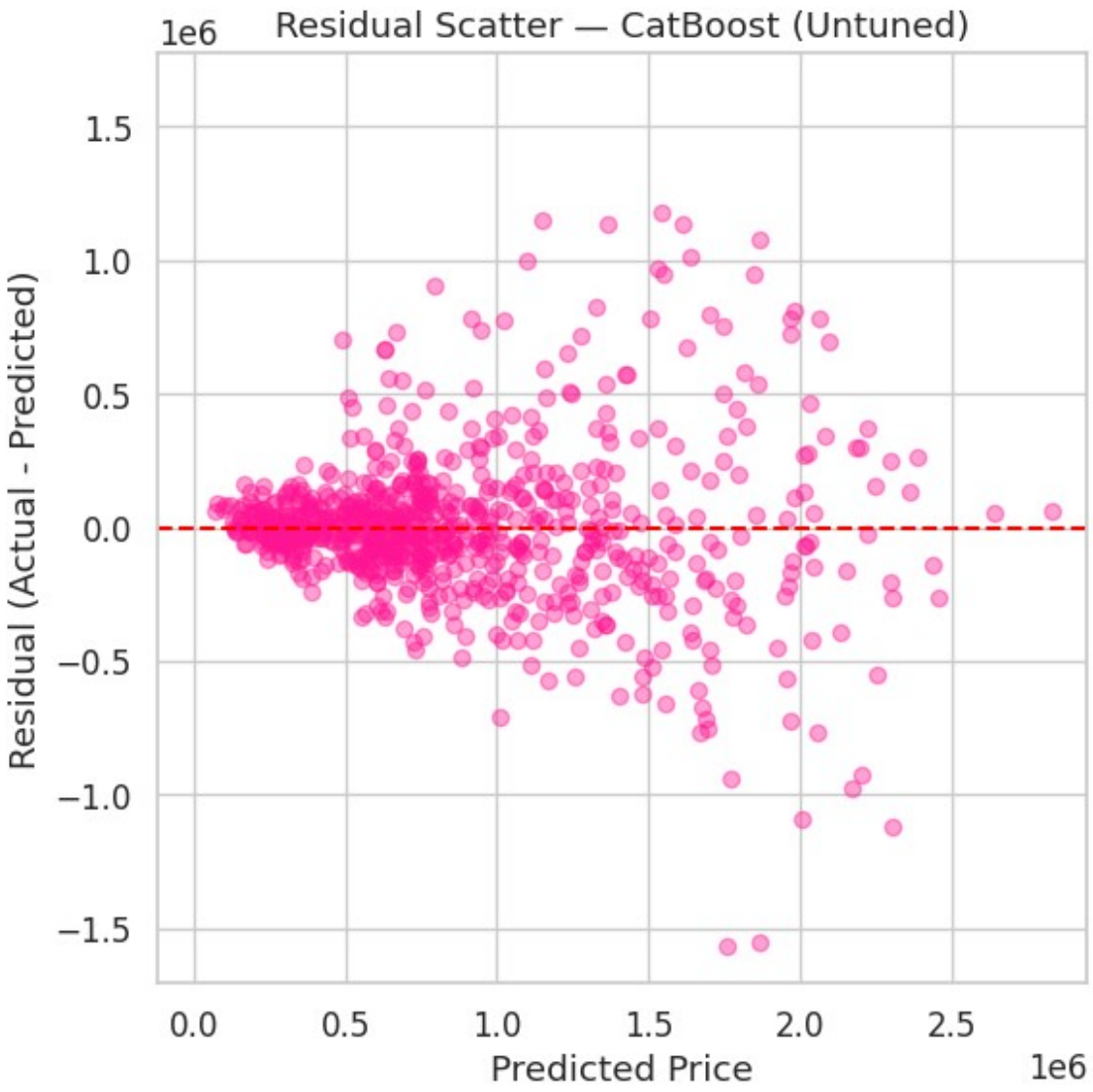

Residual Scatter — Gradient Boosting (Tuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6

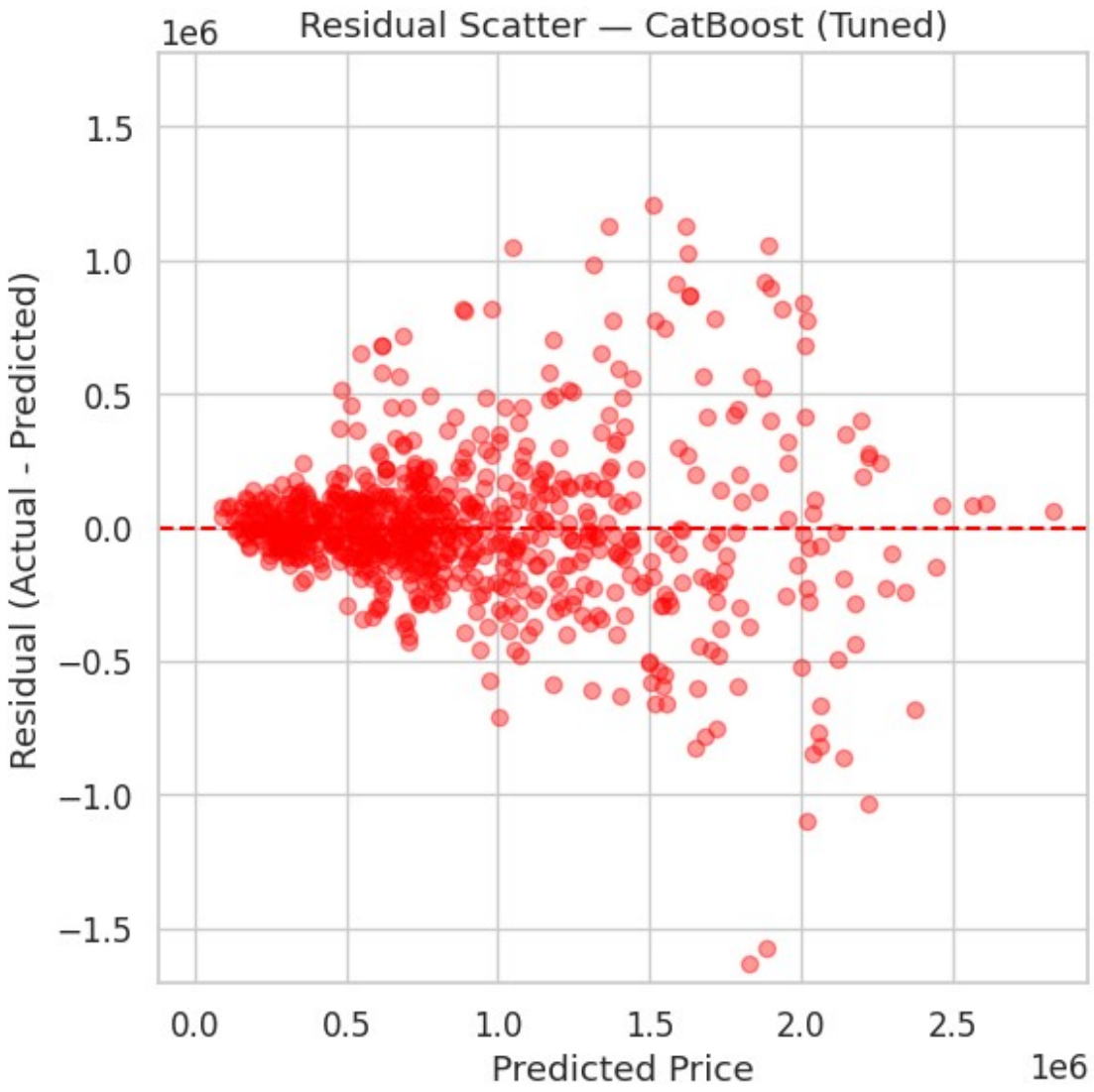

Residual Scatter — XGBoost (Untuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6


Residual Scatter — XGBoost (Tuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6


Residual Scatter — LightGBM (Untuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6


Residual Scatter — LightGBM (Tuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6

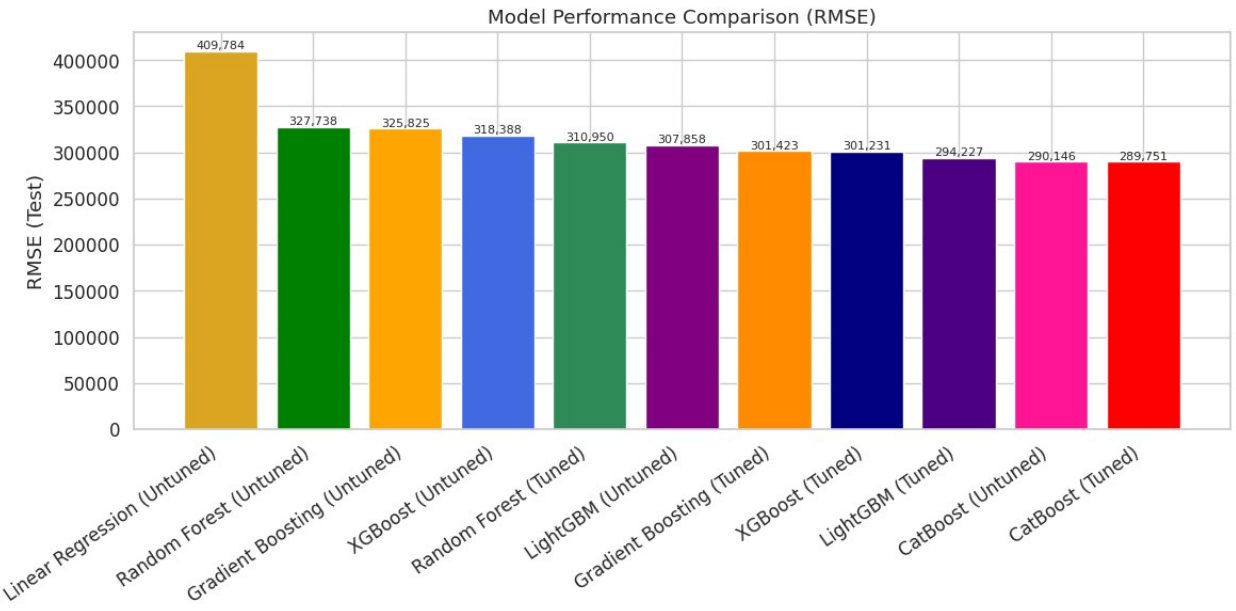

Residual Scatter — CatBoost (Untuned)
1e6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Residual (Actual - Predicted)
0.0
0.5
1.0
1.5
2.0
2.5
Predicted Price
1e6



#Results

```
rmse_values = []

for name, yhat in preds.items():
    rmse = np.sqrt(mean_squared_error(y_test, yhat))
    rmse_values.append((name, rmse))

# Sort models
rmse_values.sort(key=lambda x: x[1], reverse=True)

model_names = [m for m, _ in rmse_values]
rmse_nums   = [r for _, r in rmse_values]

plt.figure(figsize=(12,6))
bars = plt.bar(model_names, rmse_nums,
```

```
                color=[colors.get(name, "blue") for name in
model_names])

plt.ylabel("RMSE (Test)")
plt.title("Model Performance Comparison (RMSE)")

plt.xticks(rotation=35, ha="right")

# Annotate each bar
for bar, rmse in zip(bars, rmse_nums):
    plt.text(
        bar.get_x() + bar.get_width()/2,
        bar.get_height(),
        f"{rmse:,.0f}",
        ha="center", va="bottom", fontsize=8
    )

plt.tight_layout()
plt.show()
```




```
final_model_results = pd.DataFrame({
    'Model': [
        'Linear Regression',
        'Random Forest (Untuned)',
        'Random Forest (Tuned)',
        'Gradient Boosting (Untuned)',
        'Gradient Boosting (Tuned)',
        'XGBoost (Untuned)',
        'XGBoost (Tuned)',
        'LightGBM (Untuned)',
```

```
        'LightGBM (Tuned)',
        'CatBoost (Untuned)',
        'CatBoost (Tuned)'
    ],
    'RMSE': [
        lr_rmse,
        rf_rmse,
        rf_rmse_tuned,
        gb_rmse,
        gb_rmse_tuned,
        xgb_rmse,
        xgb_rmse_tuned,
        lgb_rmse,
        lgb_rmse_tuned,
        cat_rmse,
        cat_rmse_tuned
    ],
    'MAE': [
        lr_mae,
        rf_mae,
        rf_mae_tuned,
        gb_mae,
        gb_mae_tuned,
        xgb_mae,
        xgb_mae_tuned,
        lgb_mae,
        lgb_mae_tuned,
        cat_mae,
        cat_mae_tuned
    ],
    'R²': [
        lr_r2,
        rf_r2,
        rf_r2_tuned,
        gb_r2,
        gb_r2_tuned,
        xgb_r2,
        xgb_r2_tuned,
        lgb_r2,
        lgb_r2_tuned,
        cat_r2,
        cat_r2_tuned
    ]
})

final_model_results_style = final_model_results.copy()
final_model_results_style['RMSE'] =
final_model_results_style['RMSE'].apply(lambda x: f"${x:,.0f}")
final_model_results_style['MAE']  =
```

```
final_model_results_style['MAE'].apply(lambda x: f"${x:,.0f}")
final_model_results_style['R²']   =
final_model_results_style['R²'].round(3)

final_model_results_style
```

```
{"summary":"{\n  \"name\": \"final_model_results_style\",\n  \"rows\":
11,\n  \"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 11,\n        \"samples\": [\n
\"XGBoost (Untuned)\",\n          \"Linear Regression\",\n
\"CatBoost (Untuned)\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RMSE\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 11,\n        \"samples\": [\n
\"$318,388\",\n          \"$409,784\",\n          \"$290,146\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"MAE\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 11,\n
\"samples\": [\n          \"$202,538\",\n          \"$294,896\",\n
\"$186,207\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"R\\
u00b2\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.06817397530649733,\n        \"min\": 0.516,\n
\"max\": 0.758,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.708,\n          0.516,\n          0.757\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\
n}","type":"dataframe","variable_name":"final_model_results_style"}
```

```
import pandas as pd
from sklearn.model_selection import cross_val_score

def get_cv_scores(model, X, y):
    rmse_scores = -cross_val_score(model, X, y, cv=kf,

scoring="neg_root_mean_squared_error")
    mae_scores  = -cross_val_score(model, X, y, cv=kf,
                                   scoring="neg_mean_absolute_error")
    r2_scores   =  cross_val_score(model, X, y, cv=kf, scoring="r2")
    return rmse_scores.mean(), mae_scores.mean(), r2_scores.mean()

lr_rmse_cv,  lr_mae_cv,  lr_r2_cv  = get_cv_scores(lr_cv,
X_for_lr_scaled, y)

rf_rmse_cv,  rf_mae_cv,  rf_r2_cv  = get_cv_scores(rf_cv,
X_for_trees,     y)
rf_rmse_t_cv, rf_mae_t_cv, rf_r2_t_cv = get_cv_scores(rf_tuned,
X_for_trees, y)
```

```
gb_rmse_cv,  gb_mae_cv,  gb_r2_cv  = get_cv_scores(gb_cv,
X_for_trees,     y)
gb_rmse_t_cv, gb_mae_t_cv, gb_r2_t_cv = get_cv_scores(gb_tuned,
X_for_trees, y)

xgb_rmse_cv,  xgb_mae_cv,  xgb_r2_cv  = get_cv_scores(xgb_cv,
X_for_trees, y)
xgb_rmse_t_cv, xgb_mae_t_cv, xgb_r2_t_cv = get_cv_scores(xgb_tuned,
X_for_trees, y)

lgb_rmse_cv,  lgb_mae_cv,  lgb_r2_cv  = get_cv_scores(lgb_cv,
X_for_trees, y)
lgb_rmse_t_cv, lgb_mae_t_cv, lgb_r2_t_cv = get_cv_scores(lgb_tuned,
X_for_trees, y)

cat_rmse_cv,  cat_mae_cv,  cat_r2_cv  = get_cv_scores(cat_cv,
X_for_trees, y)
cat_rmse_t_cv, cat_mae_t_cv, cat_r2_t_cv = get_cv_scores(cat_tuned,
X_for_trees, y)

# ---- compact CV results table ----
cv_model_results = pd.DataFrame({
    "Model": [
        "Linear Regression",
        "Random Forest (Untuned)",
        "Random Forest (Tuned)",
        "Gradient Boosting (Untuned)",
        "Gradient Boosting (Tuned)",
        "XGBoost (Untuned)",
        "XGBoost (Tuned)",
        "LightGBM (Untuned)",
        "LightGBM (Tuned)",
        "CatBoost (Untuned)",
        "CatBoost (Tuned)"
    ],
    "RMSE": [
        lr_rmse_cv,
        rf_rmse_cv,
        rf_rmse_t_cv,
        gb_rmse_cv,
        gb_rmse_t_cv,
        xgb_rmse_cv,
        xgb_rmse_t_cv,
        lgb_rmse_cv,
        lgb_rmse_t_cv,
        cat_rmse_cv,
        cat_rmse_t_cv
    ],
    "MAE": [
        lr_mae_cv,
```

```
        rf_mae_cv,
        rf_mae_t_cv,
        gb_mae_cv,
        gb_mae_t_cv,
        xgb_mae_cv,
        xgb_mae_t_cv,
        lgb_mae_cv,
        lgb_mae_t_cv,
        cat_mae_cv,
        cat_mae_t_cv
    ],
    "R²": [
        lr_r2_cv,
        rf_r2_cv,
        rf_r2_t_cv,
        gb_r2_cv,
        gb_r2_t_cv,
        xgb_r2_cv,
        xgb_r2_t_cv,
        lgb_r2_cv,
        lgb_r2_t_cv,
        cat_r2_cv,
        cat_r2_t_cv
    ]
})

cv_model_results_style = cv_model_results.copy()
cv_model_results_style["RMSE"] =
cv_model_results_style["RMSE"].apply(lambda x: f"${x:,.0f}")
cv_model_results_style["MAE"]  =
cv_model_results_style["MAE"].apply(lambda x: f"${x:,.0f}")
cv_model_results_style["R²"]   = cv_model_results_style["R²"].round(3)

cv_model_results_style
```

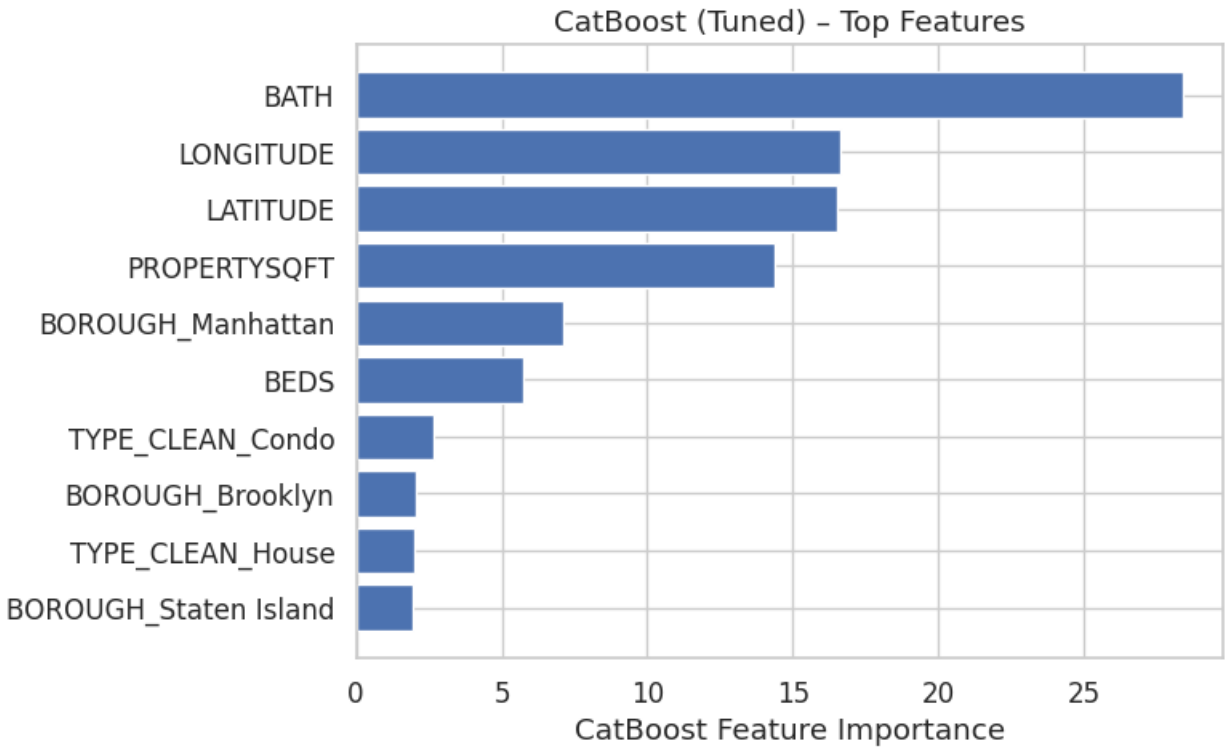

```
{"summary":"{\n  \"name\": \"cv_model_results_style\",\n  \"rows\":
11,\n  \"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 11,\n        \"samples\": [\n
\"XGBoost (Untuned)\",\n          \"Linear Regression\",\n
\"CatBoost (Untuned)\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RMSE\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 11,\n        \"samples\": [\n
\"$303,564\",\n          \"$424,661\",\n          \"$293,004\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"MAE\",\n      \"properties\": {\
n        \"dtype\": \"string\",\n        \"num_unique_values\": 11,\n
\"samples\": [\n          \"$197,072\",\n          \"$297,588\",\n
\"$191,178\"\n        ],\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"R\\
u00b2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.0731615895249562,\n   \"min\": 0.519,\n
\"max\": 0.771,\n   \"num_unique_values\": 11,\n
\"samples\": [\n   0.753,\n   0.519,\n   0.771\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\
n}","type":"dataframe","variable_name":"cv_model_results_style"}

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

cat_importances =
cat_tuned.get_feature_importance(type="FeatureImportance")

feature_importance_df = pd.DataFrame({
    "Feature": X_train_rf.columns,
    "Importance": cat_importances
}).sort_values("Importance", ascending=False)


top10_cat_importance = feature_importance_df.head(10).copy()
top10_cat_importance
```

{"summary":"{\n \"name\": \"top10_cat_importance\",\n \"rows\": 10,\
n \"fields\": [\n   {\n   \"column\": \"Feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n   \"samples\": [\n
\"TYPE_CLEAN_House\",\n   \"LONGITUDE\",\n   \"BEDS\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Importance\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
8.914309731757657,\n   \"min\": 1.9444873207972682,\n
\"max\": 28.367721293354535,\n   \"num_unique_values\": 10,\n
\"samples\": [\n   1.9932809350222367,\n
16.631114814600082,\n   5.719089219303404\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\
n}","type":"dataframe","variable_name":"top10_cat_importance"}

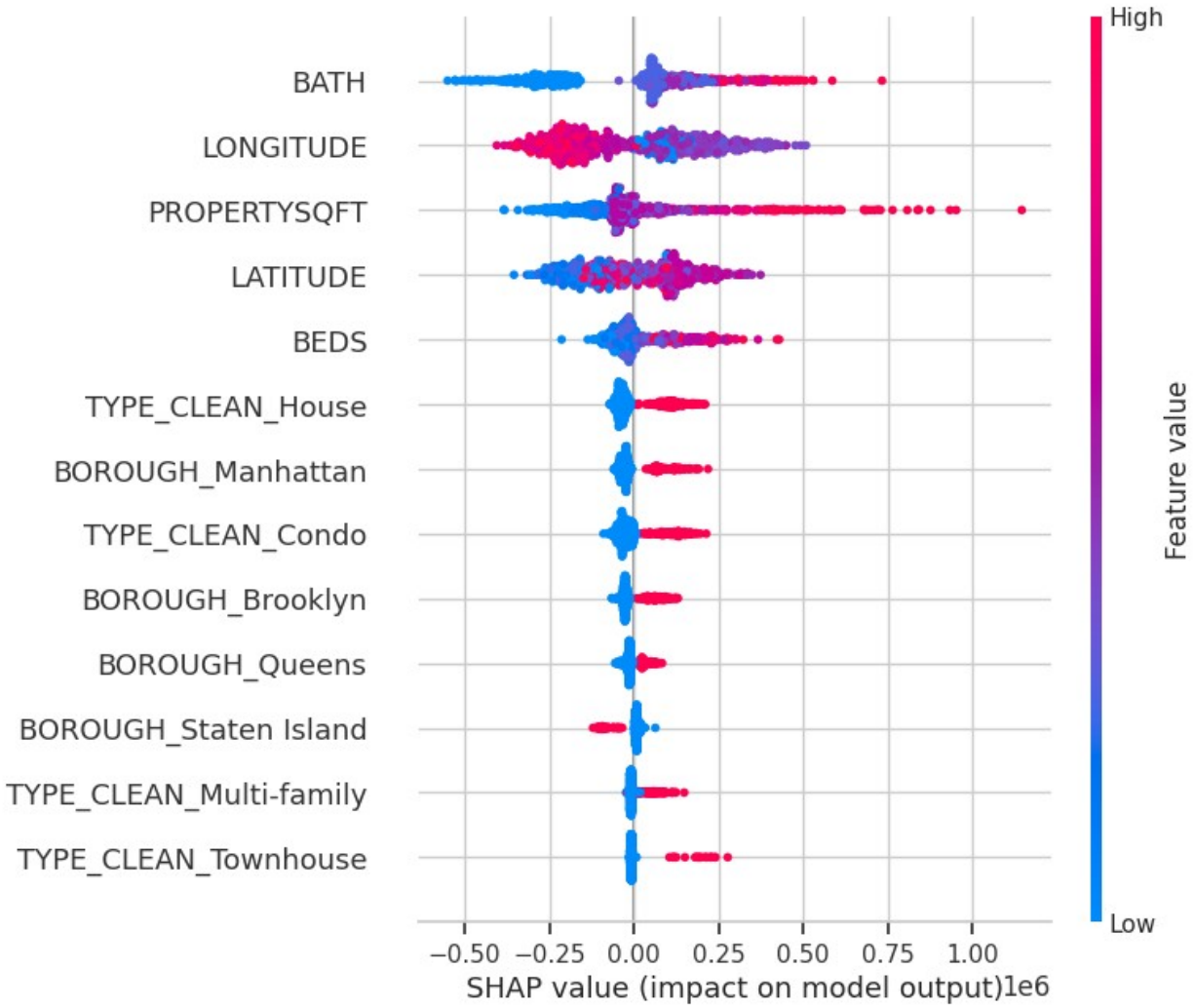
```
top_n = 10
top = feature_importance_df.head(top_n)

plt.figure(figsize=(8, 5))
plt.barh(top["Feature"][::-1], top["Importance"][::-1])
plt.xlabel("CatBoost Feature Importance")
plt.title("CatBoost (Tuned) — Top Features")
plt.tight_layout()
plt.show()
```



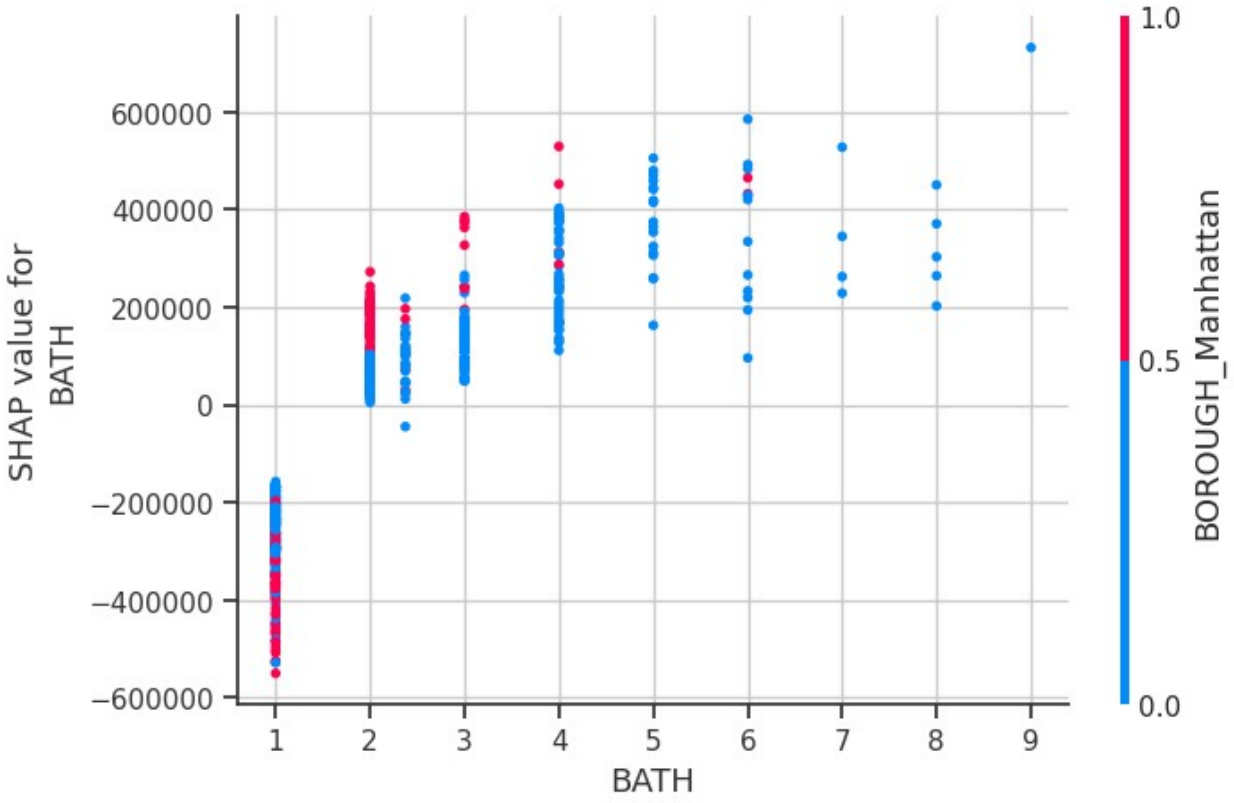
#SHAP

```
import shap
import matplotlib.pyplot as plt


explainer = shap.TreeExplainer(cat_tuned)
shap_values = explainer.shap_values(X_test_rf)

#SUMMARY PLOT
plt.figure(figsize=(8, 8))
shap.summary_plot(
    shap_values,
    X_test_rf,
    plot_type="dot",
    show=True
)

#DEPENDENCE: BATH
plt.figure(figsize=(8, 5))
shap.dependence_plot(
    "BATH",
    shap_values,
    X_test_rf,
    interaction_index="BOROUGH_Manhattan"
)
```

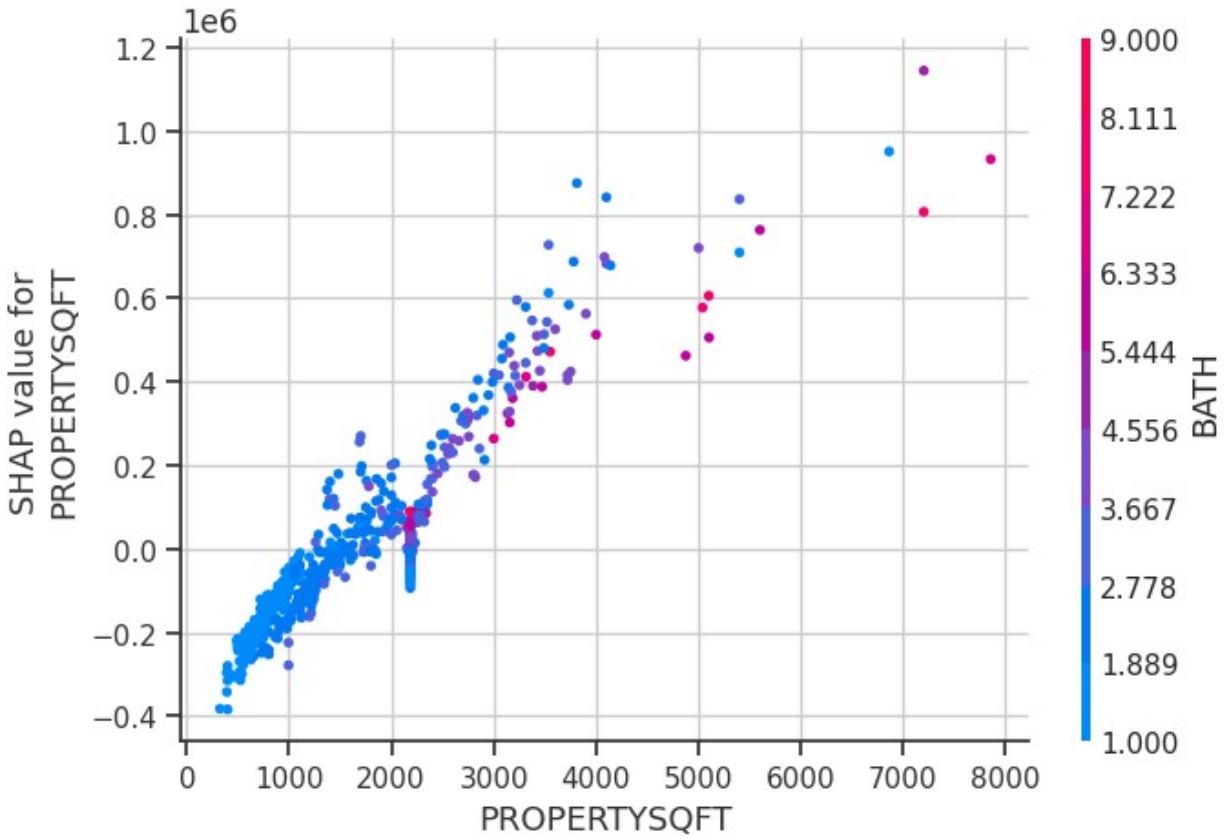
```
#DEPENDENCE: PROPERTYSQFT
plt.figure(figsize=(8, 5))
shap.dependence_plot(
    "PROPERTYSQFT",
    shap_values,
    X_test_rf,
    interaction_index="BATH"
)

# DEPENDENCE: LATITUDE
plt.figure(figsize=(8, 5))
shap.dependence_plot(
    "LATITUDE",
    shap_values,
    X_test_rf,
    interaction_index="BEDS"
)
```



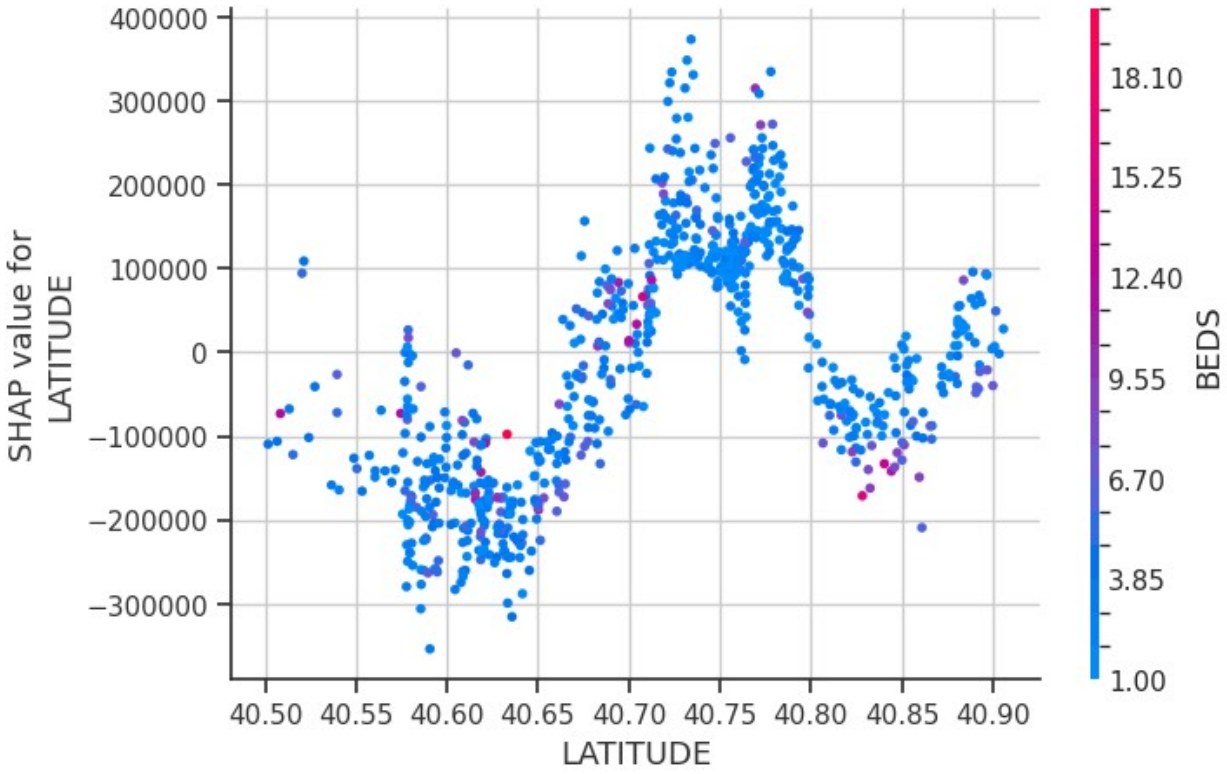
```
<Figure size 800x500 with 0 Axes>
```




```
<Figure size 800x500 with 0 Axes>
```




```
<Figure size 800x500 with 0 Axes>
```


400000
300000
200000
100000
0
−100000
−200000
−300000
SHAP value for
LATITUDE
40.50
40.55
40.60
40.65
40.70
40.75
40.80
40.85
40.90
LATITUDE
18.10
15.25
12.40
9.55
6.70
3.85
1.00
BEDS