House Prices Regression Task

Matthew Loh

Table of contents


```python
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, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error,
    r2_score,
    accuracy_score,
)
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgb
import xgboost as xgb
import catboost as cbt
```

```python
np.random.seed(42)
```

Data Loading and Exploratory Data Analysis

```python
# Load the data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

# Display basic information about the dataset
print(train.info())

# Display summary statistics
print(train.describe())

# Plot distribution of target variable
plt.figure(figsize=(10, 6))
sns.histplot(train["SalePrice"], kde=True)
plt.title("Distribution of Sale Prices")
plt.show()

# Scatter plot of GrLivArea vs SalePrice
plt.figure(figsize=(10, 6))
plt.scatter(train["GrLivArea"], train["SalePrice"], alpha=0.5)
plt.title("GrLivArea vs SalePrice")
plt.xlabel("GrLivArea (Above ground living area)")
plt.ylabel("SalePrice")
plt.show()

# Box plot of SalePrice by OverallQual
plt.figure(figsize=(12, 6))
sns.boxplot(x="OverallQual", y="SalePrice", data=train)
plt.title("SalePrice by Overall Quality")
plt.show()


# Identify numeric columns
numeric_columns = train.select_dtypes(include=[np.number]).columns

# Correlation matrix of numerical features
corr_matrix = train[numeric_columns].corr()
```

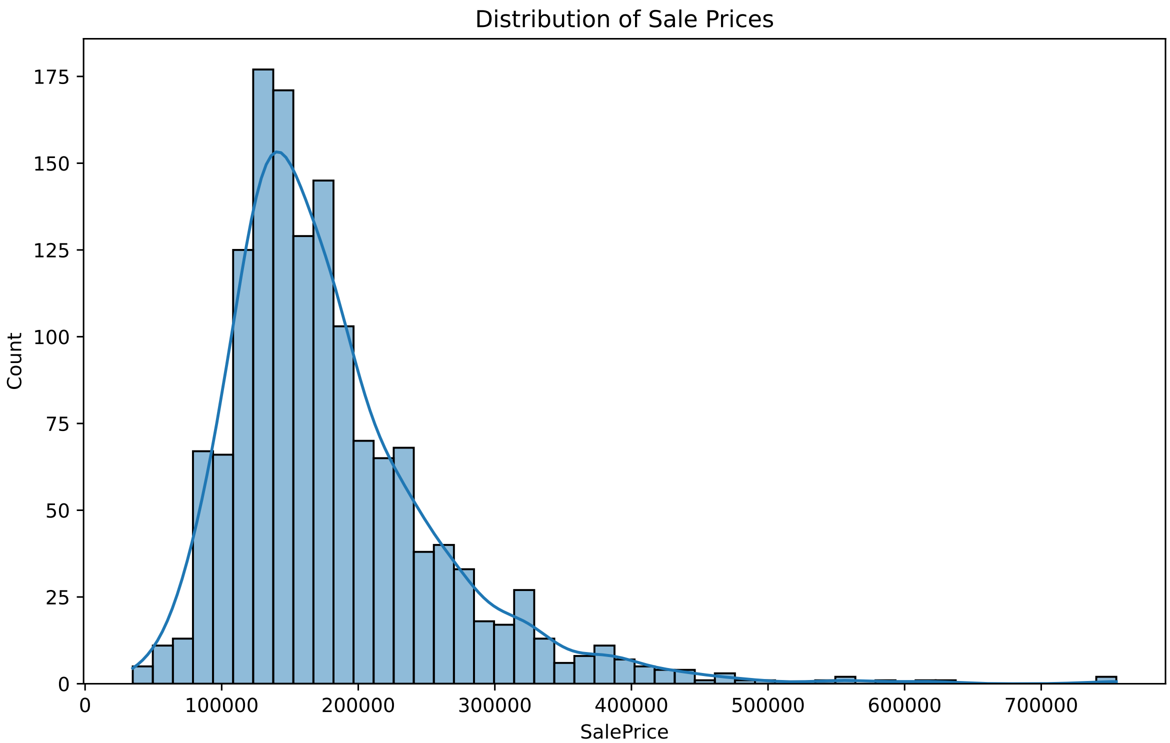
```
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, cmap="coolwarm", annot=False)
plt.title("Correlation Matrix of Numerical Features")
plt.show()

# Top 10 features correlated with SalePrice
top_corr = corr_matrix["SalePrice"].sort_values(ascending=False).head(11)
plt.figure(figsize=(10, 6))
sns.barplot(x=top_corr.index[1:], y=top_corr.values[1:])
plt.title("Top 10 Features Correlated with SalePrice")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Scatter plot of top correlated feature vs SalePrice
top_feature = top_corr.index[1]
plt.figure(figsize=(10, 6))
sns.scatterplot(x=train[top_feature], y=train["SalePrice"])
plt.title(f"{top_feature} vs SalePrice")
plt.show()
```

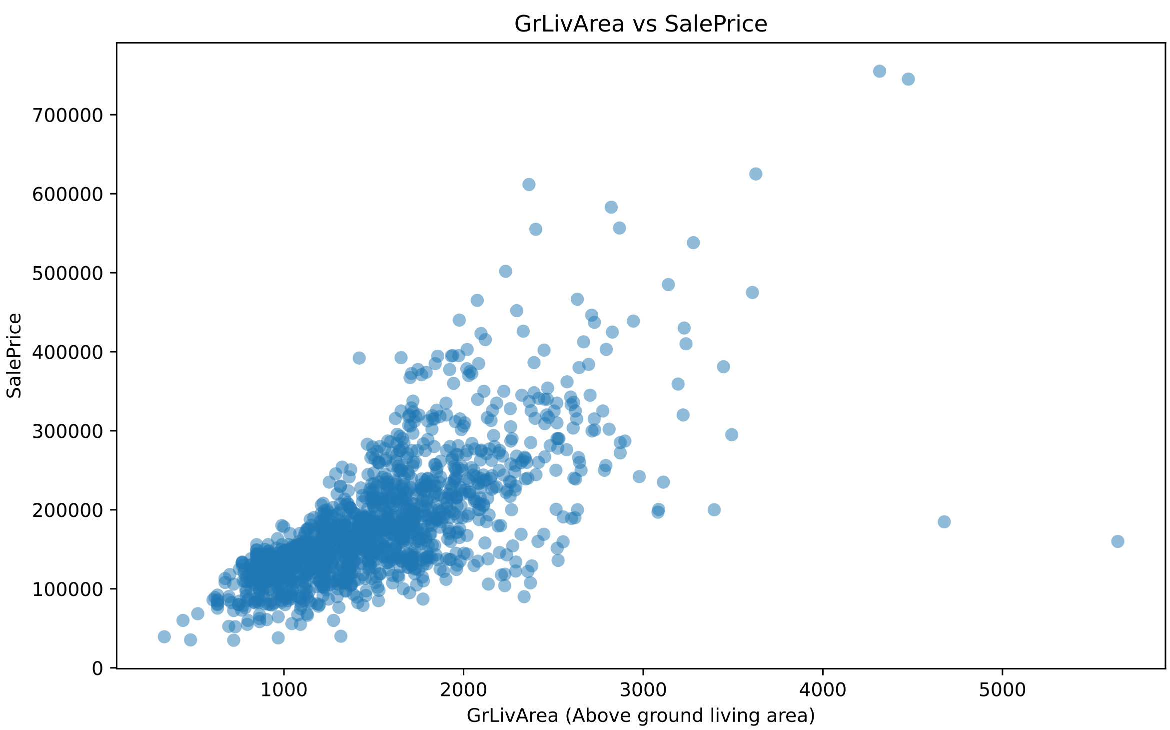
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
```

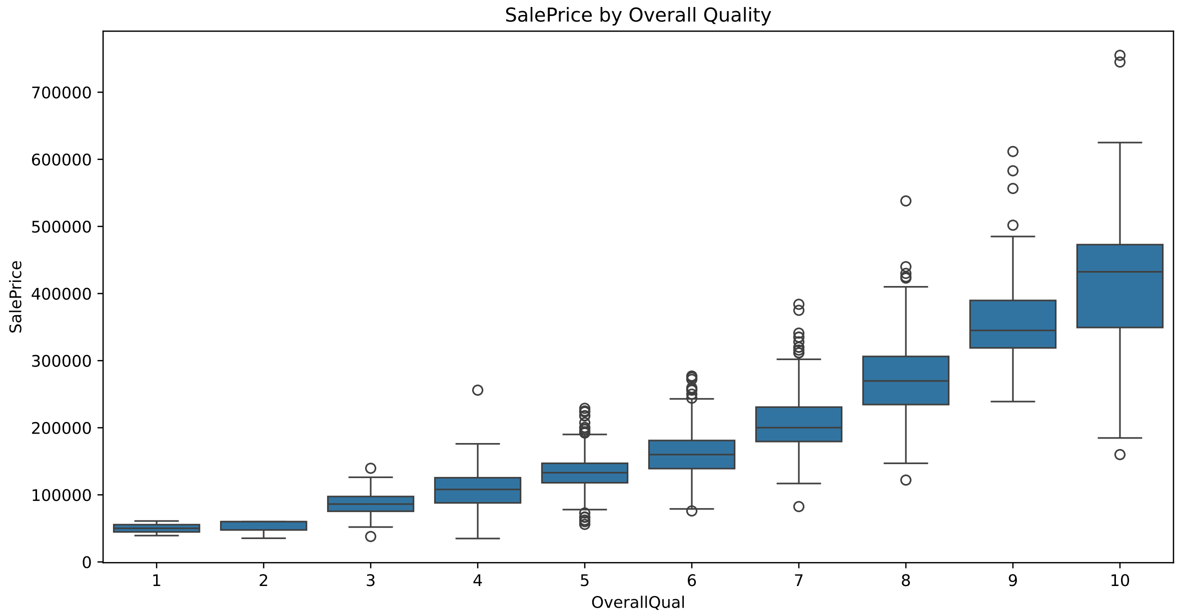
```
16  HouseStyle     1460 non-null   object
17  OverallQual    1460 non-null   int64
18  OverallCond    1460 non-null   int64
19  YearBuilt      1460 non-null   int64
20  YearRemodAdd   1460 non-null   int64
21  RoofStyle      1460 non-null   object
22  RoofMatl       1460 non-null   object
23  Exterior1st    1460 non-null   object
24  Exterior2nd    1460 non-null   object
25  MasVnrType     588 non-null    object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
```

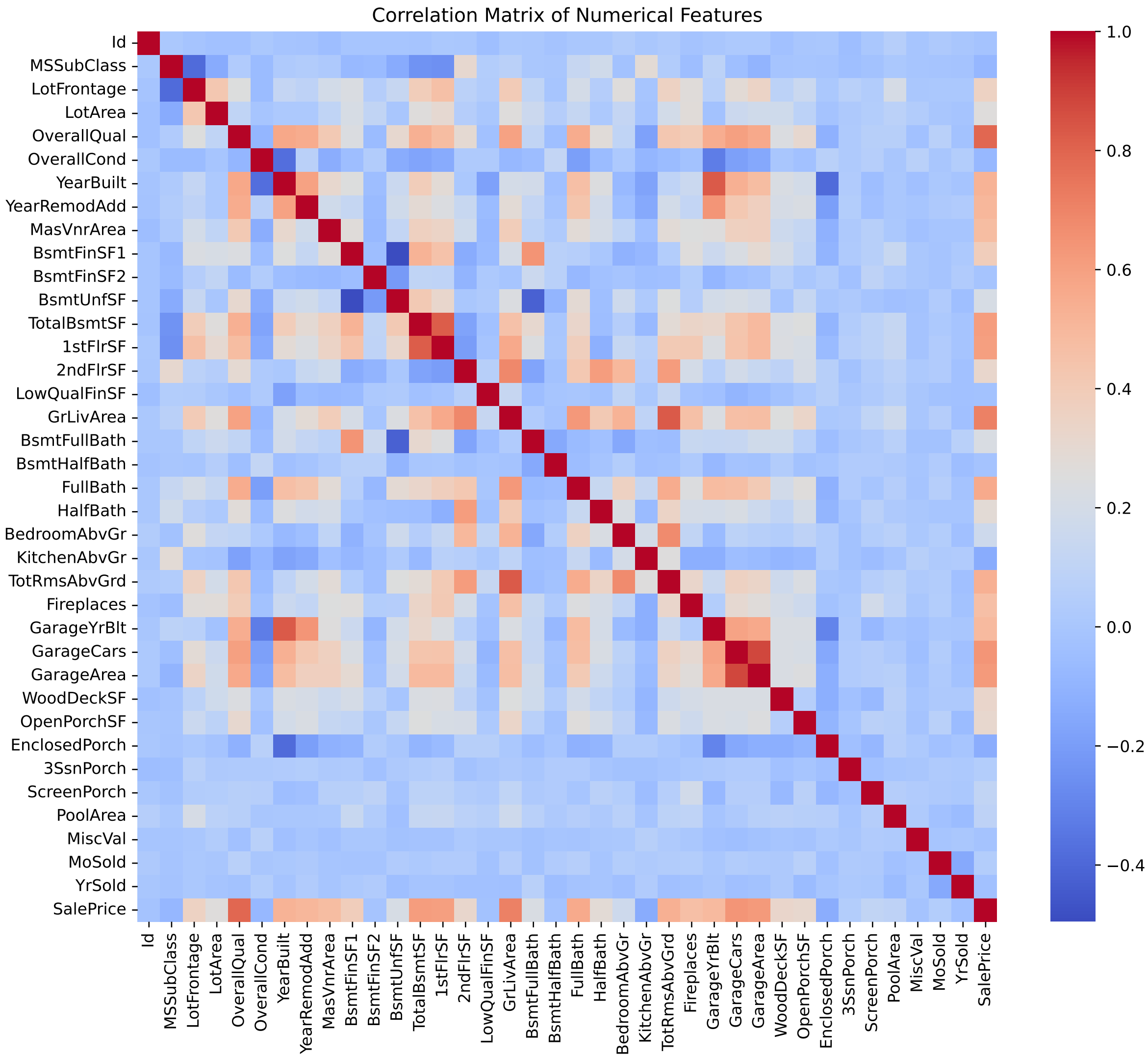
```
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
                Id   MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000  ...
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726  ...
std       1.112799    30.202904     20.645407   181.066207   456.098091  ...
min       1.000000  1872.000000   1950.000000     0.000000     0.000000  ...
25%       5.000000  1954.000000   1967.000000     0.000000     0.000000  ...
50%       5.000000  1973.000000   1994.000000     0.000000   383.500000  ...
75%       6.000000  2000.000000   2004.000000   166.000000   712.250000  ...
```

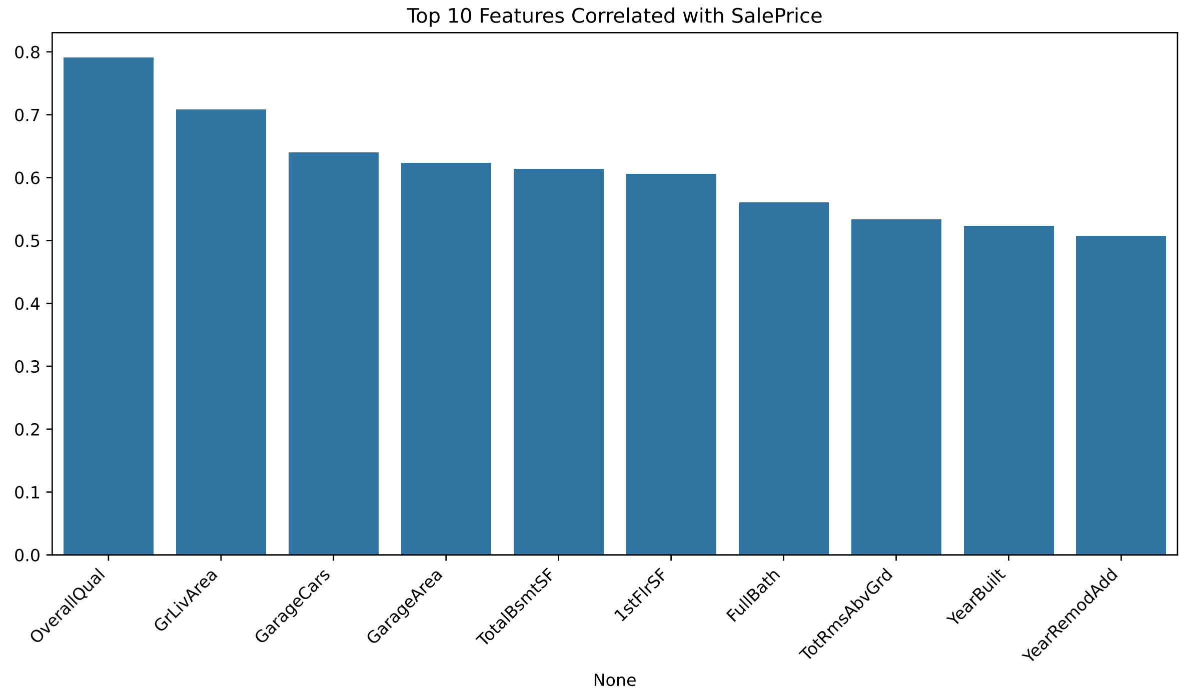
```
max        9.000000  2010.000000   2010.000000  1600.000000  5644.000000  ...

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000  1460.000000
mean     94.244521    46.660274      21.954110     3.409589    15.060959
std     125.338794    66.256028      61.119149    29.317331    55.757415
min       0.000000     0.000000       0.000000     0.000000     0.000000
25%       0.000000     0.000000       0.000000     0.000000     0.000000
50%       0.000000    25.000000       0.000000     0.000000     0.000000
75%     168.000000    68.000000       0.000000     0.000000     0.000000
max     857.000000   547.000000     552.000000   508.000000   480.000000

          PoolArea      MiscVal       MoSold       YrSold      SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000    1460.000000
mean      2.758904     43.489041     6.321918  2007.815753  180921.195890
std      40.177307    496.123024     2.703626     1.328095   79442.502883
min       0.000000      0.000000     1.000000  2006.000000   34900.000000
25%       0.000000      0.000000     5.000000  2007.000000  129975.000000
50%       0.000000      0.000000     6.000000  2008.000000  163000.000000
75%       0.000000      0.000000     8.000000  2009.000000  214000.000000
max     738.000000  15500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]
```

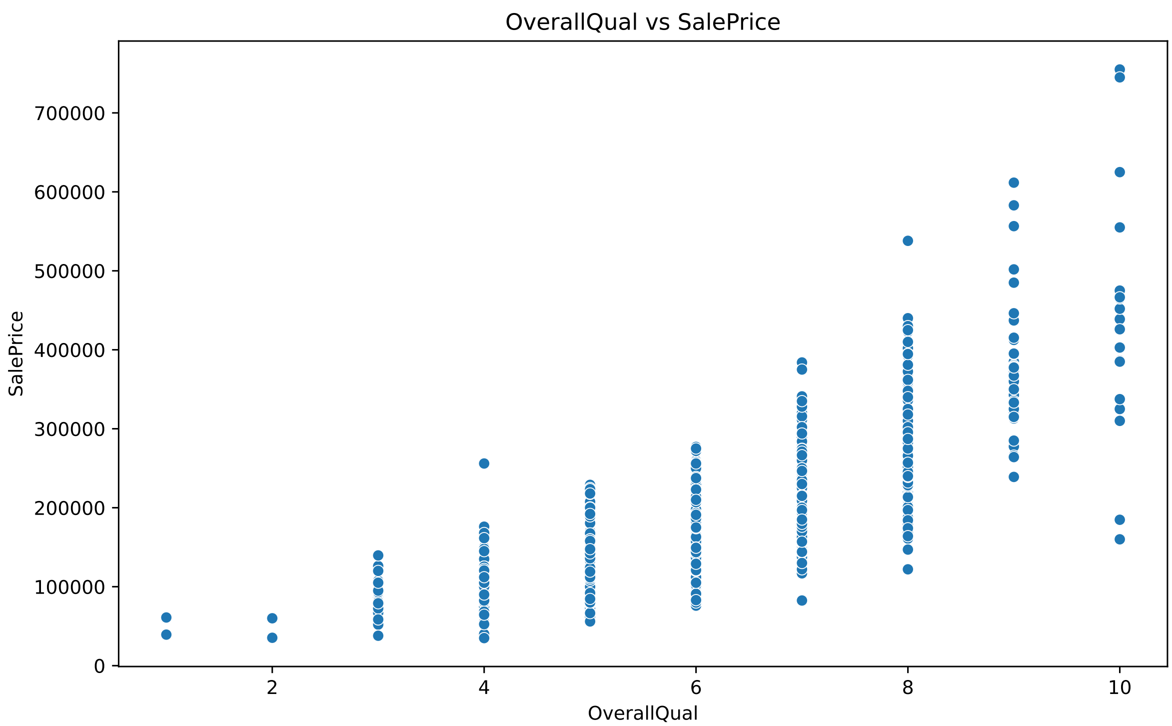

Distribution of Sale Prices
Count
SalePrice
0
25
50
75
100
125
150
175
100000
200000
300000
400000
500000
600000
700000


GrLivArea vs SalePrice
SalePrice
GrLivArea (Above ground living area)
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000
5000


SalePrice by Overall Quality
SalePrice
700000
600000
500000
400000
300000
200000
100000
0
1
2
3
4
5
6
7
8
9
10
OverallQual


Correlation Matrix of Numerical Features
Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4


Top 10 Features Correlated with SalePrice
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
TotRmsAbvGrd
YearBuilt
YearRemodAdd
None


OverallQual vs SalePrice
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
2
4
6
8
10
OverallQual

Data Preprocessing

```
def preprocess_data(df):
    # Handle missing values
    for col in df.columns:
        if df[col].dtype != "object":
            df[col] = df[col].fillna(df[col].median())
        else:
            df[col] = df[col].fillna(df[col].mode()[0])

    # Encode categorical variables
    le = LabelEncoder()
    for col in df.select_dtypes(include=["object"]).columns:
        df[col] = le.fit_transform(df[col].astype(str))

    return df


# Preprocess train and test data
X = preprocess_data(train.drop("SalePrice", axis=1))
y = np.log1p(
    train["SalePrice"]
)  # Log transform the target variable to handle skewed distribution
test_processed = preprocess_data(test)

scaler = StandardScaler()  # To standardize the features
X_scaled = scaler.fit_transform(X)
test_processed_scaled = scaler.transform(test_processed)

print("Processed data shape:", X.shape)
```

```
Processed data shape: (1460, 80)
```

Model Training and Evaluation

```
def train_and_evaluate(model, X, y, test_data, model_name):
    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
```

```
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    val_pred = model.predict(X_val)

    train_mse = mean_squared_error(y_train, train_pred)
    train_rmse = np.sqrt(train_mse)
    train_mae = mean_absolute_error(y_train, train_pred)
    train_r2 = r2_score(y_train, train_pred)

    val_mse = mean_squared_error(y_val, val_pred)
    val_rmse = np.sqrt(val_mse)
    val_mae = mean_absolute_error(y_val, val_pred)
    val_r2 = r2_score(y_val, val_pred)

    print(f"{model_name} Results:")
    print(f"Train RMSE: {train_rmse:.4f}")
    print(f"Train MAE: {train_mae:.4f}")
    print(f"Train R2 Score: {train_r2:.4f}")
    print(f"Validation RMSE: {val_rmse:.4f}")
    print(f"Validation MAE: {val_mae:.4f}")
    print(f"Validation R2 Score: {val_r2:.4f}")
    print("\n")

    return model, (y_train, train_pred, y_val, val_pred)


# Linear models
linear_models = {
    "Linear Regression": LinearRegression(),
    "Ridge": Ridge(),  # also known as L2 regularization
    "Lasso": Lasso(),  # also known as L1 regularization
    "ElasticNet": ElasticNet(),
}

linear_results = {}
for name, model in linear_models.items():
    linear_results[name] = train_and_evaluate(
        model, X_scaled, y, test_processed_scaled, name
    )
# Lasso looks weird
# ElasticNet looks weird
```

```python
# Advanced models
rf_model = RandomForestRegressor(random_state=42)
rf_trained, rf_results = train_and_evaluate(
    rf_model, X, y, test_processed, "Random Forest"
)

lgb_model = lgb.LGBMRegressor(random_state=42)
lgb_trained, lgb_results = train_and_evaluate(
    lgb_model, X, y, test_processed, "LightGBM"
)

xgb_model = xgb.XGBRegressor(random_state=42)
xgb_trained, xgb_results = train_and_evaluate(
    xgb_model, X, y, test_processed, "XGBoost"
)

cbt_model = cbt.CatBoostRegressor(random_state=42, verbose=False)
cbt_trained, cbt_results = train_and_evaluate(
    cbt_model, X, y, test_processed, "CatBoost"
)
```

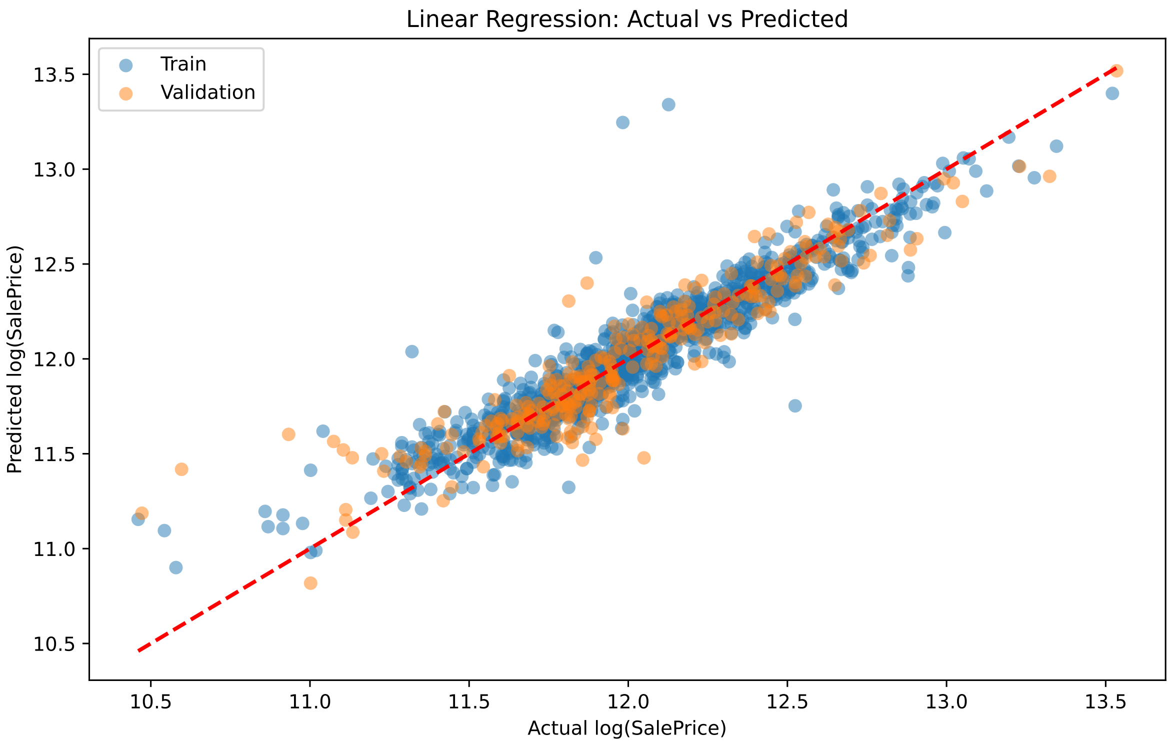
```
Linear Regression Results:
Train RMSE: 0.1306
Train MAE: 0.0894
Train R2 Score: 0.8882
Validation RMSE: 0.1553
Validation MAE: 0.1061
Validation R2 Score: 0.8708


Ridge Results:
Train RMSE: 0.1306
Train MAE: 0.0894
Train R2 Score: 0.8881
Validation RMSE: 0.1553
Validation MAE: 0.1061
Validation R2 Score: 0.8708


Lasso Results:
Train RMSE: 0.3904
Train MAE: 0.3034
```

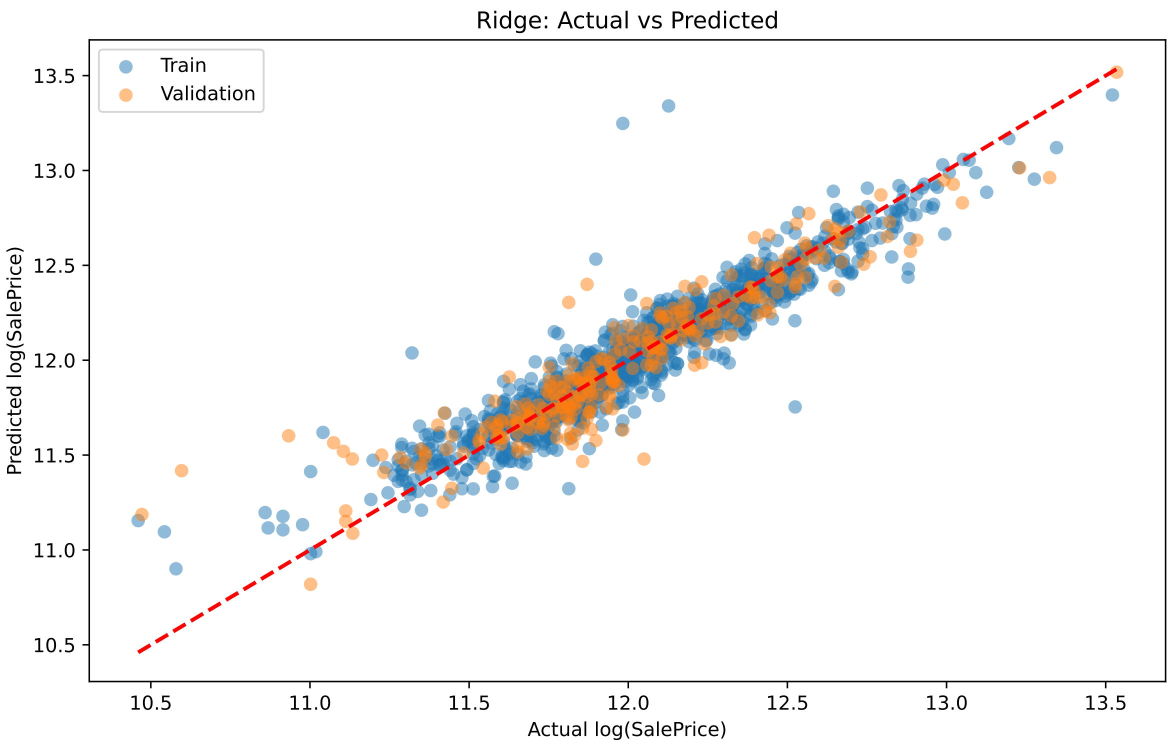
```
Train R2 Score: 0.0000
Validation RMSE: 0.4332
Validation MAE: 0.3371
Validation R2 Score: -0.0058


ElasticNet Results:
Train RMSE: 0.3904
Train MAE: 0.3034
Train R2 Score: 0.0000
Validation RMSE: 0.4332
Validation MAE: 0.3371
Validation R2 Score: -0.0058


Random Forest Results:
Train RMSE: 0.0535
Train MAE: 0.0361
Train R2 Score: 0.9812
Validation RMSE: 0.1458
Validation MAE: 0.0986
Validation R2 Score: 0.8861


[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.0006
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3364
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 73
[LightGBM] [Info] Start training from score 12.030658
LightGBM Results:
Train RMSE: 0.0420
Train MAE: 0.0256
Train R2 Score: 0.9884
Validation RMSE: 0.1413
Validation MAE: 0.0934
Validation R2 Score: 0.8930


XGBoost Results:
Train RMSE: 0.0050
Train MAE: 0.0035
Train R2 Score: 0.9998
Validation RMSE: 0.1514
```

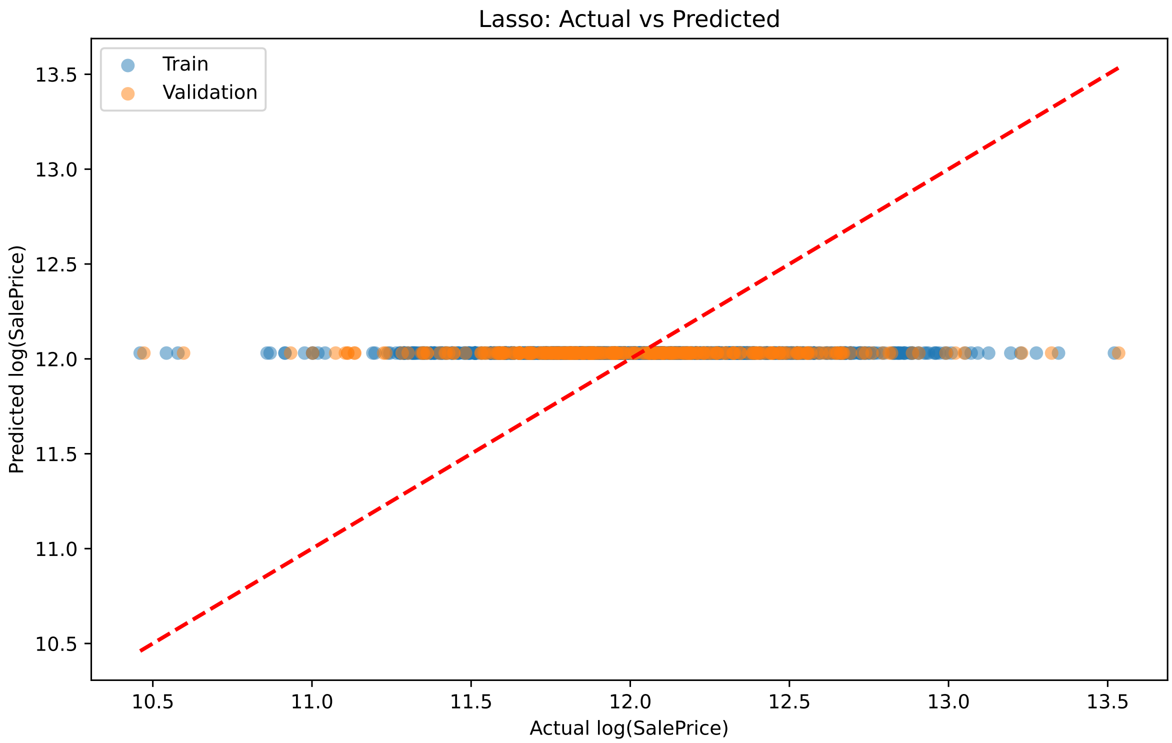
```
Validation MAE: 0.1022
Validation R2 Score: 0.8772


CatBoost Results:
Train RMSE: 0.0322
Train MAE: 0.0246
Train R2 Score: 0.9932
Validation RMSE: 0.1329
Validation MAE: 0.0866
Validation R2 Score: 0.9053
```

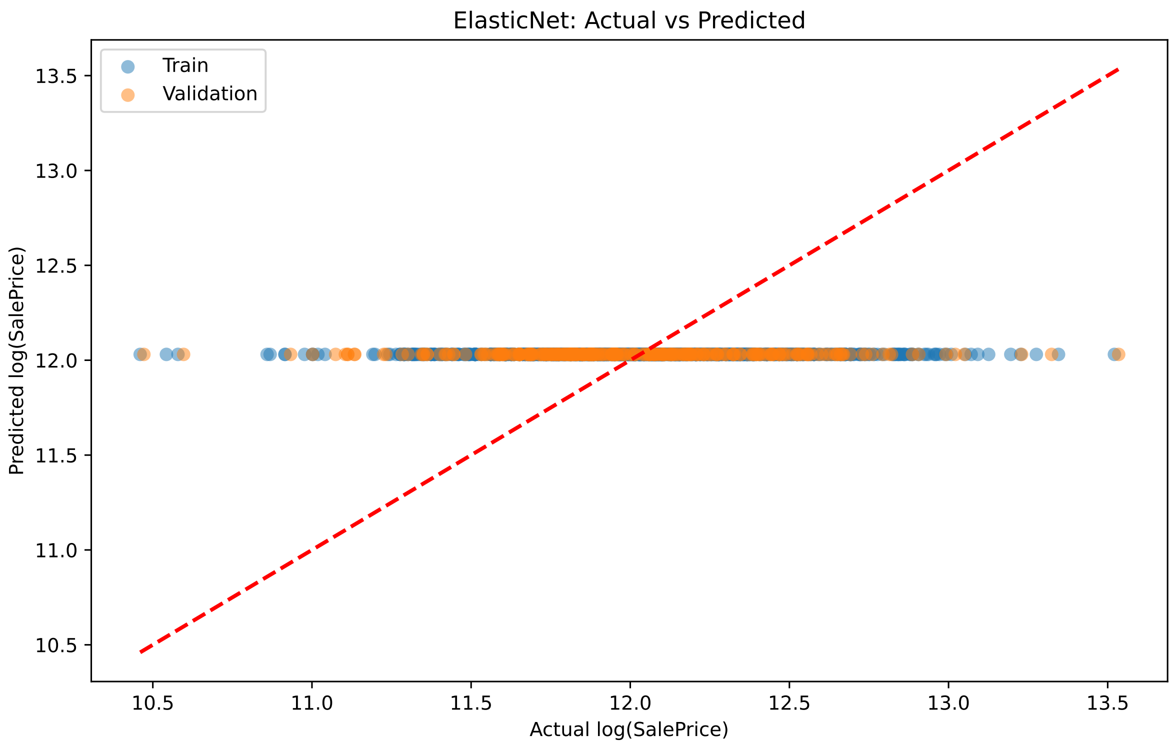
Model Performance Visualization

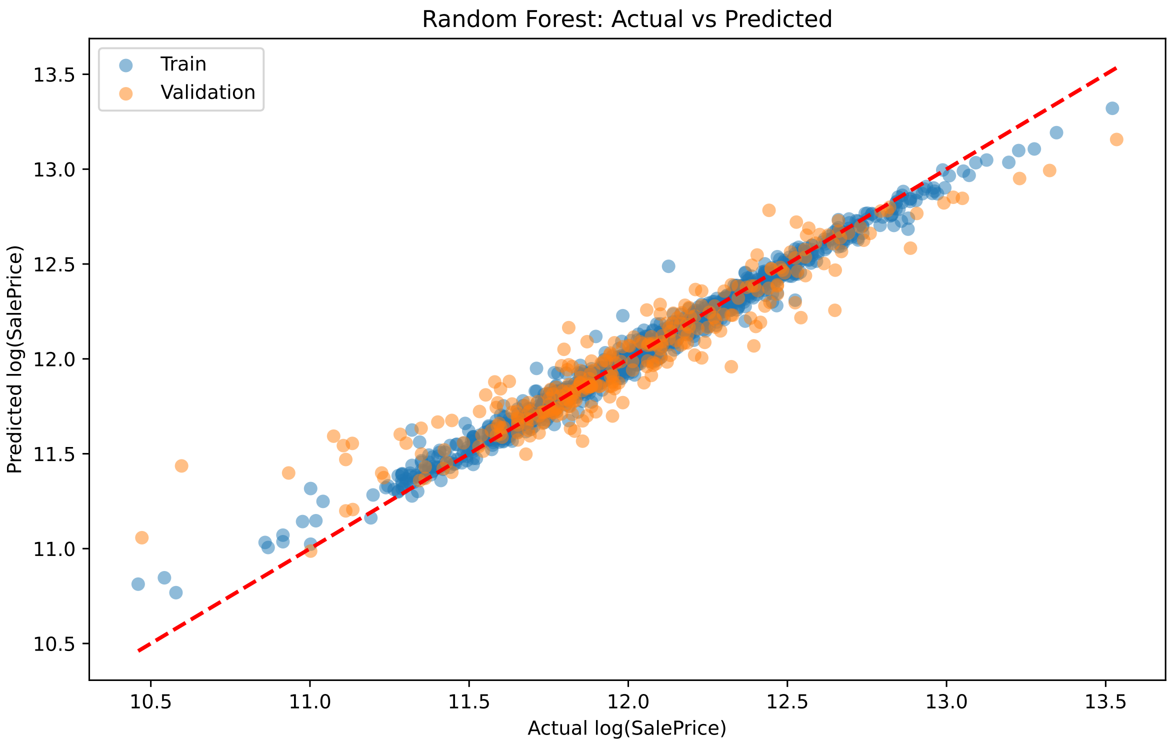
```
def plot_actual_vs_predicted(results, model_name):
    y_train, train_pred, y_val, val_pred = results
    plt.figure(figsize=(10, 6))
    plt.scatter(y_train, train_pred, alpha=0.5, label="Train")
    plt.scatter(y_val, val_pred, alpha=0.5, label="Validation")
    plt.plot([y.min(), y.max()], [y.min(), y.max()], "r--", lw=2)
    plt.xlabel("Actual log(SalePrice)")
    plt.ylabel("Predicted log(SalePrice)")
    plt.title(f"{model_name}: Actual vs Predicted")
    plt.legend()
    plt.show()


# Plot for each model
for name, (model, results) in linear_results.items():
    plot_actual_vs_predicted(results, name)

plot_actual_vs_predicted(rf_results, "Random Forest")
plot_actual_vs_predicted(lgb_results, "LightGBM")
plot_actual_vs_predicted(xgb_results, "XGBoost")
plot_actual_vs_predicted(cbt_results, "CatBoost")
```

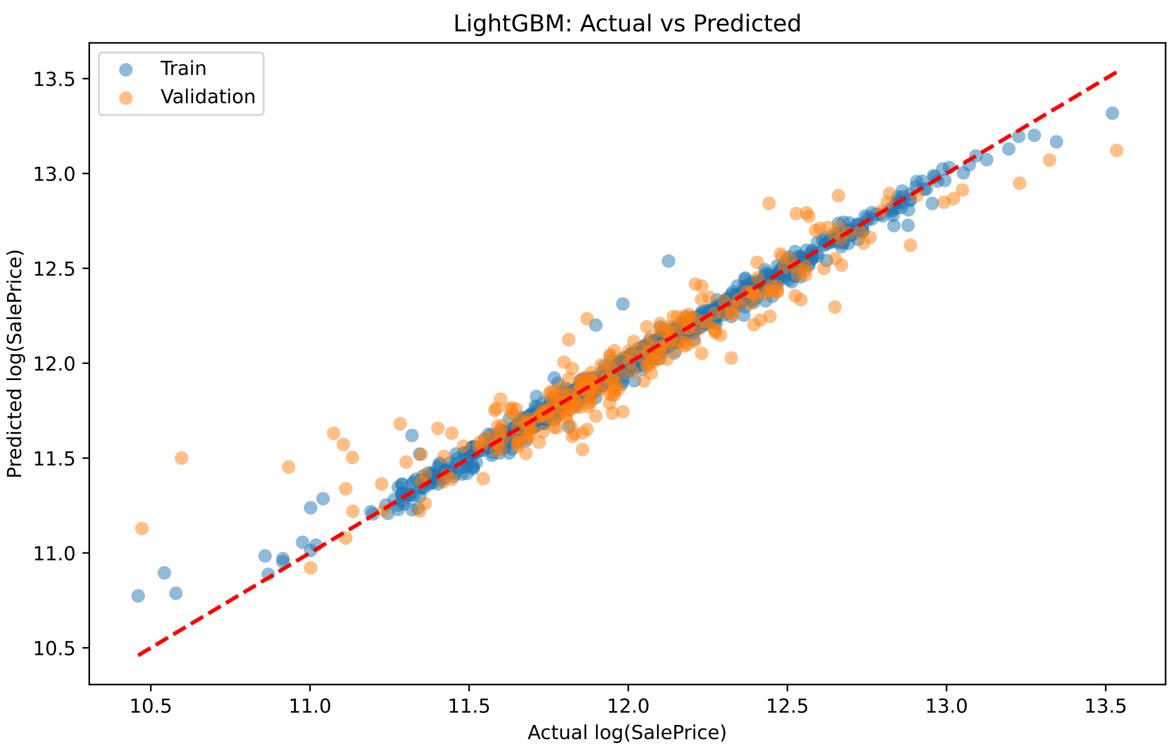

Linear Regression: Actual vs Predicted
Train
Validation
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Predicted log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5
Actual log(SalePrice)

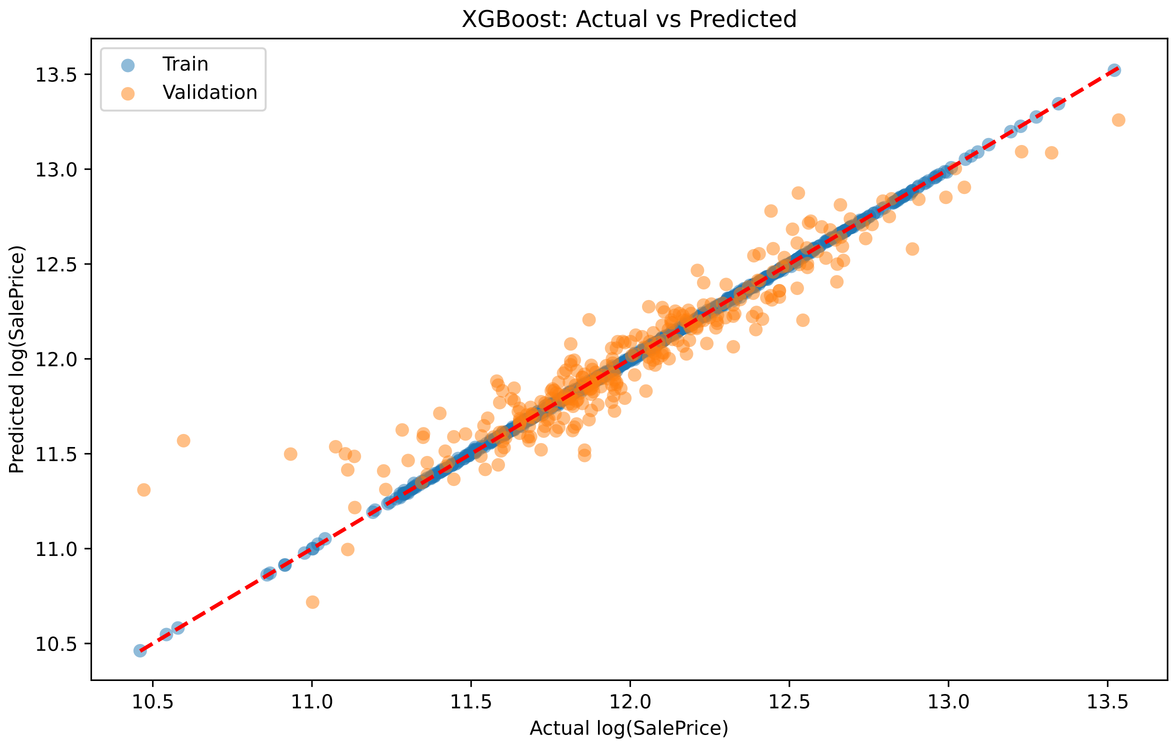

Ridge: Actual vs Predicted
Train
Validation
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Predicted log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5
Actual log(SalePrice)

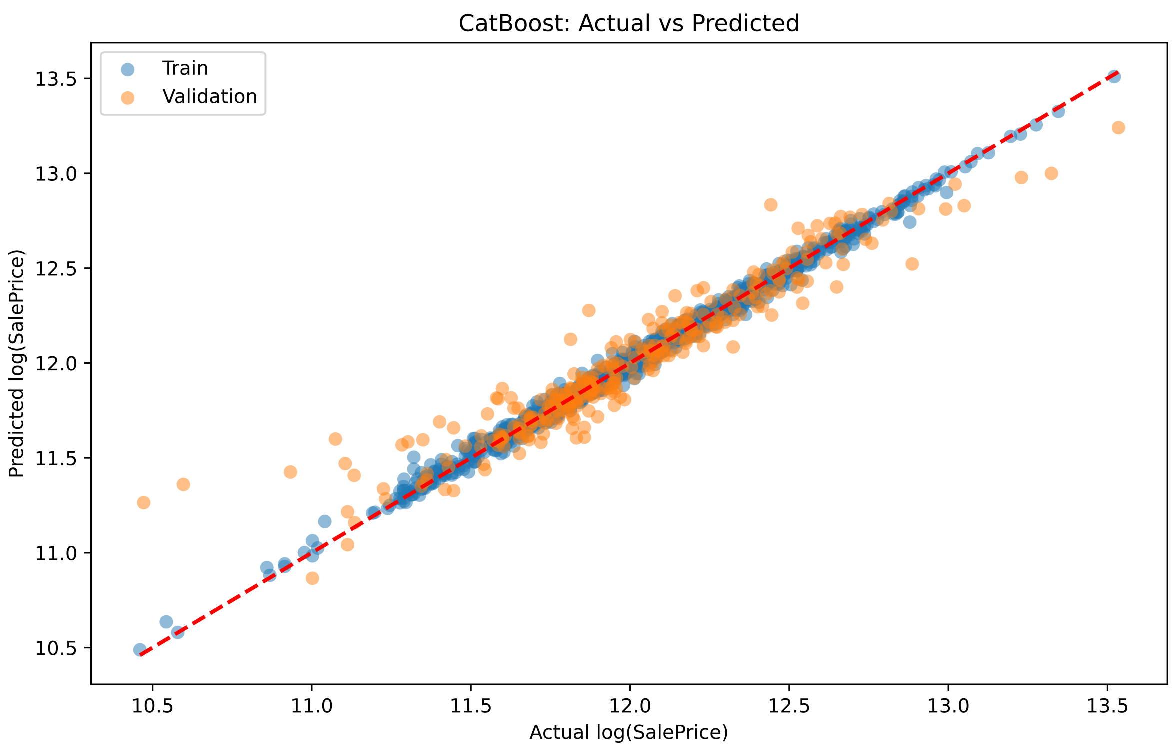

Lasso: Actual vs Predicted
Train
Validation
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Predicted log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5
Actual log(SalePrice)


ElasticNet: Actual vs Predicted
Train
Validation
13.5
13.0
12.5
12.0
11.5
11.0
10.5
Predicted log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5
Actual log(SalePrice)


Random Forest: Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5

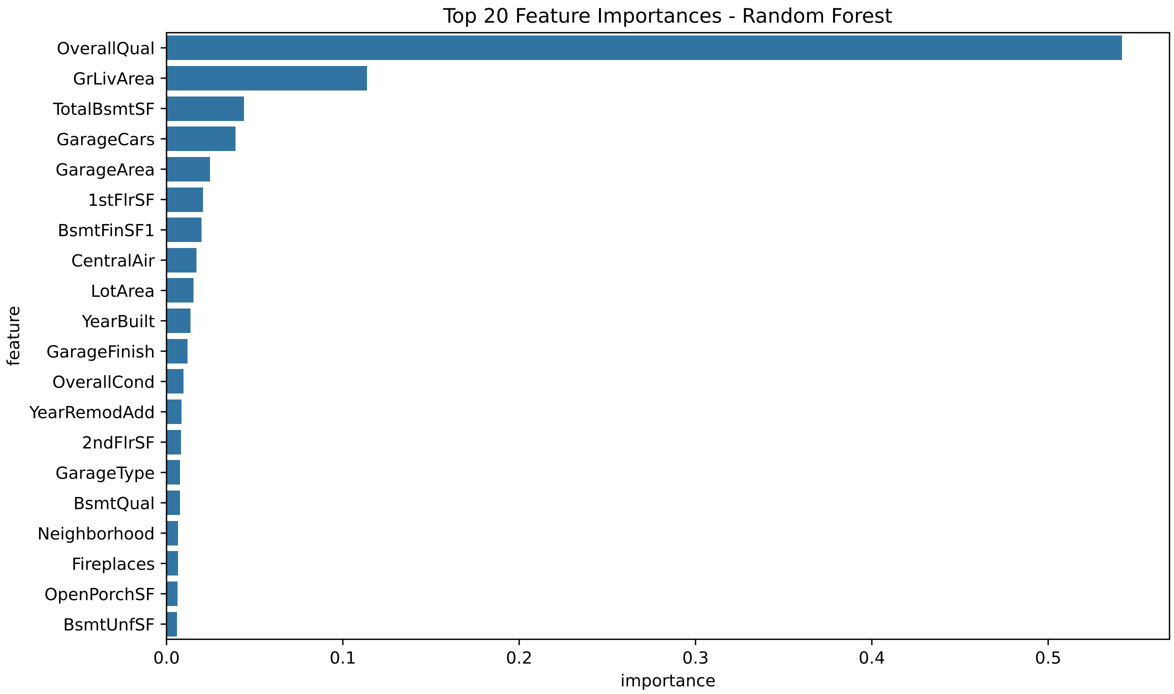

LightGBM: Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5


XGBoost: Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5

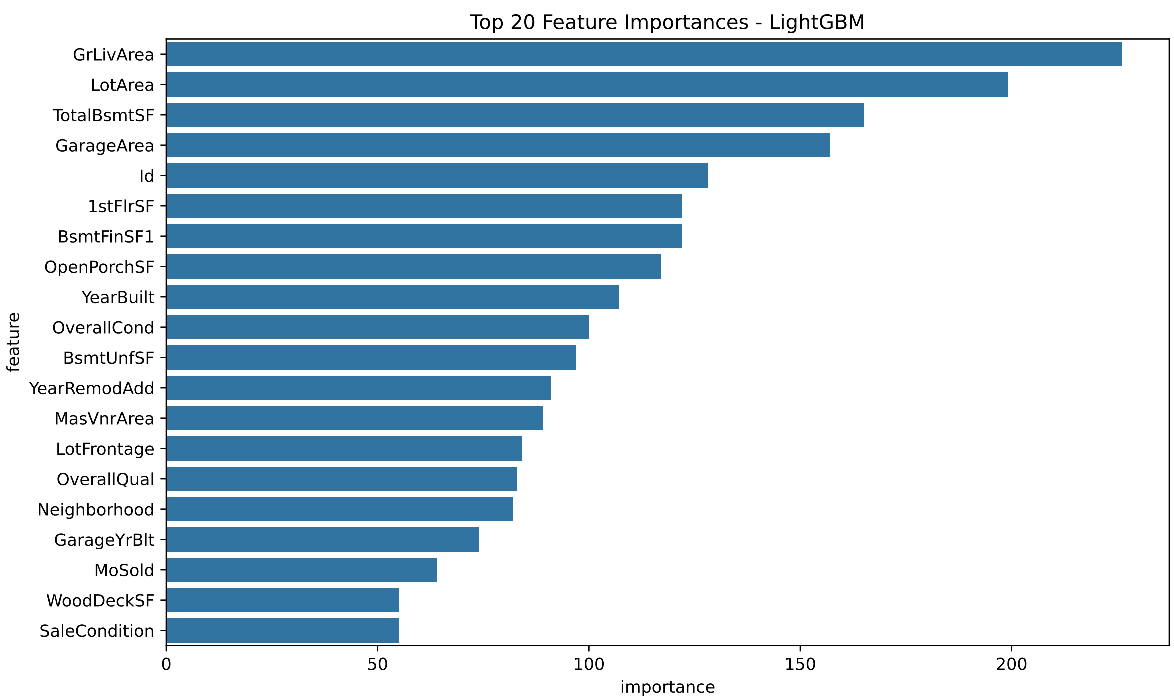

CatBoost: Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5

Feature Importance

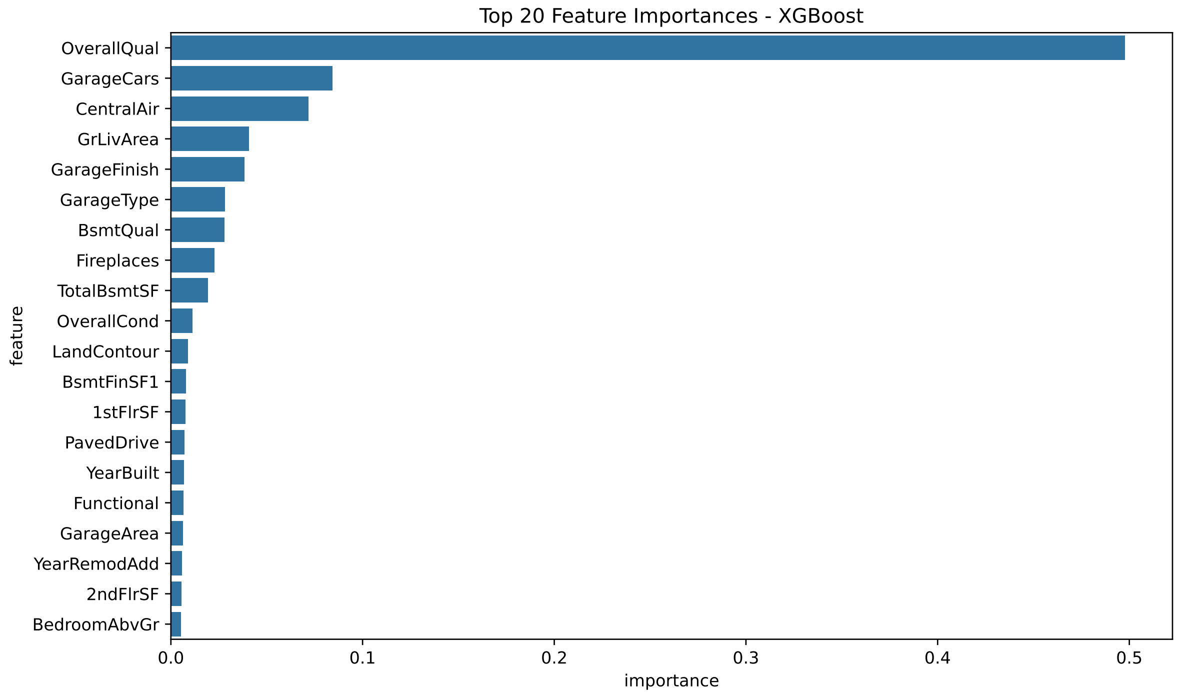
```
def plot_feature_importance(model, X, model_name):
    if hasattr(model, "feature_importances_"):
        importances = model.feature_importances_
    elif hasattr(model, "feature_importance"):
        importances = model.feature_importance()
    else:
        print(f"Feature importance not available for {model_name}")
        return

    feature_imp = pd.DataFrame({"feature": X.columns, "importance": importances})
    feature_imp = feature_imp.sort_values("importance", ascending=False).head(20)

    plt.figure(figsize=(10, 6))
    sns.barplot(x="importance", y="feature", data=feature_imp)
    plt.title(f"Top 20 Feature Importances - {model_name}")
    plt.tight_layout()
    plt.show()


plot_feature_importance(rf_trained, X, "Random Forest")
plot_feature_importance(lgb_trained, X, "LightGBM")
plot_feature_importance(xgb_trained, X, "XGBoost")
plot_feature_importance(cbt_trained, X, "CatBoost")
```



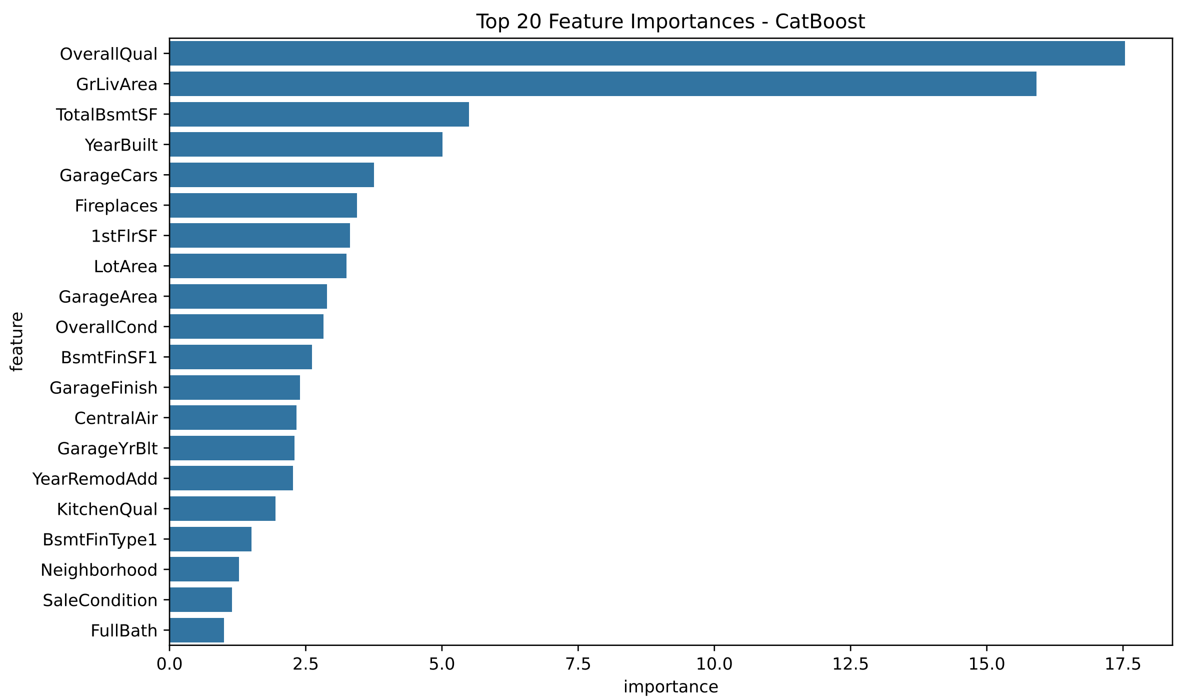
Top 20 Feature Importances - Random Forest
OverallQual
GrLivArea
TotalBsmtSF
GarageCars
GarageArea
1stFlrSF
BsmtFinSF1
CentralAir
LotArea
YearBuilt
GarageFinish
OverallCond
YearRemodAdd
2ndFlrSF
GarageType
BsmtQual
Neighborhood
Fireplaces
OpenPorchSF
BsmtUnfSF
feature
0.0
0.1
0.2
0.3
0.4
0.5
importance



Top 20 Feature Importances - LightGBM
GrLivArea
LotArea
TotalBsmtSF
GarageArea
Id
1stFlrSF
BsmtFinSF1
OpenPorchSF
YearBuilt
OverallCond
BsmtUnfSF
YearRemodAdd
MasVnrArea
LotFrontage
OverallQual
Neighborhood
GarageYrBlt
MoSold
WoodDeckSF
SaleCondition
feature
0
50
100
150
200
importance



Top 20 Feature Importances - XGBoost
OverallQual
GarageCars
CentralAir
GrLivArea
GarageFinish
GarageType
BsmtQual
Fireplaces
TotalBsmtSF
OverallCond
LandContour
BsmtFinSF1
1stFlrSF
PavedDrive
YearBuilt
Functional
GarageArea
YearRemodAdd
2ndFlrSF
BedroomAbvGr
feature
0.0
0.1
0.2
0.3
0.4
0.5
importance



Top 20 Feature Importances - CatBoost
OverallQual
GrLivArea
TotalBsmtSF
YearBuilt
GarageCars
Fireplaces
1stFlrSF
LotArea
GarageArea
OverallCond
BsmtFinSF1
GarageFinish
CentralAir
GarageYrBlt
YearRemodAdd
KitchenQual
BsmtFinType1
Neighborhood
SaleCondition
FullBath
feature
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
importance

Hyperparameter Tuning

```
def tune_hyperparameters(model, param_grid, X, y, model_name):
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=5,
        scoring="neg_mean_squared_error",
        verbose=1,
        n_jobs=-1,
    )
    grid_search.fit(X, y)

    print(f"Best parameters for {model_name}:")
    print(grid_search.best_params_)
    print(f"Best RMSE: {np.sqrt(-grid_search.best_score_):.4f}")
    print("\n")

    return grid_search.best_estimator_


# Random Forest hyperparameter tuning
rf_param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10],
    "min_samples_split": [2, 5],
}
rf_tuned = tune_hyperparameters(
    RandomForestRegressor(random_state=42), rf_param_grid, X, y, "Random Forest"
)

# LightGBM hyperparameter tuning
lgb_param_grid = {
    "num_leaves": [31, 127],
    "learning_rate": [0.01, 0.1],
    "n_estimators": [100, 200],
    "verbosity": [-1],
}
lgb_tuned = tune_hyperparameters(
    lgb.LGBMRegressor(random_state=42), lgb_param_grid, X, y, "LightGBM"
)
```

```python
# XGBoost hyperparameter tuning
xgb_param_grid = {
    "max_depth": [3, 6],
    "learning_rate": [0.01, 0.1],
    "n_estimators": [100, 200],
}
xgb_tuned = tune_hyperparameters(
    xgb.XGBRegressor(random_state=42), xgb_param_grid, X, y, "XGBoost"
)

# CatBoost hyperparameter tuning
cbt_param_grid = {
    "depth": [6, 8],
    "learning_rate": [0.01, 0.1],
    "iterations": [100, 200],
}
cbt_tuned = tune_hyperparameters(
    cbt.CatBoostRegressor(random_state=42, verbose=False),
    cbt_param_grid,
    X,
    y,
    "CatBoost",
)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for Random Forest:
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
Best RMSE: 0.1423


Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for LightGBM:
{'learning_rate': 0.1, 'n_estimators': 100, 'num_leaves': 127, 'verbosity': -1}
Best RMSE: 0.1338


Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for XGBoost:
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
Best RMSE: 0.1271
```

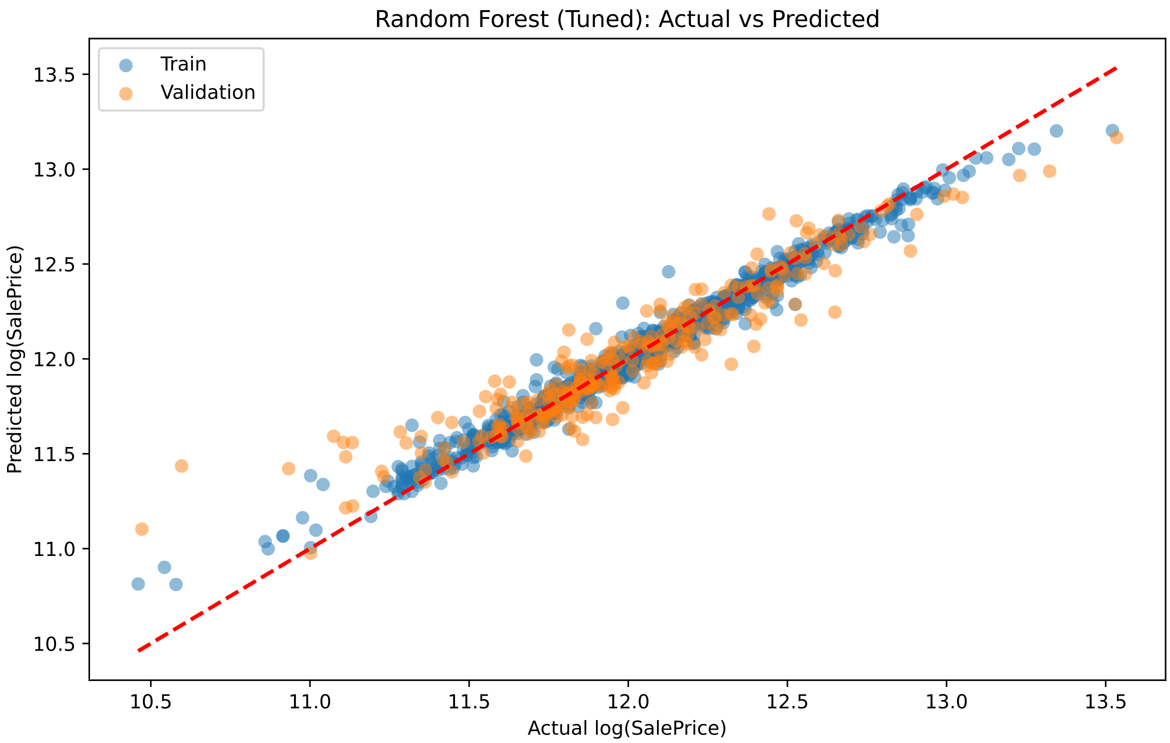
```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best parameters for CatBoost:
{'depth': 6, 'iterations': 200, 'learning_rate': 0.1}
Best RMSE: 0.1248
```

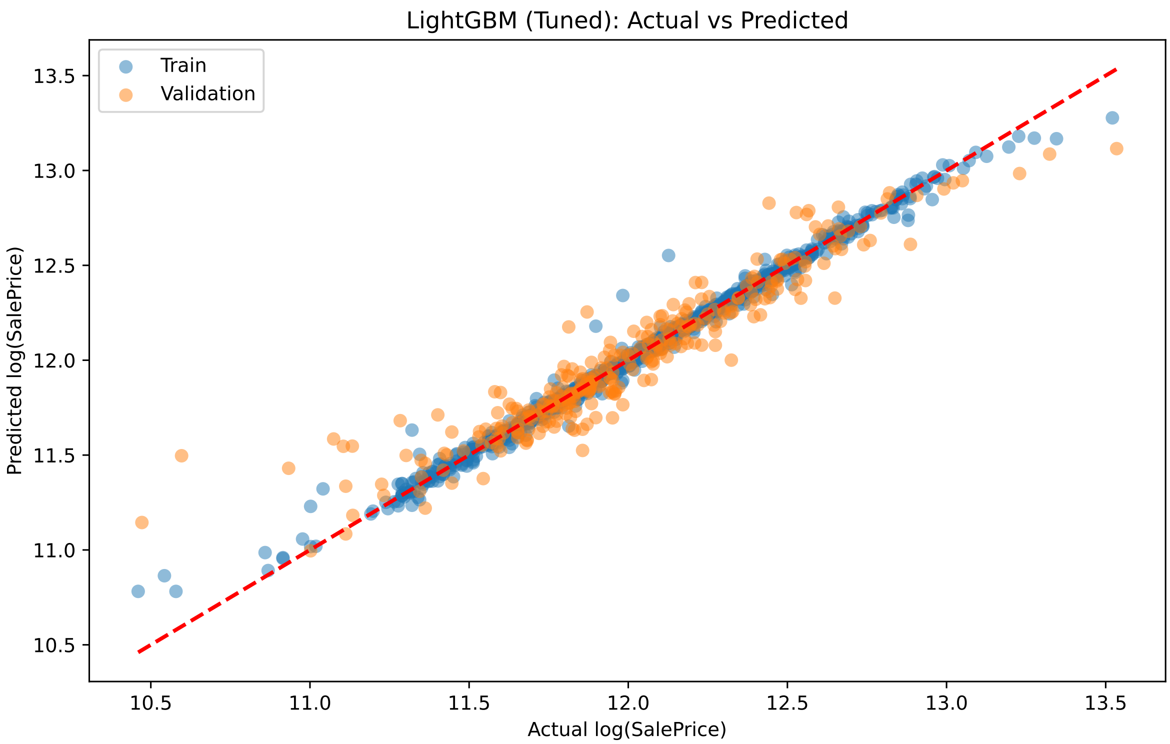
Final Model Evaluation

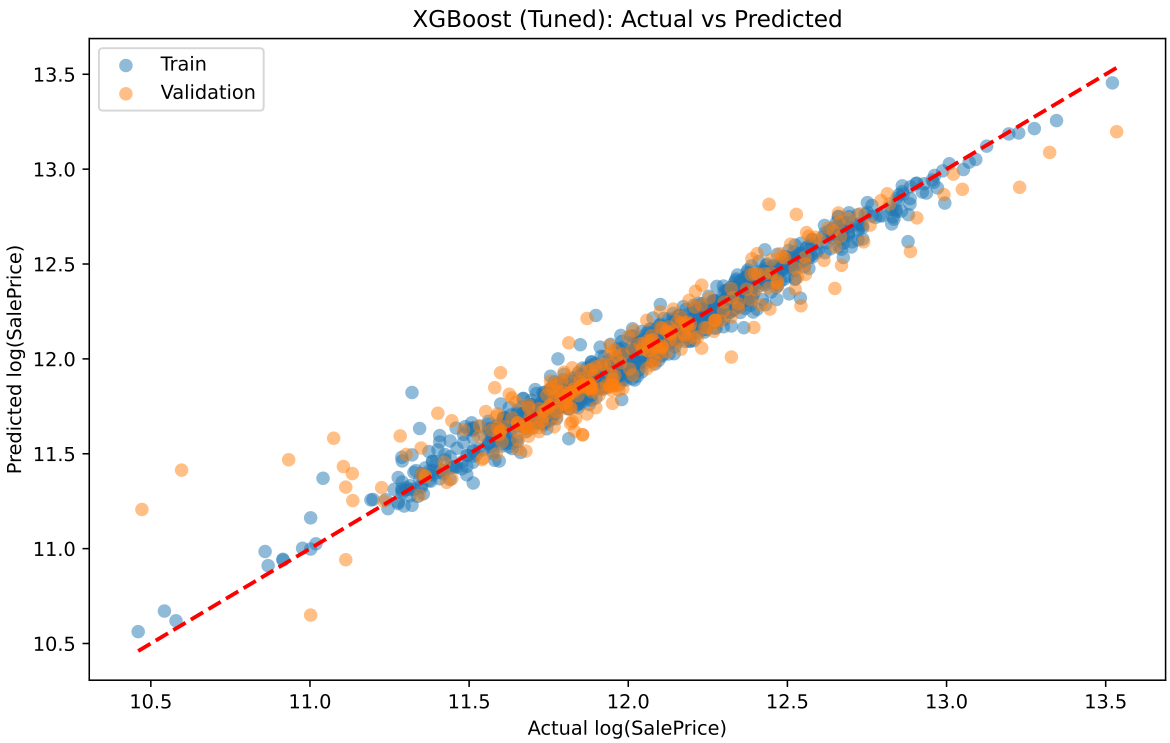
```python
print("Final Model Evaluation:")
rf_final, rf_final_results = train_and_evaluate(
    rf_tuned, X, y, test_processed, "Random Forest (Tuned)"
)
lgb_final, lgb_final_results = train_and_evaluate(
    lgb_tuned, X, y, test_processed, "LightGBM (Tuned)"
)
xgb_final, xgb_final_results = train_and_evaluate(
    xgb_tuned, X, y, test_processed, "XGBoost (Tuned)"
)
cbt_final, cbt_final_results = train_and_evaluate(
    cbt_tuned, X, y, test_processed, "CatBoost (Tuned)"
)

# Plot final model performances
plot_actual_vs_predicted(rf_final_results, "Random Forest (Tuned)")
plot_actual_vs_predicted(lgb_final_results, "LightGBM (Tuned)")
plot_actual_vs_predicted(xgb_final_results, "XGBoost (Tuned)")
plot_actual_vs_predicted(cbt_final_results, "CatBoost (Tuned)")
```

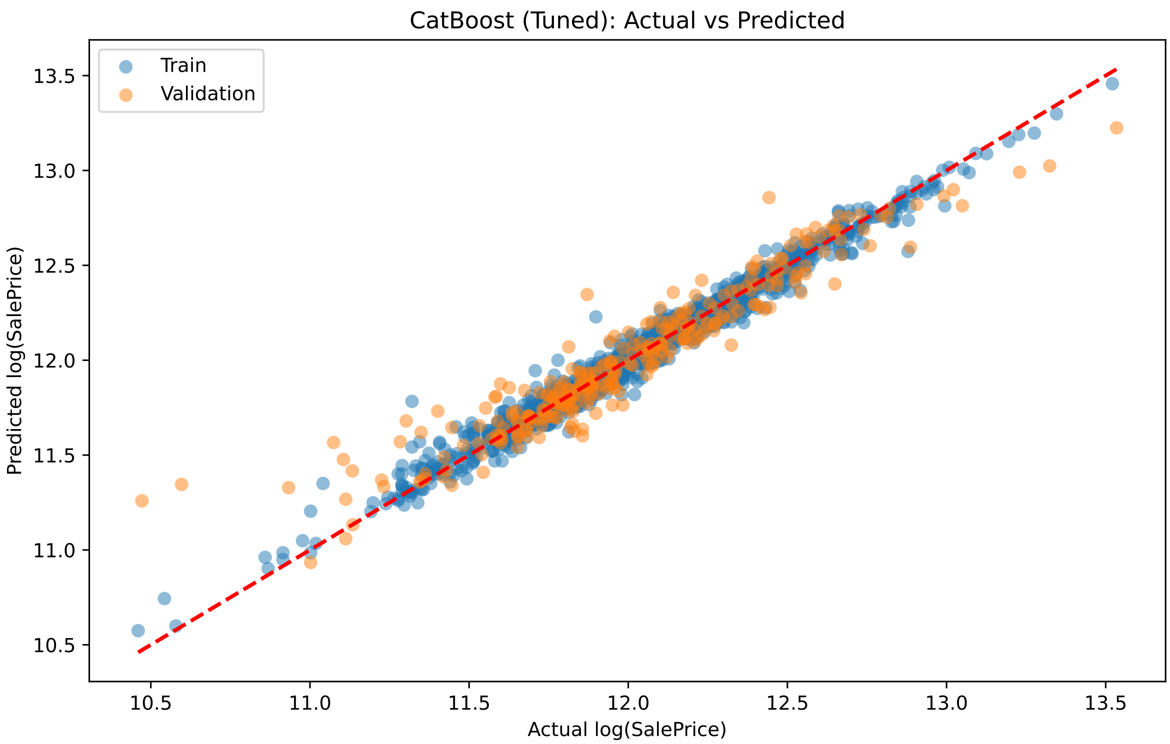
```
Final Model Evaluation:
Random Forest (Tuned) Results:
Train RMSE: 0.0597
Train MAE: 0.0395
Train R2 Score: 0.9766
Validation RMSE: 0.1463
Validation MAE: 0.0984
Validation R2 Score: 0.8853


LightGBM (Tuned) Results:
Train RMSE: 0.0381
```

```
Train MAE: 0.0192
Train R2 Score: 0.9905
Validation RMSE: 0.1410
Validation MAE: 0.0921
Validation R2 Score: 0.8935


XGBoost (Tuned) Results:
Train RMSE: 0.0622
Train MAE: 0.0456
Train R2 Score: 0.9746
Validation RMSE: 0.1372
Validation MAE: 0.0919
Validation R2 Score: 0.8991


CatBoost (Tuned) Results:
Train RMSE: 0.0606
Train MAE: 0.0450
Train R2 Score: 0.9759
Validation RMSE: 0.1334
Validation MAE: 0.0871
Validation R2 Score: 0.9047
```


Random Forest (Tuned): Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5


LightGBM (Tuned): Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
10.5
11.0
11.5
12.0
12.5
13.0
13.5

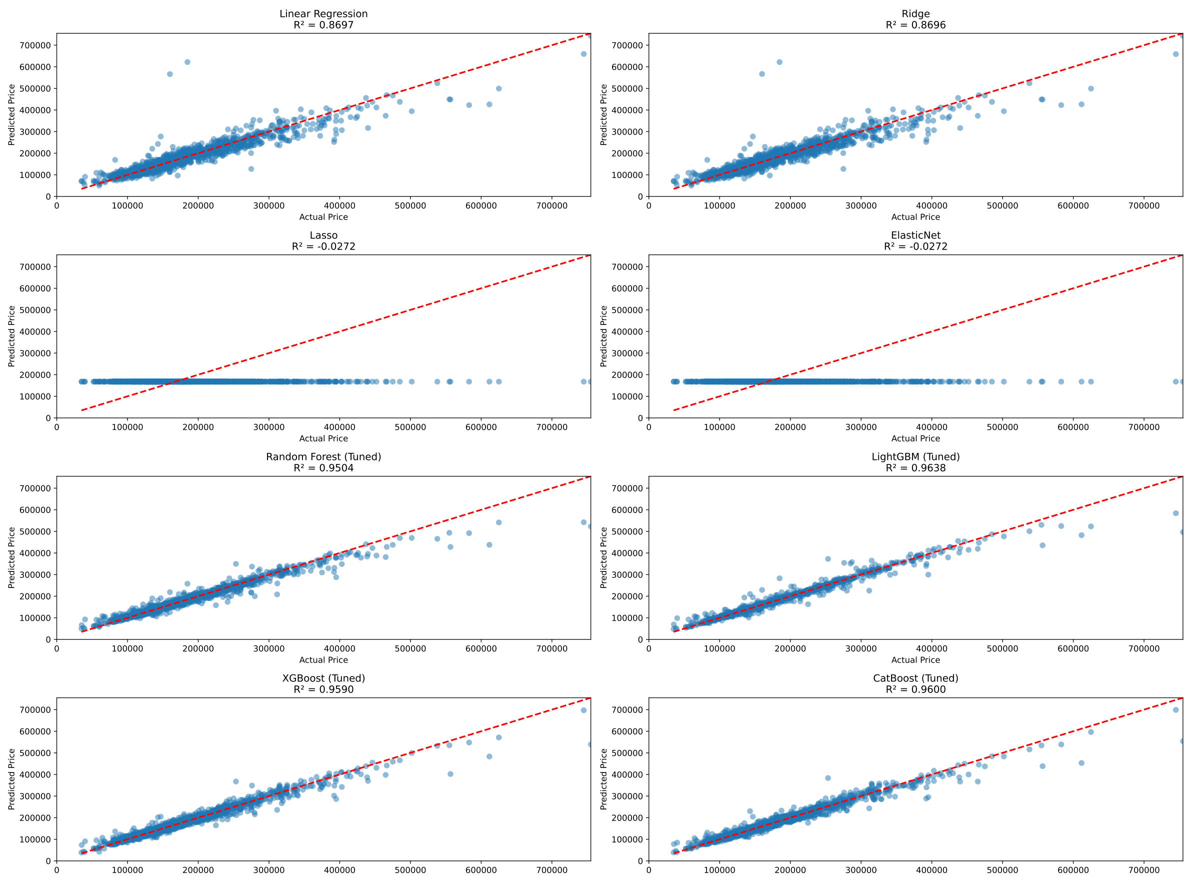

XGBoost (Tuned): Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
13.5
13.0
12.5
12.0
11.5
11.0
10.5
10.5
11.0
11.5
12.0
12.5
13.0
13.5


CatBoost (Tuned): Actual vs Predicted
Train
Validation
Predicted log(SalePrice)
Actual log(SalePrice)
13.5
13.0
12.5
12.0
11.5
11.0
10.5
10.5
11.0
11.5
12.0
12.5
13.0
13.5

Actual vs Predicted Price Comparison

```
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score


def plot_actual_vs_predicted_all_models(models_results, figsize=(20, 15)):
    n_models = len(models_results)
    fig, axes = plt.subplots(nrows=(n_models + 1) // 2, ncols=2, figsize=figsize)
    axes = axes.flatten()

    for i, (model_name, results) in enumerate(models_results.items()):
        y_true, y_pred = results

        # Convert from log scale back to original scale
        y_true_orig = np.expm1(y_true)
        y_pred_orig = np.expm1(y_pred)

        ax = axes[i]
        ax.scatter(y_true_orig, y_pred_orig, alpha=0.5)
        ax.plot(
            [y_true_orig.min(), y_true_orig.max()],
            [y_true_orig.min(), y_true_orig.max()],
            "r--",
            lw=2,
        )

        ax.set_xlabel("Actual Price")
        ax.set_ylabel("Predicted Price")
        ax.set_title(f"{model_name}\nR² = {r2_score(y_true_orig, y_pred_orig):.4f}")

        # Set axis limits
        ax.set_xlim(0, max(y_true_orig.max(), y_pred_orig.max()))
        ax.set_ylim(0, max(y_true_orig.max(), y_pred_orig.max()))

    # Remove any unused subplots
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

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

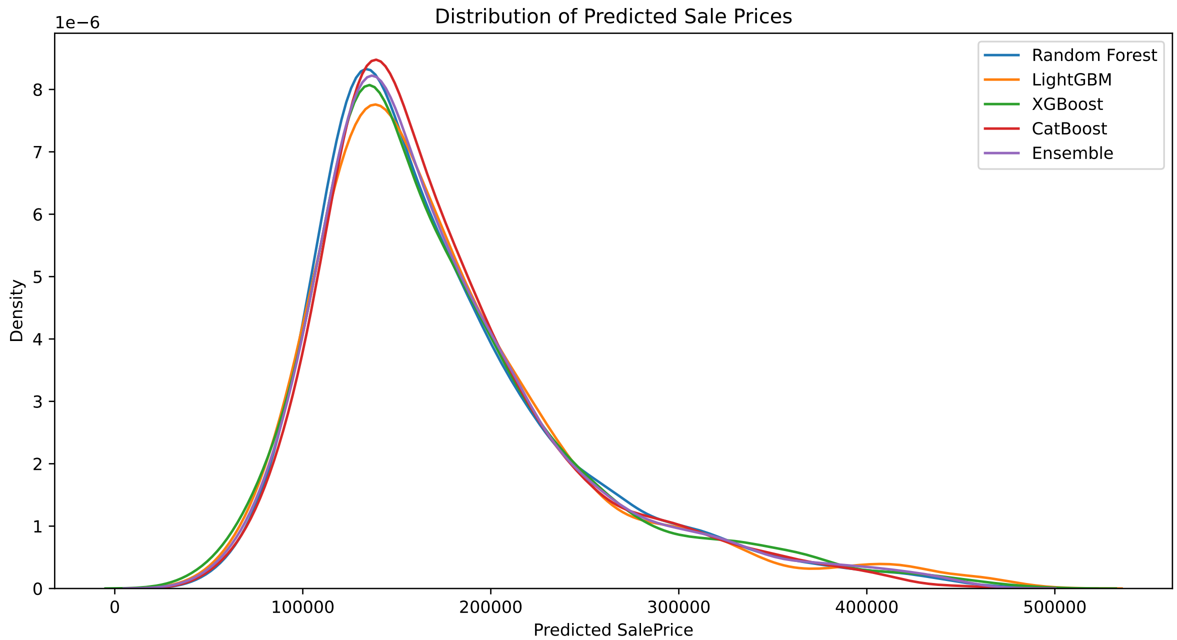
```python
# Prepare results for all models
all_models_results = {
    "Linear Regression": (y, linear_results["Linear Regression"][0].predict(X_scaled)),
    "Ridge": (y, linear_results["Ridge"][0].predict(X_scaled)),
    "Lasso": (y, linear_results["Lasso"][0].predict(X_scaled)),
    "ElasticNet": (y, linear_results["ElasticNet"][0].predict(X_scaled)),
    "Random Forest (Tuned)": (y, rf_final.predict(X)),
    "LightGBM (Tuned)": (y, lgb_final.predict(X)),
    "XGBoost (Tuned)": (y, xgb_final.predict(X)),
    "CatBoost (Tuned)": (y, cbt_final.predict(X)),
}

# Plot actual vs predicted for all models
plot_actual_vs_predicted_all_models(all_models_results)
```



Predictions on Test Data

```
def make_predictions(model, test_data):
    predictions = model.predict(test_data)
    return np.expm1(predictions)  # Inverse log transform


# Make individual model predictions
rf_predictions = make_predictions(rf_final, test_processed)
lgb_predictions = make_predictions(lgb_final, test_processed)
xgb_predictions = make_predictions(xgb_final, test_processed)
cbt_predictions = make_predictions(cbt_final, test_processed)

# Ensemble predictions (simple average)
ensemble_predictions = (
    rf_predictions + lgb_predictions + xgb_predictions + cbt_predictions
) / 4

# Plot distribution of predictions
plt.figure(figsize=(12, 6))
sns.kdeplot(rf_predictions, label="Random Forest")
sns.kdeplot(lgb_predictions, label="LightGBM")
sns.kdeplot(xgb_predictions, label="XGBoost")
sns.kdeplot(cbt_predictions, label="CatBoost")
sns.kdeplot(ensemble_predictions, label="Ensemble")
plt.xlabel("Predicted SalePrice")
plt.ylabel("Density")
plt.title("Distribution of Predicted Sale Prices")
plt.legend()
plt.show()

# Create submission file using the ensemble predictions
submission_ensemble = pd.DataFrame(
    {"Id": test["Id"], "SalePrice": ensemble_predictions}
)
submission_ensemble.to_csv("submission_ensemble.csv", index=False)

print("Submission file created successfully.")
submission_ensemble.head()

submission_rf = pd.DataFrame({"Id": test["Id"], "SalePrice": rf_predictions})
submission_rf.to_csv("submission_rf.csv", index=False)
```

```python
submission_rf.head()

submission_lgb = pd.DataFrame({"Id": test["Id"], "SalePrice": lgb_predictions})
submission_lgb.to_csv("submission_lgb.csv", index=False)
submission_lgb.head()

submission_xgb = pd.DataFrame({"Id": test["Id"], "SalePrice": xgb_predictions})
submission_xgb.to_csv("submission_xgb.csv", index=False)
submission_xgb.head()

submission_cbt = pd.DataFrame({"Id": test["Id"], "SalePrice": cbt_predictions})
submission_cbt.to_csv("submission_cbt.csv", index=False)
submission_cbt.head()
```




```
Submission file created successfully.
```

	Id	SalePrice
0	1461	123614.945579
1	1462	159661.736074
2	1463	178848.784997
3	1464	184858.933616

	Id	SalePrice
4	1465	189985.994257

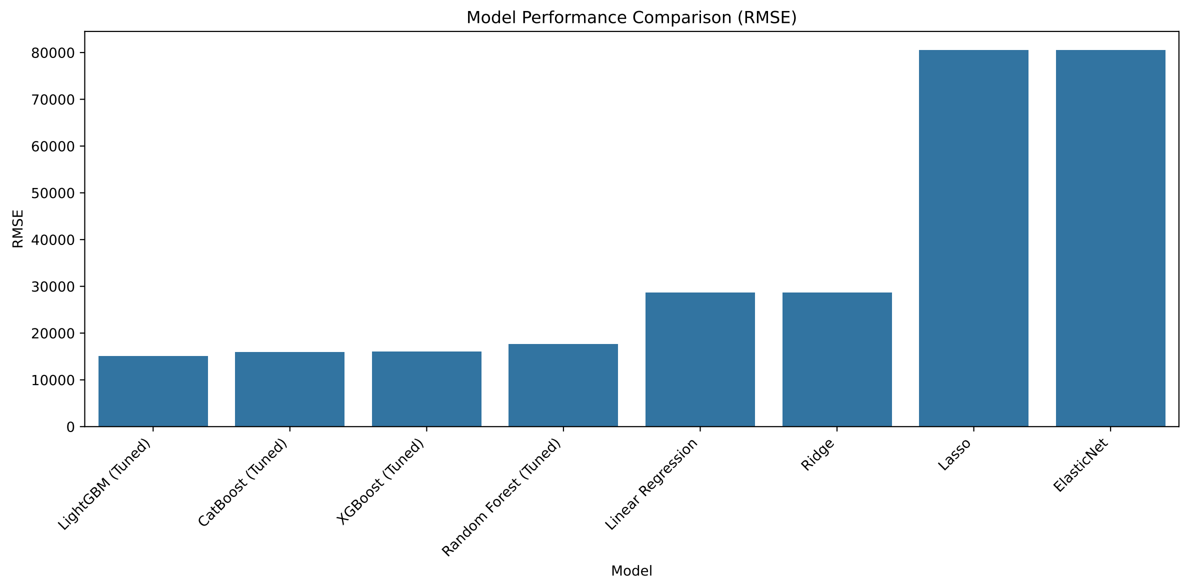
Model Comparison and Best Model Selection

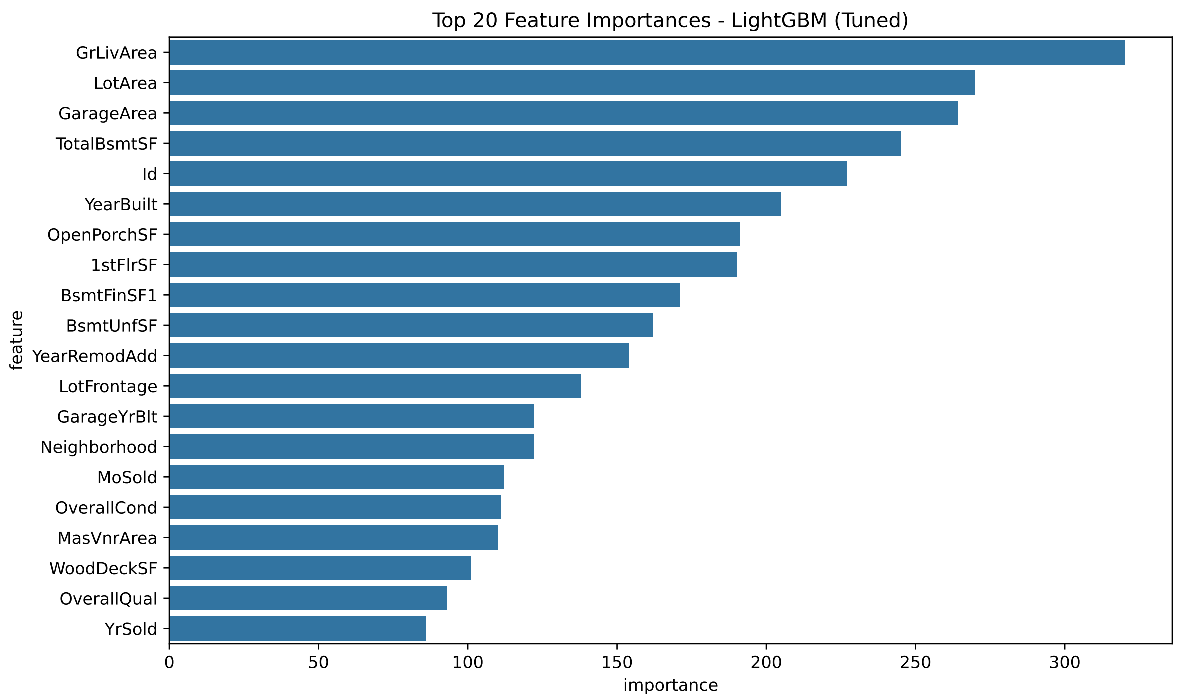
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


def evaluate_model(y_true, y_pred, model_name):
    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)
    return {"Model": model_name, "RMSE": rmse, "MAE": mae, "R2": r2}


# Prepare results for all models
model_results = []
for model_name, (y_true, y_pred) in all_models_results.items():
    model_results.append(evaluate_model(np.expm1(y_true), np.expm1(y_pred), model_name))

# Convert results to DataFrame for easy comparison
results_df = pd.DataFrame(model_results)
results_df = results_df.sort_values("RMSE")

print("Model Performance Comparison:")
print(results_df)

# Identify the best model based on RMSE
best_model = results_df.iloc[0]
print("\nBest Performing Model:")
print(f"Model: {best_model['Model']}")
print(f"RMSE: {best_model['RMSE']:.4f}")
print(f"MAE: {best_model['MAE']:.4f}")
print(f"R2 Score: {best_model['R2']:.4f}")

# Visualize model performance comparison
plt.figure(figsize=(12, 6))
sns.barplot(x="Model", y="RMSE", data=results_df)
plt.title("Model Performance Comparison (RMSE)")
```

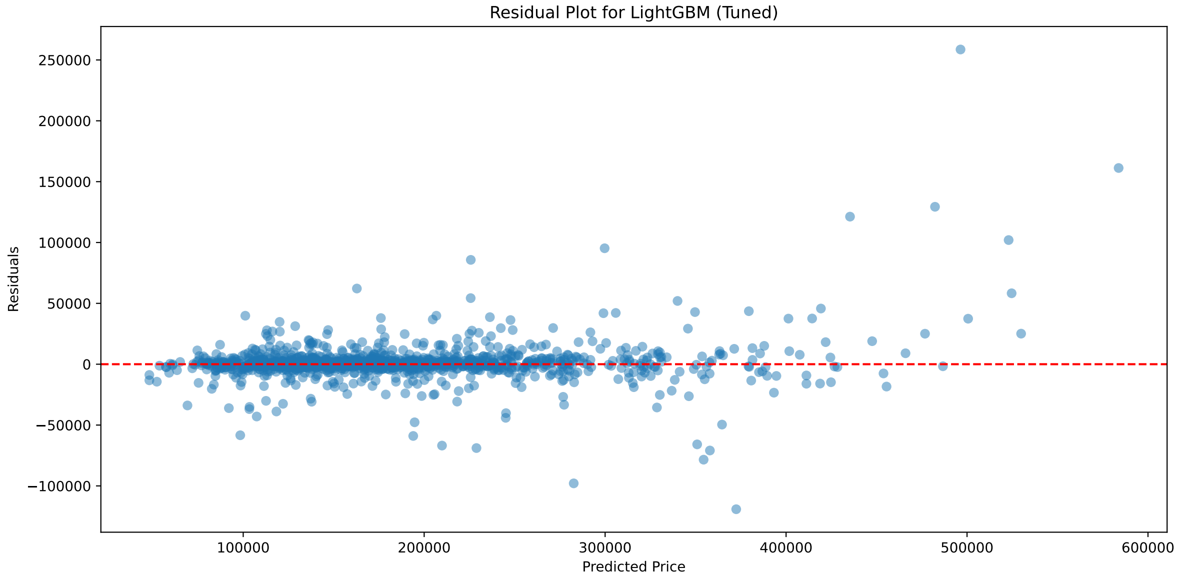
```
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Feature importance of the best model
best_model_name = best_model["Model"]
if best_model_name in [
    "Random Forest (Tuned)",
    "LightGBM (Tuned)",
    "XGBoost (Tuned)",
    "CatBoost (Tuned)",
]:
    model_name_map = {
        "Random Forest (Tuned)": rf_final,
        "LightGBM (Tuned)": lgb_final,
        "XGBoost (Tuned)": xgb_final,
        "CatBoost (Tuned)": cbt_final,
    }
    best_model_object = model_name_map[best_model_name]
    plot_feature_importance(best_model_object, X, best_model_name)
else:
    print(f"Feature importance not available for {best_model_name}")

# Residual analysis for the best model
y_true = np.expm1(y)
y_pred = np.expm1(all_models_results[best_model_name][1])
residuals = y_true - y_pred

plt.figure(figsize=(12, 6))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.xlabel("Predicted Price")
plt.ylabel("Residuals")
plt.title(f"Residual Plot for {best_model_name}")
plt.axhline(y=0, color="r", linestyle="--")
plt.tight_layout()
plt.show()

# Distribution of residuals
plt.figure(figsize=(12, 6))
sns.histplot(residuals, kde=True)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
```

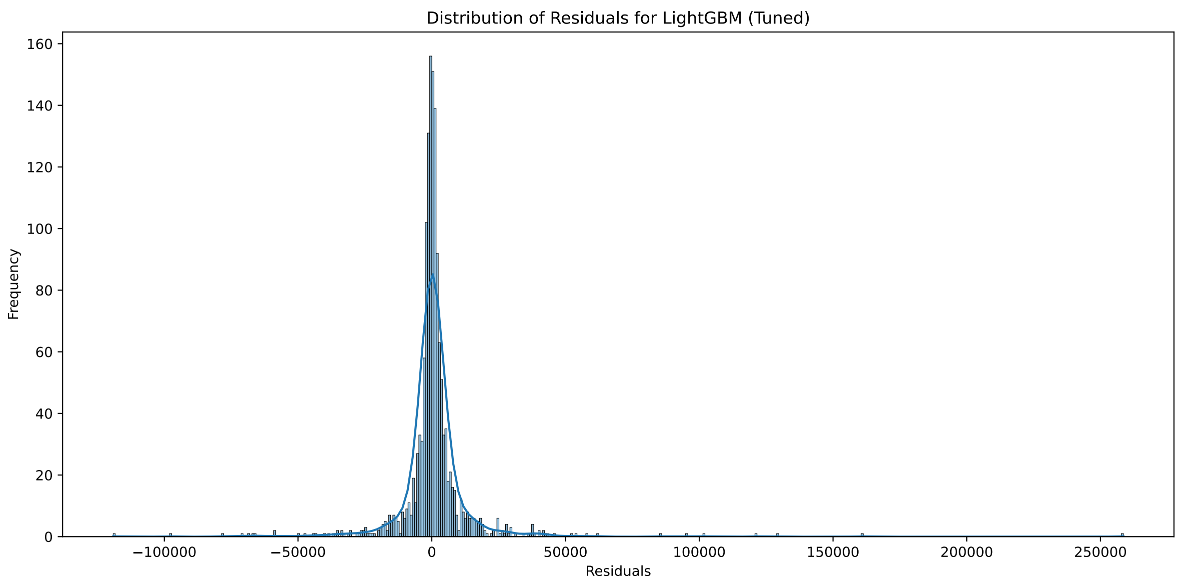
```python
plt.title(f"Distribution of Residuals for {best_model_name}")
plt.tight_layout()
plt.show()

print("\nModel Analytics:")
print(f"1. The best performing model is {best_model_name} based on RMSE.")
print(
    f"2. This model explains {best_model['R2']:.2%} of the variance in the target variable."
)
print(
    f"3. On average, this model's predictions deviate from the actual prices by ${best_model
)
print(
    "4. The residual plot and distribution can help identify any patterns in the model's erro
)
print(
    "5. Feature importance plot shows which features are most influential in the model's pred
)
```

```
Model Performance Comparison:
                   Model          RMSE           MAE        R2
5        LightGBM (Tuned)  15110.314189   6191.016792  0.963797
7        CatBoost (Tuned)  15878.735704   9573.044451  0.960022
6         XGBoost (Tuned)  16082.677926   9731.326948  0.958988
4   Random Forest (Tuned)  17689.051733   9273.214268  0.950386
0       Linear Regression  28668.601875  16544.456320  0.869682
1                   Ridge  28677.453861  16544.187325  0.869601
2                   Lasso  80488.555134  55668.344525 -0.027212
3              ElasticNet  80488.555134  55668.344525 -0.027212

Best Performing Model:
Model: LightGBM (Tuned)
RMSE: 15110.3142
MAE: 6191.0168
R2 Score: 0.9638

Model Analytics:
1. The best performing model is LightGBM (Tuned) based on RMSE.
2. This model explains 96.38% of the variance in the target variable.
3. On average, this model's predictions deviate from the actual prices by $6191.02.
4. The residual plot and distribution can help identify any patterns in the model's errors.
5. Feature importance plot shows which features are most influential in the model's predicti
```


Model Performance Comparison (RMSE)
RMSE
0
10000
20000
30000
40000
50000
60000
70000
80000
LightGBM (Tuned)
CatBoost (Tuned)
XGBoost (Tuned)
Random Forest (Tuned)
Linear Regression
Ridge
Lasso
ElasticNet
Model


Top 20 Feature Importances - LightGBM (Tuned)
feature
GrLivArea
LotArea
GarageArea
TotalBsmtSF
Id
YearBuilt
OpenPorchSF
1stFlrSF
BsmtFinSF1
BsmtUnfSF
YearRemodAdd
LotFrontage
GarageYrBlt
Neighborhood
MoSold
OverallCond
MasVnrArea
WoodDeckSF
OverallQual
YrSold
0
50
100
150
200
250
300
importance





Conclusion

This notebook implements a comprehensive approach to the House Prices regression task, including:

1. Exploratory Data Analysis (EDA) to understand the dataset
2. Data preprocessing, including handling missing values and encoding categorical variables by using LabelEncoder

3. Implementation of both basic (linear) and advanced (tree-based) regression models like LinearRegression, Ridge, Lasso, ElasticNet, RandomForestRegressor, LightGBM, XGBoost, and CatBoost
4. Visualization of model performance and feature importance using scatter plots and heatmaps
5. Hyperparameter tuning to optimize model performance using GridSearchCV
6. Final model evaluation and ensemble prediction using the average of all models

Key observations: 1. The log transformation of the target variable (SalePrice) helped to handle its skewed distribution. 2. Advanced models (Random Forest, LightGBM, XGBoost, CatBoost) generally outperformed linear models. 3. Feature importance analysis revealed key predictors of house prices, which align with domain knowledge. 4. Hyperparameter tuning improved the performance of all models. 5. The ensemble of tuned models provides a robust final prediction.

Areas for further improvement: 1. More extensive feature engineering, such as creating interaction terms or domain-specific features. 2. Experimenting with more advanced ensemble methods, such as stacking. 3. Deeper analysis of residuals to identify patterns in prediction errors and potential outliers. 4. Consideration of model interpretability for stakeholder communication. 5. Using more advanced preprocessing techniques like OneHotEncoder for categorical variables. 6. Using SHAP values to explain the predictions of the final model. 7. Using Bayesian Optimization to tune hyperparameters.