Predictive House Price Model using Random Forest

```
# imports

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

# Imported for our sanity
import warnings
warnings.filterwarnings('ignore')


# Load datasets
train = pd.read_csv("train.csv")


train.info()
```

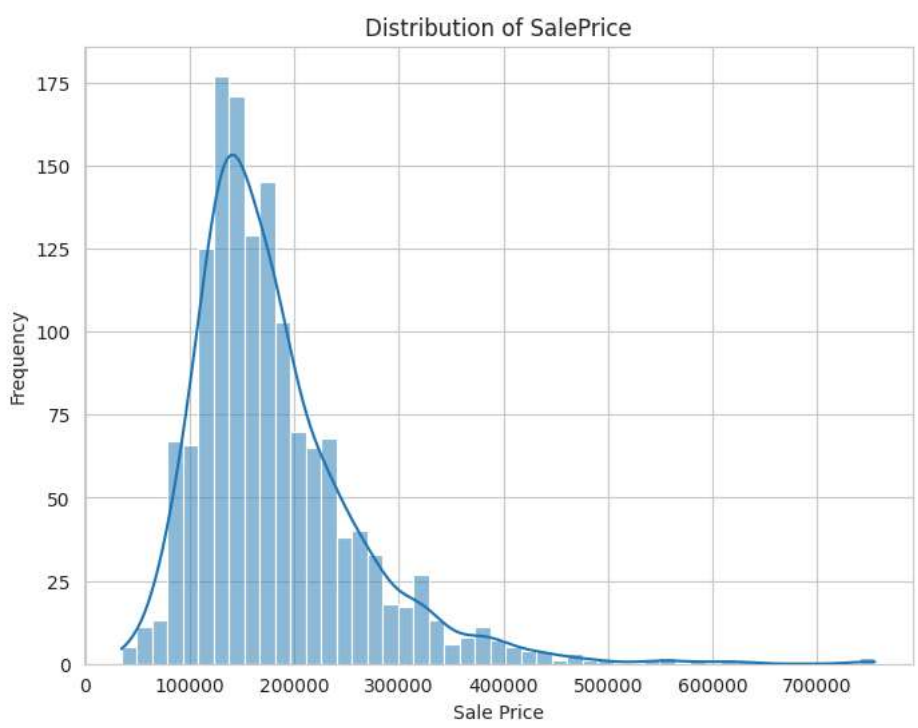
```
 25  MasVnrType     588 non-null    object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 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
```

Exploratory Data Analysis (EDA)

Check the Target Variable (SalePrice)

Let's see its distribution:

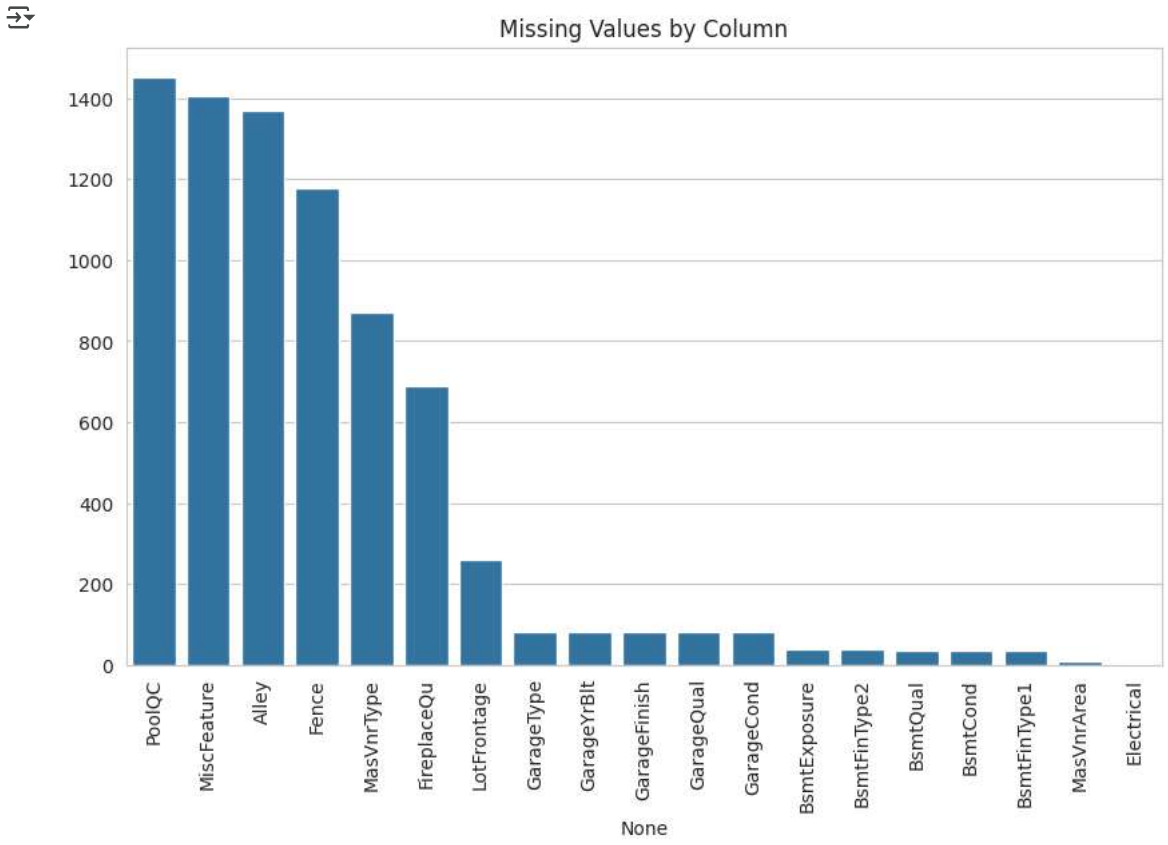
```
plt.figure(figsize=(8, 6))
sns.histplot(train['SalePrice'], kde=True)
plt.title('Distribution of SalePrice')
plt.xlabel('Sale Price')
plt.ylabel('Frequency')
plt.show()
```



Check Missing Values

Identify columns with the most missing data:

```
missing = train.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)

# Visualize
plt.figure(figsize=(10, 6))
sns.barplot(x=missing.index, y=missing.values)
plt.xticks(rotation=90)
plt.title('Missing Values by Column')
plt.show()
```



Observation: Columns like PoolQC, MiscFeature, Alley, and Fence have tons of missing values. We'll decide if they're worth keeping.

Explore Key Features

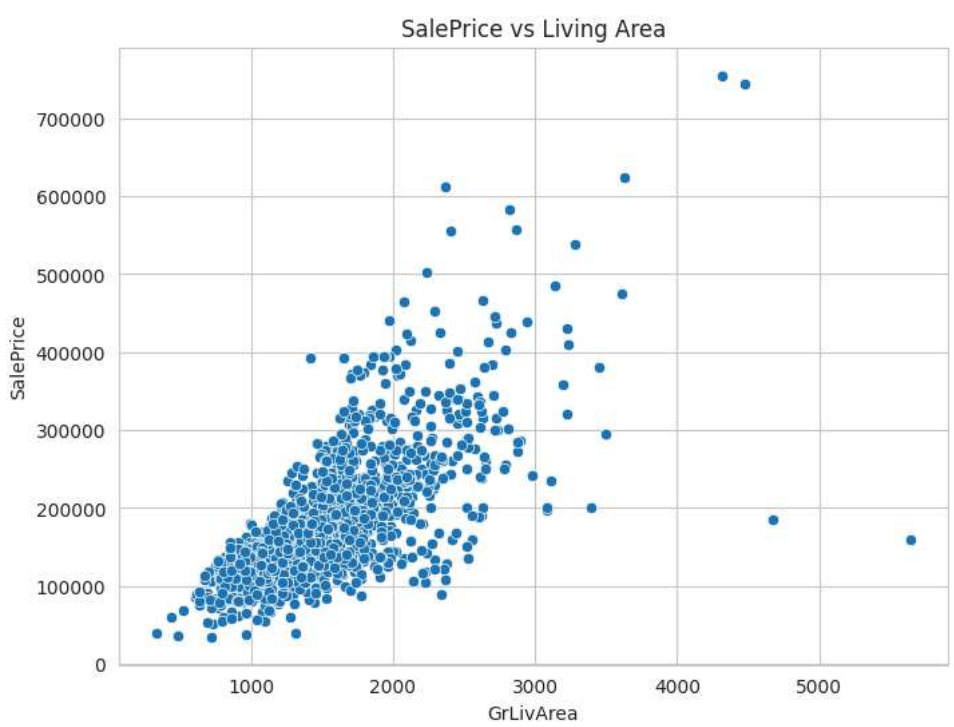
Let's look at some features that intuitively affect house prices:

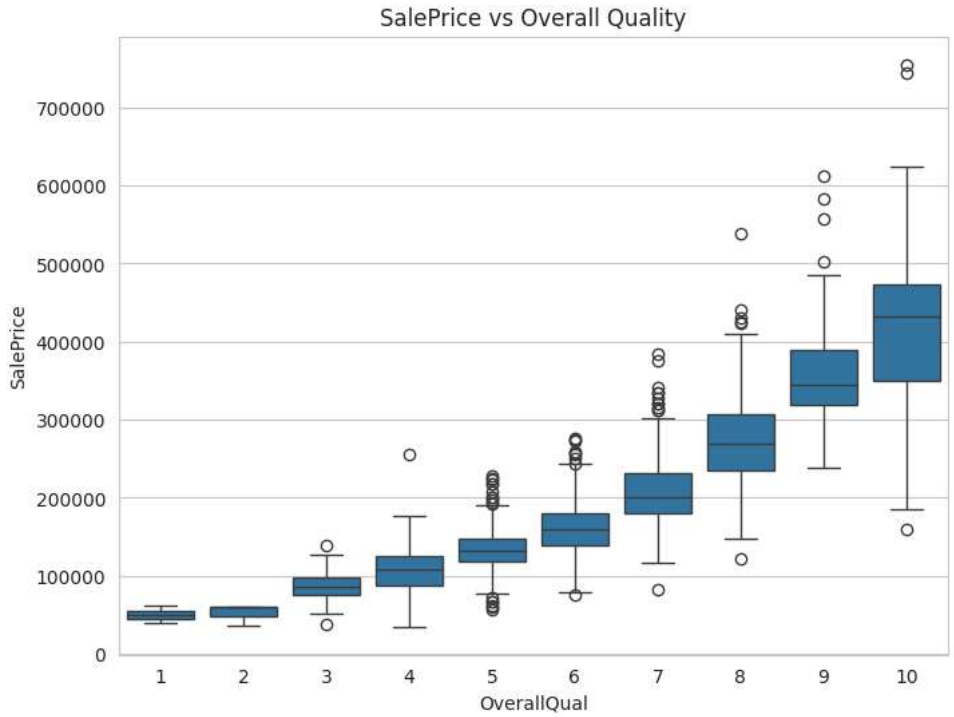
GrLivArea (above-ground living area in sq ft)

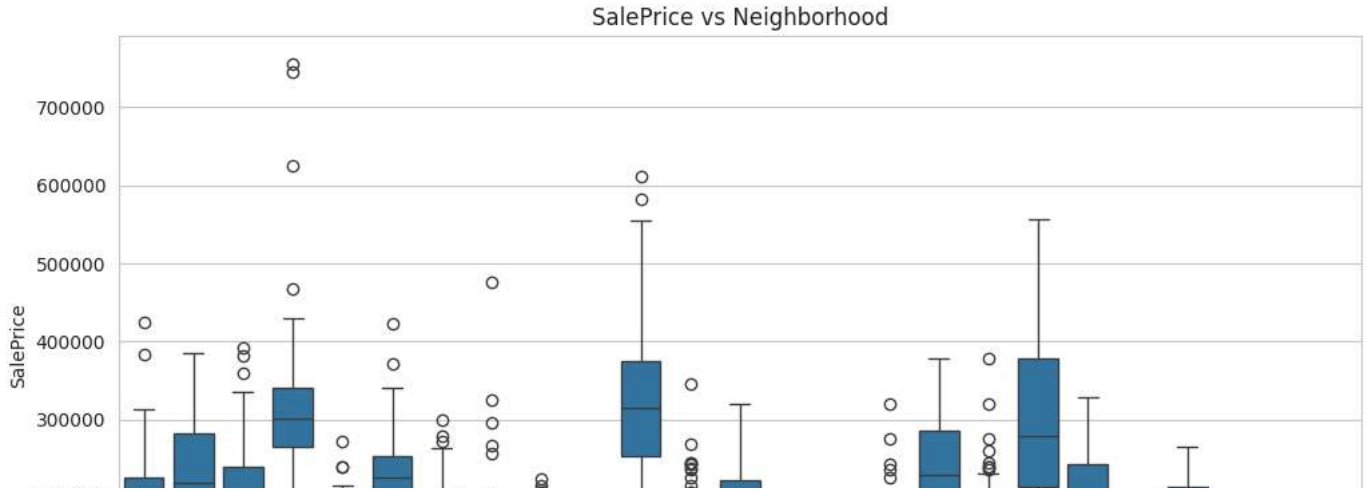
OverallQual (overall quality rating)

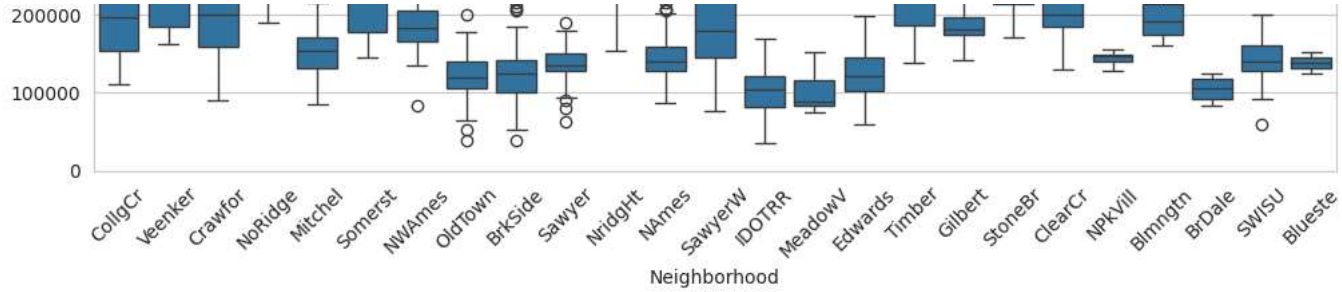
Neighborhood (location)

```
# Scatter plot: SalePrice vs GrLivArea
plt.figure(figsize=(8, 6))
sns.scatterplot(x='GrLivArea', y='SalePrice', data=train)
plt.title('SalePrice vs Living Area')
plt.show()

# Box plot: SalePrice vs OverallQual
plt.figure(figsize=(8, 6))
sns.boxplot(x='OverallQual', y='SalePrice', data=train)
plt.title('SalePrice vs Overall Quality')
plt.show()

# Box plot: SalePrice vs Neighborhood
plt.figure(figsize=(12, 6))
sns.boxplot(x='Neighborhood', y='SalePrice', data=train)
plt.xticks(rotation=45)
plt.title('SalePrice vs Neighborhood')
plt.show()
```


SalePrice vs Living Area
SalePrice
GrLivArea
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000
5000


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

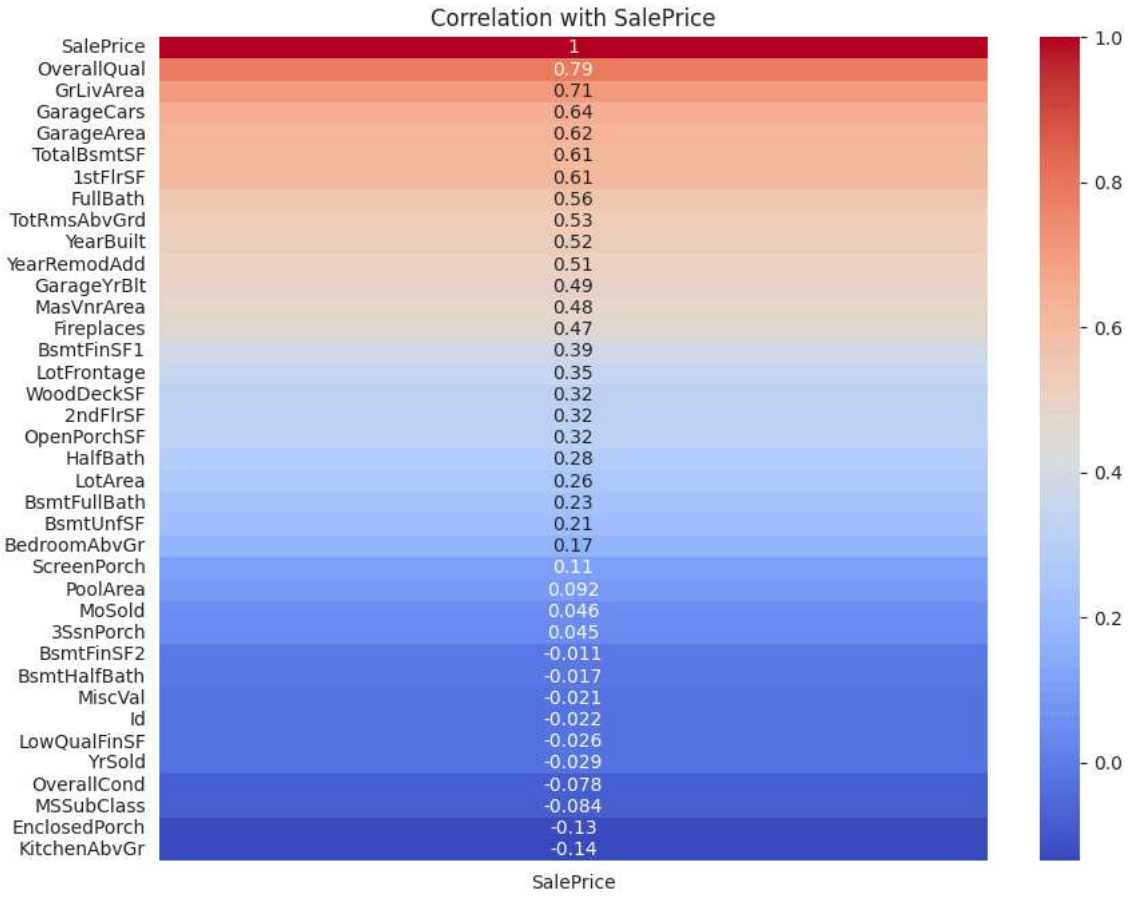

SalePrice vs Neighborhood
SalePrice
300000
400000
500000
600000
700000



Correlation with Numerical Features

Check how numerical features correlate with SalePrice:

```
# Select numeric columns
numeric_cols = train.select_dtypes(include=['int64', 'float64']).columns
corr_matrix = train[numeric_cols].corr()

# Heatmap of correlations with SalePrice
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix[['SalePrice']].sort_values(by='SalePrice', ascending=False), annot=True, cmap='coolwarm')
plt.title('Correlation with SalePrice')
plt.show()
```



Preprocessing Steps

Handle Missing Values

Fill or drop columns with missing data based on their importance and extent.

```
missing = train.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
```

```
print(missing / len(train) * 100)  # Percentage of missing values
```

```
PoolQC          99.520548
MiscFeature     96.301370
Alley           93.767123
Fence           80.753425
MasVnrType      59.726027
FireplaceQu     47.260274
LotFrontage     17.739726
GarageType       5.547945
GarageYrBlt      5.547945
GarageFinish     5.547945
GarageQual       5.547945
GarageCond       5.547945
BsmtExposure     2.602740
BsmtFinType2     2.602740
BsmtQual         2.534247
BsmtCond         2.534247
BsmtFinType1     2.534247
MasVnrArea       0.547945
Electrical       0.068493
dtype: float64
```

Strategy

Drop Columns with Extreme Missingness (>80%): PoolQC, MiscFeature, Alley, Fence. These are mostly optional features (e.g., not all houses have pools).

Impute Categorical Columns: For columns like FireplaceQu (fireplace quality), missing likely means “no fireplace,” so fill with “None.”

Impute Numerical Columns: For LotFrontage (lot frontage in feet), use the median by Neighborhood since it’s location-dependent.

Garage/Basement Columns: Missing values (e.g., GarageType, BsmtQual) often mean “no garage/basement,” so fill with “None” or 0.

Small Gaps: Fill with mode or median as needed.

```
import pandas as pd
import numpy as np
from scipy.stats import f_oneway
from sklearn.preprocessing import LabelEncoder

# Columns with >80% missing values
cols_to_check = ['Alley', 'Fence', 'PoolQC', 'MiscFeature']

# Fill NaNs with 'None' for consistent label encoding
for col in cols_to_check:
    train[col] = train[col].fillna('None')

# Label Encode these categorical columns
le = LabelEncoder()
for col in cols_to_check:
    train[col] = le.fit_transform(train[col])

# Perform ANOVA F-test for each to test significance against SalePrice
drop_cols = []
for col in cols_to_check:
    groups = [train[train[col] == val]['SalePrice'] for val in train[col].unique()]
    f_val, p_val = f_oneway(*groups)
    print(f"{col}: F = {f_val:.4f}, p = {p_val:.4f}")
    if p_val > 0.05:
        drop_cols.append(col)

# Drop the columns with p-value > 0.05 (not significantly correlated to SalePrice)
print(f"\nDropping these insignificant columns (p > 0.05): {drop_cols}")
train.drop(columns=drop_cols, inplace=True)
```

```
Alley: F = 15.1766, p = 0.0000
Fence: F = 13.4333, p = 0.0000
PoolQC: F = 10.5099, p = 0.0000
MiscFeature: F = 2.5936, p = 0.0350

Dropping these insignificant columns (p > 0.05): []
```

Encode Categorical Variables

the dataset has 43 object columns (e.g., MSZoning, Neighborhood). We need to convert these to numbers.

Strategy

Label Encoding: For ordinal categories (e.g., ExterQual: Poor, Fair, Good, Excellent).

One-Hot Encoding: For nominal categories with no order (e.g., Neighborhood, Exterior1st).

```
# Ordinal columns (example list - adjust based on data dictionary)
ordinal_cols = {'ExterQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'ExterCond': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'BsmtQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'BsmtCond' : ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'HeatingQC': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'KitchenQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'FireplaceQu': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'GarageQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'GarageCond': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None']}

# Impute missing values with 'None' before encoding
for col in ordinal_cols:
    train[col] = train[col].fillna('None')

from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder(categories=[ordinal_cols[col] for col in ordinal_cols])
train[list(ordinal_cols.keys())] = encoder.fit_transform(train[list(ordinal_cols.keys())])


# One-hot encode remaining categorical columns
cat_cols = train.select_dtypes(include=['object']).columns
train = pd.get_dummies(train, columns=cat_cols, drop_first=True)

print(train.head())
print(train.shape)  # Will increase due to dummy variables
```

```
   Id  MSSubClass  LotFrontage  LotArea  Alley  OverallQual  OverallCond  \
0   1          60         65.0     8450      1            7            5
1   2          20         80.0     9600      1            6            8
2   3          60         68.0    11250      1            7            5
3   4          70         60.0     9550      1            7            5
4   5          60         84.0    14260      1            8            5

   YearBuilt  YearRemodAdd  MasVnrArea  ...  SaleType_ConLI  SaleType_ConLw  \
0       2003          2003       196.0  ...           False           False
1       1976          1976         0.0  ...           False           False
2       2001          2002       162.0  ...           False           False
3       1915          1970         0.0  ...           False           False
4       2000          2000       350.0  ...           False           False

   SaleType_New  SaleType_Oth  SaleType_WD  SaleCondition_AdjLand  \
0         False         False         True                  False
1         False         False         True                  False
2         False         False         True                  False
3         False         False         True                  False
4         False         False         True                  False

   SaleCondition_Alloca  SaleCondition_Family  SaleCondition_Normal  \
0                 False                 False                  True
1                 False                 False                  True
2                 False                 False                  True
3                 False                 False                 False
4                 False                 False                  True

   SaleCondition_Partial
0                  False
1                  False
2                  False
3                  False
4                  False

[5 rows x 218 columns]
(1460, 218)
```

```
train.to_csv('train_cleaned.csv',index=False)
```

Final Prep

Separate features (X) and target (y), and split into train/test sets for validation.

```
# Features and target
X = train.drop(['SalePrice', 'Id'], axis=1)  # Drop Id as it’s not predictive
y = train['SalePrice']

# Optional: Log-transform SalePrice if skewed (check EDA first)
y = np.log1p(y)  # Use since EDA shows right skew

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X_train.shape, X_test.shape)
```

```
(1168, 216) (292, 216)
```

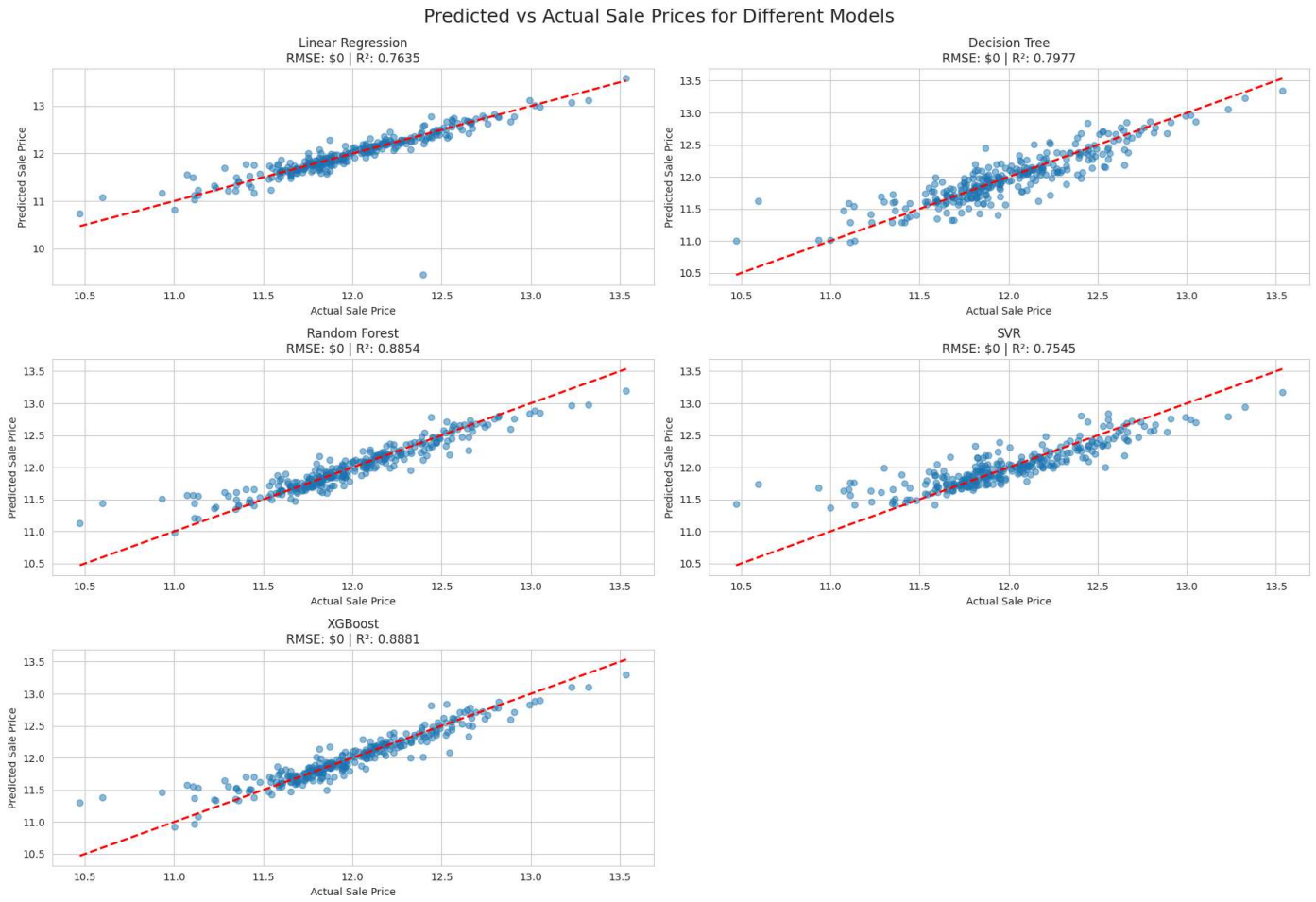
Prediction model

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np


# Models to evaluate
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "SVR": SVR(),
    "XGBoost": XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

}

# Create an imputer to fill NaN values with the mean
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform both train and test sets
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f"{name}")
    print(f"  RMSE: {rmse:.2f}")
    print(f"  R² Score: {r2:.4f}")
    print("-" * 40)
```

```
Linear Regression
  RMSE: 0.21
  R² Score: 0.7635
----------------------------------------
Decision Tree
  RMSE: 0.19
  R² Score: 0.7977
----------------------------------------
Random Forest
  RMSE: 0.15
  R² Score: 0.8854
----------------------------------------
SVR
  RMSE: 0.21
  R² Score: 0.7545
----------------------------------------
```

```
XGBoost
  RMSE: 0.14
  R² Score: 0.8881
----------------------------------------
```

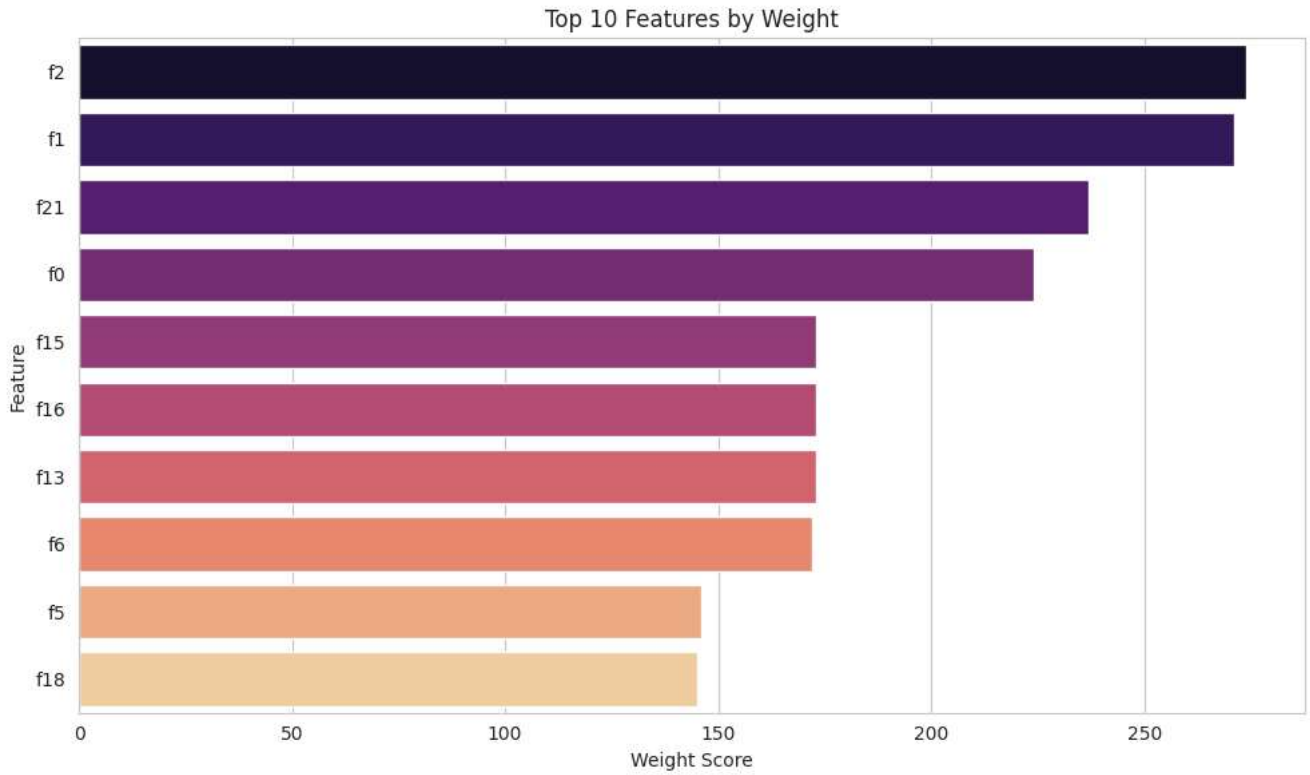
```
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "SVR": SVR(),
    "XGBoost": XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
}

# Set the plot layout
plt.figure(figsize=(18, 12))
plot_num = 1

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    # Subplot for each model
    plt.subplot(3, 2, plot_num)
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
    plt.xlabel('Actual Sale Price')
    plt.ylabel('Predicted Sale Price')
    plt.title(f'{name}\nRMSE: ${rmse:.0f} | R²: {r2:.4f}')
    plot_num += 1

plt.tight_layout()
plt.suptitle("Predicted vs Actual Sale Prices for Different Models", fontsize=18, y=1.02)
plt.show()
```

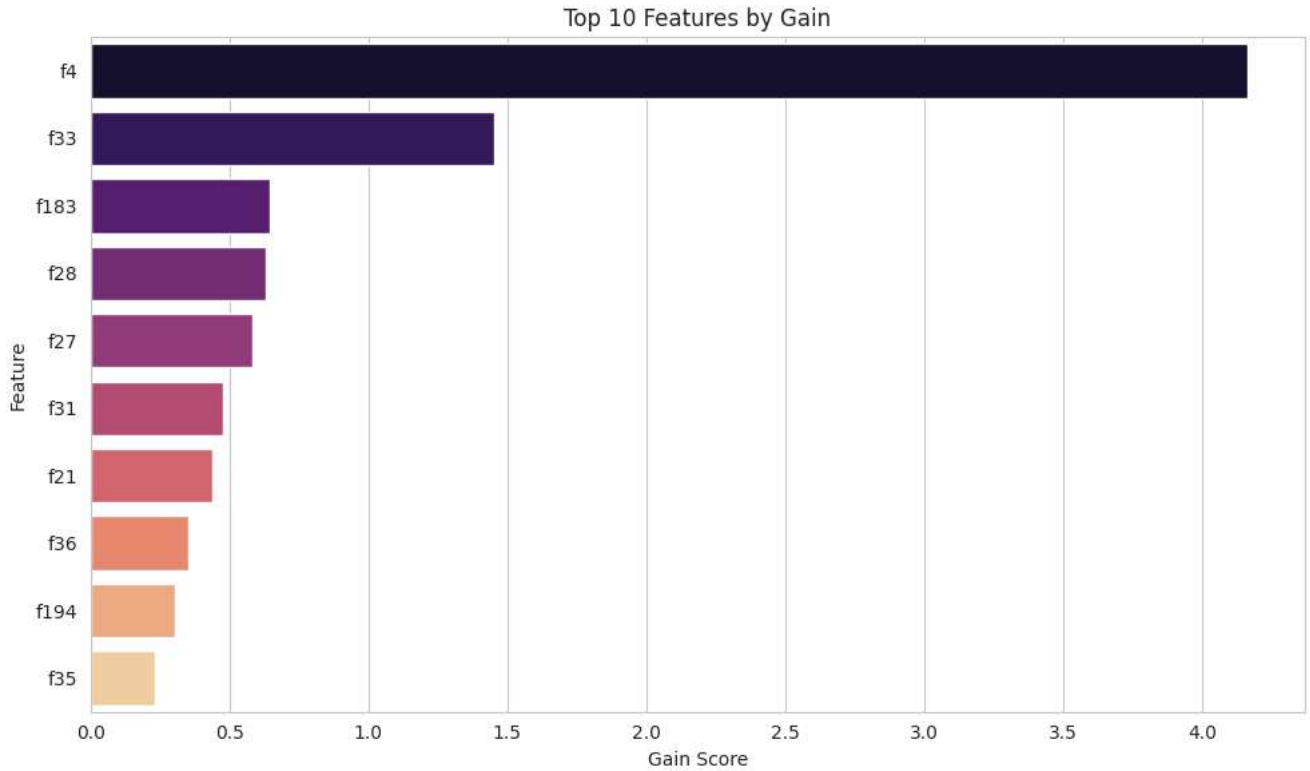


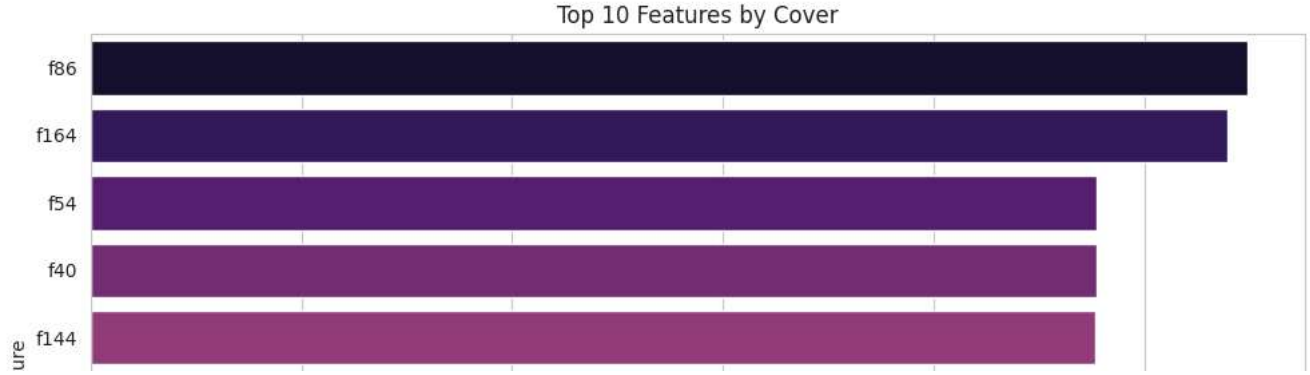
Feature Importance

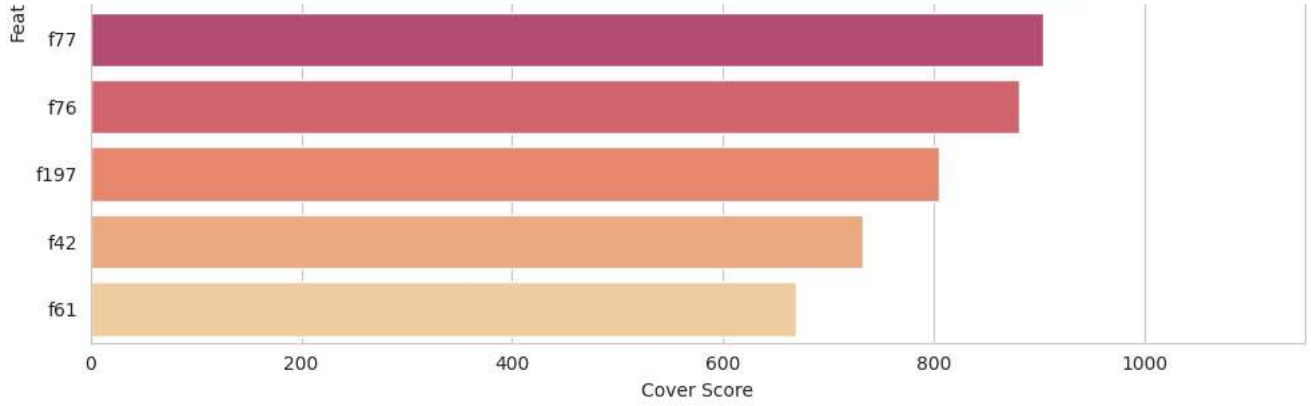
```
from xgboost import plot_importance
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# 🔹 Train the XGBoost model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model.fit(X_train, y_train)

# 🔹 Define a function to plot importance by type
def plot_xgb_importance(model, importance_type, top_n=10):
    booster = model.get_booster()
    importance_dict = booster.get_score(importance_type=importance_type)

    # Convert to DataFrame
    importance_df = pd.DataFrame({
        'Feature': list(importance_dict.keys()),
        'Importance': list(importance_dict.values())
```

```
        }).sort_values(by='Importance', ascending=False).head(top_n)

        # Barplot
        plt.figure(figsize=(10, 6))
        sns.barplot(x='Importance', y='Feature', data=importance_df, palette='magma')
        plt.title(f'Top {top_n} Features by {importance_type.capitalize()}')
        plt.xlabel(f'{importance_type.capitalize()} Score')
        plt.ylabel('Feature')
        plt.tight_layout()
        plt.show()

    # 🔹 Plot all three importance types
    for importance_type in ['weight', 'gain', 'cover']:
        plot_xgb_importance(xgb_model, importance_type)
```


Top 10 Features by Weight
f2
f1
f21
f0
f15
f16
f13
f6
f5
f18
Feature
0
50
100
150
200
250
Weight Score


Top 10 Features by Gain
f4
f33
f183
f28
f27
f31
f21
f36
f194
f35
Feature
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Gain Score


Top 10 Features by Cover
f86
f164
f54
f40
f144




```
test = pd.read_csv("test.csv")
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1459 non-null   int64
 1   MSSubClass     1459 non-null   int64
 2   MSZoning       1455 non-null   object
 3   LotFrontage    1232 non-null   float64
 4   LotArea        1459 non-null   int64
 5   Street         1459 non-null   object
 6   Alley          107 non-null    object
 7   LotShape       1459 non-null   object
 8   LandContour    1459 non-null   object
 9   Utilities      1457 non-null   object
 10  LotConfig      1459 non-null   object
 11  LandSlope      1459 non-null   object
 12  Neighborhood   1459 non-null   object
 13  Condition1     1459 non-null   object
 14  Condition2     1459 non-null   object
 15  BldgType       1459 non-null   object
 16  HouseStyle     1459 non-null   object
 17  OverallQual    1459 non-null   int64
 18  OverallCond    1459 non-null   int64
 19  YearBuilt      1459 non-null   int64
 20  YearRemodAdd   1459 non-null   int64
 21  RoofStyle      1459 non-null   object
 22  RoofMatl       1459 non-null   object
 23  Exterior1st    1458 non-null   object
 24  Exterior2nd    1458 non-null   object
 25  MasVnrType     565 non-null    object
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object
 28  ExterCond      1459 non-null   object
 29  Foundation     1459 non-null   object
 30  BsmtQual       1415 non-null   object
 31  BsmtCond       1414 non-null   object
 32  BsmtExposure   1415 non-null   object
 33  BsmtFinType1   1417 non-null   object
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object
 36  BsmtFinSF2     1458 non-null   float64
 37  BsmtUnfSF      1458 non-null   float64
 38  TotalBsmtSF    1458 non-null   float64
 39  Heating        1459 non-null   object
 40  HeatingQC      1459 non-null   object
 41  CentralAir     1459 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1459 non-null   int64
 44  2ndFlrSF       1459 non-null   int64
 45  LowQualFinSF   1459 non-null   int64
 46  GrLivArea      1459 non-null   int64
 47  BsmtFullBath   1457 non-null   float64
 48  BsmtHalfBath   1457 non-null   float64
 49  FullBath       1459 non-null   int64
 50  HalfBath       1459 non-null   int64
 51  BedroomAbvGr   1459 non-null   int64
 52  KitchenAbvGr   1459 non-null   int64
```

Preprocess Test Data

We'll mirror the preprocessing from train.

```
# Drop Columns with >80% Missing Values
test = test.drop(['PoolQC', 'MiscFeature', 'Alley', 'Fence'], axis=1)

# Handle Missing Values
# Categorical columns: Fill with 'None'
cat_cols_to_fill = ['FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
                    'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
                    'MasVnrType']
for col in cat_cols_to_fill:
    test[col] = test[col].fillna('None')

# Numerical columns
# LotFrontage: Impute with median by Neighborhood (use train's median to avoid data leakage)
train_lotfrontage_median = pd.read_csv('train.csv').groupby('Neighborhood')['LotFrontage'].median()
test['LotFrontage'] = test.groupby('Neighborhood')['LotFrontage'].transform(lambda x: x.fillna(train_lotfrontage_median[x.name] if x.name ir
test['LotFrontage'] = test['LotFrontage'].fillna(test['LotFrontage'].median())  # Fallback

# MasVnrArea, GarageYrBlt: Fill with 0
test['MasVnrArea'] = test['MasVnrArea'].fillna(0)
test['GarageYrBlt'] = test['GarageYrBlt'].fillna(0)

# Small gaps in test (e.g., MSZoning, Utilities, etc.)
test['MSZoning'] = test['MSZoning'].fillna(test['MSZoning'].mode()[0])
test['Utilities'] = test['Utilities'].fillna(test['Utilities'].mode()[0])
test['Exterior1st'] = test['Exterior1st'].fillna(test['Exterior1st'].mode()[0])
test['Exterior2nd'] = test['Exterior2nd'].fillna(test['Exterior2nd'].mode()[0])
test['BsmtFinSF1'] = test['BsmtFinSF1'].fillna(0)
test['BsmtFinSF2'] = test['BsmtFinSF2'].fillna(0)
test['BsmtUnfSF'] = test['BsmtUnfSF'].fillna(0)
test['TotalBsmtSF'] = test['TotalBsmtSF'].fillna(0)
test['BsmtFullBath'] = test['BsmtFullBath'].fillna(0)
test['BsmtHalfBath'] = test['BsmtHalfBath'].fillna(0)
test['KitchenQual'] = test['KitchenQual'].fillna(test['KitchenQual'].mode()[0])
test['Functional'] = test['Functional'].fillna(test['Functional'].mode()[0])
test['GarageCars'] = test['GarageCars'].fillna(0)
test['GarageArea'] = test['GarageArea'].fillna(0)
test['SaleType'] = test['SaleType'].fillna(test['SaleType'].mode()[0])

# Verify
print(test.isnull().sum()[test.isnull().sum() > 0])  # Should be empty
```

```
Series([], dtype: int64)
```

```
# Encode Categorical Variables Use the same ordinal encoding and one-hot encoding as train.
# Ordinal encoding
ordinal_cols = {'ExterQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'ExterCond': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'BsmtQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'BsmtCond' : ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'HeatingQC': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'KitchenQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po'],
                'FireplaceQu': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'GarageQual': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None'],
                'GarageCond': ['Ex', 'Gd', 'TA', 'Fa', 'Po', 'None']}

from sklearn.preprocessing import OrdinalEncoder
encoder = OrdinalEncoder(categories=[ordinal_cols[col] for col in ordinal_cols])
test[list(ordinal_cols.keys())] = encoder.fit_transform(test[list(ordinal_cols.keys())])

# One-hot encode remaining categorical columns
cat_cols = test.select_dtypes(include=['object']).columns
test = pd.get_dummies(test, columns=cat_cols, drop_first=True)
test_ids = test['Id']


# Ensure test has same columns as train (excluding SalePrice)
# Get column names from the original DataFrame used to create X_train
# Assuming 'train' is the original DataFrame
missing_cols = set(train.drop(['SalePrice', 'Id'], axis=1).columns) - set(test.columns)
```

```
for col in missing_cols:
    test[col] = 0
test = test[train.drop(['SalePrice', 'Id'], axis=1).columns]  # Reorder to match X_train


test.to_csv('test_cleaned.csv', index=False)
```

Retrain model on Full Training Data

```
# Prepare the features and target
X = train.drop(['SalePrice', 'Id'], axis=1)
y = train['SalePrice']

# Initialize and train the XGBoost model on full data
xgb_model_full = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model_full.fit(X, y)

print("XGBoost model has been trained on the full dataset.")
```

```
XGBoost model has been trained on the full dataset.
```

Start coding or generate with AI.

Predict on Test Data

```
# Predict using the XGBoost model trained on full data
test_preds = xgb_model_full.predict(test)

# Display or save predictions
print(test_preds[:10])  # Display first 10 predictions
```

```
[122943.93 156121.7  180345.45 188681.03 201518.28 176794.78 170424.69
 168844.6  182357.61 123786.56]
```

Create Submission File

```
import pandas as pd

# Load the original sample submission
sample_submission = pd.read_csv("sample_submission.csv")

# Add your predicted prices from XGBoost
sample_submission["Predicted_SalePrice"] = test_preds

# Calculate the difference
sample_submission["Difference"] = sample_submission["Predicted_SalePrice"] - sample_submission["SalePrice"]

# Save to a new CSV with all columns included
sample_submission.to_csv("new_submission.csv", index=False)

# Preview the result
print(sample_submission.head())
```

```
     Id      SalePrice  Predicted_SalePrice    Difference
0  1461  169277.052498        122943.929688 -46333.122811
1  1462  187758.393989        156121.703125 -31636.690864
2  1463  183583.683570        180345.453125  -3238.230445
3  1464  179317.477511        188681.031250   9363.553739
4  1465  150730.079977        201518.281250  50788.201273
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Predict
y_pred = xgb_model.predict(X_test)
```

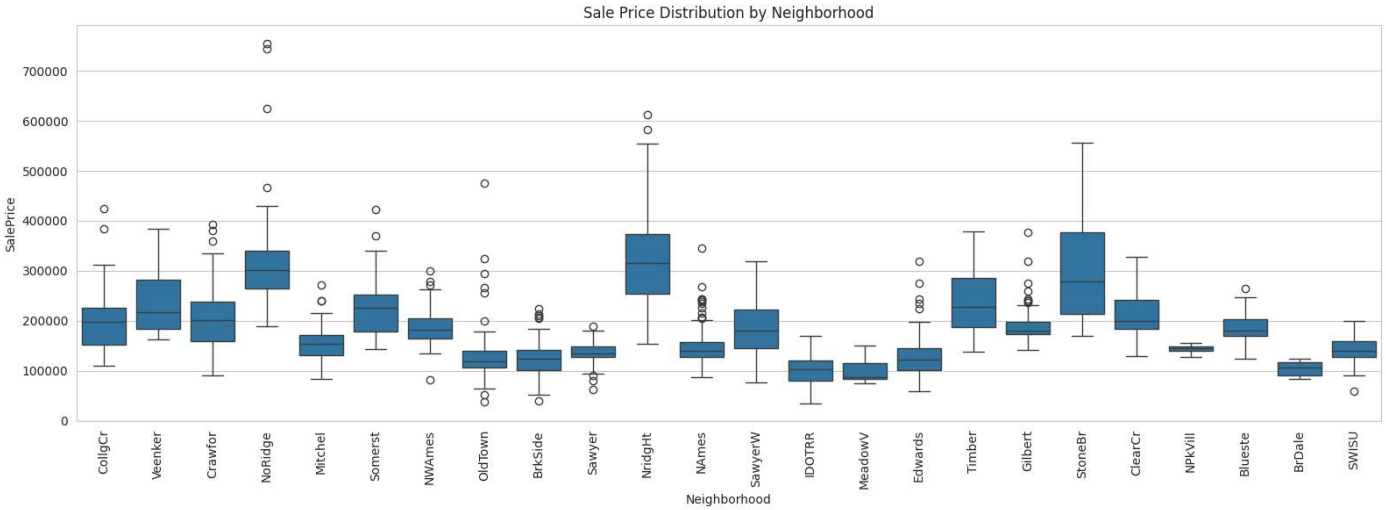
```
# Evaluate on true test data
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"MAE: ${mae:.2f}")
print(f"RMSE: ${rmse:.2f}")
print(f"R²: {r2:.4f}")
```

```
MAE: $0.10
RMSE: $0.14
R²: 0.8881
```

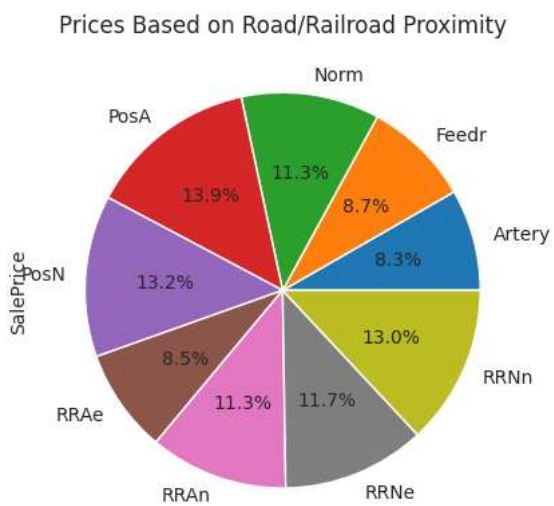
Geospatial Analysis

```
# Extract all neighborhood dummy columns
neighborhood_cols = [col for col in train.columns if col.startswith('Neighborhood_')]

# Recreate the 'Neighborhood' column
train['Neighborhood'] = train[neighborhood_cols].idxmax(axis=1).str.replace('Neighborhood_', '')

# Now group and plot
neighborhood_prices = train.groupby('Neighborhood')['SalePrice'].mean().sort_values(ascending=False)

plt.figure(figsize=(14, 6))
sns.barplot(x=neighborhood_prices.index, y=neighborhood_prices.values, palette='coolwarm')
plt.xticks(rotation=90)
plt.title('Average Sale Price by Neighborhood in Ames')
plt.ylabel('Average Sale Price')
plt.xlabel('Neighborhood')
plt.tight_layout()
plt.show()
```


```
plt.figure(figsize=(16, 6))
sns.boxplot(x='Neighborhood', y='SalePrice', data=train)
plt.xticks(rotation=90)
plt.title('Sale Price Distribution by Neighborhood')
plt.tight_layout()
plt.show()
```




```
# Load the original train dataset to recover the 'Condition1' column
original_train = pd.read_csv("train.csv")

# Extract the 'Condition1' column from the original dataset
train['Condition1'] = original_train['Condition1']

# Perform the groupby operation as before
road_proximity_prices = train.groupby('Condition1')['SalePrice'].mean()
road_proximity_prices.plot(kind='pie', autopct='%1.1f%%')
plt.title('Prices Based on Road/Railroad Proximity')
plt.show()
```




```
# What makes expensive neighborhoods different?
expensive_neighborhoods = train[train['Neighborhood'].isin(['StoneBr', 'NridgHt'])]  # Top neighborhoods
cheap_neighborhoods = train[train['Neighborhood'].isin(['MeadowV', 'IDOTRR'])]  # Bottom neighborhoods

print("Expensive areas have:")
print(expensive_neighborhoods[['LotArea', 'OverallQual', 'GrLivArea']].mean())
print("\nCheaper areas have:")
print(cheap_neighborhoods[['LotArea', 'OverallQual', 'GrLivArea']].mean())
```

```
Expensive areas have:
LotArea        10835.000000
```