In [1]:
```python
# Braden Bourgeois - HOUSE SALE PRICING PREDICTION
# Ames Housing Data Kaggle Competition

## IMPORTS
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats
from scipy.stats import norm
import statsmodels.api as sm
import matplotlib.pyplot as plt
from scipy.stats import skew
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold, cross_val_score, train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression, BayesianRidge
from lightgbm import LGBMRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from catboost import CatBoostRegressor, Pool
from mlxtend.regressor import StackingRegressor
from sklearn.feature_selection import RFE
from sklearn.preprocessing import PolynomialFeatures
from sklearn.experimental import enable_iterative_imputer  # Enable experimental
from sklearn.impute import KNNImputer, IterativeImputer
%matplotlib inline
import warnings
warnings.filterwarnings(action="ignore")

# Load data
house_data = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
test_id = test['Id']
test = test.drop(['Id'], axis=1)
data_w = house_data.copy()
data_w.columns = data_w.columns.str.replace(' ', '')  # Replace white spaces
data_w = data_w.drop(['Id'], axis=1)  # Drop Id early

data_w.info()
data_w.head()
```

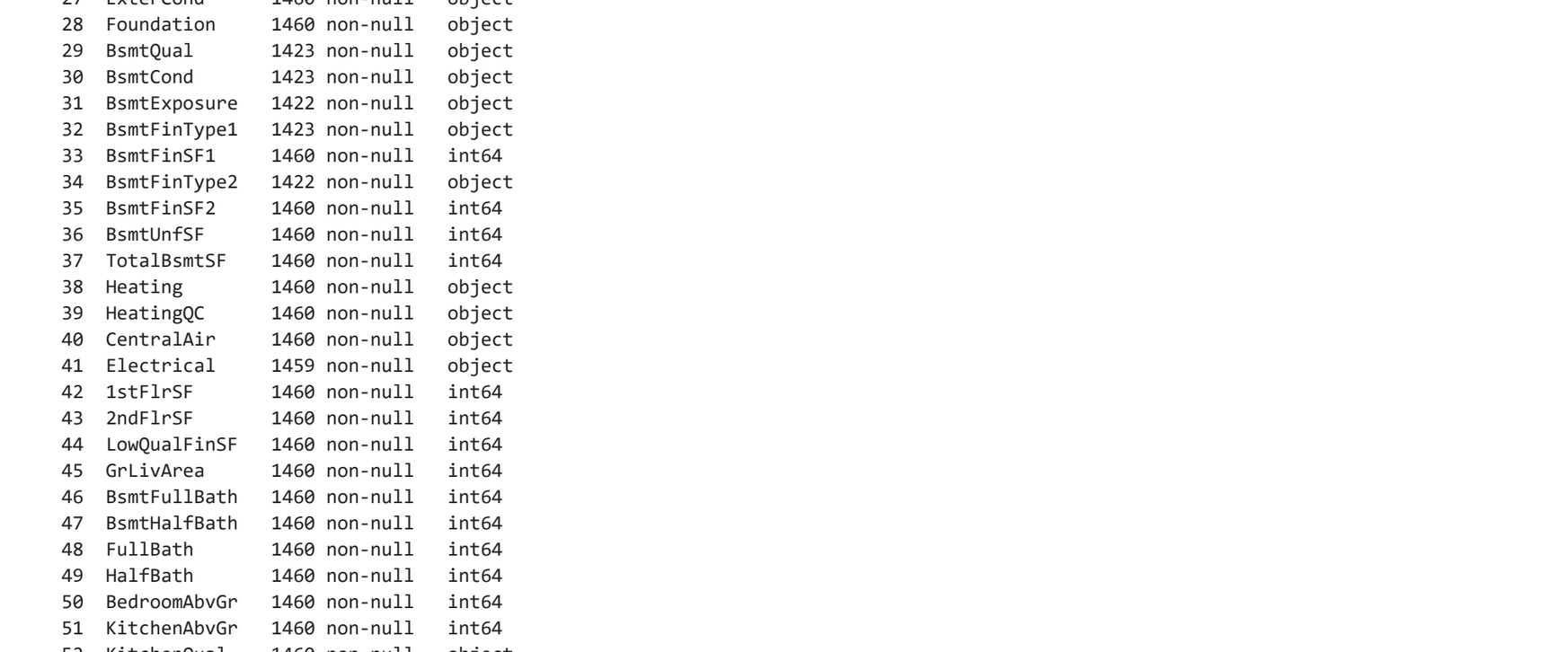
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MSSubClass     1460 non-null   int64
 1   MSZoning       1460 non-null   object
 2   LotFrontage    1201 non-null   float64
 3   LotArea        1460 non-null   int64
 4   Street         1460 non-null   object
 5   Alley          91 non-null     object
 6   LotShape       1460 non-null   object
 7   LandContour    1460 non-null   object
 8   Utilities      1460 non-null   object
 9   LotConfig      1460 non-null   object
 10  LandSlope      1460 non-null   object
 11  Neighborhood   1460 non-null   object
 12  Condition1     1460 non-null   object
 13  Condition2     1460 non-null   object
 14  BldgType       1460 non-null   object
 15  HouseStyle     1460 non-null   object
 16  OverallQual    1460 non-null   int64
 17  OverallCond    1460 non-null   int64
 18  YearBuilt      1460 non-null   int64
 19  YearRemodAdd   1460 non-null   int64
 20  RoofStyle      1460 non-null   object
 21  RoofMatl       1460 non-null   object
 22  Exterior1st    1460 non-null   object
 23  Exterior2nd    1460 non-null   object
 24  MasVnrType     588 non-null    object
 25  MasVnrArea     1452 non-null   float64
 26  ExterQual      1460 non-null   object
 27  ExterCond      1460 non-null   object
 28  Foundation     1460 non-null   object
 29  BsmtQual       1423 non-null   object
 30  BsmtCond       1423 non-null   object
 31  BsmtExposure   1422 non-null   object
 32  BsmtFinType1   1423 non-null   object
 33  BsmtFinSF1     1460 non-null   int64
 34  BsmtFinType2   1422 non-null   object
 35  BsmtFinSF2     1460 non-null   int64
 36  BsmtUnfSF      1460 non-null   int64
 37  TotalBsmtSF    1460 non-null   int64
 38  Heating        1460 non-null   object
 39  HeatingQC      1460 non-null   object
 40  CentralAir     1460 non-null   object
 41  Electrical     1459 non-null   object
 42  1stFlrSF       1460 non-null   int64
 43  2ndFlrSF       1460 non-null   int64
 44  LowQualFinSF   1460 non-null   int64
 45  GrLivArea      1460 non-null   int64
 46  BsmtFullBath   1460 non-null   int64
 47  BsmtHalfBath   1460 non-null   int64
 48  FullBath       1460 non-null   int64
 49  HalfBath       1460 non-null   int64
 50  BedroomAbvGr   1460 non-null   int64
 51  KitchenAbvGr   1460 non-null   int64
 52  KitchenQual    1460 non-null   object
 53  TotRmsAbvGrd   1460 non-null   int64
 54  Functional     1460 non-null   object
 55  Fireplaces     1460 non-null   int64
 56  FireplaceQu    770 non-null    object
 57  GarageType     1379 non-null   object
 58  GarageYrBlt    1379 non-null   float64
 59  GarageFinish   1379 non-null   object
 60  GarageCars     1460 non-null   int64
 61  GarageArea     1460 non-null   int64
 62  GarageQual     1379 non-null   object
 63  GarageCond     1379 non-null   object
 64  PavedDrive     1460 non-null   object
 65  WoodDeckSF     1460 non-null   int64
 66  OpenPorchSF    1460 non-null   int64
 67  EnclosedPorch  1460 non-null   int64
 68  3SsnPorch      1460 non-null   int64
 69  ScreenPorch    1460 non-null   int64
 70  PoolArea       1460 non-null   int64
 71  PoolQC         7 non-null      object
 72  Fence          281 non-null    object
 73  MiscFeature    54 non-null     object
 74  MiscVal        1460 non-null   int64
 75  MoSold         1460 non-null   int64
 76  YrSold         1460 non-null   int64
 77  SaleType       1460 non-null   object
 78  SaleCondition  1460 non-null   object
 79  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(34), object(43)
memory usage: 912.6+ KB
```

Out[1]:

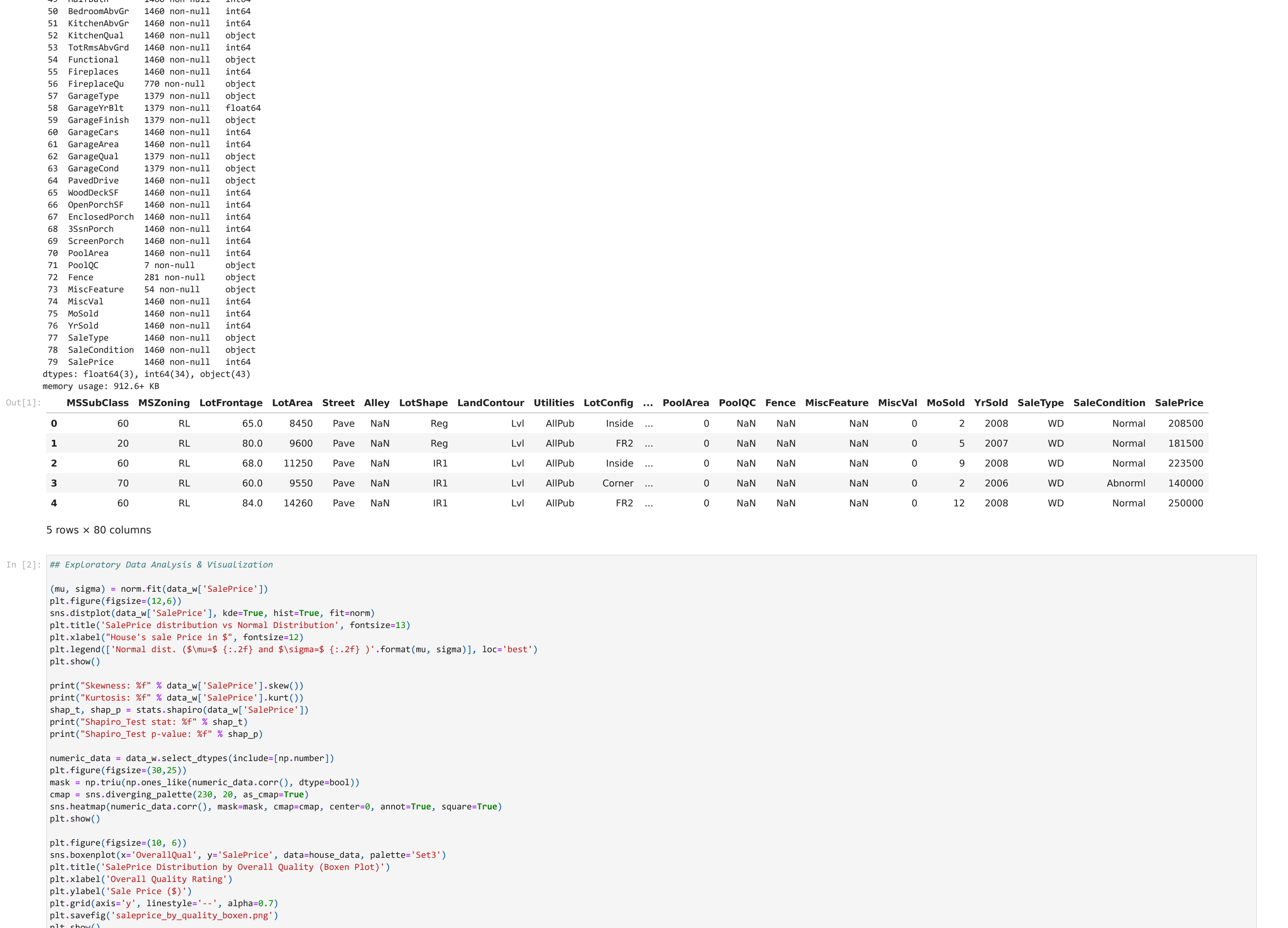
	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

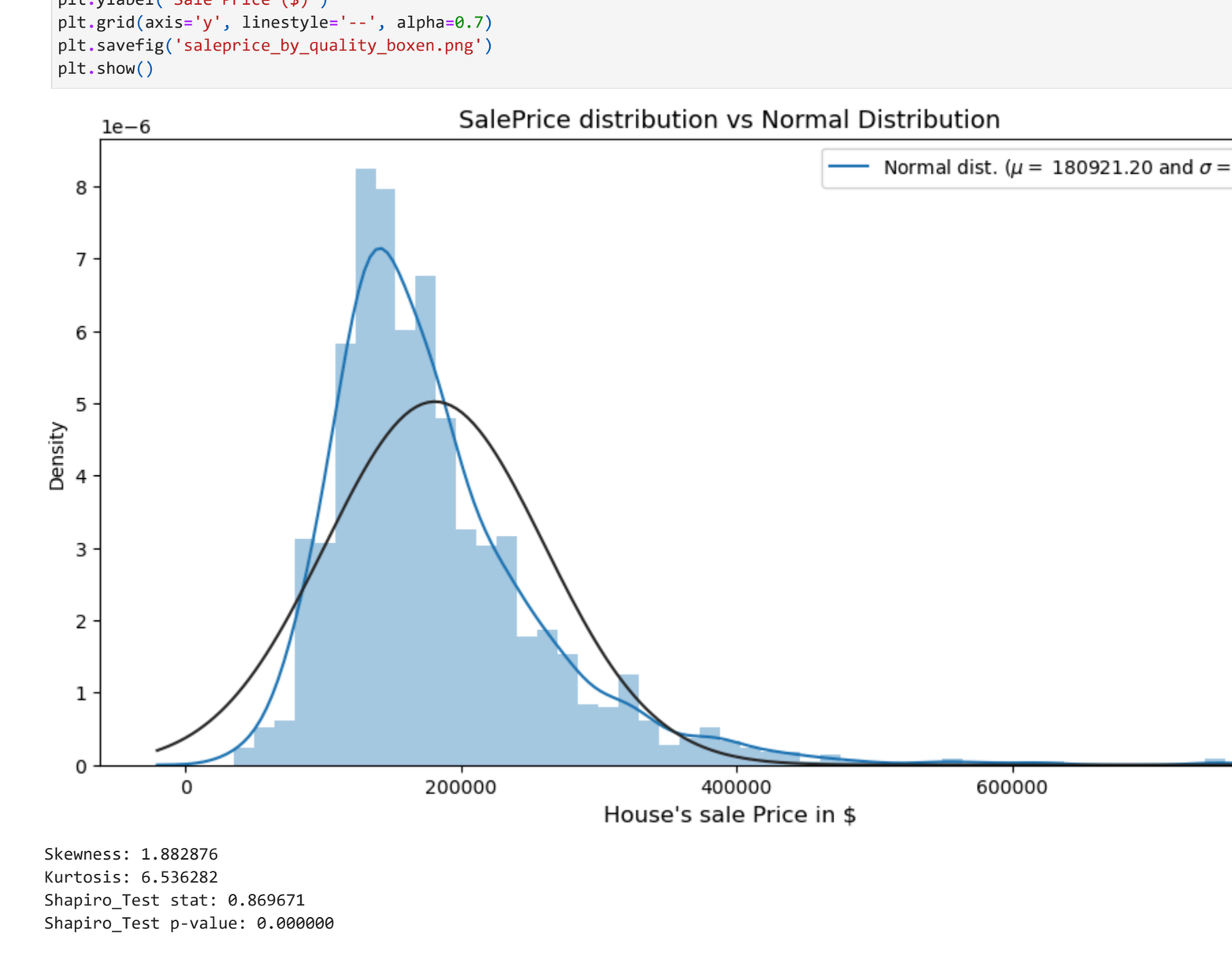
5 rows × 80 columns

In [2]:
```python
## Exploratory Data Analysis & Visualization

(mu, sigma) = norm.fit(data_w['SalePrice'])
plt.figure(figsize=(12,6))
sns.distplot(data_w['SalePrice'], kde=True, hist=True, fit=norm)
plt.title('SalePrice distribution vs Normal Distribution', fontsize=13)
plt.xlabel("House's sale Price in $", fontsize=12)
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.show()

print("Skewness: %f" % data_w['SalePrice'].skew())
print("Kurtosis: %f" % data_w['SalePrice'].kurt())
shap_t, shap_p = stats.shapiro(data_w['SalePrice'])
print("Shapiro_Test stat: %f" % shap_t)
print("Shapiro_Test p-value: %f" % shap_p)

numeric_data = data_w.select_dtypes(include=[np.number])
plt.figure(figsize=(30,25))
mask = np.triu(np.ones_like(numeric_data.corr(), dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(numeric_data.corr(), mask=mask, cmap=cmap, center=0, annot=True, square=True)
plt.show()

plt.figure(figsize=(10, 6))
sns.boxenplot(x='OverallQual', y='SalePrice', data=house_data, palette='Set3')
plt.title('SalePrice Distribution by Overall Quality (Boxen Plot)')
plt.xlabel('Overall Quality Rating')
plt.ylabel('Sale Price ($)')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig('saleprice_by_quality_boxen.png')
plt.show()
```




```
Skewness: 1.882876
Kurtosis: 6.536282
Shapiro_Test stat: 0.869671
Shapiro_Test p-value: 0.000000
```





In [3]:
```python
## Data Preprocessing
target = data_w['SalePrice']
target_log = np.log1p(target)
data_w2 = data_w.drop(['SalePrice'], axis=1)
train_test = pd.concat([data_w2, test], axis=0, sort=False, ignore_index=True)

# Categorical fills

# Drop useless
useless = ['GarageYrBlt', 'YearRemodAdd']
train_test = train_test.drop(useless, axis=1)

# Numeric imputation
ttn_train = data_w2.select_dtypes(include=[np.number])
ttn_test = test.select_dtypes(include=[np.number])
ttc = train_test.select_dtypes(exclude=[np.number])

imputer = IterativeImputer(estimator=RandomForestRegressor(n_estimators=10, random_state=42), max_iter=10, random_state=42)
imputer.fit(ttn_train)  # Fit on train
ttn_train_imputed = pd.DataFrame(imputer.transform(ttn_train), columns=ttn_train.columns, index=ttn_train.index)
ttn_test_imputed = pd.DataFrame(imputer.transform(ttn_test), columns=ttn_test.columns, index=ttn_test.index)

train_test_numeric = pd.concat([ttn_train_imputed, ttn_test_imputed], ignore_index=True)
train_test = pd.concat([train_test_numeric, ttc], axis=1)

# Categorical fills remaining
objects = train_test.select_dtypes(include='object').columns
train_test[objects] = train_test[objects].fillna('None')
```

In [4]:
```python
## Feature Engineering
epsilon = 1e-6
train_test["SqFtPerRoom"] = train_test["GrLivArea"] / (train_test["TotRmsAbvGrd"] + train_test["FullBath"] + train_test["HalfBath"] + train_test["KitchenAbvGr"] + epsilon)
train_test['Total_Home_Quality'] = train_test['OverallQual'] + train_test['OverallCond']
train_test['Total_Bathrooms'] = (train_test['FullBath'] + (0.5 * train_test['HalfBath']) + train_test['BsmtFullBath'] + (0.5 * train_test['BsmtHalfBath']))
train_test["HighQualSF"] = train_test["1stFlrSF"] + train_test["2ndFlrSF"]
train_test['HouseAge'] = train_test['YrSold'].astype(int) - train_test['YearBuilt']
train_test['TotalSF'] = train_test['TotalBsmtSF'] + train_test['1stFlrSF'] + train_test['2ndFlrSF']
train_test['Qual_Size'] = train_test['OverallQual'] * train_test['GrLivArea']

# New features
train_test['IsNew'] = (train_test['YearBuilt'] > 2000).astype(int)
train_test['RemodAge'] = train_test['YrSold'].astype(int) - train_test['YearRemodAdd']
train_test['TotalPorchSF'] = train_test['OpenPorchSF'] + train_test['3SsnPorch'] + train_test['EnclosedPorch'] + train_test['ScreenPorch'] + train_test['WoodDeckSF']
train_test['HasPool'] = train_test['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
train_test['HasFireplace'] = train_test['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
train_test['Qual_Age'] = train_test['OverallQual'] * train_test['HouseAge']

# Polynomials
poly = PolynomialFeatures(degree=2, include_bias=False)
top_feats = ['OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea', 'TotalBsmtSF', 'TotalSF', 'HouseAge']
poly_feats = poly.fit_transform(train_test[top_feats])
poly_df = pd.DataFrame(poly_feats, columns=poly.get_feature_names_out(top_feats), index=train_test.index)
poly_df = poly_df.drop(top_feats, axis=1)  # Drop originals to prevent duplicates
train_test = pd.concat([train_test, poly_df], axis=1)

train_test['MSSubClass'] = train_test['MSSubClass'].astype(str)
train_test['YrSold'] = train_test['YrSold'].astype(str)
train_test['MoSold'] = train_test['MoSold'].astype(str)
train_test_dummy = pd.get_dummies(train_test)
numeric_features = train_test_dummy.select_dtypes(include=[np.number]).columns
skewed_features = train_test_dummy[numeric_features].apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
high_skew = skewed_features[skewed_features > 0.5].index
for i in high_skew:
    train_test_dummy[i] = np.log1p(train_test_dummy[i])

train = train_test_dummy.iloc[:len(house_data)]
test = train_test_dummy.iloc[len(house_data):]
```

In [5]:
```python
## Modeling
X_train, X_val, y_train, y_val = train_test_split(train, target_log, test_size=0.1, random_state=42)

# Define RMSE scorer
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# Tune CatBoost
param_dist = {
    'depth': [4],
    'learning_rate': [0.05],
    'iterations': [1000],
    'l2_leaf_reg': [7],
    'border_count': [64],
    'subsample': [0.6],
    'bootstrap_type': ['Bernoulli']
}
cat = CatBoostRegressor(verbose=0, random_state=42)
random_search = RandomizedSearchCV(cat, param_distributions=param_dist, n_iter=50, cv=5, scoring='neg_mean_squared_error', random_state=42, n_jobs=-1)
random_search.fit(X_train, y_train)
best_cat = random_search.best_estimator_
print("Best CatBoost Params:", random_search.best_params_)
cat_score = rmse(y_val, best_cat.predict(X_val))
print("Tuned CatBoost Validation RMSE:", cat_score)

# Recursive Feature Elimination (RFE) - (top 80)
rfe = RFE(estimator=best_cat, n_features_to_select=80, step=0.05)
rfe.fit(train, target_log)
selected_features = train.columns[rfe.support_]
train_selected = train[selected_features]
test_selected = test[selected_features]
best_cat.fit(train_selected, target_log, early_stopping_rounds=50, eval_set=(X_val[selected_features], y_val))

# Train others
gbr_model = GradientBoostingRegressor(random_state=42)
gbr_model.fit(train_selected, target_log)
lgbm = LGBMRegressor(objective='regression', verbose=-1, random_state=42)
lgbm.fit(train_selected, target_log)
xgb_model = XGBRegressor(random_state=42)
xgb_model.fit(train_selected, target_log)
brr_model = BayesianRidge(compute_score=True)
brr_model.fit(train_selected, target_log)
stack_gen = StackingRegressor(regressors=(best_cat, LinearRegression(), BayesianRidge(), GradientBoostingRegressor(random_state=42)),
                              meta_regressor=best_cat, use_features_in_secondary=True)
stack_gen.fit(train_selected, target_log)

# Inf/Extreme Fix
train_selected = train_selected.replace([np.inf, -np.inf], np.nan).fillna(train_selected.median(numeric_only=True))
test_selected = test_selected.replace([np.inf, -np.inf], np.nan).fillna(train_selected.median(numeric_only=True))
train_selected = np.clip(train_selected, -1e10, 1e10)
test_selected = np.clip(test_selected, -1e10, 1e10)

# Refit on clean
best_cat.fit(train_selected, target_log)
gbr_model.fit(train_selected, target_log)
lgbm.fit(train_selected, target_log)
xgb_model.fit(train_selected, target_log)
brr_model.fit(train_selected, target_log)
stack_gen.fit(train_selected, target_log)
```

```
Best CatBoost Params: {'subsample': 0.6, 'learning_rate': 0.05, 'l2_leaf_reg': 7, 'iterations': 1000, 'depth': 4, 'border_count': 64, 'bootstrap_type': 'Bernoulli'}
Tuned CatBoost Validation RMSE: 0.10280938126395354
```

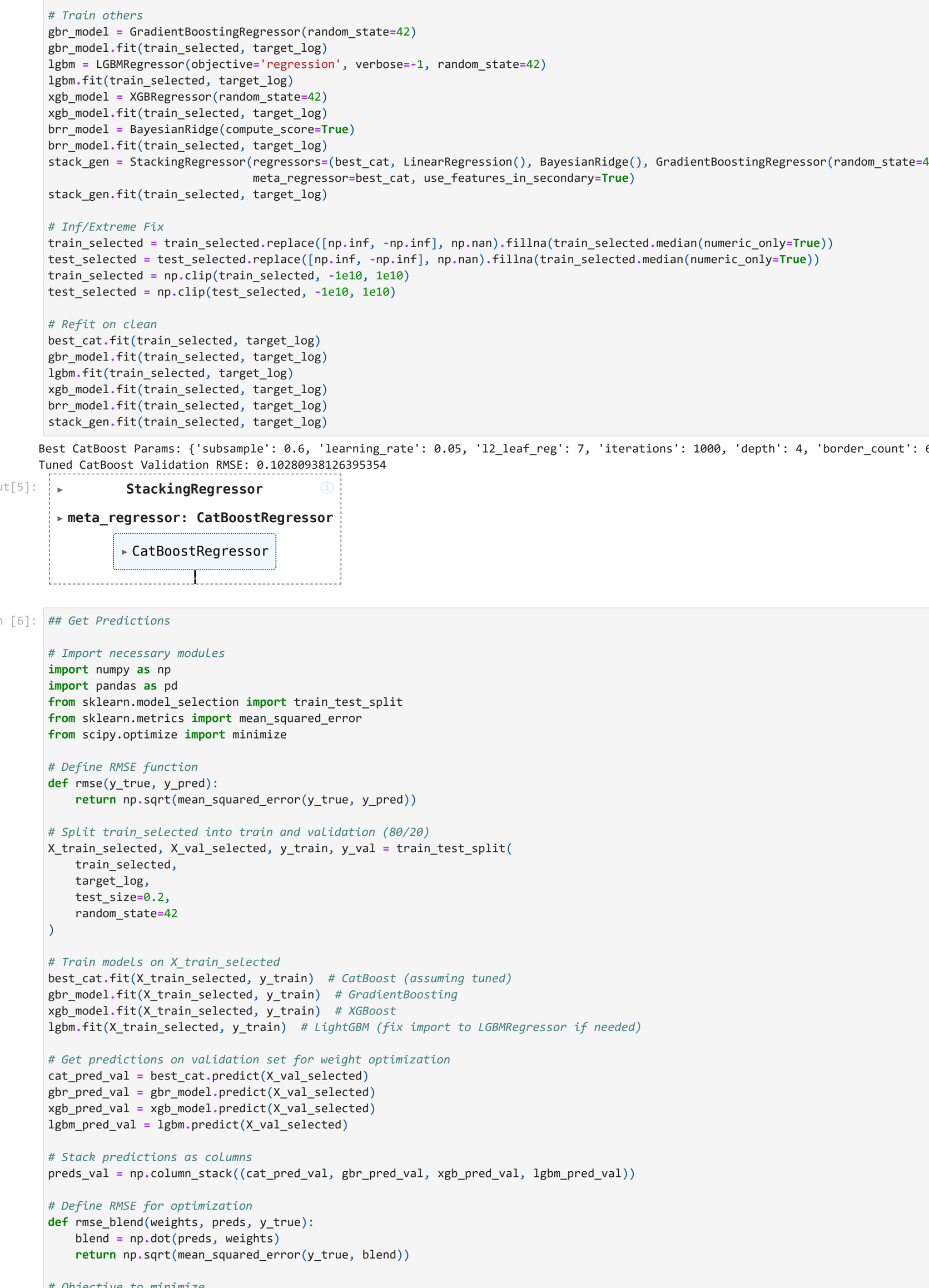
Out[5]:

```
StackingRegressor
meta_regressor: CatBoostRegressor
CatBoostRegressor
```

In [6]:
```python
## Get Predictions

# Import necessary modules
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from scipy.optimize import minimize

# Define RMSE function
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# Split train_selected into train and validation (80/20)
X_train_selected, X_val_selected, y_train, y_val = train_test_split(
    train_selected,
    target_log,
    test_size=0.2,
    random_state=42
)

# Train models on X_train_selected
best_cat.fit(X_train_selected, y_train)  # CatBoost (assuming tuned)
gbr_model.fit(X_train_selected, y_train)  # GradientBoosting
xgb_model.fit(X_train_selected, y_train)  # XGBoost
lgbm.fit(X_train_selected, y_train)  # LightGBM (fix import to LGBMRegressor if needed)

# Get predictions on validation set for weight optimization
cat_pred_val = best_cat.predict(X_val_selected)
gbr_pred_val = gbr_model.predict(X_val_selected)
xgb_pred_val = xgb_model.predict(X_val_selected)
lgbm_pred_val = lgbm.predict(X_val_selected)

# Stack predictions as columns
preds_val = np.column_stack((cat_pred_val, gbr_pred_val, xgb_pred_val, lgbm_pred_val))

# Define RMSE for optimization
def rmse_blend(weights, preds, y_true):
    blend = np.dot(preds, weights)
    return np.sqrt(mean_squared_error(y_true, blend))

# Objective to minimize
def objective(weights):
    return rmse_blend(weights, preds_val, y_val)

# Initial equal weights
initial_weights = np.array([0.25, 0.25, 0.25, 0.25])

# Constraints: sum to 1
constraints = ({'type': 'eq', 'fun': lambda w: np.sum(w) - 1})

# Bounds: each weight between 0 and 1
bounds = [(0, 1) for _ in range(4)]

# Optimize weights
result = minimize(objective, initial_weights, method='SLSQP', bounds=bounds, constraints=constraints)
optimized_weights = result.x
print("Optimized Weights:", optimized_weights)

# Get predictions on test_selected
cat_pred_test = best_cat.predict(test_selected)
gbr_pred_test = gbr_model.predict(test_selected)
xgb_pred_test = xgb_model.predict(test_selected)
lgbm_pred_test = lgbm.predict(test_selected)

# Stack test predictions
preds_test = np.column_stack((cat_pred_test, gbr_pred_test, xgb_pred_test, lgbm_pred_test))

# Blend with optimized weights
blend_pred_test = np.dot(preds_test, optimized_weights)

# Inverse log transform
blend_pred_test = np.expm1(blend_pred_test)

# Create submission
submission = pd.DataFrame({'Id': test_id, 'SalePrice': blend_pred_test})
submission.to_csv('submission_optimized_blend.csv', index=False)
print(submission.head())
```

```
Optimized Weights: [9.65709696e-01 3.42903043e-02 1.38777878e-17 4.16333634e-17]
     Id      SalePrice
0  1461  123832.312251
1  1462  159372.365658
2  1463  186378.501981
3  1464  196665.753913
4  1465  184751.934230
```

In [7]:
```python
import shap

# Explain the best_cat model on validation data
explainer = shap.Explainer(best_cat)
shap_values = explainer(X_val_selected)

# Summary plot
shap.summary_plot(shap_values, X_val_selected, plot_type='dot', show=False)
plt.title('SHAP Feature Importance for CatBoost Model')
plt.savefig('shap_summary.png')
plt.show()
```

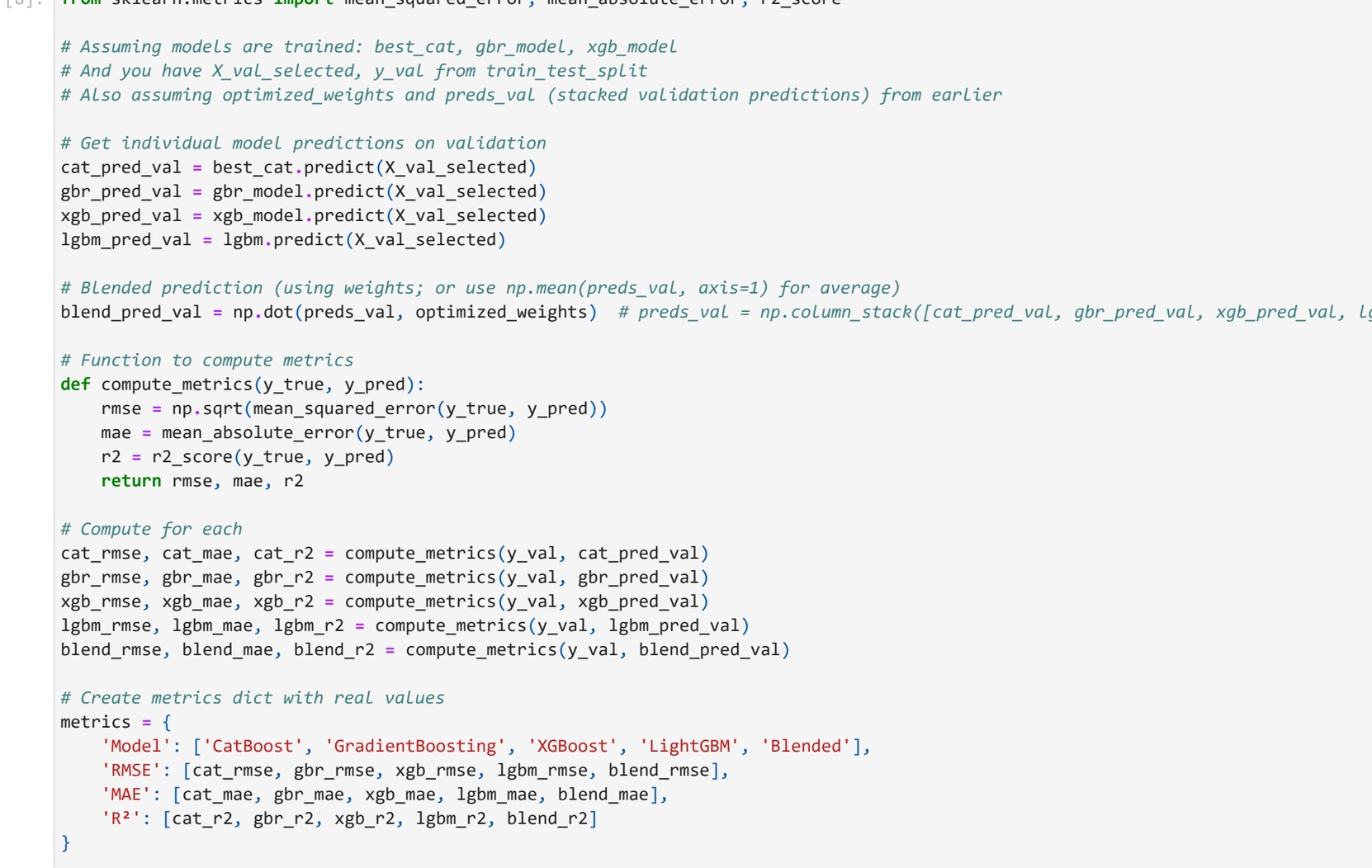


In [8]:
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Assuming models are trained: best_cat, gbr_model, xgb_model
# And you have X_val_selected, y_val from train_test_split
# Also assuming optimized_weights and preds_val (stacked validation predictions) from earlier

# Get individual model predictions on validation
cat_pred_val = best_cat.predict(X_val_selected)
gbr_pred_val = gbr_model.predict(X_val_selected)
xgb_pred_val = xgb_model.predict(X_val_selected)
lgbm_pred_val = lgbm.predict(X_val_selected)

# Blended prediction (using weights; or use np.mean(preds_val, axis=1) for average)
blend_pred_val = np.dot(preds_val, optimized_weights)  # preds_val = np.column_stack([cat_pred_val, gbr_pred_val, xgb_pred_val, lgbm_pred_val])

# Function to compute metrics
def compute_metrics(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return rmse, mae, r2

# Compute for each
cat_rmse, cat_mae, cat_r2 = compute_metrics(y_val, cat_pred_val)
gbr_rmse, gbr_mae, gbr_r2 = compute_metrics(y_val, gbr_pred_val)
xgb_rmse, xgb_mae, xgb_r2 = compute_metrics(y_val, xgb_pred_val)
lgbm_rmse, lgbm_mae, lgbm_r2 = compute_metrics(y_val, lgbm_pred_val)
blend_rmse, blend_mae, blend_r2 = compute_metrics(y_val, blend_pred_val)

# Create metrics dict with real values
metrics = {
    'Model': ['CatBoost', 'GradientBoosting', 'XGBoost', 'LightGBM', 'Blended'],
    'RMSE': [cat_rmse, gbr_rmse, xgb_rmse, lgbm_rmse, blend_rmse],
    'MAE': [cat_mae, gbr_mae, xgb_mae, lgbm_mae, blend_mae],
    'R²': [cat_r2, gbr_r2, xgb_r2, lgbm_r2, blend_r2]
}

df_metrics = pd.DataFrame(metrics)
styled = df_metrics.style.background_gradient(cmap='viridis', subset=['RMSE', 'MAE', 'R²'])
styled.to_html('model_comparison.html')  # Or display in notebook
styled
```

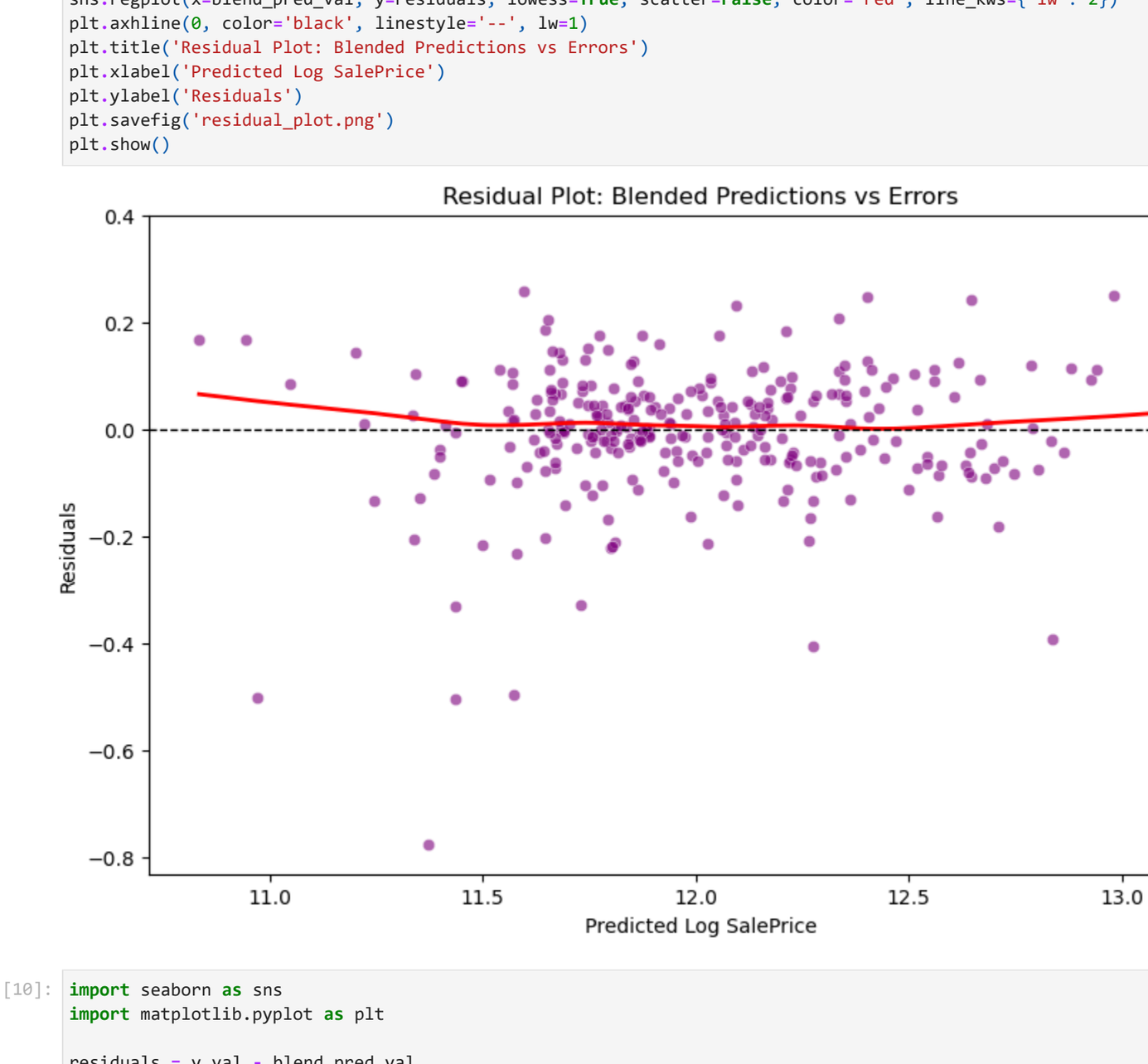
Out[8]:

	Model	RMSE	MAE	R²
0	CatBoost	0.123063	0.083248	0.918845
1	GradientBoosting	0.130496	0.088775	0.908745
2	XGBoost	0.145590	0.099197	0.886413
3	LightGBM	0.141847	0.093247	0.892180
4	Blended	0.123052	0.083166	0.918859

In [9]:
```python
# Compute blended prediction first (using weights or average)
blend_pred_val = np.dot(preds_val, optimized_weights)  # If optimized_weights from earlier
# Or simple average if no weights: blend_pred_val = np.mean(preds_val, axis=1)

# Now compute residuals
residuals = y_val - blend_pred_val

# Proceed with plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=blend_pred_val, y=residuals, alpha=0.6, color='purple')
sns.regplot(x=blend_pred_val, y=residuals, lowess=True, scatter=False, color='red', line_kws={'lw': 2})
plt.axhline(0, color='black', linestyle='--', lw=1)
plt.title('Residual Plot: Blended Predictions vs Errors')
plt.xlabel('Predicted Log SalePrice')
plt.ylabel('Residuals')
plt.savefig('residual_plot.png')
plt.show()
```



In [10]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

residuals = y_val - blend_pred_val

plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True, color='teal', bins=30)
plt.title('Distribution of Prediction Errors (Residuals)')
plt.xlabel('Residual Value')
plt.ylabel('Frequency')
plt.savefig('error_distribution.png')
plt.show()
```



In [11]:
```python
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_val, y=blend_pred_val, hue=house_data.loc[X_val_selected.index, 'OverallQual'], palette='viridis', alpha=0.7)
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--', lw=2)  # 45-degree line
plt.title('Predicted vs Actual Log SalePrice (Blended Model)')
plt.xlabel('Actual Log SalePrice')
plt.ylabel('Predicted Log SalePrice')
plt.savefig('pred_vs_actual.png')
plt.show()
```