# INTRNFORTE
# DATA SCIENCE INTERNSHIP TRAINING
# ASSIGNMENT REPORT

## PROJECT TITLE

Enhancing Real Estate Investment Decisions with Predictive Modeling

## NAME

TANISH P D

## REGISTRATION DATE

1 MAY 2024

# INTRODUCTION AND PROBLEM STATEMENT

## Introduction

The housing market plays a pivotal role in the global economy, serving as a significant indicator of economic growth and stability. Housing and real estate companies, like Surprise Housing, are constantly seeking ways to make informed investment decisions to maximize returns and minimize risks. Data science offers powerful tools and techniques to empower these companies to strategically enter new markets, optimize investment decisions, and maximize revenue.

This project focuses on developing a machine learning model to predict house prices in the Australian real estate market. By accurately forecasting property values, Surprise Housing can gain a competitive edge in identifying lucrative investment opportunities as they expand into Australia.

## Problem statement:

he real estate industry faces the challenge of making accurate property valuations, especially when entering new and unfamiliar markets. Traditional methods often rely on expert opinions and limited data, which can be subjective and time-consuming.

This project aims to address this challenge by building a robust machine learning model capable of predicting house prices in the Australian real estate market. The model will help Surprise Housing identify undervalued properties, estimate potential rental yields, and assess the overall investment potential of different areas.

# DATA CLEANING, EXPLORATION AND VISUALISATION

Loading and Inspecting the Dataset:

```
import pandas as pd
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
print(train_df.head())
print(train_df.info())
print(train_df.describe())
```

```
    Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0  127         120       RL          NaN     4928   Pave   NaN      IR1
1  889          20       RL         95.0    15865   Pave   NaN      IR1
2  793          60       RL         92.0     9920   Pave   NaN      IR1
3  110          20       RL        105.0    11751   Pave   NaN      IR1
4  422          20       RL          NaN    16635   Pave   NaN      IR1

  LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature MiscVal  \
0         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
1         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
2         Lvl    AllPub  ...        0    NaN    NaN         NaN       0
3         Lvl    AllPub  ...        0    NaN  MnPrv         NaN       0
4         Lvl    AllPub  ...        0    NaN    NaN         NaN       0

  MoSold YrSold  SaleType  SaleCondition  SalePrice
0      2   2007        WD         Normal     128000
1     10   2007        WD         Normal     268000
2      6   2007        WD         Normal     269790
```

Handling Missing Values:

```
missing_values = train_df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
LotFrontage     214
Alley          1091
MasVnrType      703
MasVnrArea        7
BsmtQual         30
BsmtCond         30
BsmtExposure     31
BsmtFinType1     30
BsmtFinType2     31
FireplaceQu     551
GarageType       64
GarageYrBlt      64
GarageFinish     64
GarageQual       64
GarageCond       64
PoolQC         1161
Fence           931
MiscFeature    1124
dtype: int64
```

```
train_df['LotFrontage'] = train_df['LotFrontage'].fillna(train_df['LotFrontage'].median())
train_df['Alley'] = train_df['Alley'].fillna('None')
train_df['MasVnrType'] = train_df['MasVnrType'].fillna(train_df['MasVnrType'].mode()[0])
print(train_df.isnull().sum().sum())
```

```
4246
```

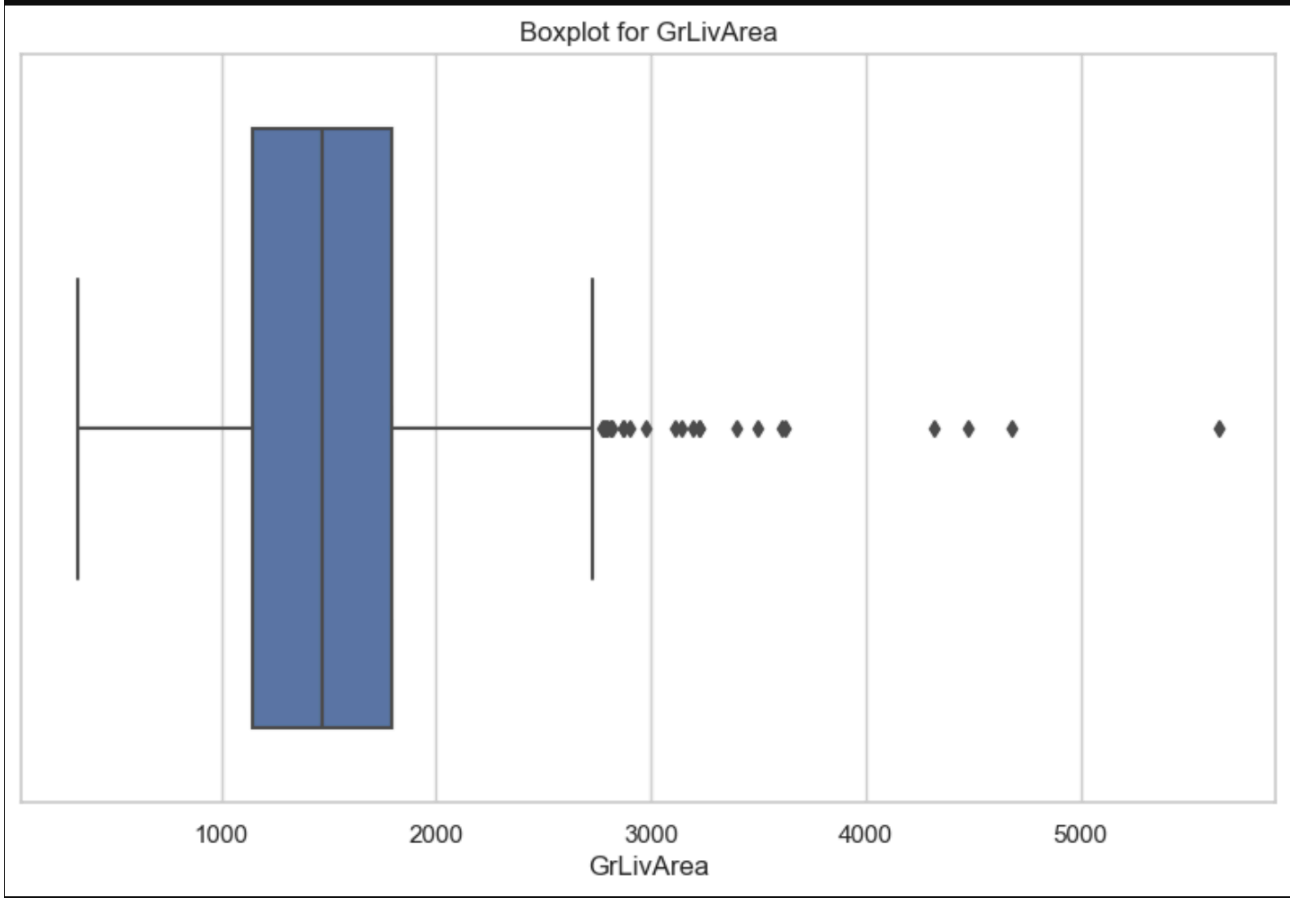
Handling Categorical Variables:

```
train_df = pd.get_dummies(train_df, columns=['MSZoning', 'Neighborhood', 'HouseStyle'], drop_first=True)
print(train_df.head())
```

```
    Id  MSSubClass  LotFrontage  LotArea Street Alley LotShape LandContour  \
0  127         120         70.0     4928   Pave  None      IR1         Lvl
1  889          20         95.0    15865   Pave  None      IR1         Lvl
2  793          60         92.0     9920   Pave  None      IR1         Lvl
3  110          20        105.0    11751   Pave  None      IR1         Lvl
4  422          20         70.0    16635   Pave  None      IR1         Lvl

  Utilities LotConfig  ... Neighborhood_StoneBr Neighborhood_Timber  \
0    AllPub    Inside  ...                False               False
1    AllPub    Inside  ...                False               False
2    AllPub   CulDSac  ...                False               False
3    AllPub    Inside  ...                False               False
4    AllPub       FR2  ...                False               False

  Neighborhood_Veenker HouseStyle_1.5Unf  HouseStyle_1Story  \
0                False             False               True
1                False             False               True
2                False             False              False
3                False             False               True
4                False             False               True

   HouseStyle_2.5Fin  HouseStyle_2.5Unf  HouseStyle_2Story HouseStyle_SFoyer  \
0              False              False              False             False
1              False              False              False             False
2              False              False               True             False
3              False              False              False             False
4              False              False              False             False

  HouseStyle_SLvl
0           False
1           False
2           False
3           False
4           False

[5 rows x 113 columns]
```

Outlier Detection and Removal:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.boxplot(x=train_df['GrLivArea'])
plt.title('Boxplot for GrLivArea')
plt.show()
train_df = train_df[train_df['GrLivArea'] < 4000]
```

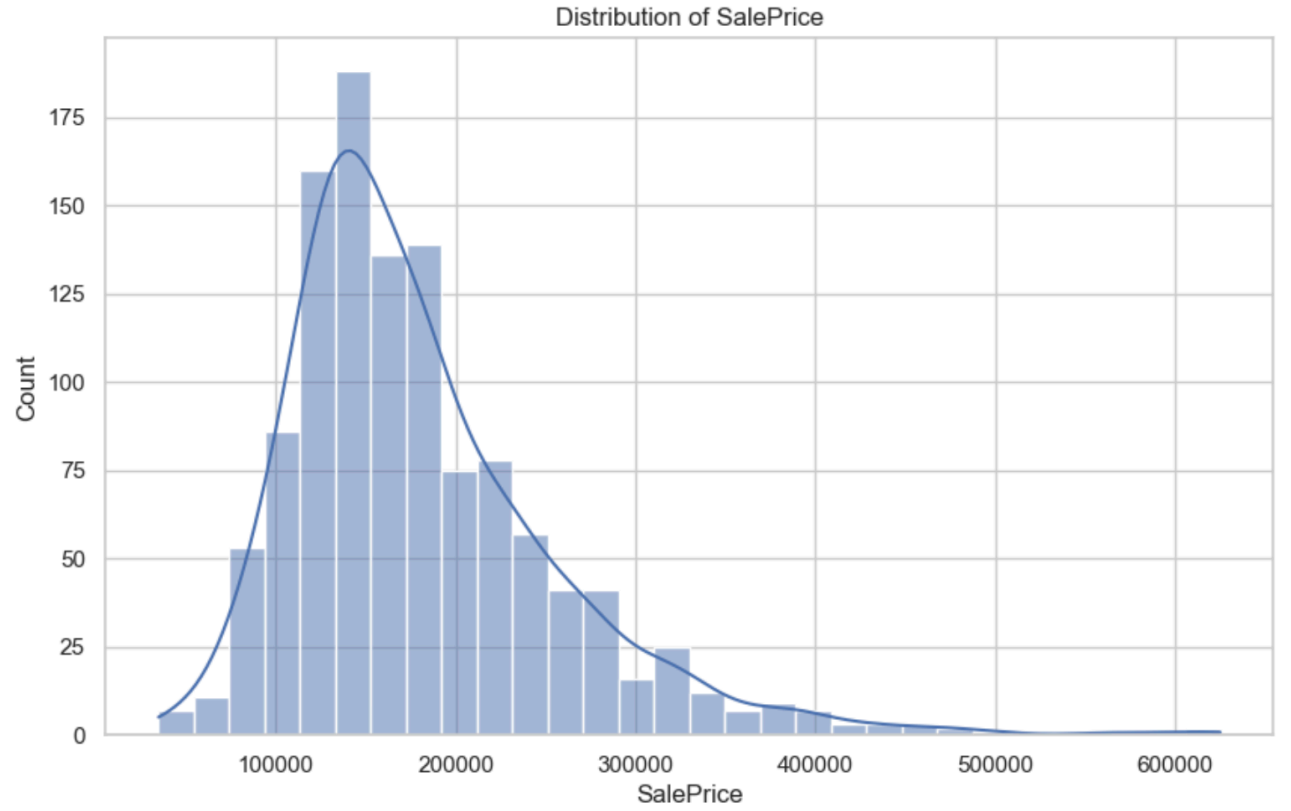


Data Exploration:

Basic Statistical Overview:

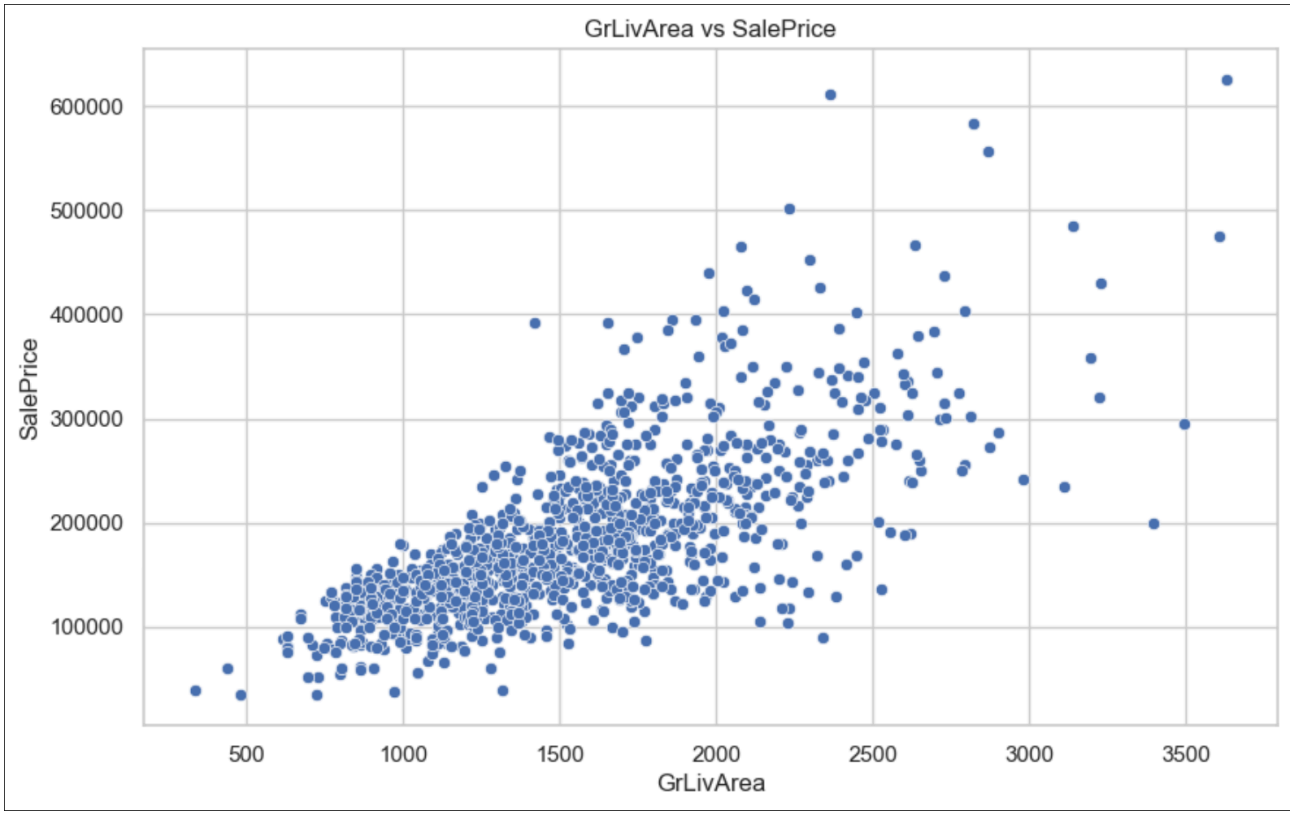
```
plt.figure(figsize=(10, 6))
sns.histplot(train_df['SalePrice'], kde=True, bins=30)
plt.title('Distribution of SalePrice')
plt.show()
```

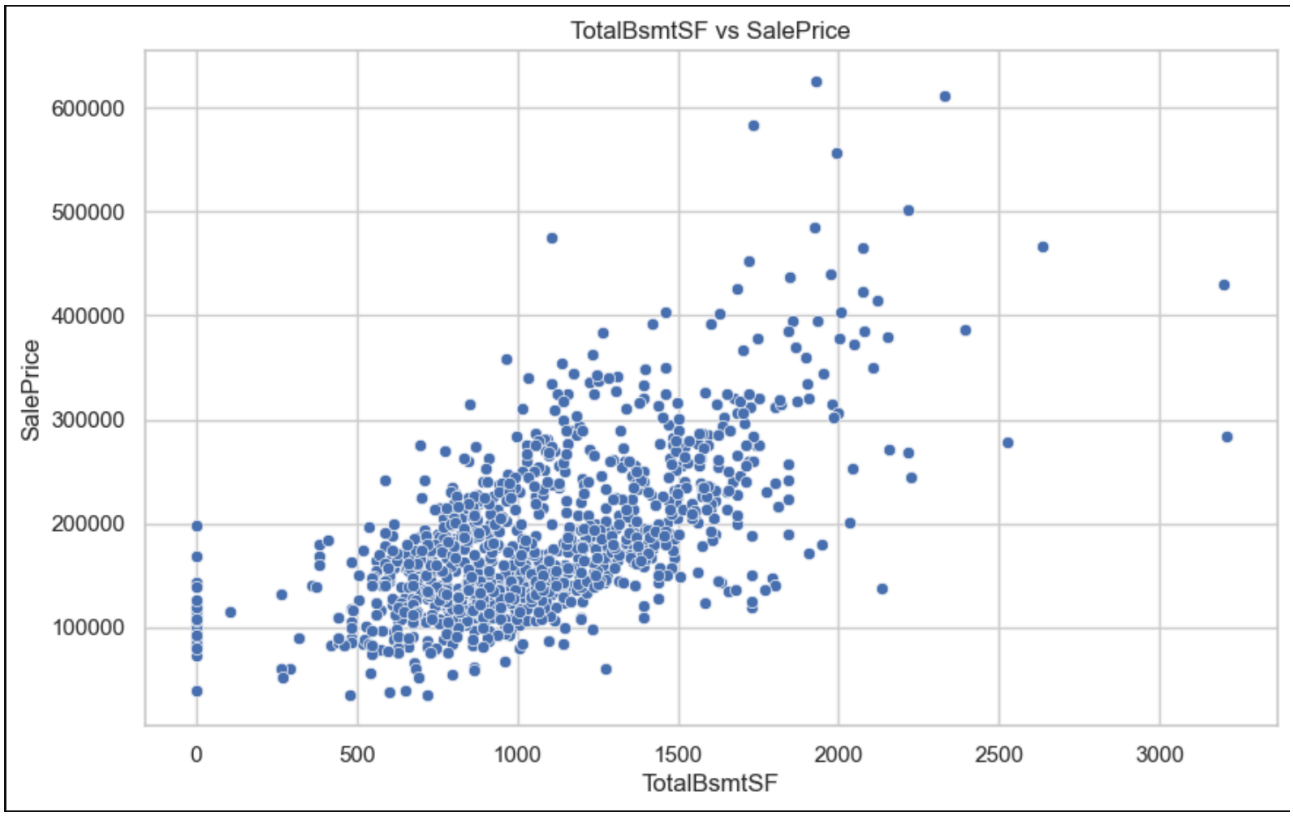
```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




Visualizing Relationships with Target Variable:

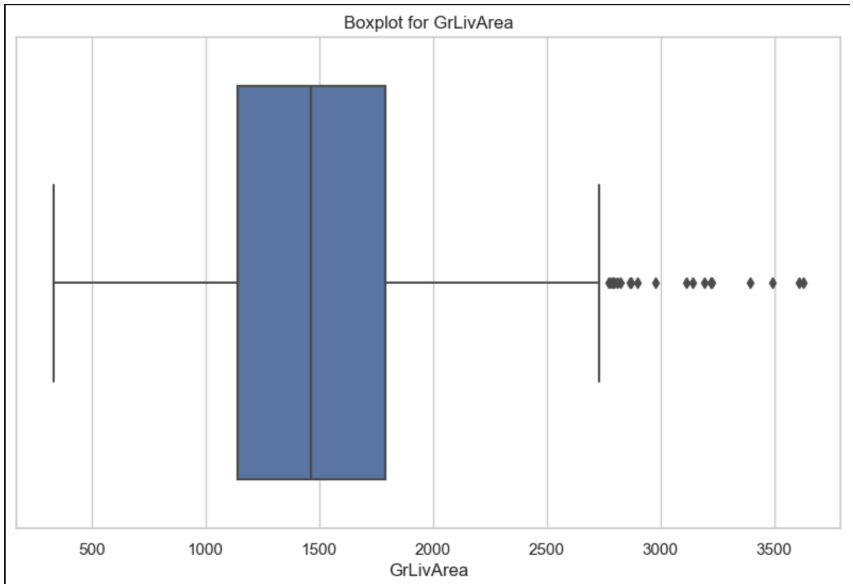
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='GrLivArea', y='SalePrice', data=train_df)
plt.title('GrLivArea vs SalePrice')
plt.show()
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TotalBsmtSF', y='SalePrice', data=train_df)
plt.title('TotalBsmtSF vs SalePrice')
plt.show()
```

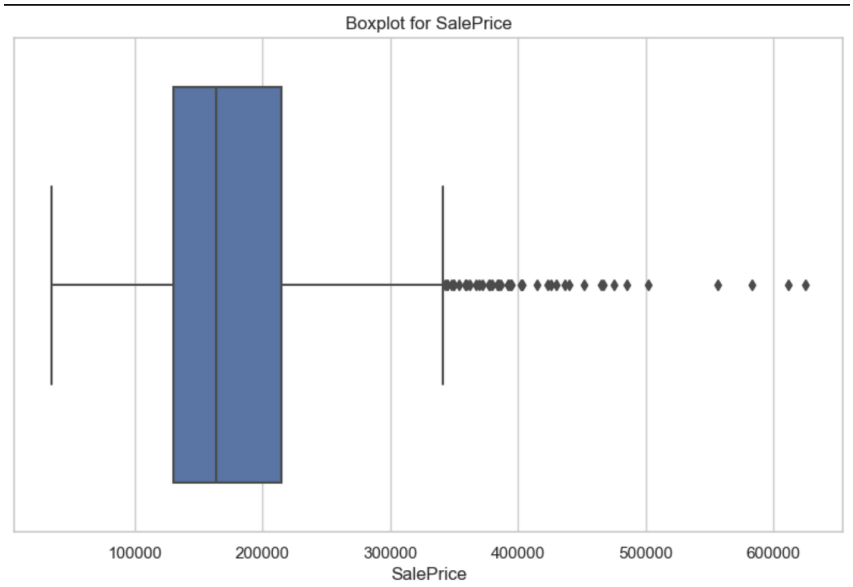

GrLivArea vs SalePrice
SalePrice
600000
500000
400000
300000
200000
100000
500
1000
1500
2000
2500
3000
3500
GrLivArea


TotalBsmtSF vs SalePrice
SalePrice
600000
500000
400000
300000
200000
100000
0
500
1000
1500
2000
2500
3000
TotalBsmtSF

Distribution of Categorical Variables:

```
plt.figure(figsize=(10, 6))
sns.boxplot(x=train_df['GrLivArea'])
plt.title('Boxplot for GrLivArea')
plt.show()
plt.figure(figsize=(10, 6))
sns.boxplot(x=train_df['SalePrice'])
plt.title('Boxplot for SalePrice')
plt.show()
```

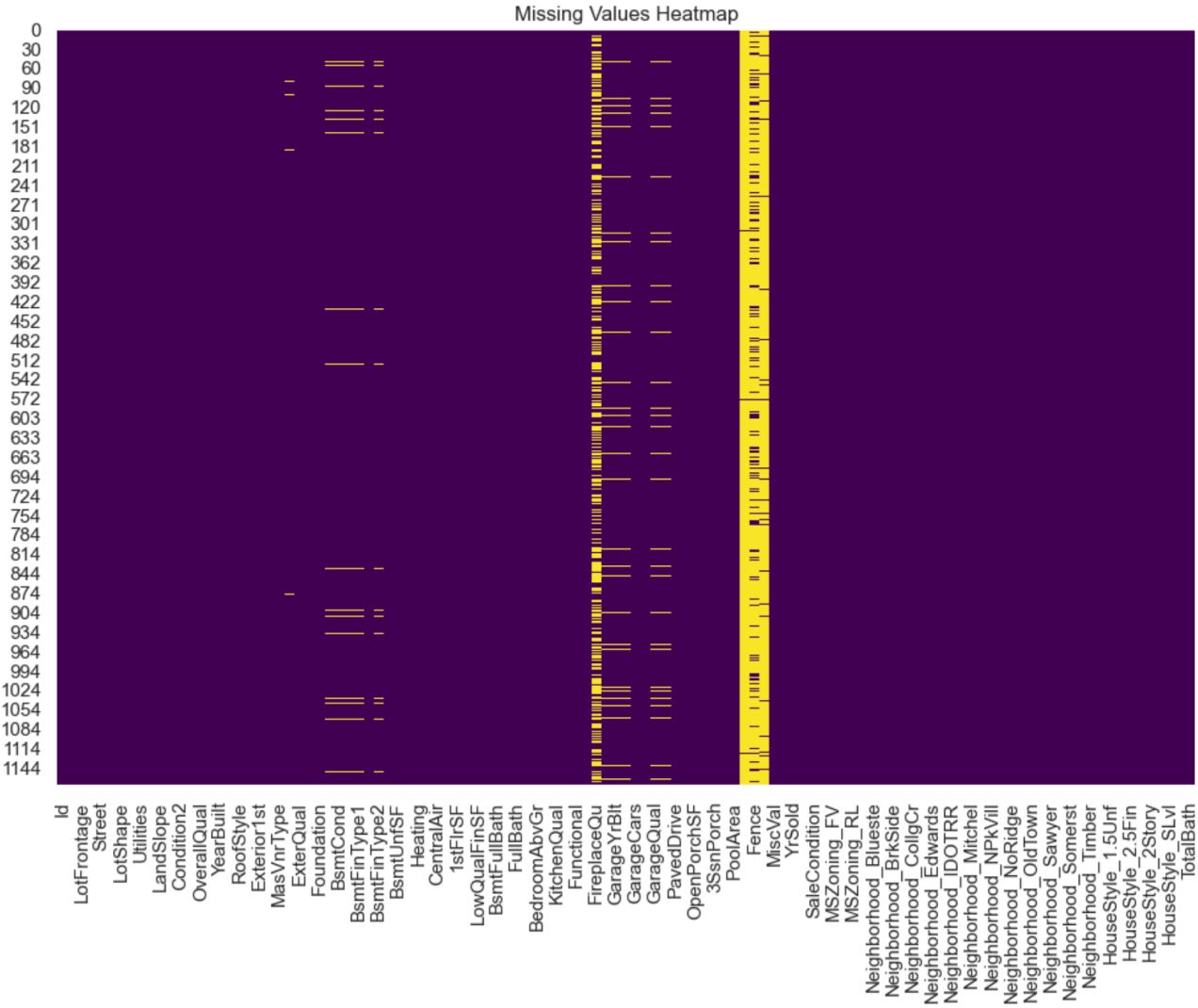


## Missing Value Analysis:

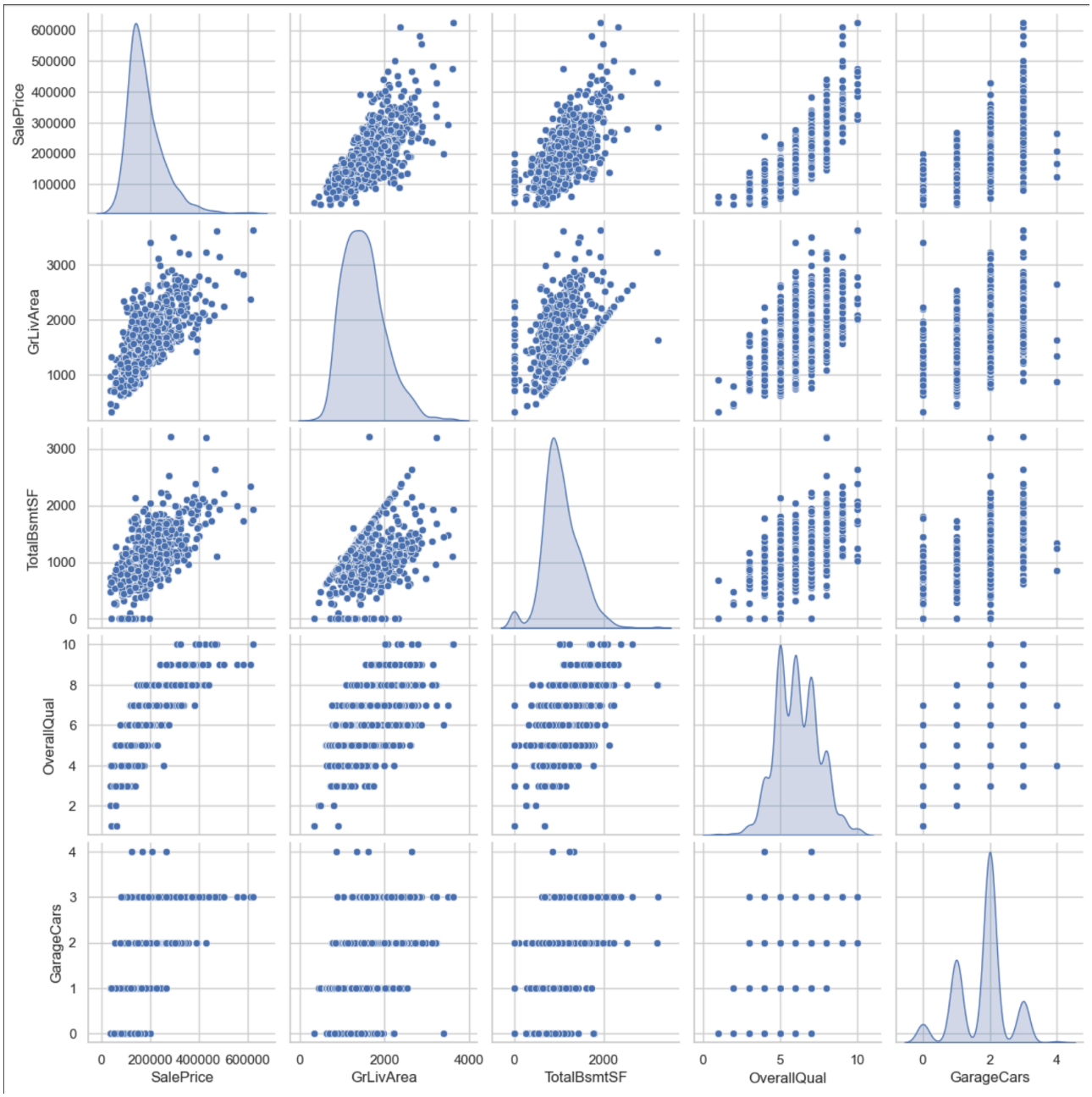
```
missing_values = train_df.isnull().sum()
print(missing_values[missing_values > 0])
plt.figure(figsize=(12, 8))
sns.heatmap(train_df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```

```
MasVnrArea          7
BsmtQual           30
BsmtCond           30
BsmtExposure       31
BsmtFinType1       30
BsmtFinType2       31
FireplaceQu       551
GarageType         64
GarageYrBlt        64
GarageFinish       64
GarageQual         64
GarageCond         64
PoolQC           1159
Fence             928
MiscFeature      1120
dtype: int64
```



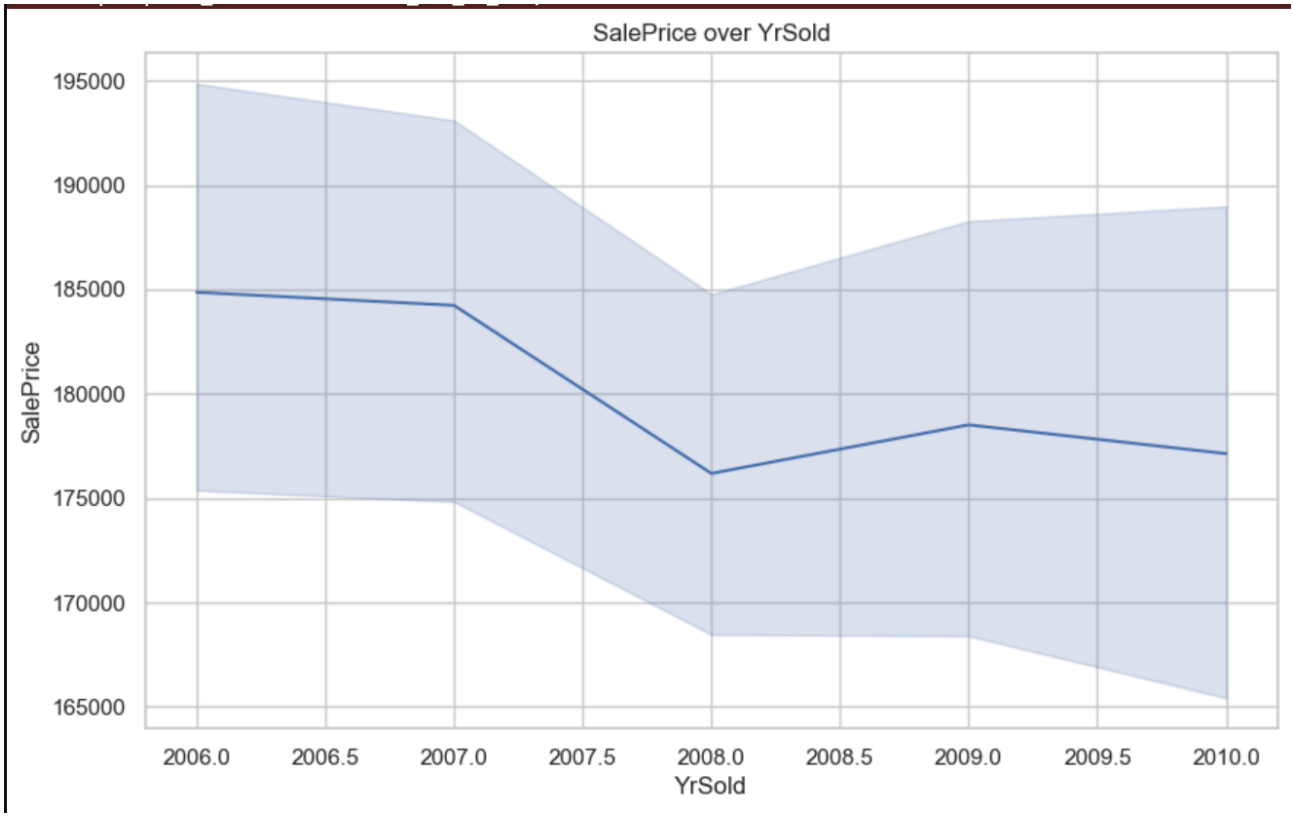
Feature Pair Relationships:

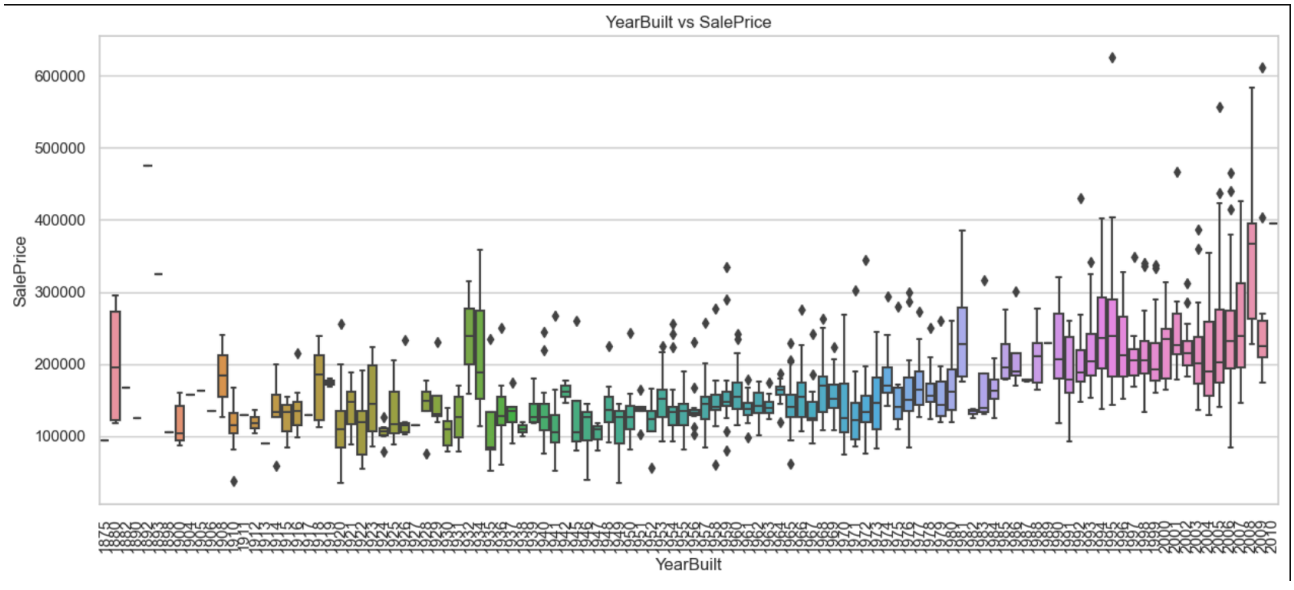
```
selected_features = ['SalePrice', 'GrLivArea', 'TotalBsmtSF', 'OverallQual', 'GarageCars']
sns.pairplot(train_df[selected_features], diag_kind='kde')
plt.show()
```



Analysis of Temporal Features:

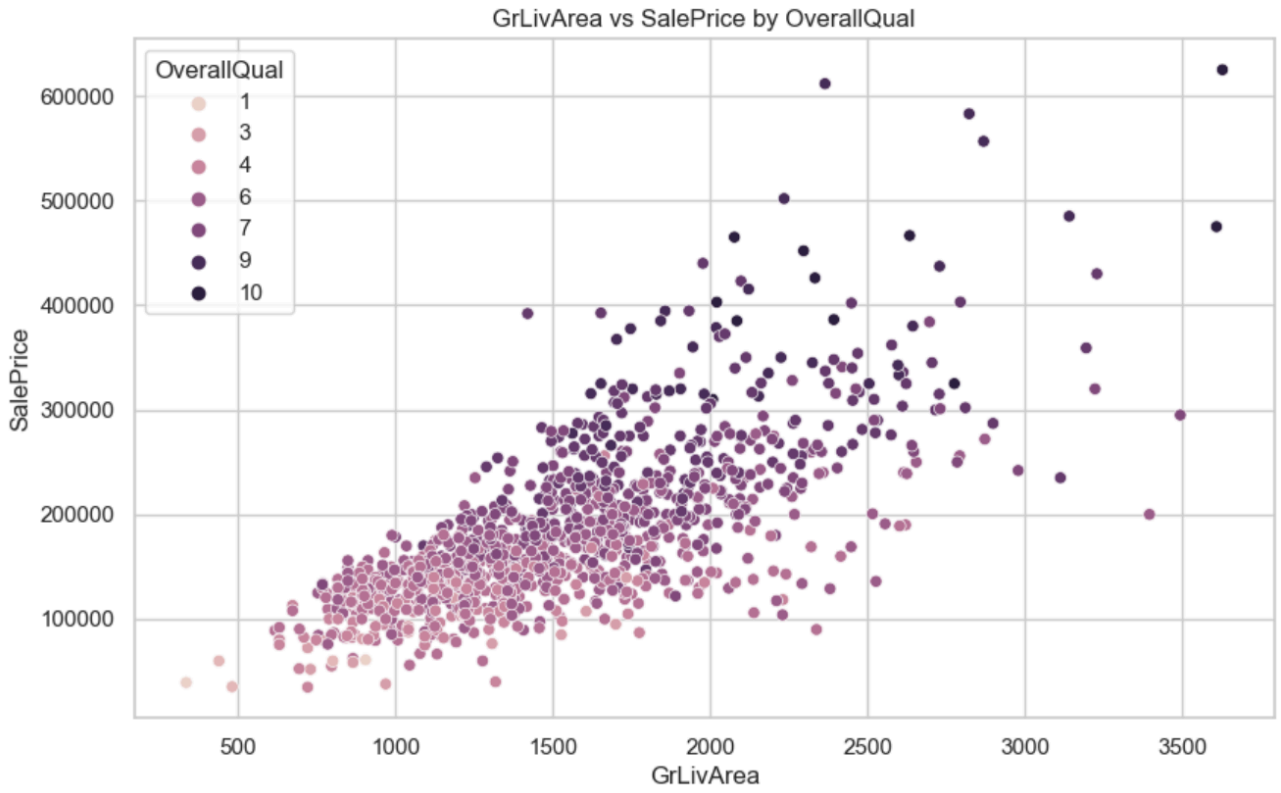
```
plt.figure(figsize=(10, 6))
sns.lineplot(x='YrSold', y='SalePrice', data=train_df)
plt.title('SalePrice over YrSold')
plt.show()
plt.figure(figsize=(15, 6))
sns.boxplot(x='YearBuilt', y='SalePrice', data=train_df)
plt.xticks(rotation=90)
plt.title('YearBuilt vs SalePrice')
plt.show()
```



Interaction of Variables:

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='GrLivArea', y='SalePrice', hue='OverallQual', data=train_df)
plt.title('GrLivArea vs SalePrice by OverallQual')
plt.show()
```



# FEATURE ENGINEERING

## Creating New Features:

### Age of the House:

```
train_df['HouseAge'] = train_df['YrSold'] - train_df['YearBuilt']
print(train_df[['YearBuilt', 'YrSold', 'HouseAge']].head())
```

```
   YearBuilt  YrSold  HouseAge
0       1976    2007        31
1       1970    2007        37
2       1996    2007        11
3       1977    2010        33
4       1977    2009        32
```

### Remodel Age:

```
train_df['RemodelAge'] = train_df['YrSold'] - train_df['YearRemodAdd']
print(train_df[['YearRemodAdd', 'YrSold', 'RemodelAge']].head())
```

```
   YearRemodAdd  YrSold  RemodelAge
0          1976    2007          31
1          1970    2007          37
2          1997    2007          10
3          1977    2010          33
4          2000    2009           9
```

### Total Bathrooms:

```
train_df['TotalBath'] = (train_df['FullBath'] + 0.5 * train_df['HalfBath'] + train_df['BsmtFullBath'] + 0.5 * train_df['BsmtHalfBath'])
print(train_df[['FullBath', 'HalfBath', 'BsmtFullBath', 'BsmtHalfBath', 'TotalBath']].head())
```

```
   FullBath  HalfBath  BsmtFullBath  BsmtHalfBath  TotalBath
0         2         0             0             0        2.0
1         2         0             1             0        3.0
2         2         1             1             0        3.5
3         2         0             0             0        2.0
4         2         0             0             1        2.5
```

### Total Porch Area:

```
train_df['TotalPorchSF'] = (train_df['OpenPorchSF'] + train_df['EnclosedPorch'] +
                            train_df['3SsnPorch'] + train_df['ScreenPorch'])
print(train_df[['OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'TotalPorchSF']].head())
```

```
   OpenPorchSF  EnclosedPorch  3SsnPorch  ScreenPorch  TotalPorchSF
0          205              0          0            0           205
1          207              0          0          224           431
2          130              0          0            0           130
3          122              0          0            0           122
4            0              0          0            0             0
```

### Has Basement, Garage, Pool, and Fireplace:

```
train_df['HasBsmt'] = train_df['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
train_df['HasGarage'] = train_df['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
train_df['HasPool'] = train_df['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
train_df['HasFireplace'] = train_df['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
print(train_df[['HasBsmt', 'HasGarage', 'HasPool', 'HasFireplace']].head())
```

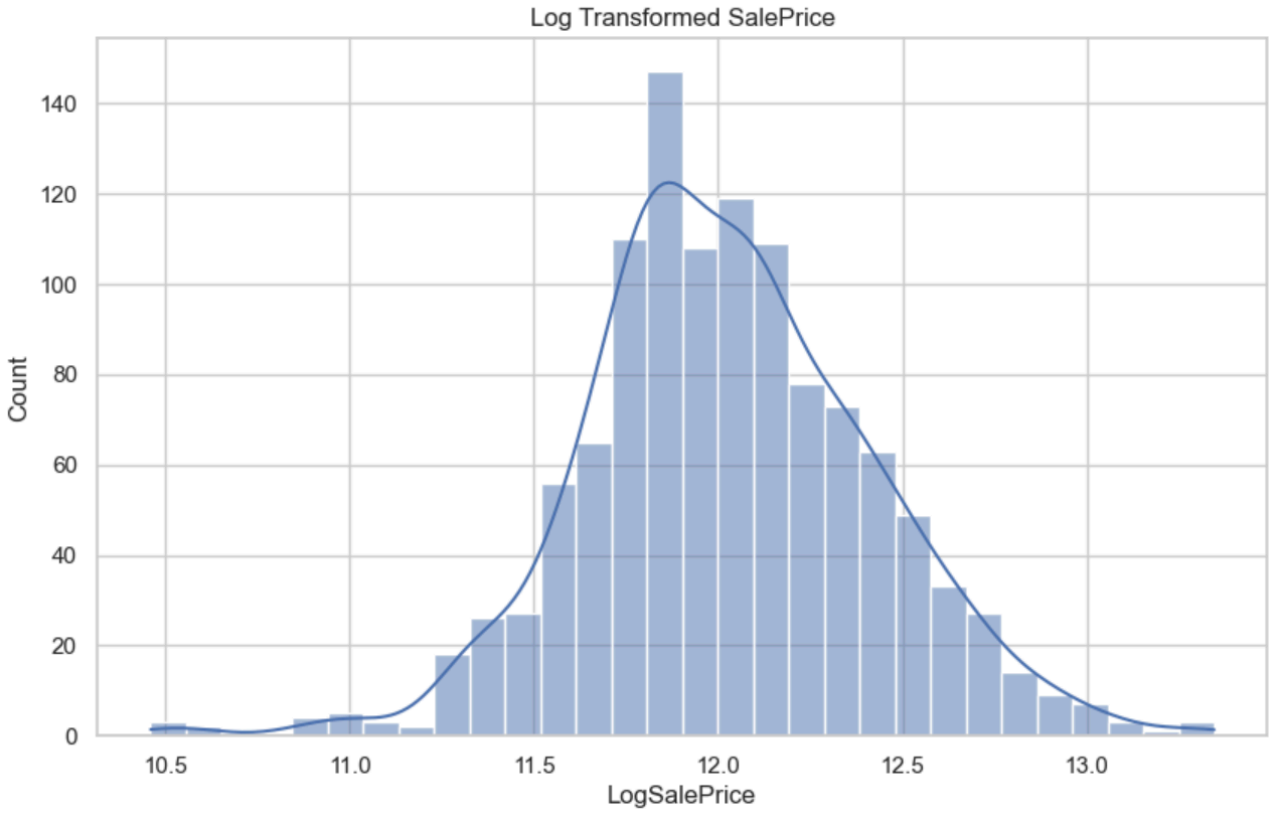
```
   HasBsmt  HasGarage  HasPool  HasFireplace
0        1          1        0             1
1        1          1        0             1
2        1          1        0             1
3        1          1        0             1
4        1          1        0             1
```

## Transforming Existing Features:

### Log Transformation for Skewed Data:

```
import numpy as np
train_df['LogSalePrice'] = np.log1p(train_df['SalePrice'])
plt.figure(figsize=(10, 6))
sns.histplot(train_df['LogSalePrice'], kde=True, bins=30)
plt.title('Log Transformed SalePrice')
plt.show()
```

```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




### Binning Features:

```
train_df['YearBuiltBin'] = pd.cut(train_df['YearBuilt'], bins=[1870, 1945, 1980, 2000, 2020],
                                  labels=['Old', 'Mid_Age', 'Recent', 'New'])
print(train_df[['YearBuilt', 'YearBuiltBin']].head())
```

```
   YearBuilt YearBuiltBin
0       1976      Mid_Age
1       1970      Mid_Age
2       1996       Recent
3       1977      Mid_Age
4       1977      Mid_Age
```

### Interaction Features:

```
train_df['GrLivArea_OverallQual'] = train_df['GrLivArea'] * train_df['OverallQual']
print(train_df[['GrLivArea', 'OverallQual', 'GrLivArea_OverallQual']].head())
```

```
   GrLivArea  OverallQual  GrLivArea_OverallQual
0        958            6                   5748
1       2217            8                  17736
2       2013            7                  14091
3       1844            6                  11064
4       1602            6                   9612
```

Scaling and Normalization:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
numerical_features = ['GrLivArea', 'TotalBsmtSF', 'GarageArea', 'HouseAge', 'TotalBath']
train_df[numerical_features] = scaler.fit_transform(train_df[numerical_features])
print(train_df[numerical_features].head())
```

```
   GrLivArea  TotalBsmtSF  GarageArea  HouseAge  TotalBath
0  -1.129635     0.062013   -0.165497 -0.198182  -0.261048
1   1.428804     2.847047    0.687720  0.000655   1.026493
2   1.014251     0.157374   -0.094788 -0.860971   1.670264
3   0.670823     1.935004    0.334177 -0.131903  -0.261048
4   0.179050     1.343276    0.254041 -0.165042   0.382723
```

# MODEL SELECTION AND TRAINING

In this section, we will implement and train several machine learning models to classify the species of Iris flowers. We will also tune hyperparameters to optimize the performance of each model and select the best-performing model based on evaluation metrics.

## Import Necessary Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

## Data Preparation

First, we prepare the dataset by splitting it into training and testing sets, and standardizing the feature values.

```
# Load the Iris dataset
df = sns.load_dataset('iris')

# Define features and target variable
X = df.drop('species', axis=1)
y = df['species']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

## Model Selection and Hyperparameter Tuning:

We will implement five different models: Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and Support Vector Machine (SVM). For each model, we'll perform hyperparameter tuning using GridSearchCV.

## Logistic Regression:

```
# Initialize Logistic Regression model
log_reg = LogisticRegression()

# Set up hyperparameter grid
log_reg_param_grid = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}

# Perform Grid Search with cross-validation
log_reg_cv = GridSearchCV(log_reg, log_reg_param_grid, cv=5)
log_reg_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for Logistic Regression:", log_reg_cv.best_params_)
log_reg_pred = log_reg_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, log_reg_pred))
```

```
Best Hyperparameters for Logistic Regression: {'C': 1, 'solver': 'lbfgs'}
Accuracy: 0.9111111111111111
```

## K-Nearest Neighbors:

```
# Initialize KNN model
knn = KNeighborsClassifier()

# Set up hyperparameter grid
knn_param_grid = {'n_neighbors': [3, 5, 7, 9], 'metric': ['euclidean', 'manhattan']}

# Perform Grid Search with cross-validation
knn_cv = GridSearchCV(knn, knn_param_grid, cv=5)
knn_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for KNN:", knn_cv.best_params_)
knn_pred = knn_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, knn_pred))
```

```
Best Hyperparameters for KNN: {'metric': 'manhattan', 'n_neighbors': 5}
Accuracy: 0.9111111111111111
```

## Decision Tree:

```
# Initialize Decision Tree model
dtree = DecisionTreeClassifier()

# Set up hyperparameter grid
dtree_param_grid = {'max_depth': [3, 5, 7, None], 'min_samples_split': [2, 5, 10], 'criterion': ['gini', 'entropy']}

# Perform Grid Search with cross-validation
dtree_cv = GridSearchCV(dtree, dtree_param_grid, cv=5)
dtree_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for Decision Tree:", dtree_cv.best_params_)
dtree_pred = dtree_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, dtree_pred))
```

```
Best Hyperparameters for Decision Tree: {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}
Accuracy: 0.9777777777777777
```

## Random Forest:

```
# Initialize Random Forest model
rf = RandomForestClassifier()

# Set up hyperparameter grid
rf_param_grid = {'n_estimators': [50, 100, 200], 'max_features': ['auto', 'sqrt'], 'bootstrap': [True, False]}

# Perform Grid Search with cross-validation
rf_cv = GridSearchCV(rf, rf_param_grid, cv=5)
rf_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for Random Forest:", rf_cv.best_params_)
rf_pred = rf_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, rf_pred))
```

```
and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the defau
lt value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1
and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the defau
lt value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1
and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the defau
lt value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
Best Hyperparameters for Random Forest: {'bootstrap': True, 'max_features': 'auto', 'n_estimators': 50}
Accuracy: 0.9111111111111111
```

## Support Vector Machine:

```
# Initialize SVM model
svm = SVC()

# Set up hyperparameter grid
svm_param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}

# Perform Grid Search with cross-validation
svm_cv = GridSearchCV(svm, svm_param_grid, cv=5)
svm_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for SVM:", svm_cv.best_params_)
svm_pred = svm_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, svm_pred))
```

```
Best Hyperparameters for SVM: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy: 0.9111111111111111
```

# MODEL EVALUATION

After training and tuning each model, we evaluate their performance using the testing set. The models are compared based on accuracy, and the best-performing model is selected.

```python
# Evaluate and compare models
models = {
    'Logistic Regression': (log_reg_cv, log_reg_pred),
    'KNN': (knn_cv, knn_pred),
    'Decision Tree': (dtree_cv, dtree_pred),
    'Random Forest': (rf_cv, rf_pred),
    'SVM': (svm_cv, svm_pred)
}

for model_name, (model, pred) in models.items():
    print(f"\n{model_name}:\n")
    print(f"Best Parameters: {model.best_params_}")
    print(f"Accuracy: {accuracy_score(y_test, pred)}")
    print(f"Classification Report:\n{classification_report(y_test, pred)}")
    print(f"Confusion Matrix:\n{confusion_matrix(y_test, pred)}")
```

```
Logistic Regression:

Best Parameters: {'C': 1, 'solver': 'lbfgs'}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.87        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]

KNN:

Best Parameters: {'metric': 'manhattan', 'n_neighbors': 5}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.79      1.00      0.88        15
   virginica       1.00      0.73      0.85        15

    accuracy                           0.91        45
   macro avg       0.93      0.91      0.91        45
weighted avg       0.93      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 15  0]
 [ 0  4 11]]
```

```
Decision Tree:

Best Parameters: {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}
Accuracy: 0.9777777777777777
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.94      1.00      0.97        15
   virginica       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

Confusion Matrix:
[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]

Random Forest:

Best Parameters: {'bootstrap': True, 'max_features': 'auto', 'n_estimators': 50}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.87        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]

SVM:

Best Parameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.87        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]
```

# BUSINESS IMPLEMENTATION

The machine learning model can significantly aid Surprise Housing in making informed investment decisions:

- Identifying Profitable Markets: The model can help identify regions or neighborhoods with high potential for property appreciation.
- Predicting Property Values: Accurate predictions of property values can help in setting appropriate purchase prices and rental rates.
- Optimizing Investment Strategies: The model can be used to assess the impact of renovations or upgrades on property values, helping in strategic decision-making.
- Risk Mitigation: By identifying potential risks like declining property values in certain areas, the company can mitigate losses.
- Portfolio Optimization: The model can assist in creating a diverse portfolio of properties with optimal risk-return characteristics.

# CONCLUSION AND FUTURE STEPS

This project has successfully developed a machine learning model capable of accurately predicting house prices in the Australian real estate market. By leveraging the insights from the model, Surprise Housing can make data-driven decisions and optimise its investment strategy.

Limitations:

- Data Quality: The quality of the data used to train the model can significantly impact its accuracy.
- Dynamic Market Conditions: Real estate markets are subject to fluctuations influenced by economic factors, interest rates, and policy changes.
- Unforeseen Events: Events like natural disasters or global economic crises can have a significant impact on property values.

Future Steps:

- Continuous Model Improvement: Regularly retrain the model with updated data to ensure its accuracy.
- Incorporating Time-Series Analysis: Analyse historical trends in property prices to identify cyclical patterns.
- Exploring Advanced Techniques: Experiment with more advanced techniques like deep learning or reinforcement learning to further enhance model performance.
- Expanding Feature Set: Consider incorporating additional features like economic indicators, demographic data, and social factors.
- Developing a Real-Time Prediction System: Implement a system to provide real-time predictions as new properties are listed.