Task Report: Predicting House Prices in Ames, Iowa

Introduction

This project focuses on predicting house prices based on a dataset that contains 79 features describing various characteristics of residential homes in Ames, Iowa. The objective of the competition is to predict the final sale price of houses using machine learning techniques. The dataset provides an excellent opportunity to practice feature engineering, handle missing data, and apply advanced regression techniques.

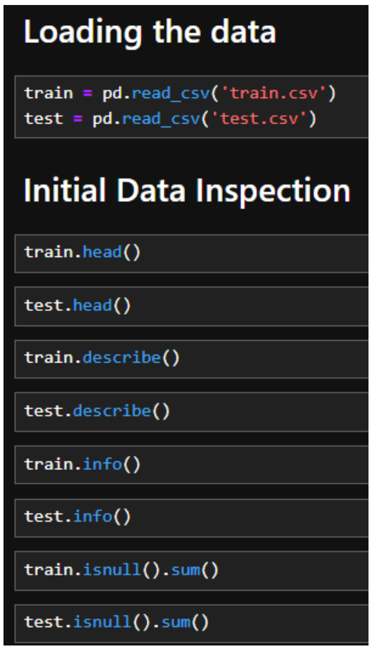
Problem Statement

The task is to build a regression model that predicts the sale price of houses using a variety of features. The model will be evaluated using Root-Mean-Squared-Error (RMSE) between the logarithms of the predicted and actual sale prices, ensuring that errors in predicting expensive and inexpensive houses affect the result equally.

Solution Approach

1. Data Exploration and Visualization

We started by loading the dataset and performing an initial inspection to understand its structure.



Loading the data

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

Initial Data Inspection

```
train.head()
```

```
test.head()
```

```
train.describe()
```

```
test.describe()
```

```
train.info()
```

```
test.info()
```

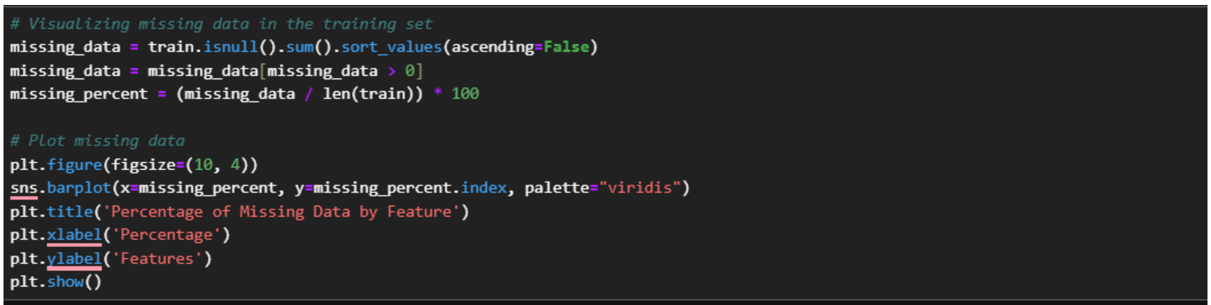
```
train.isnull().sum()
```

```
test.isnull().sum()
```

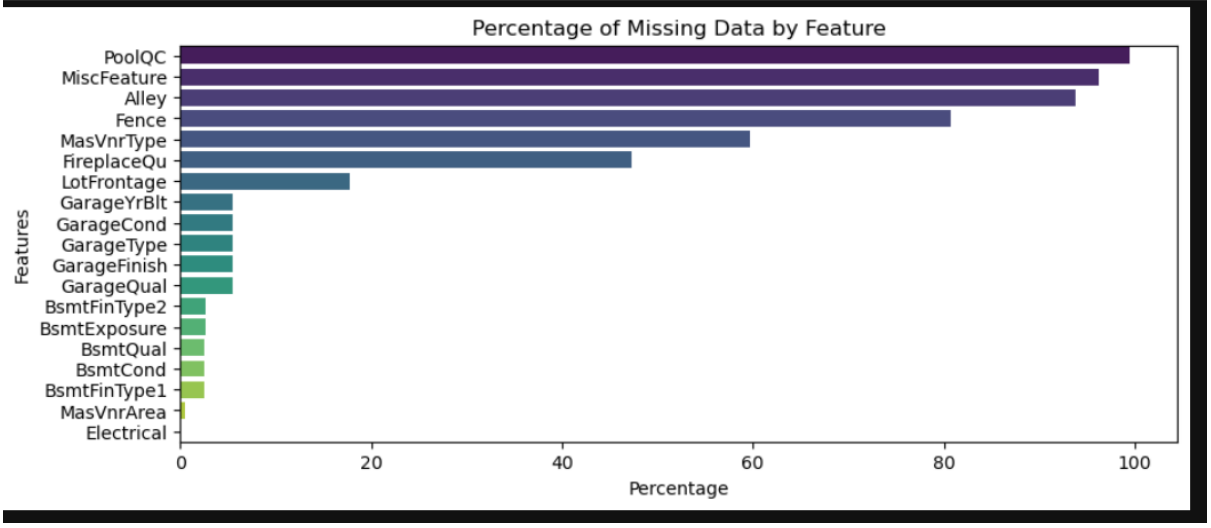
This code gives a detailed look at the dataset, providing information about the columns, data types, and missing values. The describe() method offers statistical insights into the numerical columns, while info() helps identify missing data and the types of each column.

2. Missing Data Analysis

Missing data can significantly affect model performance, so we visualized the missing data and calculated the percentage of missing values for each feature.



```
# Visualizing missing data in the training set
missing_data = train.isnull().sum().sort_values(ascending=False)
missing_data = missing_data[missing_data > 0]
missing_percent = (missing_data / len(train)) * 100

# Plot missing data
plt.figure(figsize=(10, 4))
sns.barplot(x=missing_percent, y=missing_percent.index, palette="viridis")
plt.title('Percentage of Missing Data by Feature')
plt.xlabel('Percentage')
plt.ylabel('Features')
plt.show()
```

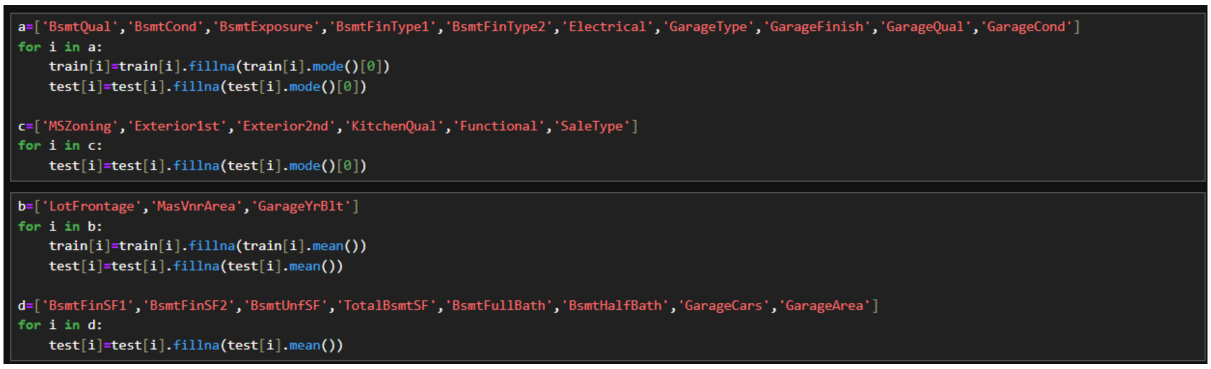


The bar plot allowed us to understand which features have missing data. Features with significant missing values, such as Alley, PoolQC, Fence, and others, were dropped because they had too many missing values to be useful in modelling.

3. Handling Missing Data

We handled missing data using several strategies depending on the nature of the feature:

- **Categorical Features**: We filled missing values with the most frequent category (mode) for categorical columns.
- **Numerical Features**: For numerical features like LotFrontage, GarageYrBlt, and MasVnrArea, we filled missing values with the mean of the column.



```python
a=['BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','Electrical','GarageType','GarageFinish','GarageQual','GarageCond']
for i in a:
    train[i]=train[i].fillna(train[i].mode()[0])
    test[i]=test[i].fillna(test[i].mode()[0])

c=['MSZoning','Exterior1st','Exterior2nd','KitchenQual','Functional','SaleType']
for i in c:
    test[i]=test[i].fillna(test[i].mode()[0])
```

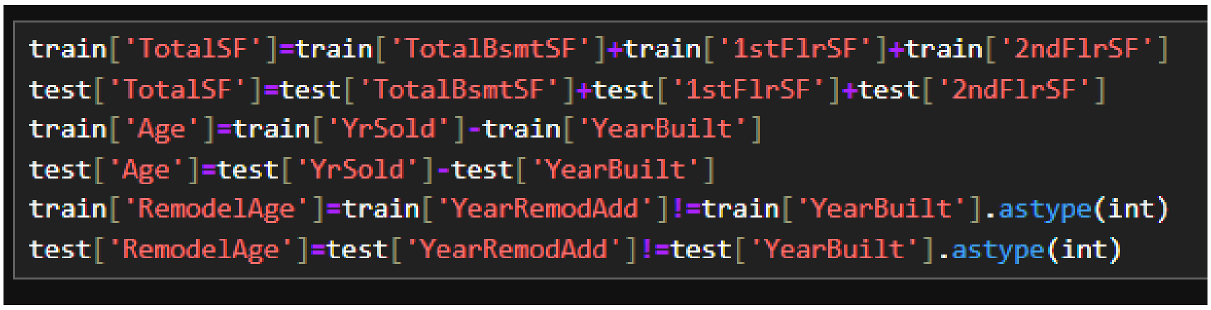
```python
b=['LotFrontage','MasVnrArea','GarageYrBlt']
for i in b:
    train[i]=train[i].fillna(train[i].mean())
    test[i]=test[i].fillna(test[i].mean())

d=['BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','BsmtFullBath','BsmtHalfBath','GarageCars','GarageArea']
for i in d:
    test[i]=test[i].fillna(test[i].mean())
```

This step was essential to ensure that missing values did not reduce the predictive power of our model. By using the mode for categorical features and the mean for numerical ones, we ensured the integrity of the data.

4. Feature Engineering

Feature engineering is a critical part of any machine learning project. We engineered several new features:

- **TotalSF**: The total square footage of the house, combining basement, first, and second floors.
- **Age**: The age of the house at the time of sale.
- **RemodelAge**: Whether the house was remodeled, as a binary feature.

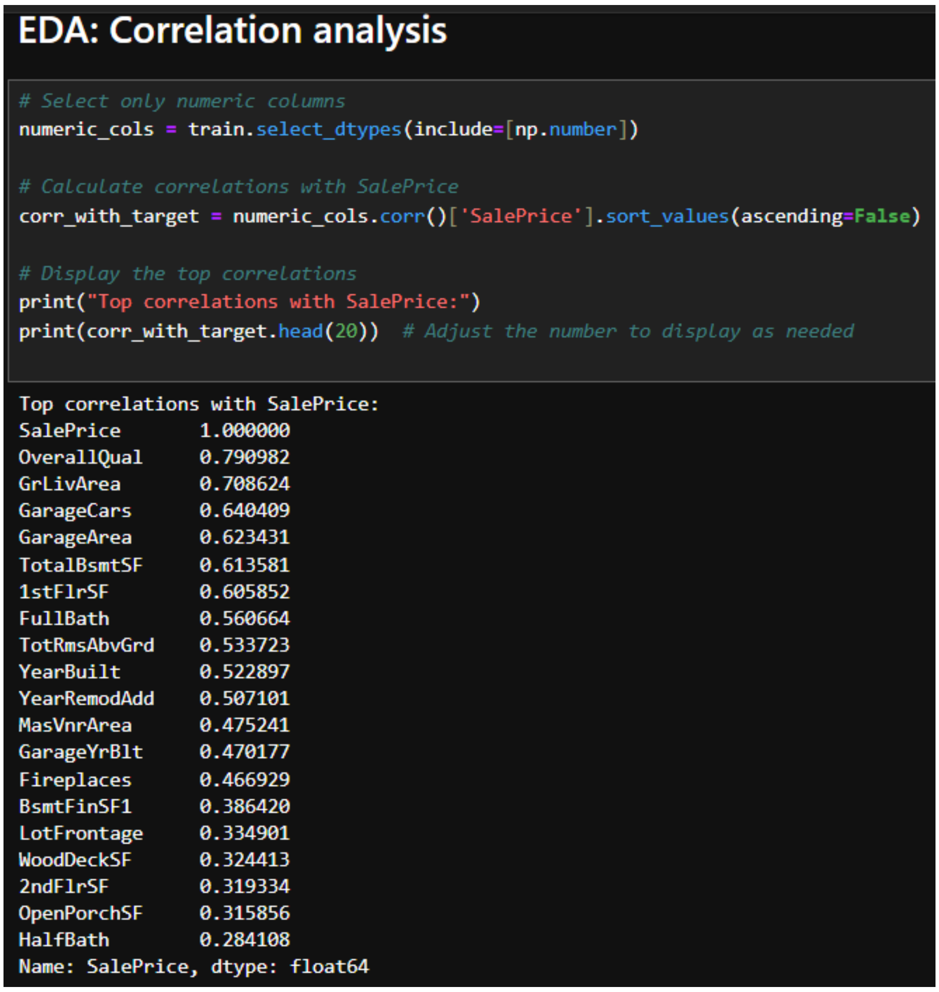


```python
train['TotalSF']=train['TotalBsmtSF']+train['1stFlrSF']+train['2ndFlrSF']
test['TotalSF']=test['TotalBsmtSF']+test['1stFlrSF']+test['2ndFlrSF']
train['Age']=train['YrSold']-train['YearBuilt']
test['Age']=test['YrSold']-test['YearBuilt']
train['RemodelAge']=train['YearRemodAdd']!=train['YearBuilt'].astype(int)
test['RemodelAge']=test['YearRemodAdd']!=test['YearBuilt'].astype(int)
```

These features capture important information such as the overall size of the house and how recent any renovations might have been. Both of these can have a significant impact on the final sale price.

5. Correlation Analysis

We performed correlation analysis to identify which features were most closely related to the target variable (SalePrice). This helps in selecting relevant features for the model.


EDA: Correlation analysis

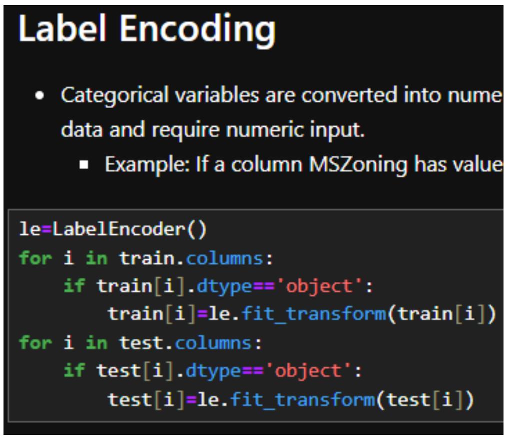
```
# Select only numeric columns
numeric_cols = train.select_dtypes(include=[np.number])

# Calculate correlations with SalePrice
corr_with_target = numeric_cols.corr()['SalePrice'].sort_values(ascending=False)

# Display the top correlations
print("Top correlations with SalePrice:")
print(corr_with_target.head(20))  # Adjust the number to display as needed
```

```
Top correlations with SalePrice:
SalePrice       1.000000
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
YearRemodAdd    0.507101
MasVnrArea      0.475241
GarageYrBlt     0.470177
Fireplaces      0.466929
BsmtFinSF1      0.386420
LotFrontage     0.334901
WoodDeckSF      0.324413
2ndFlrSF        0.319334
OpenPorchSF     0.315856
HalfBath        0.284108
Name: SalePrice, dtype: float64
```

The heatmap provided insight into which features had the strongest correlations with SalePrice. This analysis helped guide the selection of features for our model.

6. Label Encoding for Categorical Variables

Since machine learning models cannot handle categorical variables directly, we used **Label Encoding** to convert them into numerical values.


Label Encoding

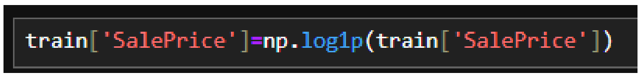
- Categorical variables are converted into nume
  data and require numeric input.
  - Example: If a column MSZoning has value

```
le=LabelEncoder()
for i in train.columns:
    if train[i].dtype=='object':
        train[i]=le.fit_transform(train[i])
for i in test.columns:
    if test[i].dtype=='object':
        test[i]=le.fit_transform(test[i])
```

Label encoding transforms categorical variables into numerical representations. This is particularly useful for models like Random Forests that work well with encoded data.

7. Log Transformation of Target Variable

The target variable, SalePrice, was log-transformed to reduce skewness and stabilize the variance. This step is important for linear regression models, as they assume normally distributed residuals.

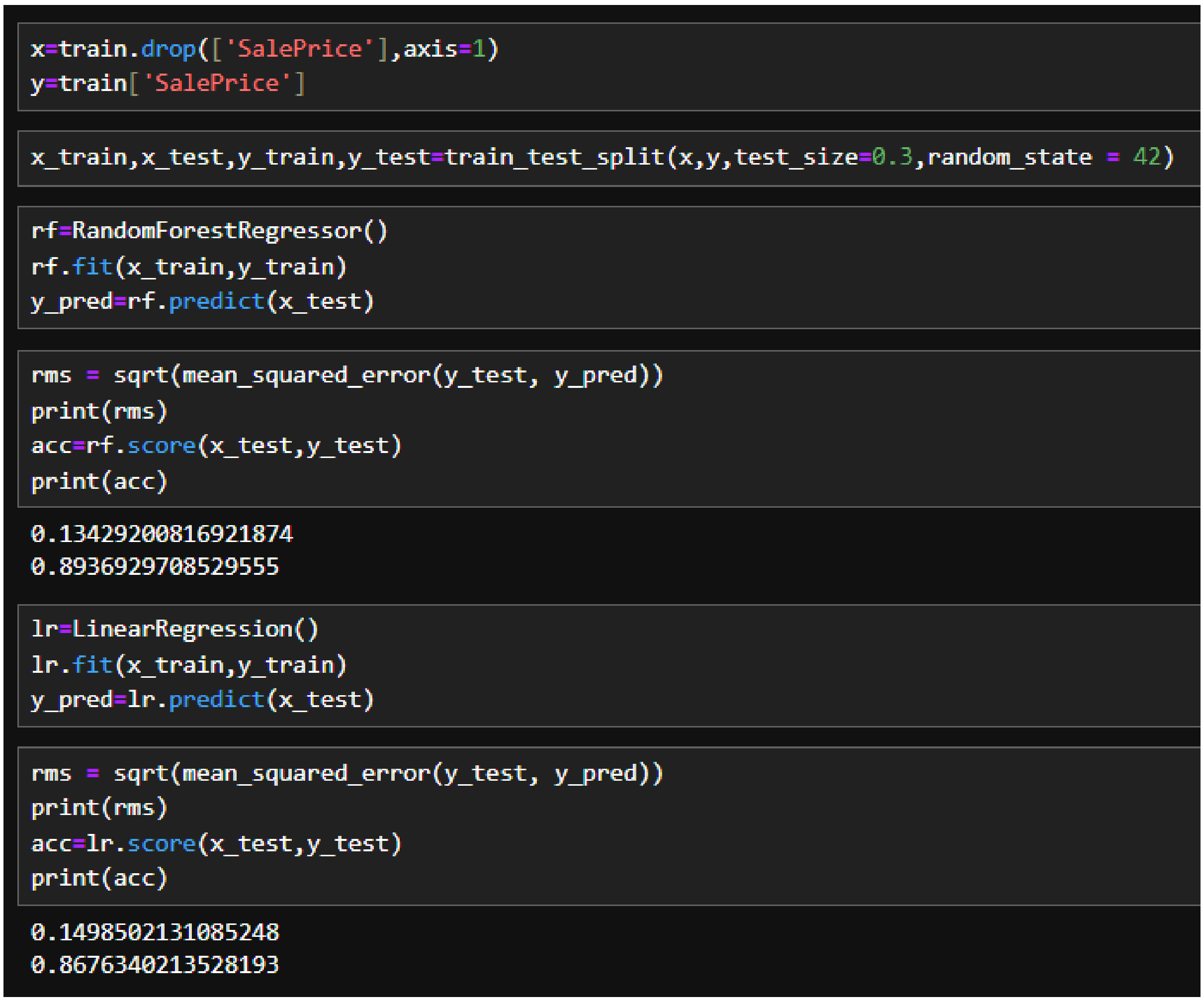


```
train['SalePrice']=np.log1p(train['SalePrice'])
```

The log transformation helped make the distribution of SalePrice more normal, which in turn improves the performance of regression models.

8. Model Training and Evaluation

We trained two models for comparison: **Random Forest** and **Linear Regression**. We split the data into training and testing sets to evaluate model performance.



```
x=train.drop(['SalePrice'],axis=1)
y=train['SalePrice']
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state = 42)
```

```
rf=RandomForestRegressor()
rf.fit(x_train,y_train)
y_pred=rf.predict(x_test)
```

```
rms = sqrt(mean_squared_error(y_test, y_pred))
print(rms)
acc=rf.score(x_test,y_test)
print(acc)
```

```
0.13429200816921874
0.8936929708529555
```

```
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
```

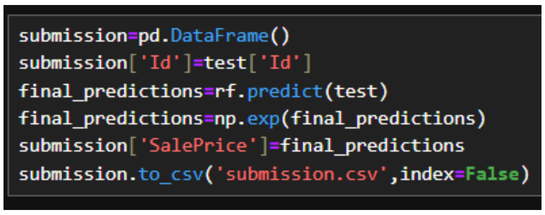
```
rms = sqrt(mean_squared_error(y_test, y_pred))
print(rms)
acc=lr.score(x_test,y_test)
print(acc)
```

```
0.1498502131085248
0.8676340213528193
```

The **Random Forest** model provided better performance in terms of RMSE and accuracy compared to the linear regression model. Random Forests are particularly good at capturing complex relationships in the data.

9. Generating Submission File

Finally, we used the trained Random Forest model to predict house prices on the test data, and we generated a CSV file for submission.



```python
submission=pd.DataFrame()
submission['Id']=test['Id']
final_predictions=rf.predict(test)
final_predictions=np.exp(final_predictions)
submission['SalePrice']=final_predictions
submission.to_csv('submission.csv',index=False)
```

We applied the inverse log transformation to the predictions to convert them back to the original sale price scale, and the results were saved in a CSV file as required by the competition.

Important Resources

Throughout the development of this project, I relied on several resources to gain insights and improve my solution. These resources helped me understand best practices for handling missing data, feature engineering, and regression modelling:

I. Feature Engineering Explained | Built In
II. Kaggle Competitions: The Complete Guide | DataCamp
III. Exploratory Data Analysis in Python - Data Science Project (Drug Reviews Dataset) - YouTube
IV. Stanford CS229: Machine Learning - Linear Regression and Gradient Descent | Lecture 2 (Autumn 2018) - YouTube
V. Basic of Numpy, Pandas , Matplotlib, Seaborn for Data Science 💚 ☀️ | Kaggle
VI. I have also gone through many articles and links on Google.