AdvancedRegression-SurpriseHousing

Surprise housing dataset using lasso and ridge regression

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them on at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below.

The company is looking at prospective properties to buy to enter the market. You are required to build a regression model using regularisation in order to predict the actual value of the prospective properties and decide whether to invest in them or not.

The company wants to know:

Which variables are significant in predicting the price of a house How well those variables describe the price of a house. Also, determine the optimal value of lambda for ridge and lasso regression.

Business Goal You are required to model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for management to understand the pricing dynamics of a new market.

DATA DESCRIPTION AT END OF FILE.

In [ ]:
```
Source Code : https://www.kaggle.com/snehac47/house-price-prediction-ridge-lasso-regression/notebook
```

1. IMPORTING DATASET AND LIBRARIES

import numpy as np import pandas as pd import matplotlib.pyplot as plt %matplotlib inline import seaborn as sns import statsmodels import statsmodels.api as sm from statsmodels.stats.outliers_influence import variance_inflation_factor from sklearn.feature_selection import RFE from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler from sklearn import linear_model from sklearn.linear_model import LinearRegression from sklearn.linear_model import Ridge from sklearn.linear_model import Lasso from sklearn.model_selection import GridSearchCV from sklearn import metrics # hide warnings import warnings warnings.filterwarnings('ignore')hs=pd.read_csv("Housing.csv")hs.head()hs.shapehs.info()

The pandas_profiling library in Python include a method named as ProfileReport() which generate a basic report on the input DataFrame.

The report consist of the following:

DataFrame overview, Each attribute on which DataFrame is defined, Correlations between attributes (Pearson Correlation and Spearman Correlation), and A sample of DataFrame.

numeric_data = hs.select_dtypes(include = ['float64','int64']) numeric_data.head()hs_missing=pd.DataFrame((round(100*(hs.isnull().sum()/hs.shape[0]), 2)), columns=['missing']) hs_missing.sort_values(by=['missing'], ascending=False).head(20)

To understand field description please refer to document file

Treating null values upon perusing field description

#addressing NaN values based on data description # In column 'PoolQC' (Pool quality), NaN stands for No Pool hs['PoolQC'] = hs['PoolQC'].fillna('No_Pool') # In column 'MiscFeature' (Miscellaneous Features), NaN stands for None, meaning the house has no miscellaneous features. hs['MiscFeature'] = hs['MiscFeature'].fillna('None') # In column 'Alley', NaN stands for No Alley Access as per the data description hs['Alley'] = hs['Alley'].fillna('No_Alley_Access') # In column 'Fence' (Fence Quality), NaN stands for No Fence as per the data description hs['Fence'] = hs['Fence'].fillna('No_Fence') # In column 'FireplaceQu' (FireplaceQu Quality), NaN stands for No Fireplace as per the data description hs['FireplaceQu'] = hs['FireplaceQu'].fillna('No_Fireplace') # LotFrontage stands for Linear feet of street connected to property, there is no explanation to impute this in data description # Let's consider imputing it with median of the lotFrontage of houses in the same neighbourhood # Group data by neighborhood and impute missing value with median LotFrontage of all the neighborhood hs["LotFrontage"] = hs.groupby("Neighborhood")["LotFrontage"].transform(lambda x: x.fillna(x.median())) # In column 'GarageYrBlt' (Gargae Year Built), NaN stands for houses with no garage, let's impute with 0 hs['GarageYrBlt'] = hs['GarageYrBlt'].fillna(0) # 'GarageType', 'GarageFinish', 'GarageQual' (Garage Quality) and 'GarageCond'(Garage Condition) # Missing values signify no garage as per data description.let's impute NaN values here with No Garage for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'): hs[col] = hs[col].fillna('No_Garage') #BsmtFinType1, BsmtFinType2 (Rating of basement finished area), #BsmtExposure (Basement Exposure), BsmtQual(Basement Quality), BsmtCond (Basement Conidtion) #These are all parameter related to basement. A NaN value probably signifies that the house does not have a basement. for col in ('BsmtFinType1', 'BsmtFinType2', 'BsmtExposure', 'BsmtQual','BsmtCond'): hs[col] = hs[col].fillna('No_Basement') # In column 'MasVnrType' (Masonry veneer type), let's impute it with mode "None" hs['MasVnrType'] = hs['MasVnrType'].fillna('None') # In column 'MasVnrType' (Masonry veneer type), let's impute it with mode 0 corresponding to None hs['MasVnrArea'] = hs['MasVnrArea'].fillna(0) # In column 'Electrical' (Electrical system), let's impute NaN with "Other" hs['Electrical'] = hs['Electrical'].fillna("Other") #checking percentage of null values in each column hs_missing=pd.DataFrame((round(100*(hs.isnull().sum()/len(hs.index)), 2)), columns=['missing']) hs_missing.sort_values(by=['missing'], ascending=False).head(20)

2. EDA and Data Preparation

#Let us first visualize the spread of Target Variable 'Sale Price' from scipy.stats import norm sns.distplot(hs['SalePrice'], fit=norm) plt.show()

The Sale Price distribution is not normally distributed, it is a little positively skewed with some houses having really high Sale Price

"MSSubClass" is a numeric column but it should actually be categorical as per the data dictionary, so let's convert that. hs=hs.replace({'MSSubClass' : { 20 : '1-STORY 1946 & NEWER ALL STYLES', 30:'1-STORY 1945 & OLDER', 40:'1-STORY W/FINISHED ATTIC ALL AGES', 45:'1-1/2 STORY - UNFINISHED ALL AGES', 50:'1-1/2 STORY FINISHED ALL AGES', 60:'2-STORY 1946 & NEWER', 70:'2-STORY 1945 & OLDER', 75:'2-1/2 STORY ALL AGES', 80:'SPLIT OR MULTI-LEVEL', 85:'SPLIT FOYER', 90:'DUPLEX - ALL STYLES AND AGES', 120:'1-STORY PUD (Planned Unit Development) - 1946 & NEWER', 150:'1-1/2 STORY PUD - ALL AGES', 160:'2-STORY PUD - 1946 & NEWER', 180:'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER', 190:'2 FAMILY CONVERSION - ALL STYLES AND AGES'}})numeric_data = hs.select_dtypes(include = ['float64','int64','int32']) numeric_data.columns

"ID" column is like row number, it has all unique values and can be ignored for analysis.

```
#function to plot scatter plot numeric variables with price def pp(w,x,y,z): sns.pairplot(hs, x_vars=[w,x,y,z], y_vars='SalePrice',height=4, aspect=1, kind='scatter') plt.show() pp('LotFrontage', 'LotArea', 'OverallQual','OverallCond') pp('YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'GrLivArea') pp('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF') pp('1stFlrSF', '2ndFlrSF','LowQualFinSF','MSSubClass') pp('BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath') pp('BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd','Fireplaces') pp('GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF') pp('OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch') pp('PoolArea','MiscVal', 'MoSold', 'YrSold')# label encode ordinal features where there is order in categories hs = hs.replace({ "Alley": {"No_Alley_Access" : 0, "Grvl" : 1, "Pave" : 2}, "BsmtCond": {"No_Basement" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "BsmtExposure": {"No_Basement" : 0, "No" : 2, "Mn" : 2, "Av": 3, "Gd" : 4}, "BsmtFinType1": {"No_Basement" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4, "ALQ" : 5, "GLQ" : 6}, "BsmtFinType2": {"No_Basement" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4, "ALQ" : 5, "GLQ" : 6}, "BsmtQual": {"No_Basement" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd" : 4, "Ex" : 5}, "CentralAir": {"None" : 0, "N" : 1, "Y" : 2}, "ExterCond": {"None" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5}, "ExterQual": {"None" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5}, "Fence": {"No_Fence" : 0, "MnWw" : 1, "GdWo" : 2, "MnPrv": 3, "GdPrv" : 4}, "FireplaceQu": {"No_Fireplace" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "Functional": {"None" : 0, "Sal" : 1, "Sev" : 2, "Maj2" : 3, "Maj1" : 4, "Mod": 5, "Min2" : 6, "Min1" : 7, "Typ" : 8}, "GarageCond": {"No_Garage" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "GarageQual": {"No_Garage" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "GarageFinish": {"No_Garage" : 0, "Unf" : 1, "RFn" : 2, "Fin" : 3}, "HeatingQC": {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "KitchenQual": {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "LandContour": {"None" : 0, "Low" : 1, "HLS" : 2, "Bnk" : 3, "Lvl" : 4}, "LandSlope": {"None" : 0, "Sev" : 1, "Mod" : 2, "Gtl" : 3}, "LotShape": {"None" : 0, "IR3" : 1, "IR2" : 2, "IR1" : 3, "Reg" : 4}, "PavedDrive": {"None" : 0, "N" : 0, "P" : 1, "Y" : 2}, "PoolQC": {"No_Pool" : 0, "Fa" : 1, "TA" : 2, "Gd" : 3, "Ex" : 4}, "Street": {"None" : 0, "Grvl" : 1, "Pave" : 2}, "Utilities": {"None" : 0, "ELO" : 1, "NoSeWa" : 2, "NoSewr" : 3, "AllPub" : 4}})
hs.BsmtCond = hs.BsmtCond.astype(int)def plot_charts(var1,var2,label_rotation): plt.figure(figsize=(12, 10)) plt.subplot(2,2,1) plt.title('Count Plot of '+ var1)
plt1=sns.countplot(hs[var1], palette=("husl")) plt1.set(xlabel = '%s'%var1, ylabel='Count of'+ '%s'%var1) if(label_rotation): plt1.set_xticklabels(plt1.get_xticklabels(),rotation=90)
plt.subplot(2,2,2) plt.title(var1+' vs Price') plt2=sns.boxplot(x=hs[var1], y=hs.SalePrice, palette=("husl")) if(label_rotation): plt2.set_xticklabels(plt2.get_xticklabels(),rotation=90)
plt.subplot(2,2,3) plt.title('Count Plot of '+ var2) plt3=sns.countplot(hs[var2], palette=("husl")) plt3.set(xlabel = '%s'%var2, ylabel='Count of'+ '%s'%var2) if(label_rotation):
plt3.set_xticklabels(plt3.get_xticklabels(),rotation=90) plt.subplot(2,2,4) plt.title(var2+' vs Price') plt4=sns.boxplot(x=hs[var2], y=hs.SalePrice, palette=("husl")) if(label_rotation):
plt4.set_xticklabels(plt4.get_xticklabels(),rotation=90) plt.show()categorical_features=hs.select_dtypes(include='object') categorical_features.columnsplot_charts('MSZoning',
'Street', label_rotation=False)print('Street:\n',hs['Street'].value_counts(dropna=False)) print('MSZoning:\n',hs['MSZoning'].value_counts(dropna=False))
```

Observation:

MsZoning identifies the general zoning classification of the sale. We see that the prices vary a lot for different values and most of the records are for "RL" Residential Low Density properties. Street identifies the Type of road access to property While we see that houses with Paved roads have higher Sale Price compared to Gravel, we must also note that this is a highly imbalanced variable with most records as "Pave"

```
plot_charts('LotShape','Alley',label_rotation=False)print('LotShape:\n',hs['LotShape'].value_counts(dropna=False))
print('Alley:\n',hs['Alley'].value_counts(dropna=False))plot_charts('LandContour','LotConfig',label_rotation=False)print('LandContour:\n',hs['LandContour'].value_counts(dropna=Fals
print('LotConfig:\n',hs['LotConfig'].value_counts(dropna=False))plot_charts('LandSlope','BldgType',label_rotation=False)print('LandSlope:\n',hs['LandSlope'].value_counts(dropna=Fa
print('BldgType:\n',hs['BldgType'].value_counts(dropna=False))plot_charts('RoofStyle',
'RoofMatl',label_rotation=True)print('RoofStyle:\n',hs['RoofStyle'].value_counts(dropna=False))
print('RoofMatl:\n',hs['RoofMatl'].value_counts(dropna=False))plot_charts('SaleType',
'SaleCondition',label_rotation=False)print('SaleType:\n',hs['SaleType'].value_counts(dropna=False)) print('SaleCondition:\n',hs['SaleCondition'].value_counts(dropna=False))
```

DATA PREPARATION

```
#changing months to categorical import calendar hs['MonthSold'] = hs['MoSold'].apply(lambda x: calendar.month_name[x]) hs=hs.drop(['MoSold'], axis=1)#changing data type of Gararge yr built to int from float hs['GarageYrBlt'] = hs['GarageYrBlt'].astype(int)#DERIVED VARIABLES which might make more sense than year hs['Age'] = hs['YrSold'] - hs['YearBuilt'] hs['Remod_Age'] = hs['YrSold'] - hs['YearRemodAdd'] hs['Garage_Age'] = hs['YrSold'] - hs['GarageYrBlt'] hs.drop(['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold'],1, inplace = True)numeric_data = hs.select_dtypes(include = ['float64','int64','int32']) numeric_data.columns#OUTLIER TREATMENT def remove_outliers(dtf, numl_list): for j in numl_list: Q1 = dtf[j].quantile(0.05) Q3 = dtf[j].quantile(0.95) IQR = Q3 - Q1 dtf = dtf[(dtf[j] >= Q1-2.5*IQR) & (dtf[j] <= Q3+2.5*IQR)] return dtfnumeric_data_list=list(numeric_data.columns)hs=remove_outliers(hs,numeric_data_list)# Outlier treatment on the variable Sale Price plt.figure(figsize=(8,5)) plt.boxplot(hs['SalePrice']) plt.show()hs.shape
```

Auto correlation heatmap plot

```
cor = numeric_data.corr() plt.figure(figsize=(20,20)) sns.heatmap(cor, annot=True) plt.show()
```

DELETING ID FEATURE AND BARPLOT FOR CORRELATION OF NUMERICAL FEATURES TO SALES PRICE

```
# we drop Id (not relevant) corr = hs.drop(["Id"], axis=1).select_dtypes(include="number").corr() plt.figure(figsize=(16,16)); corr["SalePrice"].sort_values(ascending=True)[:-1].plot(kind="barh") plt.title("Correlation of numerical features to SalePrice") plt.xlabel("Correlation to SalePrice") plt.tight_layout() plt.show()
```

Saleprice correlation matrix- nlargest (15 VARIABLES)

```
plt.figure(figsize=(7,7)) k = 15 #number of variables for heatmap cols = corr.nlargest(k, 'SalePrice')['SalePrice'].index cm = np.corrcoef(hs[cols].values.T) sns.set(font_scale=1.25) hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, cmap='viridis',yticklabels=cols.values, xticklabels=cols.values) plt.show()
```

Saleprice correlation matrix- nsmallest (15 VARIABLES)

```
#saleprice correlation matrix- nsmallest plt.figure(figsize=(7,7)) k = 15 #number of variables for heatmap cols = corr.nsmallest(k, 'SalePrice')['SalePrice'].index cm = np.corrcoef(hs[cols].values.T) sns.set(font_scale=1.25) hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, cmap='viridis',yticklabels=cols.values, xticklabels=cols.values) plt.show()
```

Converting binary variables to numeric by mapping to 0 and 1

```
hs['Street'] = hs['Street'].apply(lambda x: 1 if x == 'Pave' else 0 ) hs['CentralAir'] = hs['CentralAir'].apply(lambda x : 1 if x == 'Y' else 0) hs['PavedDrive'] = hs['PavedDrive'].apply(lambda x : 1 if x == 'Y' else 0)
```

SEGREGATING CATEGORICAL COLUMNS AND DROPPING ID COLUMN

```
df_hs = hs.drop(['Id'],axis=1) hs_categorical = df_hs.select_dtypes(include=['object']) hs_categorical.head()
```

convert categorical features into dummies

```
hs_dummies = pd.get_dummies(hs_categorical, drop_first=True) hs_dummies.head()
```

dropping original categorical columns

```
df_hs = df_hs.drop(list(hs_categorical.columns), axis=1)
```

concatenating dummy columns to original dataframe

```
df_hs = pd.concat([df_hs,hs_dummies], axis=1)df_hs.shape### Splitting data into train and test data using train test split#train_test_split
df_hs_train,df_hs_test=train_test_split(df_hs,train_size=0.70, random_state=100) df_hs_train.shape,df_hs_test.shapey_train = np.log(df_hs_train.SalePrice) X_train =
```

df_hs_train.drop("SalePrice",1) y_test= np.log(df_hs_test.SalePrice) X_test = df_hs_test.drop("SalePrice",1)X_train.shape,y_train.shape,X_test.shape,y_test.shapenum_vars=X_train.select_dtypes(include=['int64','float64','int32']).columns# numerical variables in train data num_vars### Standardisation of numerical variable using Standar Scalarscaler = StandardScaler() X_train[num_vars] = scaler.fit_transform(X_train[num_vars]) X_test[num_vars] = scaler.transform(X_test[num_vars])### linear regression model Recursive Feature Elimination : Recursive feature elimination is the process of iteratively finding the most relevant features from the parameters of a learnt ML model. is a popular feature selection algorithm. lm=LinearRegression() lm.fit(X_train,y_train) rfe = RFE(lm,20) rfe=rfe.fit(X_train,y_train) col=X_train.columns[rfe.support_] col### Using the most effective columns identified in RFE in Linear Regression Model & ordinary least SquareX_train_new=X_train[col] X_train_new = sm.add_constant(X_train_new) #create first model lr=sm.OLS(y_train,X_train_new) #fit the model lr_model=lr.fit() lr_model.summary()### RIDGE REGRESSION#RIDGE REGULARIZATION # list of alphas to tune params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100]} ridge = Ridge() # cross validation folds = 5 model_cv = GridSearchCV(estimator = ridge, param_grid = params, scoring= 'neg_mean_absolute_error', cv = folds, return_train_score=True, verbose = 1) model_cv.fit(X_train, y_train) print(model_cv.best_params_) print(model_cv.best_score_)cv_results = pd.DataFrame(model_cv.cv_results_) cv_results = cv_results[cv_results['param_alpha']<=100] cv_results# Plotting mean test and train scoes with alpha cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32') plt.figure(figsize=(16,5)) # plotting plt.plot(cv_results['param_alpha'], cv_results['mean_train_score']) plt.plot(cv_results['param_alpha'], cv_results['mean_test_score']) plt.xlabel('alpha') plt.ylabel('Negative Mean Absolute Error') plt.title("Negative Mean Absolute Error and alpha") plt.legend(['train score', 'test score'], loc='upper right') plt.show()

final ridge model

alpha = 10 ridge = Ridge(alpha=alpha) ridge.fit(X_train, y_train) ridge.coef_#lets predict the R-squared value of test and train data y_train_pred = ridge.predict(X_train) print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))y_test_pred = ridge.predict(X_test) print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred))from sklearn.metrics import mean_squared_error print ('RMSE is: \n', mean_squared_error(y_test, y_test_pred))# Ridge model parameters model_parameters_1 = list(ridge.coef_) model_parameters_1.insert(0, ridge.intercept_) model_parameters_1 = [round(x, 3) for x in model_parameters_1] cols = X_train.columns cols = cols.insert(0, "constant") list(zip(cols, model_parameters_1))

LASSO REGRESSION

#lasso params = {'alpha': [0.00005, 0.0001, 0.001, 0.008, 0.01]} lasso = Lasso() # cross validation model_cv_l = GridSearchCV(estimator = lasso, param_grid = params, scoring= 'neg_mean_absolute_error', cv = folds, return_train_score=True, verbose = 1) model_cv_l.fit(X_train, y_train)# cv results cv_results_l = pd.DataFrame(model_cv_l.cv_results_)#checking the value of optimum number of parameters print(model_cv_l.best_params_) print(model_cv_l.best_score_)### FINAL LASSO MODELalpha = 0.001 lasso = Lasso(alpha=alpha) lasso.fit(X_train, y_train) #lets predict the R-squared value of train data y_train_pred = lasso.predict(X_train) print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))#lets predict the R-squared value of testdata y_test_pred = lasso.predict(X_test) print(metrics.r2_score(y_true=y_test, y_pred=y_test_pred))from sklearn.metrics import mean_squared_error print ('RMSE is: \n', mean_squared_error(y_test, y_test_pred))#Lasso model parameters model_parameters_1 = list(lasso.coef_) model_parameters_1.insert(0, lasso.intercept_) model_parameters_1 = [round(x, 3) for x in model_parameters_1] cols = X_train.columns cols = cols.insert(0, "constant") list(zip(cols, model_parameters_1))### ERROR STATISTICS 1 LASSO RMSE 0.013496695507052504 2 RIDGE RMSE 0.014406843103375012 3 ADJUSTER R2 OF LINEAR REGRESSIN 0.698### EXTERNAL TEST DATA- FROM KAGGLE ### REPEATING DATA PREPARATION FOR TESTDATA SIMILAR TO CODES ABOVE AND USING THE LASSO PREDICTIN MODEL FOR PREDICTIONtest_data=pd.read_csv('test.txt',delimiter=',')pd.read_csv('test.txt',delimiter=',',)df_missing=pd.DataFrame((round(100*(test_data.isnull().sum()/len(test_data.index)), 2)), columns=['missing']) df_missing.sort_values(by=['missing'], ascending=False).head(20) 0.14 #addressing NaN values based on data dictionary # In column 'PoolQC' (Pool quality), NaN stands for No Pool test_data['PoolQC'] = test_data['PoolQC'].fillna('No_Pool') # In column 'MiscFeature' (Miscellaneous Features), NaN stands for None, meaning the house has no miscellaneous features. test_data['MiscFeature'] = test_data['MiscFeature'].fillna('None') # In column 'Alley', NaN stands for No Alley Access as per the data dictionary test_data['Alley'] = test_data['Alley'].fillna('No_Alley_Access') # In column 'Fence' (Fence Quality), NaN stands for No Fence as per the data dictionary test_data['Fence'] = test_data['Fence'].fillna('No_Fence') # In column 'FireplaceQu' (FireplaceQu Quality), NaN stands for No Fireplace as per the data dictionary test_data['FireplaceQu'] = test_data['FireplaceQu'].fillna('No_Fireplace') # LotFrontage stands for Linear feet of street connected to property, there is no explanation to impute this in data dictionary # Let's consider imputing it with median of the lotFrontage of houses in the same neighbourhood # Group data by neighborhood and impute missing value with median LotFrontage of all the neighborhood test_data["LotFrontage"] = test_data.groupby("Neighborhood")["LotFrontage"].transform(lambda x: x.fillna(x.median())) # In column 'GarageYrBlt' (Gargae Year Built), NaN stands for houses with no garage, let's impute with 0 test_data['GarageYrBlt'] = test_data['GarageYrBlt'].fillna(0) # 'GarageType', 'GarageFinish', 'GarageQual' (Garage Quality) and 'GarageCond'(Garage Condition) # Missing values signify no garage as per data dictionary.let's impute NaN values here with No Garage for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'): test_data[col] = test_data[col].fillna('No_Garage') #BsmtFinType1, BsmtFinType2 (Rating of basement finished area), #BsmtExposure (Basement Exposure), BsmtQual(Basement Quality), BsmtCond (Basement Conidtion) #These are all parameter related to basement. A NaN value probably signifies that the house does not have a basement. for col in ('BsmtFinType1', 'BsmtFinType2', 'BsmtExposure', 'BsmtQual','BsmtCond'): test_data[col] = test_data[col].fillna('No_Basement') # In column 'MasVnrType' (Masonry veneer type), let's impute it with mode "None" test_data['MasVnrType'] = test_data['MasVnrType'].fillna('None') # In column 'MasVnrType' (Masonry veneer type), let's impute it with mode 0 corresponding to None test_data['MasVnrArea'] = test_data['MasVnrArea'].fillna(0) # In column 'Electrical' (Electrical system), let's impute NaN with "Other" test_data['Electrical'] = test_data['Electrical'].fillna("Other")df_missing=pd.DataFrame((round(100*(test_data.isnull().sum()/len(test_data.index)), 2)), columns=['missing']) df_missing.sort_values(by=['missing'], ascending=False).head(20)catgl_feats = test_data.dtypes[test_data.dtypes == 'object'].index numrl_feats = test_data.dtypes[test_data.dtypes != 'object'].indexNan_cols = [] cols = test_data.columns for i in cols: if (test_data[i].isnull().sum()/len(df_hs))*100 > 0: Nan_cols.append(i)cat_treat_list = [] num_treat_list = [] for i in Nan_cols: if i in catgl_feats: cat_treat_list.append(i) else: num_treat_list.append(i)cat_treat_listnum_treat_listfor i in cat_treat_list: test_data[i].fillna(test_data[i].mode()[0], inplace = True)for i in num_treat_list: test_data[i].fillna(0, inplace = True)df_missing=pd.DataFrame((round(100*(test_data.isnull().sum()/len(test_data.index)), 2)), columns=['missing']) df_missing.sort_values(by=['missing'], ascending=False).head(20)# label encode ordinal features where there is order in categories test_data = test_data.replace({ "Alley": {"No_Alley_Access" : 0, "Grvl" : 1, "Pave" : 2}, "BsmtCond": {"No_Basement" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "BsmtExposure": {"No_Basement" : 0, "No" : 2, "Mn" : 2, "Av": 3, "Gd" : 4}, "BsmtFinType1": {"No_Basement" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4, "ALQ" : 5, "GLQ" : 6}, "BsmtFinType2": {"No_Basement" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3, "BLQ" : 4, "ALQ" : 5, "GLQ" : 6}, "BsmtQual": {"No_Basement" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd" : 4, "Ex" : 5}, "CentralAir": {"None" : 0, "N" : 1, "Y" : 2}, "ExterCond": {"None" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5}, "ExterQual": {"None" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" : 5}, "Fence": {"No_Fence" : 0, "MnWw" : 1, "GdWo" : 2, "MnPrv": 3, "GdPrv" : 4}, "FireplaceQu": {"No_Fireplace" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "Functional": {"None" : 0, "Sal" : 1, "Sev" : 2, "Maj2" : 3, "Maj1" : 4, "Mod": 5, "Min2" : 6, "Min1" : 7, "Typ" : 8}, "GarageCond": {"No_Garage" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "GarageQual": {"No_Garage" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "GarageFinish": {"No_Garage" : 0, "Unf" : 1, "RFn" : 2, "Fin" : 3}, "HeatingQC": {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "KitchenQual": {"None" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex" : 5}, "LandContour": {"None" : 0, "Low" : 1, "HLS" : 2, "Bnk" : 3, "Lvl" : 4}, "LandSlope": {"None" : 0, "Sev" : 1, "Mod" : 2, "Gtl" : 3}, "LotShape": {"None" : 0, "IR3" : 1, "IR2" : 2, "IR1" : 3, "Reg" : 4}, "PavedDrive": {"None" : 0, "N" : 0, "P" : 1, "Y" : 2}, "PoolQC": {"No_Pool" : 0, "Fa" : 1, "TA" : 2, "Gd" : 3, "Ex" : 4}, "Street": {"None" : 0, "Grvl" : 1, "Pave" : 2}, "Utilities": {"None" : 0, "ELO" : 1, "NoSeWa" : 2, "NoSewr" : 3, "AllPub" : 4}}) test_data.BsmtCond = test_data.BsmtCond.astype(int)test_data.GarageQual = test_data.GarageQual.astype(int)test_data.info()## "MSSubClass" is a numeric column but it should actually be categorical as per the data dictionary, so let's convert that. test_data=test_data.replace({'MSSubClass' : { 20 : '1-STORY 1946 & NEWER ALL STYLES', 30:'1-STORY 1945 & OLDER', 40:'1-STORY W/FINISHED ATTIC ALL AGES', 45:'1-1/2 STORY - UNFINISHED ALL AGES', 50:'1-1/2 STORY FINISHED ALL AGES', 60:'2-STORY 1946 & NEWER', 70:'2-STORY 1945 & OLDER', 75:'2-1/2 STORY ALL AGES', 80:'SPLIT OR MULTI-LEVEL', 85:'SPLIT FOYER', 90:'DUPLEX - ALL STYLES AND AGES', 120:'1-STORY PUD (Planned Unit Development) - 1946 & NEWER', 150:'1-1/2 STORY PUD - ALL AGES', 160:'2-STORY PUD - 1946 & NEWER', 180:'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER', 190:'2 FAMILY CONVERSION - ALL STYLES AND AGES'}})#changing months to categorical import calendar test_data['MonthSold'] = test_data['MoSold'].apply(lambda x: calendar.month_name[x]) test_data=test_data.drop(['MoSold'], axis=1)#changing data type of Gararge yr built to int from float test_data['GarageYrBlt'] = test_data['GarageYrBlt'].astype(int)test_data.info()#DERIVED VARIABLES which might make more sense than year test_data['Age'] = test_data['YrSold'] - test_data['YearBuilt'] test_data['Remod_Age'] = test_data['YrSold'] - test_data['YearRemodAdd'] test_data['Garage_Age'] = test_data['YrSold'] - test_data['GarageYrBlt'] test_data.drop(['YearBuilt','YearRemodAdd','GarageYrBlt','YrSold'],1, inplace = True)#converting binary variables to numeric by mapping to 0 and 1 test_data['Street'] = test_data['Street'].apply(lambda x: 1 if x == 'Pave' else 0) test_data['CentralAir'] = test_data['CentralAir'].apply(lambda x : 1 if x == 'Y' else 0)#converting binary variables to numeric by mapping to 0 and 1 test_data['PavedDrive'] = test_data['PavedDrive'].apply(lambda x : 1 if x == 'Y' else 0)test_data_X=test_data.drop('Id',1)test_data_X.shapetest_data_numerical=pd.DataFrame(test_data_X.select_dtypes(include=['int32','int64','float64']))test_data_numerical = scaler.transform(test_data_numerical)test_data_numericaltest_data_numerical.head()test_data_categorical = test_data_X.select_dtypes(include=['object']) test_data_categorical.head()test_data_categorical.shape# convert into dummies test_data_dummies = pd.get_dummies(test_data_categorical, drop_first=True) test_data_dummies.head()test_data_dummies.shape#dropping original categorical columns df_test = test_data_X.drop(list(test_data_categorical.columns), axis=1)df_test.shape#concatenating dummy columns to original dataframe df = pd.concat([df_test,test_data_dummies], axis=1)#lets predict the R-squared value of test and train

data y_test_predicted = lasso.predict(df) y_test_predictedfinal_predictions = np.exp(y_test_predicted) final_predictionssalespriceprediction= pd.DataFrame({'Id': test_data['Id'] ,'SalePrice': final_predictions }) salespriceprediction.to_csv("salespriceprediction.csv",index=False)

In [ ]:

Data description

The Australian housing dataset has 1460 rows and 81 columns.

MSSubClass: Identifies the type of dwelling involved in the sale.

```
 20  1-STORY 1946 & NEWER ALL STYLES
 30  1-STORY 1945 & OLDER
 40  1-STORY W/FINISHED ATTIC ALL AGES
 45  1-1/2 STORY - UNFINISHED ALL AGES
 50  1-1/2 STORY FINISHED ALL AGES
 60  2-STORY 1946 & NEWER
 70  2-STORY 1945 & OLDER
 75  2-1/2 STORY ALL AGES
 80  SPLIT OR MULTI-LEVEL
 85  SPLIT FOYER
 90  DUPLEX - ALL STYLES AND AGES
120  1-STORY PUD (Planned Unit Development) - 1946 & NEWER
150  1-1/2 STORY PUD - ALL AGES
160  2-STORY PUD - 1946 & NEWER
180  PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
190  2 FAMILY CONVERSION - ALL STYLES AND AGES
```

MSZoning: Identifies the general zoning classification of the sale.

```
A    Agriculture
C    Commercial
FV   Floating Village Residential
I    Industrial
RH   Residential High Density
RL   Residential Low Density
RP   Residential Low Density Park
RM   Residential Medium Density
```

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

```
Grvl Gravel
Pave Paved
```

Alley: Type of alley access to property

```
Grvl Gravel
Pave Paved
NA   No alley access
```

LotShape: General shape of property

```
Reg  Regular
IR1  Slightly irregular
IR2  Moderately Irregular
IR3  Irregular
```

LandContour: Flatness of the property

```
Lvl  Near Flat/Level
Bnk  Banked - Quick and significant rise from street grade to building
HLS  Hillside - Significant slope from side to side
Low  Depression
```

Utilities: Type of utilities available

```
AllPub   All public Utilities (E,G,W,& S)
NoSewr   Electricity, Gas, and Water (Septic Tank)
NoSeWa   Electricity and Gas Only
ELO  Electricity only
```

LotConfig: Lot configuration

```
Inside   Inside lot
Corner   Corner lot
CulDSac  Cul-de-sac
FR2  Frontage on 2 sides of property
FR3  Frontage on 3 sides of property
```

LandSlope: Slope of property

```
Gtl  Gentle slope
Mod  Moderate Slope
Sev  Severe Slope
```

Neighborhood: Physical locations within Ames city limits

```
Blmngtn  Bloomington Heights
Blueste  Bluestem
BrDale   Briardale
BrkSide  Brookside
ClearCr  Clear Creek
CollgCr  College Creek
Crawfor  Crawford
Edwards  Edwards
Gilbert  Gilbert
IDOTRR   Iowa DOT and Rail Road
MeadowV  Meadow Village
Mitchel  Mitchell
Names    North Ames
NoRidge  Northridge
NPkVill  Northpark Villa
NridgHt  Northridge Heights
NWAmes   Northwest Ames
OldTown  Old Town
SWISU    South & West of Iowa State University
Sawyer   Sawyer
SawyerW  Sawyer West
Somerst  Somerset
StoneBr  Stone Brook
Timber   Timberland
Veenker  Veenker
```

Condition1: Proximity to various conditions

```
Artery   Adjacent to arterial street
Feedr    Adjacent to feeder street
Norm Normal
RRNn Within 200' of North-South Railroad
RRAn Adjacent to North-South Railroad
PosN Near positive off-site feature--park, greenbelt, etc.
PosA Adjacent to postive off-site feature
RRNe Within 200' of East-West Railroad
RRAe Adjacent to East-West Railroad
```

Condition2: Proximity to various conditions (if more than one is present)

```
Artery   Adjacent to arterial street
Feedr    Adjacent to feeder street
Norm Normal
RRNn Within 200' of North-South Railroad
RRAn Adjacent to North-South Railroad
PosN Near positive off-site feature--park, greenbelt, etc.
PosA Adjacent to postive off-site feature
RRNe Within 200' of East-West Railroad
RRAe Adjacent to East-West Railroad
```

BldgType: Type of dwelling

```
1Fam Single-family Detached
2FmCon   Two-family Conversion; originally built as one-family dwelling
Duplx    Duplex
TwnhsE   Townhouse End Unit
TwnhsI   Townhouse Inside Unit
```

HouseStyle: Style of dwelling

```
1Story   One story
1.5Fin   One and one-half story: 2nd level finished
1.5Unf   One and one-half story: 2nd level unfinished
2Story   Two story
2.5Fin   Two and one-half story: 2nd level finished
2.5Unf   Two and one-half story: 2nd level unfinished
SFoyer   Split Foyer
SLvl Split Level
```

OverallQual: Rates the overall material and finish of the house

```
10   Very Excellent
9    Excellent
8    Very Good
7    Good
6    Above Average
5    Average
4    Below Average
3    Fair
2    Poor
1    Very Poor
```

OverallCond: Rates the overall condition of the house

```
10   Very Excellent
9    Excellent
8    Very Good
7    Good
6    Above Average
5    Average
4    Below Average
3    Fair
2    Poor
1    Very Poor
```

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

```
Flat Flat
Gable    Gable
Gambrel  Gabrel (Barn)
Hip  Hip
Mansard  Mansard
Shed Shed
```

RoofMatl: Roof material

```
ClyTile  Clay or Tile
CompShg  Standard (Composite) Shingle
Membran  Membrane
Metal    Metal
Roll Roll
Tar&Grv  Gravel & Tar
WdShake  Wood Shakes
WdShngl  Wood Shingles
```

Exterior1st: Exterior covering on house

```
AsbShng  Asbestos Shingles
AsphShn  Asphalt Shingles
BrkComm  Brick Common
BrkFace  Brick Face
CBlock   Cinder Block
CemntBd  Cement Board
HdBoard  Hard Board
ImStucc  Imitation Stucco
MetalSd  Metal Siding
Other    Other
Plywood  Plywood
PreCast  PreCast
Stone    Stone
Stucco   Stucco
VinylSd  Vinyl Siding
Wd Sdng  Wood Siding
WdShing  Wood Shingles
```

Exterior2nd: Exterior covering on house (if more than one material)

```
AsbShng  Asbestos Shingles
AsphShn  Asphalt Shingles
BrkComm  Brick Common
BrkFace  Brick Face
CBlock   Cinder Block
CemntBd  Cement Board
HdBoard  Hard Board
ImStucc  Imitation Stucco
MetalSd  Metal Siding
Other    Other
Plywood  Plywood
PreCast  PreCast
Stone    Stone
Stucco   Stucco
VinylSd  Vinyl Siding
Wd Sdng  Wood Siding
WdShing  Wood Shingles
```

MasVnrType: Masonry veneer type

```
BrkCmn   Brick Common
BrkFace  Brick Face
CBlock   Cinder Block
None None
Stone    Stone
```

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

```
Ex   Excellent
Gd   Good
TA   Average/Typical
Fa   Fair
Po   Poor
```

ExterCond: Evaluates the present condition of the material on the exterior

```
Ex   Excellent
Gd   Good
TA   Average/Typical
Fa   Fair
Po   Poor
```

Foundation: Type of foundation

```
BrkTil   Brick & Tile
CBlock   Cinder Block
PConc    Poured Contrete
Slab Slab
Stone    Stone
```

```
Wood Wood
```

BsmtQual: Evaluates the height of the basement

```
Ex   Excellent (100+ inches)
Gd   Good (90-99 inches)
TA   Typical (80-89 inches)
Fa   Fair (70-79 inches)
Po   Poor (<70 inches
NA   No Basement
```

BsmtCond: Evaluates the general condition of the basement

```
Ex   Excellent
Gd   Good
TA   Typical - slight dampness allowed
Fa   Fair - dampness or some cracking or settling
Po   Poor - Severe cracking, settling, or wetness
NA   No Basement
```

BsmtExposure: Refers to walkout or garden level walls

```
Gd   Good Exposure
Av   Average Exposure (split levels or foyers typically score average or above)
Mn   Mimimum Exposure
No   No Exposure
NA   No Basement
```

BsmtFinType1: Rating of basement finished area

```
GLQ  Good Living Quarters
ALQ  Average Living Quarters
BLQ  Below Average Living Quarters
Rec  Average Rec Room
LwQ  Low Quality
Unf  Unfinshed
NA   No Basement
```

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

```
GLQ  Good Living Quarters
ALQ  Average Living Quarters
BLQ  Below Average Living Quarters
Rec  Average Rec Room
LwQ  Low Quality
Unf  Unfinshed
NA   No Basement
```

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

```
Floor    Floor Furnace
GasA Gas forced warm air furnace
GasW Gas hot water or steam heat
Grav Gravity furnace
OthW Hot water or steam heat other than gas
Wall Wall furnace
```

HeatingQC: Heating quality and condition

```
Ex   Excellent
Gd   Good
```

```
TA   Average/Typical
Fa   Fair
Po   Poor
```

CentralAir: Central air conditioning

```
N    No
Y    Yes
```

Electrical: Electrical system

```
SBrkr    Standard Circuit Breakers & Romex
FuseA    Fuse Box over 60 AMP and all Romex wiring (Average)
FuseF    60 AMP Fuse Box and mostly Romex wiring (Fair)
FuseP    60 AMP Fuse Box and mostly knob & tube wiring (poor)
Mix  Mixed
```

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

```
Ex   Excellent
Gd   Good
TA   Typical/Average
Fa   Fair
Po   Poor
```

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

```
Typ  Typical Functionality
Min1 Minor Deductions 1
Min2 Minor Deductions 2
Mod  Moderate Deductions
Maj1 Major Deductions 1
Maj2 Major Deductions 2
Sev  Severely Damaged
Sal  Salvage only
```

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

```
Ex   Excellent - Exceptional Masonry Fireplace
Gd   Good - Masonry Fireplace in main level
TA   Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement
Fa   Fair - Prefabricated Fireplace in basement
Po   Poor - Ben Franklin Stove
NA   No Fireplace
```

GarageType: Garage location

```
2Types   More than one type of garage
Attchd   Attached to home
Basment  Basement Garage
BuiltIn  Built-In (Garage part of house - typically has room above garage)
CarPort  Car Port
Detchd   Detached from home
NA   No Garage
```

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

```
Fin  Finished
RFn  Rough Finished
Unf  Unfinished
NA   No Garage
```

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

```
Ex   Excellent
Gd   Good
TA   Typical/Average
Fa   Fair
Po   Poor
NA   No Garage
```

GarageCond: Garage condition

```
Ex   Excellent
Gd   Good
TA   Typical/Average
Fa   Fair
Po   Poor
NA   No Garage
```

PavedDrive: Paved driveway

```
Y    Paved
P    Partial Pavement
N    Dirt/Gravel
```

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

```
Ex   Excellent
Gd   Good
TA   Average/Typical
Fa   Fair
NA   No Pool
```

Fence: Fence quality

```
GdPrv    Good Privacy
MnPrv    Minimum Privacy
GdWo Good Wood
MnWw Minimum Wood/Wire
```

```
NA   No Fence
```

MiscFeature: Miscellaneous feature not covered in other categories

```
Elev Elevator
Gar2 2nd Garage (if not described in garage section)
Othr Other
Shed Shed (over 100 SF)
TenC Tennis Court
NA   None
```

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

```
WD   Warranty Deed - Conventional
CWD  Warranty Deed - Cash
VWD  Warranty Deed - VA Loan
New  Home just constructed and sold
COD  Court Officer Deed/Estate
Con  Contract 15% Down payment regular terms
ConLw    Contract Low Down payment and low interest
ConLI    Contract Low Interest
ConLD    Contract Low Down
Oth  Other
```

SaleCondition: Condition of sale

```
Normal   Normal Sale
Abnorml  Abnormal Sale -  trade, foreclosure, short sale
AdjLand  Adjoining Land Purchase
Alloca   Allocation - two linked properties with separate deeds, typically condo with a garage unit
Family   Sale between family members
Partial  Home was not completed when last assessed (associated with New Homes)
```

In [ ]: