Assignment 5

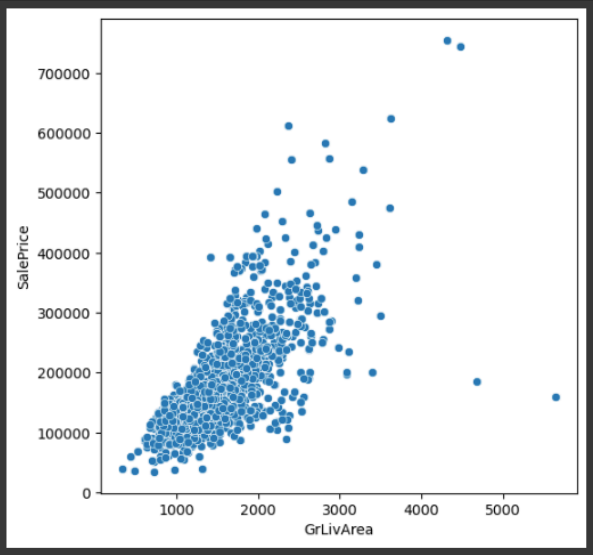
U21CS089
Garvit Shah

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from matplotlib import rcParams
train = pd.read_csv('/content/drive/MyDrive/ML/Housing_train.csv')
test = pd.read_csv('/content/drive/MyDrive/ML/Housing_test.csv')
train.fillna(method = 'ffill', inplace = True)
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'], test.loc[:,'MSSubClass':'SaleCondition']))
train.head()
# print(price_test.shape)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFea
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	

5 rows × 81 columns

```
rcParams['figure.figsize'] = (6.0, 6.0) # define size of figure
sns.scatterplot(x='GrLivArea', y='SalePrice', data=train)
plt.show()
```

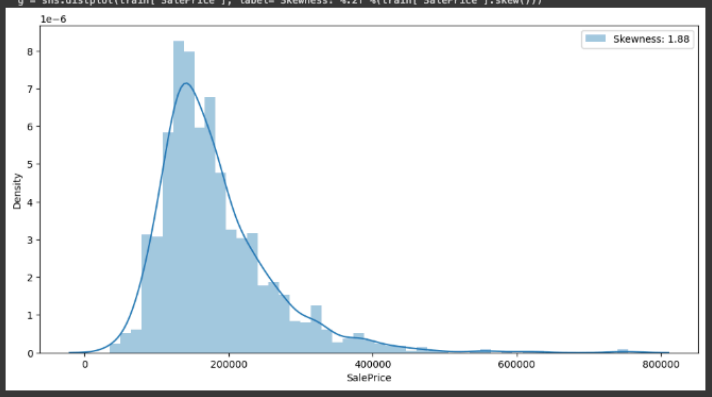



```
[159] train = train.drop(train[(train['GrLivArea']>4000) & (train['SalePrice']<200000)].index).reset_index(drop=True)

all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)
all_data['OverallCond'] = all_data['OverallCond'].astype(str)
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

```
[160] from sklearn.preprocessing import LabelEncoder

cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold')

for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))
```

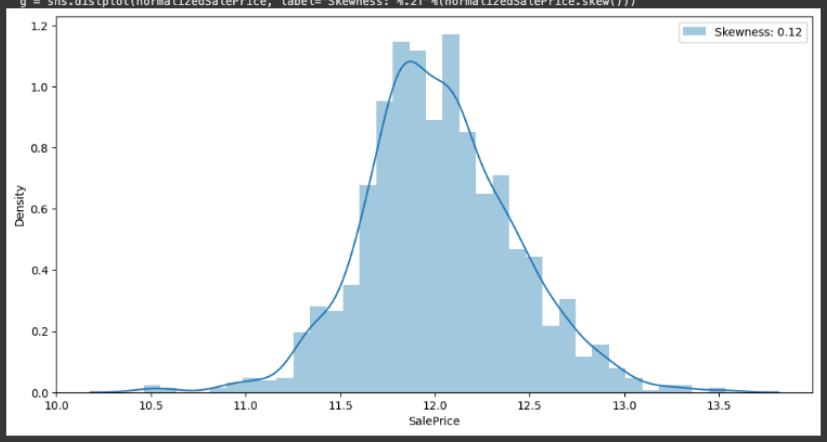
```
from scipy.stats import skew
from matplotlib import rcParams

# plot histogram of "SalePrice"
rcParams['figure.figsize'] = (12.0, 6.0) # define size of figure
g = sns.distplot(train["SalePrice"], label="Skewness: %.2f"%(train["SalePrice"].skew()))
g = g.legend(loc="best")
plt.show()
```

```
<ipython-input-161-04f8ddca7fb1>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  g = sns.distplot(train["SalePrice"], label="Skewness: %.2f"%(train["SalePrice"].skew()))
```




```
normalizedSalePrice = np.log1p(train["SalePrice"])

# plot histogram of log transformed "SalePrice"
rcParams['figure.figsize'] = (12.0, 6.0) # define size of figure
g = sns.distplot(normalizedSalePrice, label="Skewness: %.2f"%(normalizedSalePrice.skew()))
g = g.legend(loc="best")
plt.show()
```

```
<ipython-input-162-cafcf400acd3>:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  g = sns.distplot(normalizedSalePrice, label="Skewness: %.2f"%(normalizedSalePrice.skew()))
```




```
[163] train["SalePrice"] = np.log1p(train["SalePrice"])
```

```
# determine features that are heavily skewed
def get_skewed_features():
    numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
    skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())) # computes "skewness"
    skewed_feats = skewed_feats[abs(skewed_feats) > 0.75]
    return skewed_feats.index


from sklearn.preprocessing import power_transform

# find heavily skewed numerical features
skewed_feats = get_skewed_features()
print("{} heavily skewed features.".format(len(skewed_feats)))

# apply power transform to all heavily skewed numeric features
all_data[skewed_feats] = power_transform(all_data[skewed_feats], method='yeo-johnson')
print("Applied power transform.")
```

```
35 heavily skewed features.
Applied power transform.
```

```
[165] # create dummy variables
all_data = pd.get_dummies(all_data)
all_data.shape # we now have 219 features columns compared to original 79
```

```
(2919, 220)
```

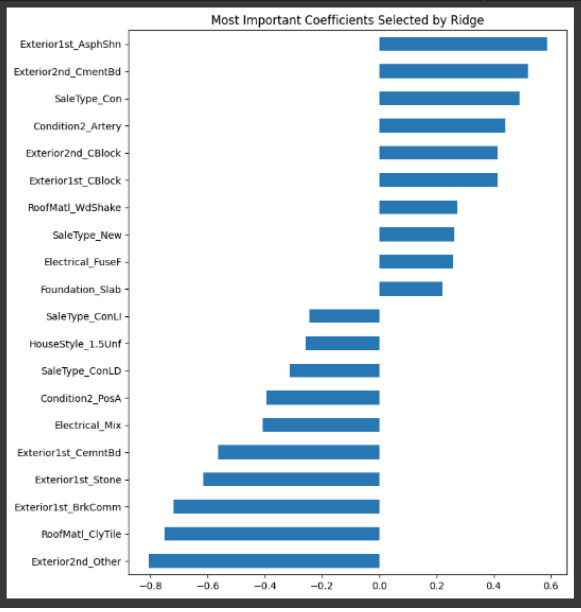
```
[166] all_data.isnull().any().any()
```

```
True
```

```
# replace NA's with the mean of the feature
all_data = all_data.fillna(all_data.mean())

# check again for any missing values
all_data.isnull().any().any()
```

```
False
```

```
[168] # create matrices for sklearn
      # fit linear model
      linearModel.fit(X_train, y)

      # get largest magnitude coefficients
      coef = pd.Series(linearModel.coef_, index = X_train.columns)
      imp_coef = pd.concat([coef.sort_values().head(10), coef.sort_values().tail(10)])

      rcParams['figure.figsize'] = (8.0, 10.0) # define size of figure
      imp_coef.plot(kind = "barh")
      plt.title("Most Important Coefficients Selected by Ridge")
      plt.show()
```



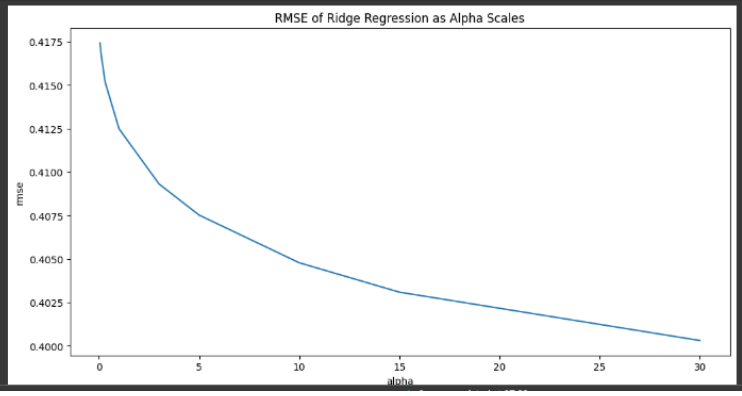
(1) **Impact of normalisation on the overall accuracy.**
Normalisation reduces the overall variance or skewness of the data. It does not change the relation between the data it just changes the scale. When the data is normalised the L1 or L2 regularisation treats all the features equally to penalise, rather than showing any kind of bias. If the data is not normalised, the regularisation techniques shows bias in penalising some features more than others in return resulting in lower accuracy.

```
[67] from sklearn.linear_model import Ridge

     # determine RMSE for ridge regression model with alpha = 0.1
     ridgeModel = Ridge(alpha = 0.1)
     rmse = rmse_cv(ridgeModel)
     print("RMSE estimate: {}, std: {}".format(rmse.mean(), rmse.std()))

     RMSE estimate: 0.4168339549168529, std: 0.04922313815088954
```

Previous value of RSME by Linear Regression - `0.41818`

New value after L2 Regularisation - `0.41683`

```
     rcParams['figure.figsize'] = (12.0, 6.0) # define size of figure

     # calculate RMSE over several alphas
     alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30]
     cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean() for alpha in alphas]
     cv_ridge = pd.Series(cv_ridge, index = alphas)

     # plot RMSE vs alpha
     cv_ridge.plot(title = "RMSE of Ridge Regression as Alpha Scales")
     plt.xlabel("alpha")
     plt.ylabel("rmse")
     plt.show()
```



```
[155] optimalRidgeAlpha = cv_ridge[cv_ridge == cv_ridge.min()].index.values[0]
      print("Optimal ridge alpha: {}".format(optimalRidgeAlpha))

      Optimal ridge alpha: 30.0

[156] # determine RMSE for ridge regression model with optimal alpha
      ridgeModel = Ridge(alpha = optimalRidgeAlpha)
      rmse = rmse_cv(ridgeModel)
      print("RMSE estimate: {}, std: {}".format(rmse.mean(), rmse.std()))

      RMSE estimate: 0.4003141912778009, std: 0.051125826836086186
```

Previous value of RSME by Linear Regression - `0.41818`

New value after L2 Regularisation - `0.41683`

Using alpha as 30 for L2 Regularisation - `0.40031`

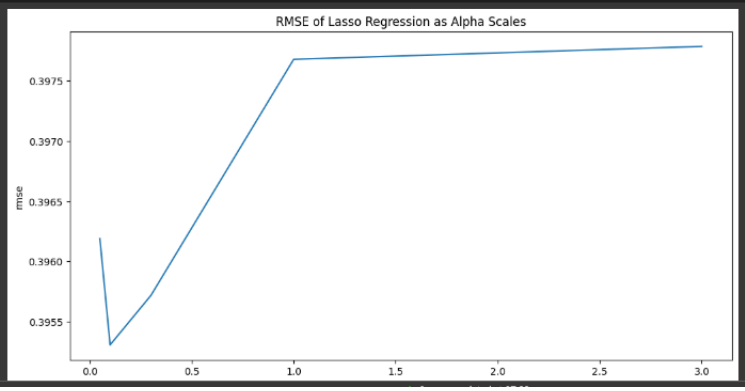
(2) Find out different RMSE values for different values of alpha in L1 and L2 regression.

```
from sklearn.linear_model import Lasso

# determine RMSE for lasso regression model with alpha = 0.1
lassoModel = Lasso(alpha = 0.1)
rmse = rmse_cv(lassoModel)
print("RMSE estimate: {}, std: {}".format(rmse.mean(), rmse.std()))

RMSE estimate: 0.3953058108170336 4, std: 0.03222919340688156
```

Previous value of RSME by Linear Regression - `0.41818`

New value of RSME by L1 Regularisation at alpha 0.1 - `0.39530`

```
[136] from sklearn.linear_model import Lasso

      rcParams['figure.figsize'] = (12.0, 6.0) # define size of figure

      # calculate RMSE over several alphas
      alphas = [0.05, 0.1, 0.3, 1, 3]
      cv_lasso = [rmse_cv(Lasso(alpha = alpha)).mean() for alpha in alphas]
      cv_lasso = pd.Series(cv_lasso, index = alphas)

      # plot RMSE vs alpha
      cv_lasso.plot(title = "RMSE of Lasso Regression as Alpha Scales")
      plt.xlabel("alpha")
      plt.ylabel("rmse")
      plt.show()
```




```
[172] from sklearn.linear_model import LassoCV

      # use built in LassoCV function to select best model for data
      lassoModel = LassoCV(alphas = np.linspace(0.0002, 0.0022, 21), cv = 5).fit(X_train, y)
      lassoModel.alpha_

      optimalLassoAlpha = lassoModel.alpha_
      print("Optimal lasso alpha: {}".format(optimalLassoAlpha))

      Optimal lasso alpha: 0.0022

[173] lassoModel = Lasso(alpha = optimalLassoAlpha)
      rmse = rmse_cv(lassoModel)
      print("RMSE estimate: {}, std: {}".format(rmse.mean(), rmse.std()))

      RMSE estimate: 0.39559685671806855, std: 0.05196854845431636

[ ]  Previous value of RSME by Linear Regression - 0.41818

     New value of RSME by L1 Regularisation at alpha 0.1 - 0.39530

     Best value of RSME by L1 Regularisation at alpha 0.0022 - `0.395596`
```