```
import os
for dirname, _, filenames in os.walk('C:\Users\KNOT\Desktop\Assignments\RealEstateAU_1000_Samples'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

#Importing the required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import datetime
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn import metrics

#Reading the CSV file
house = pd.read_csv("/\Users\KNOT\Desktop\Assignments\RealEstateAU_1000_Samples.csv")

#Daata cleaning and mapping
house.head()

#Getting an over view of the data
house.info()

#Checking % of Null values
round(100*(house.isnull().sum()/len(house.index)),2)

# As per the domain knowledge, mapping the columns.
def funct_mapper(x):
    return x.map({'Typ': 7, "Min1": 6, "Min2": 5, "Mod": 4, "Maj1": 3, 'Maj2': 2, 'Sev': 1, 'Sal': 0})

def fence_mapper(x):
    return x.map({'GdPrv': 4, "MnPrv": 3, "GdWo": 2, "MnWw": 1, "None": 0})

def rating_mapper(x):
return x.map({'Ex': 5, "Gd": 4, "TA": 3, "Fa": 2, "Po": 1, "None": 0})

def rating_mapper1(x):
    return x.map({'Gd': 4, "Av": 3, "Mn": 2, "No": 1, "None": 0})

def rating_mapper2(x):
    return x.map({'GLQ': 6, "ALQ": 5, "BLQ": 4, "Rec": 3, "LwQ": 2, 'Unf': 1, 'None': 0})

def hs_mapper(x):
    return x.map({'Fin': 3, "RFn": 2, "Unf": 1, "None": 0})
```

```
def ls_mapper(x):
    return x.map({'Reg': 3, "IR1": 2, "IR2": 1, "IR3": 0})

def landsloper_mapper(x):
return x.map({'Gtl': 2, "Mod": 1, "Sev": 0})

# Applying the function to the columns
house[['LotShape']] = house[['LotShape']].apply(ls_mapper)
house[['LandSlope']] = house[['LandSlope']].apply(landsloper_mapper)
house[['Functional']] = house[['Functional']].apply(funct_mapper)
house[['Fence']] = house[['Fence']].apply(fence_mapper)
house[['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual', 'FireplaceQu', 'GarageQual', '
GarageCond', 'PoolQC']] = house[['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual', 'Fi
replaceQu', 'GarageQual', 'GarageCond', 'PoolQC']].apply(rating_mapper)
house[['BsmtExposure']] = house[['BsmtExposure']].apply(rating_mapper1)
house[['BsmtFinType1', 'BsmtFinType2']] = house[['BsmtFinType1', 'BsmtFinType2']].apply(rating_mapper2)
house[['GarageFinish']] = house[['GarageFinish']].apply(hs_mapper)

#We see that there are a lot of null values here. But as per the domain knowledge these are not actually null values.
#Hence will replace NaN's instead of dropping them.

house['Alley'].fillna('No Alley', inplace=True)
house['BsmtQual'].fillna('No Basement', inplace=True)
house['BsmtCond'].fillna('No Basement', inplace=True)
house['BsmtExposure'].fillna('No Basement', inplace=True)
house['BsmtFinType1'].fillna('No Basement', inplace=True)
house['BsmtFinType2'].fillna('No Basement', inplace=True)
house['FireplaceQu'].fillna('No Fireplace', inplace=True)
house['GarageType'].fillna('No Garage', inplace=True)
house['GarageFinish'].fillna('No Garage', inplace=True)
house['GarageQual'].fillna('No Garage', inplace=True)
house['GarageCond'].fillna('No Garage', inplace=True)
house['PoolQC'].fillna('No Pool', inplace=True)
house['Fence'].fillna('No Fence', inplace=True)
house['MiscFeature'].fillna('None', inplace=True)
house['GarageYrBlt'].fillna(2019, inplace=True)

#Converting the following to number of years
house['YearBuilt'] = 2019 - house['YearBuilt']
house['YearRemodAdd'] = 2019 - house['YearRemodAdd']
house['GarageYrBlt'] = 2019 - house['GarageYrBlt']
house['YrSold'] = 2019 - house['YrSold']

#Checking for null values again
round(100*(house.isnull().sum()/len(house.index)),2)

# Filling mean value in LotFrontage where ever there are NaN values.

house['LotFrontage'].fillna((house['LotFrontage'].mean()), inplace = True)

# Removing rows where ever MasVnrType,MasVnrArea,Electrical is NaN
house = house[pd.notnull(house['MasVnrType'])]
house = house[pd.notnull(house['MasVnrArea'])]
house =house[pd.notnull(house['Electrical'])]
```

```
#Now the null values are cleaned.
round(100*(house.isnull().sum()/len(house.index)),2)

#Data Preparation
# Seperating the Numeric columns from the data
house_numeric = house.select_dtypes(include=['float64', 'int64'])
house_numeric.columns

Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'LotShape', 'LandSlope',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', 'HeatingQC', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
       'Functional', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')

#Checking for corelation
cor = house_numeric.corr()
cor

# Seperating the Categorical columns from the data
house_categorical = house.select_dtypes(include=['object'])
house_categorical.columns

Index(['MSZoning', 'Street', 'Alley', 'LandContour', 'Utilities', 'LotConfig',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
       'BsmtFinType2', 'Heating', 'CentralAir', 'Electrical', 'FireplaceQu',
       'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive',
       'PoolQC', 'Fence', 'MiscFeature', 'SaleType', 'SaleCondition'],
      dtype='object')

# Convert into dummies
house_dummies = pd.get_dummies(house_categorical, drop_first=True)
house_dummies.head()

#Dropping the columns for which we have dummy values.
house = house.drop(list(house_categorical.columns), axis=1)
house.columns

Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'LotShape', 'LandSlope',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', 'HeatingQC', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
       'Functional', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

```
house= pd.concat([house, house_dummies], axis=1)

house.columns

Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'LotShape', 'LandSlope',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       ...
       'SaleType_ConLI', 'SaleType_ConLw', 'SaleType_New', 'SaleType_Oth',
       'SaleType_WD', 'SaleCondition_AdjLand', 'SaleCondition_Alloca',
       'SaleCondition_Family', 'SaleCondition_Normal',
       'SaleCondition_Partial'],
      dtype='object', length=243)

house.head()

house1= house.drop(["Id", "SalePrice"], axis=1)

house1

from sklearn.preprocessing import scale
cols = house1.columns
house1 = pd.DataFrame(scale(house1))
house1.columns = cols

house1.head()

# Assigning X and y where X are the independent variables and y is the dependent variable.
X = house1
y = house["SalePrice"].values
sns.distplot(house["SalePrice"])

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    train_size=0.7,
                                    test_size = 0.3, random_state=100)

# Instantiate
lm = LinearRegression()

# Fit a line
lm.fit(X_train, y_train)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

from sklearn.feature_selection import RFE

lm = LinearRegression()
rfe1 = RFE(lm, 50)

# Fit with 15 feature
rfe1.fit(X_train, y_train)

# Print the boolean results
print(rfe1.support_)
```

```
print(rfe1.ranking_)

#Ridge Regression

# list of alphas to tune
list_alpha = [0.0001,0.001,0.01,0.1, 0.5, 1.0, 5.0, 20,
 40, 70, 100, 150, 200, 250, 300, 350, 400, 450, 500,
 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000, 2000]
params = {'alpha': list_alpha}

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)

cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results = cv_results[cv_results['param_alpha']<= 1000]
cv_results.head()

cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting
plt.figure(figsize=(10,5))
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 left')
plt.show()

model_cv.best_params_

alpha = 350
ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)
ridge.coef_

ridge = Ridge(alpha=alpha)

ridge.fit(X_train, y_train)

pred = ridge.predict(X_test)
mse = np.mean((pred - y_test)**2)
mse

from sklearn import metrics
```

```
y_train_pred = ridge.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))

#Lasso Regression

lasso = Lasso()

# cross validation
model_cv = GridSearchCV(estimator = lasso,
                        param_grid = params,
                        scoring= 'neg_mean_absolute_error',
                        cv = folds,
                        return_train_score=True,
                        verbose = 1)

model_cv.fit(X_train, y_train)


cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()

# plotting mean test and train scoes with alpha
cv_results['param_alpha'] = cv_results['param_alpha'].astype('float32')

# plotting
plt.figure(figsize=(10,5))

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 left')
plt.show()

model_cv.best_params_

alpha = 400

lasso = Lasso(alpha=alpha)

lasso.fit(X_train, y_train)

lasso.coef_

lasso.score(X_test,y_test)

from sklearn import metrics
y_train_pred = lasso.predict(X_train)
print(metrics.r2_score(y_true=y_train, y_pred=y_train_pred))
```