AML_HW1_House Price Prediction

October 2, 2020

[ ]:
```
# Team Members:
#    Scarlett Huang (sh2557)
#    Zihan Zhang (zz698)
```

[62]:
```
# import modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sbn
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
import os
```

[64]:
```
 # import data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

# drop 'Id'
Id = test['Id']
train.drop('Id',axis=1,inplace=True)
test.drop('Id',axis=1,inplace=True)
```

[35]:
```
# explore data
print (train.describe)
print (train.columns)
print (train.head(5))
print (train.shape)
print (test.shape)
```

```
<bound method NDFrame.describe of       MSSubClass MSZoning  LotFrontage
LotArea Street Alley LotShape  \
0             60       RL         65.0     8450   Pave   NaN      Reg
1             20       RL         80.0     9600   Pave   NaN      Reg
2             60       RL         68.0    11250   Pave   NaN      IR1
3             70       RL         60.0     9550   Pave   NaN      IR1
4             60       RL         84.0    14260   Pave   NaN      IR1
…          …      …          …       …    …   …        …
```

```
1455          60       RL         62.0     7917   Pave   NaN      Reg
1456          20       RL         85.0    13175   Pave   NaN      Reg
1457          70       RL         66.0     9042   Pave   NaN      Reg
1458          20       RL         68.0     9717   Pave   NaN      Reg
1459          20       RL         75.0     9937   Pave   NaN      Reg

     LandContour Utilities LotConfig  … PoolArea PoolQC  Fence MiscFeature  \
0            Lvl    AllPub    Inside  …        0    NaN    NaN         NaN
1            Lvl    AllPub       FR2  …        0    NaN    NaN         NaN
2            Lvl    AllPub    Inside  …        0    NaN    NaN         NaN
3            Lvl    AllPub    Corner  …        0    NaN    NaN         NaN
4            Lvl    AllPub       FR2  …        0    NaN    NaN         NaN
…          …       …       …   …      …    …    …          …
1455         Lvl    AllPub    Inside  …        0    NaN    NaN         NaN
1456         Lvl    AllPub    Inside  …        0    NaN  MnPrv         NaN
1457         Lvl    AllPub    Inside  …        0    NaN  GdPrv        Shed
1458         Lvl    AllPub    Inside  …        0    NaN    NaN         NaN
1459         Lvl    AllPub    Inside  …        0    NaN    NaN         NaN

     MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
0          0      2    2008        WD         Normal     208500
1          0      5    2007        WD         Normal     181500
2          0      9    2008        WD         Normal     223500
3          0      2    2006        WD        Abnorml     140000
4          0     12    2008        WD         Normal     250000
…      …    …     …       …            …        …
1455       0      8    2007        WD         Normal     175000
1456       0      2    2010        WD         Normal     210000
1457    2500      5    2010        WD         Normal     266500
1458       0      4    2010        WD         Normal     142125
1459       0      6    2008        WD         Normal     147500

[1460 rows x 80 columns]>
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley',
       'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
       'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
       'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating', 'HeatingQC',
       'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
       'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
       'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond',
       'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'PoolQC', 'Fence', 'MiscFeature', 'MiscVal',
```

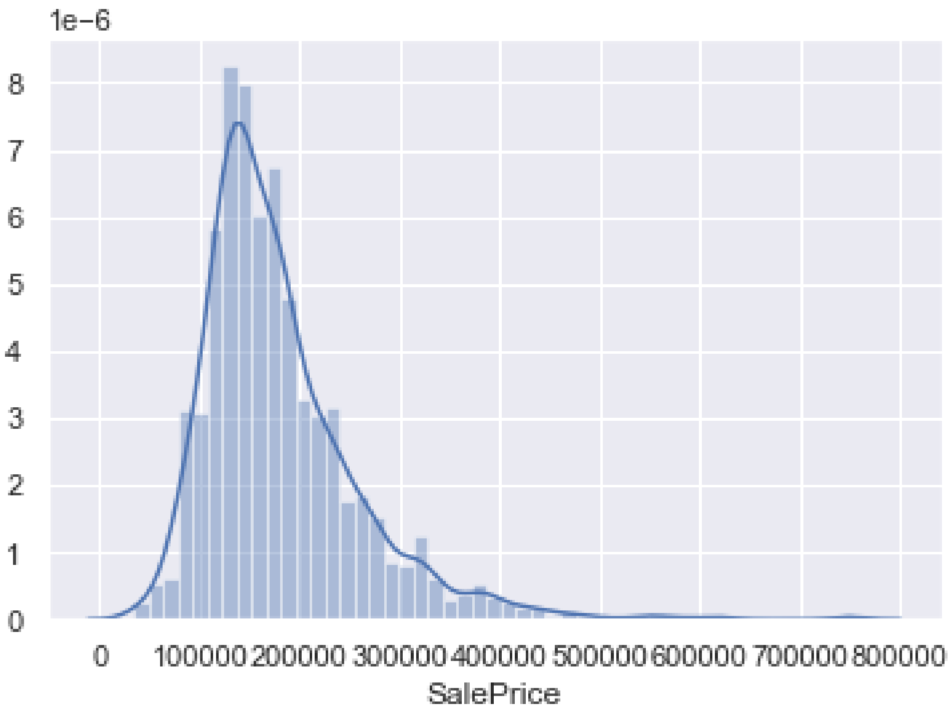
```
       'MoSold', 'YrSold', 'SaleType', 'SaleCondition', 'SalePrice'],
      dtype='object')
   MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0          60       RL         65.0     8450   Pave   NaN      Reg
1          20       RL         80.0     9600   Pave   NaN      Reg
2          60       RL         68.0    11250   Pave   NaN      IR1
3          70       RL         60.0     9550   Pave   NaN      IR1
4          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities LotConfig  … PoolArea PoolQC Fence MiscFeature  \
0         Lvl    AllPub    Inside  …        0    NaN   NaN         NaN
1         Lvl    AllPub       FR2  …        0    NaN   NaN         NaN
2         Lvl    AllPub    Inside  …        0    NaN   NaN         NaN
3         Lvl    AllPub    Corner  …        0    NaN   NaN         NaN
4         Lvl    AllPub       FR2  …        0    NaN   NaN         NaN

  MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
0       0      2    2008        WD         Normal     208500
1       0      5    2007        WD         Normal     181500
2       0      9    2008        WD         Normal     223500
3       0      2    2006        WD        Abnorml     140000
4       0     12    2008        WD         Normal     250000

[5 rows x 80 columns]
(1460, 80)
(1459, 80)
```

[36]:
```
# analyze target
target = 'SalePrice'
train[target].describe()
```

[36]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

[37]:
```
# use seaborn to plot SalePrice to see its distribution
sbn.distplot(train[target])
# result resembles normal distribution
```

[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94964f7fa0>
```

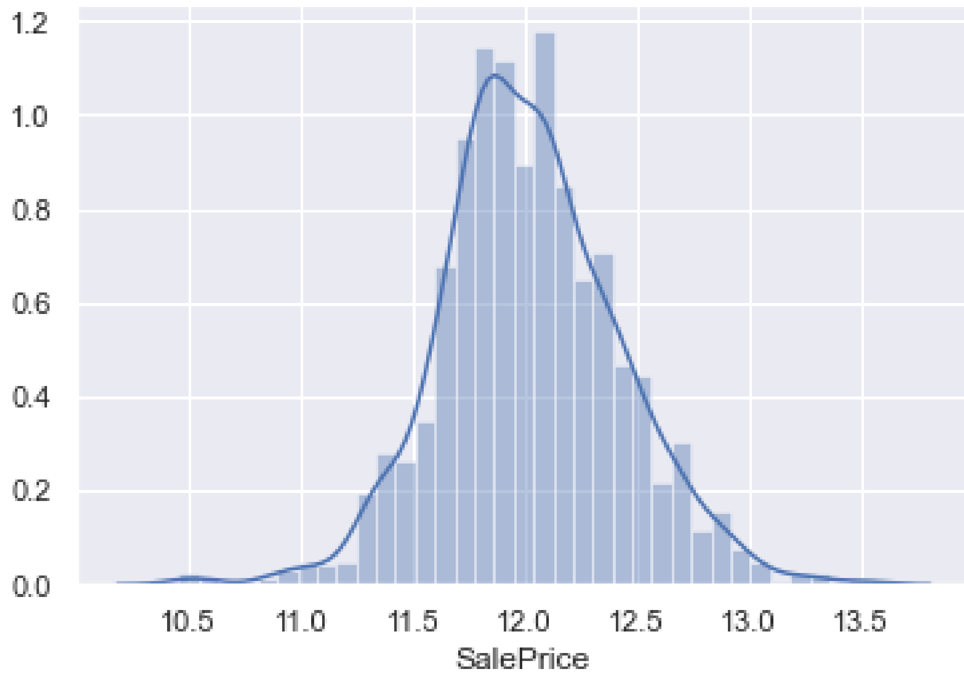


[38]:
```
# calculate kurtosis and skewness of the plot above
print ("skewness=",train[target].skew())
print ("kurtosis=",train[target].kurt())
```

```
skewness= 1.8828757597682129
kurtosis= 6.536281860064529
```

[39]:
```
# perform the logarithm operation to make the data distribution of target␣
↪function more in line with the standard normal distribution
train[target] = np.log1p(train[target])
sbn.distplot(train[target])
```

[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f94964f0df0>

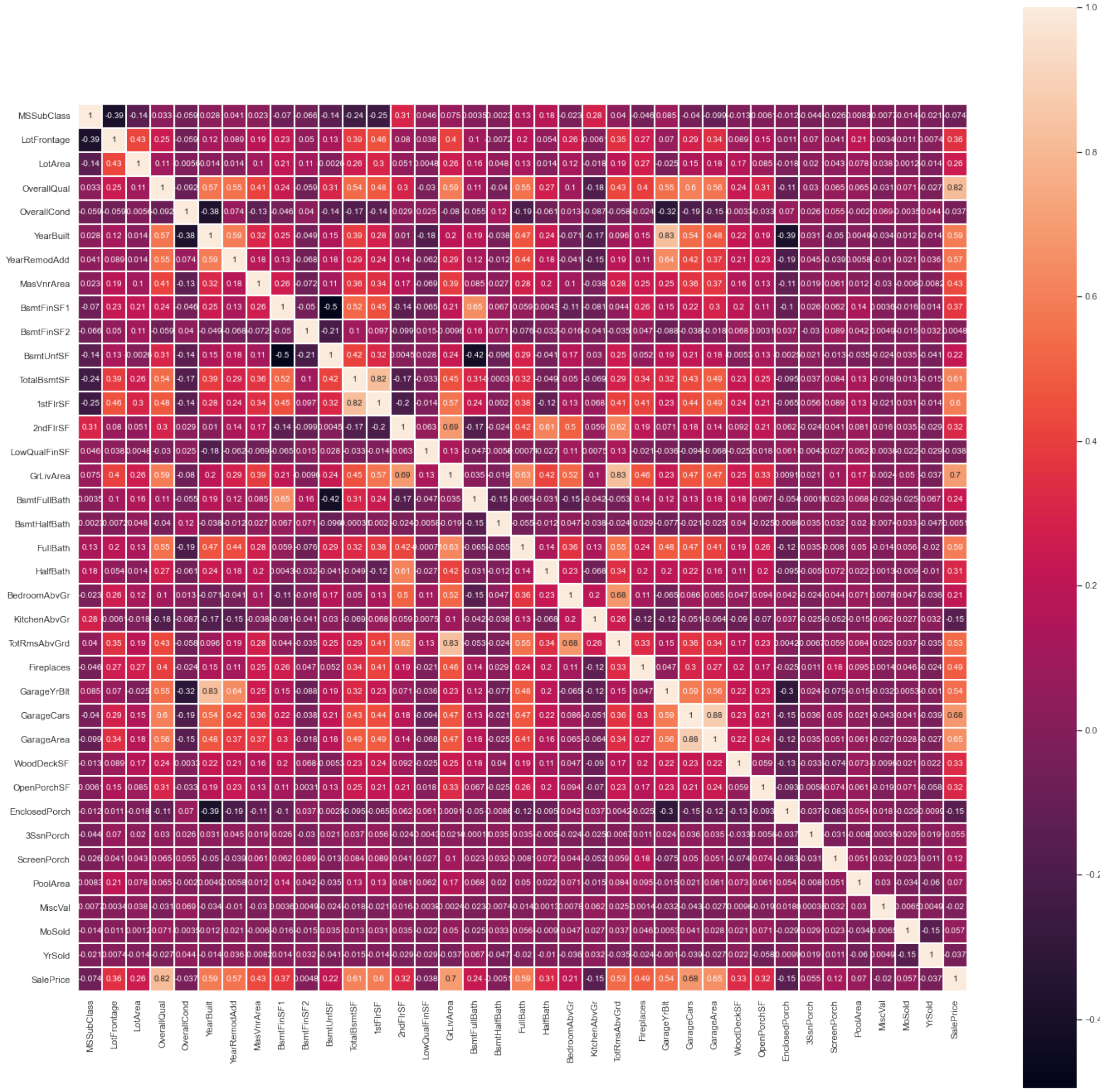


[40]:
```
# Data merging
all_data= pd.concat([train,test],axis=0,join='outer',ignore_index=True)
all_data.drop([target], axis=1, inplace=True)
all_data.shape
```

[40]: (2919, 79)

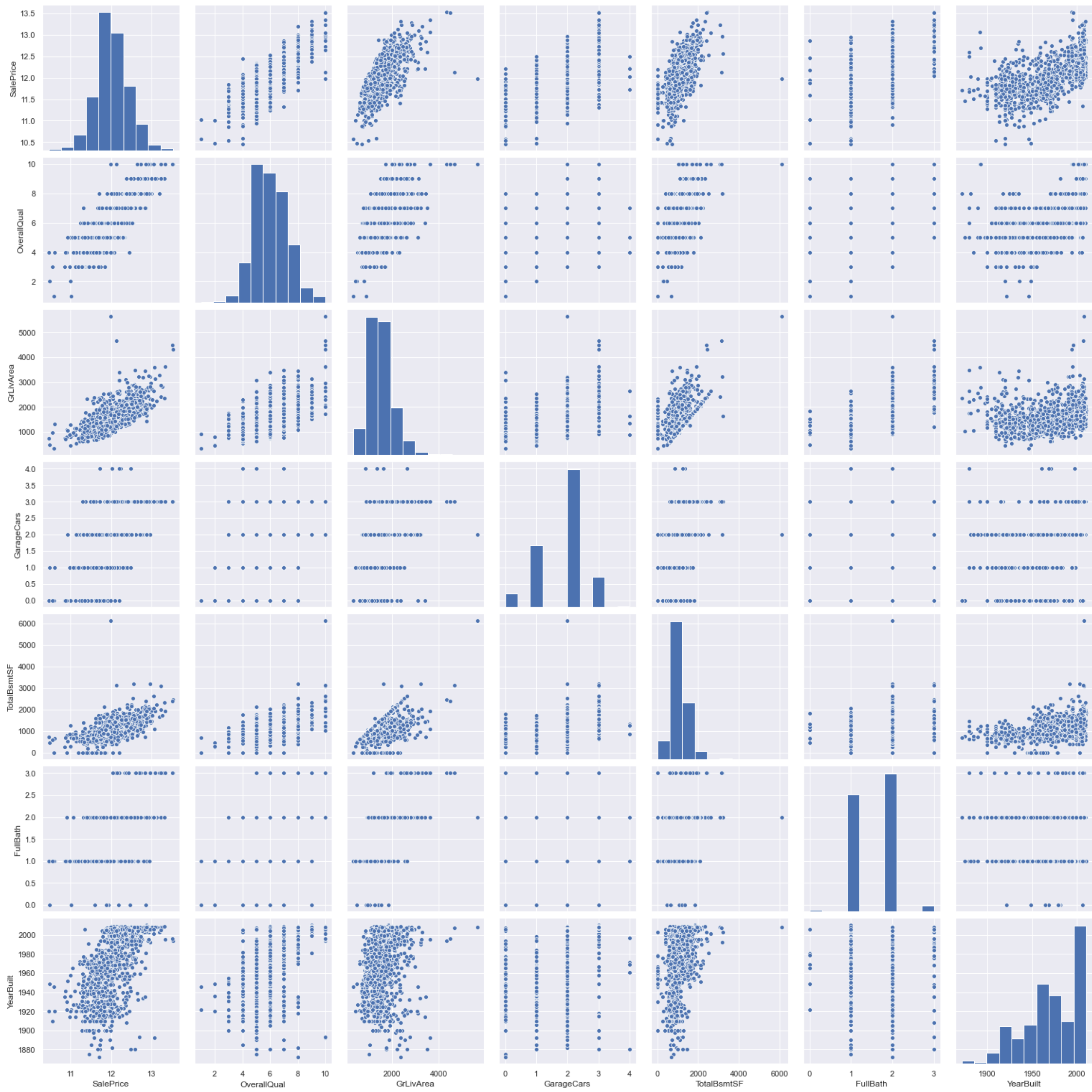
[41]:
```
# use sbn.heatmap() to conduct correlation analysis between features and target
plt.subplots(figsize=(24,24))
sbn.heatmap(train.corr(),square=True,linecolor='white',linewidths=1, annot=True)
```

[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93cc665bb0>



[42]:
```
# identified Top10 most related features from the heatmap above:'OverallQual',␣
↪'GrLivArea', 'GarageCars', 'GarageArea','TotalBsmtSF', '1stFlrSF',␣
↪'FullBath', 'YearBuilt', 'YearRemodAdd'
# these pairs of features can be merged into one:␣
↪'GarageCars'+'GarageArea' TotalBsmtSF'+'1stFlrSF' 'YearBuilt'+'YearRemodAdd'
```

[43]:
```
# observe the correlation tendency between each feature
sbn.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', ␣
↪'FullBath', 'YearBuilt']
sbn.pairplot(train[cols], height = 3)
plt.show()
# result shows that the six features are all positively correlated with target
```



[44]:
```python
# data processing & cleaning
all_data.shape
print('MSSubClass.type:', all_data.MSSubClass.dtypes)
all_data.MSSubClass=all_data.MSSubClass.astype(str)
all_data.MSSubClass.value_counts()
```

```
MSSubClass.type: int64
```

[44]:
```
20     1079
60      575
50      287
120     182
30      139
```

```
70     128
160    128
80     118
90     109
190     61
85      48
75      23
45      18
180     17
40       6
150      1
Name: MSSubClass, dtype: int64
```

[45]:

```python
# use pandas get_dummies to tranform categorical features into numerical␣
↪features
all_data = pd.get_dummies(all_data)

# fill the missing values with the mean from each column
cols_mean=all_data.mean()
all_data=all_data.fillna(cols_mean)
all_data.isnull().sum().sum()
all_data.head()
```

[45]:

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	\
0	65.0	8450	7	5	2003	2003	
1	80.0	9600	6	8	1976	1976	
2	68.0	11250	7	5	2001	2002	
3	60.0	9550	7	5	1915	1970	
4	84.0	14260	8	5	2000	2000	

	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	…	SaleType_ConLw	\
0	196.0	706.0	0.0	150.0	…	0	
1	0.0	978.0	0.0	284.0	…	0	
2	162.0	486.0	0.0	434.0	…	0	
3	0.0	216.0	0.0	540.0	…	0	
4	350.0	655.0	0.0	490.0	…	0	

	SaleType_New	SaleType_Oth	SaleType_WD	SaleCondition_Abnorml	\
0	0	0	1	0	
1	0	0	1	0	
2	0	0	1	0	
3	0	0	1	1	
4	0	0	1	0	

	SaleCondition_AdjLand	SaleCondition_Alloca	SaleCondition_Family	\
0	0	0	0	
1	0	0	0	

```
2                       0                      0                      0
3                       0                      0                      0
4                       0                      0                      0

   SaleCondition_Normal  SaleCondition_Partial
0                     1                      0
1                     1                      0
2                     1                      0
3                     0                      0
4                     1                      0

[5 rows x 303 columns]
```

[46]:
```
# standardize numeric variables
numer_cols=all_data.columns[all_data.dtypes != 'object']
numer_cols_mean=all_data.loc[:,numer_cols].mean()
numer_cols_std=all_data.loc[:,numer_cols].std()
all_data.loc[:,numer_cols]=(all_data.loc[:,numer_cols]-numer_cols_mean)/
 ↪numer_cols_std
```

[47]:
```
# separate training set and testing set
dummy_train = all_data.loc[train.index]
dummy_test = all_data.loc[test.index]
dummy_train.shape
```

[47]: (1460, 303)

[48]:
```
# do the modeling
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
```

[49]:
```
x_train = all_data[:train.shape[0]]
x_test = all_data[train.shape[0]:]
y_train = train.SalePrice
```

[50]:
```
# Use linear regression model based on least squares, and turn it into a␣
 ↪quadratic polynomial model through PolynomialFeatures

lr_model = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('lr', LinearRegression())
])
```

```
lr_model.fit(x_train, y_train)
lr_predict = cross_val_predict(lr_model, x_train, y_train, verbose=True,␣
 ↪n_jobs=-1, cv=3)
lr_mse = mean_squared_error(lr_predict, y_train)
lr_score = np.sqrt(lr_mse)
print("linear regression score: ", lr_score)

lr_model.fit(x_train, y_train)
lr_preds = np.expm1(lr_model.predict(x_test))
lr_solution = pd.DataFrame({"id":Id, target:lr_preds})
lr_solution.to_csv("lr_output.csv", index = False)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:   15.2s finished
```

```
linear regression score:  0.19982095204644656
```

[59]:
```
# Use Ridge Regression (regularized least squares-L2 regularization)

ridge_model = KernelRidge(degree=2, alpha=0.08, kernel='polynomial')
ridge_predict = cross_val_predict(ridge_model, x_train, y_train, cv=3,␣
 ↪verbose=True, n_jobs=-1)
ridge_mse = mean_squared_error(y_train, ridge_predict)
ridge_score = np.sqrt(ridge_mse)
print("ridge regression score: ", ridge_score)

ridge_model.fit(x_train, y_train)
ridge_preds = np.expm1(ridge_model.predict(x_test))
ridge_solution = pd.DataFrame({"id":Id, target:ridge_preds})
ridge_solution.to_csv("ridge_output.csv", index = False)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    0.1s finished
```

```
ridge regression score:  0.14268299026580086
```

[57]:
```
# Use Lasso Regression (regularized least squares-L1 regularization)

lasso_model = Lasso(alpha=0.0008, random_state=1, max_iter=5000)
lasso_predict = cross_val_predict(lasso_model, x_train, y_train, cv=3,␣
 ↪verbose=True, n_jobs=-1)
lasso_mse = mean_squared_error(lasso_predict, y_train)
lasso_score = np.sqrt(lasso_mse)
print("lasso regression score: ", lasso_score)

lasso_model.fit(x_train, y_train)
lasso_preds = np.expm1(lasso_model.predict(x_test))
lasso_solution = pd.DataFrame({"id":Id, target:lasso_preds})
```

```python
lasso_solution.to_csv("lasso_output.csv", index = False)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    0.1s finished
```

```
lasso regression score:  0.14412691102769676
```

[53]:
```python
# Briefly describe your findings from training regularized and unregularized␣
↪models.

# 1.unregularized models - linear regression (least squares):
#   1) exists overfitting problem.

# 2.regularized models - ridge regression & lasso regression:
# 1) when $\alpha$ is setted to be small enough, it reduces risk of overfitting
#    and also maintains the complexity of the model.
# 2) L1 regularization is useful in alleviating the overfitting problem,
#    but it may also cause the loss of precision and the problem of␣
↪insufficient generalization ability.
# 3) also exist a certain degree of overfitting.
# 4) show better performance than the unregularized model.
# 5) the scores of Lasso (L1) and Ridge (L2) appear to be very close to each␣
↪other.
```

[54]:
```python
# The result is submitted to Kaggle.

# Kaggle Link: https://www.kaggle.com/scarletty
```