```
In [1]: import numpy as np
        import random as rnd
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [2]: # importing data
        df = pd.read_excel('/Users/orland/Documents/ML_homework_1/AmesHousing.xls')
        df.head()
```

Out[2]:

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition	SalePrice
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	NaN	0	5	2010	WD	Normal	215000
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal	105000
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal	172000
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal	244000
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal	189900

5 rows × 82 columns

1. Visualize the univariate distribution of each continuous input variable, and the distribution of the target variable. Write down anything you find worth noting. Is there something that might require special treatment?

```
In [3]: # Visualizing distribution og each continuous variables and target variable
        df_contd = df[['Lot Frontage', 'Lot Area', 'Mas Vnr Area', 'BsmtFin SF 1', 'BsmtFin SF 2',
                       'Bsmt Unf SF', 'Total Bsmt SF', '1st Flr SF', '2nd Flr SF', 'Low Qual Fin SF',
                       'Gr Liv Area', 'Garage Area', 'Wood Deck SF', 'Open Porch SF',
                       'Enclosed Porch', '3Ssn Porch', 'Screen Porch', 'Pool Area', 'Misc Val', 'SalePrice']]
        print(df_contd.shape)
        df_contd.hist(figsize = (15,15))
```

(2930, 20)

Out[3]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a17f1cf70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a60fc1b>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a60f4aee10>,
        ...
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a61a6891b>]],
       dtype=object)

From histograms it looks there are several outlier in many input variable. The special treatment needed to move those outlier to make distribution normal.

```
In [4]: ########## DELET
        #from scipy import stats
        #z = np.abs(stats.zscore(df_contd))
        #print(z)
        #threshold = 3
        #df_contd_o = df_contd[(z < 3).all(axis=1)]

        #Interquartile fuc
        Q1 = df_contd.quantile(0.25)
        Q3 = df_contd.quantile(0.75)
        IQR = Q3 - Q1
        print(IQR)
        df_contd = df_contd[~((df_contd < (Q1 - 1.5 * IQR)) |(df_contd > (Q3 + 1.5 * IQR))).any(axis=1)]
        df_contd.hist(figsize = (15,15))
```

```
Lot Frontage        58.00
Lot Area         7563.25
Mas Vnr Area      256.00
BsmtFin SF 1      925.00
BsmtFin SF 2        0.00
Bsmt Unf SF       898.25
Total Bsmt SF     829.00
1st Flr SF        767.00
2nd Flr SF        857.00
Low Qual Fin SF     0.00
Gr Liv Area       653.00
Garage Area       453.00
Wood Deck SF      213.00
Open Porch SF       0.00
Enclosed Porch    104.00
3Ssn Porch         16.00
Screen Porch        0.00
Pool Area           0.00
Misc Val            0.00
SalePrice      130083.00
dtype: float64
```

(1936, 20)

Out[4]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a61fb765c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a61fe16c0>,
        ...
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a61f2567812>]],
       dtype=object)

2. Visualize the dependency of the target on each continuous feature via scatter plot.

3. Split data in training and test set. Do not use the test-set unless for a final evaluation in Step 5. For each categorical variable, cross-validate a Linear Regression model using just this variable (one-hot-encoded). Visualize the relationship of the categorical variables that provide the best R^2 value with the target.

Visualize the relationship of the categorical variables that provide the best R^2 value with the target

4. Use ColumnTransformer and pipeline to encode categorical variables. Evaluate Linear Regression (OLS, Ridge, Lasso and ElasticNet using cross-validation with the default parameters. Does scaling the data (within the pipeline) with StandardScaler help?

ElasticNet Regression using Cross Validation

```
pipeline_4 = Pipeline([('transformer_4',ohc),
                       ('model_4',ElasticNet())])

pipeline_4.fit(X_train, y_train)
y_predict = pipeline_4.predict(X_train)
R2 = r2_score(y_train, y_predict)
print('The R^2 value for ElasticNet Regression',R2)

The R^2 value for ElasticNet Regression 0.8273762212992231
```

5. Tune the parameters of the models using GridSearchCV. Do the results improve? Visualize the dependence of the validation score on the parameters for Ridge, Lasso and ElasticNet.

Tuning Parameters of Ridge Regression

```
from sklearn.model_selection import GridSearchCV
# prepare a range of alpha values to test
alphas = np.array([100, 100,10,1,0.1,0.01,0.001,0])
# create and fit a ridge regression model, testing each alpha
grid = GridSearchCV(estimator=Ridge(), param_grid=dict(alpha=alphas))
pipeline_5 = Pipeline([('transformer',ohc),
                       ('model',grid)
                       ])

pipeline_5.fit(X_train, y_train)
y_predict = pipeline_5.predict(X_train)
R2 = r2_score(y_train,y_predict)
print(R2)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
print(grid.cv_results_)
```

Visualization the dependence of the validation score on the parameters for Ridge Regression

```
cvres_val_score = np.array([0.82429369, 0.8259969, 0.83642543, 0.82556049, 0.82882266,
                            0.79557742, 0.79825927, 0.79825926, 0.79834404])
parameters = np.array([100, 100, 10, 1, 0.1, 0.01, 0.001, 0.0001, 0])
#Curve fitting
result = np.polyfit(parameters,cvres_val_score,1)
plt.plot(np.log(parameters),cvres_val_score,'o')
```


Tuning Parameters of Lasso Regression

```
# prepare a range of alpha values to test
alphas = np.array([100, 100,10,1,0.1,0.01,0.001,0])
# create and fit a ridge regression model, testing each alpha
grid = GridSearchCV(estimator=Lasso(), param_grid=dict(alpha=alphas))
pipeline_5 = Pipeline([('transformer',ohc),
                       ('model',grid)])

pipeline_5.fit(X_train, y_train)
y_predict = pipeline_5.predict(X_train)
R2 = r2_score(y_train,y_predict)
print(R2)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
print(grid.cv_results_)
```

Visualization the dependence of the validation score on the parameters for Lasso Regression

```
cvres_val_score = np.array([0.81556087, 0.81558674, 0.81556001, 0.80181517, 0.81022949,
                            0.80188883, 0.80142939, 0.80142939, 0.80122949, 0])
parameters = np.array([100, 100, 10, 1, 0.1, 0.01, 0.001, 0])
#Curve fitting
result = np.polyfit(parameters,cvres_val_score,'o')
plt.plot(np.log(parameters),cvres_val_score,'o')
plt.xlabel('Parameters(log)')
plt.ylabel('Validation Score')
plt.title('Dependence of Validation Score on Parameters')
```


Tuning Parameters of ElasticNet Regression

```
# prepare a range of alpha values to test
alphas = np.array([100, 100,10,1,0.1,0.01,0.001,0])
# create and fit a ridge regression model, testing each alpha
grid = GridSearchCV(estimator=ElasticNet(), param_grid=dict(alpha=alphas))
pipeline_5 = Pipeline([('transformer',ohc),
                       ('model',grid)])

pipeline_5.fit(X_train, y_train)
y_predict = pipeline_5.predict(X_train)
R2 = r2_score(y_train,y_predict)
print(R2)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
print(grid.cv_results_)
```

Visualization the dependence of the validation score on the parameters for ElasticNet Regression

```
cvres_val_score = np.array([0.81612562, 0.1799597, 0.4761254, 0.4512471, 0.4511217, 0.456042 ,
                            0.3203411, 0.867398 , 0.8010215])
parameters = np.array([100, 100, 10, 1, 0.1, 0.01, 0.001, 0])
#Curve fitting
result = np.polyfit(parameters,cvres_val_score,'o')
plt.plot(np.log(parameters),cvres_val_score,'o')
plt.xlabel('Parameters(log)')
plt.ylabel('Validation Score')
plt.title('Dependence of Validation Score on Parameters')
```


6. Visualize the coefficients of the resulting models. Do they agree on which features are important? Evaluate the model performance on the test set.

```
# Coefficients for Ridge Regression
pipeline = Pipeline([('transformer',ohc),
                     ('model',Ridge(alpha=1))])
pipeline.fit(X_train, y_train)
Ridge_coefficients = pipeline['model'].coef_
n, bins, patches = plt.hist(Ridge_coefficients)
```



```
# Coefficients for Lasso Regression
pipeline = Pipeline([('transformer',ohc),
                     ('model',Lasso(alpha=1))])
pipeline.fit(X_train, y_train)
Ridge_coefficients = pipeline['model'].coef_
n, bins, patches = plt.hist(Ridge_coefficients)
```



```
# Coefficients for ElasticNet Regression
pipeline = Pipeline([('transformer',ohc),
                     ('model',ElasticNet(alpha=1))])
pipeline.fit(X_train, y_train)
ElasticNet_coefficients = pipeline['model'].coef_
n, bins, patches = plt.hist(ElasticNet_coefficients)
```


The absolute value of coefficient of regression suggest its influence on the target. Greater the absolute value of coefficient, higher influence on the target. Some features have positive effect (positive coefficients) while others have negative effect (negative coefficients). In our case, the three distribution of coefficients of the three different models shows that only few features which have larger absolute value (away from zero) are important and have greater effects on the target.

Evaluate the model performance on the test set.

```
df_test = df_test[df_test['color != 'unknown']
df_test = df_test.dropna()
```

```
In [16]: X_test = df_test.drop('SalePrice',1,axis=1)
         y_test = df_test['SalePrice']
```

Linear Regression Model

```
In [17]: y_predict = pipeline_1.predict(X_test)
         R2 = r2_score(y_test,y_predict)
         print('The R^2 coefficient ', R2)
         The R^2 coefficient  0.9537352594187803
```

Ridge Regression Model

```
In [18]: y_predict = pipeline_2.predict(X_test)
         R2 = r2_score(y_test,y_predict)
         print('The R^2 coefficient ', R2)
         The R^2 coefficient  0.9163312721361228
```

Lasso Regression Model

```
In [19]: y_predict = pipeline_3.predict(X_test)
         R2 = r2_score(y_test,y_predict)
         print('The R^2 coefficient ', R2)
         The R^2 coefficient  0.9117563367353716
```

ElasticNet Regression Model

```
In [20]: y_predict = pipeline_4.predict(X_test)
         R2 = r2_score(y_test,y_predict)
         print('The R^2 coefficient ', R2)
         The R^2 coefficient  0.8773336561533242
```

```
In [ ]:
```