Loading the data

Used house prediction because previous dataset of Airplane Crash had most of the values from 0 to 10.

```
# all the libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import missingno as msno

from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import itertools
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()

fetch_housing_data()
import pandas as pd
```

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)

houseprediction = load_housing_data()
houseprediction.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hou
0	-122.23	37.88	41.0	880.0	129.0	322.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	
3	-122.25	37.85	52.0	1274.0	235.0	558.0	
4	-122.25	37.85	52.0	1627.0	280.0	565.0	

Data Exploration

```
houseprediction.shape
```

```
(20640, 10)
```

```
houseprediction.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	pc
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	206
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	14
std	2.003532	2.135952	12.585558	2181.615252	421.385070	11
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	7
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	11
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	17
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	356

```
houseprediction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
```

```
data_drop = houseprediction.drop(['longitude','latitude','total_bedrooms','population','house
```

```
data_drop.dtypes
```

```
housing_median_age    float64
total_rooms           float64
median_income         float64
median_house_value    float64
dtype: object
```

Checking if there are any missing value. If true, missing values are present. If false not present

```
houseprediction.isna().values.any()
```

```
True
```

```
data_drop.info
```

```
<bound method DataFrame.info of        housing_median_age  total_rooms  median_income  r
0                    41.0        880.0         8.3252          452600.0
1                    21.0       7099.0         8.3014          358500.0
2                    52.0       1467.0         7.2574          352100.0
3                    52.0       1274.0         5.6431          341300.0
4                    52.0       1627.0         3.8462          342200.0
...                   ...          ...            ...               ...
20635                25.0       1665.0         1.5603           78100.0
20636                18.0        697.0         2.5568           77100.0
20637                17.0       2254.0         1.7000           92300.0
20638                18.0       1860.0         1.8672           84700.0
20639                16.0       2785.0         2.3886           89400.0

[20640 rows x 4 columns]>
```

```
data_drop.isnull().sum()
```

```
housing_median_age    0
total_rooms           0
median_income         0
median_house_value    0
dtype: int64
```

```
remove_data = houseprediction[houseprediction.isnull().any(axis=1)] # dropping missing values
remove_data.shape
```

```
(207, 10)
```

```
# Use the sklearn imputer class, select median as method
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

```
[(x, y) for x, y in zip(houseprediction.isna().sum(), houseprediction.isna().sum().index) if
```

```
[(207, 'total_bedrooms')]
```

Total bedrooms has missing values - fill median values for it

```
average = houseprediction["total_bedrooms"].median()
houseprediction["total_bedrooms"].fillna(average, inplace=True)
```

```
houseprediction.isna().values.any()
```

```
False
```

Checking corrolation between the dependent and independent variables

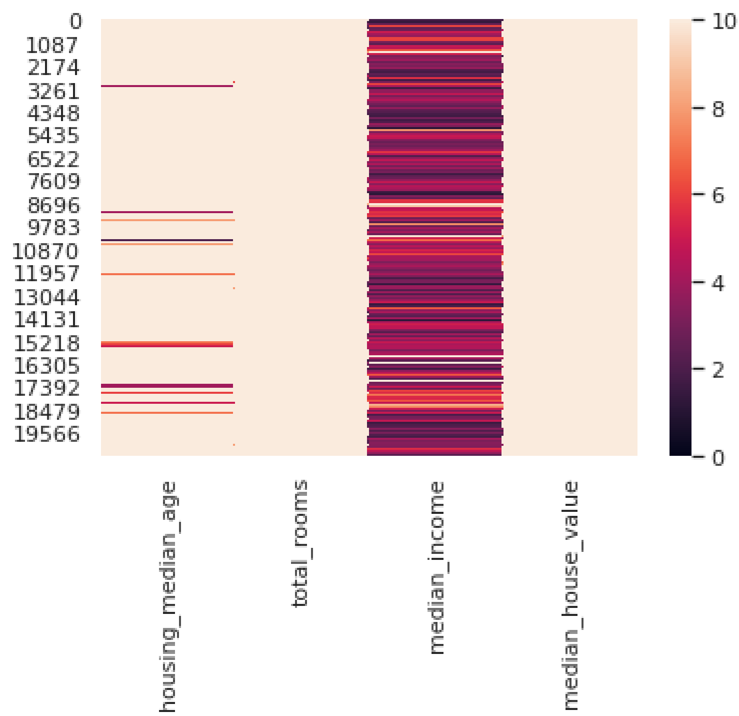
```
data_drop.corr()
```

	housing_median_age	total_rooms	median_income	median_house_value
housing_median_age	1.000000	-0.361262	-0.119034	0.105623
total_rooms	-0.361262	1.000000	0.198050	0.134153
median_income	-0.119034	0.198050	1.000000	0.688075
median_house_value	0.105623	0.134153	0.688075	1.000000

```
correlation = data_drop.corr()
correlation.style.background_gradient()
correlation.style.background_gradient().set_precision(3)
#precision(3) is set for 2 decimal palces.
```

	housing_median_age	total_rooms	median_income	median_house_value
housing_median_age	1.000	-0.361	-0.119	0.106
total_rooms	-0.361	1.000	0.198	0.134
median_income	-0.119	0.198	1.000	0.688
median_house_value	0.106	0.134	0.688	1.000

```
ax = sns.heatmap(data_drop, vmin=0, vmax=10)
```




```
data_drop.describe()
```

	housing_median_age	total_rooms	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	28.639486	2635.763081	3.870671	206855.816909
std	12.585558	2181.615252	1.899822	115395.615874
min	1.000000	2.000000	0.499900	14999.000000
25%	18.000000	1447.750000	2.563400	119600.000000
50%	29.000000	2127.000000	3.534800	179700.000000
75%	37.000000	3148.000000	4.743250	264725.000000
max	52.000000	39320.000000	15.000100	500001.000000

```
X = data_drop[['housing_median_age','total_rooms','median_income']]
Y = data_drop['median_house_value']
```

Training and splitting data

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
```

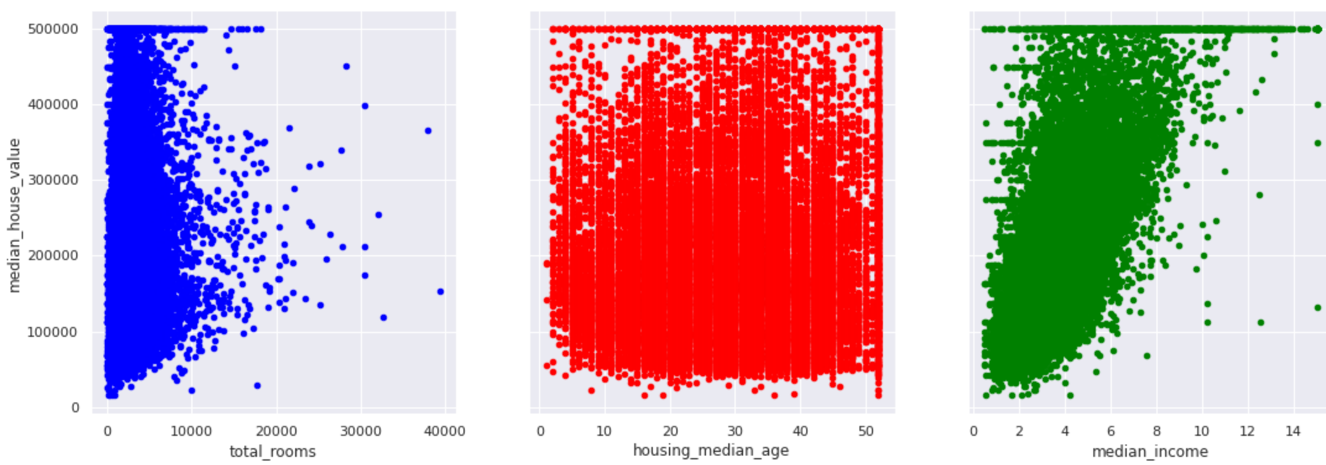
```
x_train.shape, x_test.shape, Y.shape,y_train.shape, y_test.shape
```

```
((14448, 3), (6192, 3), (20640,), (14448,), (6192,))
```

Data Visualization

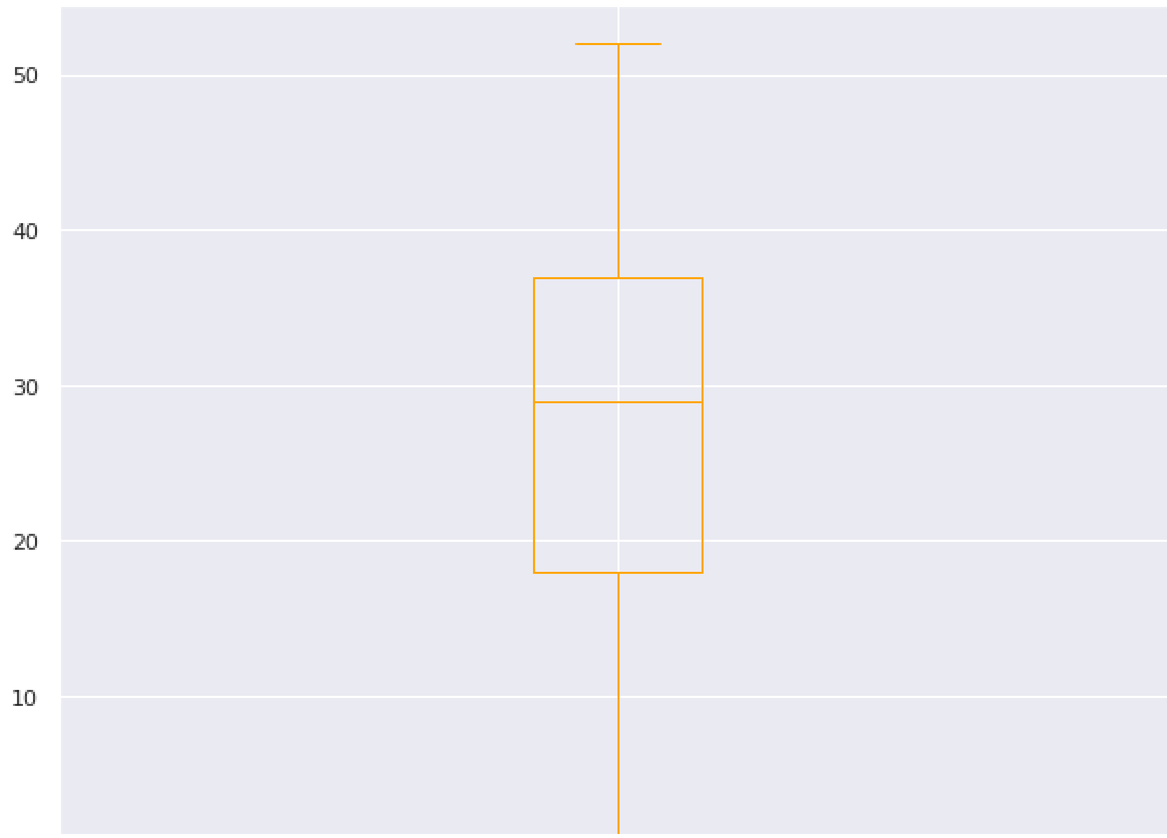
```
#visualize the relationship between the features and the response using scatterplots
fig, axs = plt.subplots(1, 3, sharey=True)
data_drop.plot(kind='scatter', x='total_rooms', y='median_house_value', ax=axs[0], figsize=(1
data_drop.plot(kind='scatter', x='housing_median_age', y='median_house_value', ax=axs[1], col
data_drop.plot(kind='scatter', x='median_income', y='median_house_value', ax=axs[2], color='g
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6523a0a940>
```




```
x_train.boxplot('housing_median_age', figsize=(10,8),grid=True, color='orange')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6523897ac8>

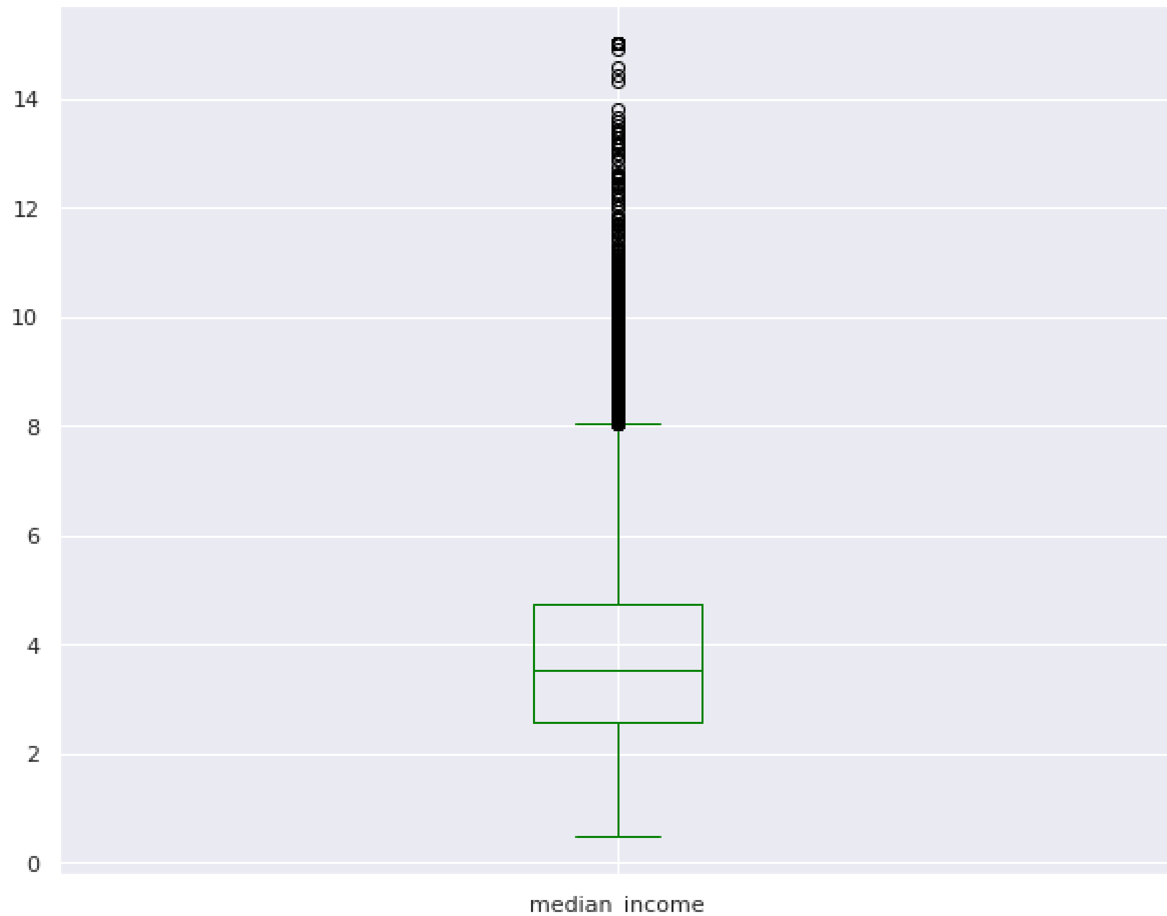



```
x_train.boxplot('total_rooms', figsize=(10,8),grid=True, color='red')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6522055be0>
```

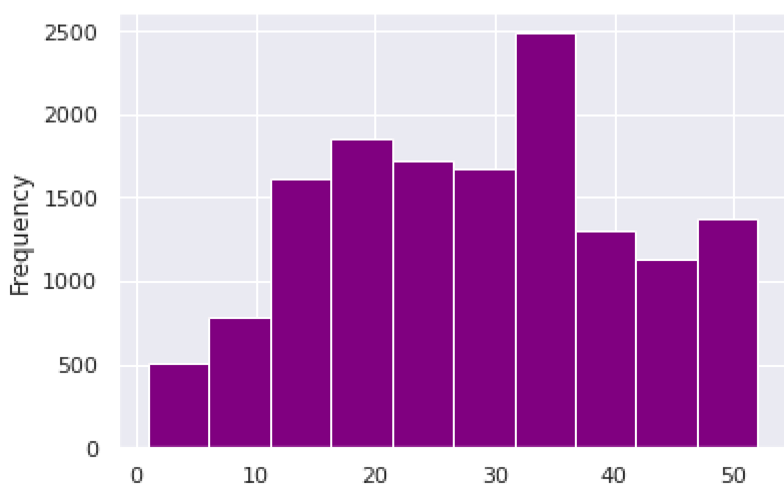
```
x_train.boxplot('median_income', figsize=(10,8),grid=True, color='green')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6521fcb4a8>
```



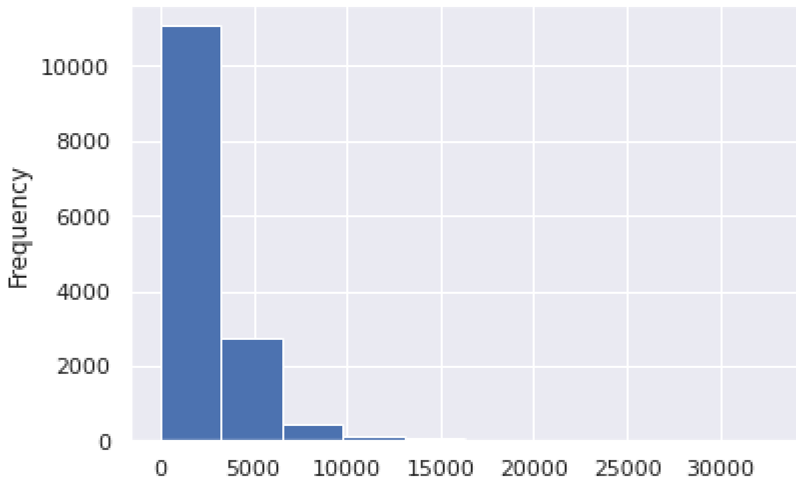

```
x_train['housing_median_age'].plot(kind='hist',color='purple')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6521fb3940>
```



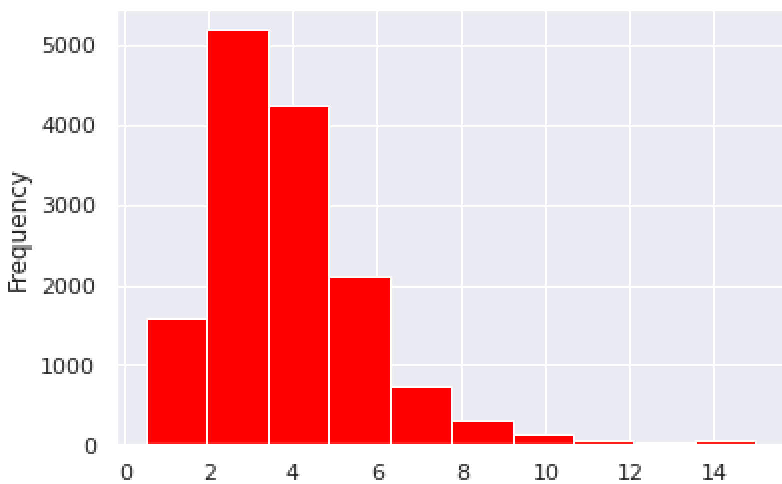

```
x_train['total_rooms'].plot(kind='hist')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6521f63c88>




```
x_train['median_income'].plot(kind='hist',color='red') # Distributed normally towards median_
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6521ea5860>




```
x_train[x_train.isnull().any(axis=1)]
```

housing_median_age	total_rooms	median_income

Standardise data

Note that once the data is normalize, we cannot further make a summary statistics. For that reason we have assigned different names for normalised data (norm_train or norm_test)

```
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
norm_train = std.fit_transform(x_train)
norm_test = std.fit_transform(x_test)
```

Transformed data

```
norm_train
```

```
array([[ 0.50935748, -0.11324158,  0.13350629],
       [-0.67987313, -0.21356615, -0.53221805],
       [-0.36274497, -0.48263943,  0.1709897 ],
       ...,
       [ 0.58863952, -0.25147682, -0.49478713],
       [-1.07628333,  0.42999055,  0.96717102],
       [ 1.85715216,  0.73096426, -0.68320166]])
```

```
norm_test
```

```
array([[-0.30267793, -0.49877093, -1.15209909],
       [ 0.09672999,  0.14776196, -0.70179147],
       [ 1.85412486,  0.54656215, -0.19920134],
       ...,
       [-1.02161219,  2.403883  , -0.18230355],
       [-1.5009017 ,  2.9631924 , -0.10920338],
       [-0.94173061,  1.19938952, -0.42597077]])
```

Simple Linear Regression

```
numInstances = (data_drop.shape[0])
print(numInstances)
X = np.random.rand(numInstances,1).reshape(-1,1)
y_true = -3*X + 1
y = y_true + np.random.normal(size=numInstances).reshape(-1,1)

plt.scatter(X, y,  color='black')
plt.plot(X, y_true, color='blue', linewidth=3)
plt.title('True function: y = -3X + 1')
plt.xlabel('X')
plt.ylabel('y')
```

```
20640
Text(0, 0.5, 'y')
```

Perform Regression with one Independent Variable


```
slr = data_drop.median_income.values
slr = slr.reshape(-1,1)
```

```
from sklearn.model_selection import train_test_split
X_train_rp,X_test_rp,Y_train_rp,Y_test_rp = train_test_split(slr,Y,test_size=0.3, random_stat
```

```
from sklearn.linear_model import LinearRegression
modelslr = LinearRegression()
modelslr.fit(X_train_rp,Y_train_rp) # model fitting for the training set
```

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

R-squared on the training dataset

```
modelslr.score(X_train_rp,Y_train_rp)
```

```
0.48062719927664055
```

```
modelslr.score(X_test_rp,Y_test_rp)
# R-squared value on the test set
```

```
0.4564966485656325
```

Training dataset has slighter better score than Test dataset

```
Y_pred_rp = modelslr.predict(X_test_rp)
```

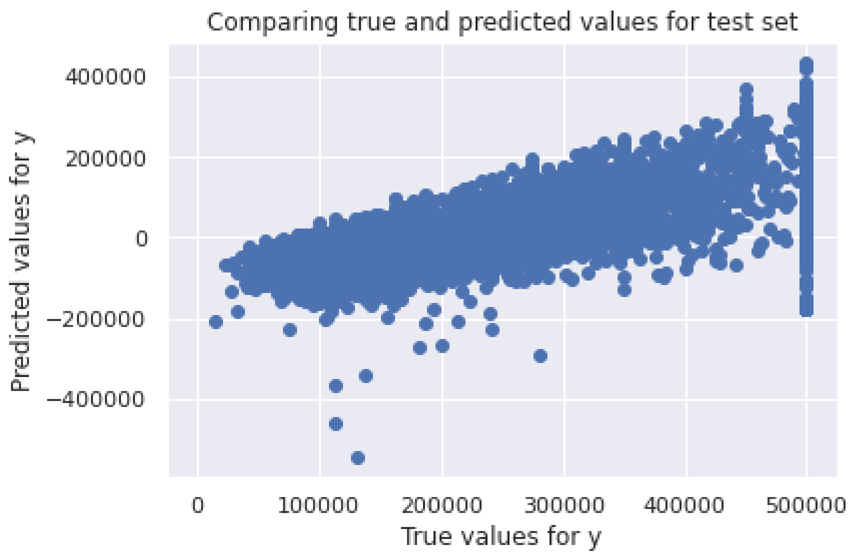
```
Y_pred_rp.mean()
```

```
207389.01699770137
```

```
from sklearn.metrics import mean_squared_error, r2_score
# Comparing true versus predicted values
plt.scatter(Y_test_rp, Y_test_rp - Y_pred_rp, color='b')
x = np.random.rand(30)
plt.plot(x, x*0 )
plt.title('Comparing true and predicted values for test set')
plt.xlabel('True values for y')
```

```
plt.ylabel('Predicted values for y')

# Model evaluation
print("Root mean squared error = %.4f" % np.sqrt(mean_squared_error(Y_test_rp, Y_pred_rp)))
print('R-squared = %.4f' % r2_score(Y_test_rp, Y_pred_rp))
```

```
Root mean squared error = 85124.5362
R-squared = 0.4565
```




```
from sklearn.metrics import mean_squared_error
print(np.sqrt(mean_squared_error(Y_test_rp,Y_pred_rp)))
```

```
85124.53624841144
```

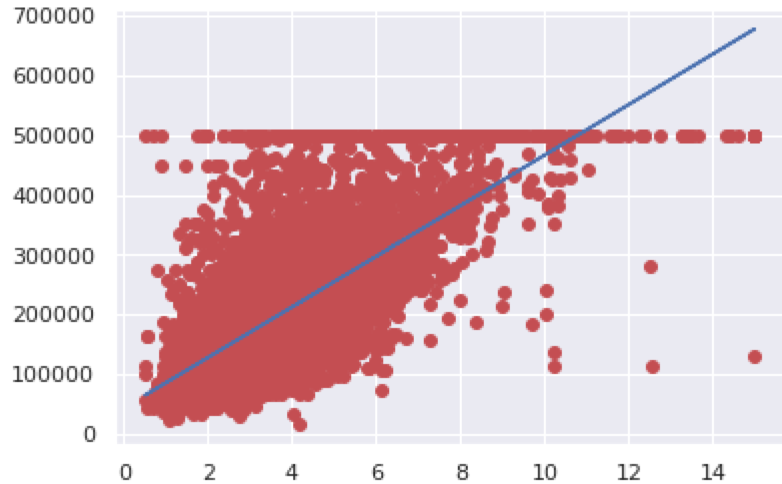
Plotting the Least Squares Line

```
#Ploting on train data Set
plt.scatter(X_train_rp,Y_train_rp,color='r')
plt.plot(X_train_rp,modelslr.predict(X_train_rp), color='b')
```

```
[<matplotlib.lines.Line2D at 0x7f768525a978>]
```

```
#Ploting on Test data Set
plt.scatter(X_test_rp,Y_test_rp,color='r')
plt.plot(X_test_rp,modelslr.predict(X_test_rp), color='b')
```

```
[<matplotlib.lines.Line2D at 0x7f7684e426a0>]
```




```
modelslr
```

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

Multiple Linear Regression

```
#split data
numTrain = 20   # number of training instances
numTest = numInstances - numTrain

X_train = X[:-numTest]
X_test = X[-numTest:]
y_train = y[:-numTest]
y_test = y[-numTest:]
```

Summary of training data set

```
import statsmodels.api as sm
xstat = sm.add_constant(norm_train)
est = sm.OLS(y_train,xstat)
statfit = est.fit()
print(statfit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     median_house_value   R-squared:                       0.514
Model:                            OLS   Adj. R-squared:                  0.514
Method:                 Least Squares   F-statistic:                     5096.
Date:                Sun, 05 Apr 2020   Prob (F-statistic):               0.00
Time:                        01:05:43   Log-Likelihood:            -1.8374e+05
No. Observations:               14448   AIC:                         3.675e+05
Df Residuals:                   14444   BIC:                         3.675e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.069e+05    671.261    308.262      0.000    2.06e+05    2.08e+05
x1          2.504e+04    721.211     34.715      0.000    2.36e+04    2.65e+04
x2          8834.2420    730.735     12.090      0.000    7401.907    1.03e+04
x3          8.085e+04    685.723    117.903      0.000    7.95e+04    8.22e+04
==============================================================================
Omnibus:                     2998.715   Durbin-Watson:                   1.975
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7420.517
Skew:                           1.147   Prob(JB):                         0.00
Kurtosis:                       5.659   Cond. No.                         1.53
==============================================================================
```

Summary of the model on the Test set

```
import statsmodels.api as sm
xstat = sm.add_constant(norm_test)
est = sm.OLS(y_test,xstat)
statfit = est.fit()
print(statfit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     median_house_value   R-squared:                       0.513
Model:                            OLS   Adj. R-squared:                  0.512
Method:                 Least Squares   F-statistic:                     2169.
Date:                Sun, 05 Apr 2020   Prob (F-statistic):               0.00
```

```
#Defined y_predict
pred_y = reg.predict(norm_test)
```

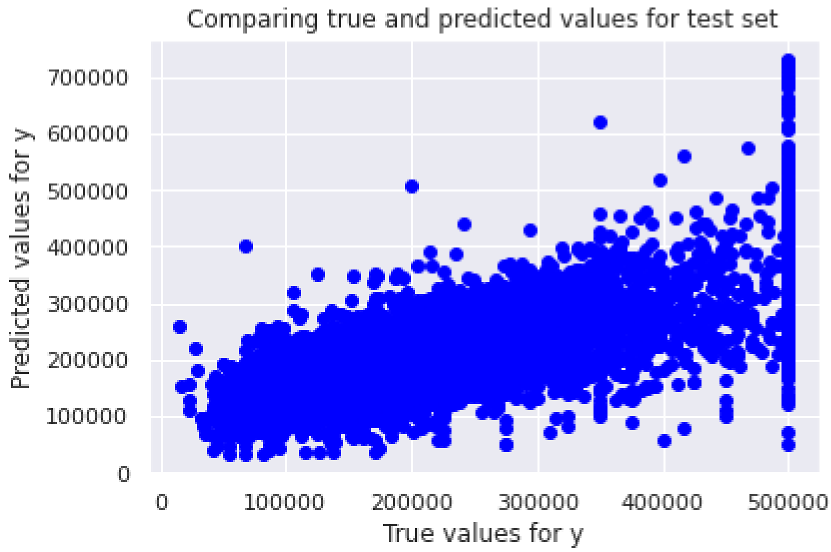
```
Df Model:                           3
```

Evaluate model on test data set

```
                 coef    std err          t      P>|t|      [0.025      0.975]
```

```
# Comparing true versus predicted values
plt.scatter(y_test, pred_y, color='blue')
plt.title('Comparing true and predicted values for test set')
plt.xlabel('True values for y')
plt.ylabel('Predicted values for y')
```

```
Text(0, 0.5, 'Predicted values for y')
```




```
from sklearn.metrics import mean_squared_error, r2_score
```

```
pred_y
```

```
array([101793.6345172 , 153912.1246265 , 242069.08675016, ...,
       187843.23022605, 186694.36205061, 159502.22858421])
```

```
pred_y.mean()
```

```
206923.96089424143
```

```
#Score for training dataset
reg.score(norm_train,y_train)
```

```
0.5141931111690983
```

```
#Score for testing dataset
reg.score(norm_test, y_test)
```

```
0.5124347436482521
```

```
#Model prediction
from sklearn.metrics import mean_squared_error
print(np.sqrt(mean_squared_error(y_test,pred_y)))
```

```
79996.80957002078
```

```
regr = LinearRegression()
regr.fit(norm_train,y_train)
```

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

```
pred_y = regr.predict(X_test)
```

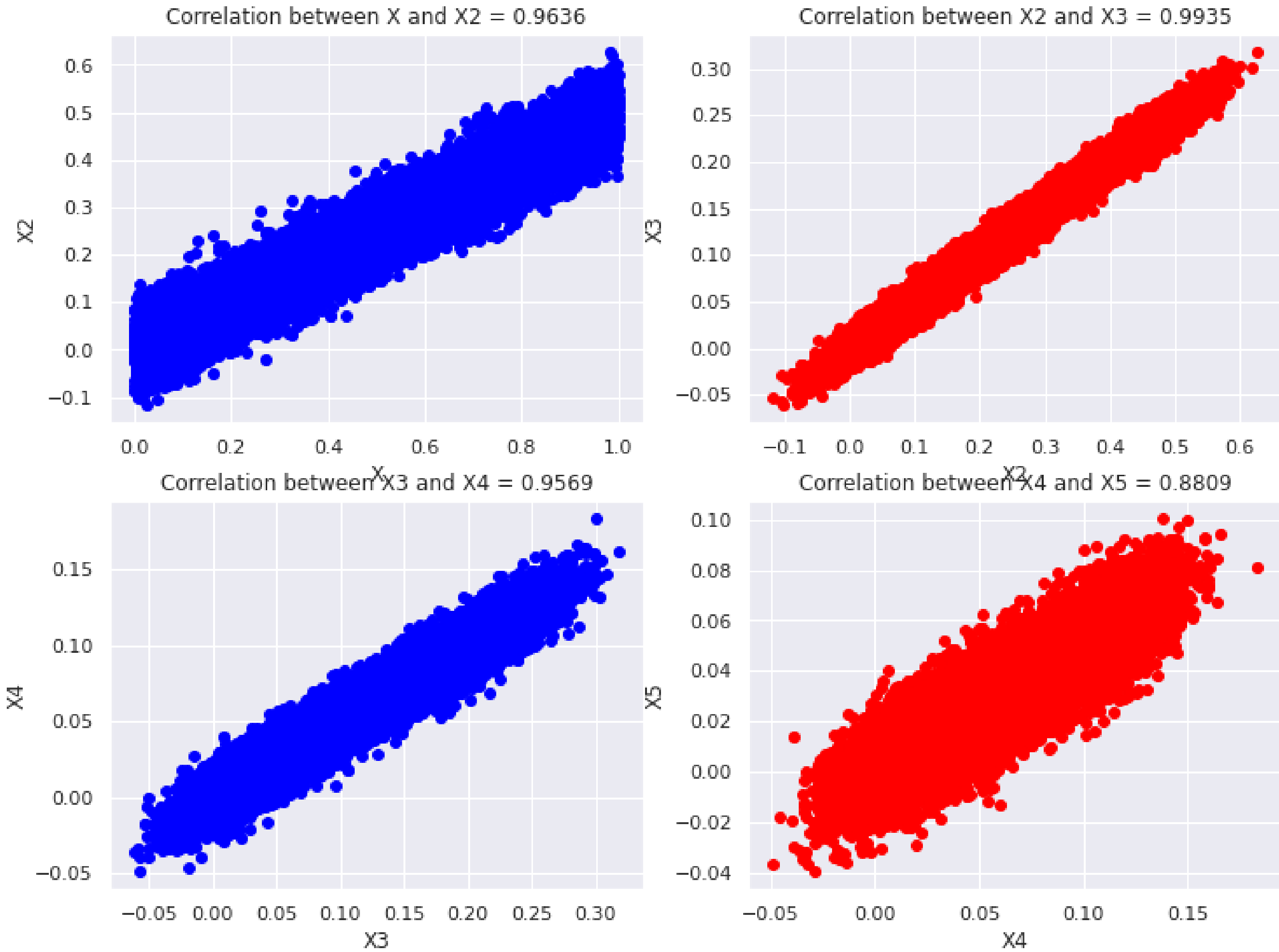
```
numInstances = (houseprediction.shape[0])
```

```
numTrain = 20   # number of training instances
numTest = numInstances - numTrain
```

```
seed = 1

np.random.seed(seed)
X2 = 0.5*X + np.random.normal(0, 0.04, size=numInstances).reshape(-1,1)
X3 = 0.5*X2 + np.random.normal(0, 0.01, size=numInstances).reshape(-1,1)
X4 = 0.5*X3 + np.random.normal(0, 0.01, size=numInstances).reshape(-1,1)
X5 = 0.5*X4 + np.random.normal(0, 0.01, size=numInstances).reshape(-1,1)

fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(12,9))
ax1.scatter(X, X2, color='blue')
ax1.set_xlabel('X')
ax1.set_ylabel('X2')
c = np.corrcoef(np.column_stack((X[:-numTest],X2[:-numTest])).T)
titlestr = 'Correlation between X and X2 = %.4f' % (c[0,1])
ax1.set_title(titlestr)

ax2.scatter(X2, X3, color='red')
ax2.set_xlabel('X2')
ax2.set_ylabel('X3')
c = np.corrcoef(np.column_stack((X2[:-numTest],X3[:-numTest])).T)
titlestr = 'Correlation between X2 and X3 = %.4f' % (c[0,1])
ax2.set_title(titlestr)
```

```
ax3.scatter(X3, X4, color='blue')
ax3.set_xlabel('X3')
ax3.set_ylabel('X4')
c = np.corrcoef(np.column_stack((X3[:-numTest],X4[:-numTest])).T)
titlestr = 'Correlation between X3 and X4 = %.4f' % (c[0,1])
ax3.set_title(titlestr)

ax4.scatter(X4, X5, color='red')
ax4.set_xlabel('X4')
ax4.set_ylabel('X5')
c = np.corrcoef(np.column_stack((X4[:-numTest],X5[:-numTest])).T)
titlestr = 'Correlation between X4 and X5 = %.4f' % (c[0,1])
ax4.set_title(titlestr)
```

```
Text(0.5, 1.0, 'Correlation between X4 and X5 = 0.8809')
```




```
X_train2 = np.column_stack((X[:-numTest],X2[:-numTest]))
X_test2 = np.column_stack((X[-numTest:],X2[-numTest:]))
X_train3 = np.column_stack((X[:-numTest],X2[:-numTest],X3[:-numTest]))
X_test3 = np.column_stack((X[-numTest:],X2[-numTest:],X3[-numTest:]))
X_train4 = np.column_stack((X[:-numTest],X2[:-numTest],X3[:-numTest],X4[:-numTest]))
```

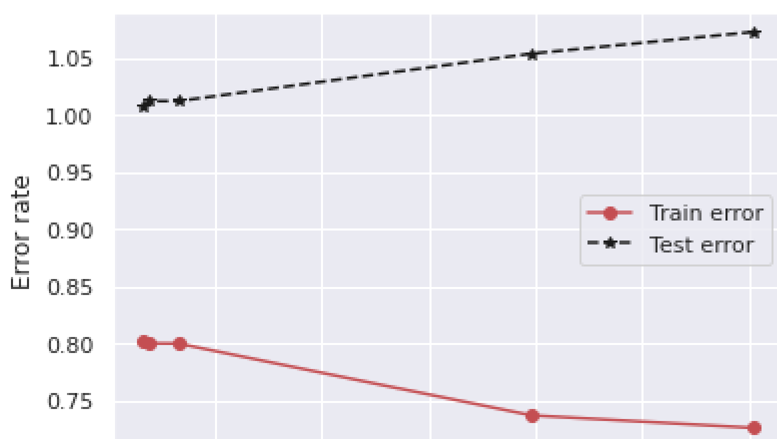
```
X_test4 = np.column_stack((X[-numTest:],X2[-numTest:],X3[-numTest:],X4[-numTest:]))
X_train5 = np.column_stack((X[:-numTest],X2[:-numTest],X3[:-numTest],X4[:-numTest],X5[:-numTe
X_test5 = np.column_stack((X[-numTest:],X2[-numTest:],X3[-numTest:],X4[-numTest:],X5[-numTest


regr2 = linear_model.LinearRegression()
regr2.fit(X_train2, y_train)

regr3 = linear_model.LinearRegression()
regr3.fit(X_train3, y_train)

regr4 = linear_model.LinearRegression()
regr4.fit(X_train4, y_train)

regr5 = linear_model.LinearRegression()
regr5.fit(X_train5, y_train)
```

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

```
y_pred_train = regr.predict(X_train)
y_pred_test = regr.predict(X_test)
y_pred_train2 = regr2.predict(X_train2)
y_pred_test2 = regr2.predict(X_test2)
y_pred_train3 = regr3.predict(X_train3)
y_pred_test3 = regr3.predict(X_test3)
y_pred_train4 = regr4.predict(X_train4)
y_pred_test4 = regr4.predict(X_test4)
y_pred_train5 = regr5.predict(X_train5)
y_pred_test5 = regr5.predict(X_test5)


import pandas as pd
import matplotlib.pyplot as plt

columns = ['Model', 'Train error', 'Test error', 'Sum of Absolute Weights']
model1 = "%.2f X + %.2f" % (regr.coef_[0][0], regr.intercept_[0])
values1 = [ model1, np.sqrt(mean_squared_error(y_train, y_pred_train)),
           np.sqrt(mean_squared_error(y_test, y_pred_test)),
           np.absolute(regr.coef_[0]).sum() + np.absolute(regr.intercept_[0])]

model2 = "%.2f X + %.2f X2 + %.2f" % (regr2.coef_[0][0], regr2.coef_[0][1], regr2.intercept_[
values2 = [ model2, np.sqrt(mean_squared_error(y_train, y_pred_train2)),
           np.sqrt(mean_squared_error(y_test, y_pred_test2)),
           np.absolute(regr2.coef_[0]).sum() + np.absolute(regr2.intercept_[0])]

model3 = "%.2f X + %.2f X2 + %.2f X3 + %.2f" % (regr3.coef_[0][0], regr3.coef_[0][1],
                                                 regr3.coef_[0][2], regr3.intercept_[0])
values3 = [ model3, np.sqrt(mean_squared_error(y_train, y_pred_train3)),
           np.sqrt(mean_squared_error(y_test, y_pred_test3)),
           np.absolute(regr3.coef_[0]).sum() + np.absolute(regr3.intercept_[0])]

model4 = "%.2f X + %.2f X2 + %.2f X3 + %.2f X4 + %.2f" % (regr4.coef_[0][0], regr4.coef_[0][1
```

```
                                        regr4.coef_[0][2], regr4.coef_[0][3], regr4.intercept
values4 = [ model4, np.sqrt(mean_squared_error(y_train, y_pred_train4)),
           np.sqrt(mean_squared_error(y_test, y_pred_test4)),
           np.absolute(regr4.coef_[0]).sum() + np.absolute(regr4.intercept_[0])]

model5 = "%.2f X + %.2f X2 + %.2f X3 + %.2f X4 + %.2f X5 + %.2f" % (regr5.coef_[0][0],
                                        regr5.coef_[0][1], regr5.coef_[0][2],
                                        regr5.coef_[0][3], regr5.coef_[0][4], regr5.intercept
values5 = [ model5, np.sqrt(mean_squared_error(y_train, y_pred_train5)),
           np.sqrt(mean_squared_error(y_test, y_pred_test5)),
           np.absolute(regr5.coef_[0]).sum() + np.absolute(regr5.intercept_[0])]

results = pd.DataFrame([values1, values2, values3, values4, values5], columns=columns)

plt.plot(results['Sum of Absolute Weights'], results['Train error'], 'ro-')
plt.plot(results['Sum of Absolute Weights'], results['Test error'], 'k*--')
plt.legend(['Train error', 'Test error'])
plt.xlabel('Sum of Absolute Weights')
plt.ylabel('Error rate')

results
```

	Model	Train error	Test error	Sum of Absolute Weights
0	-2.57 X + 0.76	0.802360	1.007736	3.328980
1	-1.84 X + -1.32 X2 + 0.71	0.800471	1.012590	3.877417
2	-1.90 X + -2.13 X2 + 1.86 X3 + 0.71	0.800320	1.012596	6.593432
3	-1.66 X + 0.60 X2 + 9.25 X3 + -27.43 X4 + 0.67	0.737331	1.054231	39.614817
4	-1.76 X + 3.99 X2 + 3.61 X3 + -35.83 X4 + 14.5...	0.726365	1.073214	60.364800



Ridge regression

```
from sklearn import linear_model
```

```
ridge = linear_model.Ridge(alpha=0.4)
ridge.fit(X_train5, y_train)
y_pred_train_ridge = ridge.predict(X_train5)
y_pred_test_ridge = ridge.predict(X_test5)

model6 = "%.2f X + %.2f X2 + %.2f X3 + %.2f X4 + %.2f X5 + %.2f" % (ridge.coef_[0][0],
                                        ridge.coef_[0][1], ridge.coef_[0][2],
                                        ridge.coef_[0][3], ridge.coef_[0][4], ridge.intercept
values6 = [ model6, np.sqrt(mean_squared_error(y_train, y_pred_train_ridge)),
           np.sqrt(mean_squared_error(y_test, y_pred_test_ridge)),
           np.absolute(ridge.coef_[0]).sum() + np.absolute(ridge.intercept_[0])]

ridge_results = pd.DataFrame([values6], columns=columns, index=['Ridge'])
pd.concat([results, ridge_results])
```

	Model	Train error	Test error	Sum of Absolute Weights
0	-2.57 X + 0.76	0.802360	1.007736	3.328980
1	-1.84 X + -1.32 X2 + 0.71	0.800471	1.012590	3.877417
2	-1.90 X + -2.13 X2 + 1.86 X3 + 0.71	0.800320	1.012596	6.593432
3	-1.66 X + 0.60 X2 + 9.25 X3 + -27.43 X4 + 0.67	0.737331	1.054231	39.614817
4	-1.76 X + 3.99 X2 + 3.61 X3 + -35.83 X4 + 14.5...	0.726365	1.073214	60.364800

Lasso Regression

```
from sklearn import linear_model

lasso = linear_model.Lasso(alpha=0.01)
lasso.fit(X_train5, y_train)
y_pred_train_lasso = lasso.predict(X_train5)
y_pred_test_lasso = lasso.predict(X_test5)

model7 = "%.2f X + %.2f X2 + %.2f X3 + %.2f X4 + %.2f X5 + %.2f" % (lasso.coef_[0],
                                        lasso.coef_[1], lasso.coef_[2],
                                        lasso.coef_[3], lasso.coef_[4], lasso.intercept_[0])
values7 = [ model7, np.sqrt(mean_squared_error(y_train, y_pred_train_lasso)),
           np.sqrt(mean_squared_error(y_test, y_pred_test_lasso)),
           np.absolute(lasso.coef_[0]).sum() + np.absolute(lasso.intercept_[0])]

lasso_results = pd.DataFrame([values7], columns=columns, index=['Lasso'])
pd.concat([results, ridge_results, lasso_results])
```

	Model	Train error	Test error	Sum of Absolute Weights
0	-2.57 X + 0.76	0.802360	1.007736	3.328980
1	-1.84 X + -1.32 X2 + 0.71	0.800471	1.012590	3.877417
2	-1.90 X + -2.13 X2 + 1.86 X3 + 0.71	0.800320	1.012596	6.593432
3	-1.66 X + 0.60 X2 + 9.25 X3 + -27.43 X4 + 0.67	0.737331	1.054231	39.614817
4	-1.76 X + 3.99 X2 + 3.61 X3 + -35.83 X4 +	0.726365	1.073214	60.364800

```
# This is formatted as code
```