# Linear Regression

We will start with some artificially created data sets but later on, we will progress to real and messier data sets from kaggle.

## Problem Statement

Your neighbour is a real estate agent and wants some help predicting housing prices for regions in the USA. It would be great if you could somehow create a model for her that allows her to put in a few features of a house and returns back an estimate of what the house would sell for. She has asked you if you could help her out with your new data science skills. You say yes, and decide that Linear Regression might be a good path to solve this problem!

Your neighbour then gives you some information about a bunch of houses in regions of the United States, it is all in the data set: USA\_Housing.csv.

The data contains the following columns:

* Avg. Area Income': Avg. Income of residents of the city house is located in.
* Avg. Area House Age': Avg Age of Houses in same city
* Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city
* Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city
* Area Population': Population of city house is located in
* Price': Price that the house sold at
* Address': Address for the house

We've been able to get some data from your neighbor for housing prices as a csv set, let's get our environment ready with the libraries we'll need and then import the data!

### Checking data by randomly plotting it

Use .head(), .describe(), .info() and .columns methods to see quick information about your data or dataframe.

Use some plots to visualise the data esp the target –

.pairplot(), .distplot([Price]) or heatmap() on correlated matrix.

### Split data into X and y features

Let's now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column. We will toss out the Address column because it only has text info that the linear regression model can't use.

X = usaHousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area Number of Bedrooms', 'Area Population']]  
y = usaHousing["Price"]

### Split into train and test set

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=101)

### Instantiating model

Once splitting is done, we will then create our model and instantiate it.

# Instantiating the model  
lm = LinearRegression()

### Training model

After that, we have to train our model on our training data and that is done using .fit() method but remember to pass our training data only which is X\_train and y\_train.

# Fitting/Training our model using our training data  
lm.fit(X\_test,y\_test)

If we print our model, we can see that our model has been trained-

print(lm)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

### Evaluating model

Let's evaluate the model by checking out it's coefficients and how we can interpret them.

We can check coefficients as-

print(lm.coef\_)

and intercept as-

print(lm.intercept\_)

We will then put our coefficients in a data frame

coeff\_df = pd.DataFrame(lm.coef\_,X.columns,columns=["Coefficeints"])

Interpreting the coefficients:

* Holding all other features fixed, a 1 unit increase in \*\*Avg. Area Income\*\* is associated with an \*\*increase of $21.52 \*\*.
* Holding all other features fixed, a 1 unit increase in \*\*Avg. Area House Age\*\* is associated with an \*\*increase of $164883.28 \*\*.
* Holding all other features fixed, a 1 unit increase in \*\*Avg. Area Number of Rooms\*\* is associated with an \*\*increase of $122368.67 \*\*.
* Holding all other features fixed, a 1 unit increase in \*\*Avg. Area Number of Bedrooms\*\* is associated with an \*\*increase of $2233.80 \*\*.
* Holding all other features fixed, a 1 unit increase in \*\*Area Population\*\* is associated with an \*\*increase of $15.15 \*\*.

This may not make sense since this is artificial data. We can use original data which is present in scikit learn-

from sklearn.datasets import load\_boston  
boston = load\_boston() # -----> this is a dictionary having data, features, target and descr  
print(boston.DESCR)  
boston\_df = boston.data

### Predictions from test set

Let's grab predictions off our test set and see how well it did!

# Predictions on test data  
predictions = lm.predict(X\_test)

### Plotting predictions and y\_test

# Plotting predictions and y\_test using scatter plot  
sns.scatterplot(x=y\_test,y=predictions)

If it gives some what straight line, then we can say that our predictions are approximately close to out y\_test

### Residual Histogram

Residual is the difference between y\_test and predictions which we can plot using a histogram. If residuals are normally distributed, then we can say that linear regression model was good choice for our data otherwise not.

# Residual Histogram  
sns.distplot((y\_test-predictions))

### Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

* Mean Absolute Error\*\* (MAE) is the mean of the absolute value of the errors
* Mean Squared Error\*\* (MSE) is the mean of the squared errors
* Root Mean Squared Error\*\* (RMSE) is the square root of the mean of the squared errors

Comparing these metrics:

* \*\*MAE\*\* is the easiest to understand, because it's the average error.
* \*\*MSE\*\* is more popular than MAE, because MSE \"punishes\" larger errors, which tends to be useful in the real world.
* \*\*RMSE\*\* is even more popular than MSE, because RMSE is interpretable in the \"y\" units.

All of these are \*\*loss functions\*\*, because we want to minimize them.

# Evaluation metrics  
mae = met.mean\_absolute\_error(y\_test,predictions)  
print("Mean absolute error : " + mae)  
mse = met.mean\_squared\_error(y\_test,predictions)  
print("Mean squared erro : " + mse)  
rmse = np.sqrt(mse)  
print("Root mean squared error : " + rmse)

Complete code –

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn import metrics as met  
  
# Reading the data  
usaHousing = pd.read\_csv("C:/Users/G01212601/Downloads/Py-DS-ML-Bootcamp-master/Refactored\_Py\_DS\_ML\_Bootcamp-master/11-Linear-Regression/USA\_Housing.csv")  
  
# Assinging the X and y features from the data  
X = usaHousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area Number of Bedrooms', 'Area Population']]  
y = usaHousing["Price"]  
  
# Spliting the data into test data and training data  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=101)  
  
# Creating and training the model  
# Instantiating the model  
lm = LinearRegression()  
  
# Fitting/Training our model using our training data  
lm.fit(X\_test,y\_test)  
  
# Check intercepts and coefficients  
print("Intercept of model : " + str(lm.intercept\_))  
print("Coefficients of the model : " + str(lm.coef\_))  
print()  
  
# Storing coefficients in a data frame  
coeff\_df = pd.DataFrame(lm.coef\_,X.columns,columns=["Coefficeints"])  
print("Coefficient data frame : ")  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
print(coeff\_df)  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
# Predictions on test data  
predictions = lm.predict(X\_test)  
print()  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
print("y\_test's head : ")  
print(pd.DataFrame(y\_test).head())  
print()  
print("Predictions' head : ")  
print(pd.DataFrame(predictions).head())  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
  
# Plotting predictions and y\_test using scatter plot  
plt.figure(1)  
sns.scatterplot(x=y\_test,y=predictions)  
  
# Residual Histogram  
plt.figure(2)  
sns.distplot((y\_test-predictions))  
  
# Evaluation metrics  
mae = met.mean\_absolute\_error(y\_test,predictions)  
print("Mean absolute error : " + str(mae))  
mse = met.mean\_squared\_error(y\_test,predictions)  
print("Mean squared erro : " + str(mse))  
rmse = np.sqrt(mse)  
print("Root mean squared error : " + str(rmse))  
  
plt.show()