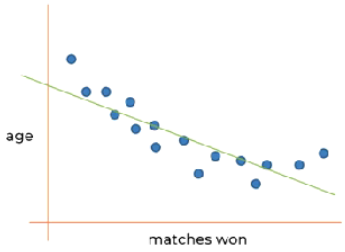
L1 and L2 Regularization

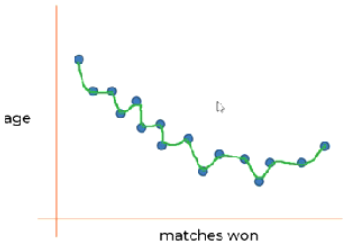
L1 & L2 Regularaization are two of the methods to solve Over fitting issue.

underfit



$$\text{match won} = \theta_0 + \theta_1 * age$$

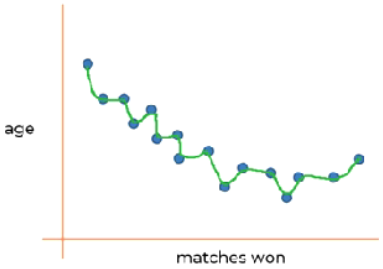
overfit



$$\text{match won} = \theta_0 + \theta_1 * age + \theta_2 * age^2 + \theta_3 * age^3 + \theta_4 * age^4$$

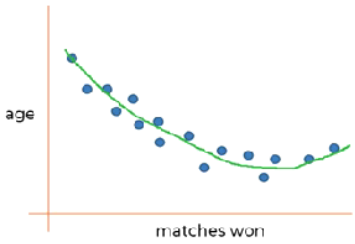
balanced fit

$$\text{match won} = \theta_0 + \theta_1 * age + \theta_2 * age^2$$



$$\text{match won} = \theta_0 + \theta_1 * age + \theta_2 * age^2 + \theta_3 * age^3 + \theta_4 * age^4$$

Try to make θ_3 and $\theta_4$almost close to zero



$$\text{match won} = \theta_0 + \theta_1 * age + \theta_2 * age^2$$

Mean Squared Error

$$ms\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_{predicted}\right)^2$$

Mean Squared Error

$$ms\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_\theta(x_i)\right)^2$$

$$h_\theta(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

L2 Regularization

$$mse = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_\theta(x_i)\right)^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

$$h_\theta(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

L1 Regularization

$$mse = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - h_\theta(x_i)\right)^2 + \lambda \sum_{i=1}^{n} |\theta_i|$$

$$h_\theta(x_i) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

In [1]:
```
# import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [2]:
```
# Suppress Warnings for clean notebook
import warnings
warnings.filterwarnings('ignore')
```

We are going to use Melbourne House Price Dataset where we'll predict House Predictions based on various features.

The Dataset Link is

https://www.kaggle.com/anthonypino/melbourne-housing-market
(https://www.kaggle.com/anthonypino/melbourne-housing-market)

In [4]:
```
# read dataset
dataset = pd.read_csv('D:/Data_Science/My Github/Machine-Learning-with-Python/16.
```

In [4]:
```
dataset.head()
```

Out[4]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	2.5	3067.0
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	2.5	3067.0
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0

5 rows × 21 columns

In [6]:
```
dataset.nunique()
```

Out[6]:
```
Suburb           351
Address        34009
Rooms             12
Type               3
Price           2871
Method             9
SellerG          388
Date              78
Distance         215
Postcode         211
Bedroom2          15
Bathroom          11
Car               15
Landsize        1684
BuildingArea     740
YearBuilt        160
CouncilArea       33
Lattitude      13402
Longtitude     14524
```

In [7]:
```
# let's use limited columns which makes more sense for serving our purpose
cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG', 'Regionname', 'Pro
               'Distance', 'CouncilArea', 'Bedroom2', 'Bathroom', 'Car', 'Landsiz
dataset = dataset[cols_to_use]
```

In [8]:
```
dataset.head()
```

Out[8]:

	Suburb	Rooms	Type	Method	SellerG	Regionname	Propertycount	Distance	CouncilArea
0	Abbotsford	2	h	SS	Jellis	Northern Metropolitan	4019.0	2.5	Yarra City Council
1	Abbotsford	2	h	S	Biggin	Northern Metropolitan	4019.0	2.5	Yarra City Council
2	Abbotsford	2	h	S	Biggin	Northern Metropolitan	4019.0	2.5	Yarra City Council
3	Abbotsford	3	u	VB	Rounds	Northern Metropolitan	4019.0	2.5	Yarra City Council
4	Abbotsford	3	h	SP	Biggin	Northern Metropolitan	4019.0	2.5	Yarra City Council

In [9]:
```
dataset.shape
```

Out[9]:
```
(34857, 15)
```

Checking for Nan values

In [10]:
```
dataset.isna().sum()
```

Out[10]:
```
Suburb             0
Rooms              0
Type               0
Method             0
SellerG            0
Regionname         3
Propertycount      3
Distance           1
CouncilArea        3
Bedroom2        8217
Bathroom        8226
Car             8728
Landsize       11810
BuildingArea   21115
Price           7610
dtype: int64
```

Handling Missing values

In [11]:
```
# Some feature's missing values can be treated as zero (another class for NA valu
# like 0 for Propertycount, Bedroom2 will refer to other class of NA values
# like 0 for Car feature will mean that there's no car parking feature with house
cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2', 'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

# other continuous features can be imputed with mean for faster results since our
# using Lasso and Ridge Regression
dataset['Landsize'] = dataset['Landsize'].fillna(dataset.Landsize.mean())
dataset['BuildingArea'] = dataset['BuildingArea'].fillna(dataset.BuildingArea.mea
```

Drop NA values of Price, since it's our predictive variable we won't impute it

In [12]:
```
dataset.dropna(inplace=True)
```

In [13]:
```
dataset.shape
```

Out[13]:
```
(27244, 15)
```

Let's one hot encode the categorical features

In [14]:
```
dataset = pd.get_dummies(dataset, drop_first=True)
```

In [15]:
```
dataset.head()
```

Out[15]:

	Rooms	Propertycount	Distance	Bedroom2	Bathroom	Car	Landsize	BuildingArea	Price
1	2	4019.0	2.5	2.0	1.0	1.0	202.0	160.2564	1480000.0
2	2	4019.0	2.5	2.0	1.0	0.0	156.0	79.0000	1035000.0
4	3	4019.0	2.5	3.0	2.0	0.0	134.0	150.0000	1465000.0
5	3	4019.0	2.5	3.0	2.0	1.0	94.0	160.2564	850000.0
6	4	4019.0	2.5	3.0	1.0	2.0	120.0	142.0000	1600000.0

5 rows × 745 columns

Let's bifurcate our dataset into train and test dataset

In [16]:
```
X = dataset.drop('Price', axis=1)
y = dataset['Price']
```

In [17]:
```
from sklearn.model_selection import train_test_split
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3, random_s
```

Let's train our Linear Regression Model on training dataset and check the accuracy on test set

In [18]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(train_X, train_y)
```

In [19]:
```
reg.score(test_X, test_y)
```

Out[19]: 0.13853683161510377

In [20]:
```
reg.score(train_X, train_y)
```

Out[20]: 0.6827792395792723

Here training score is 68% but test score is 13.85% which is very low

Normal Regression is clearly overfitting the data, let's try other models

Using Lasso (L1 Regularized) Regression Model

In [21]:
```
from sklearn import linear_model
lasso_reg = linear_model.Lasso(alpha=50, max_iter=100, tol=0.1)
lasso_reg.fit(train_X, train_y)
```

Out[21]: Lasso(alpha=50, max_iter=100, tol=0.1)

You can play with alpha value to get a better score.

In [22]:
```
lasso_reg.score(test_X, test_y)
```

Out[22]: 0.6636111369404487

In [23]:
```
lasso_reg.score(train_X, train_y)
```

Out[23]: 0.6766985624766824

Using Ridge (L2 Regularized) Regression Model

In [24]:
```
from sklearn.linear_model import Ridge
ridge_reg= Ridge(alpha=50, max_iter=100, tol=0.1)
ridge_reg.fit(train_X, train_y)
```

Out[24]: Ridge(alpha=50, max_iter=100, tol=0.1)

In [25]:
```
ridge_reg.score(test_X, test_y)
```

Out[25]: 0.6670848945194957

In [26]:
```
ridge_reg.score(train_X, train_y)
```

Out[26]:
```
0.6622376739684328
```

We see that Lasso and Ridge Regularizations prove to be beneficial when our Simple Linear Regression Model overfits. These results may not be that contrast but significant in most cases.Also that L1 & L2 Regularizations are used in Neural Networks too

Date	Author
11-14-2021	**Ehsan Zia**