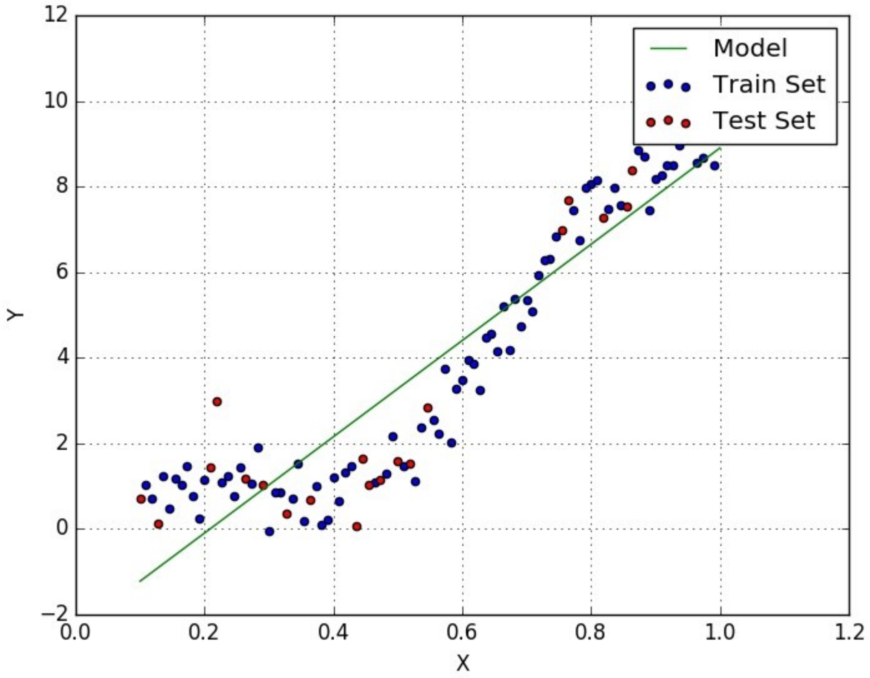
LINEAR REGRESSION

Anshu Pandey

- A Supervised Learning Algorithm that learns from a set of training samples

- It estimates relationship between a dependent variable (target/label) and one or more independent variable (predictors).



WHAT IS LINEAR REGRESSION?

Linear Regression

Univariate Linear Regression

$$y = m_1x_1+\mathrm{c}$$

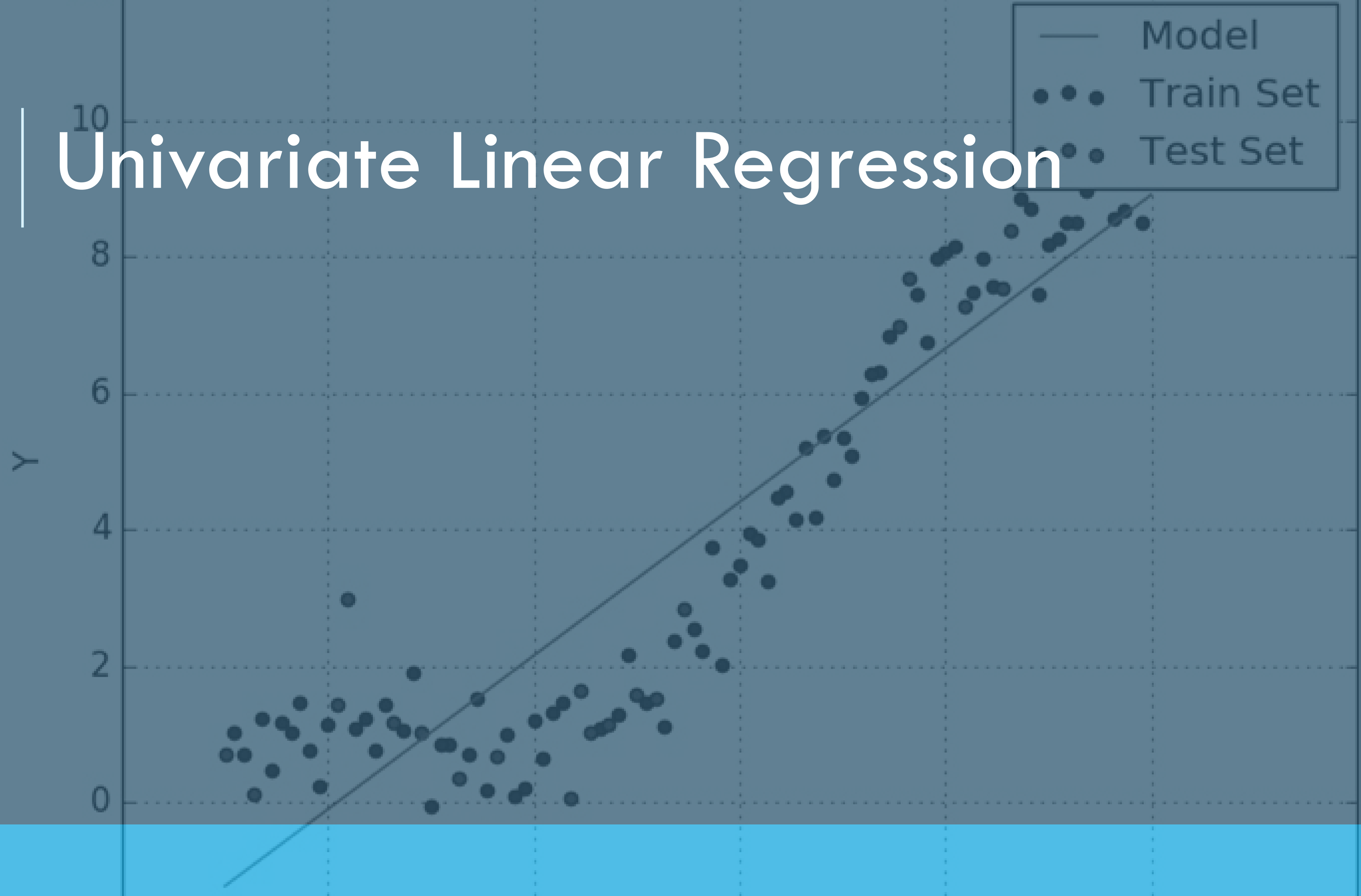
Multivariate Linear Regression

$$y = m_1x_1 + m_2x_2 + m_3x_3 + \ldots + m_nx_n + \mathrm{c}$$

Polynomial Linear Regression

$$y = m_1x_1 + m_2x_1^2 + m_3x_1^3 + \ldots + m_nx_1^n + \mathrm{c}$$

Univariate Linear Regression

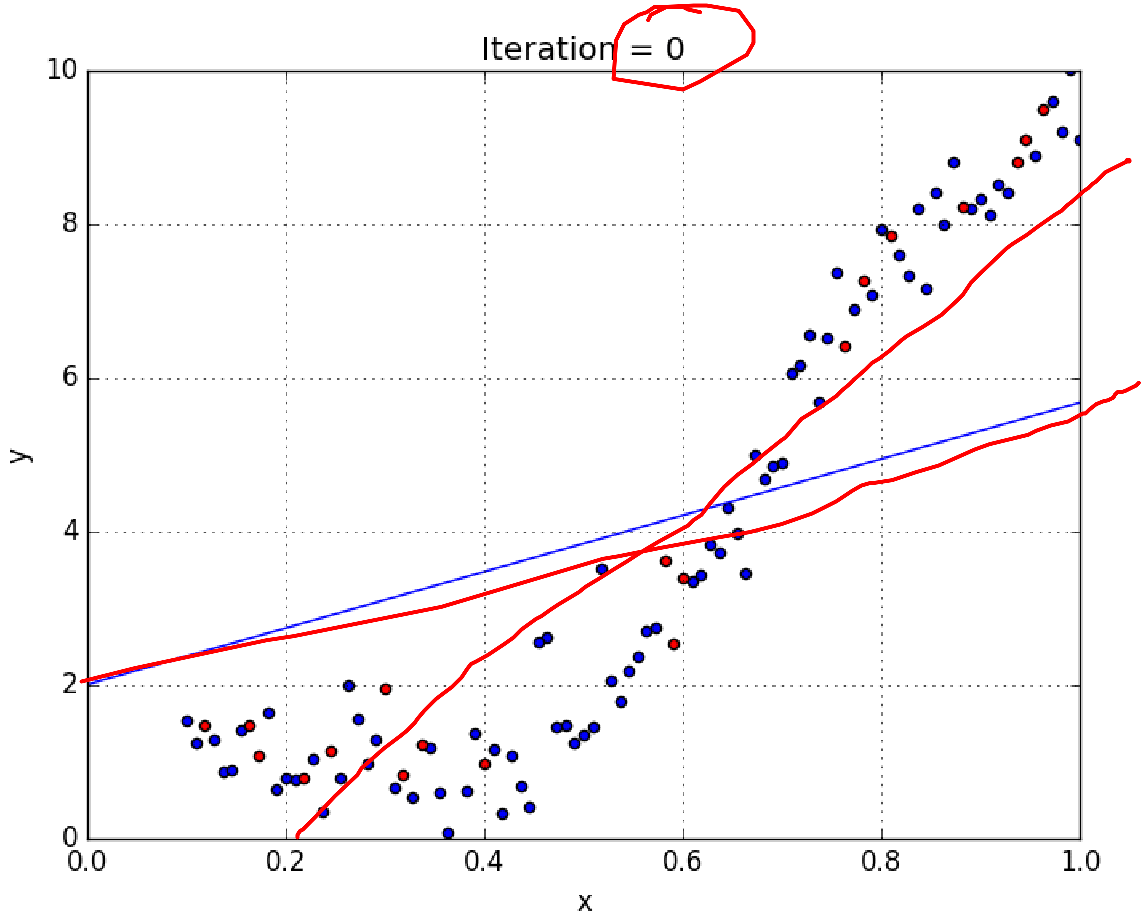


$\hat{y} = m_1 x_1 + m_2 x_2 + m_3 x_3 + c$

multiple linear regression

During the training period the regression line is getting more fit.

$\hat{y} = mx + c$



UNIVARIATE LINEAR REGRESSION

Uncle 15yrs

friend 6 months

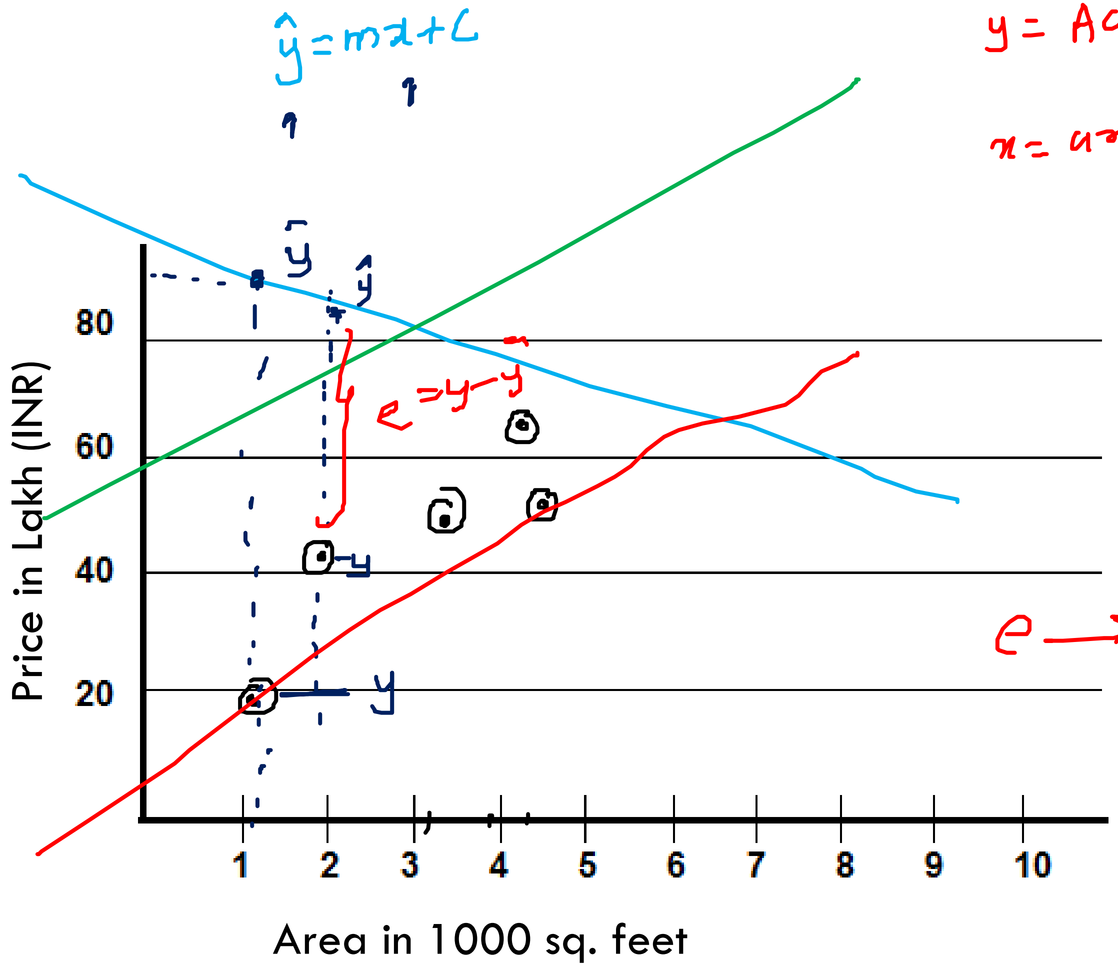
Housing Prices Prediction

Area (sq ft) x	Price In INR y
1200	1,800,000
1800	4,200,000
3200	4,400,000
3800	62,00,000
4200	5,050,000

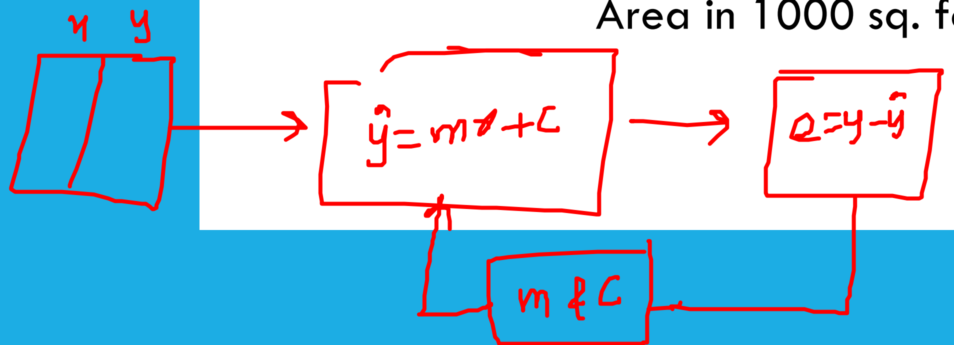
$\hat{y}$ = predicted value

y = Actual price

x = area of home



$e \rightarrow m\&c$

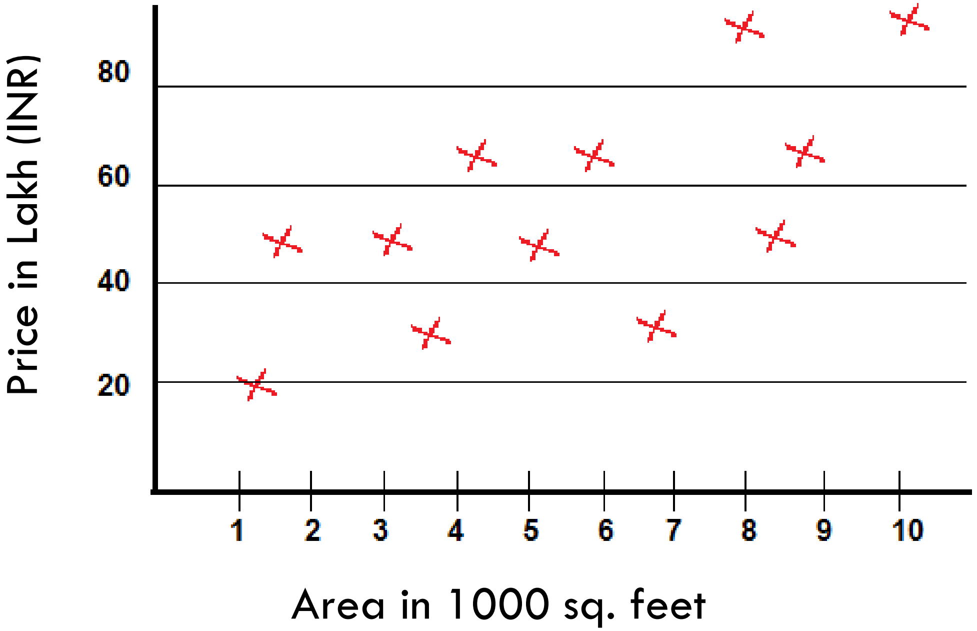


Housing Prices Prediction

Area (sq ft) (x)	Price In INR (y)
1200	20,00,000
1800	42,00,000
3200	44,00,000
3800	25,00,000
4200	62,00,000

y: Dependent Variable, criterion variable, or regressand.

x: Independent variable, predictor variables or regressors.



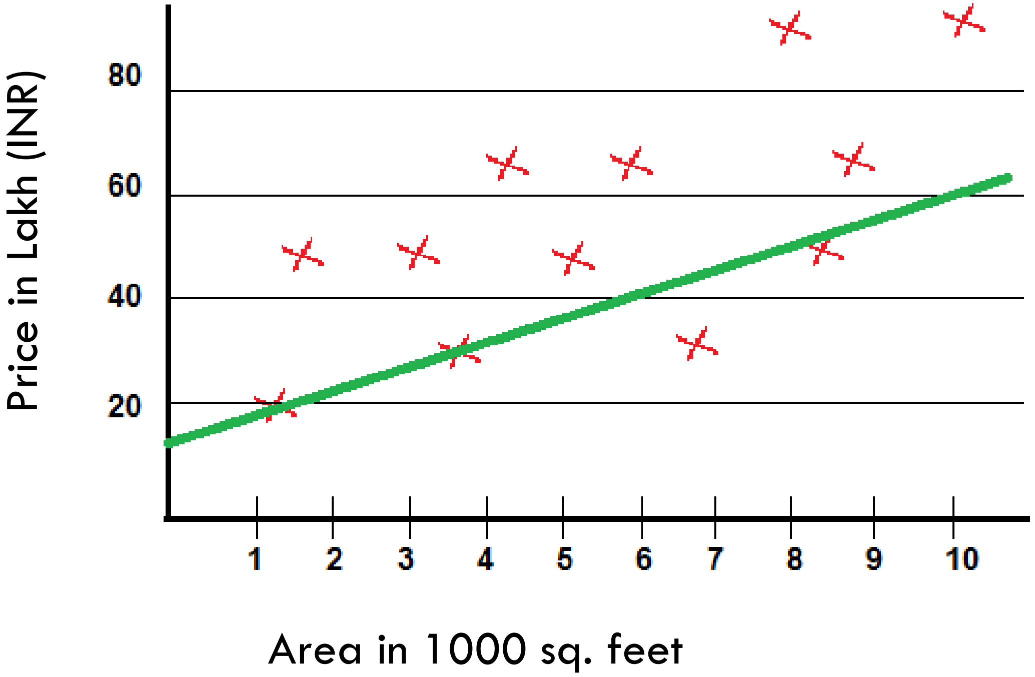
Housing Prices Prediction

Area (sq ft)	Price In INR
1200	20,00,000
1800	42,00,000
3200	44,00,000
3800	25,00,000
4200	62,00,000

$$\hat{y} = mx + c$$

$\hat{y} = Value\ predicted\ by\ current\ Algorithm$

Linear Regression in one Variable



Housing Prices Prediction

MSE

minimize

$$(y_n - \hat{y})^2$$

$$E = \frac{1}{n} \sum_{i=1} (y - \hat{y})$$

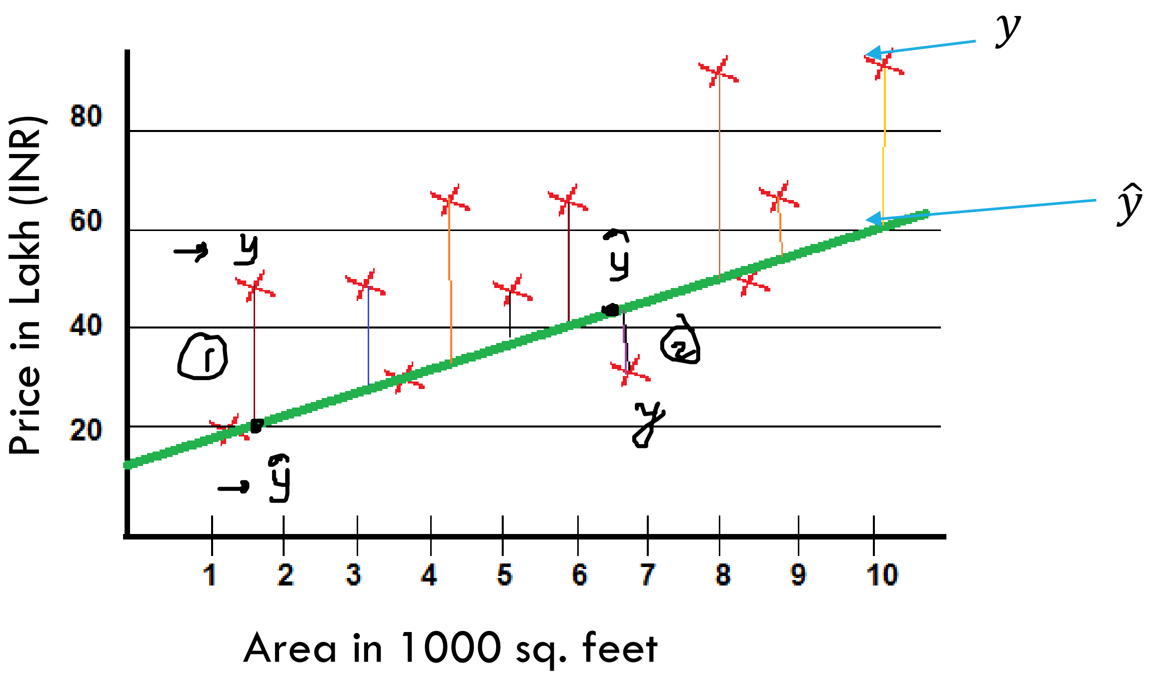
MAE

$\hat{y} = mx + c$

Predictor

$e = y - \hat{y}$

$$\hat{y} = mx + c$$



Variables affecting Regression Equation

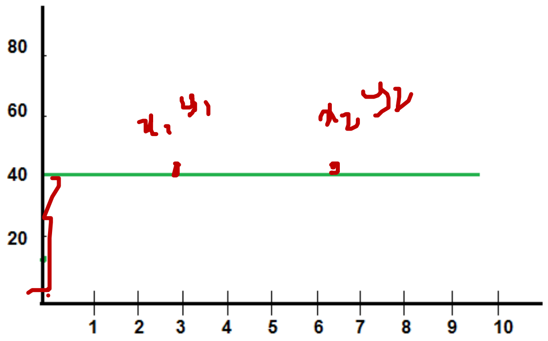
$c = y$ at $x=0$

slope

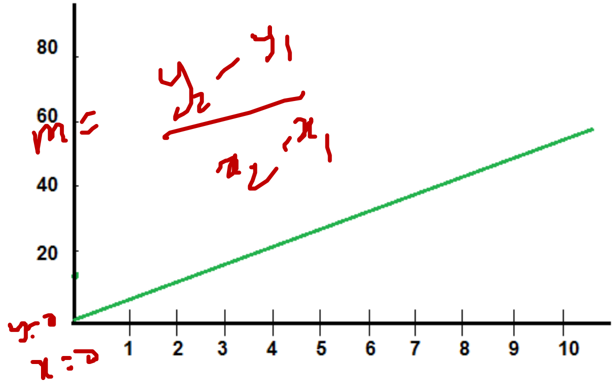
$y = mx + c$

intercept

$m = 0$ ✓
$c = 40$ ✓

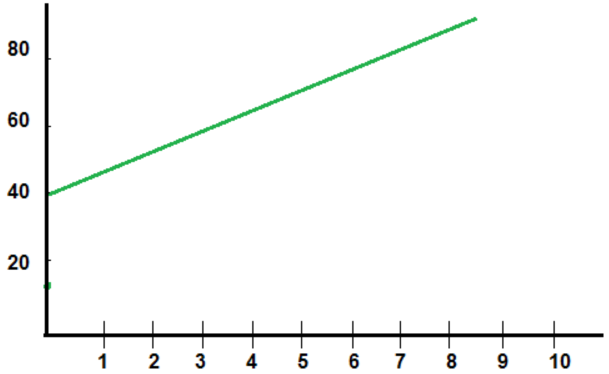


$m = 0.8$ ✓
$c = 0$ ✓



$m = \frac{y_2 - y_1}{x_2 - x_1}$

$m = 0.8$
$c = 40$



$$\hat{y} = mx + c$$

Housing Prices Prediction

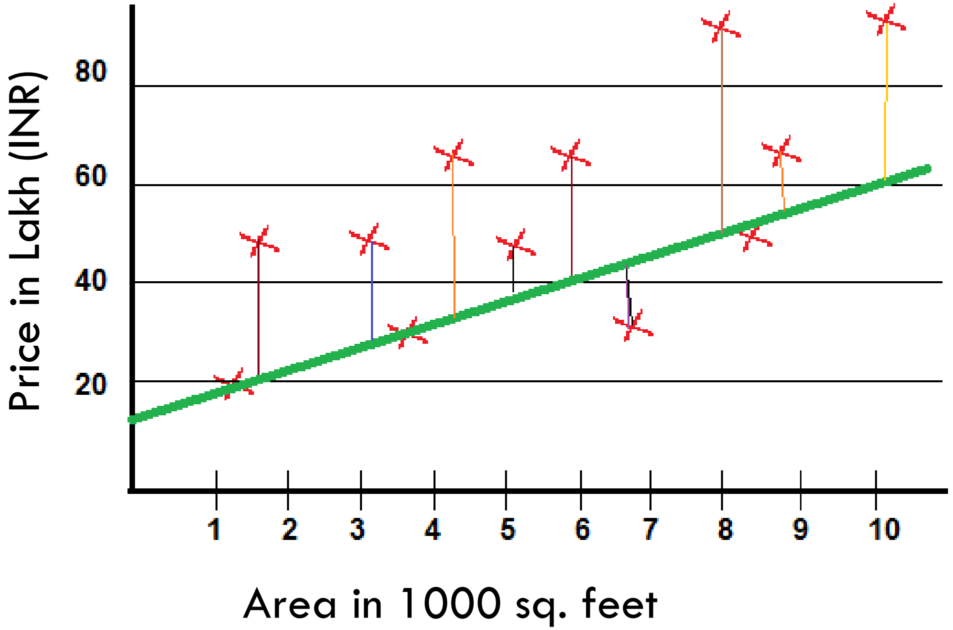
Cost Function

$$J = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$$

$$j(m_i, c\ ) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$$

Predictor

$$\hat{y} = mx + c$$



Optimizers

Regression Equation: $\hat{y} = mx + C$

$$\hat{y} = mx + c$$

Gradient Descent

Parameters m & C

$$m_i, c$$

$m_n = m - LR \times \frac{\partial E}{\partial m}$

$c_n = C - LR \times \frac{\partial E}{\partial C}$

Learning Rate

0.001

Error / Cost Function: $\underline{E} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

$$J(m_i, c) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Goal minimize E by changing m & C

$$\underset{m_i,c}{minimize}\, J(m_i, c)$$

α

η

Gradient Descent Algorithm

Repeat Until converge

$$w_j := w_j - lr \frac{\partial}{\partial w} J(w_j)$$

simultaneously update, $j = 0, j = 1$

where, w=parameter (coefficient & constant)

Learning Rate lr

Learning Rate lr controls how big step we take while updating our parameter w.

If lr is too small, gradient descent can be slow.

If lr is too big, gradient descent can overshoot the minimum, it may fail to converge

Gradient Descent Algorithm

Repeat Until converge

$$w_j := w_j - lr\frac{\partial}{\partial w}J(w_j)$$

simultaneously update, $j = 0, j = 1$
where, w=parameter (coefficient & constant)

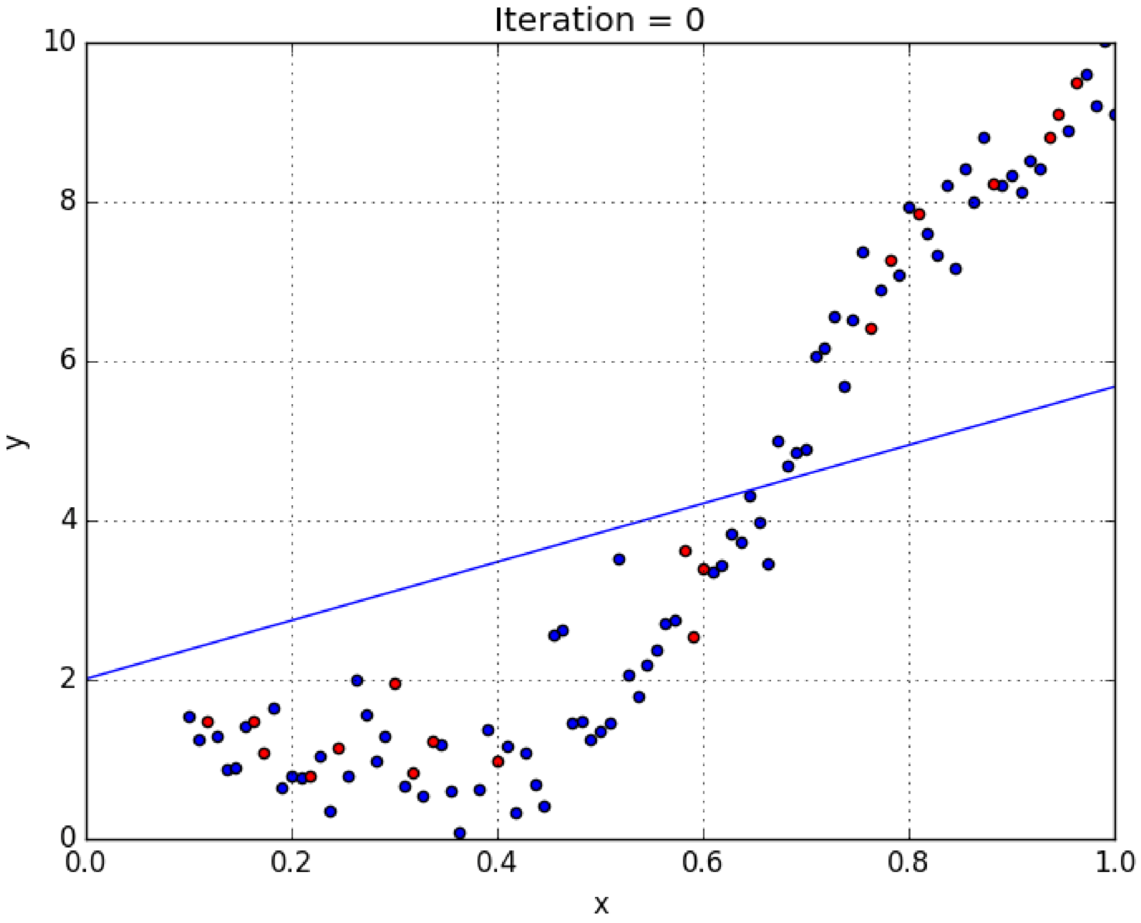
Linear Regression Model

$$\hat{y} = mx + c$$

$$j(m_i, c) = \frac{1}{2n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^{\boldsymbol{2}}$$

UNIVARIATE LINEAR REGRESSION

Linear Regression Process Visualization



Objective of Linear Regression

- Establish If there is a relationship between two variables.

  Examples – relationship between housing process and area of house, no of hours of study and the marks obtained, income and spending etc.

- Prediction of new possible values

  Based on the area of house predicting the house prices in a particular month; based on number of hour studied predicting the possible marks. Sales in next 3months etc.

LINEAR REGRESSION USE CASES

Real Estate	•To model residential home prices as a function of the home's living area, bathrooms, number of bedrooms, lot size.
Medicine	•To analyze the effect of a proposed radiation treatment on reducing tumor sizes based on patient attributes such as age or weight.
Demand Forecasting	•To predict demand for goods and services. For example, restaurant chains can predict the quantity of food depending on weather.
Marketing	•To predict company's sales based on previous month's sales and stock prices of a company.

Simple Linear Regression

Python code using sklearn

Import the libraries

```
import numpy

import matplotlib as plt

Import pandas
```

Import dataset

```
dataset=pandas.read_csv(‘salary_data.csv’)

X=dataset.iloc[:,:-1].values

Y=dataset.iloc[:,1].values
```

Train test split

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest =
train_test_split(X,Y,test_size=0.2,random_state=0)
```

Simple Linear Regression

```
from sklearn import linear_model

alg = linear_model.LinearRegression()

alg.fit(xtrain,ytrain)
```

Predicting the test results

```
ypred=alg.predict(xtest)
```

Visualizing the training results

```
plt.scatter(xtrain,ytrain,‘g’)

plt.plot(xtrain,alg.predict(xtrain),’r’)

plt.title(“Training set”)

plt.xlabel(“Experience”)

plt.ylabel(“Salary”)

plt.show()
```

Visualizing the test results

```
plt.scatter(xtest,ytest,‘g’)

plt.plot(xtest,alg.predict(xtest),’r’)

plt.title(“Test set”)

plt.xlabel(“Experience”)

plt.ylabel(“Salary”)

plt.show()
```

Test Score (Accuracy on test data)

```
accuracy=alg.score(xtest,ytest)

print(accuracy)
```

Coefficient and intercept value

```
#for printing coefficient
alg.coef_
# for printing intercept value
alg.intercept_
```

Performance Analysis

```
from sklearn.metrics import mean_squared_error, r2_score
# The mean squared error
print("Mean squared error: %.2f"%mean_squared_error(ytest,ypred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(ytest, ypred))
```

Multivariate Linear Regression

Python code using sklearn

One Hot Encoding

When some inputs are categories (e.g. gender) rather than numbers (e.g. age) we need to represent the category values as numbers so they can be used in our linear regression equations.

Dummy Variables

Salary	Credit Score	Age	State
192,451	485	42	New York
118,450	754	35	California
258,254	658	28	California
200,123	755	48	New York
152,485	654	52	California

Dummy Variables

New York	California
1	0
0	1
0	1
1	0
0	1

Encoding Categorical Data

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
labelencoder = LabelEncoder()
#considering X is dataset from above slide
# 3 is the index number of state
X[:, 3] = labelencoder.fit_transform(X[:, 3])
onehotencoder = OneHotEncoder(categorical_features = [3])
X = onehotencoder.fit_transform(X).toarray()
```

Avoiding the Dummy variable trap

```
X=X[:,1:]
```

NOTE : if you have n dummy variables remove one dummy variable to avoid the dummy variable trap. However the linear regression model that is built in R and Python takes care of this. But there is no harm in removing it by ourselves

Feature Scaling

Standardization	Normalization
$X_{stand} = \dfrac{x \ - mean(x)}{standard\_deviation(x)}$	$X_{norm} = \dfrac{x \ - min(x)}{\max(x) - \min(x)}$

Standard Scale using sklearn

```
from sklearn.preprocessing import StandardScaler

sc_x = StandardScaler()

sc_y = StandardScaler()

X_std = sc_x.fit_transform(X)

y_std = sc_y.fit_transform(y)
```

Boston housing data

```
In [1]: boston = pd.read_csv('boston.csv’)

In [2]: print(boston.head()
    CRIM       ZN      INDUS  CHAS NX    RM     AGE     DIS        RAD    TAX \
0   0.00632    18.0    2.31   0   0.538  6.575  65.2    4.0900  1  296.0
1   0.02731    0.0     7.07   0   0.469  6.421  78.9    4.9671  2  242.0
2   0.02729    0.0     7.07   0   0.469  7.185  61.1    4.9671  2  242.0
3   0.03237    0.0     2.18   0   0.458  6.998  45.8    6.0622  3  222.0
4   0.06905    0.0     2.18   0   0.458  7.147  54.2    6.0622  3  222.0
    PTRATIO B      LSTAT   MEDV
0   15.3   396.90  4.98   24.0
1   17.8   396.90  9.14   21.6
2   17.8   392.83  4.03   34.7
3   18.7   394.63  2.94   33.4
4   18.7   396.90  5.33   36.2
```

Creating feature and target arrays

```
In [3]: X = boston.drop('MEDV', axis=1).values

In [4]: y = boston['MEDV'].values
```

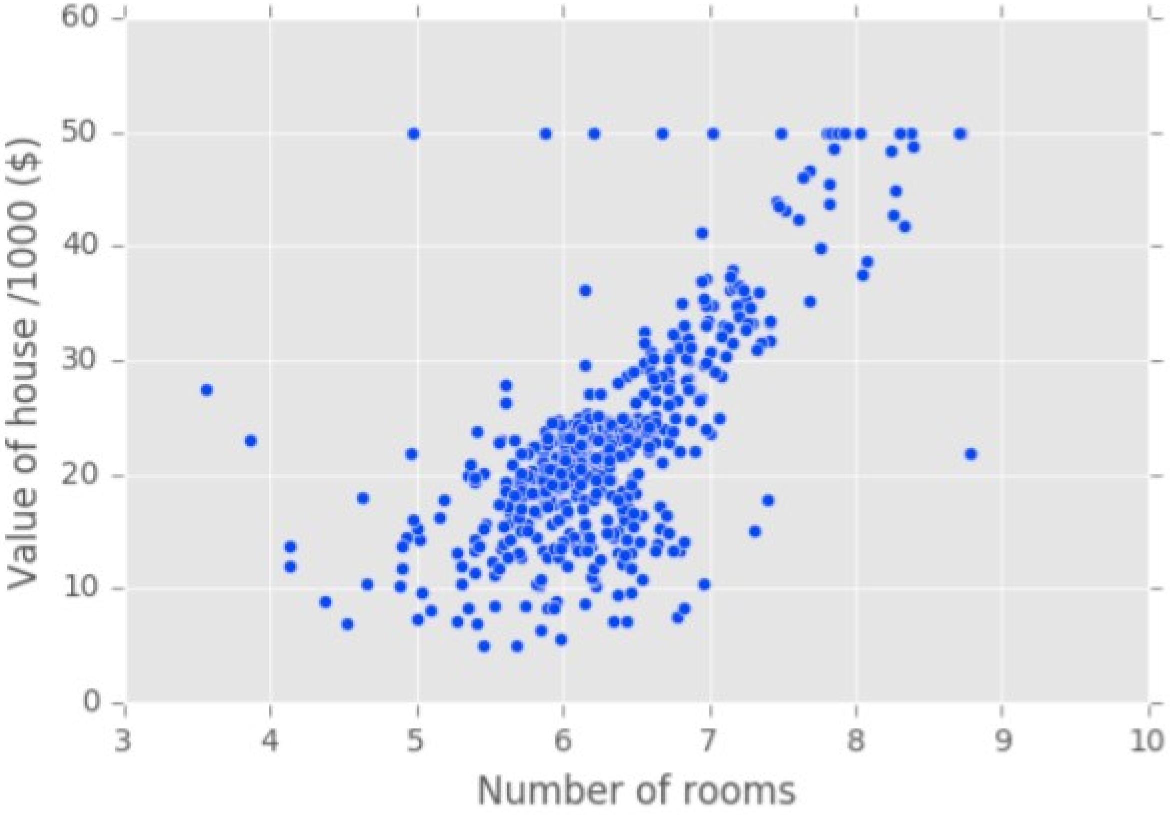
Predicting house value from a single feature

```
In [5]: X_rooms = X[:,5]

In [6]: type(X_rooms), type(y)
Out[6]: (numpy.ndarray, numpy.ndarray)

In [7]: y = y.reshape(-1, 1)

In [8]: X_rooms = X_rooms.reshape(-1, 1)
```

Plotting house value vs. number of rooms

```
In [9]: plt.scatter(X_rooms, y)

In [10]: plt.ylabel('Value of house /1000 ($)’)

In [11]: plt.xlabel('Number of rooms’)

In [12]: plt.show()
```

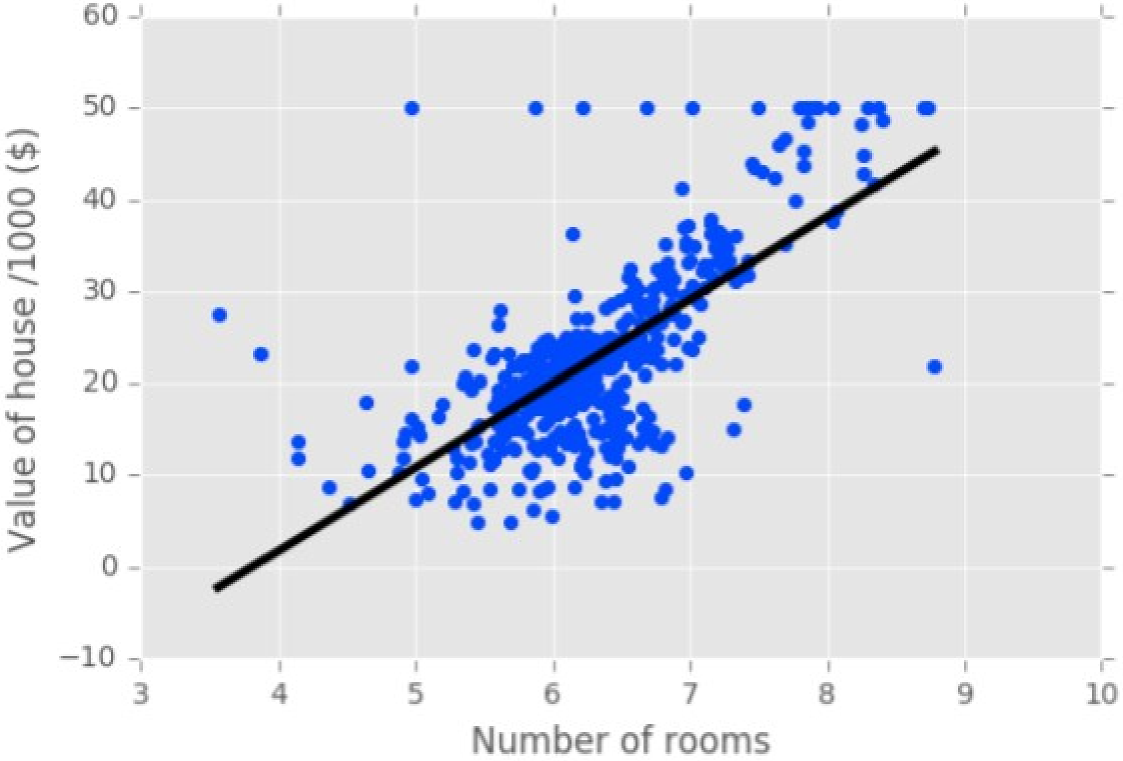
Plotting house value vs. number of rooms



Fitting a regression model

```
In [13]: from numpy import linspace
In [14]: from sklearn import linear_model
In [15]: alg = linear_model.LinearRegression()
In [16]: alg.fit(X_rooms, y)
In [17]: k=linspace(min(X_rooms),max(X_rooms)).reshape(-1,1)
In [18]: plt.scatter(X_rooms, y, color='blue’)
In [19]: plt.plot(k, alg.predict(k),'b', linewidth=3)
In [20]: plt.show()
```

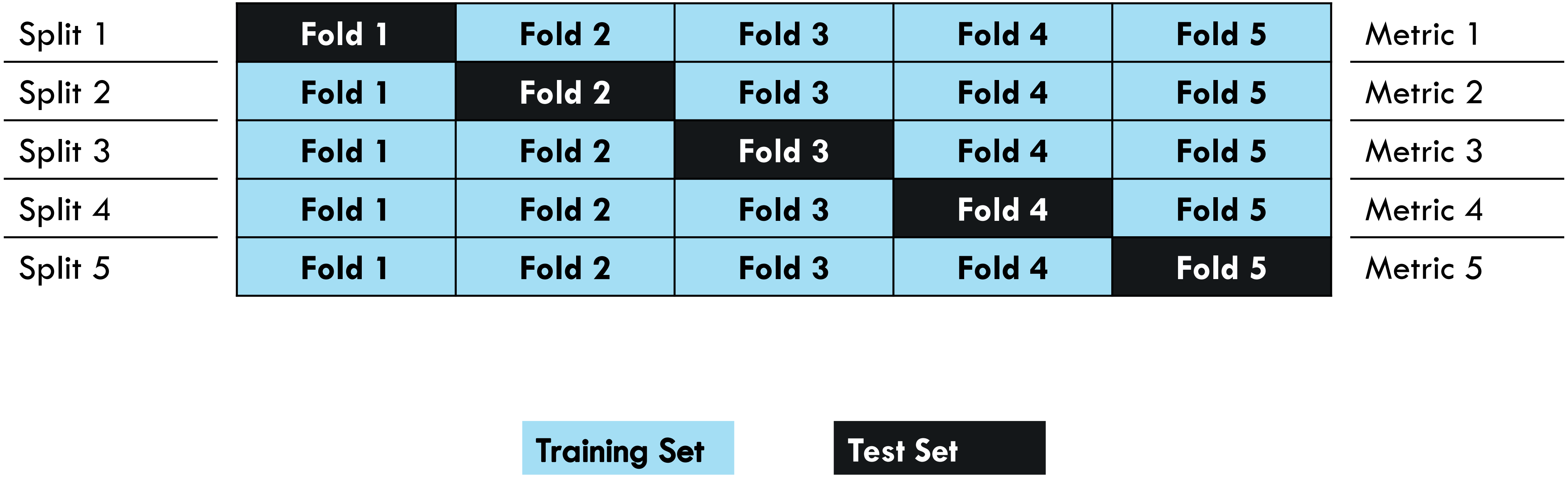
Fitting a regression model



Linear regression on all features

```
In [1]: from sklearn.model_selection import train_test_split
In [2]: X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size = 0.3, random_state=42)
In [3]: alg2 = linear_model.LinearRegression()
In [4]: alg2.fit(X_train, y_train)
In [5]: y_pred = alg2.predict(X_test)
In [6]: alg2.score(X_test, y_test)
Out[6]: 0.71122600574849526
```

K fold Cross Validation



Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 4
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 5

Training Set

Test Set

Cross-validation and model performance

- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k-fold CV
- More folds = More computationally expensive

Cross-validation in scikit-learn

```
In [1]: from sklearn.model_selection import cross_val_score

In [2]: alg = linear_model.LinearRegression()

In [3]: cv_results = cross_val_score(alg, X, y, cv=5)

In [4]: print(cv_results)

[ 0.63919994 0.71386698 0.58702344 0.07923081 -0.25294154]

In [5]: numpy.mean(cv_results)

Out[5]: 0.35327592439587058
```

Overfitting & Generalisation

As we train our model with more and more data the it may start to fit the training data more and more accurately, but become worse at handling test data that we feed to it later.

This is known as “over-fitting” and results in an increased generalization error.

Large coefficients lead to overfitting

Penalizing large coefficients: Regularization

How to minimize?

- To minimize the generalization error we should
- Collect as much sample data as possible.
- Use a random subset of our sample data for training.
- Use the remaining sample data to test how well our model copes with data it was not trained with.

L1 Regularisation (Lasso)

(Least Absolute Shrinkage and Selection Operator)

- Having a large number of samples (n) with respect to the number of dimensionality (d) increases the quality of our model.
- One way to reduce the effective number of dimensions is to use those that most contribute to the signal and ignore those that mostly act as noise.
- L1 regularization achieves this by adding a penalty that results in the weight for the dimensions that act as noise becoming 0.
- L1 regularization encourages a sparse vector of weights in which few are non-zero and many are zero.

L1 Regularisation (Lasso)

Depending on the regularization strength, certain weights can become zero, which makes the LASSO also useful as a supervised feature selection technique:

$$j(w_i\ ) = \frac{1}{2n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{y_i} - \boldsymbol{\widehat{y}_i})^{\boldsymbol{2}} + \boldsymbol{\lambda}||\boldsymbol{w_i}||$$

A limitation of the LASSO is that it selects at most n variables if m > n.

Lasso regression in scikit-learn

```
In [1]: from sklearn.linear_model import Lasso
In [2]: X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size = 0.3, random_state=42)
In [3]: lasso = Lasso(alpha=0.1, normalize=True)
In [4]: lasso.fit(X_train, y_train)
In [5]: lasso_pred = lasso.predict(X_test)
In [6]: lasso.score(X_test, y_test)
Out[6]: 0.5950229535328551
```

L2 Regularisation (Ridge)

- Another way to reduce the complexity of our model and prevent overfitting to outliers is L2 regression, which is also known as ridge regression.

- In L2 Regularization we introduce an additional term to the cost function that has the effect of penalizing large weights and thereby minimizing this skew.

L2 Regularisation (Ridge)

Ridge regression is an L2 penalized model where we simply add the squared sum of the weights to our least-squares cost function:

$$j(w_i\ ) = \frac{1}{2n}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^2 + \boldsymbol{\lambda}||\boldsymbol{w_i}||^2$$

By increasing the value of the hyperparameter λ , we increase the regularization strength and shrink the weights of our model.

Ridge regression in scikit-learn

```
In [1]: from sklearn.linear_model import Ridge
In [2]: X_train, X_test, y_train, y_test = train_test_split(X,
y,test_size = 0.3, random_state=42)
In [3]: ridge = Ridge(alpha=0.1, normalize=True)
In [4]: ridge.fit(X_train, y_train)
In [5]: ridge_pred = ridge.predict(X_test)
In [6]: ridge.score(X_test, y_test)
Out[6]: 0.6996938275127313179
```

L1 & L2 Regularisation (Elastic Net)

- L1 Regularisation minimises the impact of dimensions that have low weights and are thus largely “noise”.
- L2 Regularisation minimise the impacts of outliers in our training data.
- L1 & L2 Regularisation can be used together and the combination is referred to as Elastic Net regularisation.
- Because the differential of the error function contains the sigmoid which has no inverse, we cannot solve for w and must use gradient descent.

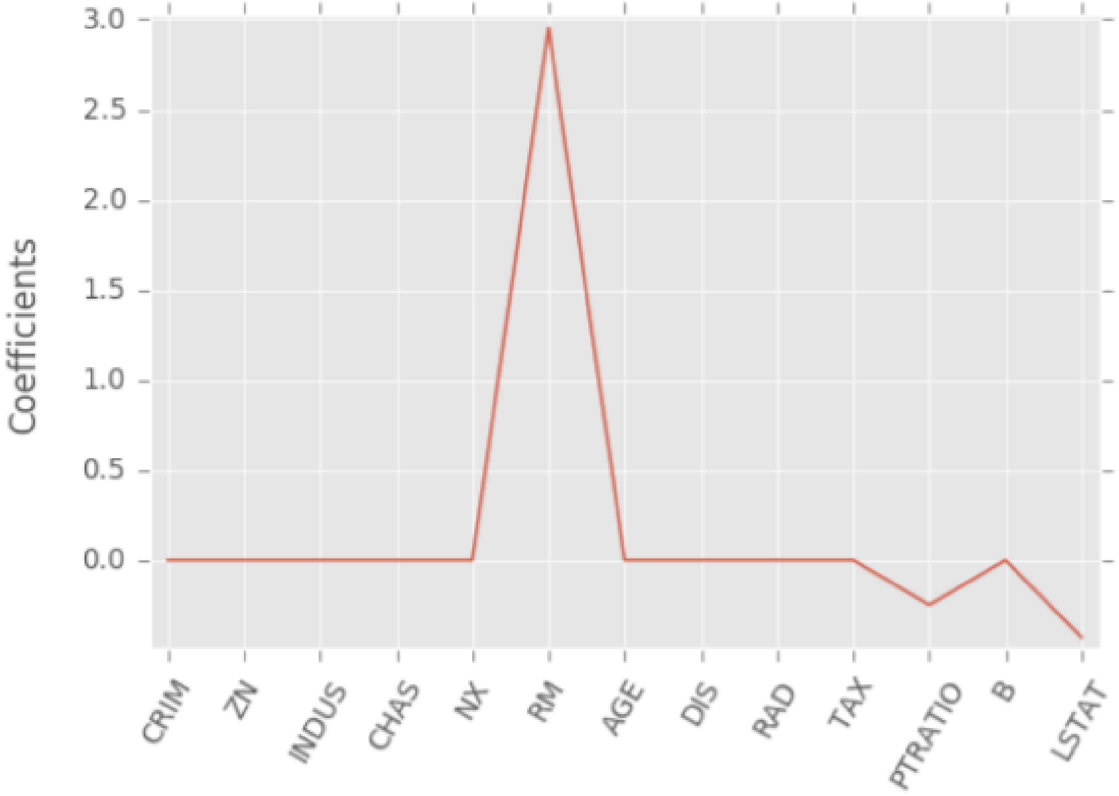
Lasso regression for feature selection

- Can be used to select important features of a dataset
- Shrinks the coefficients of less important features to exactly 0.

Lasso regression for feature selection

```
In [1]: from sklearn.linear_model import Lasso

In [2]: names = boston.drop('MEDV', axis=1).columns

In [3]: lasso = Lasso(alpha=0.1)

In [4]: lasso_coef = lasso.fit(X, y).coef_

In [5]: plt.plot(range(len(names)), lasso_coef)

In [6]: plt.xticks(range(len(names)), names, rotation=60)

In [7]: plt.ylabel('Coefficients')

In [8]: plt.show()
```

Lasso regression for feature selection



Practice Datasets

https://openmv.net/info/unlimited-time-test

https://openmv.net/info/distillation-tower

https://openmv.net/info/oil-company-doe

http://www.stat.ufl.edu/~winner/datasets.html

Stay tuned for practicing Linear Regression with datasets

THANK YOU