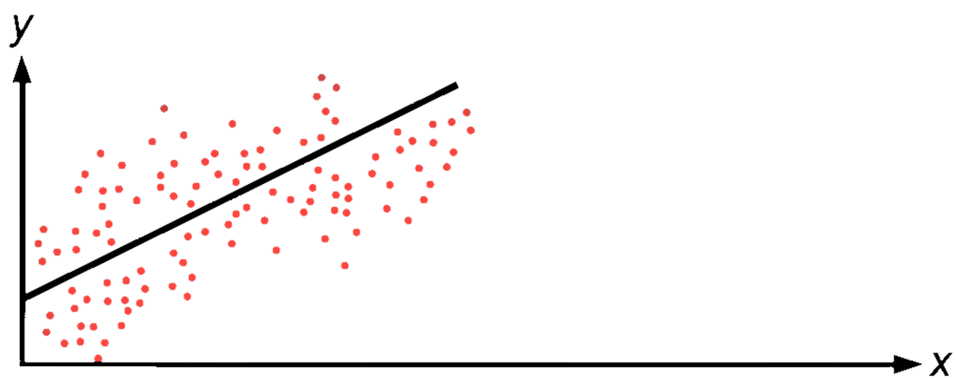
Linear Regression



Overview

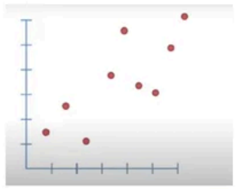
- **Linear Regression** is a powerful and widely used technique in machine learning, serving as the **foundation** for many predictive modeling tasks.
- Used to predict **continuous** values.
- It is one of the most important and **mostly used** supervised learning algorithms.
- At its core, Linear Regression aims to establish a **relationship** between a **dependent variable/target** and one or more **independent variables/feature**.

Types of Linear Regression

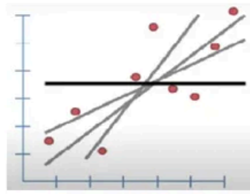
1. **Simple Linear Regression**:
   - If a **single** independent variable is used to predict the value of a numerical dependent variable.

2. **Multiple Linear regression**:
   - If **more than one** independent variable is used to predict the value of a numerical dependent variable.

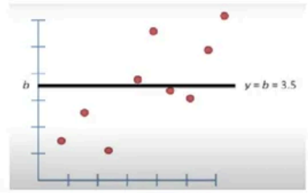
Linear regression algorithm



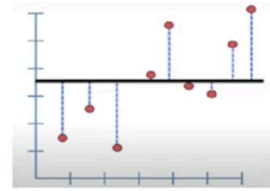
1. We collected some data



2. We should fit the line to the data. But which line is better?



3. Start with the line across the mean value!

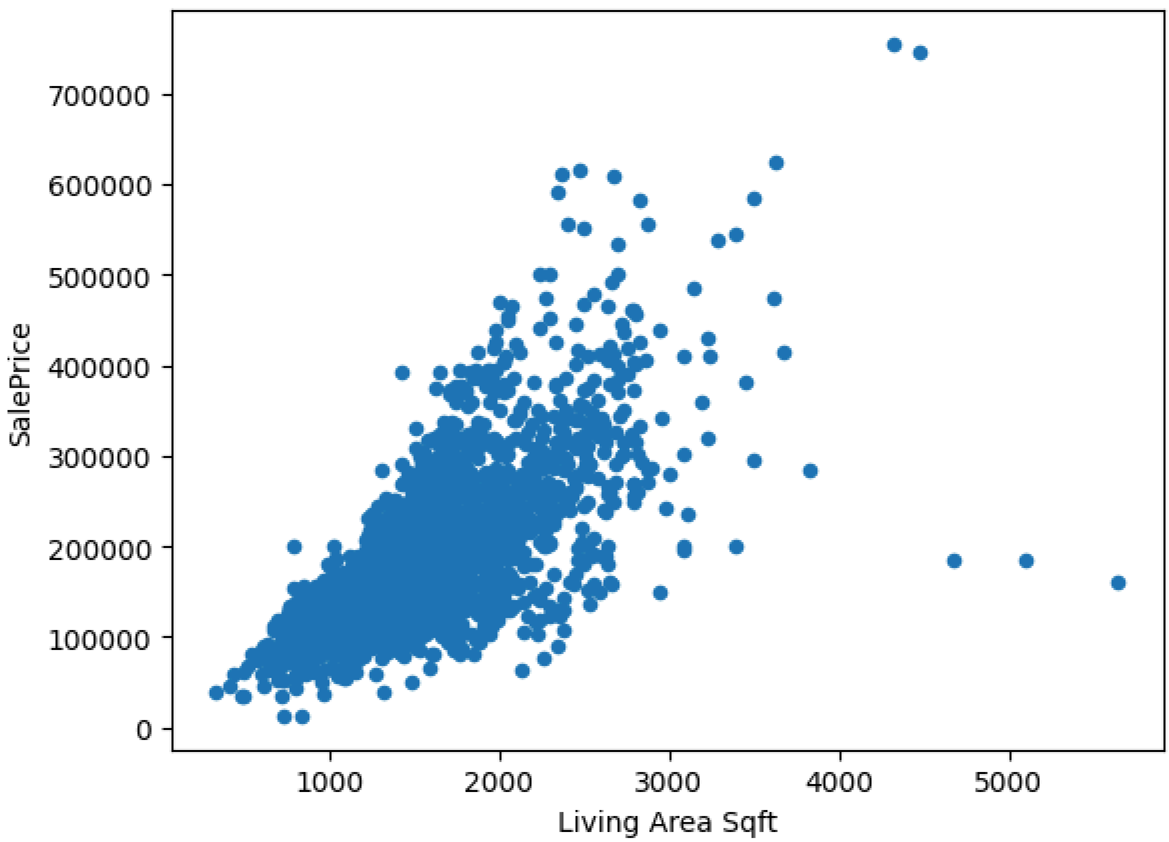


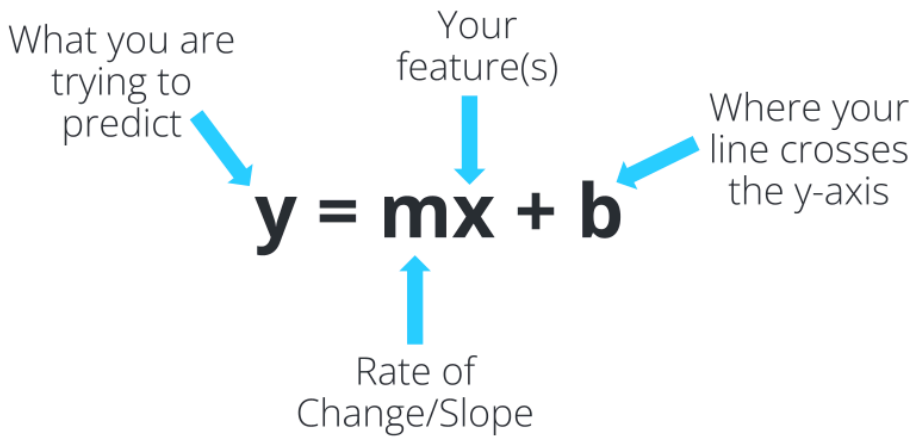
4. Find the distances of all data points from that line

Learn by example

In [164…

```
import pandas as pd
df = pd.read_csv("ames-housing.csv")
df.plot.scatter(x='Living Area Sqft', y='SalePrice');
```





Linear equation general form:

$\hat{y}$: predicted y

x_1, x_2, x_3...etc: input features

$\beta_1, \beta_2, \beta_3$...etc: coefficients for the corresponding xs.

β_0 : intercept

$$\hat{y} = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3 + ... + \beta_n * x_n$$

How to find the *optimal* line ?

- **Optimizer**: Its primary role is to find the best values for model parameters that minimize the model’s error or loss function.
- Model parameters = weights = coefficients
- Examples of ML Optimizers:
  - **GD**: (Gradient Descent)
  - **SGD**: Stochastic Gradient Descent)
  - **Adagrad** (Adaptive Gradient Algorithm)
  - **Adam** (Adaptive Moment Estimation)

Training a linear regression model

Execute preprocessing steps

In [165…

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
```

```
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler


from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn import set_config
set_config(transform_output='pandas')

df = pd.read_csv("galton_handson.csv")
df.head()
```

Out[165…

	family	father	mother	midparentHeight	children	childNum	gender	childHeight	fan
0	1	78.5	67.0	75.43	4	1.0	male	73.2	
1	1	78.5	67.0	75.43	4	2.0	female	69.2	
2	1	78.5	67.0	75.43	4	3.0	female	69.0	
3	1	78.5	67.0	75.43	4	4.0	female	69.0	
4	2	75.5	66.5	73.66	4	1.0	male	73.5	

Building a machine learning model that predicts a child's height based on historical data.

First step : prepare and clean dataset

In [166…

```
#Drop non relevant columns
df = df.drop(columns='family')

#Fix inconsistency
df['midparentHeight'] = df['midparentHeight'].str.replace(",", '.')
df['midparentHeight'] =  df['midparentHeight'].astype(float)

df = df.drop_duplicates()

# The target we are trying to predict
y = df['childHeight']
# The features we will use to make the prediction
X = df.drop(columns = 'childHeight')
X.head(5)
```

Out[166…

	father	mother	midparentHeight	children	childNum	gender	familySize
0	78.5	67.0	75.43	4	1.0	male	Mid
1	78.5	67.0	75.43	4	2.0	female	Mid
2	78.5	67.0	75.43	4	3.0	female	Mid
3	78.5	67.0	75.43	4	4.0	female	Mid
4	75.5	66.5	73.66	4	1.0	male	Mid

In [167…

```
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

## Three types of data
num_cols = X_train.select_dtypes("number").columns
ordinal_cols = ['familySize']
ohe_cols = X_train.select_dtypes("object").drop(columns=ordinal_cols).columns

########### Numrical pipeline
# instantiate preprocessors
# Fills missing values with the median
impute_median = SimpleImputer(strategy='median')
#cales features to have mean = 0, std = 1
scaler = StandardScaler()

# Make a numeric preprocessing pipeline
num_pipe = make_pipeline(impute_median, scaler)

########### Ordinal pipeline
'''Fills missing with 'NA'
Encodes ordered categories (Small, Mid, Large)
Scales them numerically'''
impute_na_ord = SimpleImputer(strategy='constant', fill_value='NA')

# Specifying the order of categories in quality/condition columns
fam_size_order = ["Small", "Mid", "Large"]
# Making the list of order lists for OrdinalEncoder
ordinal_category_orders = [fam_size_order]
# Instantiate the encoder and include the list of ordered values as an argument
ord_encoder = OrdinalEncoder(categories=ordinal_category_orders)

# Making a final scaler to scale category #'s
scaler_ord = StandardScaler()

ord_pipe = make_pipeline(impute_na_ord, ord_encoder, scaler_ord)

######### Nominal pipeline
impute_na = SimpleImputer(strategy='constant', fill_value = "male")

# Instantiate one hot encoder
'''handle_unknown='ignore' : Ignores unseen (unknown) categories during transformat
Fills the one-hot vector with all zeros for that value to prevents the code from cr
ohe_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
```

```python
# Make pipeline with imputer and encoder
ohe_pipe = make_pipeline(impute_na, ohe_encoder)

######### Build pipelines tuples
num_tuple = ('numeric', num_pipe, num_cols)
ord_tuple = ('ordinal', ord_pipe, ordinal_cols)
ohe_tuple = ('categorical', ohe_pipe, ohe_cols)

######### Create ColumnTransformer
# Instantiate with verbose_feature_names_out=False
# remainder: Keep unlisted columns unchanged
'''False => Keeps clean column names (e.g., 'gender_female'),
True => Adds transformer name as prefix (e.g., 'categorical__gender_female')'''
col_transformer = ColumnTransformer([num_tuple, ord_tuple, ohe_tuple],
                                    remainder='passthrough',
                                    verbose_feature_names_out=False)

# Fit on training data
col_transformer.fit(X_train)
# Transform the training data
X_train_tf = col_transformer.transform(X_train)
# Transform the testing data
X_test_tf = col_transformer.transform(X_test)
# View the processed training data
X_train_tf
```

Out[167…

	father	mother	midparentHeight	children	childNum	familySize	gender_fema
385	0.519228	-0.922169	-0.260641	0.676793	1.898582	0.813335	1
453	0.096910	1.030101	0.834581	-0.784453	-0.262833	-0.698689	1
347	0.308069	-0.054494	0.210361	1.042104	1.898582	0.813335	1
602	-0.325409	0.379344	0.091192	0.676793	-0.695116	0.813335	0
622	-0.536568	-0.054494	-0.357112	1.407416	1.034016	0.813335	1
...	...	...	...	...	...	...	
767	-0.958887	-0.271412	-0.794066	0.676793	0.169450	0.813335	0
72	1.448329	2.114696	2.508626	0.676793	0.601733	0.813335	1
908	-1.592365	-1.789845	-2.292194	-0.419141	0.169450	-0.698689	0
235	0.730388	-2.657521	-1.344514	-1.515075	-0.695116	-2.210713	0
37	1.997344	-0.922169	0.732436	0.676793	0.601733	0.813335	1

699 rows × 8 columns

Fit a Linear regression model

In [168…

```
from sklearn.linear_model import LinearRegression
# from sklearn.linear_model SGDRegressor
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train_tf, y_train)
model
```

Out[168…

```
▾ LinearRegression
LinearRegression()
```

By calling the *fit* function, the model studied the patterns between the features and the target, and found the optimal weights which mimize the loss

Look to the model parameters/weights:

In [169…

```
print(X_train.describe())
```

```
           father      mother  midparentHeight    children    childNum
count  683.000000  675.000000       693.000000  699.000000  695.000000
mean    69.265154   64.130074        69.188603    6.147353    3.611511
std      2.396947    2.347244         1.771073    2.739351    2.321156
min     62.000000   58.000000        64.400000    1.000000    1.000000
25%     68.000000   63.000000        68.230000    4.000000    2.000000
50%     69.500000   64.000000        69.270000    6.000000    3.000000
75%     71.000000   66.000000        70.140000    8.000000    5.000000
max     78.500000   70.500000        75.430000   15.000000   11.000000
```

In [170…

```
b_0 = model.intercept_.round(2)
b_i = model.coef_.round(2)
feature_names = col_transformer.get_feature_names_out()
print(f"features: {feature_names}")
print(f"coffiecients (weights): {b_i}")
print(f"intercept (bias): {b_0}")
```

```
features: ['father' 'mother' 'midparentHeight' 'children' 'childNum' 'familySize'
 'gender_female' 'gender_male']
coffiecients (weights): [ 0.63  0.33  0.45  0.58 -1.55  0.19 -1.73  1.73]
intercept (bias): 66.62
```

In [171…

```
eq = " +\n".join([ f"{'' if i == 0 else ' '*24}{b_i[i]} * {feature_names[i]}"  for
print(f"Model equation: y_hat = {eq}")
```

```
Model equation: y_hat = 0.63 * father +
                        0.33 * mother +
                        0.45 * midparentHeight +
                        0.58 * children +
                        -1.55 * childNum +
                        0.19 * familySize +
                        -1.73 * gender_female +
                        1.73 * gender_male
```

Interpret 0.63 (father 'weight'):

- When the father height incease by one centemeter, the child hight expected to increase by 0.63, holding other features constant.

Is the above sentence correct after scaling ?

or :

When the father height incease by one **STD**, the child hight expected to increase by 0.63, holding other features constant.

A 1 cm increase in father's height leads to a 0.263 cm increase in child height, holding all other features constant

1 cm = 0.63 (Coef) / 2.396947 (std)

Interpret 0.33 (mother weight):

Interpret -1.55 (childNum weight):

Interpret -1.73 (gender_F weight):

Predict new childs

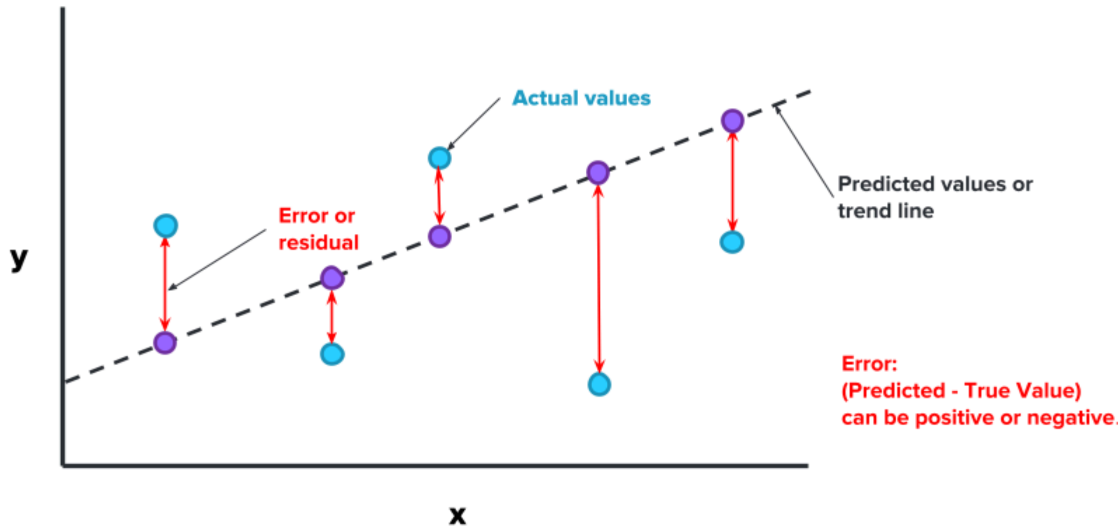
In [175…

```
# import numpy as np
#['father' 'mother' 'midparentHeight' 'children' 'childNum' 'familySize' 'gender_fe
new_child = [72, 64, 69.6, 3, 3, 'male', 'Small']
X = pd.DataFrame(columns=X_train.columns)
X.loc[0] = new_child
X = col_transformer.transform(X)
#print(X)
model.predict(X)
```

Out[175…

```
array([68.48076626])
```

Model evaluation



$$\text{Error/Residual} = y - \hat{y}$$

where:

- y is the true values for the target.
- $\hat{y}$: (y-hat) is the predicted values

Mean Absolute Error (MAE)

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

where:

- $|y_i - \hat{y}_i|$ represents taking the absolute value of the error.
- $\sum_{i=1}^{n}$: represents summing the values for every row/prediction.
- n represents the total number of rows/predictions.

Mean Squared Error (MSE)

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

where:

- $(y_i - \hat{y}_i)^2$ represents squaring the error.
- $\sum_{i=1}^{n}$: represents summing the values for every row/prediction.
- n represents the total number of rows/predictions.

Which is better, MSE or MAE ?

- Mean squared error penalizes large errors more because the errors are being squared.
- Interpretation is hard

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

where:

- $\sqrt{\ldots}$ represents taking the square root
- $(y_i - \hat{y}_i)^2$ represents squaring the error.
- $\sum_{i=1}^{n}$: represents summing the values for every row/prediction.
- n represents the total number of rows/predictions.

How to decide if the error metrics have acceptable/good values ?

MSE and RMSE scores are all dependent on the scale and units of the target. 5,000 USD on the sale of a house isn't too bad, but being off by 5,000 USD on the sale of a car would be horrible!

MAE, MSE, and RMSE scores are all dependent on the scale and units of the target. 5,000 USD on the sale of a house isn't too bad, but being off by 5,000 USD on the sale of a car would be horrible!

Let's evaluate our model

In [140…

```
# Calculating RMSE with sklearn
from sklearn.metrics import root_mean_squared_error

#Get predictions for test data
y_pred_test = model.predict(X_test_tf)

test_RMSE = root_mean_squared_error(y_test, y_pred_test)
print(f'Model Testing RMSE: {test_RMSE:,.2f}')
```

```
Model Testing RMSE: 1.86
```

What this result means : On average, our model's predictions are off by ~1.86 inch from the actual child height

R-Squared

- Coefficient of determination
- Describes the percentage of the variation in the target variable that a model can explain by using all the features togethe

Pros

- Consistent scale

Cons

- Difficult to interpret
- A high R2 doesn't always mean a good model and a low one doesn't always mean a bad one.

R^2 (Coefficient of Determination)

The R^2 score measures how well the regression model explains the variability in the target variable.

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

Interpretation:

- **SSE** (Sum of Squared Errors) measures how far predictions are from actual values.
- **SST** (Total Sum of Squares) measures how far actual values are from their mean.
- The closer R^2 is to **1**, the better the model explains the data.

$$R^2 = 1 - \frac{\sum(y_{\text{true}} - y_{\text{pred}})^2}{\sum(y_{\text{true}} - \bar{y})^2}$$

In [141…

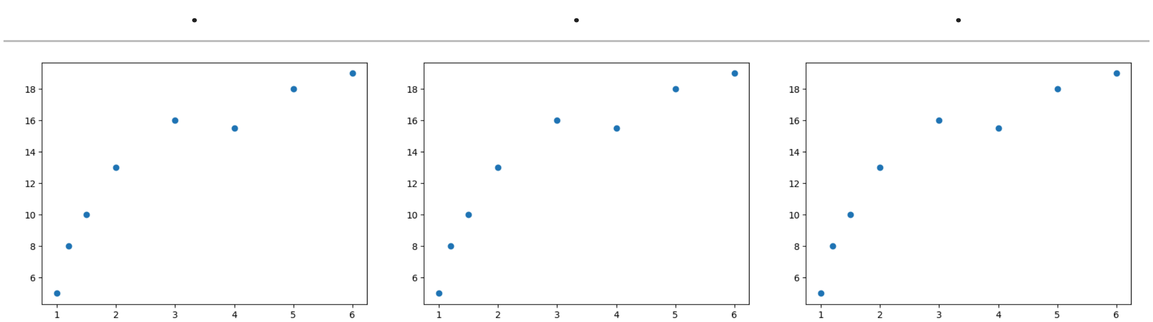
```
from sklearn.metrics import r2_score

y_pred = model.predict(X_test_tf)
r2 = r2_score(y_test, y_pred)
print("R² score:", r2)
```

```
R² score: 0.7112764091636733
```

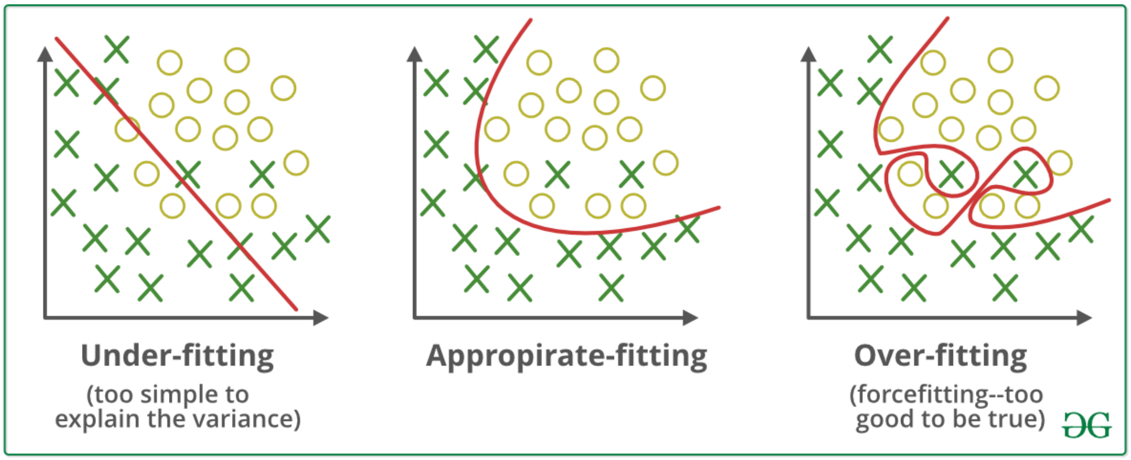
R^2 Score Interpretation

- 0.9 – 1.0 Excellent fit
- 0.7 – 0.9 Good fit
- 0.5 – 0.7 Moderate — model explains some, but not all
- < 0.5 Weak — model is underperforming

Bias vs. Variance



Bias and Variance on LR



How to detect underfit ?

The model shows bad quality in both train and test data

How to detect overfit ?

1. **Evaluate the model on both the training and testing data, then**
2. **If there is a significant gap between both quality => Overfit**

Example 1 : How you interpret the quality of a model predicting house prices ?

Regression Metrics: Train Data

- MSE = 15,876.00
- RMSE = 126.00

Regression Metrics: Test Data

- MSE = 302,500.00
- RMSE = 550.00

Overfit/High varianc since there is a significant gap in RMSE/MSE between train and test data

Example 2 : How you interpret the quality of a model predicting child height ?

Regression Metrics: Train Data

- MSE = 36.00
- RMSE = 6.00

Regression Metrics: Test Data

- MSE = 37.00
- RMSE = 6.08

Underfit/High bias since MSE/RMSE is high on both train and test data

Example 3 : How you interpret the quality of a model predicting number of students will enroll in BZU next semester?

Regression Metrics: Train Data

- MSE = 90,000.00
- RMSE = 300.00

Regression Metrics: Test Data

- MSE = 100,200.00
- RMSE = 316.54

- **No overfit**: since the error on both train and test is close
- **No underfit**: since the error amout of ~300 student is acceptable comparing to the total number of students usually enroll (~5,000?)

=> Acceptable model

High bias (underfitting):

Reasons:

- A model that is too simple
- Not enough data.
- Features that do not correlate well with the target.

How to fix:

- Increasing the complexity of your model. (Add more features)
- Adding more data.
- Adding features with higher correlation to the target.

High Variance (Overfitting)

Reasons:

- Too complex of a model
- Not enough data.

- Training data that is not a representative sample of the population, all new data the model might encounter.

How to fix:

- Decrease the complexity of the model. (feature selection)
- Add regularization.
- Add more data to the training set.
- Ensure that the training set is a representative subset of all data the model will encounter.

End-to-End example

In [1]:
```
import pandas as pd
from sklearn.compose import ColumnTransformer, make_column_selector
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn import set_config
set_config(transform_output='pandas')
```

In [2]:
```
df = pd.read_csv('medical_data.csv')
df.head()
```

Out[2]:

	State	Lat	Lng	Area	Children	Age	Income	Marital	Gender	ReAd
0	AL	34.34960	-86.72508	Suburban	1.0	53	86575.93	Divorced	Male	
1	FL	30.84513	-85.22907	Urban	3.0	51	46805.99	Married	Female	
2	SD	43.54321	-96.63772	Suburban	3.0	53	14370.14	Widowed	Female	
3	MN	43.89744	-93.51479	Suburban	0.0	78	39741.49	Married	Male	
4	VA	37.59894	-76.88958	Rural	1.0	22	1209.56	Widowed	Female	

5 rows × 32 columns

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 32 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   State               995 non-null    object 
 1   Lat                 1000 non-null   float64
 2   Lng                 1000 non-null   float64
 3   Area                995 non-null    object 
 4   Children            993 non-null    float64
 5   Age                 1000 non-null   int64  
 6   Income              1000 non-null   float64
 7   Marital             995 non-null    object 
 8   Gender              995 non-null    object 
 9   ReAdmis             1000 non-null   int64  
 10  VitD_levels         1000 non-null   float64
 11  Doc_visits          1000 non-null   int64  
 12  Full_meals_eaten    1000 non-null   int64  
 13  vitD_supp           1000 non-null   int64  
 14  Soft_drink          1000 non-null   int64  
 15  Initial_admin       995 non-null    object 
 16  HighBlood           1000 non-null   int64  
 17  Stroke              1000 non-null   int64  
 18  Complication_risk   995 non-null    object 
 19  Overweight          1000 non-null   int64  
 20  Arthritis           994 non-null    float64
 21  Diabetes            994 non-null    float64
 22  Hyperlipidemia      998 non-null    float64
 23  BackPain            992 non-null    float64
 24  Anxiety             998 non-null    float64
 25  Allergic_rhinitis   994 non-null    float64
 26  Reflux_esophagitis  1000 non-null   int64  
 27  Asthma              1000 non-null   int64  
 28  Services            995 non-null    object 
 29  Initial_days        1000 non-null   float64
 30  TotalCharge         1000 non-null   float64
 31  Additional_charges  1000 non-null   float64
dtypes: float64(14), int64(11), object(7)
memory usage: 250.1+ KB
```

In [144…

```
# Drop State
droplist = ['State']
df = df.drop(droplist, axis=1)

# Correct inconsistencies in values in Gender column
df['Gender'] = df['Gender'].replace(['male', 'm', 'M'], 'Male')
df['Gender'] = df['Gender'].replace(['F', 'f'], 'Female')

# Define X and y
X = df.drop(columns='Additional_charges')
y = df['Additional_charges']
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Create the preprocessing pipeline for categorical data
# (New) Select columns with make_column_selector
```

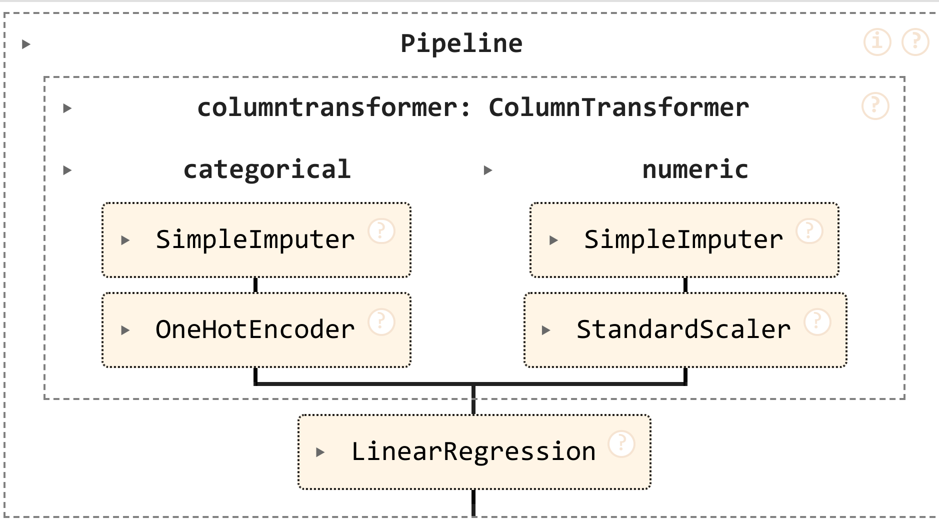
```python
cat_selector = make_column_selector(dtype_include='object')
# Insantiate transfomers
freq_imputer = SimpleImputer(strategy='most_frequent')
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
# Instantiate the pipeline
cat_pipe = make_pipeline(freq_imputer, ohe)
# Make a tuple for column transformer
cat_tuple = ('categorical',cat_pipe, cat_selector)

# Create the preprocessing pipeline for numeric data
# (New) Select columns wiht make_column)selector
num_selector = make_column_selector(dtype_include='number')
# Instantiate the transformers
scaler = StandardScaler()
mean_imputer = SimpleImputer(strategy='mean')
# Instantiate the pipeline
num_pipe = make_pipeline(mean_imputer, scaler)
# Make the tuple for ColumnTransformer
num_tuple = ('numeric',num_pipe, num_selector)

# Create the preprocessing ColumnTransformer
preprocessor = ColumnTransformer([cat_tuple, num_tuple],
                                 verbose_feature_names_out=False)
```

Create model pipeline

In [145…

```python
from sklearn.linear_model import LinearRegression

# Instantiate a linear regression model
model = LinearRegression()
# Combine the preprocessing ColumnTransformer and the linear regression model in a
model_pipe = make_pipeline(preprocessor, model)
model_pipe
```

Out[145…

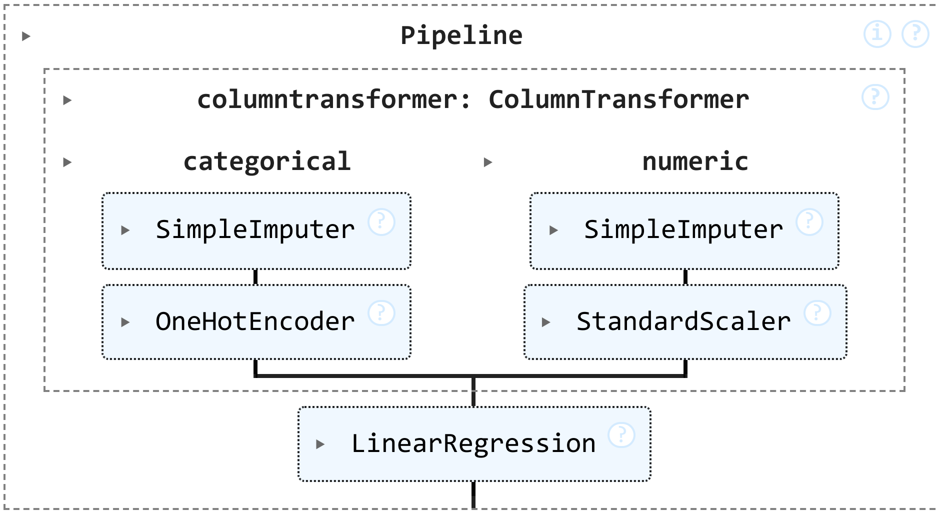


Fit the model pipeline on the training data

In [146…

```python
model_pipe.fit(X_train, y_train)
```

Out[146…



Model evaluation

Steps:

1. **Evaluate the model on the train data**
    - Use the model to predict the train data (`X_train`) => `y_train_pred`
    - Compare between predicted values(`y_train_pred`) and the actual values (`y_train`)
    - Calculate the quality metrics (MSE/RMSE) using `mean_squared_error` and `root_mean_squared_error` from `sklearn` module
2. **Evaluate the model on the test data**
    - Use the model to predict the train data (`X_test`) => `y_test_pred`
    - Compare between predicted values(`y_test_pred`) and the actual values (`y_test`)
    - Calculate the quality metrics (MSE/RMSE) using `mean_squared_error` and `root_mean_squared_error` from `sklearn` module
3. **Identify underfit and overfit (if any)**

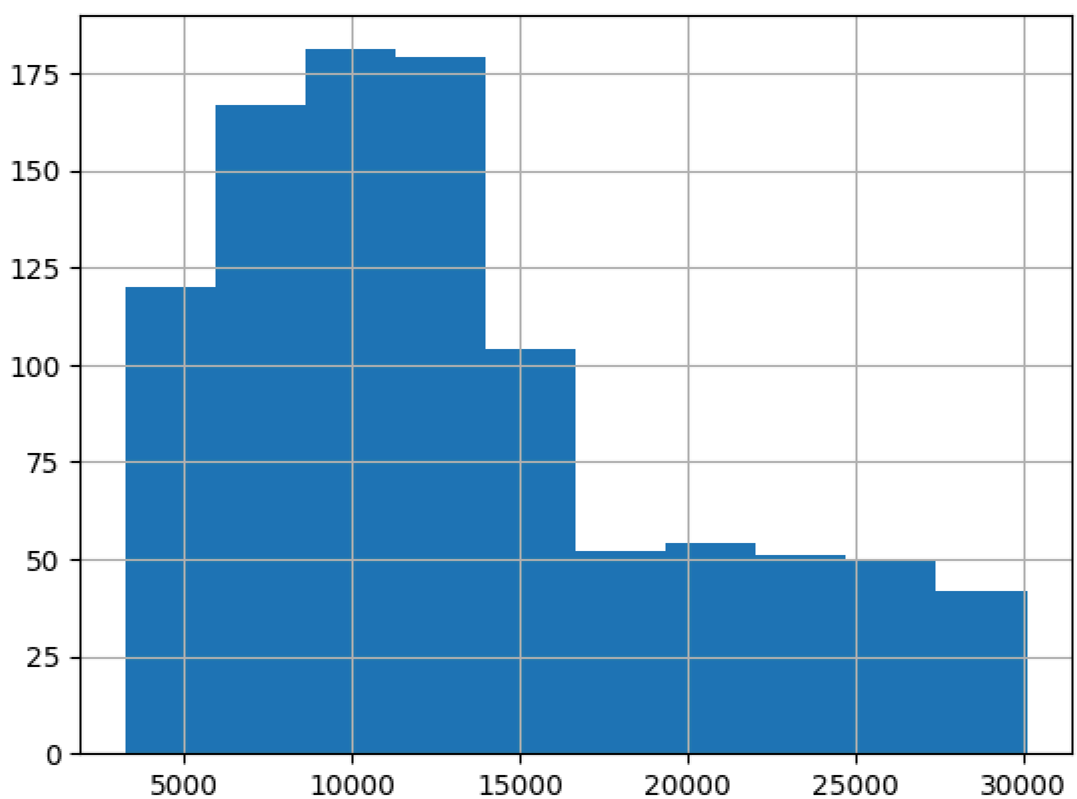
In [147…

```python
from sklearn.metrics import mean_squared_error, root_mean_squared_error, mean_absol

def regression_metrics(y_true, y_pred, label='', verbose = True, output_dict=False)
    # Get metrics
    rmse = root_mean_squared_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    if verbose == True:
        # Print Result with Label and Header
        header = "-"*60
        print(header, f"Regression Metrics: {label}", header, sep='\n')
        print(f"- MSE = {mse:,.3f}")
        print(f"- RMSE = {rmse:,.3f}")
    if output_dict == True:
        metrics = {'Label':label, 'MSE':mse, 'RMSE':rmse}
```

```python
# Get predictions for training data
y_train_pred = model_pipe.predict(X_train)

# Call the helper function to obtain regression metrics for training data
results_train = regression_metrics(y_train, y_train_pred, label='Training Data')
print()
# Get predictions for test data
y_test_pred = model_pipe.predict(X_test)
# Call the helper function to obtain regression metrics for test data
results_test = regression_metrics(y_test, y_test_pred, label='Test Data' )

df['Additional_charges'].hist()
```

```
------------------------------------------------------------
Regression Metrics: Training Data
------------------------------------------------------------
- MSE = 2,614,407.658
- RMSE = 1,616.913

------------------------------------------------------------
Regression Metrics: Test Data
------------------------------------------------------------
- MSE = 2,866,953.025
- RMSE = 1,693.208
```

Out[147… <Axes: >



Summary of training a model

- Clean and explore the data using EDA

- Preprocess the data
- Fit a machine learning algorithm (Linear Regression)
- Evaluate the quality of the model
- Improve model quality with proper techniques in respect to underfit and overfit problems

In [148…

```python
def regression_metrics(y_true, y_pred, label='', verbose = True, output_dict=False)
    # Get metrics
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    r_squared = r2_score(y_true, y_pred)
    if verbose == True:
        # Print Result with Label and Header
        header = "-"*60
        print(header, f"Regression Metrics: {label}", header, sep='\n')
        print(f"- MAE = {mae:,.3f}")
        print(f"- MSE = {mse:,.3f}")
        print(f"- RMSE = {rmse:,.3f}")
        print(f"- R^2 = {r_squared:,.3f}")
    if output_dict == True:
        metrics = {'Label':label, 'MAE':mae,
                  'MSE':mse, 'RMSE':rmse, 'R^2':r_squared}


# Get predictions for training data
y_train_pred = model_pipe.predict(X_train)

# Call the helper function to obtain regression metrics for training data
results_train = regression_metrics(y_train, y_train_pred, label='Training Data')
print()
# Get predictions for test data
y_test_pred = model_pipe.predict(X_test)
# Call the helper function to obtain regression metrics for test data
results_test = regression_metrics(y_test, y_test_pred, label='Test Data' )
```

```
------------------------------------------------------------
Regression Metrics: Training Data
------------------------------------------------------------
- MAE = 1,370.542
- MSE = 2,614,407.658
- RMSE = 1,616.913
- R^2 = 0.943

------------------------------------------------------------
Regression Metrics: Test Data
------------------------------------------------------------
- MAE = 1,389.304
- MSE = 2,866,953.025
- RMSE = 1,693.208
- R^2 = 0.930
```

```
C:\Users\asabb\anaconda3\envs\EnvML\Lib\site-packages\sklearn\metrics\_regression.p
y:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the function'root_mean_squared_er
ror'.
  warnings.warn(
C:\Users\asabb\anaconda3\envs\EnvML\Lib\site-packages\sklearn\metrics\_regression.p
y:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the function'root_mean_squared_er
ror'.
  warnings.warn(
```

Assignment 7

- Explore Kaggle and select a dataset for regression modeling. Ensure the dataset adheres to the following criteria:
  - Must include at least 5 features (excluding the target), with a mix of ordinal and nominal features.
  - The target feature must be numeric.
- Perform necessary data cleaning and preprocessing, The splitting of the data should use 80% train and 20% test ratio. **Use your student ID as the random_state when splitting**.
- Fit a linear regression model on the transformed data.
- Print the model parameters and provide an interpretation for each.
- Evaluate the model's quality using appropriate metrics.
- Determine if the model exhibits overfitting or underfitting and explain your reasoning.
- Provide link to your selected data set in Kaggle.

Any attempt of using generative AI will be considered as cheating !

In [149…

```
print(f"Regression Metrics: Train Data", sep='\n')
print(f"- MAE = {88:,.2f}")
print(f"- MSE = {15876:,.2f}")
print(f"- RMSE = {126:,.2f}")
print(f"- R^2 = {0.87:,.2f}")
print()
print(f"Regression Metrics: Test Data", sep='\n')
print(f"- MAE = {484:,.2f}")
print(f"- MSE = {302500:,.2f}")
print(f"- RMSE = {550:,.2f}")
print(f"- R^2 = {0.61:,.2f}")
```

```
Regression Metrics: Train Data
- MAE = 88.00
- MSE = 15,876.00
- RMSE = 126.00
- R^2 = 0.87

Regression Metrics: Test Data
- MAE = 484.00
- MSE = 302,500.00
- RMSE = 550.00
- R^2 = 0.61
```

In [150…

```python
import math
print(f"Regression Metrics: Train Data", sep='\n')
print(f"- MSE = {90000:,.2f}")
print(f"- RMSE = {math.sqrt(90000):,.2f}")
print()
print(f"Regression Metrics: Test Data", sep='\n')
print(f"- MSE = {100200:,.2f}")
print(f"- RMSE = {math.sqrt(100200):,.2f}")
```

```
Regression Metrics: Train Data
- MSE = 90,000.00
- RMSE = 300.00

Regression Metrics: Test Data
- MSE = 100,200.00
- RMSE = 316.54
```