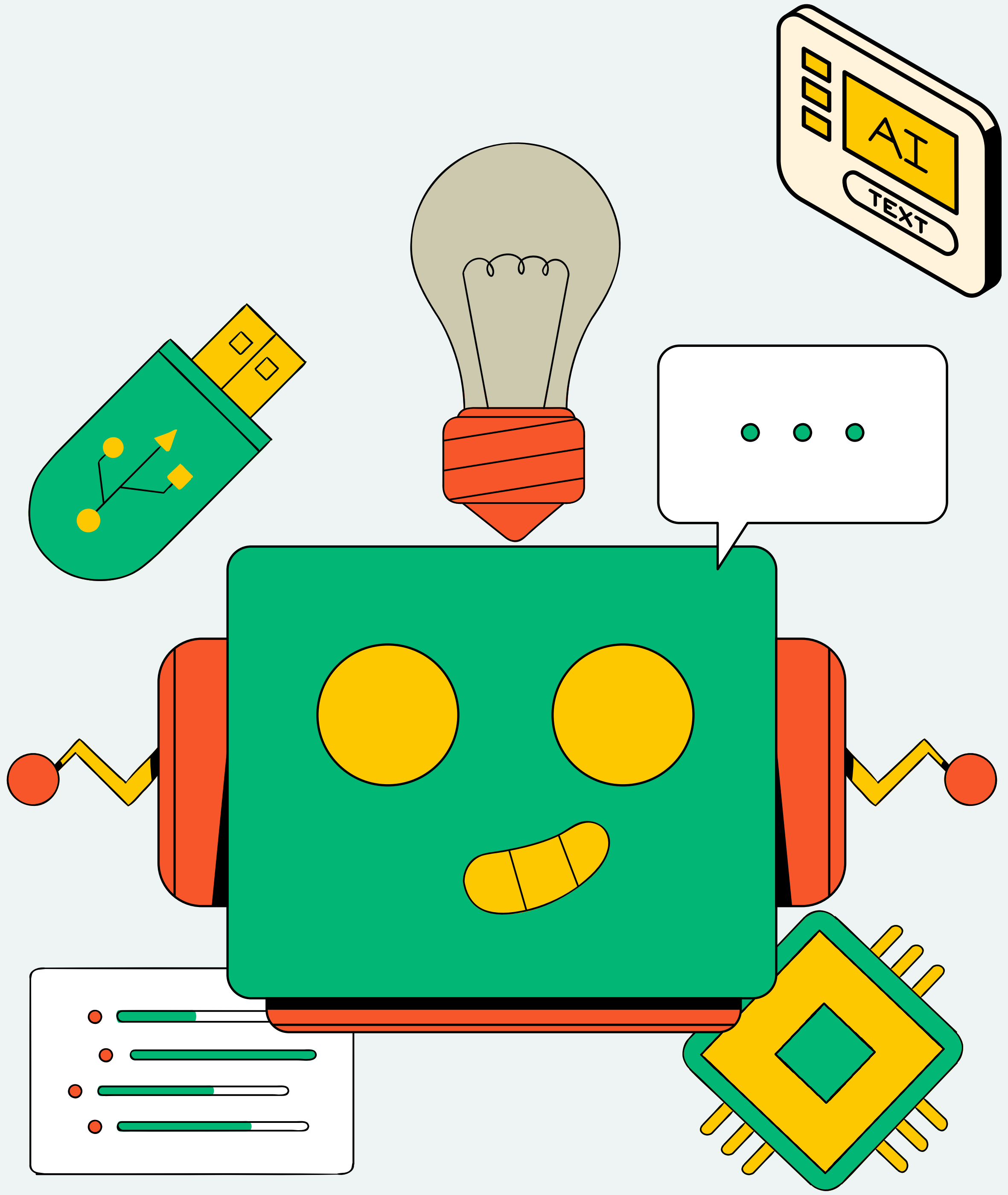
THYNK UNLIMITED

WE LEARN FOR THE FUTURE

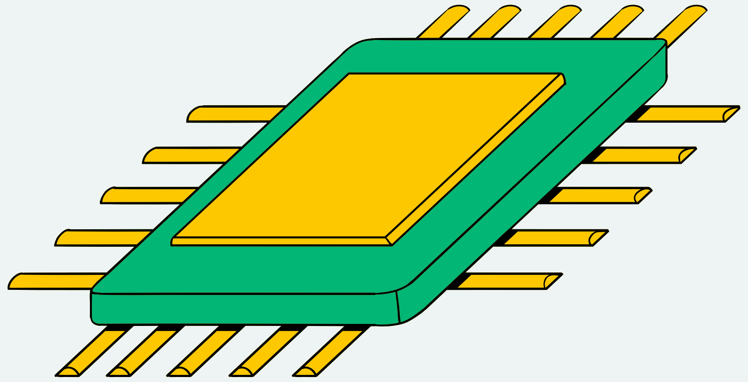


TRAFFIC PREDICTION

USING LOGISTIC REGRESSION

PRESENTED BY:

JIRATCHAYA PANPHINIJ
6310400941

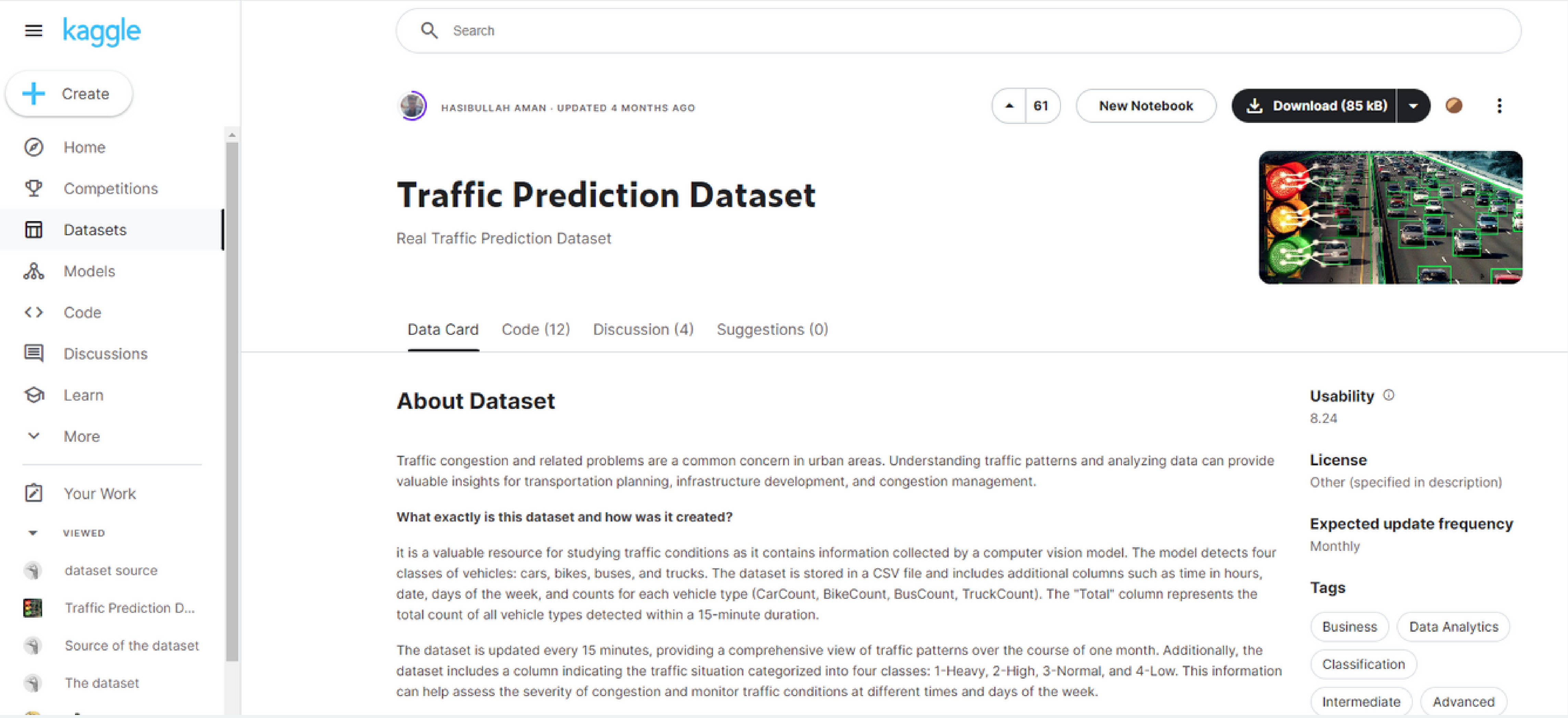


PRESENTATION OUTLINE

- Dataset
- Preprocessing Data
- Training
- Testing

DATASET

Afghanistan, Kabul, Abdul-haq Crossroad



DATASET

Time	: Time
Day of the week	: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
CarCount	: Number
BikeCount	: Number
BusCount	: Number
TruckCount	: Number
Total	: Number
Traffic Situation	: low, nurmal, high, heavy

	Time	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic Situation
0	12:00:00 AM	Tuesday	13	2	2	24	41	normal
1	12:15:00 AM	Tuesday	14	1	1	36	52	normal
2	12:30:00 AM	Tuesday	10	2	2	32	46	normal
3	12:45:00 AM	Tuesday	10	2	2	36	50	normal
4	1:00:00 AM	Tuesday	11	2	1	34	48	normal

PREPROCESSIONG DATA

	Time	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic Situation
0	12:00:00 AM	Tuesday	13	2	2	24	41	normal
1	12:15:00 AM	Tuesday	14	1	1	36	52	normal
2	12:30:00 AM	Tuesday	10	2	2	32	46	normal
3	12:45:00 AM	Tuesday	10	2	2	36	50	normal
4	1:00:00 AM	Tuesday	11	2	1	34	48	normal

	Time	Day of the week	CarCount	BikeCount	BusCount	TruckCount	Total	Traffic Situation
0	0	2	13	2	2	24	41	0
1	0	2	14	1	1	36	52	0
2	0	2	10	2	2	32	46	0
3	0	2	10	2	2	36	50	0
4	1	2	11	2	1	34	48	0

PREPROCESSIONG DATA

SPLIT TRAIN AND TEST DATA

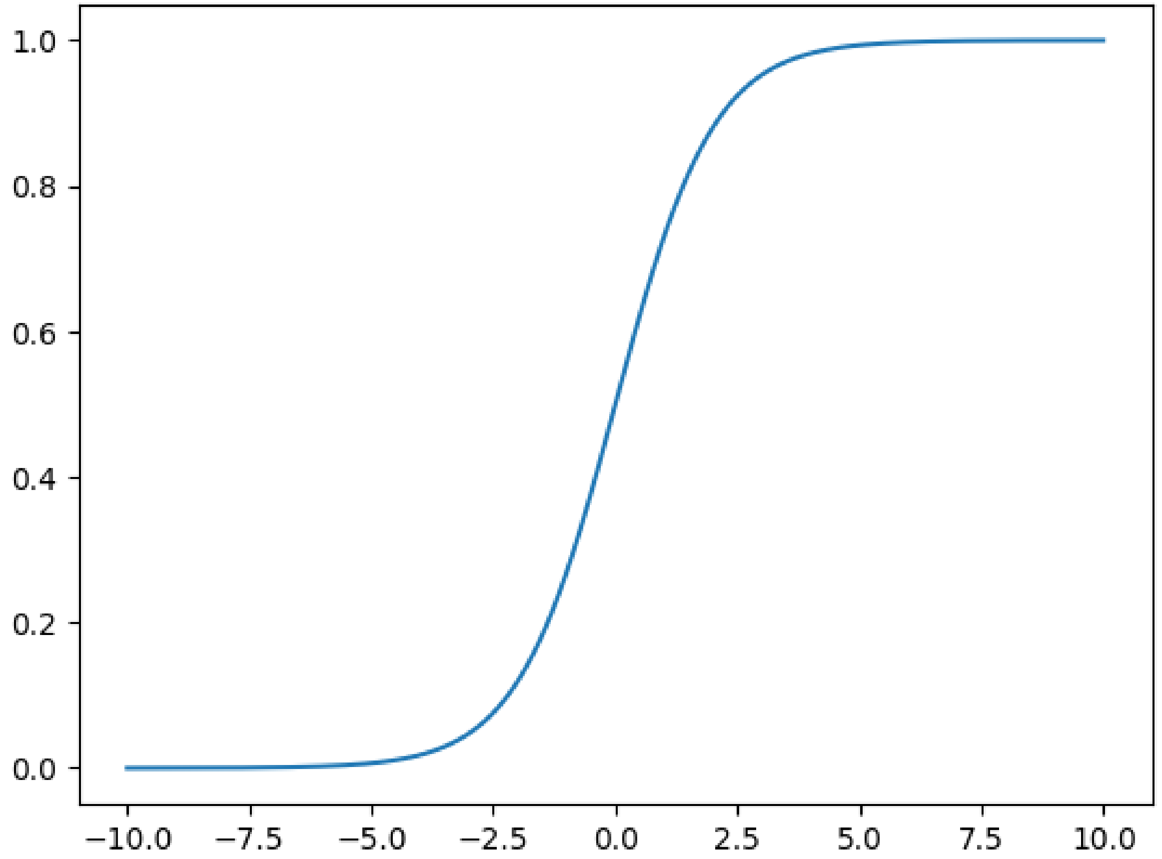
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

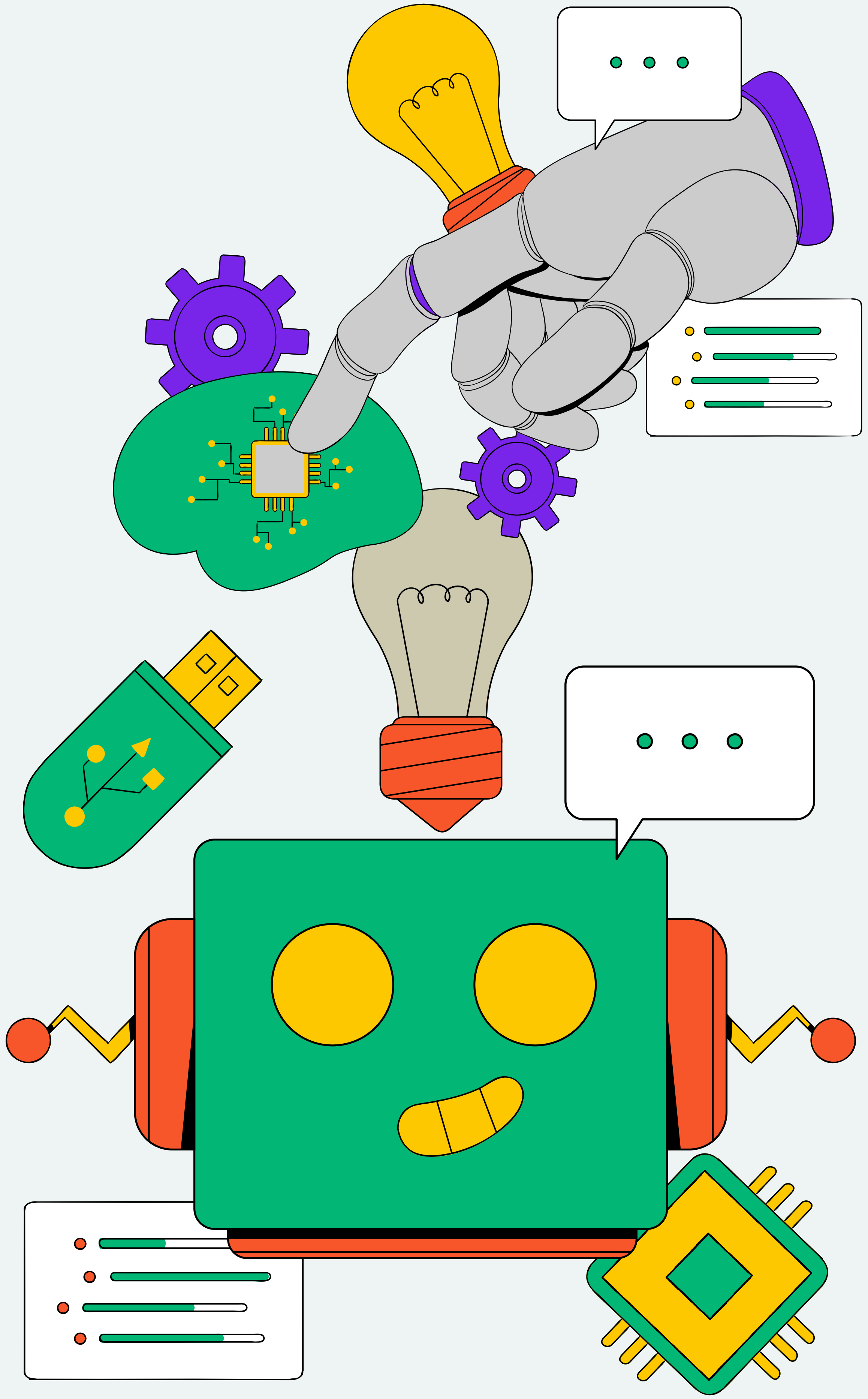
Training 80%	Test 20%

TRAINING

SIGMOID FUNCTION

$$P(y|x) = \frac{1}{1 + e^{-(w^T x + b)}}$$





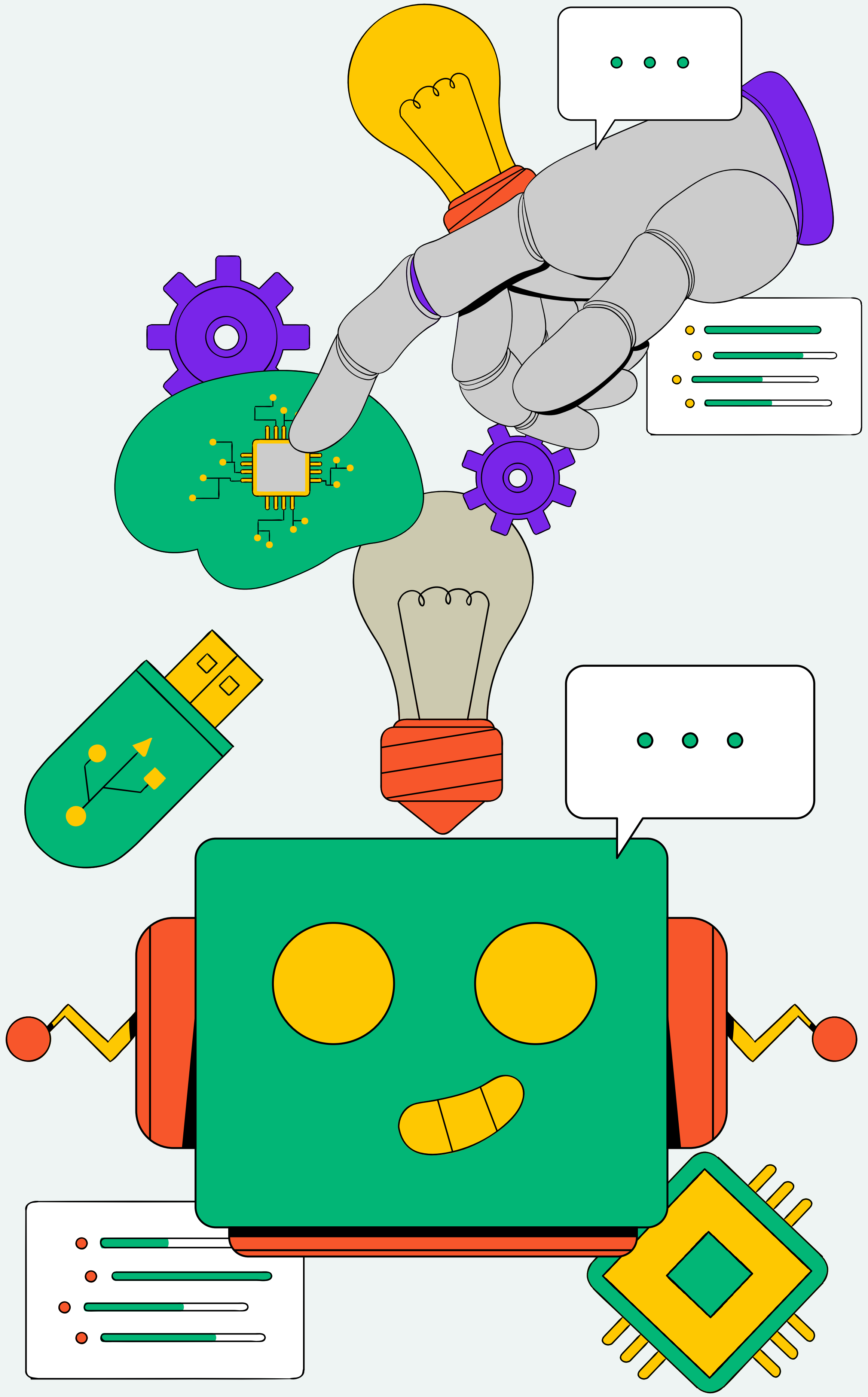
TRAINING

LOSS FUNCTION : BINARY CROSS ENTROPY

$$h_{w,b}(x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

$$J(w,b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(h_{w,b}(x_i)\right) + (1-y_i)\log\left(1 - h_{w,b}(x_i)\right)\right] + \frac{\lambda}{2}\sum_{j=1}^{d}|w_j|^2$$

```
def sigmoid(self, z):
 return 1/(1+np.exp(-z))

def cross_entropy(self, x, y):
 eps = 1e-15 # Small constant value to prevent division by zero
 z = np.dot(self.w, x.T) + self.b
 y_pred = self.sigmoid(z)
 return -(np.dot(y.T,np.log(y_pred + eps)) + np.dot((1-y).T, np.log(1-y_pred + eps)))/x.shape[0] +
self.c*np.sum(np.square(self.w))/(2*x.shape[1])
```



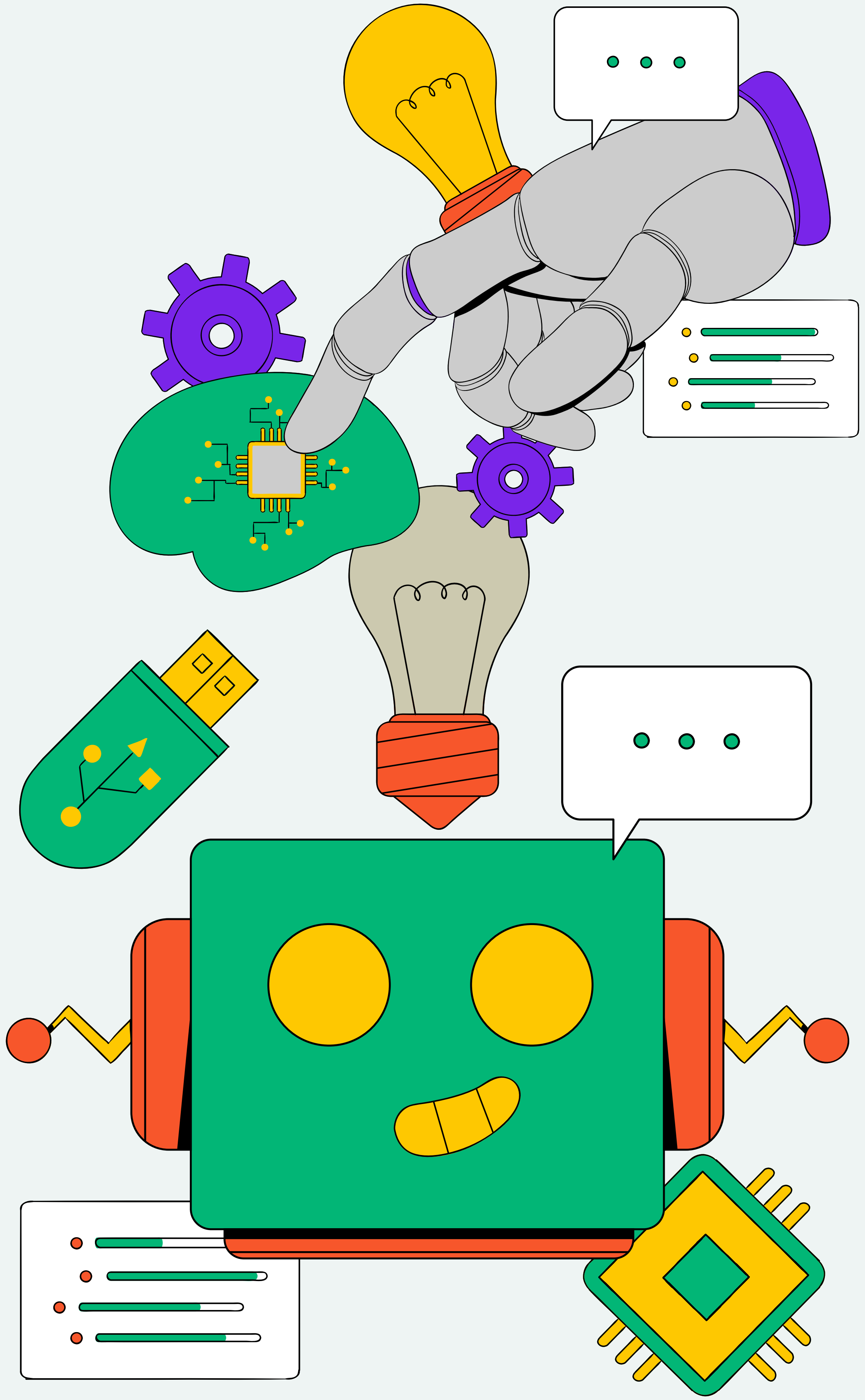
TRAINING

GRADIENT DESCENT

$$h_{w,b}(x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

$$\frac{\partial}{\partial w_j} J(w,b) = \frac{1}{n} \sum_{i=1}^{n} [(h_{w,b}(x_i) - y_i) x_i^j] + \lambda w_j$$

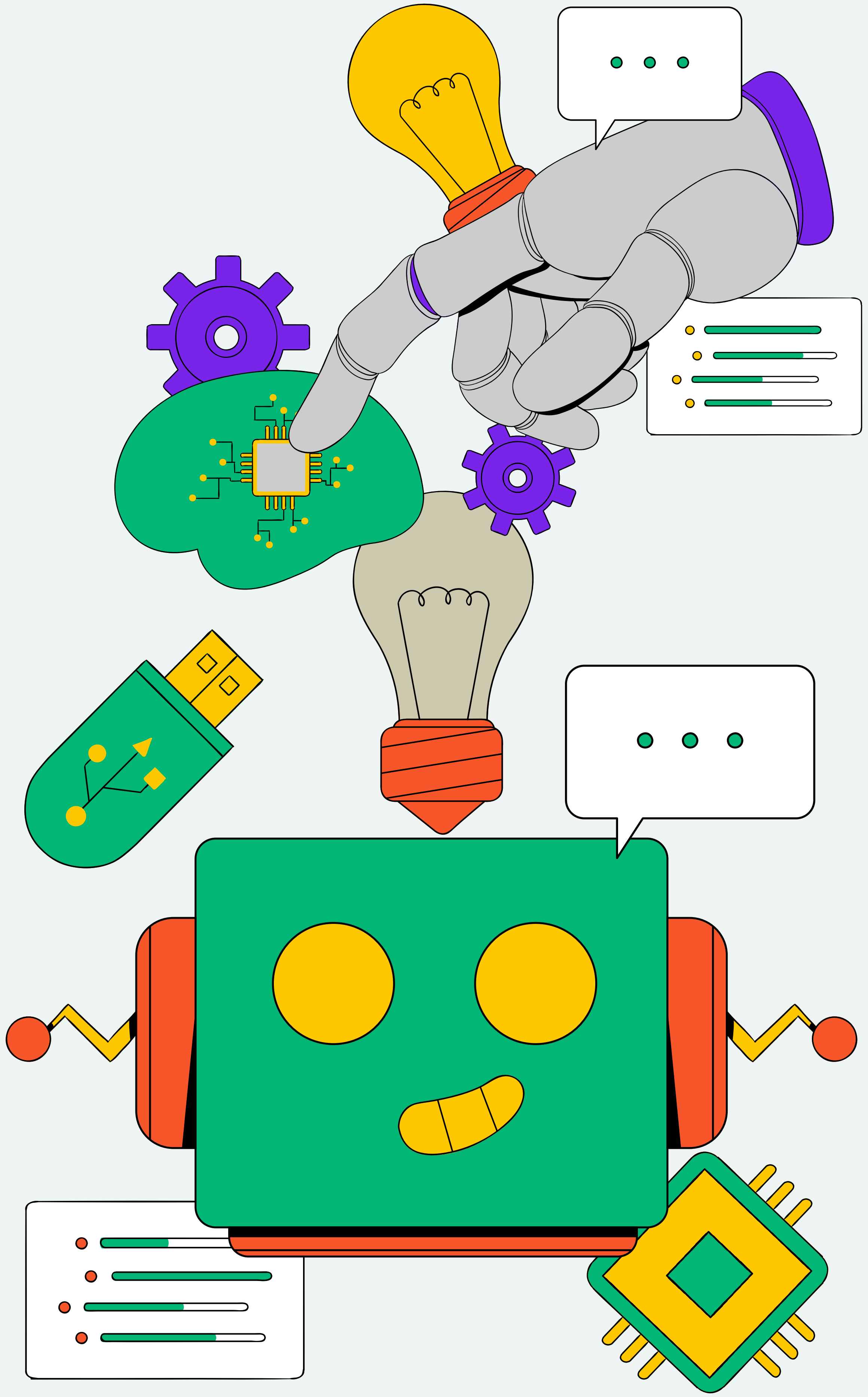
$$\frac{\partial}{\partial b} J(w,b) = \frac{1}{n} \sum_{i=1}^{n} (h_{w,b}(x_i) - y_i)$$



TRAINING

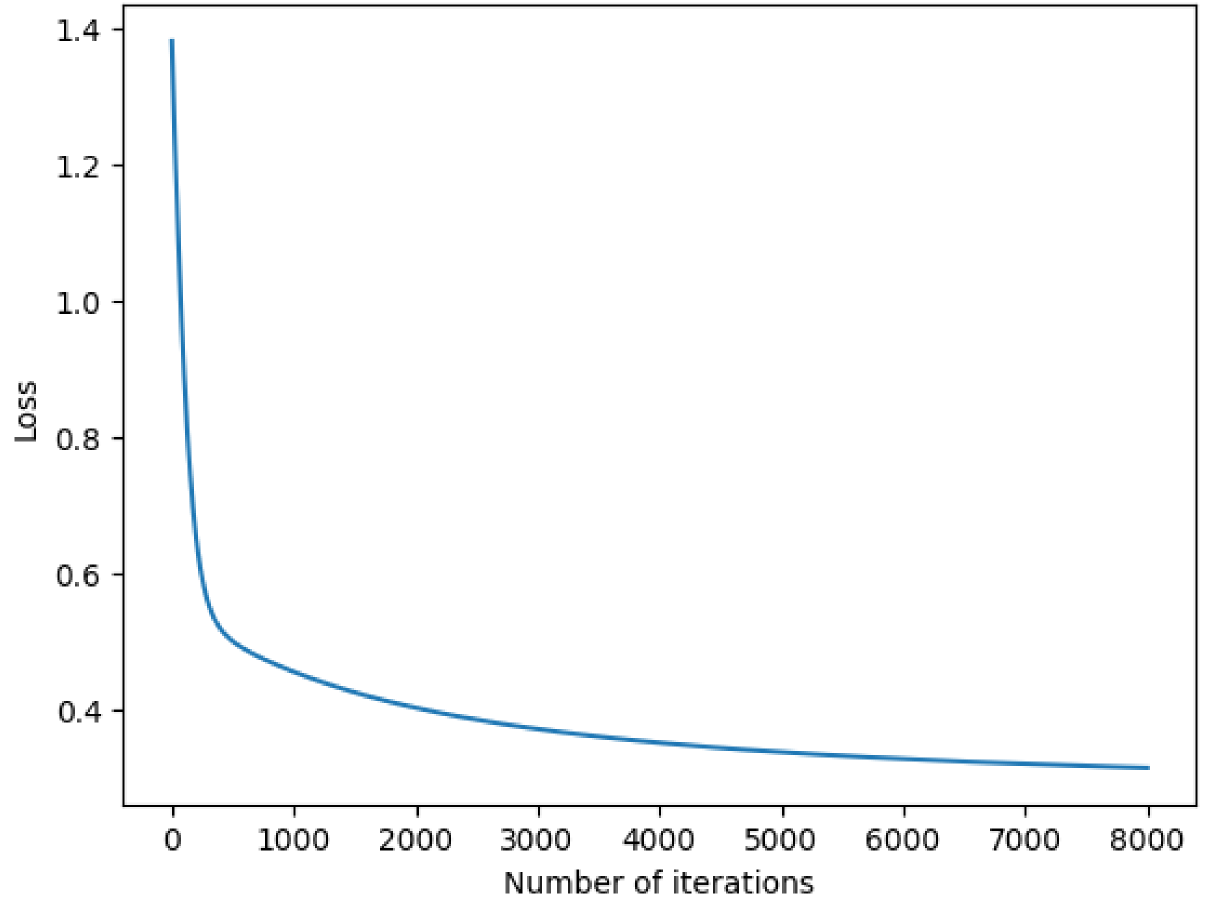
GRADIENT DESCENT

```
for i in range(self.iteration):
  # Random training data
  idx = np.random.choice(num_samples, int(batch_size*num_samples))
  x_batch = x.iloc[idx]
  y_batch = y[idx]

  # Sigmoid
  z = np.dot(self.w, x_batch.T) + self.b
  y_pred = self.sigmoid(z)

  # Calculate gradient
  gred_w = np.dot(x_batch.T, (y_pred – y_batch))/num_samples
  gred_b = np.sum(y_pred – y_batch)/num_samples

  # Regularization
  gred_w = gred_w + self.c*self.w

  # Update parameter
  self.w = self.w – (self.learning_rete*gred_w)
  self.b = self.b – (self.learning_rete*gred_b)
```



TRAINING

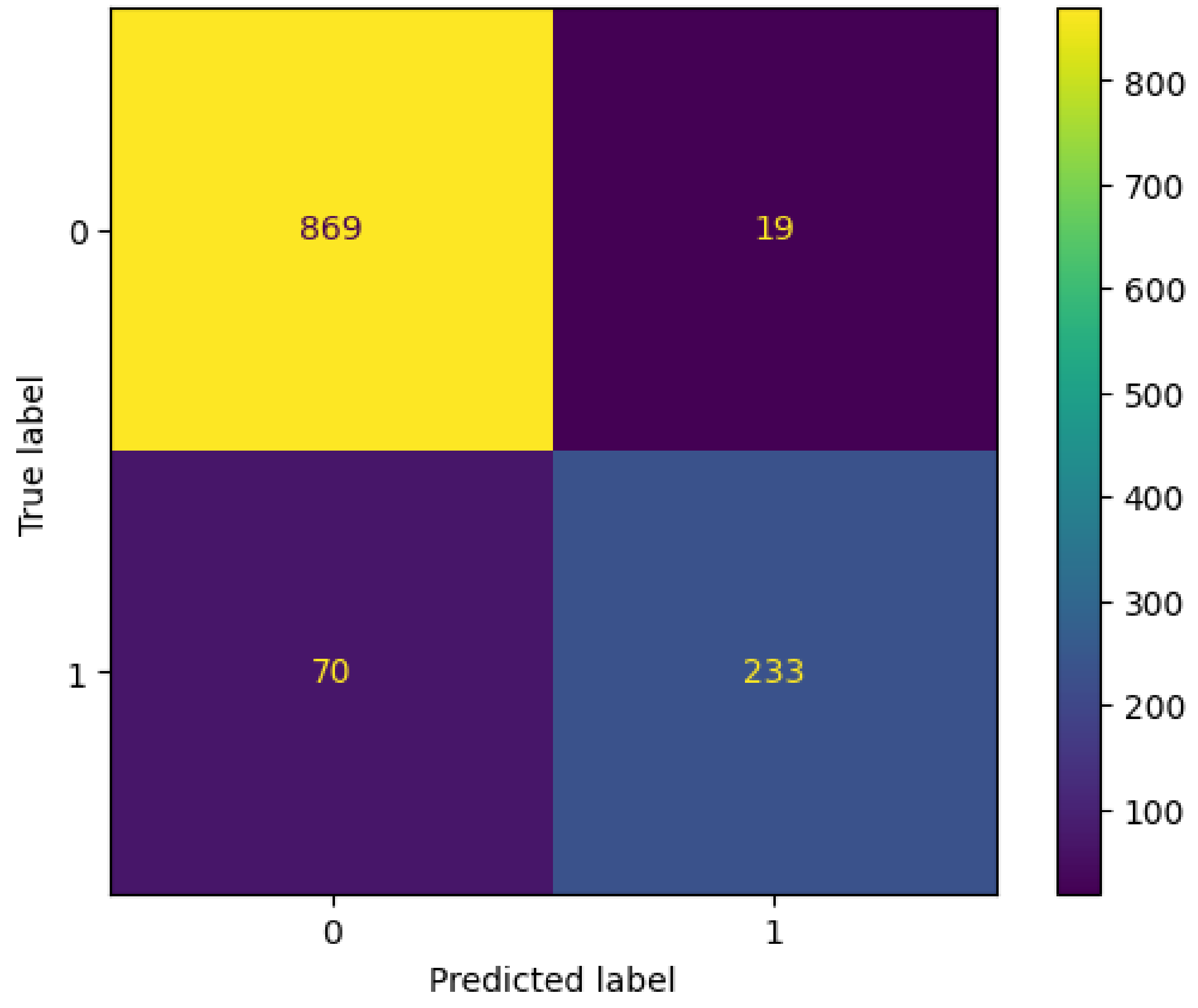
```
clf = LogisticRegression(iteration=8000, learning_rete=0.01, c=0.01, penalty='l2')
clf.fit(x_train, y_train, batch_size=0.8)
```



accuracy : 0.92
loss : 0.31

TESTING

accuracy : 0.92



THANK YOU