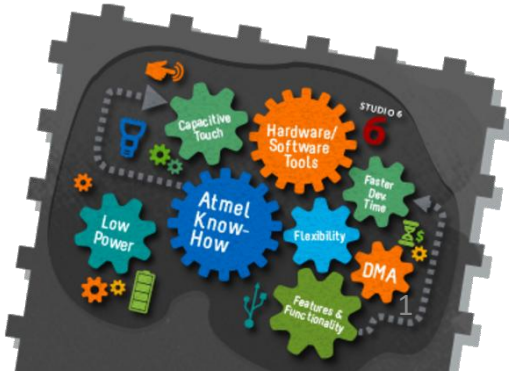
tiny Machine Learning



Intelligent edge devices with rich sensors (e.g., billions of mobile phones and IoT devices) have been ubiquitous in our lives.

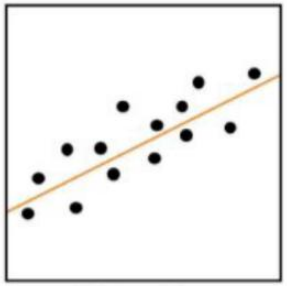
Deploying artificial intelligence (AI) to these edge devices is attractive for many real-world applications: smart home, smart retail, smart manufacturing, autonomous driving, and more.

However, state-of-the-art deep learning systems typically require tremendous amount of resources (e.g., memory, computation, data, AI experts), both for training and inference.

This hinders the application powerful deep learning systems on edge devices.

TinyML project aims to improve the efficiency of deep learning by requiring ***less computation***, ***fewer engineers***, and ***less data***, from algorithm to hardware, from inference to training. Embrace the era of edge AI and AIoT.

Common Challenges

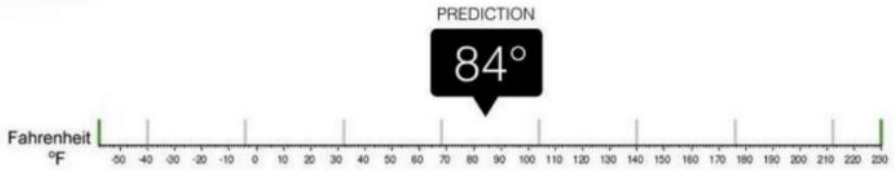


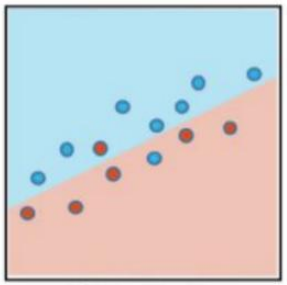
a) Regression

Regression

Regression

What is the temperature going to be tomorrow?



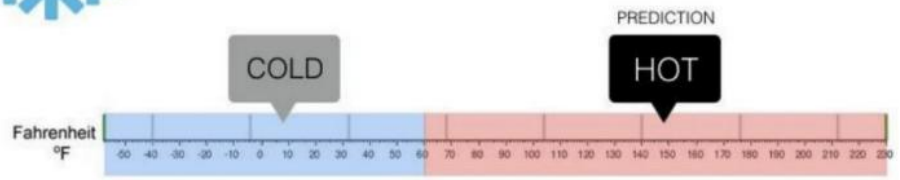


b) Classification

Classification

Classification

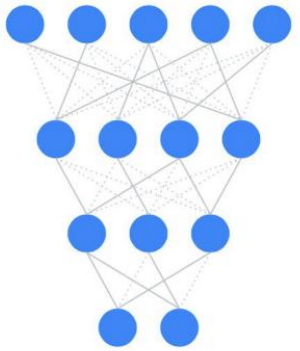
Will it be Cold or Hot tomorrow?



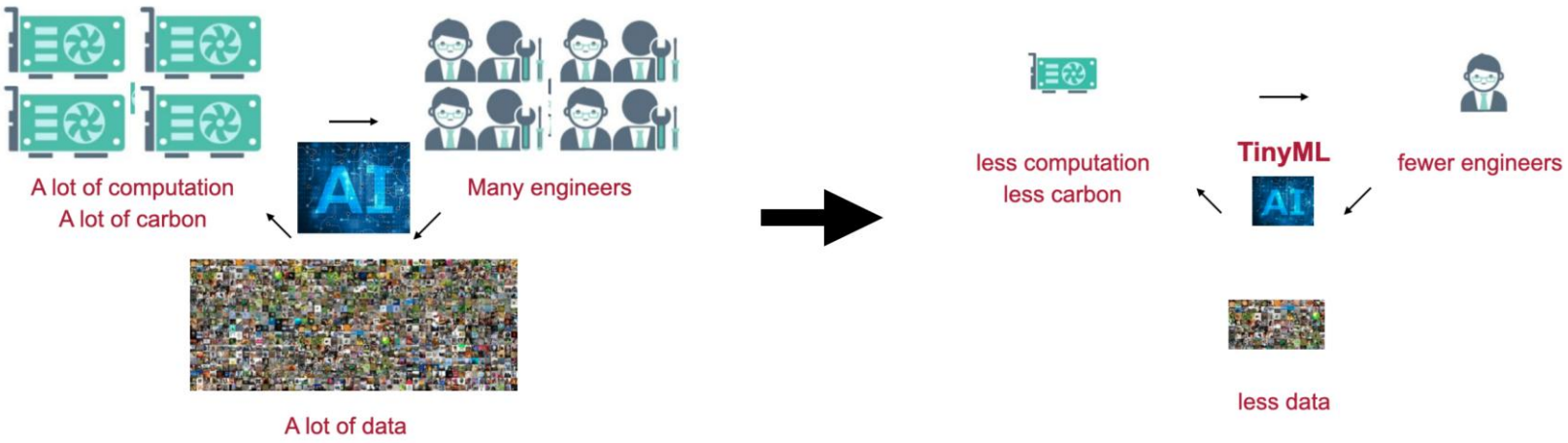
tiny Machine Learning

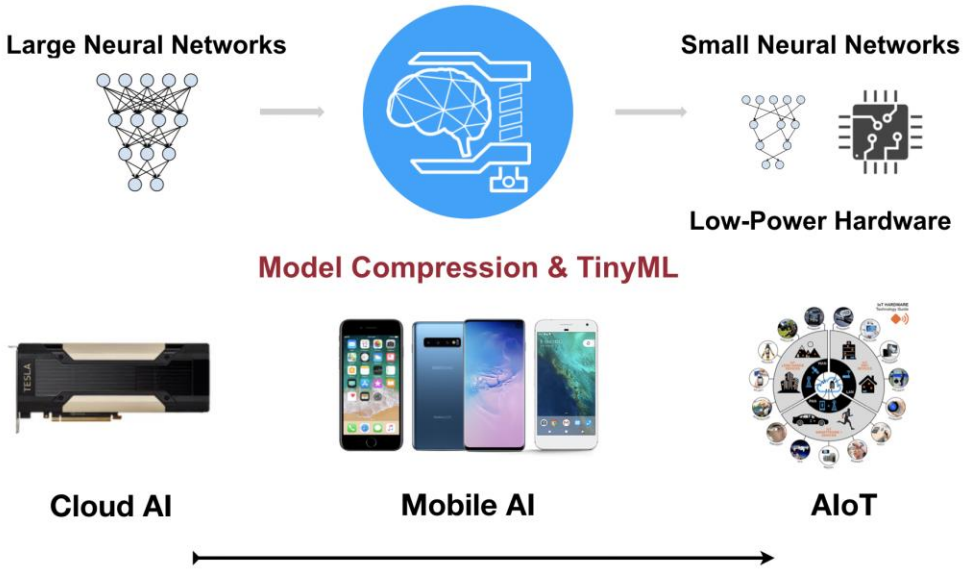
Machine Learning vs Deep Learning

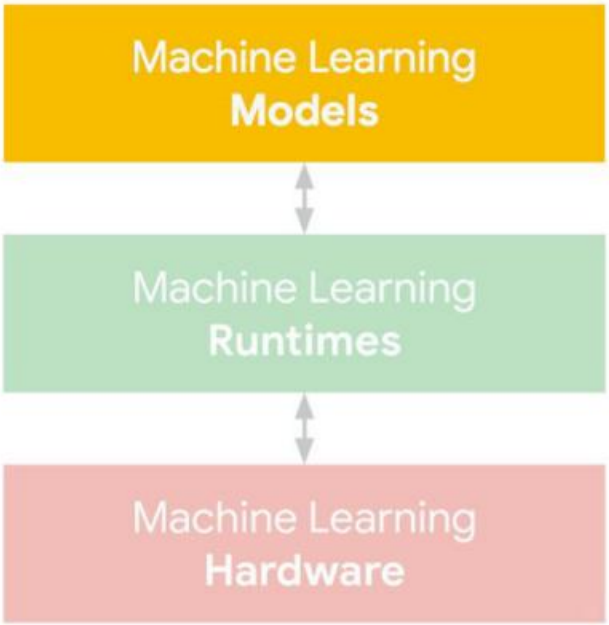
- DecisionTree
- RandomForest
- XGBoost
- GaussianNB
- Support Vector Machines (SVC and OneClassSVM)
- Relevant Vector Machines (from skbayes.rvm_ard_models package)
- SEFR
- PCA



- CNN
- DNN
- RNN
- AutoEncoder
- GAN
- Transformers


A lot of computation
A lot of carbon
AI
Many engineers
A lot of data
less computation
less carbon
TinyML
AI
fewer engineers
less data


Large Neural Networks
Small Neural Networks
Low-Power Hardware
Model Compression & TinyML
Cloud AI
Mobile AI
AIoT



Model Compression Techniques

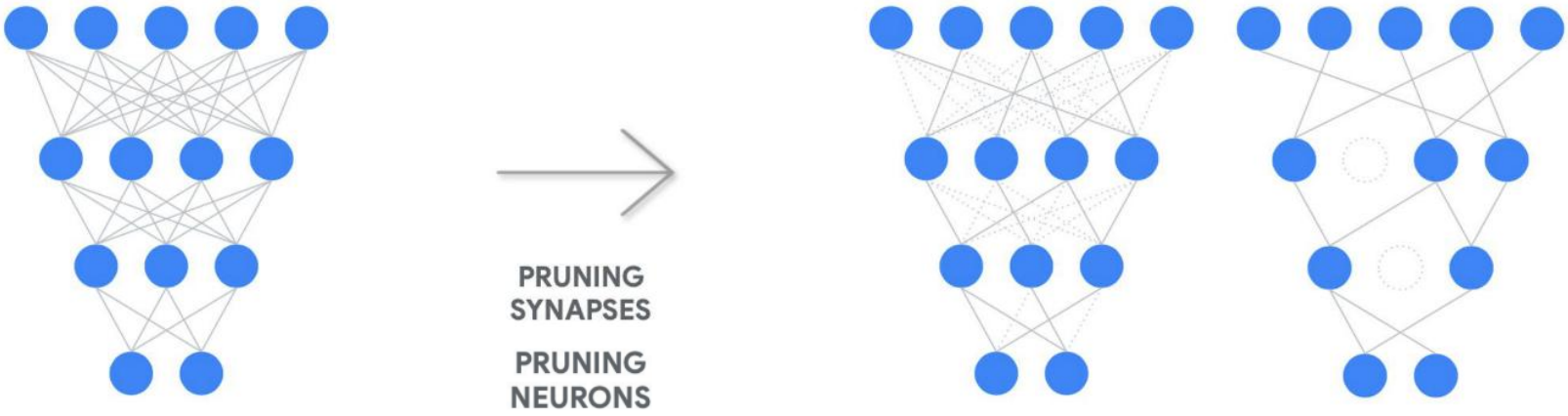
Pruning

Quantization

Knowledge Distillation

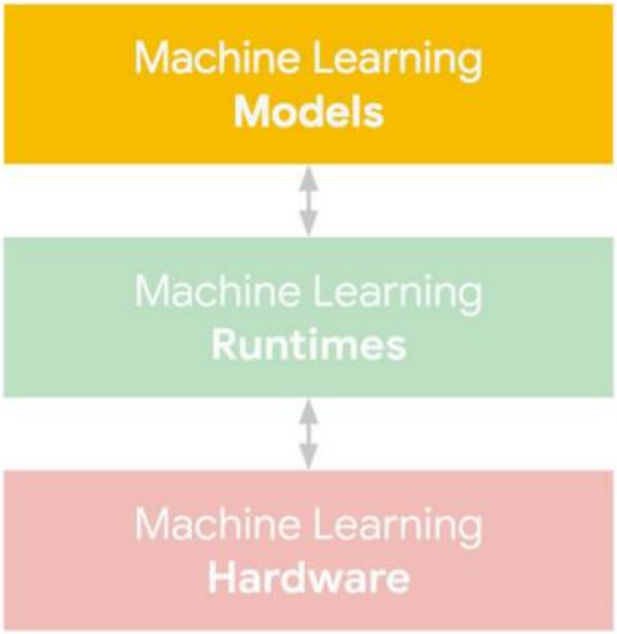
...

Pruning



Weight pruning means eliminating unnecessary values in the weight tensors. Practically setting the neural network parameters' values to zero to remove what we estimate are unnecessary connections between the layers of a neural network.

This is done during the training process to allow the neural network to adapt to the changes.



Model Compression Techniques

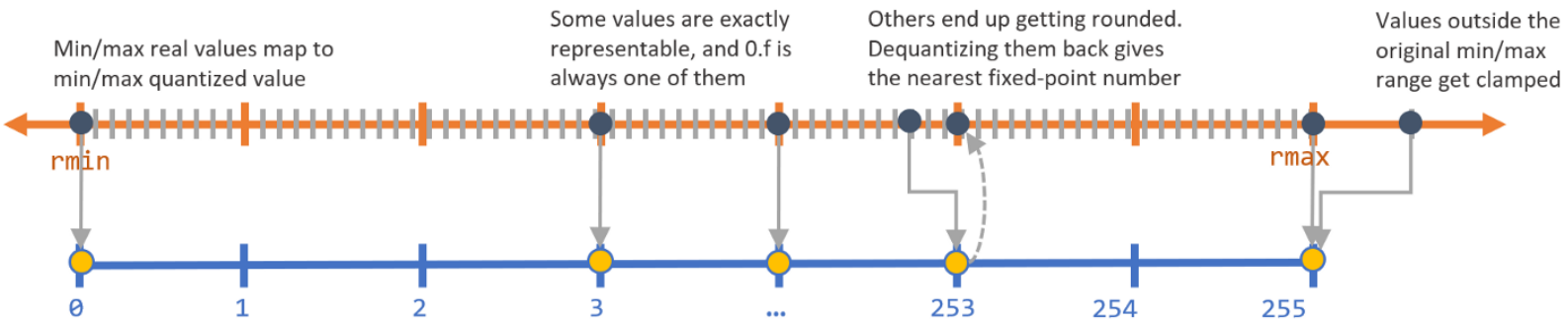
Pruning

Quantization

Knowledge Distillation

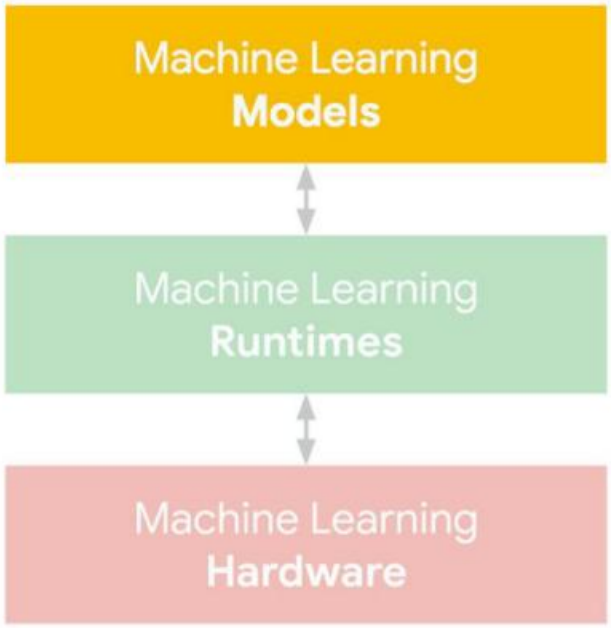
...

Quantization



Quantization is the process of transforming an ML model into an equivalent representation that uses parameters and computations at a lower precision. This improves the model's execution performance and efficiency. For example, TensorFlow Lite 8-bit integer quantization results in models that are up to 4x smaller in size, 1.5x-4x faster in computations, and lower power consumption on CPUs.

However, the process of going from higher to lower precision is lossy in nature. As seen in the image below, quantization squeezes a small range of floating-point values into a fixed number of information buckets.



Model Compression Techniques

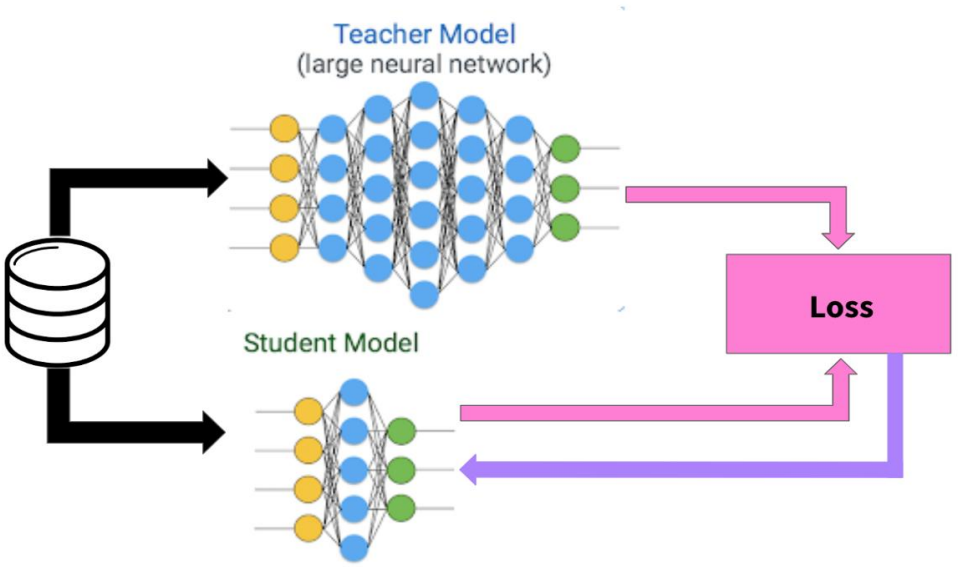
Pruning

Quantization

Knowledge Distillation

...

Knowledge distillation

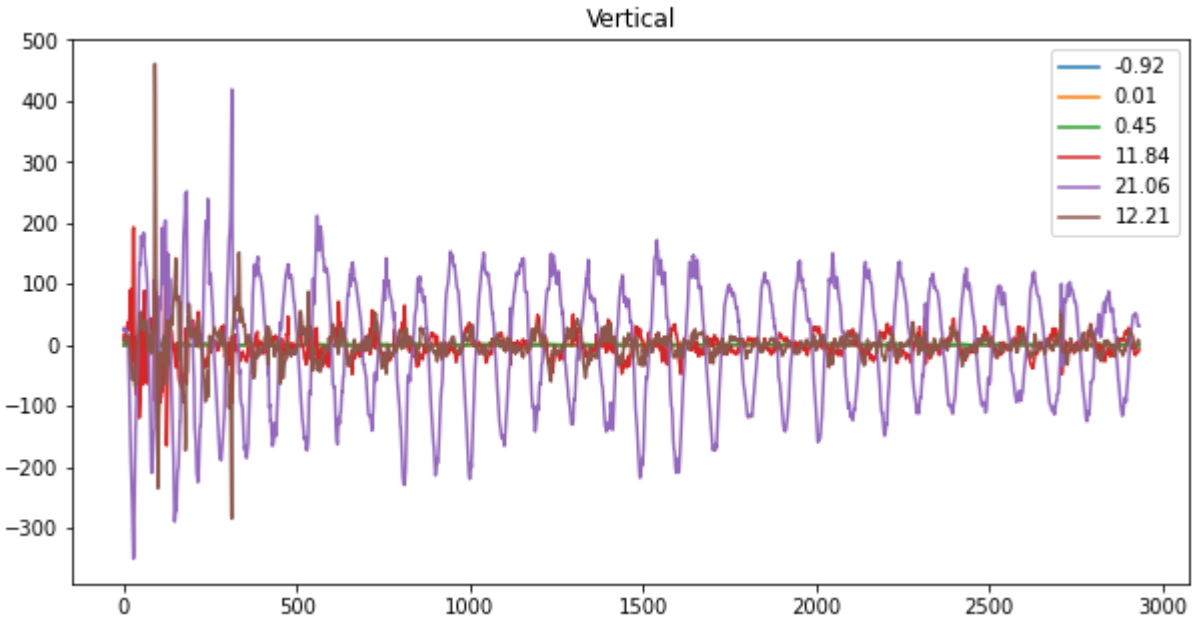


Knowledge Distillation is a procedure for model compression, in which a small (student) model is trained to match a large pre-trained (teacher) model. Knowledge is transferred from the teacher model to the student by minimizing a loss function, aimed at matching softened teacher logits as well as ground-truth labels.

Learn by doing…

Machine Learning version vs Deep Learning Model

Gesture Recognition: the Machine Learning way

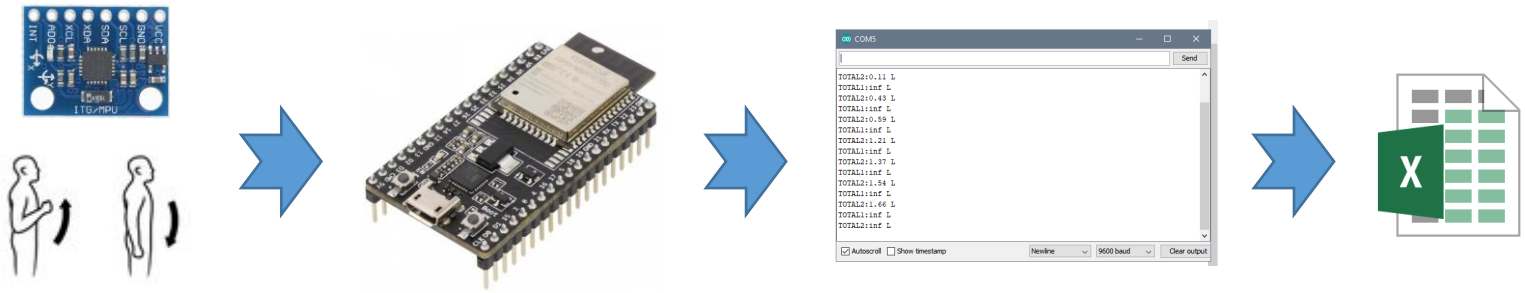


DEMO

pip install everywhereml

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from everywhereml.sklearn.ensemble import RandomForestClassifier


def load_data_from_csv(filename: str, label_column: str) -> tuple:
    """
    Convert csv file to X and y
    :param label_column:
    :param filename:
    :return:
    """
    df = pd.read_csv(filename)
    x_columns = [c for c in df.columns if c != label_column]
    X = df[x_columns].to_numpy(dtype=float)
    y_string = df[label_column]
    label_encoder = LabelEncoder().fit(y_string)
    y_numeric = label_encoder.transform(y_string)
    print('Label mapping', {label: i for i, label in enumerate(label_encoder.classes_)})

    return X, y_numeric


X, y = load_data_from_csv('iris.csv', label_column='variety')
clf = RandomForestClassifier(n_estimators=5).fit(X, y)

import sys
sys.stdout = open('IrisClassifier.h','wt')
print(clf.to_arduino(instance_name='irisClassifier'))
```

Exercise: Capture and train data



Step 1.1 Record the "no gesture" gesture

Record.ino

Arm at rest position

Move slowly to add some noise

min. of 20 secs

save data from terminal to file “rest.csv”

Step 1.2 Record all the other gestures

4 different movements (Left, Right, Front, Up)

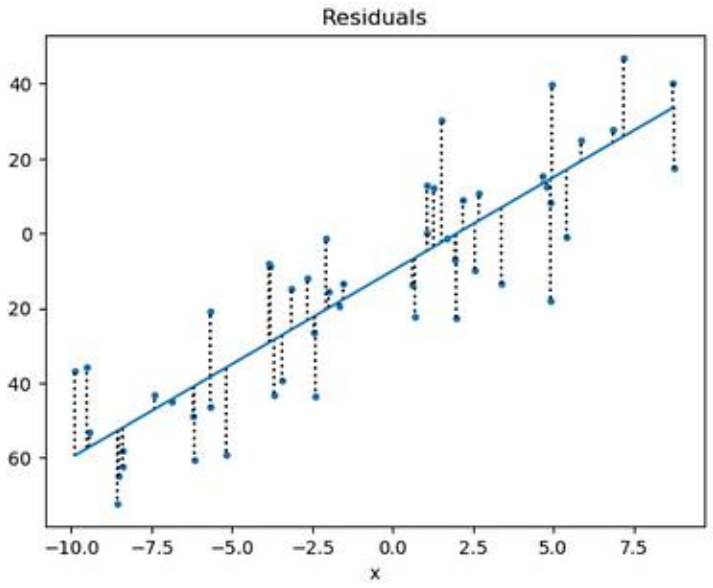
min. of 20 secs for each motion/repetition

save data from terminal to file “xxx.csv”

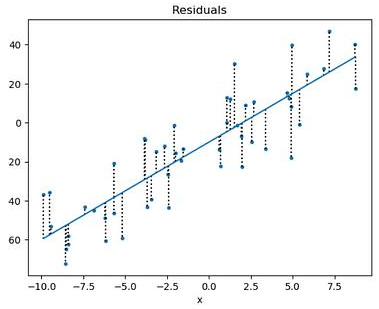
tiny Machine Learning

Linear Regression

…is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. It is one of the simplest and most used techniques in statistical modelling and machine learning for predictive analysis.



tiny Machine Learning



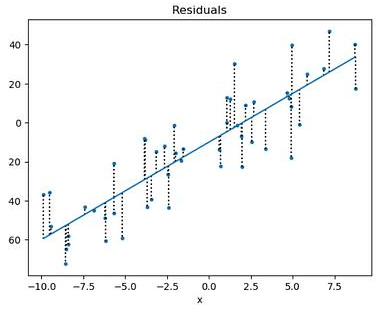
Linear Regression

…relies on several key assumptions to provide reliable and valid results.

A. **Linearity:** The relationship between dependent and independent variables should follow a straight-line pattern, meaning changes in the independent variables proportionally affect the dependent variable.

B. **Independence of Errors:** Residuals should not be correlated with each other; each observation's error should be independent. Autocorrelation may indicate missing variables or model issues.

tiny Machine Learning



Linear Regression

C. **Homoscedasticity:** The variance of errors should remain constant across all values of the independent variables. Inconsistent variance (heteroscedasticity) can lead to biased estimates.

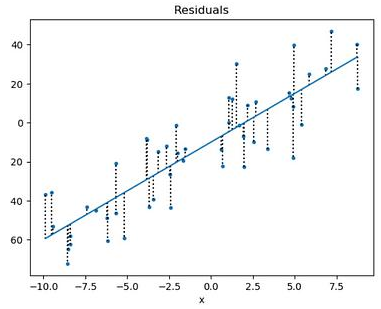
D. **Normality of Errors:** Residuals should follow a normal distribution to ensure accurate statistical inference, hypothesis testing, and confidence intervals.

E. **No Perfect Multicollinearity:** Independent variables should not be perfectly correlated, as this makes coefficient estimation unstable and unreliable.

F. **Zero Conditional Mean:** The average error should be zero across all values of independent variables, ensuring unbiased predictions without systematic over- or underestimation.

tiny Machine Learning

Linear Regression - Model

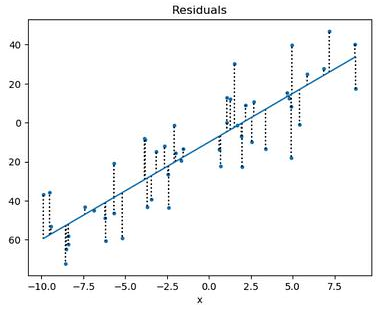


$$Y = \beta_0 + \beta_1 X + \varepsilon$$

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon$$

- *Y* is the dependent variable.
- *X* is the independent variable.
- *β*0 is the intercept (the value of *Y* when *X*=0).
- *β*1 is the slope (the change in *Y* for a one-unit change in *X*).
- *ε* represents the error term, which captures the difference between the observed and predicted values.

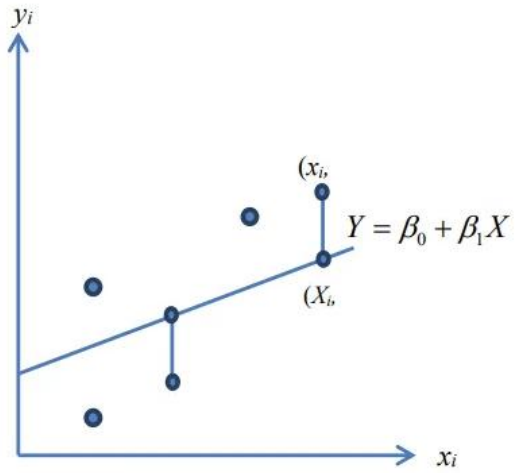
tiny Machine Learning



Linear Regression - Coefficients

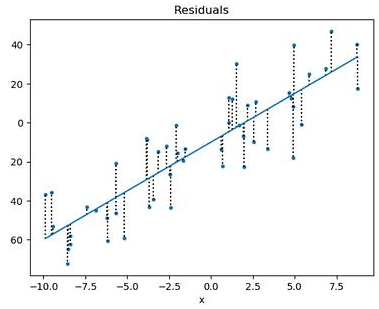
The coefficients *β*0,*β*1,…,*βn* are estimated using the method of least squares. This method minimizes the sum of the squared differences between the observed values of the dependent variable and the values predicted by the linear model.

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



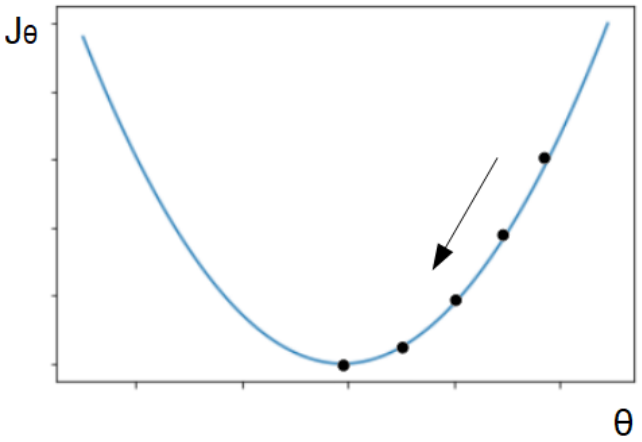
- *yi* is the observed value of the dependent variable for the *i*th observation.
- *y^i* is the predicted value of the dependent variable for the *i*th observation based on the linear regression model.
- *n* is the number of observations.

tiny Machine Learning



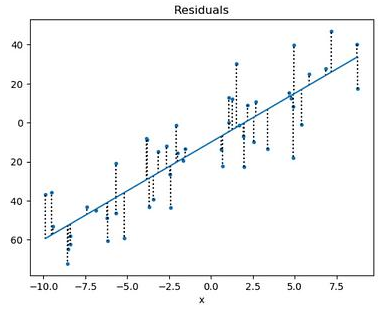
Linear Regression –
Minimizing the Objective Function

Gradient descent is an iterative optimization algorithm used to minimize the MSE by adjusting the coefficients in the direction of the negative gradient of the MSE with respect to each coefficient.



$$\beta_j^{(t+1)} = \beta_j^{(t)} - \alpha \frac{\partial}{\partial \beta_j} \text{MSE}$$

tiny Machine Learning

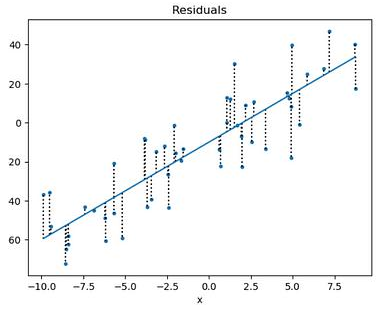


Linear Regression -

In practice (step by step)

Importing libraries
Load Dataset
Exploratory Data Analysis
Splitting the data
Feature selection
Training the model
Evaluating the model
Export Model
Run in Arduino

tiny Machine Learning

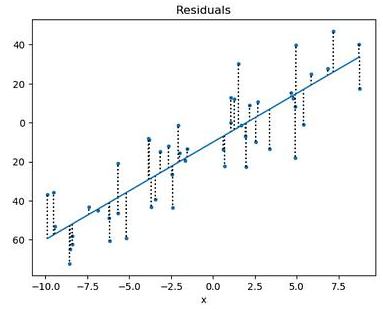


Linear Regression -

In practice (step by step)

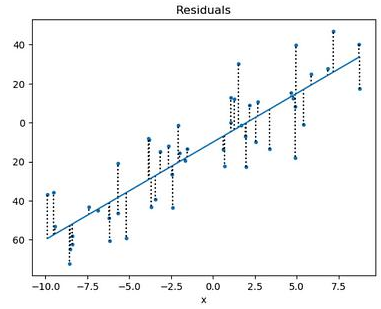
```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.calibration import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from micromlgen import port
import pylab
import scipy.stats as stats
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

tiny Machine Learning




```
# Load dataset
data = load_diabetes() # load data

# Create a DataFrame
df_diabetes = pd.DataFrame(data.data, columns=data.feature_names)


# Add target variable to the DataFrame
df_diabetes['target'] = data.target

# Remove NaN values
df = df_diabetes.dropna(axis='rows') #remove NaN

# Display the DataFrame
df_diabetes.head()
df_diabetes.info()
df_diabetes.describe()
```

tiny Machine Learning

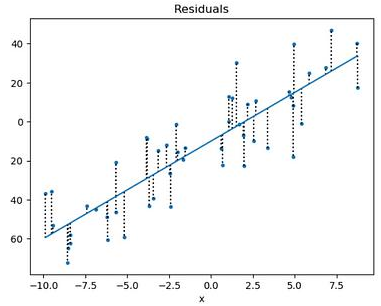


	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
count	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	4.420000e+02	442.000000
mean	-2.511817e-19	1.230790e-17	-2.245564e-16	-4.797570e-17	-1.381499e-17	3.918434e-17	-5.777179e-18	-9.042540e-18	9.293722e-17	1.130318e-17	152.133484
std	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	4.761905e-02	77.093005
min	-1.072256e-01	-4.464164e-02	-9.027530e-02	-1.123988e-01	-1.267807e-01	-1.156131e-01	-1.023071e-01	-7.639450e-02	-1.260971e-01	-1.377672e-01	25.000000
25%	-3.729927e-02	-4.464164e-02	-3.422907e-02	-3.665608e-02	-3.424784e-02	-3.035840e-02	-3.511716e-02	-3.949338e-02	-3.324559e-02	-3.317903e-02	87.000000
50%	5.383060e-03	-4.464164e-02	-7.283766e-03	-5.670422e-03	-4.320866e-03	-3.819065e-03	-6.584468e-03	-2.592262e-03	-1.947171e-03	-1.077698e-03	140.500000
75%	3.807591e-02	5.068012e-02	3.124802e-02	3.564379e-02	2.835801e-02	2.984439e-02	2.931150e-02	3.430886e-02	3.243232e-02	2.791705e-02	211.500000
max	1.107267e-01	5.068012e-02	1.705552e-01	1.320436e-01	1.539137e-01	1.987880e-01	1.811791e-01	1.852344e-01	1.335973e-01	1.356118e-01	346.000000

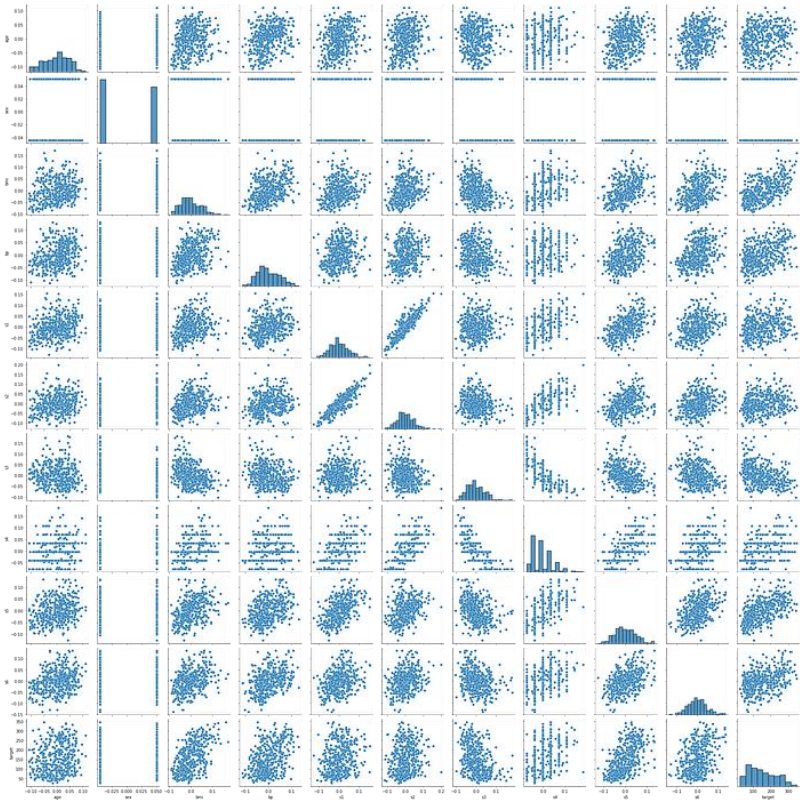
	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
```

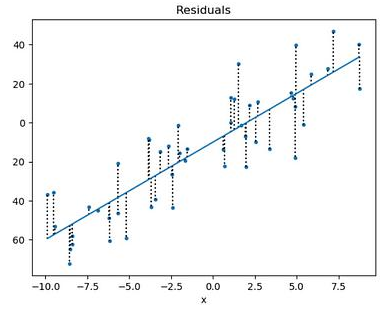
tiny Machine Learning




```
# Seaborn analysis
sns.pairplot(df_diabetes)
```

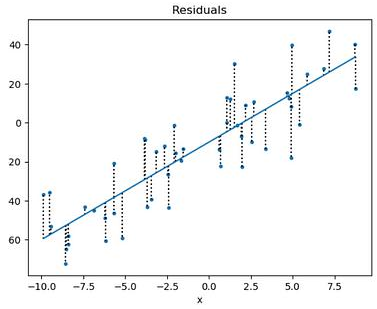


tiny Machine Learning



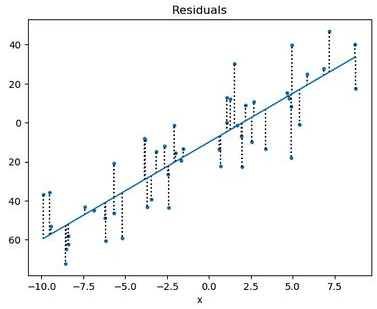

```
df = df_diabetes.iloc[:100,0:10]

X=df.to_numpy()

# Converting string value to int type for labels: Setosa = 0, Versicolor = 1
y=df_diabetes.iloc[:100,-1]
y = LabelEncoder().fit_transform(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

tiny Machine Learning




```
df = df_diabetes.iloc[:100,0:10]

X=df.to_numpy()

# Converting string value to int type for labels: Setosa = 0, Versicolor = 1
y=df_diabetes.iloc[:100,-1]
y = LabelEncoder().fit_transform(y)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

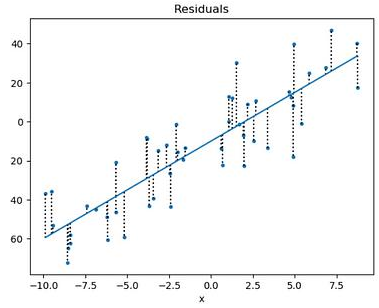
tiny Machine Learning

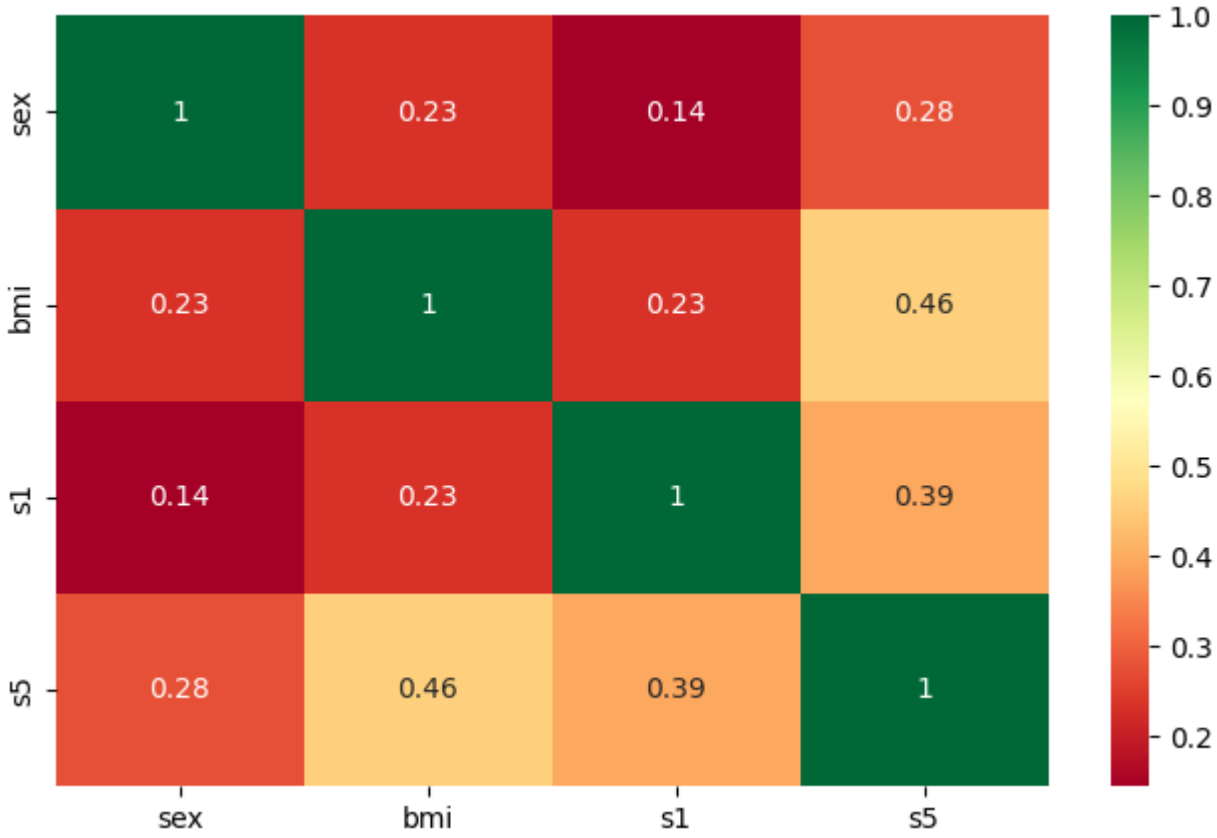


Recursive Feature Elimination (RFE)

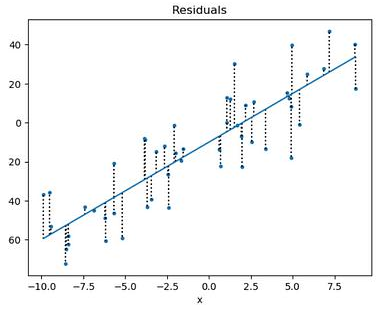
```
# Build a logreg and compute the feature importances
linearRegressor = LinearRegression()

# create the RFE model and select 5 attributes
rfe = RFE(estimator=linearRegressor, n_features_to_select=5, step=30)
rfe = rfe.fit(X_train, y_train)

# summarize the selection of the attributes
print('Selected features: %s' % list(df.columns[rfe.support_]))

selected_features = ['sex', 'bmi', 's1', 's5']
df_selected = df[selected_features]

plt.subplots(figsize=(8, 5))
sns.heatmap(df_selected.corr(), annot=True, cmap="RdYlGn")
plt.show()
```

tiny Machine Learning





tiny Machine Learning

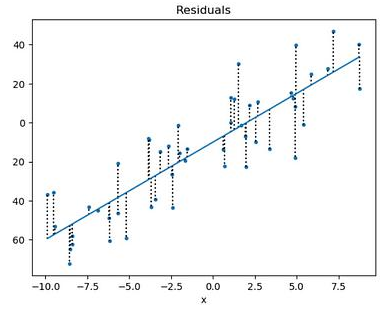


Ordinary Least Squares (OLS)

… regression analysis is a statistical method used to estimate the relationship between one or more independent variables (predictors) and a dependent variable (response).

```
# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train).fit()

# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()

# Print the parameters, i.e. the intercept and the slope of the regression line fitted
parameters = lr.params
print('Model fit parameter: ', parameters)

# Performing a summary operation lists out all the different parameters
print(lr.summary())
```

tiny Machine Learning



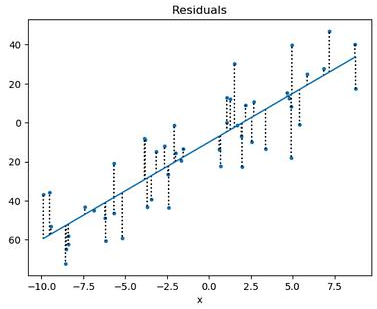
Ordinary Least Squares (OLS)

… regression analysis is a statistical method used to estimate the relationship between one or more independent variables (predictors) and a dependent variable (response).

```
predict =   parameters[0] +
            parameters[0]*X_train[:,0]+
            parameters[1]*X_train[:,1]+
            parameters[2]*X_train[:,2]+
            parameters[3]*X_train[:,3]+
            parameters[4]*X_train[:,4]+
            parameters[5]*X_train[:,5]+
            parameters[6]*X_train[:,6]+
            parameters[7]*X_train[:,7]+
            parameters[8]*X_train[:,8]+
            parameters[9]*X_train[:,9]
```

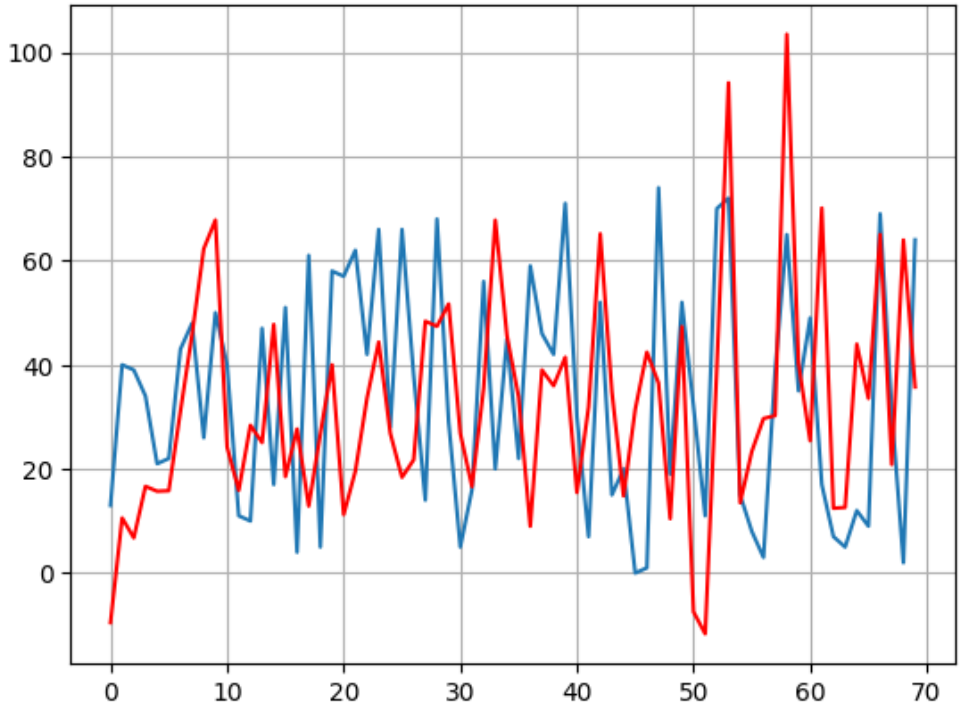
```
plt.plot(y_train)
plt.plot(predict, 'r')
plt.grid()
plt.show()
```

tiny Machine Learning

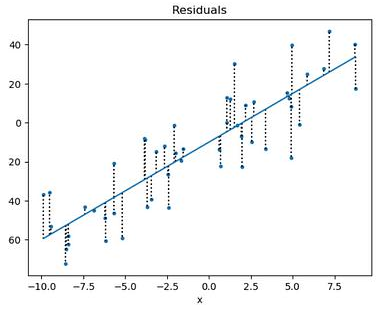


Ordinary Least Squares (OLS)

… regression analysis is a statistical method used to estimate the relationship between one or more independent variables (predictors) and a dependent variable (response).



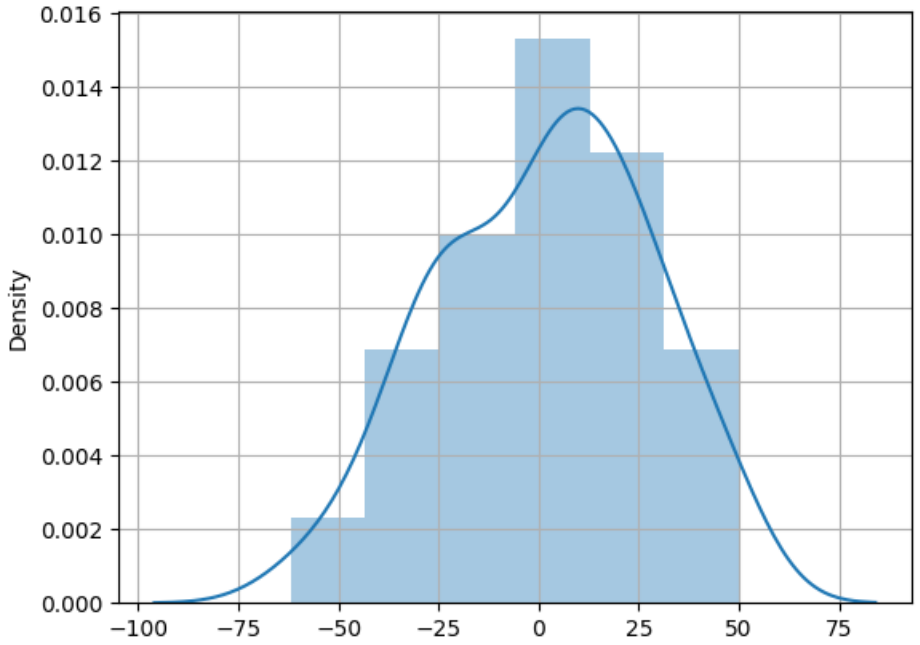
tiny Machine Learning



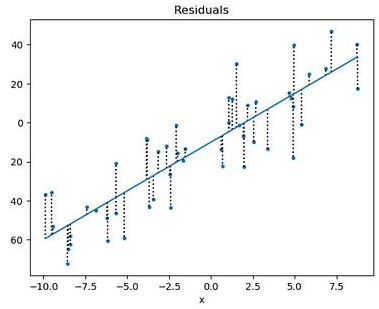
Ordinary Least Squares (OLS)

… regression analysis is a statistical method used to estimate the relationship between one or more independent variables (predictors) and a dependent variable (response).

```
residuals = y_train-predict
sns.distplot(residuals)
plt.grid()
```



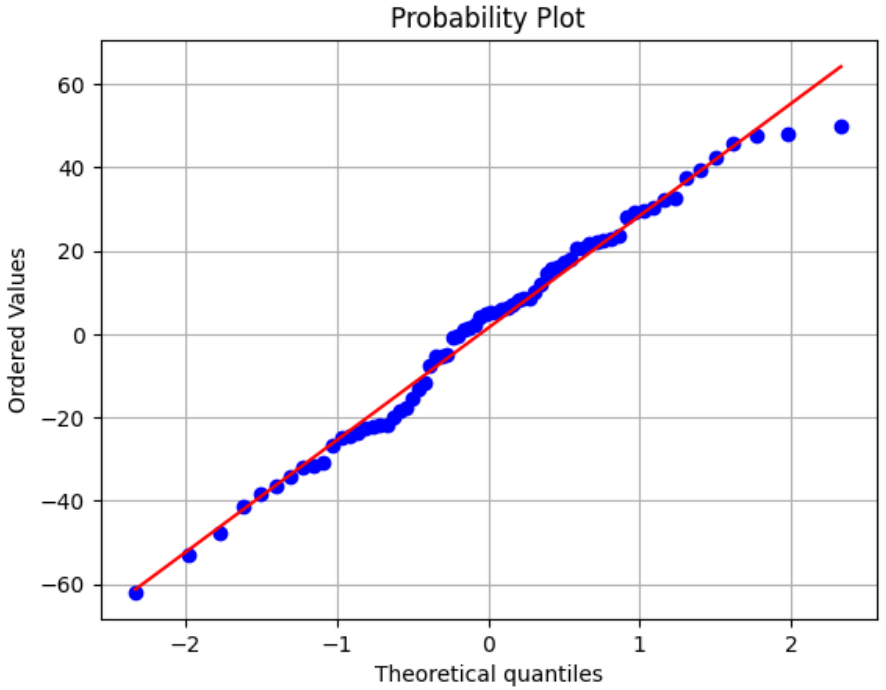
tiny Machine Learning



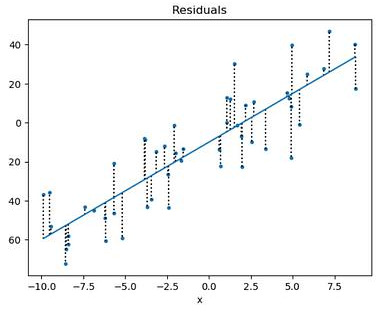
Ordinary Least Squares (OLS)

… regression analysis is a statistical method used to estimate the relationship between one or more independent variables (predictors) and a dependent variable (response).

```
stats.probplot(residuals, dist="norm", plot=pylab)
plt.grid()
pylab.show()
```

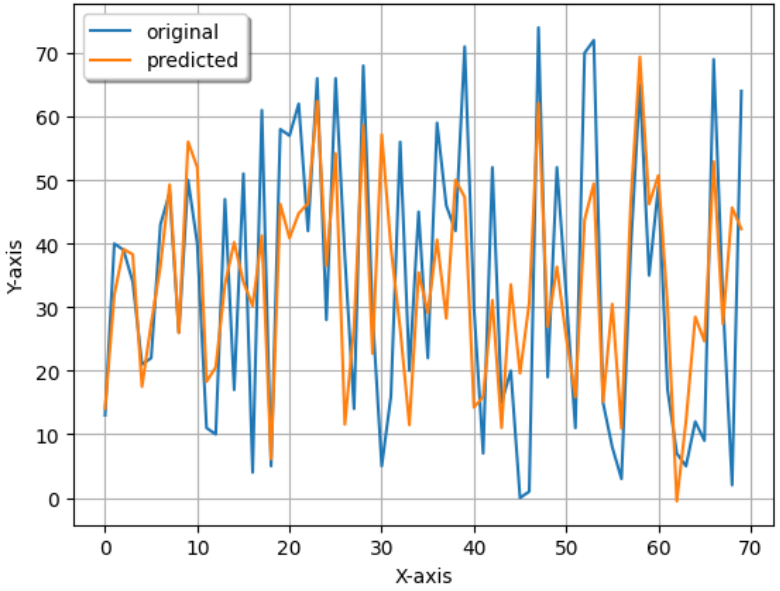


tiny Machine Learning

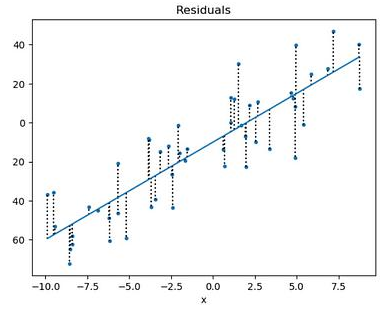


Model Evaluation

```
model = linearRegressor.fit(X_train,y_train)

predict = model.predict(X_train)

x_ax = range(len(y_train))
plt.plot(x_ax, y_train, label="original")
plt.plot(x_ax, predict, label="predicted")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



tiny Machine Learning

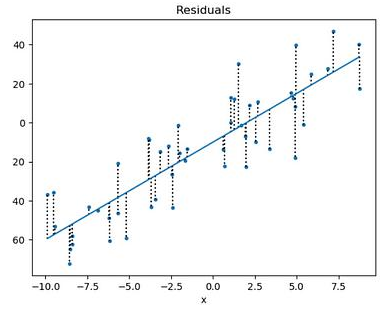


Model Evaluation

```
score = model.score(X_train, y_train)
training_predict = model.predict(X_train)
mse = mean_squared_error(y_train, training_predict)

print("R-squared:", score)
print("MSE: ", mse)
print("RMSE: ", mse*(1/2.0))
```

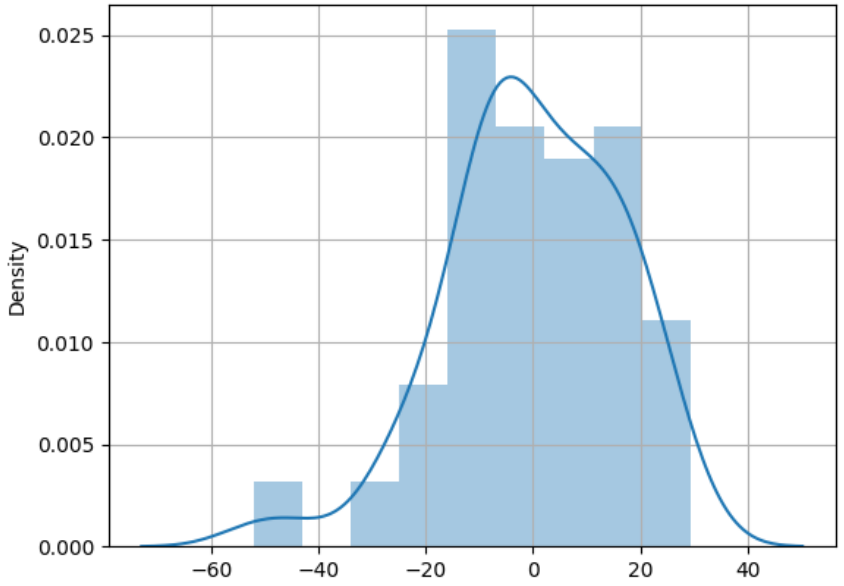
```
R-squared: 0.4665605455873243
MSE:  261.72727823900914
RMSE:  130.86363911950457
```

tiny Machine Learning

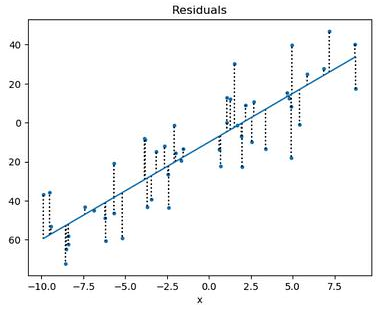


Residuals

```
residuals = y_train-training_predict
sns.distplot(residuals)
plt.grid()
```

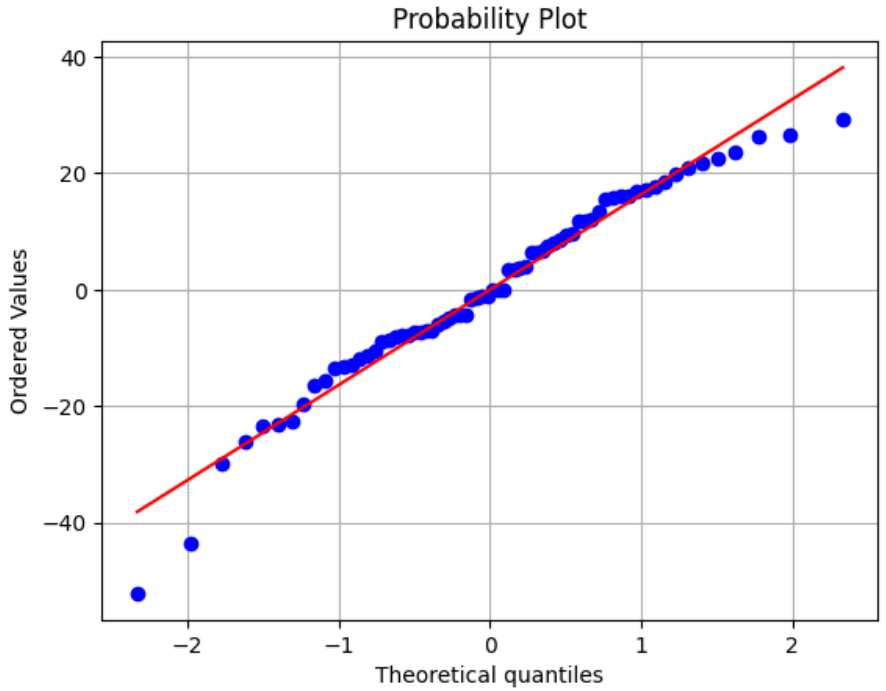


tiny Machine Learning

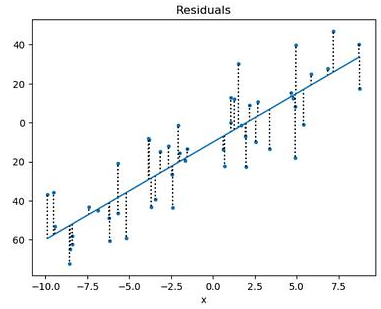


Residuals

```
stats.probplot(residuals, dist="norm", plot=pylab)
plt.grid()
pylab.show()
```

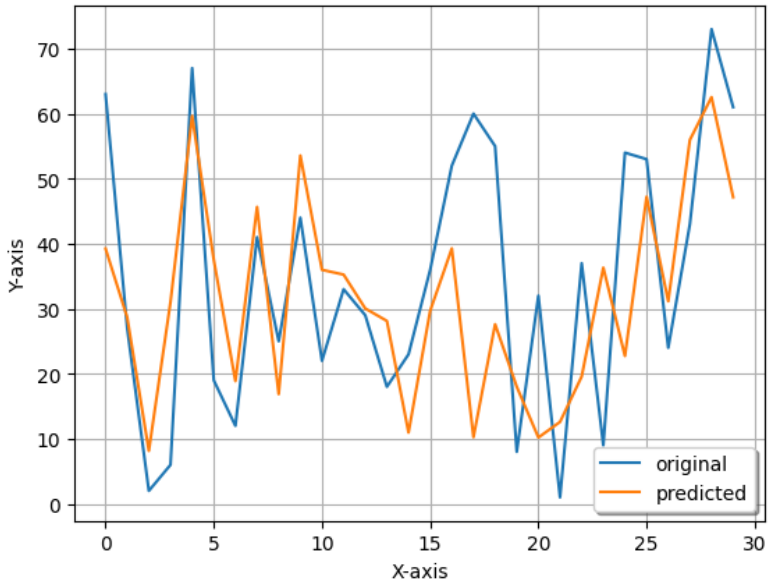


tiny Machine Learning

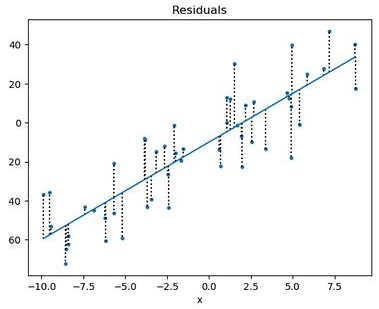


Residuals

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, test_predict, label="predicted")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



tiny Machine Learning

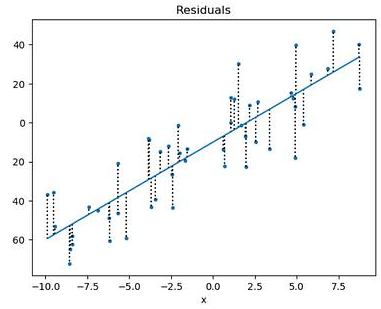


Residuals

```
score = model.score(X_test, y_test)
test_predict = model.predict(X_test)
mse = mean_squared_error(y_test, test_predict)

print("R-squared:", score)
print("MSE: ", mse)
print("RMSE: ", mse*(1/2.0))
```

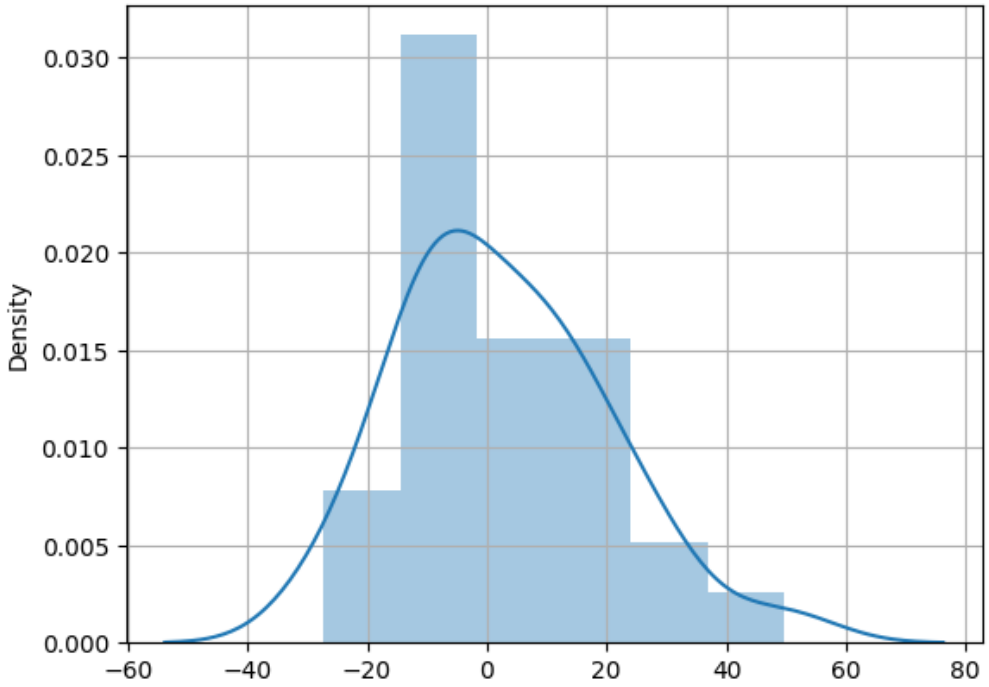
```
R-squared: 0.2670722134668l4
MSE:  300.75203101864935
RMSE:  150.37601550932467
```

tiny Machine Learning

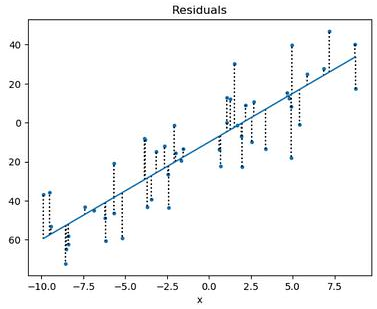


Residuals

```
residuals = y_test-test_predict
sns.distplot(residuals)
plt.grid()
```

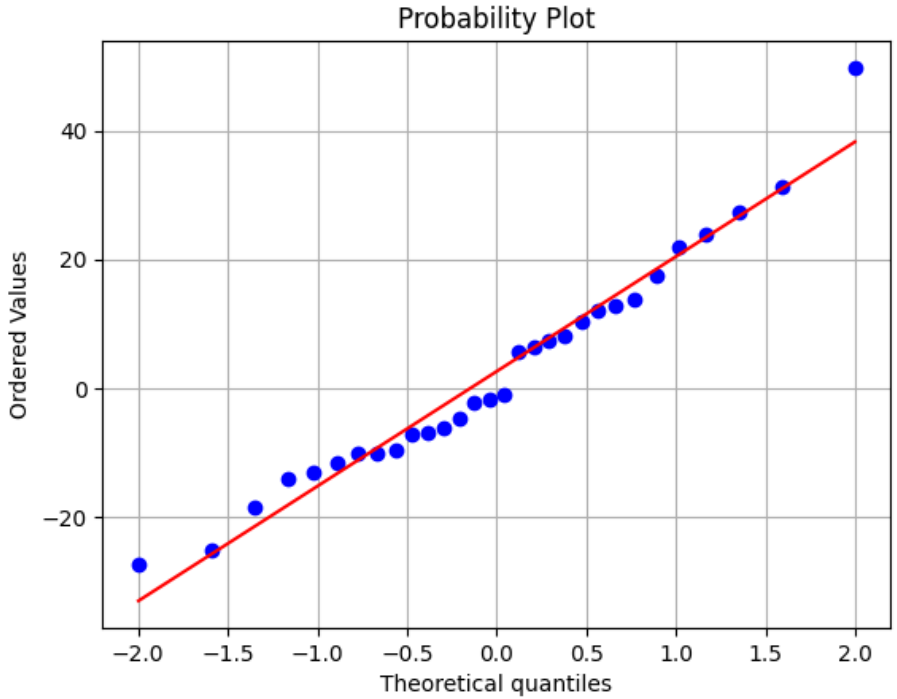


tiny Machine Learning

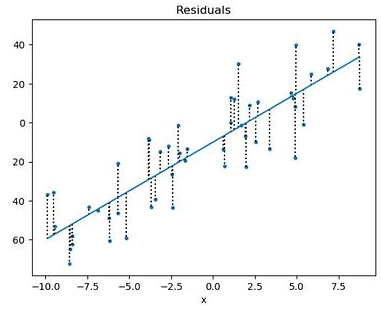


Residuals

```
stats.probplot(residuals, dist="norm", plot=pylab)
plt.grid()
pylab.show()
```



tiny Machine Learning



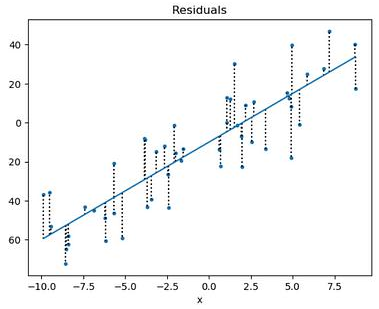
Export & Arduino

```
print(port(model))

with open('.\LinearRegressor.h', 'w') as file:
        file.write(port(model))
```

NOTE: in case of Arduino Compile Errors
In file LinearRegressor.h verify if include is :

#include <cstdarg> => **change to** => #include <stdarg.h>

tiny Machine Learning



Export & Arduino

```
#include "LinearRegressor.h"

Eloquent::ML::Port::LinearRegression LinearRegressor;

void setup()
{
Serial.begin(115200);
}

void loop()
{
float X_1[] = {0.02717829, 0.05068012, 0.01750591, -0.03321323, -0.00707277, 0.04597154,
-0.06549067, 0.07120998, -0.09643495, -0.05906719};
int result_1 = LinearRegressor.predict(X_1);
Serial.print("Result of predict with input X1 (real value = 13):");
Serial.println(result_1);
delay(2000);
}
```

tiny Machine Learning
CNN

tiny Machine Learning


```
pip install "everywhereml>=0.2.32”
pip install tensorflow
```

```
import numpy as np

from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt

from everywhereml.code_generators.tensorflow import convert_model
```

tiny Machine Learning


```
def load_Xy():
  """
  Load data
  It may be a CSV file or whatever
  """
  return load_iris(return_X_y=True)
```

```
def get_Xy():
  """
  Normalize X
  One-hot encode y
  """
  X, y = load_Xy()
  X = np.asarray(X)
  y = np.asarray(y)
  num_classes = y.max()
  eye = np.eye(num_classes + 1)
  X_norm = MinMaxScaler().fit_transform(X)
  y_hot = np.asarray([eye[yi] for yi in y], dtype=int)

  return X_norm, y_hot
```

tiny Machine Learning


```
"""
Instantiate NN for classification
"""
def instantiate_nn_for_classification(input_shape, num_classes):
  model = tf.keras.Sequential()
  model.add(layers.Dense(32, activation='relu', input_shape=input_shape))
  model.add(layers.Dense(16, activation='relu'))
  model.add(layers.Dense(num_classes, activation='softmax'))
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

  return model
```

```
"""
Split data between train, validation and test
"""
from sklearn.model_selection import train_test_split

X, y = get_Xy()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)
X_train, X_val, y_train, y_val = train_test_split(X_train,
y_train, test_size=0.3)
```

tiny Machine Learning


```
"""
Instantiate NN for classification
"""
def instantiate_nn_for_classification(input_shape, num_classes):
  model = tf.keras.Sequential()
  model.add(layers.Dense(32, activation='relu', input_shape=input_shape))
  model.add(layers.Dense(16, activation='relu'))
  model.add(layers.Dense(num_classes, activation='softmax'))
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

  return model
```

```
"""
Split data between train, validation and test
"""
from sklearn.model_selection import train_test_split

X, y = get_Xy()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)
X_train, X_val, y_train, y_val = train_test_split(X_train,
y_train, test_size=0.3)
```

tiny Machine Learning


```
"""
Train model
"""
input_shape = X.shape[1:]
num_classes = y.shape[1]

model = instantiate_nn_for_classification(input_shape, num_classes)
model.summary()

history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_val, y_val))

print(history.history.keys())
```

```
# plot history accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

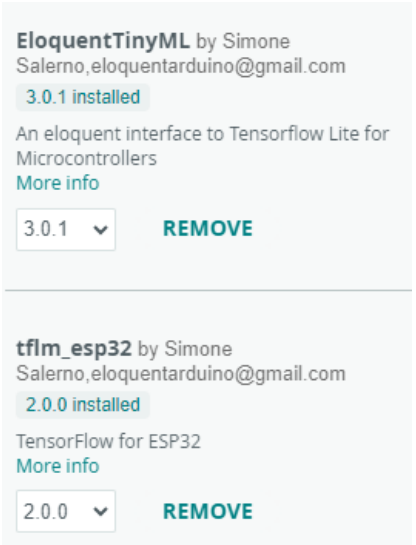
tiny Machine Learning


```
"""
Export NN to C++
Copy-paste the generated code inside a file named model.h or irisModel.h
in your Arduino project
"""
from everywhereml.code_generators.tensorflow import convert_model

out = convert_model(model, X, y, model_name='irisModel')
print(out)

with open('.\cnn.h', 'w') as file:
    file.write(out)
```

tiny Machine Learning

ARDUINO

install libs =>




```
#include "cnn.h"
#include <tflm_esp32.h>
#include <eloquent_tinyml.h>

#define ARENA_SIZE 2000 // this is trial-and-error process

Eloquent::TF::Sequential<TF_NUM_OPS, ARENA_SIZE> tf;

...
```

tiny Machine Learning

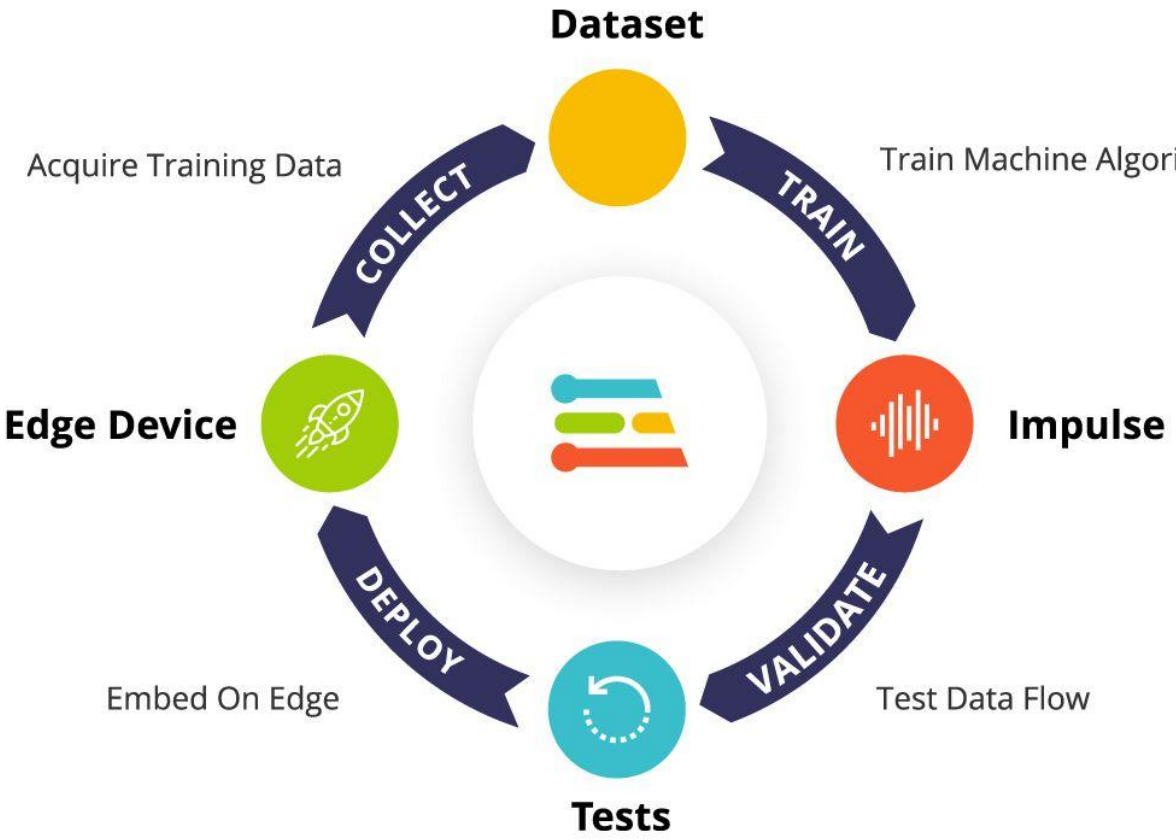

```
void setup() {
    Serial.begin(115200);
    delay(3000);
    Serial.println("__TENSORFLOW IRIS__");

    // configure input/output
    tf.setNumInputs(4);
    tf.setNumOutputs(3);

    // add required ops
    tf.resolver.AddFullyConnected();
    tf.resolver.AddSoftmax();

    while (!tf.begin(irisModel).isOk())
        Serial.println(tf.exception.toString());
}
```

tiny Machine Learning


```
void loop() {
    // classify class 0
    if (!tf.predict(x0).isOk()) {
        Serial.println(tf.exception.toString());
        return;
    }

    Serial.print("expected class 0, predicted class ");
    Serial.println(tf.classification);

    // how long does it take to run a single prediction?
    Serial.print(tf.benchmark.microseconds());
    Serial.println("us");

    delay(1000);
}
```

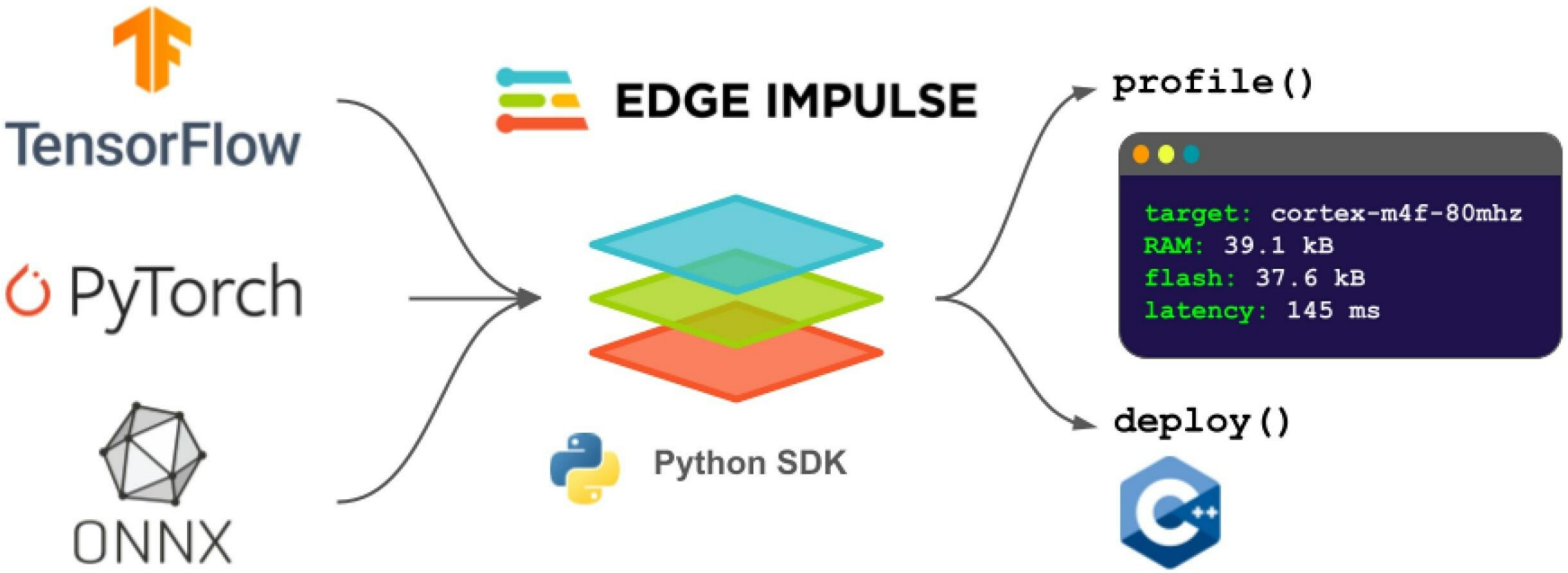
tiny Machine Learning

EDGE IMPULSE



tiny Machine Learning

EDGE IMPULSE



Todo: Register and try online…

tiny Machine Learning
LSTM

tiny Machine Learning

pip install embedded_window
pip install eloquent_tensorflow

```
import numpy as np
import pandas as pd

from embedded_window import Window

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import layers
```

tiny Machine Learning


```
window_size = 150
window_shift = 20
window = Window(length=window_size,
shift=window_shift)
```

```
def read_file(filename, plot: bool = True):
  df = pd.read_csv(filename)
  acc = df[['ax(mg)', 'ay(mg)', 'az(mg)']]

  if plot:
    acc.plot(title=filename)

  # normalize data
  # in this case, accel is in the range -3, 3
  accel = np.clip(acc.to_numpy() / 3, -1, 1)
  y = np.zeros(len(accel))

  # apply window
  # returns a (N, window_length, accel.columns) array
  accel_windows, _, _ = window.fit_transform(accel, y)
  return accel_windows
```

tiny Machine Learning

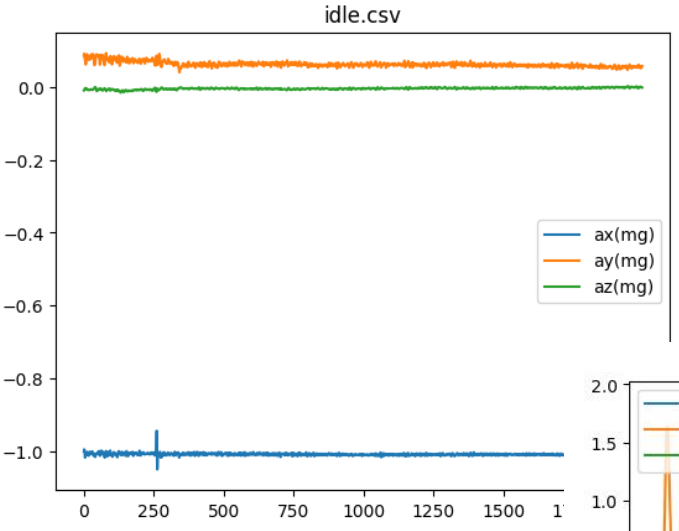

```
def get_Xy():
  """
  Read files
  One-hot encode y
  """
  x_idle = read_file('idle.csv')
  x_horizontal = read_file('left-right.csv')
  x_vertical = read_file('up-down.csv')

  y_idle = [0] * len(x_idle)
  y_horizontal = [1] * len(x_horizontal)
  y_vertical = [2] * len(x_vertical)

  X = np.vstack((x_idle, x_horizontal, x_vertical))
  y = np.concatenate((y_idle, y_horizontal, y_vertical))
  num_classes = y.max()
  eye = np.eye(num_classes + 1)
  y_hot = np.asarray([eye[yi] for yi in y], dtype=int)

  return X, y_hot
```

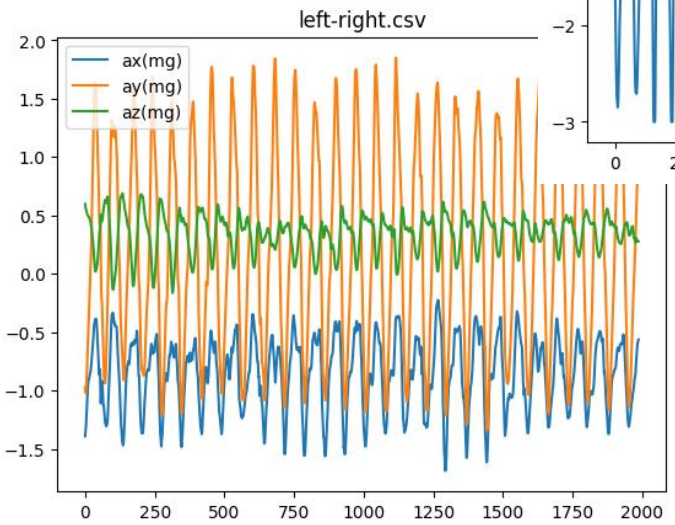
tiny Machine Learning

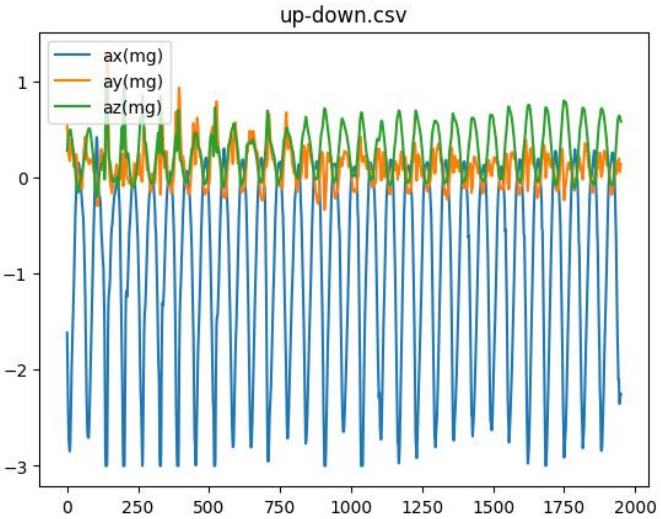

```
X, y = get_Xy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.3)
num_classes = y.shape[1]
print("X.shape", X.shape)
```

```
"""
Print samples of each class
"""
def print_sample(variable_name: str, filename: str):
  sample = read_file(filename, plot=False)[0].flatten()
  values = ', '.join('%.4f' % x for x in sample)
  print(f'float {variable_name}[{len(sample)}] = {{ {values} }};')


print_sample('idle', 'idle.csv')
print_sample('horizontal', 'left-right.csv')
print_sample('vertical', 'up-down.csv')
```

tiny Machine Learning







tiny Machine Learning


```
model = instantiate_lstm_for_classification(X.shape[1:], num_classes)
print(model.summary())
```

```
"""
Train model
"""
history = model.fit(X_train, y_train, epochs=30, validation_data=(X_val, y_val))

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

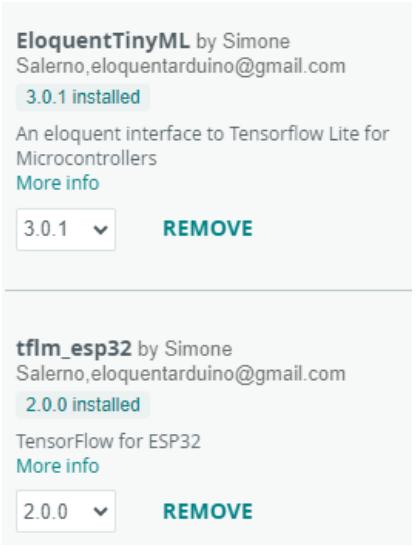
tiny Machine Learning


```
"""
Evaluate on test data
"""
model.evaluate(X_test, y_test)

"""
Save model
"""
from eloquent_tensorflow import convert_model
out = convert_model(model)
print(out)

with open('.\lstm.h', 'w') as file:
    file.write(out)
```

tiny Machine Learning

ARDUINO

install libs =>




```
#include <Arduino.h>
#include "lstm.h"
#include <tflm_esp32.h>
#include <eloquent_tinyml.h>
```

tiny Machine Learning


```
// this is trial-and-error process
#define ARENA_SIZE 30000

Eloquent::TF::Sequential<TF_NUM_OPS, ARENA_SIZE> tf;


float idle[450] = { -0.3334, 0.0289, ...
float horizontal[450] = { -0.4638, ...
float vertical[450] = { -0.5387, ...

void predictSample(const char *classLabel, float *input, uint8_t expectedOutput)
{
    // classify class 0
    if (!tf.predict(input).isOk()) {
        Serial.println(tf.exception.toString());
        return;
    }

    Serial.print("Predicting sample of ");
    Serial.print(classLabel);
    Serial.print(": expcted id=");
    Serial.print(expectedOutput);
    Serial.print(", predicted=");
    Serial.println(tf.classification);
}
```

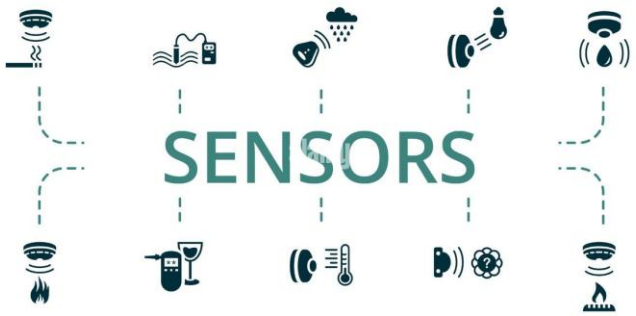
tiny Machine Learning


```
void setup() {
    Serial.begin(115200);
    delay(3000);
    Serial.println("__LSTM__");

    // configure input/output
    tf.setNumInputs(TF_NUM_INPUTS);
    tf.setNumOutputs(TF_NUM_OUTPUTS);

    registerNetworkOps(tf);

    while (!tf.begin(tfModel).isOk()) {
        Serial.println(tf.exception.toString());
        delay(1000);
    }
}
```

tiny Machine Learning


```
void loop() {
    predictSample("idle", idle, 0);
    predictSample("horizontal", horizontal, 1);
    predictSample("vertical", vertical, 2);

    // how long does it take to run a single prediction?
    Serial.print(tf.benchmark.microseconds());
    Serial.println("us");

    delay(1000);
}
```

Final Project



ESPRESSIF

TensorFlow Lite

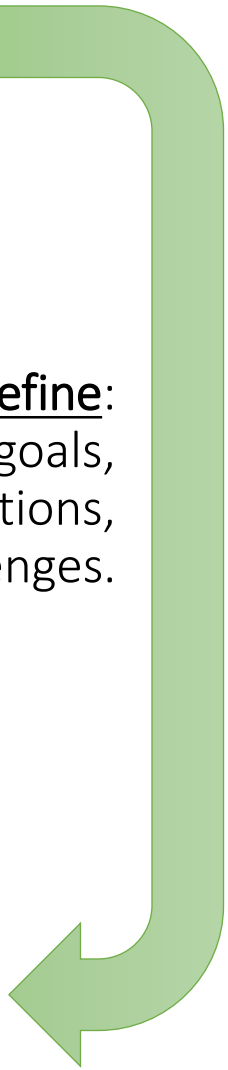
TOPICS

1. Gesture Recognition
2. Person Fall detection
3. Surface recognition
4. Machine Anomaly
5. …
6. (open idea)

Smart Sensor Devices

"a thing to save the day…"

Final Project



Define: goals, limitations, challenges.

1. Define the Problem
Identify the problem to solve (e.g., anomaly detection, voice recognition, gesture detection).
Consider constraints like power consumption, latency, and hardware limitations.

2. Select Hardware & Tools
Choose a microcontroller (e.g., Arduino, Raspberry Pi Pico, ESP32).
Select sensors (e.g., accelerometers, microphones, sensors).
Development tools (e.g., TensorFlow Lite for Microcontrollers, Edge Impulse, Arduino IDE).

3. Data Collection & Preprocessing
Gather relevant data (sensor readings, audio, analog/digital).
Clean and normalize data (remove noise, scale values).
Augment data (if needed) (e.g., data balancing, synthetic data).

4. Model Selection & Training
Choose an appropriate ML model (CNN, RNN, ML models).
Train the model.

5. Deploying the Model
Convert the trained model to C/C++.
Deploy to the microcontroller.
Optimize for real-time performance and memory efficiency.

6. Testing & Validation
Run real-time tests on embedded hardware.
Validate performance metrics (accuracy, inference speed, power consumption).