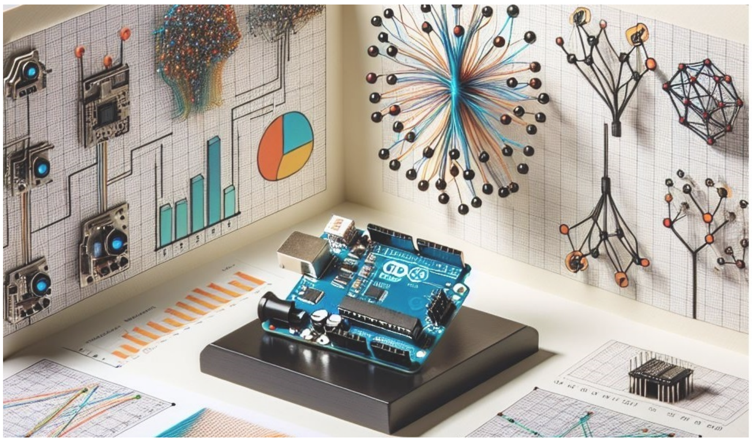
Convert a Python Machine Learning Model to Arduino Code (C++)

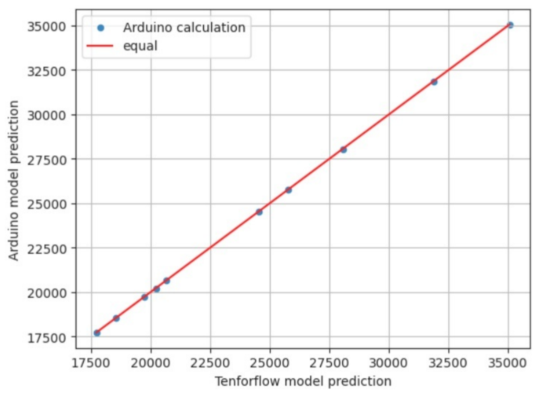
Introduction

Motivation

What ?

This project demonstrates the conversion of Python machine learning (ML) models to Arduino C++ code.
We will use some ML models purely as examples; the goal is not to find the best model or achieve minimal error.
Why ?
In certain applications, such as embedded systems, small microcontrollers with limited memory and computing resources are used. The idea is to train a machine learning model in a Python environment and then convert the trained model to C++ for deployment on a microcontroller.
In this project, we will use the Arduino Uno as an example, but the approach can be applied to other microcontrollers as well.
How ?
Follow the step-by-step guide below, or go directly to the PyPi package mltoarduino





Hardware

In this project, the Arduino Uno was used, but you can use other boards like Arduino Nano or Micro, Miga 2560, ESP32...
below a comparaison of some Arduino boards:

Feature	Arduino Uno	Arduino Nano	Arduino Micro	Arduino Mega 2560	ESP32
Microcontroller	ATmega328P	ATmega328P	ATmega32U4	ATmega2560	Tensilica Xtensa LX6
Operating Voltage	5V	5V	5V	5V	3.3V
Input Voltage	7-12V	7-12V	7-12V	7-12V	5V via USB or 7-12V
Digital I/O Pins	14 (6 PWM)	14 (6 PWM)	20 (7 PWM)	54 (15 PWM)	34
Analog Input Pins	6	8	12	16	18
Flash Memory	32 KB	32 KB	32 KB	256 KB	Up to 16 MB
SRAM	2 KB	2 KB	2.5 KB	8 KB	520 KB
EEPROM	1 KB	1 KB	1 KB	4 KB	None
Clock Speed	16 MHz	16 MHz	16 MHz	16 MHz	240 MHz (dual-core)
Connectivity	UART, I2C, SPI	UART, I2C, SPI	UART, I2C, SPI	UART, I2C, SPI	Wi-Fi, Bluetooth
USB Interface	USB-B	Mini USB	Micro USB	USB-B	Micro USB
Dimensions	68.6 x 53.4 mm	45 x 18 mm	48 x 18 mm	101.52 x 53.3 mm	51 x 25.5 mm
Power Consumption	~50 mA	~50 mA	~50 mA	~70 mA	Varies (~80-240 mA)
Special Features	Simple and robust	Compact	USB HID support	High I/O count	Wi-Fi and BLE
Price Range	Low	Low	Medium	Medium	Medium-High

Table of contents

Libraries

Libraries

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor

import xgboost as xgb

import tensorflow as tf

import matplotlib.pyplot as plt
import seaborn as sns

# save models
import joblib
import pickle

import os
import re
import json
```

Load dataset

All inputs

In [67]:
```
# Load dataset
file = r"https://raw.githubusercontent.com/bouz1/Manipulation_of_second_hand_vehicles_data/refs/heads/main/data:
df=pd.read_csv(file)
```

In [68]:
```
len(df)
```

Out[68]:
```
7250
```

In [69]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7250 entries, 0 to 7249
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   model1             7250 non-null   object 
 1   model2             7250 non-null   object 
 2   version            7250 non-null   object 
 3   price              7250 non-null   float64
 4   km                 7250 non-null   float64
 5   fuel               7250 non-null   float64
 6   CV_fisc            7250 non-null   float64
 7   HorseP             7250 non-null   float64
 8   Gearbox_auto       7250 non-null   float64
 9   L_by_100km         7250 non-null   float64
 10  numbe_seats        7250 non-null   float64
 11  doors_nb           7250 non-null   float64
 12  Euro_stand         7250 non-null   float64
 13  Length             7250 non-null   float64
 14  Nb_option          7250 non-null   float64
 15  registration_date  7250 non-null   float64
dtypes: float64(13), object(3)
memory usage: 906.4+ KB
```

In [70]:
```
df.head(2)
```

Out[70]:

	model1	model2	version	price	km	fuel	CV_fisc	HorseP	Gearbox_auto	L_by_100km	numbe_seats	doors_nb	Euro_stand
0	RENAULT	MEGANE 4	IV 1.6 TCE 205 ENERGY GT EDC7	23440.0	78325.0	0.0	11.0	205.0	1.0	4.9	5.0	5.0	6.0
1	RENAULT	CLIO 5	V 1.0 TCE 100 INTENS	19930.0	27008.0	0.0	5.0	101.0	0.0	5.3	5.0	5.0	6.0

In [71]:
```
df.isna().sum()
```

Out[71]:
```
model1               0
model2               0
version              0
price                0
km                   0
fuel                 0
CV_fisc              0
HorseP               0
Gearbox_auto         0
L_by_100km           0
numbe_seats          0
doors_nb             0
Euro_stand           0
Length               0
Nb_option            0
registration_date    0
dtype: int64
```

In [72]:
```
df.columns
```

Out[72]:
```
Index(['model1', 'model2', 'version', 'price', 'km', 'fuel', 'CV_fisc',
       'HorseP', 'Gearbox_auto', 'L_by_100km', 'numbe_seats', 'doors_nb',
       'Euro_stand', 'Length', 'Nb_option', 'registration_date'],
      dtype='object')
```

In [73]:
```
df.price.describe().to_frame().T
```

Out[73]:

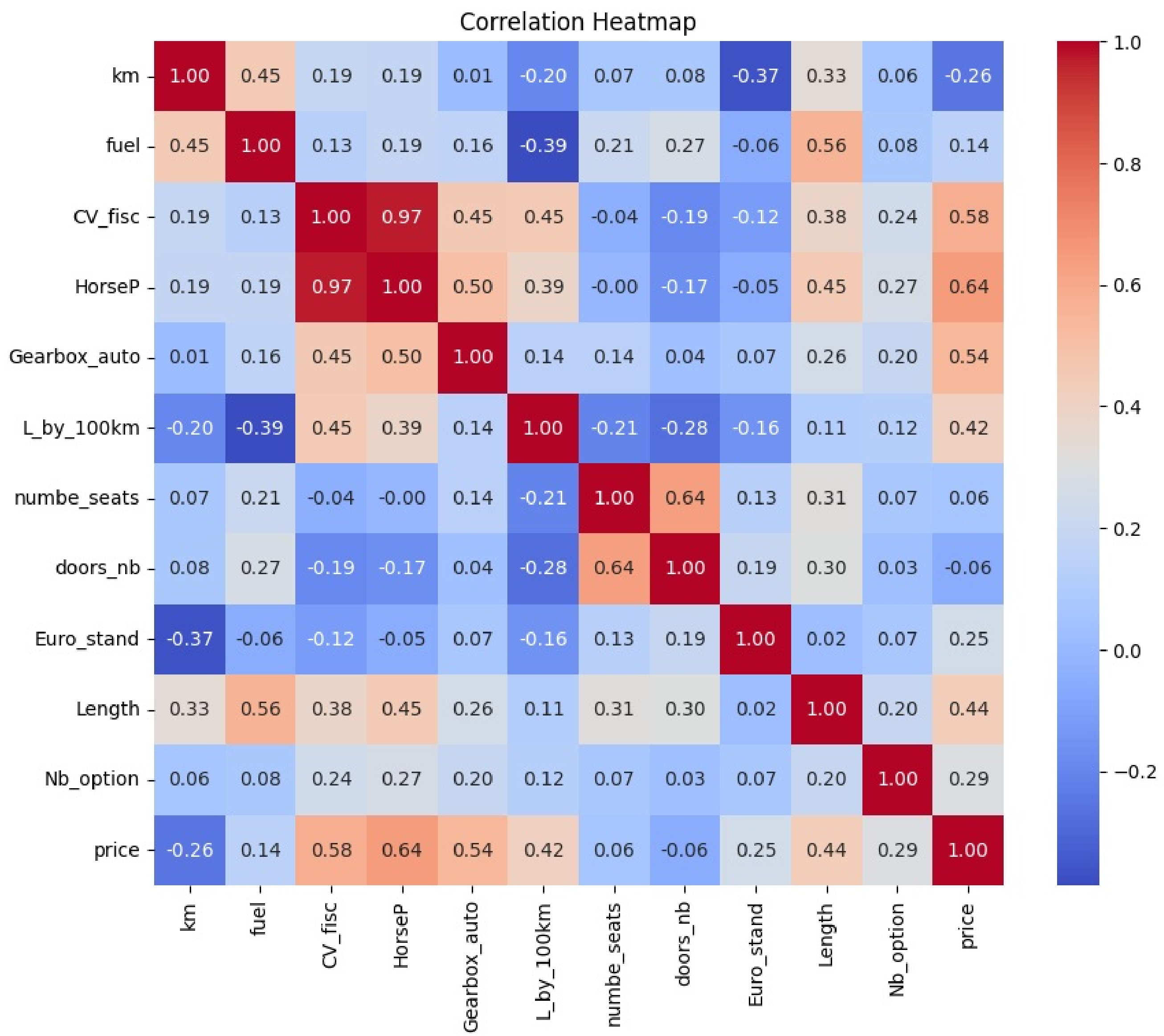
	count	mean	std	min	25%	50%	75%	max
price	7250.0	41867.362759	69438.583438	4910.0	18010.0	24470.0	35680.0	793620.0

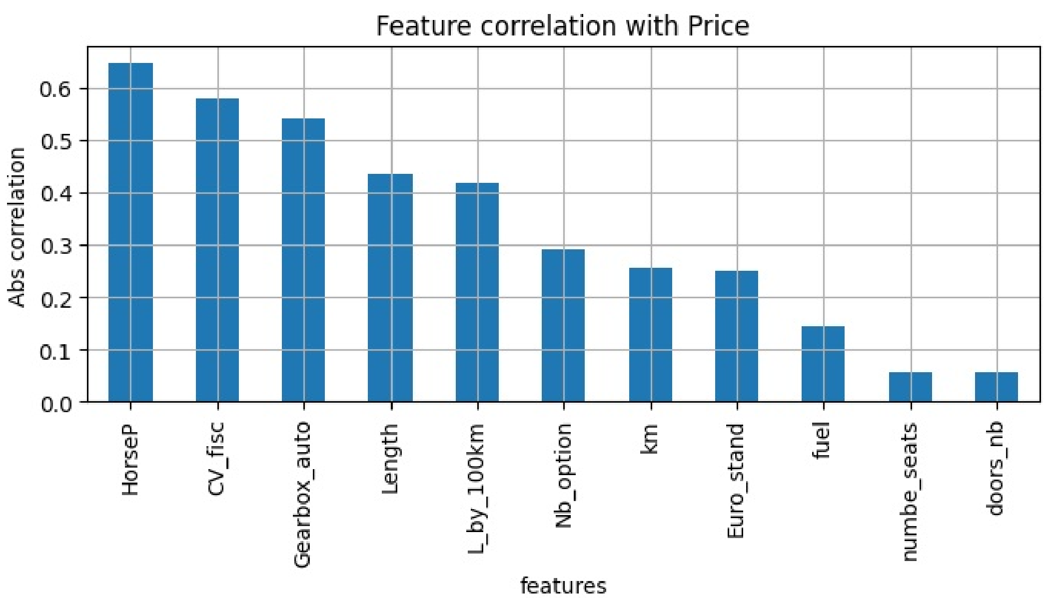
In [74]:
```
df2= df[df.price <40000]
len(df2)/len(df)
```

Out[74]:
```
0.8137931034482758
```

In [75]:
```
df3= df2[['km', 'fuel', 'CV_fisc','HorseP', 'Gearbox_auto',
       'L_by_100km', 'numbe_seats', 'doors_nb',
       'Euro_stand', 'Length', 'Nb_option', 'price']]
```

In [109…
```
correlation_matrix= df3.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()

plt.figure(figsize=(8, 3))
S= correlation_matrix["price"].drop("price")
S=S.abs().sort_values(ascending=False)
S.plot.bar()
plt.grid()
plt.xlabel("features")
plt.ylabel("Abs correlation")
plt.title("Feature correlation with Price")
plt.show()
```





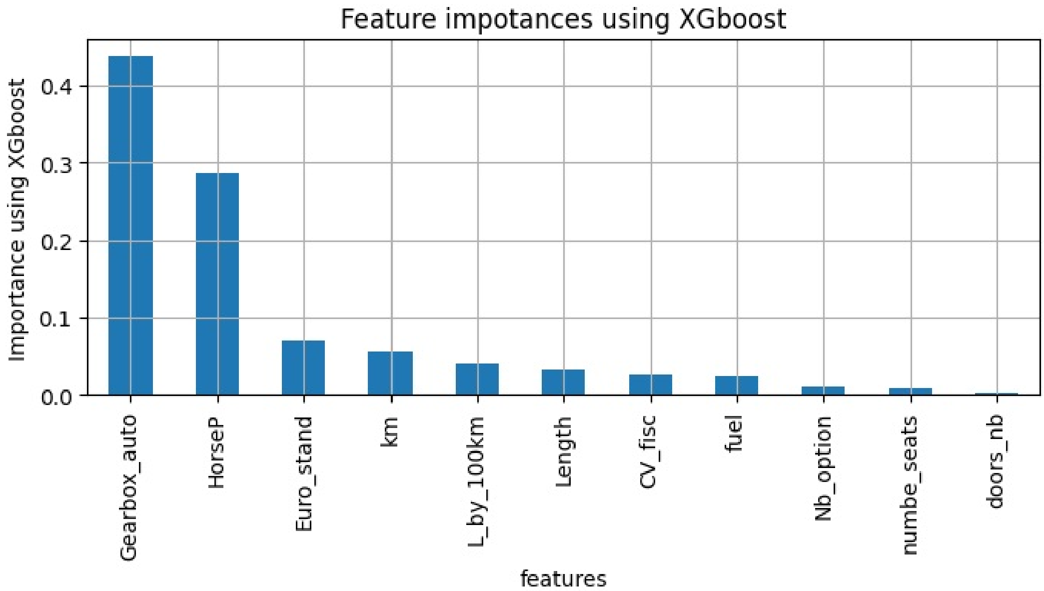
In [77]:
```
colsx=['km', 'fuel', 'CV_fisc','HorseP', 'Gearbox_auto',
       'L_by_100km', 'numbe_seats', 'doors_nb',
       'Euro_stand', 'Length', 'Nb_option']
coly= 'price'
```

In [78]:
```
X= df3[colsx].values
y= df3[ coly].values
```

In [79]:
```
# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Use Xgboost to select only 4 inputs

In [92]:
```
# Define and train the model
model = xgb.XGBRegressor(
    n_estimators=100,    # Number of trees
    max_depth=3,         # Maximum tree depth
    eta=0.1,             # Learning rate
    objective='reg:squarederror'  # Regression objective
    ,random_state=42
)

model.fit(X_train, y_train)

importance = model.feature_importances_
S=pd.Series(importance, index = colsx)
S=S.sort_values(ascending=False)
plt.figure(figsize=(8, 3))
S.plot.bar()
plt.grid()
plt.xlabel("features")
plt.ylabel("Importance using XGboost")
plt.title("Feature impotances using XGboost")
plt.show()
```



In [102…
```
print("The 4 important features: ", list(S.head(4).index))
```

```
The 4 important features:  ['Gearbox_auto', 'HorseP', 'Euro_stand', 'km']
```

In [103…
```
NewColx= list(S.head(4).index)
NewColx
```

Out[103]:
```
['Gearbox_auto', 'HorseP', 'Euro_stand', 'km']
```

In [104…
```
FileName="..\data\processed\df_price_4inputs.csv"
df3[NewColx+["price"]].astype("float32").\
        to_csv(FileName,
                index = False)
```

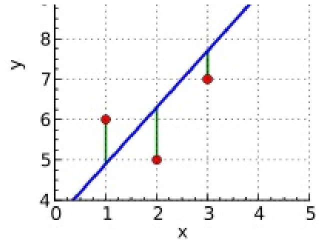
Dataset with 4 inputs

In [6]:
```
FileName="..\data\processed\df_price_4inputs.csv"
dfnew= pd.read_csv(FileName).astype("float32")
print("Df columns: ", list(dfnew.columns))
X= dfnew.iloc[:,:4].values
y= dfnew.iloc[:,4].values
# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
Df columns:  ['Gearbox_auto', 'HorseP', 'Euro_stand', 'km', 'price']
```

Example of ML models

Linear Regression



In [8]:
```
from sklearn.linear_model import LinearRegression
```

In [9]:
```
LR = LinearRegression()
```

In [119…
```
_=LR.fit(X_train, y_train)
```

In [120…
```
# Save the model
joblib.dump(LR, r'../models/LinearReg/LR_model.pkl')
```

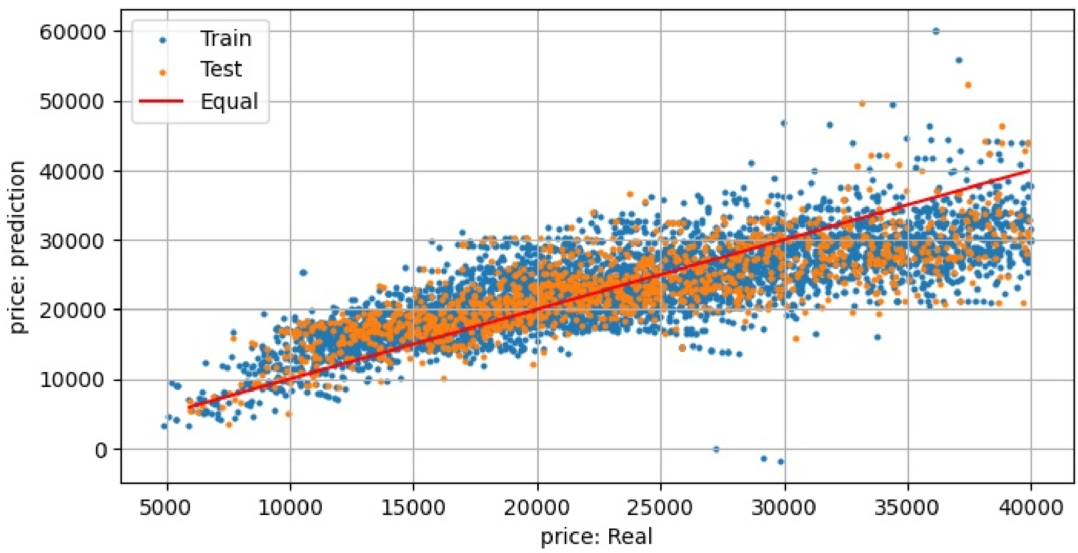
Out[120]:
```
['../models/LinearReg/LR_model.pkl']
```

In [121…
```
# Load the model
LR_model = joblib.load(r'../models/LinearReg/LR_model.pkl')
```

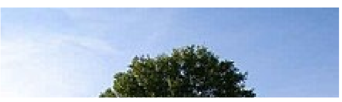
In [122…
```
y_pred_test= LR_model.predict(X_test)
y_pred_train= LR_model.predict(X_train)
```

In [141…
```
# Evaluate the model
maeTrain = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error Train: {maeTrain:.2f}")

maeTest = mean_absolute_error(y_test, y_pred_test)
print(f"Mean Squared Error Test: {maeTest:.2f}")

## Plot
plt.figure(figsize=(8, 4))
plt.scatter(y_train, y_pred_train, s= 4, label="Train")
plt.scatter(y_test, y_pred_test, s=4 , label="Test")

plt.plot([y_test.min(),y_test.max()],
         [y_test.min(),y_test.max()],
         c="r",
        label = "Equal")

plt.xlabel("price: Real")
plt.ylabel("price: prediction")
plt.grid()
plt.legend()
plt.show()
```

```
Mean Squared Error Train: 3759.17
Mean Squared Error Test: 3758.83
```



Decision Tree Regressor



In [142…

```
DTR=DecisionTreeRegressor(max_depth=10, random_state=0)
```

In [144…

```
_=DTR.fit(X_train, y_train)
```

In [107…

```
# Save the model
joblib.dump(DTR, r'../models/Trees/dtr_model.pkl')
```

Out[107]:

```
['../models/Trees/dtr_model.pkl']
```

In [145…

```
# Load the model
dtr_model = joblib.load(r'../models/Trees/dtr_model.pkl')
```

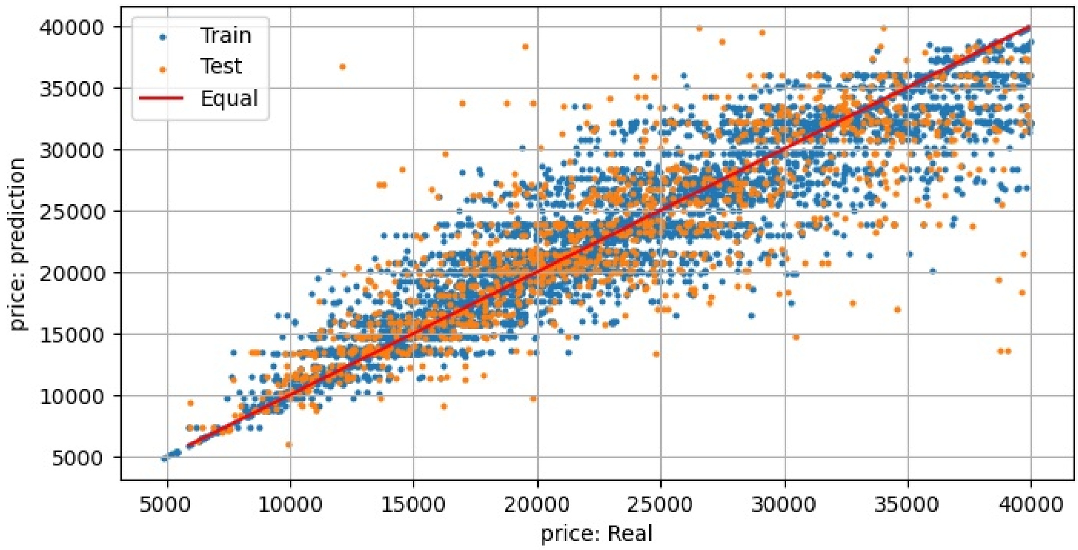
In [147…

```
y_pred_test= dtr_model.predict(X_test)
y_pred_train= dtr_model.predict(X_train)
```

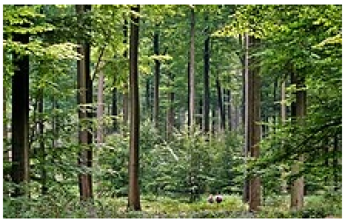
In [148…

```
# Evaluate the model
maeTrain = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error Train: {maeTrain:.2f}")

maeTest = mean_absolute_error(y_test, y_pred_test)
print(f"Mean Squared Error Test: {maeTest:.2f}")

## Plot
plt.figure(figsize=(8, 4))
plt.scatter(y_train, y_pred_train, s= 4, label="Train")
plt.scatter(y_test, y_pred_test, s=4 , label="Test")

plt.plot([y_test.min(),y_test.max()],
         [y_test.min(),y_test.max()],
         c="r",
        label = "Equal")

plt.xlabel("price: Real")
plt.ylabel("price: prediction")
plt.grid()
plt.legend()
plt.show()
```

```
Mean Squared Error Train: 2135.29
Mean Squared Error Test: 3099.48
```



Random forest regressor



In [150…
```python
RF= RandomForestRegressor(n_estimators=3,
                          max_depth=8, random_state=0)
```

In [151…
```python
_=RF.fit(X_train, y_train)
```

In [267…
```python
import joblib

# Save the model to a file
joblib.dump(RF, '../models/RF/random_forest_model.pkl')
```

Out[267]: ['../models/RF/random_forest_model.pkl']

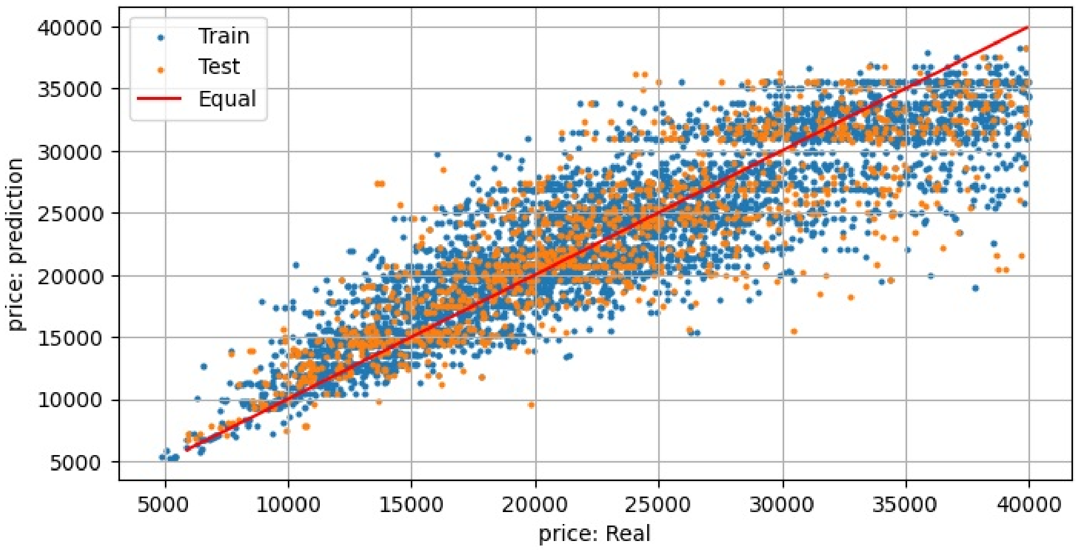
In [152…
```python
# Load the model from the file
RF_model = joblib.load('../models/RF/random_forest_model.pkl')
```

In [153…
```python
y_pred= RF_model .predict(X_test)
```

In [154…
```python
y_pred_test= RF_model.predict(X_test)
y_pred_train= RF_model.predict(X_train)
```

In [155…
```python
# Evaluate the model
maeTrain = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error Train: {maeTrain:.2f}")

maeTest = mean_absolute_error(y_test, y_pred_test)
print(f"Mean Squared Error Test: {maeTest:.2f}")

## Plot
plt.figure(figsize=(8, 4))
plt.scatter(y_train, y_pred_train, s= 4, label="Train")
plt.scatter(y_test, y_pred_test, s=4 , label="Test")

plt.plot([y_test.min(),y_test.max()],
         [y_test.min(),y_test.max()],
         c="r",
        label = "Equal")

plt.xlabel("price: Real")
plt.ylabel("price: prediction")
plt.grid()
plt.legend()
plt.show()
```

```
Mean Squared Error Train: 2712.65
Mean Squared Error Test: 3124.63
```



XGBOOST

XGBoost

In [156…

```
base_score=X_train.mean()
base_score
```

Out[156]:

```
18476.805
```

In [158…

```
# Define and train the model
model = xgb.XGBRegressor(
    n_estimators=100,   # Number of trees
    max_depth=3,        # Maximum tree depth
    eta=0.1,            # Learning rate
    objective='reg:squarederror'  # Regression objective
    ,random_state=42
    ,base_score=base_score
)

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

In [677…

```
# Save the model
joblib.dump(model, r'../models/xgboost/xgb_model.pkl')
```

Out[677]:

```
['../models/xgboost/xgb_model.pkl']
```

In [160…

```
# Load the model
xgb_model = joblib.load(r'../models/xgboost/xgb_model.pkl')
```

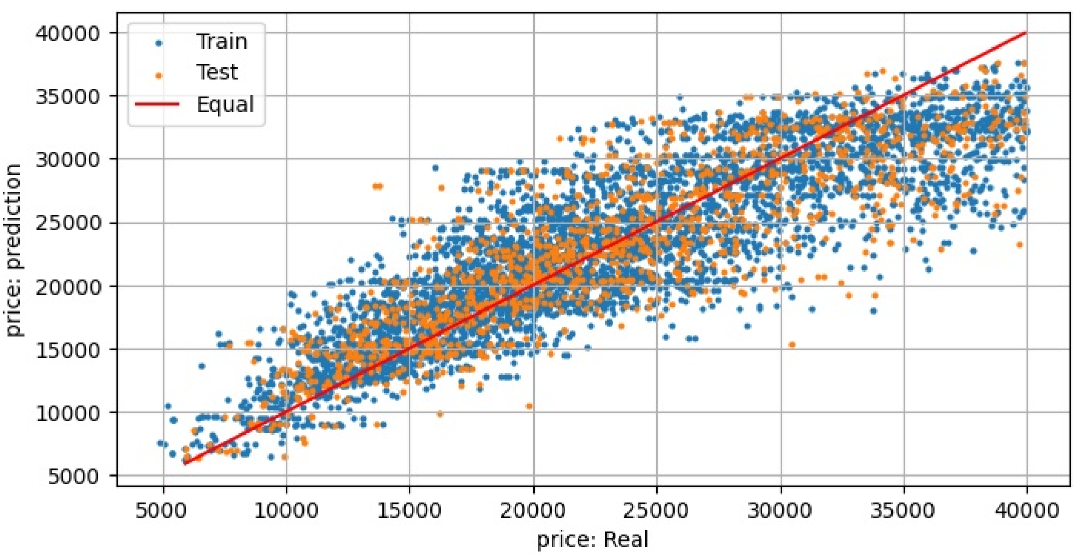
In [161…

```
y_pred_test= xgb_model.predict(X_test)
y_pred_train= xgb_model.predict(X_train)
```

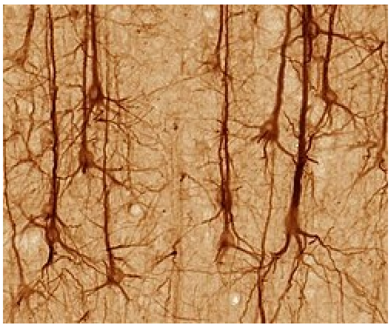
In [162…

```
# Evaluate the model
maeTrain = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error Train: {maeTrain:.2f}")

maeTest = mean_absolute_error(y_test, y_pred_test)
print(f"Mean Squared Error Test: {maeTest:.2f}")

## Plot
plt.figure(figsize=(8, 4))
plt.scatter(y_train, y_pred_train, s= 4, label="Train")
plt.scatter(y_test, y_pred_test, s=4 , label="Test")

plt.plot([y_test.min(),y_test.max()],
         [y_test.min(),y_test.max()],
         c="r",
        label = "Equal")

plt.xlabel("price: Real")
plt.ylabel("price: prediction")
plt.grid()
plt.legend()
plt.show()
```

```
Mean Squared Error Train: 3011.73
Mean Squared Error Test: 3161.42
```



DNN



In [163…
```
X_train.shape, X_test.shape
```

Out[163]:
```
((4720, 4), (1180, 4))
```

In [165…
```
# Define the DNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)),  # Input and 1st hidden laye
    tf.keras.layers.Dense(8, activation='relu'),  # 2nd hidden layer
    tf.keras.layers.Dense(4, activation='relu'),  # 3rd hidden layer
    tf.keras.layers.Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

In [90]:
```
# Train the model
history = model.fit(X_train, y_train,
                    validation_split=0.2, epochs=30,
                    batch_size=32, verbose=0)
```

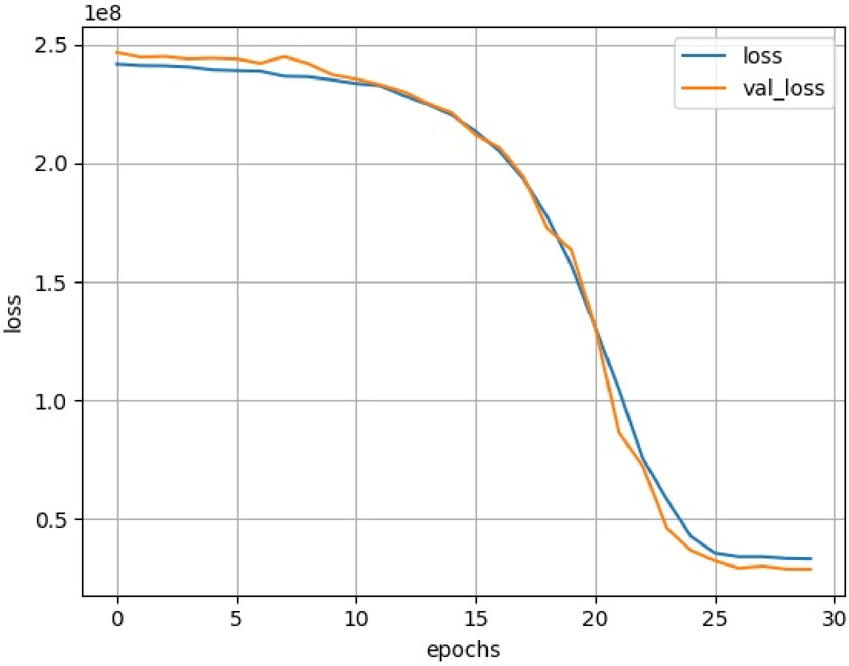
In [98]:
```
hist=history.history
```

In [122…
```
# Save the dictionary
with open("../data/processed/tf_hist.pkl", "wb") as file:
    pickle.dump(hist, file)

model.save('../models/DNN/tf_model.keras', include_optimizer=False)
```

In [167…
```
# Load the dictionary
with open("../data/processed/tf_hist.pkl", "rb") as file:
    load_hist= pickle.load(file)

tf_model=tf.keras.models.load_model('../models/DNN/tf_model.keras')
```

In [168…
```
print(list(load_hist.keys()))
for c in load_hist.keys():
    if 'loss' in c:
        plt.plot(load_hist[c], label = c)
plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
plt.grid()
plt.show()
```

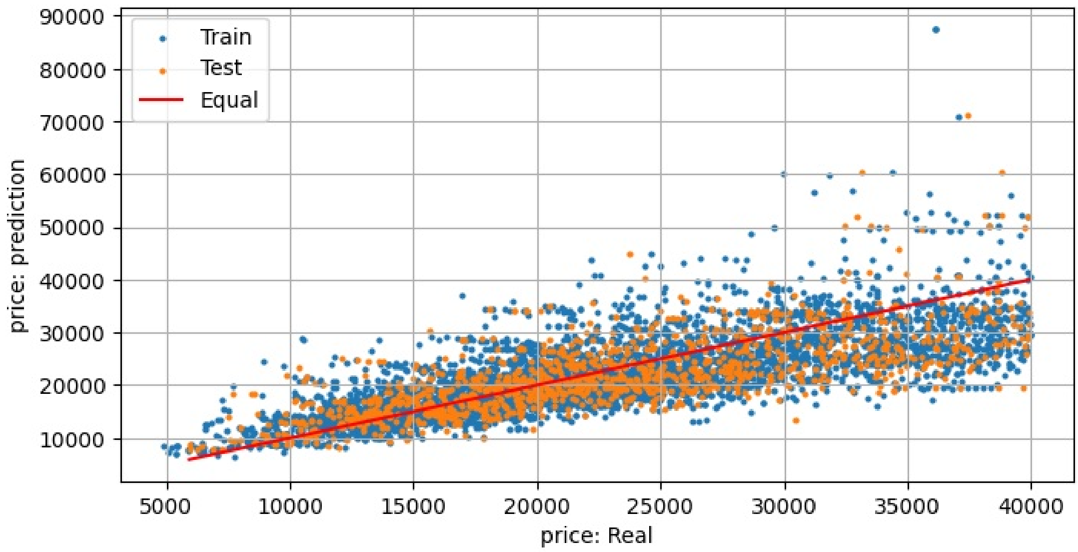
```
['loss', 'mae', 'val_loss', 'val_mae']
```



In [172…
```
y_pred_test= tf_model.predict(X_test,batch_size=32,verbose=0)
y_pred_train= tf_model.predict(X_train,batch_size=32,verbose=0)
```

In [173…
```
# Evaluate the model
maeTrain = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error Train: {maeTrain:.2f}")

maeTest = mean_absolute_error(y_test, y_pred_test)
print(f"Mean Squared Error Test: {maeTest:.2f}")

## Plot
plt.figure(figsize=(8, 4))
plt.scatter(y_train, y_pred_train, s= 4, label="Train")
plt.scatter(y_test, y_pred_test, s=4 , label="Test")

plt.plot([y_test.min(),y_test.max()],
         [y_test.min(),y_test.max()],
         c="r",
        label = "Equal")

plt.xlabel("price: Real")
plt.ylabel("price: prediction")
plt.grid()
plt.legend()
plt.show()
```

```
Mean Squared Error Train: 4090.79
Mean Squared Error Test: 4334.73
```



Model to C++ (Arduino Language)

Utils : for all models

In [3]:
```
def array_to_arduino(x):
    """
    Helper function to convert a Python list or NumPy array to Arduino array format
    for use in the generated Arduino code.
    It converts the input into a string format where square brackets [] are replaced
    with curly braces {}.

    Input:
    - x: List or array to be converted

    Output:
    - Formatted string that can be used in Arduino code
    """
    x = str(x.tolist())  # Convert array to list and then string
    x = x.replace('[', '{')  # Replace square brackets with curly braces
    x = x.replace(']', '}')  # Replace closing square bracket with closing curly brace
    return x
```

Linear Regression

Conversion of Linear Regression model to C++ (arduino language)

In [10]:
```
# Load the model
LR_model = joblib.load(r'../models/LinearReg/LR_model.pkl')
```

In [11]:
```
# Sub inputs/outpusts to test the arduino model: 10 samples
sub_X=X_train[:10]
sub_y=LR_model.predict(sub_X)
```

In [12]:
```
# Get linear regression parameters
coef = LR_model.coef_
bias = LR_model.intercept_
print("coef: ", coef.tolist())
print("bias: ", bias )
```

```
coef:  [3661.1865234375, 119.34064483642578, 2962.884521484375, -0.054161868989467 62]
bias:  -7844.6016
```

In [13]:
```
sub_X.shape, coef.shape
```

Out[13]:
```
((10, 4), (4,))
```

In [14]:
```
# Understund the Linear regression algo
print("y with model predict\n" ,sub_y)
print("y with matrix calculation: Y = X.coef + bias \n",
    (sub_X.dot(coef.reshape(-1,1))+bias).flatten())
```

```python
print("the result is the same")
```

```
y with model predict
 [19074.861 22590.434 18458.254 20624.408 20219.445 29240.262 32525.
 27160.86  25408.605 26429.555]
y with matrix calculation: Y = X.coef + bias
 [19074.861 22590.434 18458.254 20624.408 20219.445 29240.262 32525.
 27160.86  25408.605 26429.555]
the result is the same
```

In [16]:

```python
def LinearRegToC (model, X, y):
    """Convert a Linear regression model (sklearn) to C++ (Arduino)
    Model : trained LR model
    X,y : input outputs to test the arduino code
    """
    codeInit="""

const int Nv = NvReplace;
const int dimX = dimXReplace;

/////// Xy //////
const float X [] PROGMEM  = Xreplace;

const float y[] PROGMEM  = yreplace;




///////////////// Model
const float coef[] PROGMEM = coefreplace;
const float Bias = Biasreplace;
float LinearReg ( float X[] ) {
float Out=Bias;
for(int j = 0; j<dimX;j++){
    Out+=X[j]*pgm_read_float_near(&coef[j]);
}

return Out;
}




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

void loop() {
unsigned long timestart;
unsigned long timeend;
float Xi[dimX];
float yc;


Serial.println("Cal_Ardui,Expected,Delta_time(us)");
for (int l=0;l<Nv;l++){
for(int j = 0; j<dimX;j++){
    Xi[j]=pgm_read_float_near(&X[l*dimX+j]);
}
timestart=micros();
yc=LinearReg(Xi);
timeend=micros();
Serial.print(yc);
Serial.print(",");
Serial.print(pgm_read_float_near(&y[l]),6);
Serial.print(",");
Serial.println(timeend-timestart);
}
Serial.println("====The End=====");
while(1);
}
"""

    Nv, dimX= X.shape
    Nv, dimX= str(Nv), str(dimX)
    Xs=array_to_arduino(X.flatten())
    ys=array_to_arduino(y)
    coef = array_to_arduino(model.coef_)
    bias = str(model.intercept_)



    codeInit= codeInit.replace("NvReplace",Nv)
    codeInit= codeInit.replace("dimXReplace",dimX)
```

```
    codeInit= codeInit.replace("Xreplace",Xs)
    codeInit= codeInit.replace("yreplace",ys)
    codeInit= codeInit.replace("coefreplace",coef)
    codeInit= codeInit.replace("Biasreplace", bias)

    return codeInit
```

In [17]:
```
# Convert the model
arduino_code= LinearRegToC (LR_model, sub_X, sub_y)
```

In [194…
```
# save the arduino code
ino_file="../ArduinoCode/LinearReg.ino" # Path of the file
ino_file=ino_file.replace(".ino" ,"")
current_directory = os.getcwd()
new_directory_path = os.path.join(current_directory, ino_file)
try:
    os.makedirs(new_directory_path)
except: pass

path=ino_file+"/"+ino_file.split("/")[-1]+".ino"
with open(path,'w+') as f:
    f.write(arduino_code)

    print(path, "saved")
```

```
../ArduinoCode/LinearReg/LinearReg.ino saved
```

The arduino memory usnig

Sketch uses 3906 bytes (12%) of program storage space. Maximum is 30720 bytes. Global variables use 252 bytes (12%) of dynamic memory, leaving 1796 bytes for local variables. Maximum is 2048 bytes.

In [18]:
```
# The arduino serial print result
serialPrint="""
Cal_Ardui,Expected,Delta_time(us)
19074.86,19074.861328,68
22590.43,22590.433593,76
18458.25,18458.253906,80
20624.41,20624.408203,76
20219.45,20219.445312,76
29240.26,29240.261718,80
32525.00,32525.000000,84
27160.86,27160.859375,80
25408.60,25408.605468,80
26429.56,26429.554687,88
====The End====="""
```

In [19]:
```
# Convert the serial result to DF
data = serialPrint.split("\n")[1:-1]
data=[x.split(",") for x in data]
DF_serial= pd.DataFrame( data[1:], columns= data[0]).astype("float32")
DF_serial
```

Out[19]:

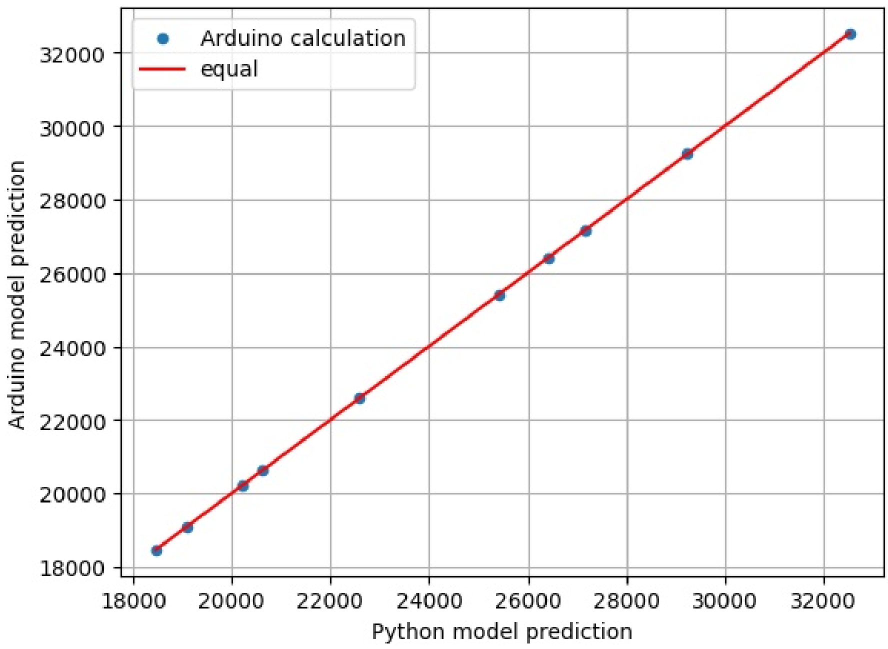
	Cal_Ardui	Expected	Delta_time(us)
0	19074.859375	19074.861328	68.0
1	22590.429688	22590.433594	76.0
2	18458.250000	18458.253906	80.0
3	20624.410156	20624.408203	76.0
4	20219.449219	20219.445312	76.0
5	29240.259766	29240.261719	80.0
6	32525.000000	32525.000000	84.0
7	27160.859375	27160.859375	80.0
8	25408.599609	25408.605469	80.0
9	26429.560547	26429.554688	88.0

In [20]:
```
print("The AVG prediction time of one input is",
      (DF_serial['Delta_time(us)'].mean()/1000).round(2),
      "ms"
     )
```

```
The AVG prediction time of one input is 0.08 ms
```

In [21]:
```
# Ploting
DF_serial.plot.scatter(x='Expected', y='Cal_Ardui',  marker='o', label="Arduino calculation")
xx=[DF_serial['Expected'].min(), DF_serial['Expected'].max()]
plt.plot(xx,xx, c='r', label="equal")
```

```
plt.legend()
plt.xlabel("Python model prediction")
plt.ylabel("Arduino model prediction")
plt.grid()
plt.show()
```



In [199…

```
# The arduino and python model have the same result.
```

Decision tree Regressor

Conversion of Decision tree Regressor model to C++ (arduino language)

In [200…

```
# Load the model
dtr_model = joblib.load(r'../models/Trees/dtr_model.pkl')
```

In [ ]:

```
# Sub inputs/outpusts to test the arduino model: 10 samples
sub_X=X_train[:10]
sub_y=dtr_model.predict(sub_X)
```

In [215…

```
# Export and print the tree structure
tree_text = export_text(dtr_model)
print("Example of tree txt")
print(tree_text[:300])
```

```
Example of tree txt
|--- feature_1 <= 127.50
|   |--- feature_1 <= 90.50
|   |   |--- feature_2 <= 5.50
|   |   |   |--- feature_3 <= 146700.00
|   |   |   |   |--- feature_2 <= 4.50
|   |   |   |   |   |--- feature_3 <= 105865.00
|   |   |   |   |   |   |--- feature_3 <= 92399.00
|   |   |   |   |   |   |   |--- featu
```

In [218…

```
def get_cpp_code_from_tree(tree, feature_names):
    """
    Convert a decision tree to if/else code C++
    """
    left      = tree.tree_.children_left
    right     = tree.tree_.children_right
    threshold = tree.tree_.threshold
    features  = [feature_names[i] for i in tree.tree_.feature]
    value = tree.tree_.value
    code = ""
    def recurse(left, right, threshold, features, node):
            nonlocal code
            if (threshold[node] != -2):
                    code+="if ( " + features[node] + " <= " + str(threshold[node]) + " ) {\n"
                    if left[node] != -1:
                            recurse (left, right, threshold, features,left[node])
                    code+="} else {\n"
                    if right[node] != -1:
                            recurse (left, right, threshold, features,right[node])
                    code+="}\n"
            else:
```

```python
                    code+="return " + str(value[node]).replace("[","").replace("]","")+";\n"

    recurse(left, right, threshold, features, 0)
    return code
```

In [220…

```python
# Example of conversion
TXT=get_cpp_code_from_tree(dtr_model, ["a","b","c","d"])
print(TXT[:250])
```

```
if ( b <= 127.5 ) {
if ( b <= 90.5 ) {
if ( c <= 5.5 ) {
if ( d <= 146700.0 ) {
if ( c <= 4.5 ) {
if ( d <= 105865.0 ) {
if ( d <= 92399.0 ) {
if ( d <= 85367.0 ) {
return 10060.;
} else {
return 10690.;
}
} else {
return 8360.;
}
} else {
if ( b <=
```

In [221…

```python
def convert_DecTree_To_C(model, X,y):
    codeInit="""

const int Nv = NvReplace;
const int dimX = dimXReplace;

/////// Xy //////
const float X [] PROGMEM  = Xreplace;

const float y[] PROGMEM  = yreplace;



///////////////// TREE
float DecisionTreeReg ( float X[] ) {
IF_ELSE_CONDITION_replace
}




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

void loop() {
unsigned long timestart;
unsigned long timeend;
float Xi[dimX];
float yc;


Serial.println("Cal_Ardui,Expected,Delta_time(us)");
for (int l=0;l<Nv;l++){
for(int j = 0; j<dimX;j++){
    Xi[j]=pgm_read_float_near(&X[l*dimX+j]);
}
timestart=micros();
yc=DecisionTreeReg(Xi);
timeend=micros();
Serial.print(yc);
Serial.print(",");
Serial.print(pgm_read_float_near(&y[l]),6);
Serial.print(",");
Serial.println(timeend-timestart);
}
Serial.println("====The End=====");
while(1);
}
"""

    Nv, dimX= X.shape
    Nv, dimX= str(Nv), str(dimX)
    Xs=array_to_arduino(X.flatten())
    ys=array_to_arduino(y)

    features = ["X["+str(i)+"]" for i in range(X.shape[1])]
```

```
    ifelsecode = get_cpp_code_from_tree(model, features)

    codeInit= codeInit.replace("NvReplace",Nv)
    codeInit= codeInit.replace("dimXReplace",dimX)
    codeInit= codeInit.replace("Xreplace",Xs)
    codeInit= codeInit.replace("yreplace",ys)
    codeInit= codeInit.replace("IF_ELSE_CONDITION_replace",ifelsecode)

    return codeInit
```

In [122…

```
arduino_code = convert_DecTree_To_C(dtr_model, X,y)
```

In [123…

```
# save the arduino code
ino_file="../ArduinoCode/DecisionTree"
ino_file=ino_file.replace(".ino" ,"")
current_directory = os.getcwd()
new_directory_path = os.path.join(current_directory, ino_file)
try:
    os.makedirs(new_directory_path)
except: pass

path=ino_file+"/"+ino_file.split("/")[-1]+".ino"
with open(path,'w+') as f:
    f.write(arduino_code)

    print(path, "saved")
```

```
../ArduinoCode/DecisionTree/DecisionTree.ino saved
```

The arduino memory usnig

Sketch uses 27532 bytes (89%) of program storage space. Maximum is 30720 bytes. Global variables use 252 bytes (12%) of dynamic memory, leaving 1796 bytes for local variables. Maximum is 2048 bytes.

In [22]:

```
# The arduino serial print result
serialPrint="""
Cal_Ardui,Expected,Delta_time(us)
19870.97,19870.972656,40
26722.96,26722.962890,48
16522.86,16522.857421,48
18817.56,18817.560546,44
17535.56,17535.554687,48
17620.00,17620.000000,44
35083.11,35083.109375,48
32269.54,32269.535156,44
23814.67,23814.671875,40
31409.13,31409.130859,48
====The End====="""
```

In [23]:

```
# Convert the serial result to DF
data = serialPrint.split("\n")[1:-1]
data=[x.split(",") for x in data]
DF_serial= pd.DataFrame( data[1:], columns= data[0]).astype("float32")
DF_serial
```

Out[23]:

	Cal_Ardui	Expected	Delta_time(us)
0	19870.970703	19870.972656	40.0
1	26722.960938	26722.962891	48.0
2	16522.859375	16522.857422	48.0
3	18817.560547	18817.560547	44.0
4	17535.560547	17535.554688	48.0
5	17620.000000	17620.000000	44.0
6	35083.109375	35083.109375	48.0
7	32269.539062	32269.535156	44.0
8	23814.669922	23814.671875	40.0
9	31409.130859	31409.130859	48.0

In [24]:

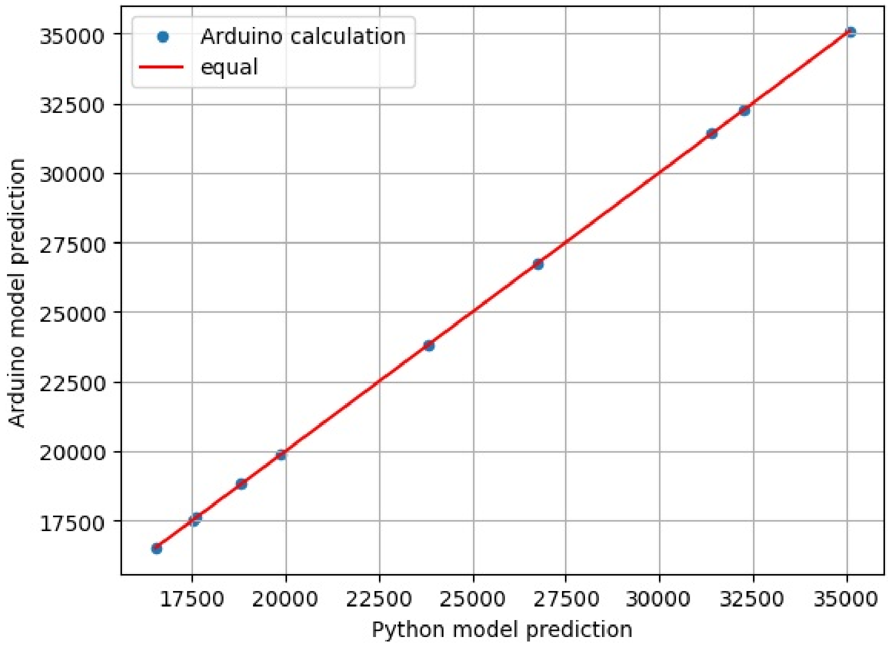
```
print("The AVG prediction time of one input is",
      (DF_serial['Delta_time(us)'].mean()/1000).round(2),
      "ms"
     )
```

```
The AVG prediction time of one input is 0.05 ms
```

In [25]:

```
# Ploting
DF_serial.plot.scatter(x='Expected', y='Cal_Ardui',  marker='o', label="Arduino calculation")
```

```
xx=[DF_serial['Expected'].min(), DF_serial['Expected'].max()]
plt.plot(xx,xx, c='r', label="equal")
plt.legend()
plt.xlabel("Python model prediction")
plt.ylabel("Arduino model prediction")
plt.grid()
plt.show()
```



Random forest regressor

Conversion of Random forest regressor model to C++ (arduino language)

In [236…
```
# Load the model from the file
RF_model = joblib.load('../models/RF/random_forest_model.pkl')
```

In [ ]:
```
# Sub inputs/outpusts to test the arduino model: 10 samples
sub_X=X_train[:10]
sub_y=RF_model.predict(sub_X)
```

In [238…
```
def convert_RandForest_To_C(model, X,y):
    codeInit="""

const int Nv = NvReplace;
const int dimX = dimXReplace;

/////// Xy //////
const float X [] PROGMEM  = Xreplace;

const float y[] PROGMEM  = yreplace;



///////////////// TREES

TREES_replace


///////////////// RANDOM FOREST

RF_replace



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

void loop() {
unsigned long timestart;
unsigned long timeend;
float Xi[dimX];
float yc;


Serial.println("Cal_Ardui,Expected,Delta_time(us)");
```

```
for (int l=0;l<Nv;l++){
for(int j = 0; j<dimX;j++){
Xi[j]=pgm_read_float_near(&X[l*dimX+j]);
}
timestart=micros();
yc=RandForestReg(Xi);
timeend=micros();
Serial.print(yc);
Serial.print(",");
Serial.print(pgm_read_float_near(&y[l]),6);
Serial.print(",");
Serial.println(timeend-timestart);
}
Serial.println("====The End=====");
while(1);
}
"""
    code_trees=""
    code_randForest="\n\n\nfloat RandForestReg ( float X[] ) {\nfloat out=0;\n"
    features = ["X["+str(i)+"]" for i in range(X.shape[1])]
    trees = model.estimators_
    for i, tree in enumerate(trees):
        code_tree=get_cpp_code_from_tree(tree,  features )
        code_tree="\n\n\nfloat Tree"+str(i)+" ( float X[] ) {\n"+code_tree+"\n}\n"
        code_trees+=code_tree

        code_randForest+="out+=Tree"+str(i)+" (X);\n";

    code_randForest+="out=out/"+str(model.n_estimators)+";\nreturn out;\n}\n"




    Nv, dimX= X.shape
    Nv, dimX= str(Nv), str(dimX)
    Xs=array_to_arduino(X.flatten())
    ys=array_to_arduino(y)



    codeInit= codeInit.replace("NvReplace",Nv)
    codeInit= codeInit.replace("dimXReplace",dimX)
    codeInit= codeInit.replace("Xreplace",Xs)
    codeInit= codeInit.replace("yreplace",ys)

    codeInit= codeInit.replace("TREES_replace",code_trees)
    codeInit= codeInit.replace("RF_replace",code_randForest)


    return codeInit
```

In [ ]:

```
arduino_code = convert_RandForest_To_C(RF_model, X,y)
```

In [239…

```
# save the arduino code
ino_file="../ArduinoCode/RandForest"
ino_file=ino_file.replace(".ino" ,"")
current_directory = os.getcwd()
new_directory_path = os.path.join(current_directory, ino_file)
try:
    os.makedirs(new_directory_path)
except: pass

path=ino_file+"/"+ino_file.split("/")[-1]+".ino"
with open(path,'w+') as f:
    f.write(arduino_code)

    print(path, "saved")
```

```
../ArduinoCode/RandForest/RandForest.ino saved
```

The arduino memory usnig

Sketch uses 25234 bytes (82%) of program storage space. Maximum is 30720 bytes. Global variables use 252 bytes (12%) of dynamic memory, leaving 1796 bytes for local variables. Maximum is 2048 bytes.

In [26]:

```
# The arduino serial print result
serialPrint="""
Cal_Ardui,Expected,Delta_time(us)
20217.69,20217.689453,120
24530.45,24530.447265,116
18560.16,18560.160156,124
19753.04,19753.039062,120
17726.35,17726.345703,120
```

```
20670.88,20670.882812,136
35056.59,35056.593750,132
31866.39,31866.384765,120
25756.68,25756.675781,120
28062.48,28062.480468,120
====The End====="""
```

In [27]:
```python
# Convert the serial result to DF
data = serialPrint.split("\n")[1:-1]
data=[x.split(",") for x in data]
DF_serial= pd.DataFrame( data[1:], columns= data[0]).astype("float32")
DF_serial
```

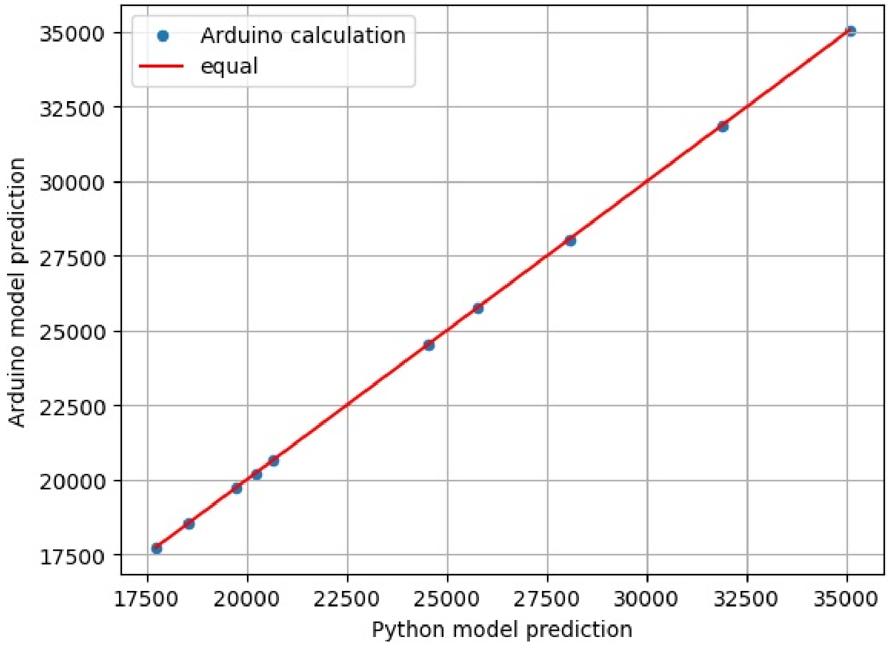
Out[27]:

	Cal_Ardui	Expected	Delta_time(us)
0	20217.689453	20217.689453	120.0
1	24530.449219	24530.447266	116.0
2	18560.160156	18560.160156	124.0
3	19753.039062	19753.039062	120.0
4	17726.349609	17726.345703	120.0
5	20670.880859	20670.882812	136.0
6	35056.589844	35056.593750	132.0
7	31866.390625	31866.384766	120.0
8	25756.679688	25756.675781	120.0
9	28062.480469	28062.480469	120.0

In [28]:
```python
print("The AVG prediction time of one input is",
      (DF_serial['Delta_time(us)'].mean()/1000).round(2),
      "ms"
     )
```

```
The AVG prediction time of one input is 0.12 ms
```

In [29]:
```python
# Ploting
DF_serial.plot.scatter(x='Expected', y='Cal_Ardui',  marker='o', label="Arduino calculation")
xx=[DF_serial['Expected'].min(), DF_serial['Expected'].max()]
plt.plot(xx,xx, c='r', label="equal")
plt.legend()
plt.xlabel("Python model prediction")
plt.ylabel("Arduino model prediction")
plt.grid()
plt.show()
```



XGBoost

Conversion of Xgboost model to C++ (arduino language)

In [248…
```python
# Load the model
xgb_model = joblib.load(r'../models/xgboost/xgb_model.pkl')
```

In [ ]:
```python
# Sub inputs/outpusts to test the arduino model: 10 samples
```

```python
sub_X=X_train[:10]
sub_y=xgb_model .predict(sub_X)
```

In [249…
```python
base_score=X_train.mean()
base_score
```

Out[249]: 18476.805

In [250…
```python
xgb_model.base_score
```

Out[250]: 18476.805

In [251…
```python
# Function: TreesCode
# Description:
# This function generates C++ code representing the decision trees of an XGBoost model.
# It parses the model's JSON representation and recursively converts each tree into a C++ function.
def TreesCode(model):
    """
    Generates C++ code for each decision tree in an XGBoost model.

    The function extracts the tree structure in JSON format from the model and recursively
    traverses each tree to generate a corresponding C++ function. Each function represents
    the decision logic of a single tree, taking an input array `X` and returning the output.

    Args:
        model: The trained XGBoost model containing the decision trees.

    Returns:
        str: A string containing the complete C++ code for all trees in the model.
    """

    # Extract the JSON representation of the tree
    booster = model.get_booster()
    trees = booster.get_dump(dump_format="json")
    cpp_code = ""

    def recurse(node, depth=0):
        """
        Recursive helper function to traverse a tree node and generate corresponding C++ code.
        - If the node is a leaf, it appends a return statement with the leaf value.
        - Otherwise, it generates a conditional statement based on the split condition.

        :param node: Dictionary representation of a tree node.
        :param depth: Current depth of the node for indentation purposes.
        """
        nonlocal cpp_code
        indent = "    " * depth

        # Leaf node
        if "leaf" in node:
            cpp_code += f"{indent}return {node['leaf']};\n"
            return

        split_condition = node['split_condition']
        INDEX_INP= int(node['split'][1:])
        cpp_code += f"{indent}if (X[{INDEX_INP}] < {split_condition}) {{\n"
        recurse(node['children'][0], depth + 1)
        cpp_code += f"{indent}}} else {{\n"
        recurse(node['children'][1], depth + 1)
        cpp_code += f"{indent}}}\n"

    # Generate code for each tree
    for tree_index, tree_json in enumerate(trees):
        cpp_code += f"\n///////////////// TREE {tree_index}\n"
        cpp_code += f"float tree{tree_index}(float X[]) {{\n"
        tree_dict = json.loads(tree_json)
        recurse(tree_dict)
        cpp_code += "}\n\n"

    return cpp_code



# Function: code_trees
# Description:
# Generates the cumulative summation of the predictions from all trees, formatted as C++ code.
# The summation depends on the learning rate and number of trees.
def code_trees(N, learning_rate):
    XGBOOST_CODE= ""
    for index in range(N):
        if learning_rate  == "1":
            XGBOOST_CODE+= f"out+= tree{index}(X);\n"
        else:
```

```python
            XGBOOST_CODE+= f"out+= learning_rate*tree{index}(X);\n"
    return XGBOOST_CODE




# Function: XGBOOST_to_CPP
# Description:
# Converts an XGBoost model to a complete C++ implementation for predictions.
# This includes tree code, model initialization, and a prediction function.
def XGBOOST_to_CPP(model, X, y, base_score):
    # Template for the C++ implementation
    codeInit="""

const int Nv = NvReplace;
const int dimX = dimXReplace;

float base_score =   base_score_Replace ;
float learning_rate = learning_rate_Replace ;

/////// Xy //////
const float X [] PROGMEM  = Xreplace;

const float y[] PROGMEM  = yreplace;



////////////////// TREES ////////////////////////////
////////////////////////////////////////////////////
TREES_CODE_replace

////////////////// XGBOOST MODEL /////////////////
/////////////////////////////////////////////////
float XGBpred(float X[]){
float out = 0;
XGBOOST_CODE_replace
out = out+base_score;
return out;}



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

void loop() {
unsigned long timestart;
unsigned long timeend;
float Xi[dimX];
float yc;


Serial.println("Cal_Ardui,Expected,Delta_time(us)");
for (int l=0;l<Nv;l++){
for(int j = 0; j<dimX;j++){
Xi[j]=pgm_read_float_near(&X[l*dimX+j]);
}
timestart=micros();
yc=XGBpred(Xi);
timeend=micros();
Serial.print(yc);
Serial.print(",");
Serial.print(pgm_read_float_near(&y[l]),6);
Serial.print(",");
Serial.println(timeend-timestart);
}
Serial.println("====The End=====");
while(1);
}
"""



    if model.base_score is not None:
        base_score = str(model.base_score)
    elif base_score is not None:
        base_score = str(base_score)
    else :
        base_score = "0"

    if model.learning_rate is not None:
        learning_rate = str(model.learning_rate)
    else:
```

```
        learning_rate = "1"
    learning_rate, base_score


    N= model.n_estimators
    XGBOOST_CODE = code_trees(N, learning_rate)
    TREES_CODE = TreesCode(xgb_model)

    Nv, dimX= X.shape
    Nv, dimX= str(Nv), str(dimX)
    Xs=array_to_arduino(X.flatten())
    ys=array_to_arduino(y)

    codeInit= codeInit.replace("NvReplace",Nv)
    codeInit= codeInit.replace("dimXReplace",dimX)
    codeInit= codeInit.replace("Xreplace",Xs)
    codeInit= codeInit.replace("yreplace",ys)
    codeInit= codeInit.replace("base_score_Replace",base_score)
    codeInit= codeInit.replace("learning_rate_Replace",learning_rate)
    codeInit= codeInit.replace("TREES_CODE_replace", TREES_CODE)
    codeInit= codeInit.replace("XGBOOST_CODE_replace", XGBOOST_CODE)
    return codeInit
```

In [252…

```
arduino_code = XGBOOST_to_CPP(xgb_model, sub_X, sub_y, base_score)
```

In [253…

```
ino_file="../ArduinoCode/Xgboost_Model2.ino"
ino_file=ino_file.replace(".ino", "")
```

In [254…

```
current_directory = os.getcwd()
new_directory_path = os.path.join(current_directory, ino_file)
try:
    os.makedirs(new_directory_path)
except: pass

path=ino_file+"/"+ino_file.split("/")[-1]+".ino"
with open(path,'w+') as f:
    f.write(arduino_code)

    print(path, "saved")
```

```
../ArduinoCode/Xgboost_Model2/Xgboost_Model2.ino saved
```

The arduino memory usnig

Sketch uses 28940 bytes (94%) of program storage space. Maximum is 30720 bytes. Global variables use 252 bytes (12%) of dynamic memory, leaving 1796 bytes for local variables. Maximum is 2048 bytes.

In [30]:

```
# The arduino serial print result
serialPrint="""
Cal_Ardui,Expected,Delta_time(us)
19879.69,19879.697265,2076
24714.27,24714.275390,1984
16844.20,16844.208984,1952
19767.67,19767.681640,1956
20115.12,20115.128906,1960
25036.53,25036.535156,1964
34841.58,34841.546875,1988
29939.02,29939.027343,1980
25749.97,25749.972656,1960
28292.76,28292.769531,1968
====The End====="""
```

In [31]:

```
# Convert the serial result to DF
data = serialPrint.split("\n")[1:-1]
data=[x.split(",") for x in data]
DF_serial= pd.DataFrame( data[1:], columns= data[0]).astype("float32")
DF_serial
```

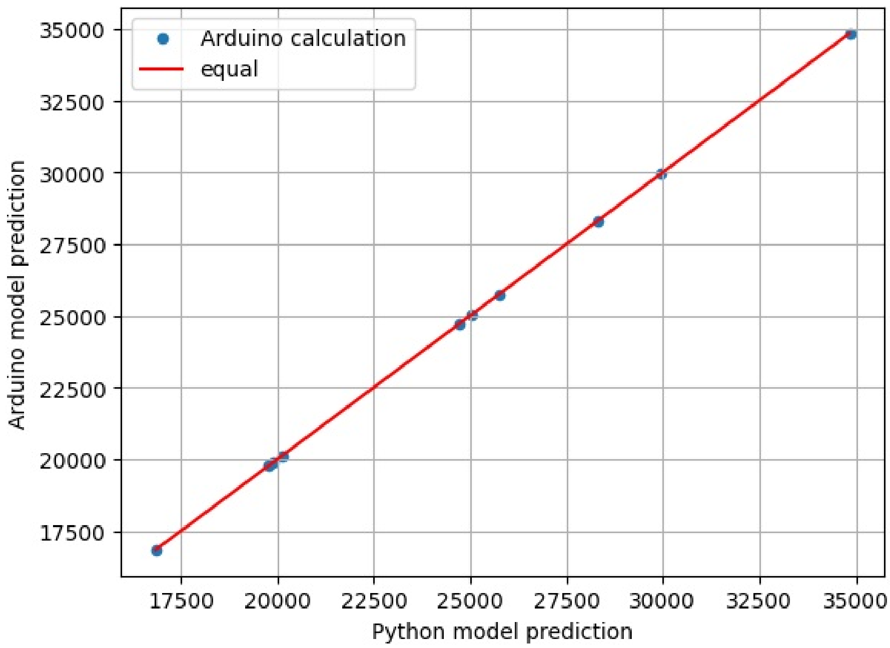
Out[31]:

	Cal_Ardui	Expected	Delta_time(us)
0	19879.689453	19879.697266	2076.0
1	24714.269531	24714.275391	1984.0
2	16844.199219	16844.208984	1952.0
3	19767.669922	19767.681641	1956.0
4	20115.119141	20115.128906	1960.0
5	25036.529297	25036.535156	1964.0
6	34841.578125	34841.546875	1988.0
7	29939.019531	29939.027344	1980.0
8	25749.970703	25749.972656	1960.0
9	28292.759766	28292.769531	1968.0

In [32]:
```
print("The AVG prediction time of one input is",
      (DF_serial['Delta_time(us)'].mean()/1000).round(2),
      "ms"
     )
```

```
The AVG prediction time of one input is 1.98 ms
```

In [33]:
```
# Ploting
DF_serial.plot.scatter(x='Expected', y='Cal_Ardui',  marker='o', label="Arduino calculation")
xx=[DF_serial['Expected'].min(), DF_serial['Expected'].max()]
plt.plot(xx,xx, c='r', label="equal")
plt.legend()
plt.xlabel("Python model prediction")
plt.ylabel("Arduino model prediction")
plt.grid()
plt.show()
```



DNN

Conversion of tensorflow / keras model to C++ (arduino language)

In [263…
```
# Load the model
tf_model=tf.keras.models.load_model('../models/DNN/tf_model.keras')
```

In [265…
```
sub_X=X_train[:10]
sub_y=tf_model.predict(X, verbose = 0)
```

In [269…
```
def tf_model_to_arduino_code(inp_model, sub_X, sub_y, code):
    """
    This function converts a trained TensorFlow model into an Arduino-compatible code
    for forward propagation. The model's weights, biases, and activation functions
    are extracted, and Arduino code is generated to represent the model for use on
    an embedded system.

    Inputs:
    - inp_model: Trained TensorFlow model (Keras model) whose layers and weights
                 will be used for forward propagation.
```

```
    - sub_X: Input data (not used in the function directly, but likely required
             for the context or future extension).
    - sub_y: Output data (not used directly, similar to `sub_X`).
    - code: Template code (as a string) that will be modified and returned,
            with model weights, biases, and activation functions.

    Outputs:
    - code2: Arduino code with initialized model weights, biases, and forward
             propagation logic embedded.
    """


    init_code="""
#include <math.h>
#include <Arduino.h>
#include <avr/pgmspace.h> // Include the PROGMEM functions

INIT_1

// Activation function//////////////
float sigmoid (float x){
    return 1./(1.+exp(-x));
}

float relu (float x){
    return max(x,0.);
}

float tanh_ (float x){
// make difference between tanh of C++ and tanh_ the activation func
    return tanh(x);
}

float linear(float x){
    return x;
}
///// You can add other activation function ////

void print_arr(float arr[], int N) {
    Serial.print("[");
    for (int i = 0; i < N; i++) {
        Serial.print(arr[i],4);
        if (i < N-1) {
            Serial.print(",");
            }
    }
    Serial.print("]");
}


void propagation(const float *WTf,  float *VEC, const float *B,float *out,  int M, int N, float (*act_func)(floa

  // Perform matrix-vector multiplication and activation
  for (int i = 0; i < M; ++i) {
    out[i] = pgm_read_float_near(&B[i]);
    for (int j = 0; j < N; ++j) {
      out[i] += pgm_read_float_near(&WTf[i * N + j]) * VEC[j];
    }
    out[i] = act_func(out[i]);
  }
}

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

void loop() {
unsigned long timestart;
unsigned long timeend;
float Xi[dimX];
INIT_2

Serial.println("Cal_Ardui,Expected,Delta_time(us)");
for (int l=0;l<Nv;l++){
for(int j = 0; j<dimX;j++){
    Xi[j]=pgm_read_float_near(&X[l*dimX+j]);
}
LOOP_
for (int k=0;k<M__final;k++){
Serial.print(OUTPUT__final[k],6);
Serial.print(" , ");
Serial.print(pgm_read_float_near(&y[l]),6);
Serial.print(" , ");
```

```
Serial.println(timeend-timestart);
}
}
Serial.println("====The End=====");
while(1);
}
"""
    WTfs = []  # List to store flattened weight matrices for each layer
    Bs = []  # List to store bias vectors for each layer
    acts = []  # List to store activation functions for each layer
    INIT = ""  # String to hold the initialization section of Arduino code

    # Loop through each layer of the model
    for i, layer in enumerate(inp_model.layers):
        W, B = layer.get_weights()  # Get weights and biases for the current layer
        WTf = W.T.flatten()  # Flatten the weight matrix and store it
        actfun = layer.activation.__name__  # Get the activation function name
        WTfs.append(WTf)  # Append flattened weights to the list
        Bs.append(B)  # Append biases to the list
        acts.append(actfun)  # Append activation function name to the list
        print("Layer", i, "W shape", W.shape, "Bias shape", B.shape, "Activation Function", actfun)

    # Define dimensions of weight matrix W
    M, N = W.T.shape

    # Get shape of the input data X (not used directly in the function)
    xshape = X.shape
    NvdimX = "const int Nv = " + str(xshape[0]) + ";\nconst int dimX = " + str(xshape[1]) + ";\n"

    # Convert X and y to Arduino-compatible format and store as strings
    Xystr = "\n////// Xy ////// \nconst float X [] PROGMEM  = " + array_to_arduino(X.flatten()) + ";\n\n" + \
            "const float y[] PROGMEM  = " + array_to_arduino(y.flatten()) + " ;\n\n"

    initstr = ""  # String to hold initialization section for each layer

    # Loop through each layer again to generate initialization strings for weights and biases
    for i, layer in enumerate(inp_model.layers):
        W, B = layer.get_weights()  # Get weights and biases for the current layer
        M, N = W.T.shape  # Get dimensions of the weight matrix
        WTf = W.T.flatten()  # Flatten the weights

        # Prepare the Arduino code initialization for this layer
        Mstr = "const int M" + str(i) + " = " + str(M) + " ;"
        Nstr = "const int N" + str(i) + " = " + str(N) + " ;"
        WTfstr = "const float WTf" + str(i) + "[] PROGMEM  = " + str(WTf.tolist()).replace("[", "{").replace("]
        Bstr = "const float BIAS" + str(i) + "[] PROGMEM= " + str(B.tolist()).replace("[", "{").replace("]", "}
        Outstr = "float OUTPUT" + str(i) + "[" + str(M) + "] ;"
        layerstr = "// Layer" + str(i) + " init \n" + Nstr + "\n" + Mstr + "\n" + WTfstr + "\n" + Bstr + "\n" +

        # Append the layer initialization to the overall initialization string
        initstr += layerstr + "\n\n"

    # Define the forward propagation logic in Arduino code
    prostr = "\n///////// Forward Propagation ///////////\ntimestart=micros();\n"
    funcstr = "propagation(WTf_, VEC, BIAS_, OUTPUT_, M_, N_,  activation); // Layer_\n"

    # Generate forward propagation code for each layer
    for i, layer in enumerate(inp_model.layers):
        W, B = layer.get_weights()  # Get weights and biases
        M, N = W.T.shape  # Get dimensions of the weight matrix
        WTf = W.T.flatten()  # Flatten the weights
        actfunc = layer.activation.__name__  # Get activation function name
        actfunc = actfunc.replace('tanh', 'tanh_')  # Replace 'tanh' with 'tanh_' for Arduino compatibility
        prostr += funcstr.replace("_", str(i)) \
            .replace('activation', actfunc) \
            .replace("VEC", "OUTPUT" + str(i - 1)) \
            .replace("OUTPUT-1", "Xi")

    # Final Arduino code section
    prostr += "timeend=micros();"

    # Replace placeholders in the code template with the generated code
    code2 = code.replace("INIT_1", NvdimX + initstr + Xystr)
    code2 = code2.replace("INIT_2", "")
    code2 = code2.replace("LOOP_", prostr)
    code2 = code2.replace("__final", str(i))  # Replace the final placeholder with the last layer index

    return code2  # Return the generated Arduino code
```

In [270…

```
arduino_code=tf_model_to_arduino_code(tf_model, X, y, init_code)
```

```
Layer 0 W shape (4, 16) Bias shape (16,) Activation Function relu
Layer 1 W shape (16, 8) Bias shape (8,) Activation Function relu
Layer 2 W shape (8, 4) Bias shape (4,) Activation Function relu
Layer 3 W shape (4, 1) Bias shape (1,) Activation Function linear
```

In [271…

```
# save the arduino code
ino_file="../ArduinoCode/Tf_Model"
ino_file=ino_file.replace(".ino" ,"")

current_directory = os.getcwd()
new_directory_path = os.path.join(current_directory, ino_file)
try:
    os.makedirs(new_directory_path)
except: pass

path=ino_file+"/"+ino_file.split("/")[-1]+".ino"
with open(path,'w+') as f:
    f.write(arduino_code)
    print(path, "saved")
```

```
../ArduinoCode/Tf_Model/Tf_Model.ino saved
```

The arduino memory usnig

Sketch uses 5048 bytes (16%) of program storage space. Maximum is 30720 bytes.
Global variables use 370 bytes (18%) of dynamic memory, leaving 1678 bytes for local variables. Maximum is 2048 bytes.

In [34]:

```
# The arduino serial print result
serialPrint="""
Cal_Ardui,Expected,Delta_time(us)
17770.185546 , 17770.187500 , 4556
22208.960937 , 22208.962890 , 4420
16064.545898 , 16064.555664 , 4444
19372.082031 , 19372.083984 , 4412
19566.919921 , 19566.931640 , 4436
28578.988281 , 28578.988281 , 4528
33195.054687 , 33195.054687 , 4512
24006.271484 , 24006.269531 , 4508
22752.718750 , 22752.728515 , 4520
23988.726562 , 23988.726562 , 4536
====The End====="""
```

In [35]:

```
# Convert the serial result to DF
data = serialPrint.split("\n")[1:-1]
data=[x.split(",") for x in data]
DF_serial= pd.DataFrame( data[1:], columns= data[0]).astype("float32")
DF_serial
```

Out[35]:

	Cal_Ardui	Expected	Delta_time(us)
0	17770.185547	17770.187500	4556.0
1	22208.960938	22208.962891	4420.0
2	16064.545898	16064.555664	4444.0
3	19372.082031	19372.083984	4412.0
4	19566.919922	19566.931641	4436.0
5	28578.988281	28578.988281	4528.0
6	33195.054688	33195.054688	4512.0
7	24006.271484	24006.269531	4508.0
8	22752.718750	22752.728516	4520.0
9	23988.726562	23988.726562	4536.0

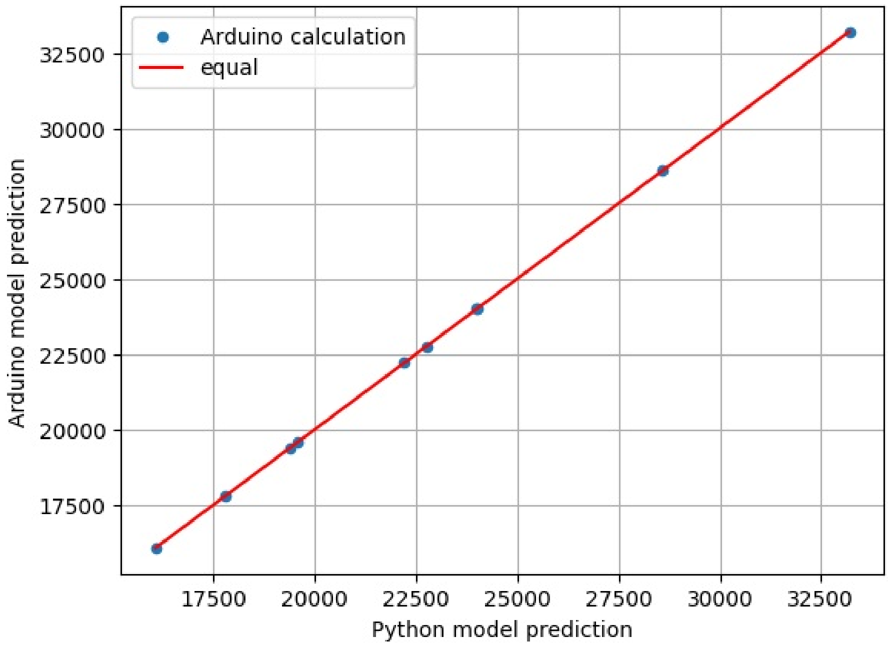
In [36]:

```
print("The AVG prediction time of one input is",
      (DF_serial['Delta_time(us)'].mean()/1000).round(2),
      "ms"
     )
```

```
The AVG prediction time of one input is 4.49 ms
```

In [37]:

```
# Ploting
DF_serial.plot.scatter(x='Expected', y='Cal_Ardui',  marker='o', label="Arduino calculation")
xx=[DF_serial['Expected'].min(), DF_serial['Expected'].max()]
plt.plot(xx,xx, c='r', label="equal")
plt.legend()
plt.xlabel("Python model prediction")
plt.ylabel("Arduino model prediction")
plt.grid()
plt.show()
```



Make a PyPi package of all this project function

See the package link
https://pypi.org/project/mltoarduino

Annexes

Other solution for XGBoost

Conversion of Xgboost model to C++ (arduino language)

In [39]:
```
# Load the model
xgb_model = joblib.load(r'../models/xgboost/xgb_model.pkl')
```

In [40]:
```
sub_X=X_train[:10]
sub_y=xgb_model.predict(sub_X)
```

In [44]:
```
X=sub_X
y=sub_y
```

In [46]:
```
# Get the base score
base_score = xgb_model.base_score
print("Base Score:", base_score)
```

```
Base Score: 18476.805
```

In [47]:
```
booster=xgb_model.get_booster()
print(booster.get_dump(dump_format='text')[0])
```

```
0:[f1<129] yes=1,no=2,missing=2
	1:[f1<91] yes=3,no=4,missing=4
		3:[f2<6] yes=7,no=8,missing=8
			7:leaf=-905.217285
			8:leaf=-422.910553
		4:[f3<47435] yes=9,no=10,missing=10
			9:leaf=442.025391
			10:leaf=-22.4555092
	2:[f1<177] yes=5,no=6,missing=6
		5:[f3<84687] yes=11,no=12,missing=12
			11:leaf=948.198547
			12:leaf=342.00119
		6:[f3<136640] yes=13,no=14,missing=14
			13:leaf=1401.38916
			14:leaf=803.22821
```

for i, x in enumerate (booster.get_dump(dump_format='text')): pass def txt_c_nodes(tree_string): out="" # Define the pattern for extracting the desired parts pattern = r'(\d+):

$$(\backslash w+)([<>=]+)(-?[\backslash d.]+)$$

\s+yes=(\d+),no=(\d+),missing=(\d+)' lines=tree_string.replace("\t","").split('\n') for l in lines: if "[" in l and "]" in l: # Use re.findall to extract matching groups matches = re.findall(pattern, l) # Extracted parts if matches: for match in matches: #condition, yes, no, missing = match node,feature, cond, value, yes, no, missing = match index=feature.replace("f","") cond=cond.replace("=","==") out+="node"+node+": if (X ["+index+"] "+cond+value+") goto node"+ yes+" ; else goto node"+no+" ; \n" else: if 'leaf=' in l : #print(l) node=l.split(':leaf=')[0] leaf=l.split(':leaf=')[1] out+='node'+node+': return ' + leaf+" ;\n" return out print(txt_c_nodes(x))

In [48]:
```
l1=list(dfnew.columns)
l2=list(range(len(l1)))
dic={x1:'f'+str(x2) for (x1,x2) in zip(l1,l2)}
dic
```

Out[48]:
```
{'Gearbox_auto': 'f0',
 'HorseP': 'f1',
 'Euro_stand': 'f2',
 'km': 'f3',
 'price': 'f4'}
```

In [49]:
```
l1=list(dic.keys())
# TO AVOID MISTAKE IN 'evHvBatteryEnergyLevel_lag' AND 'evHvBatteryEnergyLevel',
l1.sort()
l1=l1[::-1]
d=dict()
for x in l1:
    d[x]=dic[x]
dic=d
dic
```

Out[49]:
```
{'price': 'f4',
 'km': 'f3',
 'HorseP': 'f1',
 'Gearbox_auto': 'f0',
 'Euro_stand': 'f2'}
```

In [50]:
```
print(booster.get_dump(dump_format='text')[0])
```

```
0:[f1<129] yes=1,no=2,missing=2
	1:[f1<91] yes=3,no=4,missing=4
		3:[f2<6] yes=7,no=8,missing=8
			7:leaf=-905.217285
			8:leaf=-422.910553
		4:[f3<47435] yes=9,no=10,missing=10
			9:leaf=442.025391
			10:leaf=-22.4555092
	2:[f1<177] yes=5,no=6,missing=6
		5:[f3<84687] yes=11,no=12,missing=12
			11:leaf=948.198547
			12:leaf=342.00119
		6:[f3<136640] yes=13,no=14,missing=14
			13:leaf=1401.38916
			14:leaf=803.22821
```

In [51]:
```
txt=booster.get_dump(dump_format='text')[0]

for k in dic.keys():
    txt= txt.replace(k, dic[k])

print(txt)
```

```
0:[f1<129] yes=1,no=2,missing=2
	1:[f1<91] yes=3,no=4,missing=4
		3:[f2<6] yes=7,no=8,missing=8
			7:leaf=-905.217285
			8:leaf=-422.910553
		4:[f3<47435] yes=9,no=10,missing=10
			9:leaf=442.025391
			10:leaf=-22.4555092
	2:[f1<177] yes=5,no=6,missing=6
		5:[f3<84687] yes=11,no=12,missing=12
			11:leaf=948.198547
			12:leaf=342.00119
		6:[f3<136640] yes=13,no=14,missing=14
			13:leaf=1401.38916
			14:leaf=803.22821
```

In [53]:
```
def txt_c_nodes2(tree_string,dic):
    out=""
    # Define the pattern for extracting the desired parts
    pattern = r'(\d+):\[(\w+)([<>=]+)(-?[\d.]+)\]\s+yes=(\d+),no=(\d+),missing=(\d+)'
    for k in dic.keys():
        tree_string= tree_string.replace(k, dic[k])
    #print(tree_string)
    lines=tree_string.replace("\t","").split('\n')
    for l in lines:
```

```
        if "[" in l and "]" in l:
            # Use re.findall to extract matching groups
            matches = re.findall(pattern, l)
            # Extracted parts
            if matches:
                for match in matches:
                    #condition, yes, no, missing = match
                    node,feature, cond, value, yes, no, missing = match
                    index=feature.replace("f","")
                    cond=cond.replace("=","==")
                    out+="node"+node+": if (X ["+index+"] "+cond+value+") goto node"+ yes+" ; else goto node"+n
        else:
            if 'leaf=' in l :
                #print(l)
                node=l.split(':leaf=')[0]
                leaf=l.split(':leaf=')[1]
                out+='node'+node+': return ' + leaf+" ;\n"
    return out
print(txt_c_nodes2(x,dic))
```

```
node0: if (X [1] <132) goto node1 ; else goto node2 ;
node1: if (X [1] <129) goto node3 ; else goto node4 ;
node3: if (X [2] <5) goto node7 ; else goto node8 ;
node7: return 67.2528915 ;
node8: return -4.22500849 ;
node4: if (X [3] <55480) goto node9 ; else goto node10 ;
node9: return 125.454178 ;
node10: return -0.75947547 ;
node2: if (X [1] <150) goto node5 ; else goto node6 ;
node5: if (X [3] <176320) goto node11 ; else goto node12 ;
node11: return -50.4777794 ;
node12: return 346.403595 ;
node6: if (X [3] <23600) goto node13 ; else goto node14 ;
node13: return -66.9801254 ;
node14: return 12.834815 ;
```

In [57]:
```
def trees_to_C2(booster,dic):
    code=""
    for i, x in enumerate (booster.get_dump(dump_format='text')):
        code += "///////////////// TREE_"+str(i+1)
        code += "\n"
        code += "float tree"+str(i)+" ( float X[] ) {"
        code += "\n"
        code += txt_c_nodes2(x,dic)
        code += "}"
        code += "\n"
    return code
```

In [58]:
```
base_score=X_train.mean()
base_score
```

Out[58]: 18476.805

In [60]:
```
sub_X=X_train[:10]
sub_y=xgb_model.predict(sub_X)

X=sub_X
y=sub_y
```

In [61]:
```
base_score
```

Out[61]: 18476.805

In [62]:
```
def all_arduino_code4(model,X,y, dic, base_score):
    Xs=str(list(X)).replace('[','{').replace(']','}')
    booster=model.get_booster()

    code ="""
    INIT_1
    """
    xshape = X.shape
    NvdimX = "const int Nv = " + str(xshape[0]) +\
    ";\nconst int dimX = " + str(xshape[1]) + ";\n"

    Xystr = "\n/////// Xy ////// \nconst float X [] PROGMEM  = " +\
    array_to_arduino(X.flatten()) + ";\n\n" + \
    "const float y[] PROGMEM  = " + \
    array_to_arduino(y.flatten()) + " ;\n\n"


    code=code.replace("INIT_1", NvdimX + Xystr)
    #print(code)
```

```python
    if model.base_score is not None:
        code += "float base_score = " + str(model.base_score)+" ;"
    else:
        code += "float base_score =  " + str(base_score)+" ;"
    code += "\n"
    if model.learning_rate is not None:
        code += "float learning_rate = "+ str(model.learning_rate)+" ;"
    else:
        code += "float learning_rate = 1 ;"
    code += "\n"
    #code += "float X[]= "+Xs+" ;"
    code += "\n"
    code +=  trees_to_C2(booster, dic)
    code += '////////////////// XGBpredict'
    code += "\n"
    code +='float XGBpred(float X[]){'
    code += "\n"
    code +='float out = 0;'
    code += "\n"
    for i, x in enumerate (booster.get_dump(dump_format='text')):
        code +="out= tree"+str(i)+"(X)+out;"
        code += "\n"
    code += "\n"
    #code += "out = out*learning_rate+base_score;"
    code += "out = out+base_score;"
    code += "\n"
    code +="return out;}"
    code += "\n"
    code += "\n"
    code += """void setup() {
    Serial.begin(115200);
}

void loop() {
unsigned long timestart;
unsigned long timeend;
float Xi[dimX];
float yc;


Serial.println("Cal_Ardui,Expected,Delta_time(us)");
for (int l=0;l<Nv;l++){
for(int j = 0; j<dimX;j++){
    Xi[j]=pgm_read_float_near(&X[l*dimX+j]);
}
timestart=micros();
yc=XGBpred(Xi);
timeend=micros();
Serial.print(yc);
Serial.print(",");
Serial.print(pgm_read_float_near(&y[l]),6);
Serial.print(",");
Serial.println(timeend-timestart);
}
Serial.println("====The End=====");
while(1);
}
"""

    return code
```

In [ ]:

In [63]:
```python
base_score=y_train.mean()
base_score
```

Out[63]: 22734.895

In [65]:
```python
X=X_test[0]
arduino_code=all_arduino_code4(xgb_model,sub_X, \
                    sub_y, dic,base_score)
```

In [297…
```python
ino_file="../ArduinoCode/Xgboost_Model"
```

In [298…
```python
current_directory = os.getcwd()
new_directory_path = os.path.join(current_directory, ino_file)
try:
    os.makedirs(new_directory_path)
```

```python
except: pass

path=ino_file+"/"+ino_file.split("/")[-1]+".ino"
with open(path,'w+') as f:
    f.write(arduino_code)

    print(path, "saved")
```

```
../ArduinoCode/Xgboost_Model/Xgboost_Model.ino saved
```

Sketch uses 28784 bytes (93%) of program storage space. Maximum is 30720 bytes. Global variables use 252 bytes (12%) of dynamic memory, leaving 1796 bytes for local variables. Maximum is 2048 bytes.

In [38]:
```python
serialPrint="""
Cal_Ardui,Expected,Delta_time(us)
20222.40,20222.404296,2088
24689.11,24689.107421,1976
16387.31,16387.304687,1952
19894.63,19894.634765,1940
20383.30,20383.296875,1960
25561.64,25561.638671,1964
34748.62,34748.605468,2000
29990.77,29990.771484,1980
26085.88,26085.880859,1948
28298.82,28298.818359,1956
====The End====="""
```

In [39]:
```python
data = serialPrint.split("\n")[1:-1]
data=[x.split(",") for x in data]
DF_serial= pd.DataFrame( data[1:], columns= data[0]).astype("float32")
DF_serial
```

Out[39]:

	Cal_Ardui	Expected	Delta_time(us)
0	20222.400391	20222.404297	2088.0
1	24689.109375	24689.107422	1976.0
2	16387.310547	16387.304688	1952.0
3	19894.630859	19894.634766	1940.0
4	20383.300781	20383.296875	1960.0
5	25561.640625	25561.638672	1964.0
6	34748.621094	34748.605469	2000.0
7	29990.769531	29990.771484	1980.0
8	26085.880859	26085.880859	1948.0
9	28298.820312	28298.818359	1956.0

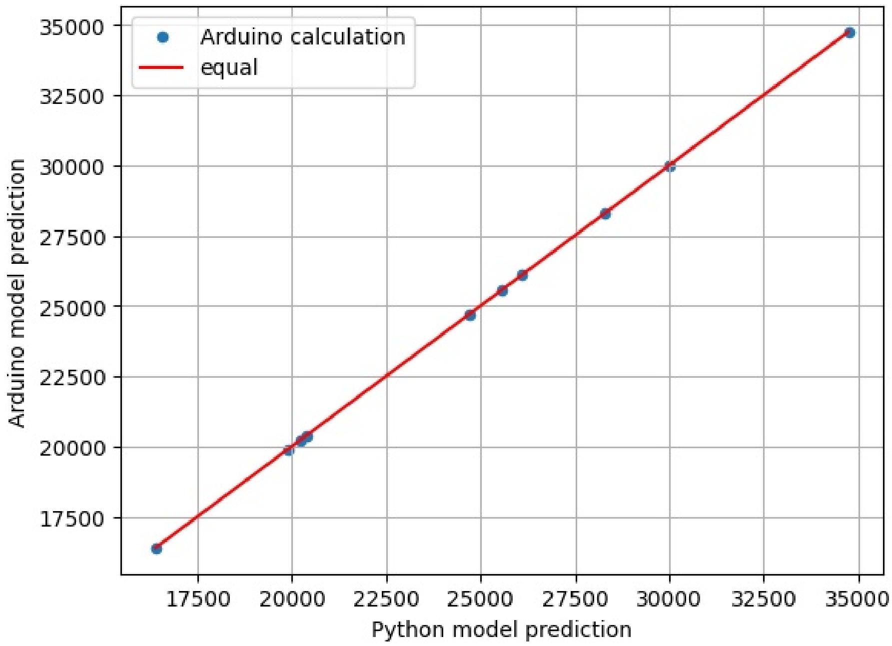
In [40]:
```python
DF_serial.columns
```

Out[40]:
```
Index(['Cal_Ardui', 'Expected', 'Delta_time(us)'], dtype='object')
```

In [41]:
```python
print("The AVG prediction time of one input is",
      (DF_serial['Delta_time(us)'].mean()/1000).round(2),
      "ms"
     )
```

```
The AVG prediction time of one input is 1.98 ms
```

In [42]:
```python
DF_serial.plot.scatter(x='Expected', y='Cal_Ardui',  marker='o', label="Arduino calculation")
xx=[DF_serial['Expected'].min(), DF_serial['Expected'].max()]
plt.plot(xx,xx, c='r', label="equal")
plt.legend()
plt.xlabel("Python model prediction")
plt.ylabel("Arduino model prediction")
plt.grid()
plt.show()
```



Processing math: 100%