

APPENDIX F: MACHINE LEARNING PIPELINE EXAMPLE

Build a machine learning pipeline in Mage.AI to train a model on the Titanic dataset.

In this example, we'll create a pipeline that does the following:

1. Load data from an online endpoint
2. Select columns and fill in missing values
3. Train a model to predict which passengers will survive

1. Setup

1a. Add Python packages to project

In the left sidebar (aka file browser), click on the requirements.txt file under the {"demo_project"}/ folder.

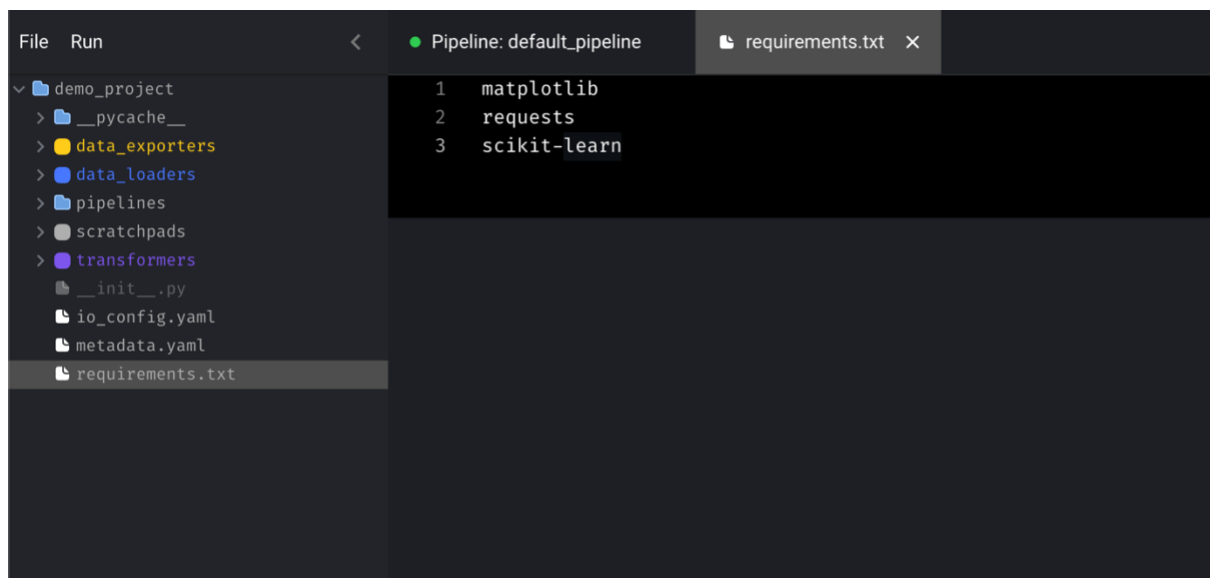


FIGURE 7 PYTHON REQUIREMENTS

Then add the following dependencies to that file:

- matplotlib
- requests
- scikit-learn

Then, save the file by pressing ⌘ + S, or winkey + S.

2a. Install dependencies

The simplest way is to run pip install from the tool.

Add a scratchpad block by pressing the + Scratchpad button. Then run the following command:

pip install -r demo_project/requirements.txt (run this in the terminal which is located on the right sidebar).

2. Create new pipeline

In the top left corner, click File > New pipeline. Then, click the name of the pipeline next to the green dot to rename it to titanic survivors.

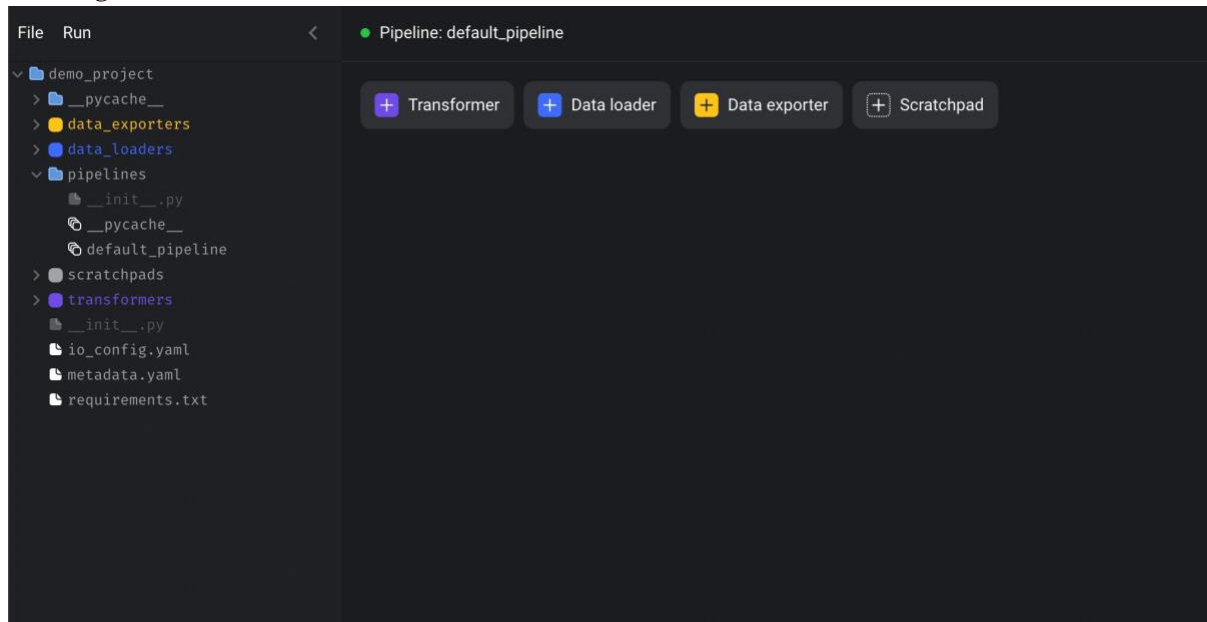


FIGURE 8 GIF OF PIPELINE CREATION

3. Play around with scratchpad

There are 4 buttons, click on the + Scratchpad button to add a block.

Paste the following sample code in the block:

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2*np.pi*t)
plt.plot(t, s)

plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('About as simple as it gets, folks')
plt.grid(True)
plt.show()
```

Then click the Play button on the right side of the block to run the code. Alternatively, you can use the following keyboard shortcuts to execute code in the block:

- ⌘ + Enter
- Control + Enter
- Shift + Enter (run code and add a new block)
-

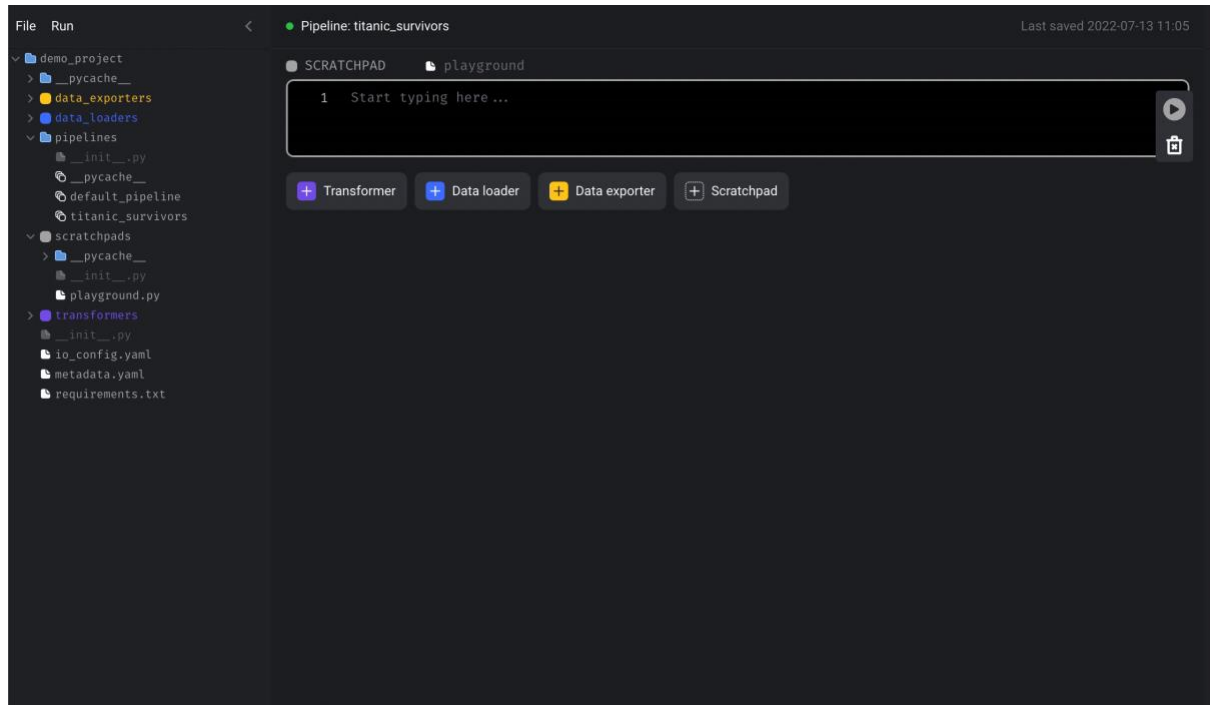


FIGURE 9 GIF OF RUNNING A CODE BLOCK

Now that we're done with the scratchpad, we can leave it there or delete it. To delete a block, click the trash can icon on the right side or use the keyboard shortcut by typing the letter D and then D again.

4. Load data

1. Click the + Data loader button, select Python, then click the template called API.
2. Rename the block to load dataset.
3. In the function named `load_data_from_api`, set the url variable to: <https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv>
4. Run the block by clicking the play icon button or using the keyboard shortcuts ⌘ + Enter, Control + Enter, or Shift + Enter.

After you run the block (⌘ + Enter), you can immediately see a sample of the data in the blocks output.

Here is what the code should look like:

```
import io
import pandas as pd
import requests
from pandas import DataFrame

if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test

@data_loader
def load_data_from_api(**kwargs) -> DataFrame:
    """
    Template for loading data from API
    """
    url='https://raw.githubusercontent.com/mageai/datasets/master/titanic_survival.csv'

    response = requests.get(url)
    return pd.read_csv(io.StringIO(response.text), sep=',')

@test
def test_output(df) -> None:
    """
    Template code for testing the output of the block.
    """
    assert df is not None, 'The output is undefined'
```

5. Transform data

We're going to select numerical columns from the original dataset, then fill in missing values for those columns (aka impute).

1. Click the + Transformer button, select Python, then click Generic (no template).
2. Rename the block to extract and impute numbers.
3. Paste the following code in the block:

```
from pandas import DataFrame
import math

if 'transformer' not in globals():
    from mage_ai.data_preparation.decorators import transformer

def select_number_columns(df: DataFrame) -> DataFrame:
    return df[['Age', 'Fare', 'Parch', 'Pclass', 'SibSp', 'Survived']]
```

```
def fill_missing_values_with_median(df: DataFrame) -> DataFrame:
    for col in df.columns:
        values = sorted(df[col].dropna().tolist())
        median_age = values[math.floor(len(values) / 2)]
        df[[col]] = df[[col]].fillna(median_age)
    return df

@transformer
def transform_df(df: DataFrame, *args) -> DataFrame:
    return fill_missing_values_with_median(select_number_columns(df))
```

After you run the block (⌘ + Enter), you can immediately see a sample of the data in the blocks output.

6. Train model

In this part, we're going to accomplish the following:

1. Split the dataset into a training set and a test set.
2. Train logistic regression model.
3. Calculate the models accuracy score.
4. Save the training set, test set, and model artifact to disk.

Here are the steps to take:

1. Add a new data exporter block by clicking + Data exporter button, select Python, then click Generic (no template).
2. Rename the block to train model.
3. Paste the following code in the block:

```
from pandas import DataFrame
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import os
import pickle

if 'data_exporter' not in globals():
    from mage_ai.data_preparation.decorators import data_exporter

LABEL_COLUMN = 'Survived'

def build_training_and_test_set(df: DataFrame) -> None:
    X = df.drop(columns=[LABEL_COLUMN])
    y = df[LABEL_COLUMN]

    return train_test_split(X, y)
```

```
def train_model(X, y) -> None:
    model = LogisticRegression()
    model.fit(X, y)

    return model

def score_model(model, X, y) -> None:
    y_pred = model.predict(X)

    return accuracy_score(y, y_pred)

@data_exporter
def export_data(df: DataFrame) -> None:
    X_train, X_test, y_train, y_test = build_training_and_test_set(df)
    model = train_model(X_train, y_train)

    score = score_model(model, X_test, y_test)
    print(f'Accuracy: {score}')

    cwd = os.getcwd()
    filename = f'{cwd}/finalized_model.lib'
    print(f'Saving model to {filename}')
    pickle.dump(model, open(filename, 'wb'))

    print(f'Saving training and test set')
    X_train.to_csv(f'{cwd}/X_train')
    X_test.to_csv(f'{cwd}/X_test')
    y_train.to_csv(f'{cwd}/y_train')
    y_test.to_csv(f'{cwd}/y_test')
```

Run the block (⌘ + Enter).

7. Run pipeline

We can now run the entire pipeline end-to-end from the UI. Click on the **Execute pipeline** in the right bottom panel.

APPENDIX G: PROTOTYPE EXPLAINED

To validate the architectural design a prototype was build, below each component of the prototype will be explained with some screenshots.

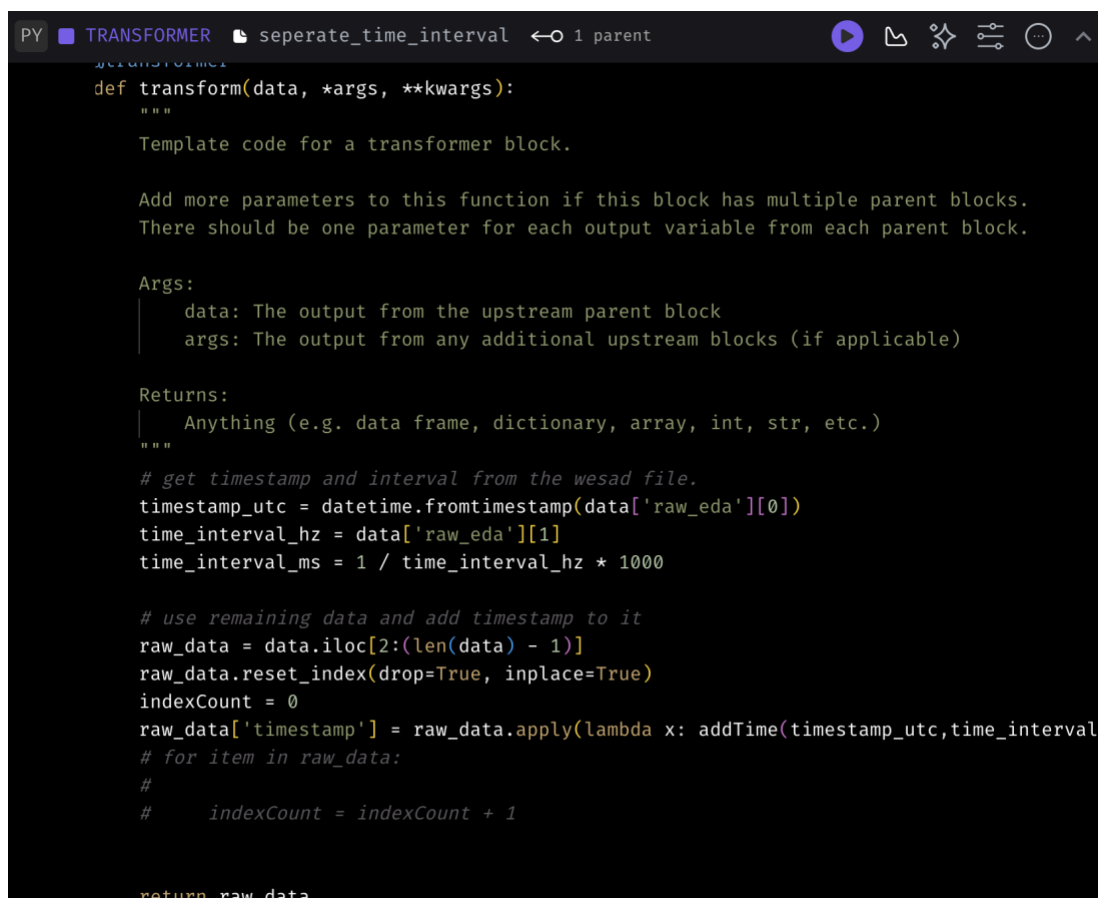
DATA INTAKE WITH MAGE.AI

Firstly intaking data, in the prototype multiple pipelines where setup (see *Figure 10 Mage.AI pipelines*), 2 of these pipelines where data intake pipelines. Both data intake pipelines loaded data from a .CSV file (easy to do, completely local).

After which the data intake pipelines transformed the given loaded data to one useful for processing. (see *Figure 11 data transformation*).

Status	Name	Description	Type
no schedules	marks_model		Standard
no schedules	simple_data_forward		Standard
no schedules	wesad_through_model		Standard
▶ inactive	wesad_to_db	wesad csv files to database.	Standard

FIGURE 10 MAGE.AI PIPELINES



```

PY TRANSFORMER separate_time_interval ← 1 parent
def transform(data, *args, **kwargs):
    """
    Template code for a transformer block.

    Add more parameters to this function if this block has multiple parent blocks.
    There should be one parameter for each output variable from each parent block.

    Args:
        data: The output from the upstream parent block
        args: The output from any additional upstream blocks (if applicable)

    Returns:
        Anything (e.g. data frame, dictionary, array, int, str, etc.)
    """
    # get timestamp and interval from the wesad file.
    timestamp_utc = datetime.fromtimestamp(data['raw_eda'][0])
    time_interval_hz = data['raw_eda'][1]
    time_interval_ms = 1 / time_interval_hz * 1000

    # use remaining data and add timestamp to it
    raw_data = data.iloc[2:(len(data) - 1)]
    raw_data.reset_index(drop=True, inplace=True)
    indexCount = 0
    raw_data['timestamp'] = raw_data.apply(lambda x: addTime(timestamp_utc, time_interval_ms, x['index']), axis=1)
    # for item in raw_data:
    #     indexCount = indexCount + 1

    return raw_data

```

FIGURE 11 DATA TRANSFORMATION

Then a second transformation was done to add chunk numbers (can be used later when inputting the data into the ML model).

Lastly, the data was exported to the back-end as a service database. These steps result in the completion of a data intake pipeline (see *Figure 12 Data pipeline - overview tree*).

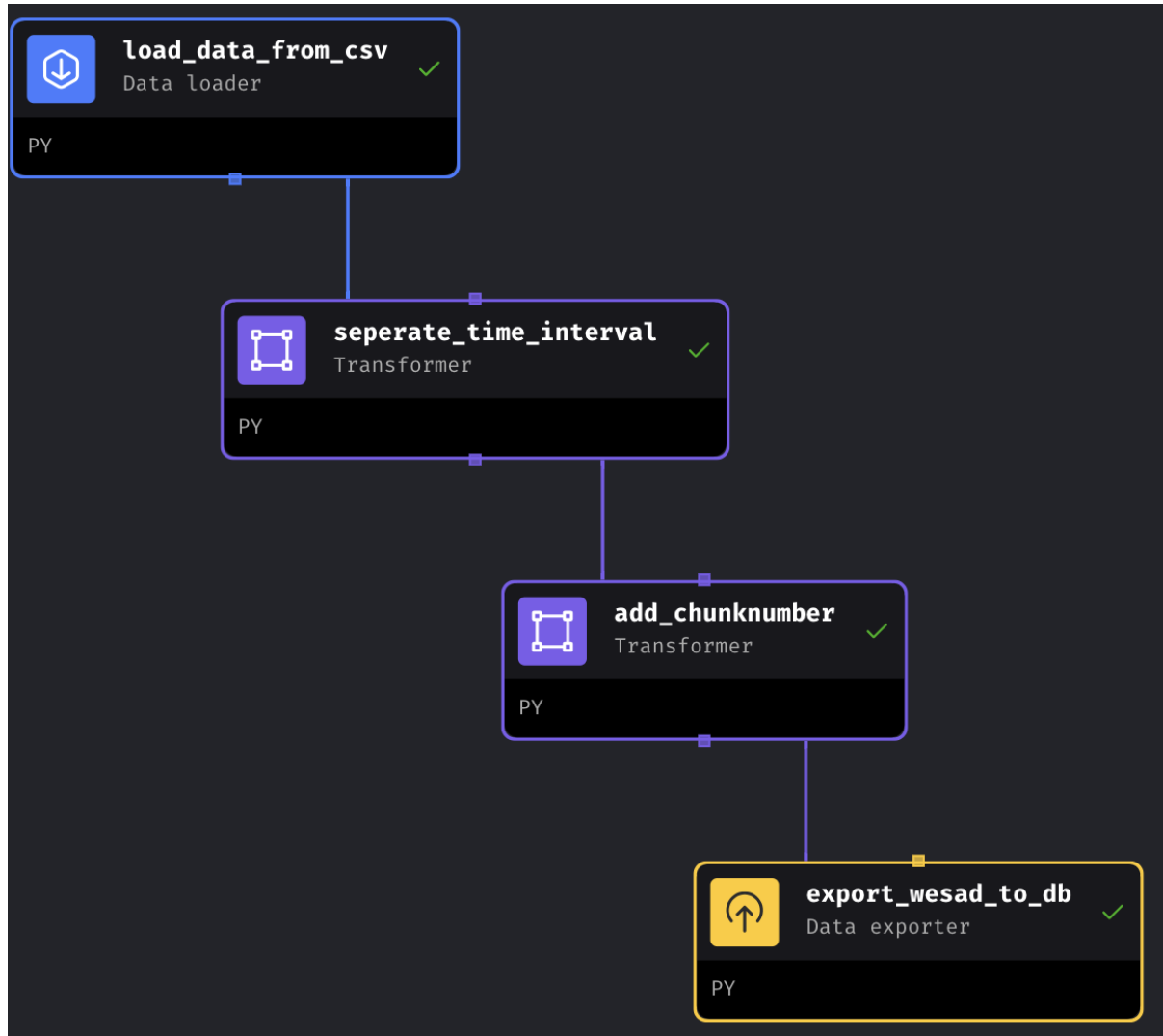


FIGURE 12 DATA PIPELINE - OVERVIEW TREE

STRESS DETECTION WITH A ML MODEL USING MAGE.AI

For ease of use and reproducibility the raw data from the csv was loaded from the database (see *Figure 13 Load from database script*).

Then that data is passed onto the next code block, which is a transform block containing a Keras model. The Keras model is loaded from disk and then used on the data to predict stress and tag the chunks. (see *Figure 14 using a ML model in Mage.AI*)

To complete the pipeline the data is again exported to the back-end as a service database.

```
query = 'select * from raw_mark_csv' # Specify your SQL query here
config_path = path.join(get_repo_path(), 'io_config.yaml')
config_profile = 'default'

with Postgres.with_config(ConfigFileLoader(config_path, config_profile)) as loader:
    return loader.load(query)
```

FIGURE 13 LOAD FROM DATABASE SCRIPT

```
import keras
from keras.models import load_model
import os
import numpy as np
from sklearn.preprocessing import StandardScaler
from keras.preprocessing.sequence import pad_sequences

model = load_model(os.getcwd() + '/etl-demo/mark_model/best_model.keras')
scaler = StandardScaler() # Initialize StandardScaler for preprocessing

# Preprocess function for input data
def preprocess_input(data):
    max_sequence_length = 128
    num_chunks = len(data) // max_sequence_length + 1
    predictions = []
    for i in range(num_chunks):
        start_idx = i * max_sequence_length
        end_idx = min((i + 1) * max_sequence_length, len(data))
        chunk = data[start_idx:end_idx]
        padded_chunk = pad_sequences([chunk], maxlen=max_sequence_length, dtype='float32', padding='post', trunc
        preprocessed_chunk = np.asarray(padded_chunk).astype(np.float32).reshape(-1, max_sequence_length, 1)
        predictions.append(model.predict(preprocessed_chunk))
    return predictions

def your_label_mapping_function(index):
    # No stress = false, stress = true
    labels = [False, True] # Assuming these are your original labels
    return labels[index]
```

FIGURE 14 USING A ML MODEL IN MAGE.AI

SERVING STRESS DETECTION DATA TO THE END USERS

The final part of the prototype revolved around showcasing the flexibility of the back-end as service. As a demonstration a low code UI tool (Budibase) was used to create a small stress graph UI. The low code UI connected to the auto generated REST API of Supabase to request the needed data from the database.

It should be noted that the data setup might not be ideal, but for prototype purposes it showcases the architectures capabilities well.

	PrimaryKey uuid	_time float8	w_eda float8	chunknu... i...
	21fa8761-0450-4dd9-8dcc-f78d25cb3222	0	1.138257	1
	2c6a5ba6-57f9-4068-ac4d-d14849bf670f	0.25	1.125444	1
	8bca8a41-b86c-40b0-87a3-bfa59f1a039c	0.5	1.011405	1
	de8c2900-4c90-4fee-ad66-91ce578c465	0.75	1.033188	1
	0a932c66-a8e9-47de-966f-a1e3824facd5	1	0.935807	1
	10af7f8c-5f04-47a4-8141-4426823a1752	1.25	0.935708	1
	05d14712-66ab-4f43-865f-f03f0f18e6cb	1.5	0.962616	1
	f9fc959b-9f15-4390-93af-84da468dab61	1.75	0.934426	1
	ff9db23d-bc9a-4b5f-ace9-56cb8aa5fcb8	2	0.967741	1
	fd3408f7-c2b6-40b9-a89d-3126f3a4106e	2.25	1.094593	1
	18dabdeb-c34b-4200-b4c9-64767ab265e	2.5	1.175317	1

FIGURE 15 DATABASE TABLE OF PRE-PROCESSED DATA

	PrimKey uuid	chunknu... i...	predicted_label bool
	e6727ffe-3acb-4133-a133-d74355595dee	1	FALSE
	34ebb7f7-ebbd-4c15-b821-327bc3c3e6ee	2	FALSE
	c7b21e92-98a5-4522-b77c-5fcf0de10ac4	3	FALSE
	973f8e05-6b72-4ac8-ae5b-6f10464957ce	4	FALSE
	45f15f72-c03e-4c7f-a61d-21e7e5e38f1f	5	FALSE
	223b4373-281f-4477-ad63-7025532968be	6	FALSE
	d665d4b3-a61a-4d33-bbf4-3c17bd679713	7	FALSE
	2677f388-a384-4594-a761-ac9f7cbdab3f	8	FALSE
	994d7628-3246-44ba-98a4-03cca96794f	9	FALSE
	c5a53059-0bca-4f1d-ade7-34f445cb8a5e	10	FALSE
	9e0cec6e-7a14-44f1-a3d2-0b3b82c5018e	11	FALSE

FIGURE 16 DATABASE TABLE OF PROCESSED DATA

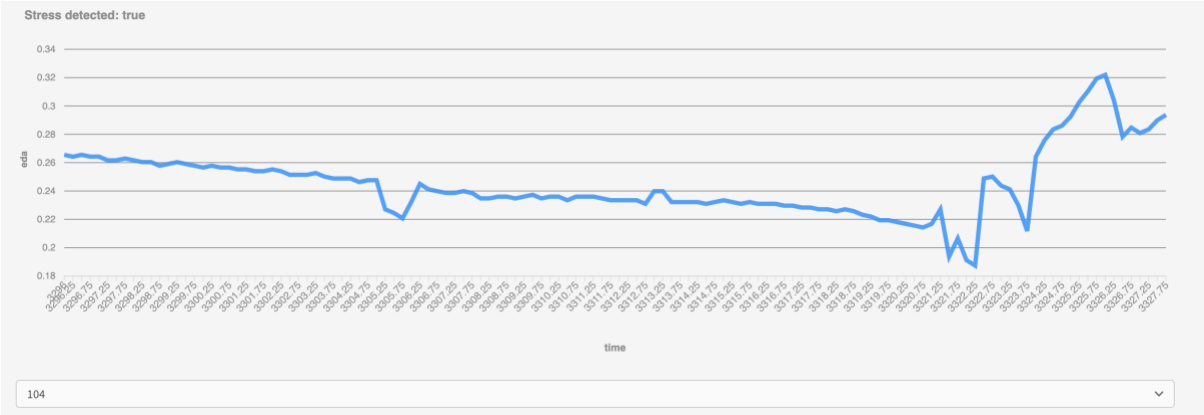


FIGURE 17 LOWCODE GRAPHS - STRESS

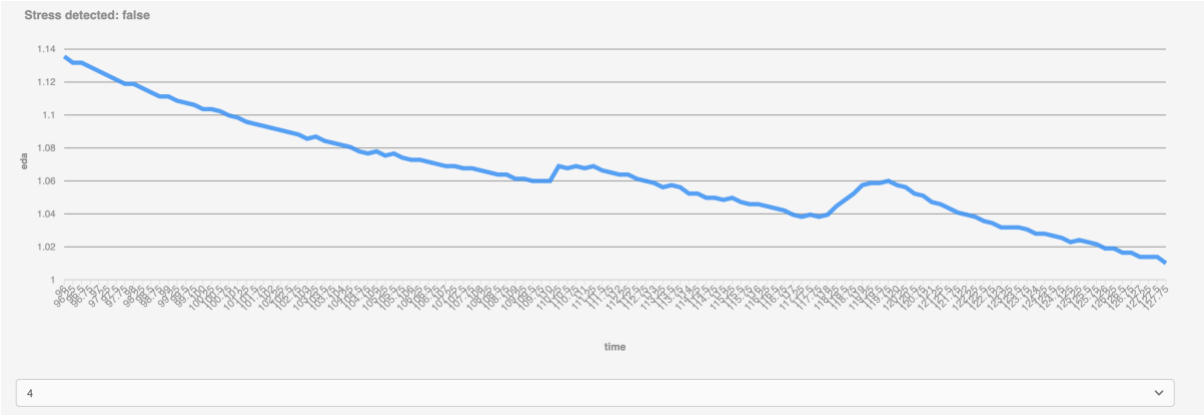


FIGURE 18 LOWCODE GRAPHS - NO STRESS