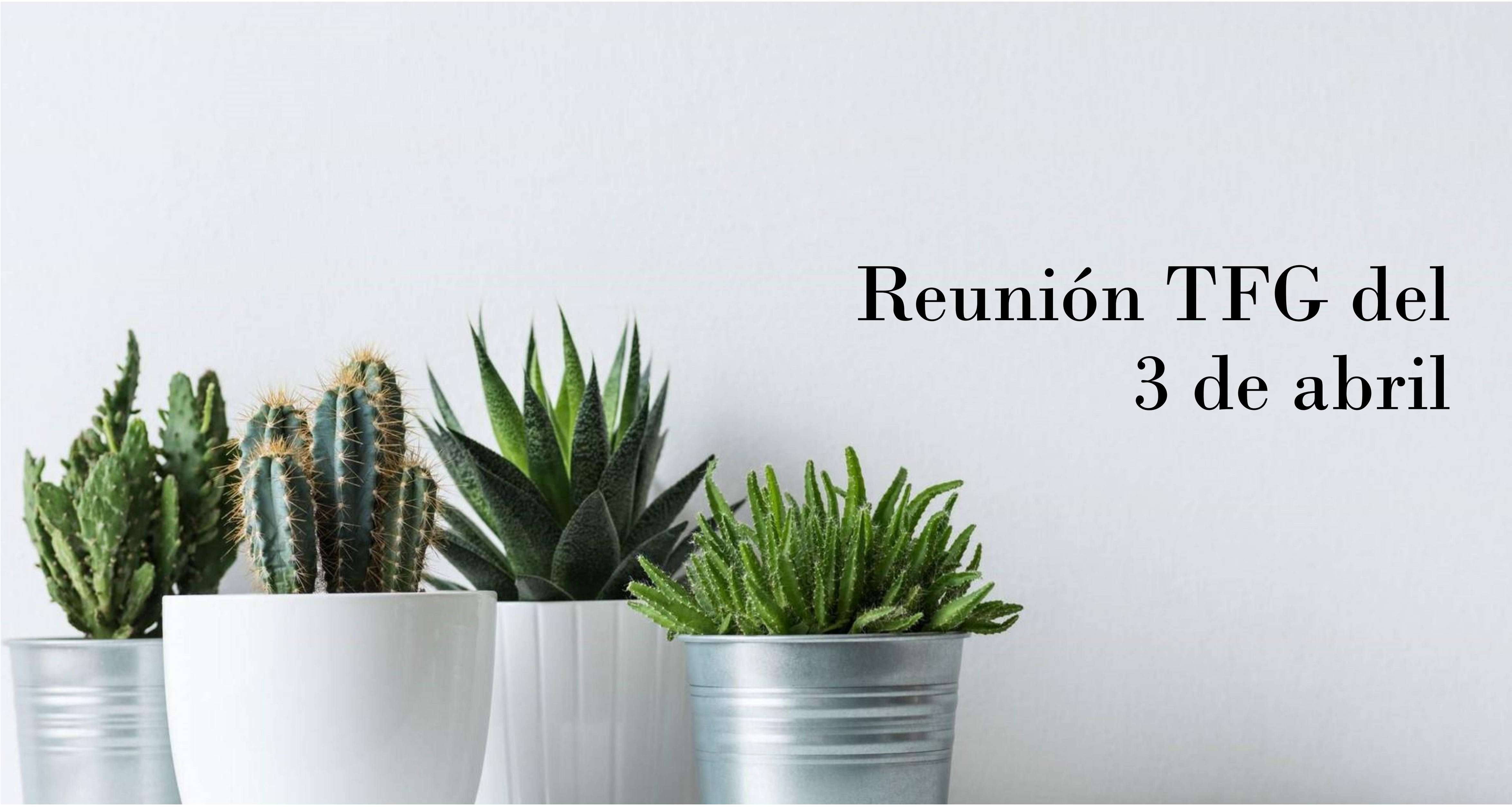
Reunión TFG del 3 de abril



Modelos que quiero desarrollar:

- ❑ Transformer (Univariate) with downloaded dataset -> Para aprender
- ❑ Transformer (Multivariate) with our dataset but univariate
- ❑ Transformer (Multivariate) with downloaded dataset -> Para aprender
- ❑ Transformer (Multivariate) with our dataset
- ❑ Informer (Multivariate) with downloaded dataset -> Para aprender
- ❑ Informer (Multivariate) with our dataset
- ❑ Autofomer (Multivariate) with downloaded dataset -> Para aprender
- ❑ Autoformer (Multivariate) with our dataset

Dataset downloaded

- https://forecastingdata.org/
- https://huggingface.co/datasets/monash_tsf/viewer/tourism_monthly

A sample from the training set is provided below:

```
{
  'start': datetime.datetime(2012, 1, 1, 0, 0),
  'target': [14.0, 18.0, 21.0, 20.0, 22.0, 20.0, ...],
  'feat_static_cat': [0],
  'feat_dynamic_real': [[0.3, 0.4], [0.1, 0.6], ...],
  'item_id': '0'
}
```

Data Fields

For the univariate regular time series each series has the following keys:

- `start`: a datetime of the first entry of each time series in the dataset
- `target`: an array[float32] of the actual target values
- `feat_static_cat`: an array[uint64] which contains a categorical identifier of each time series in the dataset
- `feat_dynamic_real`: optional array of covariate features
- `item_id`: a string identifier of each time series in a dataset for reference

For the multivariate time series the `target` is a vector of the multivariate dimension for each time point.

Code Structure Transformer Univariate

main

load_and_preprocess_dataset

1. Load dataset
2. Split the data in train and test
3. Convert to pd.Period

define_my_model

1 - Lags_sequence
2 - Time_features
3 – Define config

```
# prediction length:
prediction_length=prediction_length,
# context length:
context_length=prediction_length * 2,
# lags coming from helper given the freq:
lags_sequence=lags_sequence,
# we'll add 2 time features ("month of year" and "age", see further):
num_time_features=len(time_features) + 1,
# we have a single static categorical feature, namely time series ID:
num_static_categorical_features=1,
# it has 366 possible values:
cardinality=[len(train_dataset)],
# the model will learn an embedding of size 2 for each of the 366 possible values:
embedding_dimension=[2],
# transformer params:
encoder_layers=4,
decoder_layers=4,
d_model=32,
```

Create Data Loaders

DataLoaders, allow us to have batches of (input, output) pairs - or in other words (past_values, future_values).

create_train_dataloader

1 - Definition of input names for prediction and training: Lists of field names are defined for prediction and training inputs.

2 - create_transformation

1) **Remove static/dynamic fields if not specified:** Fields are removed if the number of specified static or dynamic features is zero.
2) **Convert the data to NumPy:** Data is converted to NumPy arrays for static categorical and real features if they exist.
3) **Handle NaN values:** Indicators for observed values are added to handle NaN values in the target field.
4) **Add temporal features:** Features like month of the year are added based on the data frequency.
5) **Add an age feature**: A feature indicating the "age" of the time series is added, representing how long it has been since the series started.
6) **Vertically stack temporal features**: All temporal features are stacked vertically into a single field.
7) **Rename fields**: Fields are renamed to match HuggingFace library conventions.

3 - Data caching (optional): Transformed data is cached if specified.

4 - create_instance_splitter

5 - Training instance sampling: The instance splitter randomly samples training instances from transformed time series data.

6 - Batch creation: Batches of training data are created according to the model's input dimensions and batch size, potentially shuffling the batches.

create_backtest_dataloader

Train_model

1) **Accelerator initialization**
2) **Model transfer to device:** The model is moved to the appropriate device (GPU or CPU) using the to() method of the Accelerator.
3) **Optimizer initialization:** An AdamW optimizer is initialized with model parameters and specified hyperparameters.
4) **Model and optimizer preparation:** The model and optimizer are prepared for training using the **prepare() method** of the Accelerator.
5) **Training mode activation:** The model's training mode is activated using the **train()** method.
6) **Training loop:** The training loop iterates over the specified number of epochs:
   1) **Model output computation:** Model outputs are computed for the current batch using the **forward()** method.
   2) **Loss calculation:** The loss is calculated based on the model outputs.
   3) **Backpropagation and parameter update:** Backpropagation is performed to compute gradients, and model parameters are updated using the optimizer.

Evaluate Model

- forecasting
- see_metrics
- plot

Code Structure Transformer Multivariate

main

load_and_preprocess_dataset

1. Load dataset
2. Split the data in train and test
3. Convert to pd.Period

define_my_model

1 - Lags_sequence
2 - Time_features
3 – Define config

```
# prediction length:
prediction_length=prediction_length,
# context length:
context_length=prediction_length * 2,
# lags coming from helper given the freq:
lags_sequence=lags_sequence,
# we'll add 2 time features ("month of year" and "age", see further):
num_time_features=len(time_features) + 1,
# we have a single static categorical feature, namely time series ID:
num_static_categorical_features=1,
# it has 366 possible values:
cardinality=[len(train_dataset)],
# the model will learn an embedding of size 2 for each of the 366 possible values:
embedding_dimension=[2],
# transformer params:
encoder_layers=4,
decoder_layers=4,
d_model=32,
```

Create Data Loaders

DataLoaders, allow us to have batches of (input, output) pairs - or in other words (past_values, future_values).

create_train_dataloader

1 - Definition of input names for prediction and training: Lists of field names are defined for prediction and training inputs.

2 - create_transformation

1) **Remove static/dynamic fields if not specified:** Fields are removed if the number of specified static or dynamic features is zero.
2) **Convert the data to NumPy:** Data is converted to NumPy arrays for static categorical and real features if they exist.
3) **Handle NaN values:** Indicators for observed values are added to handle NaN values in the target field.
4) **Add temporal features:** Features like month of the year are added based on the data frequency.
5) **Add an age feature**: A feature indicating the "age" of the time series is added, representing how long it has been since the series started.
6) **Vertically stack temporal features**: All temporal features are stacked vertically into a single field.
7) **Rename fields**: Fields are renamed to match HuggingFace library conventions.

3 - Data caching (optional): Transformed data is cached if specified.

4 - create_instance_splitter

5 - Training instance sampling: The instance splitter randomly samples training instances from transformed time series data.

6 - Batch creation: Batches of training data are created according to the model's input dimensions and batch size, potentially shuffling the batches.

create_backtest_dataloader

Train_model

1) **Accelerator initialization**
2) **Model transfer to device:** The model is moved to the appropriate device (GPU or CPU) using the to() method of the Accelerator.
3) **Optimizer initialization:** An AdamW optimizer is initialized with model parameters and specified hyperparameters.
4) **Model and optimizer preparation:** The model and optimizer are prepared for training using the **prepare() method** of the Accelerator.
5) **Training mode activation:** The model's training mode is activated using the **train()** method.
6) **Training loop:** The training loop iterates over the specified number of epochs:
    1) **Model output computation:** Model outputs are computed for the current batch using the **forward()** method.
    2) **Loss calculation:** The loss is calculated based on the model outputs.
    3) **Backpropagation and parameter update:** Backpropagation is performed to compute gradients, and model parameters are updated using the optimizer.

Evaluate Model

- forecasting
- see_metrics
- plot