Projeto4_Final

June 20, 2024

1 Data Science Academy

2 Matemática Para Data Science

2.1 Projeto 4

2.1.1 Matemática da Arquitetura Transformer na Análise e Forecast de Séries Temporais

ATENÇÃO É TUDO QUE VOCÊ PRECISA!

https://arxiv.org/abs/1706.03762

2.2 Instalando e Carregando os Pacotes

[ ]:
```
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de␣
 ↪comando:
# pip install -U nome_pacote

# Para instalar a versão exata de um pacote, execute o comando abaixo no␣
 ↪terminal ou prompt de comando:
# !pip install nome_pacote==versão_desejada

# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.

# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste␣
 ↪jupyter notebook.
!pip install -q -U watermark
```

Visite o PyPi e pesquise sobre cada um dos pacotes abaixo:

https://pypi.org/

Usaremos o HuggingFace como fonte de dados e do modelo pré-treinado.

https://huggingface.co/

[ ]:
```
!pip install -q transformers==4.37.2
```

[ ]:
```
!pip install -q datasets==2.16.1
```

```
ERROR: pip's dependency resolver does not currently take into account all the
packages that are installed. This behaviour is the source of the following
dependency conflicts.
s3fs 2024.2.0 requires fsspec==2024.2.0, but you have fsspec 2023.10.0 which is
incompatible.
```

[ ]:
```
!pip install -q evaluate==0.4.1
```

[ ]:
```
!pip install -q accelerate==0.26.1
```

[ ]:
```
!pip install -q -U gluonts==0.14.4
```

[ ]:
```
!pip install -q ujson==5.4.0
```

[ ]:
```
!pip install -q urllib3==1.26.16
```

[ ]:
```
%env TF_CPP_MIN_LOG_LEVEL=3
```

```
env: TF_CPP_MIN_LOG_LEVEL=3
```

[ ]:
```
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
```

[ ]:
```
import os
os.environ['PYTORCH_ENABLE_MPS_FALLBACK'] = '1'
```

https://ts.gluon.ai

[ ]:
```
# Imports
import evaluate
import torch
import transformers
import accelerate
import gluonts
import pandas as pd
import numpy as np
from datasets import load_dataset
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from functools import lru_cache
from functools import partial
from transformers import TimeSeriesTransformerConfig,␣
 ↪TimeSeriesTransformerForPrediction
from transformers import PretrainedConfig
from typing import Optional
from accelerate import Accelerator
from torch.optim import AdamW
from evaluate import load
from typing import Iterable
```

```
from gluonts.itertools import Cached, Cyclic
from gluonts.dataset.loader import as_stacked_batches
from gluonts.time_feature import get_seasonality
from gluonts.time_feature import get_lags_for_frequency
from gluonts.time_feature import time_features_from_frequency_str
from gluonts.transform.sampler import InstanceSampler
from gluonts.time_feature import (time_features_from_frequency_str,␣
 ↪TimeFeature, get_lags_for_frequency)
from gluonts.dataset.field_names import FieldName
from gluonts.transform import (
    AddAgeFeature,
    AddObservedValuesIndicator,
    AddTimeFeatures,
    AsNumpyArray,
    Chain,
    ExpectedNumInstanceSampler,
    InstanceSplitter,
    RemoveFields,
    SelectFields,
    SetField,
    TestSplitSampler,
    Transformation,
    ValidationSplitSampler,
    VstackFeatures,
    RenameFields,
)
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
%reload_ext watermark
%watermark -a "Data Science Academy"
```

```
Author: Data Science Academy
```

2.3 Carregando os Dados de Séries Temporais

https://huggingface.co/datasets/monash_tsf

[ ]:
```
# Carrega o dataset
dsa_dataset = load_dataset("monash_tsf", "tourism_monthly")
```

```
Downloading builder script:   0%|          | 0.00/25.6k [00:00<?, ?B/s]

Downloading readme:   0%|          | 0.00/31.2k [00:00<?, ?B/s]

Downloading extra modules:   0%|          | 0.00/7.54k [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/200k [00:00<?, ?B/s]

Generating train split:   0%|          | 0/366 [00:00<?, ? examples/s]
```

```
Generating test split:   0%|          | 0/366 [00:00<?, ? examples/s]

Generating validation split:   0%|          | 0/366 [00:00<?, ? examples/s]
```

[ ]:
```
print(dsa_dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
'item_id'],
        num_rows: 366
    })
    test: Dataset({
        features: ['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
'item_id'],
        num_rows: 366
    })
    validation: Dataset({
        features: ['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
'item_id'],
        num_rows: 366
    })
})
```

[ ]:
```
# Dados de treino
dsa_dataset['train']
```

[ ]:
```
Dataset({
    features: ['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
'item_id'],
    num_rows: 366
})
```

2.4 Explorando os Dados de Séries Temporais

[ ]:
```
# Vamos extrair os dados no índice zero
exemplo_treino = dsa_dataset['train'][0]
```

[ ]:
```
type(exemplo_treino)
```

[ ]:
```
dict
```

[ ]:
```
exemplo_treino
```

[ ]:
```
{'start': datetime.datetime(1979, 1, 1, 0, 0),
 'target': [1149.8699951171875,
  1053.8001708984375,
  1388.8797607421875,
  1783.3702392578125,
```

1921.025146484375,
2704.94482421875,
4184.41357421875,
4148.35400390625,
2620.72509765625,
1650.300048828125,
1115.9200439453125,
1370.6251220703125,
1096.31494140625,
978.4600219726562,
1294.68505859375,
1480.465087890625,
1748.865234375,
2216.920166015625,
4690.5185546875,
4682.8642578125,
2459.579833984375,
1484.4901123046875,
1028.985107421875,
1109.3648681640625,
960.8751220703125,
896.35009765625,
1118.6551513671875,
1619.9949951171875,
1847.994873046875,
2367.044921875,
4991.16015625,
4772.9443359375,
2894.678466796875,
1860.4801025390625,
1185.150146484375,
1313.659912109375,
1160.9150390625,
1061.5048828125,
1301.77001953125,
1794.3797607421875,
2106.455078125,
2789.034912109375,
4917.8466796875,
4994.4833984375,
3016.754150390625,
1941.505126953125,
1234.135009765625,
1378.72021484375,
1182.9749755859375,
1081.6600341796875,
1424.110107421875,

1774.5350341796875,
2115.420166015625,
2804.840087890625,
4849.498046875,
4937.47509765625,
3074.2236328125,
2063.42529296875,
1297.355224609375,
1350.710205078125,
1224.360107421875,
1165.815185546875,
1409.3299560546875,
2116.5498046875,
2357.135009765625,
2995.0703125,
5295.2119140625,
4957.90478515625,
3321.959228515625,
2221.18017578125,
1345.9000244140625,
1514.01513671875,
1239.5501708984375,
1172.159912109375,
1518.9752197265625,
1996.8751220703125,
2248.68505859375,
3053.440185546875,
5019.45361328125,
5466.7802734375,
3235.167724609375,
2157.97998046875,
1379.7252197265625,
1728.0400390625,
1350.10986328125,
1216.014892578125,
1751.3251953125,
1805.320068359375,
2570.02490234375,
3204.240234375,
5395.72021484375,
6078.82861328125,
3587.098388671875,
2285.195068359375,
1582.18994140625,
1787.4298095703125,
1554.8701171875,
1409.8648681640625,

```
 1612.125,
 2286.239990234375,
 2913.755126953125,
 3645.908447265625,
 5956.70849609375,
 6326.97509765625,
 3914.66015625,
 2617.675048828125,
 1675.1650390625,
 2139.219970703125,
 1715.4898681640625,
 1663.5799560546875,
 2053.699951171875,
 2354.929931640625,
 3038.591796875,
 3470.609375,
 6606.18359375,
 6587.63671875,
 4133.78271484375,
 2960.0244140625,
 1762.5849609375,
 2125.64013671875,
 1815.9150390625,
 1632.31494140625,
 2210.39501953125,
 2210.215087890625,
 3099.269287109375,
 3468.77783203125,
 6482.92529296875,
 6665.48486328125,
 4006.36181640625,
 2882.3349609375,
 1775.2498779296875,
 2171.64990234375,
 1796.4749755859375,
 1692.349853515625,
 1949.78515625,
 2680.630126953125,
 2645.949951171875,
 3414.742919921875,
 5772.876953125],
'feat_static_cat': [0],
'feat_dynamic_real': None,
'item_id': 'T1'}
```

[ ]:
```
# Como é um dicionário, vamos extrair a chave
exemplo_treino.keys()
```

[ ]:
```
dict_keys(['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
'item_id'])
```

[ ]:
```
print(exemplo_treino['start'])
```

```
1979-01-01 00:00:00
```

[ ]:
```
print(exemplo_treino['target'])
```

```
[1149.8699951171875, 1053.8001708984375, 1388.8797607421875, 1783.3702392578125,
1921.025146484375, 2704.94482421875, 4184.41357421875, 4148.35400390625,
2620.72509765625, 1650.300048828125, 1115.9200439453125, 1370.6251220703125,
1096.31494140625, 978.4600219726562, 1294.68505859375, 1480.465087890625,
1748.865234375, 2216.920166015625, 4690.5185546875, 4682.8642578125,
2459.579833984375, 1484.4901123046875, 1028.985107421875, 1109.3648681640625,
960.8751220703125, 896.35009765625, 1118.6551513671875, 1619.9949951171875,
1847.994873046875, 2367.044921875, 4991.16015625, 4772.9443359375,
2894.678466796875, 1860.4801025390625, 1185.150146484375, 1313.659912109375,
1160.9150390625, 1061.5048828125, 1301.77001953125, 1794.3797607421875,
2106.455078125, 2789.034912109375, 4917.8466796875, 4994.4833984375,
3016.754150390625, 1941.505126953125, 1234.135009765625, 1378.72021484375,
1182.9749755859375, 1081.6600341796875, 1424.110107421875, 1774.5350341796875,
2115.420166015625, 2804.840087890625, 4849.498046875, 4937.47509765625,
3074.2236328125, 2063.42529296875, 1297.355224609375, 1350.710205078125,
1224.360107421875, 1165.815185546875, 1409.3299560546875, 2116.5498046875,
2357.135009765625, 2995.0703125, 5295.2119140625, 4957.90478515625,
3321.959228515625, 2221.18017578125, 1345.9000244140625, 1514.01513671875,
1239.5501708984375, 1172.159912109375, 1518.9752197265625, 1996.8751220703125,
2248.68505859375, 3053.440185546875, 5019.45361328125, 5466.7802734375,
3235.167724609375, 2157.97998046875, 1379.7252197265625, 1728.0400390625,
1350.10986328125, 1216.014892578125, 1751.3251953125, 1805.320068359375,
2570.02490234375, 3204.240234375, 5395.72021484375, 6078.82861328125,
3587.098388671875, 2285.195068359375, 1582.18994140625, 1787.4298095703125,
1554.8701171875, 1409.8648681640625, 1612.125, 2286.239990234375,
2913.755126953125, 3645.908447265625, 5956.70849609375, 6326.97509765625,
3914.66015625, 2617.675048828125, 1675.1650390625, 2139.219970703125,
1715.4898681640625, 1663.5799560546875, 2053.699951171875, 2354.929931640625,
3038.591796875, 3470.609375, 6606.18359375, 6587.63671875, 4133.78271484375,
2960.0244140625, 1762.5849609375, 2125.64013671875, 1815.9150390625,
1632.31494140625, 2210.39501953125, 2210.215087890625, 3099.269287109375,
3468.77783203125, 6482.92529296875, 6665.48486328125, 4006.36181640625,
2882.3349609375, 1775.2498779296875, 2171.64990234375, 1796.4749755859375,
1692.349853515625, 1949.78515625, 2680.630126953125, 2645.949951171875,
3414.742919921875, 5772.876953125]
```

[ ]:
```
# Vamos extrair um elemento de validação
exemplo_valid = dsa_dataset['validation'][0]
```

```
[ ]: exemplo_valid.keys()
```

```
[ ]: dict_keys(['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
     'item_id'])
```

```
[ ]: print(exemplo_valid['start'])
```

```
1979-01-01 00:00:00
```

```
[ ]: print(exemplo_valid['target'])
```

```
[1149.8699951171875, 1053.8001708984375, 1388.8797607421875, 1783.3702392578125,
1921.025146484375, 2704.94482421875, 4184.41357421875, 4148.35400390625,
2620.72509765625, 1650.300048828125, 1115.9200439453125, 1370.6251220703125,
1096.31494140625, 978.4600219726562, 1294.68505859375, 1480.465087890625,
1748.865234375, 2216.920166015625, 4690.5185546875, 4682.8642578125,
2459.579833984375, 1484.4901123046875, 1028.985107421875, 1109.3648681640625,
960.8751220703125, 896.35009765625, 1118.6551513671875, 1619.9949951171875,
1847.994873046875, 2367.044921875, 4991.16015625, 4772.9443359375,
2894.678466796875, 1860.4801025390625, 1185.150146484375, 1313.659912109375,
1160.9150390625, 1061.5048828125, 1301.77001953125, 1794.3797607421875,
2106.455078125, 2789.034912109375, 4917.8466796875, 4994.4833984375,
3016.754150390625, 1941.505126953125, 1234.135009765625, 1378.72021484375,
1182.9749755859375, 1081.6600341796875, 1424.110107421875, 1774.5350341796875,
2115.420166015625, 2804.840087890625, 4849.498046875, 4937.47509765625,
3074.2236328125, 2063.42529296875, 1297.355224609375, 1350.710205078125,
1224.360107421875, 1165.815185546875, 1409.3299560546875, 2116.5498046875,
2357.135009765625, 2995.0703125, 5295.2119140625, 4957.90478515625,
3321.959228515625, 2221.18017578125, 1345.9000244140625, 1514.01513671875,
1239.5501708984375, 1172.159912109375, 1518.9752197265625, 1996.8751220703125,
2248.68505859375, 3053.440185546875, 5019.45361328125, 5466.7802734375,
3235.167724609375, 2157.97998046875, 1379.7252197265625, 1728.0400390625,
1350.10986328125, 1216.014892578125, 1751.3251953125, 1805.320068359375,
2570.02490234375, 3204.240234375, 5395.72021484375, 6078.82861328125,
3587.098388671875, 2285.195068359375, 1582.18994140625, 1787.4298095703125,
1554.8701171875, 1409.8648681640625, 1612.125, 2286.239990234375,
2913.755126953125, 3645.908447265625, 5956.70849609375, 6326.97509765625,
3914.66015625, 2617.675048828125, 1675.1650390625, 2139.219970703125,
1715.4898681640625, 1663.5799560546875, 2053.699951171875, 2354.929931640625,
3038.591796875, 3470.609375, 6606.18359375, 6587.63671875, 4133.78271484375,
2960.0244140625, 1762.5849609375, 2125.64013671875, 1815.9150390625,
1632.31494140625, 2210.39501953125, 2210.215087890625, 3099.269287109375,
3468.77783203125, 6482.92529296875, 6665.48486328125, 4006.36181640625,
2882.3349609375, 1775.2498779296875, 2171.64990234375, 1796.4749755859375,
1692.349853515625, 1949.78515625, 2680.630126953125, 2645.949951171875,
3414.742919921875, 5772.876953125, 6053.7041015625, 3878.12841796875,
2806.514892578125, 1735.5382080078125, 2128.919921875, 1608.01416015625,
1441.330078125, 2068.235107421875, 2207.610107421875, 2918.409912109375,
```

```
3400.81787109375, 6048.7421875, 6483.14013671875, 4063.502685546875,
2900.22998046875, 1907.094970703125, 2338.510009765625, 1787.1650390625,
1699.6451416015625, 1979.105224609375, 2824.260009765625, 3076.5048828125,
3402.5849609375, 5985.830078125]
```

[ ]:
```
# Vamos extrair um elemento de teste
exemplo_test = dsa_dataset['test'][0]
```

[ ]:
```
print(exemplo_test['start'])
```

```
1979-01-01 00:00:00
```

[ ]:
```
print(exemplo_test['target'])
```

```
[1149.8699951171875, 1053.8001708984375, 1388.8797607421875, 1783.3702392578125,
1921.025146484375, 2704.94482421875, 4184.41357421875, 4148.35400390625,
2620.72509765625, 1650.300048828125, 1115.9200439453125, 1370.6251220703125,
1096.31494140625, 978.4600219726562, 1294.68505859375, 1480.465087890625,
1748.865234375, 2216.920166015625, 4690.5185546875, 4682.8642578125,
2459.579833984375, 1484.4901123046875, 1028.985107421875, 1109.3648681640625,
960.8751220703125, 896.35009765625, 1118.6551513671875, 1619.9949951171875,
1847.994873046875, 2367.044921875, 4991.16015625, 4772.9443359375,
2894.678466796875, 1860.4801025390625, 1185.150146484375, 1313.659912109375,
1160.9150390625, 1061.5048828125, 1301.77001953125, 1794.3797607421875,
2106.455078125, 2789.034912109375, 4917.8466796875, 4994.4833984375,
3016.754150390625, 1941.505126953125, 1234.135009765625, 1378.72021484375,
1182.9749755859375, 1081.6600341796875, 1424.110107421875, 1774.5350341796875,
2115.420166015625, 2804.840087890625, 4849.498046875, 4937.47509765625,
3074.2236328125, 2063.42529296875, 1297.355224609375, 1350.710205078125,
1224.360107421875, 1165.815185546875, 1409.3299560546875, 2116.5498046875,
2357.135009765625, 2995.0703125, 5295.2119140625, 4957.90478515625,
3321.959228515625, 2221.18017578125, 1345.9000244140625, 1514.01513671875,
1239.5501708984375, 1172.159912109375, 1518.9752197265625, 1996.8751220703125,
2248.68505859375, 3053.440185546875, 5019.45361328125, 5466.7802734375,
3235.167724609375, 2157.97998046875, 1379.7252197265625, 1728.0400390625,
1350.10986328125, 1216.014892578125, 1751.3251953125, 1805.320068359375,
2570.02490234375, 3204.240234375, 5395.72021484375, 6078.82861328125,
3587.098388671875, 2285.195068359375, 1582.18994140625, 1787.4298095703125,
1554.8701171875, 1409.8648681640625, 1612.125, 2286.239990234375,
2913.755126953125, 3645.908447265625, 5956.70849609375, 6326.97509765625,
3914.66015625, 2617.675048828125, 1675.1650390625, 2139.219970703125,
1715.4898681640625, 1663.5799560546875, 2053.699951171875, 2354.929931640625,
3038.591796875, 3470.609375, 6606.18359375, 6587.63671875, 4133.78271484375,
2960.0244140625, 1762.5849609375, 2125.64013671875, 1815.9150390625,
1632.31494140625, 2210.39501953125, 2210.215087890625, 3099.269287109375,
3468.77783203125, 6482.92529296875, 6665.48486328125, 4006.36181640625,
2882.3349609375, 1775.2498779296875, 2171.64990234375, 1796.4749755859375,
1692.349853515625, 1949.78515625, 2680.630126953125, 2645.949951171875,
3414.742919921875, 5772.876953125, 6053.7041015625, 3878.12841796875,
```

```
2806.514892578125, 1735.5382080078125, 2128.919921875, 1608.01416015625,
1441.330078125, 2068.235107421875, 2207.610107421875, 2918.409912109375,
3400.81787109375, 6048.7421875, 6483.14013671875, 4063.502685546875,
2900.22998046875, 1907.094970703125, 2338.510009765625, 1787.1650390625,
1699.6451416015625, 1979.105224609375, 2824.260009765625, 3076.5048828125,
3402.5849609375, 5985.830078125, 6611.115234375, 4150.2392578125, 2841.0,
1813.43994140625, 2261.080078125, 1873.60498046875, 1772.8399658203125,
2049.56494140625, 2932.264892578125, 3113.2548828125, 3461.5048828125,
6265.740234375, 6857.7998046875, 4346.08984375, 3154.72998046875,
2142.2099609375, 2375.72509765625, 1981.1099853515625, 1959.864990234375,
2466.31005859375, 2851.715087890625, 3671.804931640625, 3806.780029296875,
6995.0498046875]
```

[ ]:
```
len(exemplo_treino['target'])
```

[ ]:
```
139
```

[ ]:
```
len(exemplo_valid['target'])
```

[ ]:
```
163
```

[ ]:
```
len(exemplo_test['target'])
```

[ ]:
```
187
```

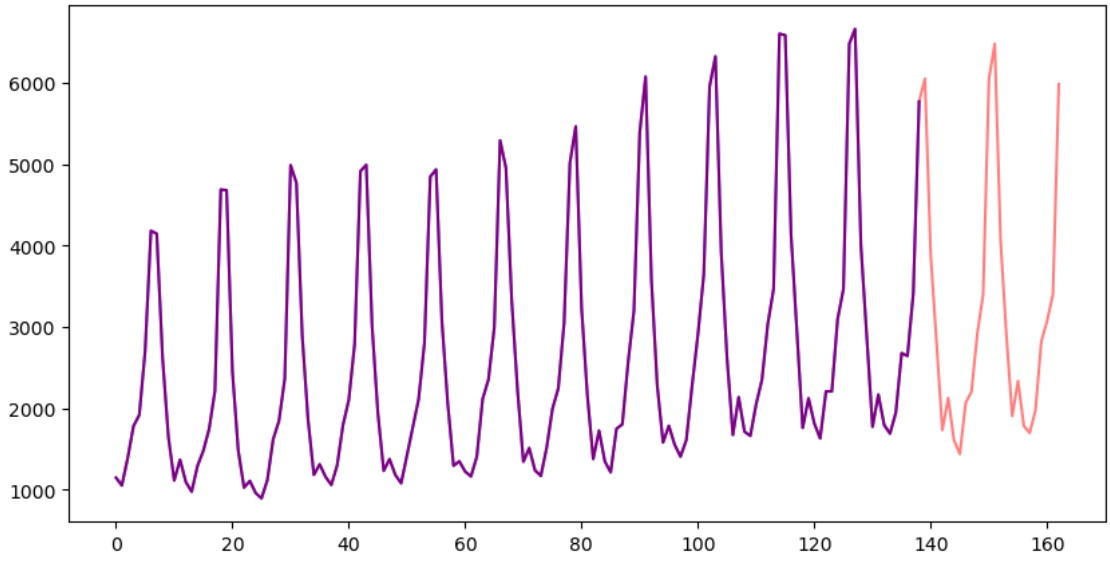
2.5 Visualizando a Série Temporal

[ ]:
```
# Frequência da série temporal (1 mês)
freq = "1M"
```

[ ]:
```
# Janela de previsão (24 meses)
prediction_length = 24
```

[ ]:
```
# Verifica se o comprimento dos dados de validação permite a janela de previsão
assert len(exemplo_treino["target"]) + prediction_length ==␣
 ↪len(exemplo_valid["target"])
```

[ ]:
```
# Plot
figure, axes = plt.subplots(figsize = (10, 5))
axes.plot(exemplo_treino["target"], color = "blue")
axes.plot(exemplo_valid["target"], color = "red", alpha = 0.5)
plt.show()
```



2.6 Convertendo o Formato de Data Para Períodos

[ ]:
```
# Datasets de treino e teste
dsa_dataset_treino = dsa_dataset["train"]
dsa_dataset_teste = dsa_dataset["test"]
```

A função abaixo é usada para converter objetos de data em períodos de datas usando pandas. Essa conversão é útil para trabalhar com séries temporais, pois os períodos podem representar melhor a granularidade dos dados (por exemplo, dia, mês, ano), dependendo da frequência (freq) fornecida. Esta função recebe uma data e a frequência desejada como argumentos, retornando um objeto pd.Period correspondente.

[ ]:
```
# Função para converter datas para períodos de datas (muda o formato da data)
def dsa_convert_to_pandas_period(date, freq):
    return pd.Period(date, freq)
```

A função abaixo é usada para ajustar o início de cada batch de dados. Ela aplica a função de conversão dsa_convert_to_pandas_period() a cada elemento na chave "start" do batch, transformando as datas de início em períodos pandas. Isso é feito para cada elemento do batch, garantindo que todos os pontos de dados tenham um marcador de início formatado corretamente de acordo com a frequência especificada. A função retorna o batch com os inícios ajustados.

[ ]:
```
# Função para definir o início do batch de dados
def dsa_define_start(batch, freq):
    batch["start"] = [dsa_convert_to_pandas_period(date, freq) for date in␣
 ↪batch["start"]]
    return batch
```

Finalmente, os datasets de treino e teste são ajustados para utilizar a função dsa_define_start() como uma transformação. Isso é feito usando o método set_transform() e aplicando a função

partial() para fixar o argumento freq na função dsa_define_start(), garantindo que a frequência especificada seja usada na conversão de todas as datas de início nos datasets. Essa etapa prepara os datasets para que estejam no formato adequado para as etapas subsequentes de modelagem, com todos os pontos de dados tendo um marcador de período de início consistente de acordo com a granularidade temporal desejada.

Ou seja, preparamos os dados no formato de sequência, exatamete o que um modelo com arquitetura Transformer espera receber.

[ ]:
```
# Ajusta os datasets das séries temporais no formato apropriado
dsa_dataset_treino.set_transform(partial(dsa_define_start, freq = freq))
dsa_dataset_teste.set_transform(partial(dsa_define_start, freq = freq))
```

[ ]:
```
len(dsa_dataset_treino)
```

[ ]:
```
366
```

[ ]:
```
dsa_dataset_treino
```

[ ]:
```
Dataset({
    features: ['start', 'target', 'feat_static_cat', 'feat_dynamic_real',
'item_id'],
    num_rows: 366
})
```

[ ]:
```
dsa_dataset_treino['start']
```

[ ]:
```
[Period('1979-01', 'M'),
 Period('1979-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1985-01', 'M'),
 Period('1986-01', 'M'),
 Period('1986-01', 'M'),
 Period('1986-01', 'M'),
 Period('1986-01', 'M'),
 Period('1980-01', 'M'),
 Period('1980-01', 'M'),
 Period('1980-01', 'M'),
```

```
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
```

```
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
```

```
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('2000-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
```

```
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1991-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
```

```
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
```

```
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
```

```
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
Period('1980-01', 'M'),
```

```
 Period('1980-01', 'M'),
 Period('1980-01', 'M'),
 Period('1980-01', 'M'),
 Period('1980-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M'),
 Period('1981-01', 'M')]
```

2.7 TimeSeries Transformer Config

Lembre-se de visitar sua amiga documentação periodicamente:

https://huggingface.co/docs/transformers/en/model_doc/time_series_transformer

[ ]:
```
# TimeSeries Transformer Config
dsa_config = TimeSeriesTransformerConfig(

    # Comprimento de previsão
    prediction_length = prediction_length,

    # Comprimento do contexto
    context_length = prediction_length * 2,

    # Lags sequence
    # "Lags" em séries temporais referem-se a pontos de dados anteriores em uma␣
 ↪série de tempo.
    # Em outras palavras, um "lag" é um atraso temporal. Por exemplo, em uma␣
 ↪série temporal mensal, o "lag"
    # de um mês refere-se aos dados do mês anterior.
    lags_sequence = get_lags_for_frequency(freq),

    # Adicionaremos 2 características de tempo ("mês do ano" e "idade da␣
 ↪série"):
    num_time_features = len(time_features_from_frequency_str(freq)) + 1,

    # Temos um único recurso categórico estático, ou seja, o ID da série␣
 ↪temporal
    num_static_categorical_features = 1,
```

```
    # Temos 366 valores possíveis
    cardinality = [len(dsa_dataset_treino)],

    # O modelo receberá uma embedding de tamanho 2 para cada um dos 366 valores␣
 ↪possíveis:
    embedding_dimension = [2],

    # Parâmetros da rede neural do Transformer
    encoder_layers = 4,
    decoder_layers = 4,
    d_model = 32,
)
```

2.8 TimeSeries Transformer For Prediction

Agora criamos o modelo (instância da classe).

[ ]:
```
# Cria o modelo com a config criada
modelo_dsa = TimeSeriesTransformerForPrediction(dsa_config)
```

Vamos compreender parte da Matemática por trás do Transformer, especificamente o módulo de atenção.

2.8.1 Matemática do Modelo Transformer

https://arxiv.org/pdf/1706.03762.pdf

Vamos descrever as partes do modelo Transformer com algumas fórmulas. Os termos a seguir serão úteis para entender os cálculos.

- Q: vetor de consulta
- K: vetor de chave
- V: vetor de valor

As fórmulas para o mecanismo de atenção no Transformer são as seguintes:

1- Atenção Escalada por Produto Escalar

A função de atenção é usada para calcular a importância de diferentes partes da entrada. Ela recebe três entradas: Q, K e V. A saída é calculada como:

Attention(Q, K, V) = softmax((QK^T) / sqrt(d_k))V

onde d_k é a dimensão dos vetores-chave e o operador ^T indica a transposição de uma matriz. A operação de produto escalar entre Q e K ajuda a determinar a relevância entre cada par de consulta e chave. O resultado é então dividido pela raiz quadrada de d_k para evitar que os valores do produto escalar fiquem muito grandes. Por fim, a função softmax é aplicada para transformar os pesos em probabilidades que somam 1. Esses pesos são então usados para ponderar os vetores de valor.

2- Atenção Multi-cabeça

A atenção multi-cabeça permite que o modelo se concentre em diferentes partes da entrada para cada cabeça de atenção. Suponha que temos h cabeças de atenção. Para cada cabeça, primeiro

transformamos Q, K e V com diferentes pesos aprendidos:

- Q_i = QW^Q_i
- K_i = KW^K_i
- V_i = VW^V_i

onde W^Q_i, W^K_i e W^V_i são os pesos aprendidos para a i-ésima cabeça.

Em seguida, aplicamos a atenção escalada por produto escalar para cada conjunto de Q_i, K_i e V_i:

head_i = Attention(Q_i, K_i, V_i)

A saída de todas as cabeças é então concatenada e linearmente transformada para produzir a saída final:

MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O

onde W^O é uma matriz de pesos aprendida.

3- Codificador e Decodificador

No codificador, cada camada consiste em atenção multi-cabeça seguida por uma rede neural feed-forward. A entrada passa pela atenção multi-cabeça e é então somada à entrada original (conexão residual) e normalizada. O resultado passa pela rede feed-forward, é somado à entrada e normalizado novamente.

No decodificador, temos uma camada adicional de atenção multi-cabeça que leva a saída do codificador como K e V. Isso permite que o decodificador leve em consideração a entrada inteira ao produzir cada token de saída.

2.9 Pré-Processamento dos Dados

[ ]:
```
# Função para criar a transformação (sequência de dados)
def dsa_cria_transformacao(freq: str, config: PretrainedConfig) ->␣
 ↪Transformation:

    remove_field_names = []

    if config.num_static_real_features == 0:
        remove_field_names.append(FieldName.FEAT_STATIC_REAL)

    if config.num_dynamic_real_features == 0:
        remove_field_names.append(FieldName.FEAT_DYNAMIC_REAL)

    if config.num_static_categorical_features == 0:
        remove_field_names.append(FieldName.FEAT_STATIC_CAT)

    return Chain(

        # Passo 1: Remove campos estáticos/dinâmicos se não for especificado
        [RemoveFields(field_names = remove_field_names)]
```

```
        # Passo 2: Converte os dados para formato NumPy
        + (
            [
                AsNumpyArray(field = FieldName.FEAT_STATIC_CAT, expected_ndim =␣
↪1, dtype = int)
            ]
            if config.num_static_categorical_features > 0
            else []
        )
        + (
            [
                AsNumpyArray(field = FieldName.FEAT_STATIC_REAL, expected_ndim␣
↪= 1)
            ]
            if config.num_static_real_features > 0
            else []
        )
        + [
            AsNumpyArray(field = FieldName.TARGET, expected_ndim = 1 if config.
↪input_size == 1 else 2,
            ),

            # Passo 3: Trata os NaN's preenchendo o alvo com zero e retornando␣
↪a máscara
            AddObservedValuesIndicator(target_field = FieldName.TARGET,␣
↪output_field = FieldName.OBSERVED_VALUES),

            # Passo 4: Adiciona recursos temporais com base na freq do mês do␣
↪ano do conjunto de dados
            # no caso em que freq="M" eles servem como codificações posicionais
            AddTimeFeatures(
                start_field = FieldName.START,
                target_field = FieldName.TARGET,
                output_field = FieldName.FEAT_TIME,
                time_features = time_features_from_frequency_str(freq),
                pred_length = config.prediction_length,
            ),

            # Passo 5: Adiciona outro recurso temporal (apenas um único número)
            # Informa ao modelo onde está o valor da série temporal, uma␣
↪espécie de contador em execução
            AddAgeFeature(
                target_field = FieldName.TARGET,
                output_field = FieldName.FEAT_AGE,
                pred_length = config.prediction_length,
```

```
                log_scale = True,
            ),

            # Passo 6: Empilha verticalmente todos os recursos temporais na␣
 ↪chave FEAT_TIME
            VstackFeatures(
                output_field = FieldName.FEAT_TIME,
                input_fields = [FieldName.FEAT_TIME, FieldName.FEAT_AGE]
                + (
                    [FieldName.FEAT_DYNAMIC_REAL]
                    if config.num_dynamic_real_features > 0
                    else []
                ),
            ),

            # Passo 7: Renomeia para corresponder aos nomes no dataset extraído␣
 ↪do HuggingFace
            RenameFields(
                mapping = {
                    FieldName.FEAT_STATIC_CAT: "static_categorical_features",
                    FieldName.FEAT_STATIC_REAL: "static_real_features",
                    FieldName.FEAT_TIME: "time_features",
                    FieldName.TARGET: "values",
                    FieldName.OBSERVED_VALUES: "observed_mask",
                }
            ),
        ]
    )
```

2.10 Divisão nas Amostras de Treino, Validação e Teste

[ ]:
```
# Define uma função para criar um divisor de instâncias e separar os dados em␣
 ↪conjuntos de treino, validação e teste
def dsa_create_instance_splitter(
    config: PretrainedConfig,
    mode: str,
    train_sampler: Optional[InstanceSampler] = None,
    validation_sampler: Optional[InstanceSampler] = None,
) -> Transformation:

    # Garante que o modo especificado seja um dos modos aceitos: treino,␣
 ↪validação ou teste
    assert mode in ["train", "validation", "test"]

    # Define um dicionário mapeando cada modo para um sampler específico ou␣
 ↪utiliza samplers padrões baseados na configuração
    instance_sampler = {
```

```
        "train": train_sampler
        or ExpectedNumInstanceSampler(

            # Sampler para treino, esperando um número específico de instâncias␣
↪com um mínimo de pontos futuros
            num_instances = 1.0, min_future = config.prediction_length
        ),
        "validation": validation_sampler

        # Sampler para validação, dividindo com base em um mínimo de pontos␣
↪futuros
        or ValidationSplitSampler(min_future = config.prediction_length),

        # Sampler para teste, sem necessidade de pontos futuros específicos
        "test": TestSplitSampler(),
    }[mode]

    # Cria e retorna um divisor de instâncias configurado com o sampler␣
↪adequado e outras definições relevantes
    return InstanceSplitter(

        # Campo alvo para a previsão
        target_field = "values",

        # Campo que indica se um ponto de dados é um preenchimento (padding)
        is_pad_field = FieldName.IS_PAD,

        # Campo que indica o início da série temporal
        start_field = FieldName.START,

        # Campo que indica o início da previsão
        forecast_start_field = FieldName.FORECAST_START,

        # O sampler de instâncias escolhido baseado no modo
        instance_sampler = instance_sampler,

        # Define o comprimento do contexto passado
        past_length = config.context_length + max(config.lags_sequence),

        # Define o comprimento da previsão futura
        future_length = config.prediction_length,

        # Campos adicionais da série temporal a serem incluídos
        time_series_fields = ["time_features", "observed_mask"],
    )
```

2.11 Criação dos Dataloaders

[ ]:
```
# Define uma função para criar um DataLoader de treinamento com base nas␣
↪configurações e dados fornecidos
def dsa_cria_dataloader_treino(config: PretrainedConfig,
                               freq,
                               data,
                               batch_size: int,
                               num_batches_per_epoch: int,
                               shuffle_buffer_length: Optional[int] = None,
                               cache_data: bool = True,
                               **kwargs) -> Iterable:

    # Inicia a lista de nomes de entradas para previsão baseando-se na␣
↪configuração de características estáticas e temporais
    PREDICTION_INPUT_NAMES = ["past_time_features", "past_values",␣
↪"past_observed_mask", "future_time_features"]

    # Adiciona o nome da característica categórica estática à lista se presente␣
↪na configuração
    if config.num_static_categorical_features > 0:
        PREDICTION_INPUT_NAMES.append("static_categorical_features")

    # Adiciona o nome da característica real estática à lista se presente na␣
↪configuração
    if config.num_static_real_features > 0:
        PREDICTION_INPUT_NAMES.append("static_real_features")

    # Estende a lista de nomes de entradas para incluir os dados de treinamento␣
↪específicos
    TRAINING_INPUT_NAMES = PREDICTION_INPUT_NAMES + [
        "future_values",
        "future_observed_mask",
    ]

    # Cria a transformação dos dados com base na frequência e configuração␣
↪fornecida
    transformation = dsa_cria_transformacao(freq, config)

    # Aplica a transformação nos dados no modo de treinamento
    transformed_data = transformation.apply(data, is_train = True)

    # Se solicitado, armazena em cache os dados transformados para otimizar o␣
↪treinamento
    if cache_data:
        transformed_data = Cached(transformed_data)
```

```
    # Cria um divisor de instâncias para o treinamento que define como as janelas de dados serão amostradas
    instance_splitter = dsa_create_instance_splitter(config, "train")

    # Cria um fluxo cíclico dos dados transformados para amostragem contínua
    stream = Cyclic(transformed_data).stream()

    # Aplica o divisor de instâncias ao fluxo de dados para gerar instâncias de treinamento
    training_instances = instance_splitter.apply(stream, is_train = True)

    # Retorna os lotes empilhados das instâncias de treinamento conforme especificado
    return as_stacked_batches(
        training_instances,
        batch_size = batch_size,
        shuffle_buffer_length = shuffle_buffer_length,  # Define o comprimento do buffer de embaralhamento, se aplicável
        field_names = TRAINING_INPUT_NAMES,  # Especifica os nomes dos campos a serem incluídos nos lotes
        output_type = torch.tensor,  # Define o tipo de saída dos lotes
        num_batches_per_epoch = num_batches_per_epoch,  # Especifica o número de lotes por época
    )
```

[ ]:
```
# Define uma função para criar um DataLoader de teste a partir de configurações pré-definidas e dados
def dsa_cria_dataloader_teste(config: PretrainedConfig, freq, data, batch_size: int, **kwargs):

    # Define os nomes das entradas necessárias para previsão baseadas na configuração
    PREDICTION_INPUT_NAMES = ["past_time_features", "past_values", "past_observed_mask", "future_time_features"]

    # Se houver características categóricas estáticas, adiciona ao conjunto de entradas de previsão
    if config.num_static_categorical_features > 0:
        PREDICTION_INPUT_NAMES.append("static_categorical_features")

    # Se houver características reais estáticas, adiciona ao conjunto de entradas de previsão
    if config.num_static_real_features > 0:
        PREDICTION_INPUT_NAMES.append("static_real_features")
```

```
    # Cria a transformação a ser aplicada nos dados com base na frequência e␣
↪configuração
    transformation = dsa_cria_transformacao(freq, config)

    # Aplica a transformação aos dados no modo de teste (não treinamento)
    transformed_data = transformation.apply(data, is_train=False)

    # Cria um amostrador de instâncias para o modo de teste que irá selecionar␣
↪a última janela de contexto vista durante o treinamento
    instance_sampler = dsa_create_instance_splitter(config, "test")

    # Aplica o amostrador de instâncias aos dados transformados no modo de teste
    testing_instances = instance_sampler.apply(transformed_data, is_train =␣
↪False)

    # Retorna os lotes empilhados como tensores PyTorch, com um tamanho de lote␣
↪especificado e usando os nomes de campos definidos
    return as_stacked_batches(
        testing_instances,
        batch_size = batch_size,
        output_type = torch.tensor,
        field_names = PREDICTION_INPUT_NAMES,
    )
```

[ ]:
```
# Cria o dataloader de treino
dl_treino = dsa_cria_dataloader_treino(config = dsa_config,
                                       freq = freq,
                                       data = dsa_dataset_treino,
                                       batch_size = 256,
                                       num_batches_per_epoch = 100)
```

[ ]:
```
# Imprime as chaves de um batch de dados
batch = next(iter(dl_treino))
for k, v in batch.items():
    print(k)
```

```
past_time_features
past_values
past_observed_mask
future_time_features
static_categorical_features
future_values
future_observed_mask
```

[ ]:
```
# Imprime as chaves e respectivos valores de um batch de dados
batch = next(iter(dl_treino))
for k, v in batch.items():
```

```
    print(k, v)
```

```
past_time_features tensor([[[ 0.1364,  2.1106],
         [ 0.2273,  2.1139],
         [ 0.3182,  2.1173],
         …,
         [-0.0455,  2.3243],
         [ 0.0455,  2.3263],
         [ 0.1364,  2.3284]],

        [[-0.4091,  1.7993],
         [-0.3182,  1.8062],
         [-0.2273,  1.8129],
         …,
         [ 0.5000,  2.1614],
         [-0.5000,  2.1644],
         [-0.4091,  2.1673]],

        [[ 0.0000,  0.0000],
         [ 0.0000,  0.0000],
         [ 0.0000,  0.0000],
         …,
         [ 0.2273,  1.5315],
         [ 0.3182,  1.5441],
         [ 0.4091,  1.5563]],

        …,

        [[ 0.0455,  1.5051],
         [ 0.1364,  1.5185],
         [ 0.2273,  1.5315],
         …,
         [-0.1364,  2.0569],
         [-0.0455,  2.0607],
         [ 0.0455,  2.0645]],

        [[ 0.0000,  0.0000],
         [ 0.0000,  0.0000],
         [ 0.0000,  0.0000],
         …,
         [-0.4091,  1.7993],
         [-0.3182,  1.8062],
         [-0.2273,  1.8129]],

        [[ 0.0000,  0.0000],
         [ 0.0000,  0.0000],
         [ 0.0000,  0.0000],
         …,
```

```
         [ 0.0455,  0.9031],
         [ 0.1364,  0.9542],
         [ 0.2273,  1.0000]]])
past_values tensor([[  110.,    70.,   121.,  …,   112.,    51.,    96.],
        [  856.,   648.,   656.,  …,  6468.,  1352.,  2220.],
        [    0.,     0.,     0.,  …,   120.,   108.,   136.],
        …,
        [ 8900., 10300., 12800.,  …, 10000., 11400., 10400.],
        [    0.,     0.,     0.,  …,  3300.,  4300.,  3500.],
        [    0.,     0.,     0.,  …,   400.,  1000.,   800.]])
past_observed_mask tensor([[1., 1., 1.,  …, 1., 1., 1.],
        [1., 1., 1.,  …, 1., 1., 1.],
        [0., 0., 0.,  …, 1., 1., 1.],
        …,
        [1., 1., 1.,  …, 1., 1., 1.],
        [0., 0., 0.,  …, 1., 1., 1.],
        [0., 0., 0.,  …, 1., 1., 1.]])
future_time_features tensor([[[ 0.2273,  2.3304],
         [ 0.3182,  2.3324],
         [ 0.4091,  2.3345],
         …,
         [-0.0455,  2.3711],
         [ 0.0455,  2.3729],
         [ 0.1364,  2.3747]],

        [[-0.3182,  2.1703],
         [-0.2273,  2.1732],
         [-0.1364,  2.1761],
         …,
         [ 0.5000,  2.2279],
         [-0.5000,  2.2304],
         [-0.4091,  2.2330]],

        [[ 0.5000,  1.5682],
         [-0.5000,  1.5798],
         [-0.4091,  1.5911],
         …,
         [ 0.2273,  1.7634],
         [ 0.3182,  1.7709],
         [ 0.4091,  1.7782]],


        …,

        [[ 0.1364,  2.0682],
         [ 0.2273,  2.0719],
         [ 0.3182,  2.0755],
         …,
         [-0.1364,  2.1399],
```

```
          [-0.0455,  2.1430],
          [ 0.0455,  2.1461]],

         [[-0.1364,  1.8195],
          [-0.0455,  1.8261],
          [ 0.0455,  1.8325],
          …,
          [-0.4091,  1.9395],
          [-0.3182,  1.9445],
          [-0.2273,  1.9494]],

         [[ 0.3182,  1.0414],
          [ 0.4091,  1.0792],
          [ 0.5000,  1.1139],
          …,
          [ 0.0455,  1.5051],
          [ 0.1364,  1.5185],
          [ 0.2273,  1.5315]]])
static_categorical_features tensor([[267],
        [269],
        [271],
        [272],
        [272],
        [273],
        [273],
        [275],
        [275],
        [275],
        [275],
        [276],
        [280],
        [280],
        [280],
        [281],
        [281],
        [282],
        [282],
        [283],
        [284],
        [285],
        [285],
        [287],
        [288],
        [291],
        [291],
        [291],
        [293],
        [293],
```

[293],
[295],
[296],
[296],
[297],
[297],
[298],
[298],
[299],
[300],
[300],
[300],
[301],
[302],
[302],
[305],
[306],
[307],
[308],
[308],
[308],
[308],
[309],
[309],
[311],
[313],
[313],
[314],
[316],
[317],
[317],
[319],
[319],
[319],
[322],
[325],
[330],
[332],
[333],
[333],
[333],
[333],
[335],
[336],
[338],
[338],
[338],
[338],

```
[339],
[340],
[340],
[340],
[342],
[342],
[343],
[343],
[343],
[344],
[344],
[344],
[345],
[345],
[345],
[346],
[348],
[348],
[348],
[349],
[350],
[351],
[352],
[352],
[353],
[353],
[353],
[354],
[355],
[355],
[355],
[356],
[356],
[361],
[362],
[362],
[  0],
[  0],
[  3],
[  4],
[  5],
[  5],
[  7],
[  8],
[  8],
[  8],
[  9],
[  9],
```

```
[ 10],
[ 10],
[ 10],
[ 11],
[ 11],
[ 13],
[ 14],
[ 15],
[ 17],
[ 17],
[ 18],
[ 18],
[ 19],
[ 19],
[ 20],
[ 22],
[ 22],
[ 22],
[ 24],
[ 25],
[ 25],
[ 25],
[ 26],
[ 26],
[ 28],
[ 29],
[ 30],
[ 31],
[ 32],
[ 33],
[ 33],
[ 35],
[ 36],
[ 37],
[ 38],
[ 40],
[ 41],
[ 42],
[ 42],
[ 43],
[ 44],
[ 46],
[ 50],
[ 53],
[ 53],
[ 55],
[ 56],
[ 57],
```

```
[ 57],
[ 58],
[ 59],
[ 61],
[ 63],
[ 64],
[ 65],
[ 65],
[ 65],
[ 67],
[ 69],
[ 69],
[ 69],
[ 71],
[ 71],
[ 72],
[ 74],
[ 76],
[ 79],
[ 79],
[ 80],
[ 80],
[ 82],
[ 82],
[ 82],
[ 83],
[ 83],
[ 83],
[ 83],
[ 83],
[ 84],
[ 85],
[ 86],
[ 87],
[ 88],
[ 88],
[ 88],
[ 88],
[ 90],
[ 90],
[ 90],
[ 91],
[ 91],
[ 91],
[ 92],
[ 94],
[ 94],
[ 95],
```

```
        [ 96],
        [ 97],
        [ 97],
        [ 99],
        [101],
        [101],
        [101],
        [102],
        [102],
        [109],
        [110],
        [111],
        [111],
        [113],
        [113],
        [116],
        [118],
        [123],
        [126],
        [127],
        [128],
        [130],
        [132],
        [137],
        [141],
        [144],
        [144],
        [148],
        [149],
        [150],
        [152],
        [155],
        [156],
        [157]], dtype=torch.int32)
future_values tensor([[  130.,   132.,   276.,  …,   286.,   294.,   247.],
        [ 2057.,  3620.,  2200.,  …,  9114.,  2240.,  5292.],
        [  100.,   200.,   132.,  …,   124.,   212.,   220.],
        …,
        [10500., 12000., 13100.,  …, 11800., 10300.,  9900.],
        [ 3200.,  4000.,  3400.,  …,  4000.,  5600.,  4900.],
        [  900.,   700.,  1000.,  …,  1000.,  1500.,  1400.]])
future_observed_mask tensor([[1., 1., 1.,  …, 1., 1., 1.],
        [1., 1., 1.,  …, 1., 1., 1.],
        [1., 1., 1.,  …, 1., 1., 1.],
        …,
        [1., 1., 1.,  …, 1., 1., 1.],
        [1., 1., 1.,  …, 1., 1., 1.],
        [1., 1., 1.,  …, 1., 1., 1.]])
```

[ ]:
```
# Cria o dataloader de teste
dl_teste = dsa_cria_dataloader_teste(config = dsa_config,
                                     freq = freq,
                                     data = dsa_dataset_teste,
                                     batch_size = 64)
```

2.12 Loop de Treinamento do Modelo

[ ]:
```
# Cria o acelerador
accelerator = Accelerator()
```

[ ]:
```
# Registra o device
device = accelerator.device
```

[ ]:
```
device
```

[ ]:
```
device(type='cpu')
```

[ ]:
```
# Envia o modelo para o device
modelo_dsa.to(device)
```

[ ]:
```
TimeSeriesTransformerForPrediction(
  (model): TimeSeriesTransformerModel(
    (scaler): TimeSeriesMeanScaler()
    (embedder): TimeSeriesFeatureEmbedder(
      (embedders): ModuleList(
        (0): Embedding(366, 2)
      )
    )
    (encoder): TimeSeriesTransformerEncoder(
      (value_embedding): TimeSeriesValueEmbedding(
        (value_projection): Linear(in_features=22, out_features=32, bias=False)
      )
      (embed_positions): TimeSeriesSinusoidalPositionalEmbedding(72, 32)
      (layers): ModuleList(
        (0-3): 4 x TimeSeriesTransformerEncoderLayer(
          (self_attn): TimeSeriesTransformerAttention(
            (k_proj): Linear(in_features=32, out_features=32, bias=True)
            (v_proj): Linear(in_features=32, out_features=32, bias=True)
            (q_proj): Linear(in_features=32, out_features=32, bias=True)
            (out_proj): Linear(in_features=32, out_features=32, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
          (activation_fn): GELUActivation()
          (fc1): Linear(in_features=32, out_features=32, bias=True)
          (fc2): Linear(in_features=32, out_features=32, bias=True)
```

```
          (final_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
    )
    (decoder): TimeSeriesTransformerDecoder(
      (value_embedding): TimeSeriesValueEmbedding(
        (value_projection): Linear(in_features=22, out_features=32, bias=False)
      )
      (embed_positions): TimeSeriesSinusoidalPositionalEmbedding(72, 32)
      (layers): ModuleList(
        (0-3): 4 x TimeSeriesTransformerDecoderLayer(
          (self_attn): TimeSeriesTransformerAttention(
            (k_proj): Linear(in_features=32, out_features=32, bias=True)
            (v_proj): Linear(in_features=32, out_features=32, bias=True)
            (q_proj): Linear(in_features=32, out_features=32, bias=True)
            (out_proj): Linear(in_features=32, out_features=32, bias=True)
          )
          (activation_fn): GELUActivation()
          (self_attn_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
          (encoder_attn): TimeSeriesTransformerAttention(
            (k_proj): Linear(in_features=32, out_features=32, bias=True)
            (v_proj): Linear(in_features=32, out_features=32, bias=True)
            (q_proj): Linear(in_features=32, out_features=32, bias=True)
            (out_proj): Linear(in_features=32, out_features=32, bias=True)
          )
          (encoder_attn_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
          (fc1): Linear(in_features=32, out_features=32, bias=True)
          (fc2): Linear(in_features=32, out_features=32, bias=True)
          (final_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
    )
  )
  (parameter_projection): ParameterProjection(
    (proj): ModuleList(
      (0-2): 3 x Linear(in_features=32, out_features=1, bias=True)
    )
    (domain_map): LambdaLayer()
  )
```

```
)
```

[ ]:
```
# Otimizador
optimizer = AdamW(modelo_dsa.parameters(), lr = 6e-4, betas = (0.9, 0.95),␣
 ↪weight_decay = 1e-1)
```

[ ]:
```
# Carrega o modelo, o otimizador e o dataloader de treino
modelo_dsa, optimizer, dl_treino = accelerator.prepare(modelo_dsa, optimizer,␣
 ↪dl_treino)
```

2.12.1 Matemática da Otimização do Modelo

Aqui estão as etapas do AdamW em termos matemáticos:

Cálculo dos Momentos de Primeira e Segunda Ordem

Para cada parâmetro , Adam mantém uma estimativa do primeiro momento (a média móvel dos gradientes passados) e do segundo momento (a média móvel dos quadrados dos gradientes passados). Para uma dada etapa t, gradiente g_t e parâmetros de decaimento 1 e 2, essas estimativas são atualizadas da seguinte forma:

- m_t = 1 * m_(t-1) + (1 - 1) * g_t
- v_t = 2 * v_(t-1) + (1 - 2) * g_t^2

Correção de Viés

Como m_t e v_t são inicializados como zero, eles são tendenciosos para zero no início do treinamento. Portanto, Adam realiza uma correção de viés para compensar isso:

- m_t_hat = m_t / (1 - 1^t)
- v_t_hat = v_t / (1 - 2^t)

Atualização de Peso

Finalmente, os pesos são atualizados com uma taxa de aprendizado e um termo de decaimento de peso w:

= - * (m_t_hat / (sqrt(v_t_hat) +) + w *)

Onde é um termo de suavização para evitar a divisão por zero (geralmente algo como 1e-8).

AdamW difere do Adam na forma como o termo de decaimento de peso é aplicado. No Adam original, o decaimento de peso é aplicado antes do cálculo do gradiente, o que pode levar a um acoplamento entre a atualização do peso e a escala do gradiente. AdamW aplica o decaimento de peso diretamente na etapa de atualização do peso, o que "desacopla" a regularização do decaimento de peso da escala do gradiente.

O cálculo das derivadas é uma parte fundamental do treinamento de redes neurais e é aplicado durante o processo de retropropagação (backpropagation), que é usado para atualizar os pesos da rede. No contexto do otimizador AdamW (ou qualquer otimizador baseado em gradiente), a derivada é usada para calcular o gradiente da função de perda com relação a cada peso na rede.

Na prática, você não precisa calcular essas derivadas manualmente. Frameworks modernos de aprendizado profundo, como TensorFlow e PyTorch, usam diferenciação automática para calcular

as derivadas. Você simplesmente define a função de perda e o framework cuida de calcular os gradientes para você.

No caso específico do AdamW, a derivada é usada para calcular o gradiente g_t na etapa de atualização de momentos. Esse gradiente é simplesmente a derivada da função de perda com relação ao peso específico que está sendo atualizado. O gradiente indica a direção e a magnitude da mudança no peso que resultará no maior decréscimo na função de perda.

Então, em resumo, a derivada é usada no cálculo do gradiente, que é então usado para atualizar os pesos da rede neural na direção que minimiza a função de perda. Queremos os pesos que levem ao menor erro possível.

[ ]:
```
%%time

modelo_dsa.train()

# Inicia o loop de treinamento para um número pré-definido de épocas
for epoch in range(10):

    # Itera sobre os lotes do DataLoader de treinamento
    for idx, batch in enumerate(dl_treino):

        # Zera os gradientes do otimizador para evitar acumulação de gradientes␣
↪de iterações anteriores
        optimizer.zero_grad()

        # Gera as saídas do modelo passando as características adequadas do␣
↪lote atual
        outputs = modelo_dsa(

            # Passa características categóricas estáticas para o dispositivo se␣
↪configurado
            static_categorical_features = batch["static_categorical_features"].
↪to(device)
            if dsa_config.num_static_categorical_features > 0
            else None,

            # Passa características reais estáticas para o dispositivo se␣
↪configurado
            static_real_features = batch["static_real_features"].to(device)
            if dsa_config.num_static_real_features > 0
            else None,

            # Passa características temporais passadas para o dispositivo
            past_time_features = batch["past_time_features"].to(device),

            # Passa valores passados para o dispositivo
            past_values = batch["past_values"].to(device),
```

```
            # Passa características temporais futuras para o dispositivo
            future_time_features = batch["future_time_features"].to(device),

            # Passa valores futuros para o dispositivo (usado para treinamento␣
↪supervisionado)
            future_values = batch["future_values"].to(device),

            # Passa a máscara de observações passadas para o dispositivo
            past_observed_mask = batch["past_observed_mask"].to(device),

            # Passa a máscara de observações futuras para o dispositivo
            future_observed_mask = batch["future_observed_mask"].to(device),
        )

        # Atribui a perda calculada pelas saídas do modelo
        loss = outputs.loss

        # Realiza a retropropagação do erro para ajustar os pesos do modelo
        accelerator.backward(loss)

        # Atualiza os pesos do modelo com base nos gradientes calculados
        optimizer.step()

        # A cada 100 lotes, imprime o erro atual do modelo
        if idx % 100 == 0:
            print("Erro do Modelo:", loss.item())
```

```
Erro do Modelo: 8.604155540466309
Erro do Modelo: 7.957498550415039
Erro do Modelo: 7.3465962409973145
Erro do Modelo: 7.789668560028076
Erro do Modelo: 7.436311721801758
Erro do Modelo: 6.830202579498291
Erro do Modelo: 7.599137783050537
Erro do Modelo: 6.999067783355713
Erro do Modelo: 7.547626495361328
Erro do Modelo: 7.114990234375
CPU times: total: 9min 34s
Wall time: 3min 35s
```

2.13 Avaliação do Modelo

[ ]:
```
# Coloca o modelo em modo de avaliação
modelo_dsa.eval()
```

```
[ ]: TimeSeriesTransformerForPrediction(
       (model): TimeSeriesTransformerModel(
         (scaler): TimeSeriesMeanScaler()
         (embedder): TimeSeriesFeatureEmbedder(
           (embedders): ModuleList(
             (0): Embedding(366, 2)
           )
         )
         (encoder): TimeSeriesTransformerEncoder(
           (value_embedding): TimeSeriesValueEmbedding(
             (value_projection): Linear(in_features=22, out_features=32, bias=False)
           )
           (embed_positions): TimeSeriesSinusoidalPositionalEmbedding(72, 32)
           (layers): ModuleList(
             (0-3): 4 x TimeSeriesTransformerEncoderLayer(
               (self_attn): TimeSeriesTransformerAttention(
                 (k_proj): Linear(in_features=32, out_features=32, bias=True)
                 (v_proj): Linear(in_features=32, out_features=32, bias=True)
                 (q_proj): Linear(in_features=32, out_features=32, bias=True)
                 (out_proj): Linear(in_features=32, out_features=32, bias=True)
               )
               (self_attn_layer_norm): LayerNorm((32,), eps=1e-05,
     elementwise_affine=True)
               (activation_fn): GELUActivation()
               (fc1): Linear(in_features=32, out_features=32, bias=True)
               (fc2): Linear(in_features=32, out_features=32, bias=True)
               (final_layer_norm): LayerNorm((32,), eps=1e-05,
     elementwise_affine=True)
             )
           )
           (layernorm_embedding): LayerNorm((32,), eps=1e-05,
     elementwise_affine=True)
         )
         (decoder): TimeSeriesTransformerDecoder(
           (value_embedding): TimeSeriesValueEmbedding(
             (value_projection): Linear(in_features=22, out_features=32, bias=False)
           )
           (embed_positions): TimeSeriesSinusoidalPositionalEmbedding(72, 32)
           (layers): ModuleList(
             (0-3): 4 x TimeSeriesTransformerDecoderLayer(
               (self_attn): TimeSeriesTransformerAttention(
                 (k_proj): Linear(in_features=32, out_features=32, bias=True)
                 (v_proj): Linear(in_features=32, out_features=32, bias=True)
                 (q_proj): Linear(in_features=32, out_features=32, bias=True)
                 (out_proj): Linear(in_features=32, out_features=32, bias=True)
               )
               (activation_fn): GELUActivation()
```

```
          (self_attn_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
          (encoder_attn): TimeSeriesTransformerAttention(
            (k_proj): Linear(in_features=32, out_features=32, bias=True)
            (v_proj): Linear(in_features=32, out_features=32, bias=True)
            (q_proj): Linear(in_features=32, out_features=32, bias=True)
            (out_proj): Linear(in_features=32, out_features=32, bias=True)
          )
          (encoder_attn_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
          (fc1): Linear(in_features=32, out_features=32, bias=True)
          (fc2): Linear(in_features=32, out_features=32, bias=True)
          (final_layer_norm): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((32,), eps=1e-05,
elementwise_affine=True)
    )
  )
  (parameter_projection): ParameterProjection(
    (proj): ModuleList(
      (0-2): 3 x Linear(in_features=32, out_features=1, bias=True)
    )
    (domain_map): LambdaLayer()
  )
)
```

[ ]:
```
# Cria lista para armazenar o forecast
forecasts = []
```

[ ]:
```
# Itera sobre os lotes do DataLoader de teste
for batch in dl_teste:

    # Gera previsões usando o modelo para o lote atual, passando as␣
↪características conforme a configuração
    outputs = modelo_dsa.generate(

        # Passa características categóricas estáticas para o dispositivo se␣
↪houver alguma
        static_categorical_features = batch["static_categorical_features"].
↪to(device)
        if dsa_config.num_static_categorical_features > 0
        else None,

        # Passa características reais estáticas para o dispositivo se houver␣
↪alguma
```

```
        static_real_features = batch["static_real_features"].to(device)
        if dsa_config.num_static_real_features > 0
        else None,

        # Passa características temporais passadas para o dispositivo
        past_time_features = batch["past_time_features"].to(device),

        # Passa valores passados para o dispositivo
        past_values = batch["past_values"].to(device),

        # Passa características temporais futuras para o dispositivo
        future_time_features = batch["future_time_features"].to(device),

        # Passa a máscara de observações passadas para o dispositivo
        past_observed_mask = batch["past_observed_mask"].to(device),
    )

    # Adiciona as sequências de previsões geradas à lista de previsões, movendo␣
↪para a CPU e convertendo para numpy
    forecasts.append(outputs.sequences.cpu().numpy())
```

[ ]:
```
# Shape das previsões
forecasts[0].shape
```

[ ]:
```
(64, 100, 24)
```

[ ]:
```
# Ajuste do shape
forecasts = np.vstack(forecasts)
```

[ ]:
```
print(forecasts.shape)
```

```
(366, 100, 24)
```

2.14 Calculando as Métricas MASE e SMAPE

[ ]:
```
# Métricas
mase_metric = load("evaluate-metric/mase")
smape_metric = load("evaluate-metric/smape")
```

```
Downloading builder script:   0%|          | 0.00/5.50k [00:00<?, ?B/s]
```

```
Downloading builder script:   0%|          | 0.00/6.65k [00:00<?, ?B/s]
```

[ ]:
```
# Mediana das previsões
forecast_median = np.median(forecasts, 1)
```

[ ]:
```
# Listas das métricas
mase_metrics = []
smape_metrics = []
```

[ ]:
```
# Inicializa o loop para percorrer os itens do conjunto de teste
for item_id, ts in enumerate(dsa_dataset_teste):

    # Separa os dados de treinamento excluindo o comprimento da previsão do␣
 ↪final
    training_data = ts["target"][:-prediction_length]

    # Separa os dados reais (ground truth) para o comprimento da previsão
    ground_truth = ts["target"][-prediction_length:]

    # Calcula o MASE usando as previsões, os dados reais, os dados de␣
 ↪treinamento e a periodicidade
    mase = mase_metric.compute(predictions = forecast_median[item_id],
                               references = np.array(ground_truth),
                               training = np.array(training_data),
                               periodicity = get_seasonality(freq))

    # Adiciona o valor calculado de MASE à lista de métricas MASE
    mase_metrics.append(mase["mase"])

    # Calcula o sMAPE usando as previsões e os dados reais
    smape = smape_metric.compute(predictions = forecast_median[item_id],
                                 references = np.array(ground_truth))

    # Adiciona o valor calculado de sMAPE à lista de métricas sMAPE
    smape_metrics.append(smape["smape"])
```

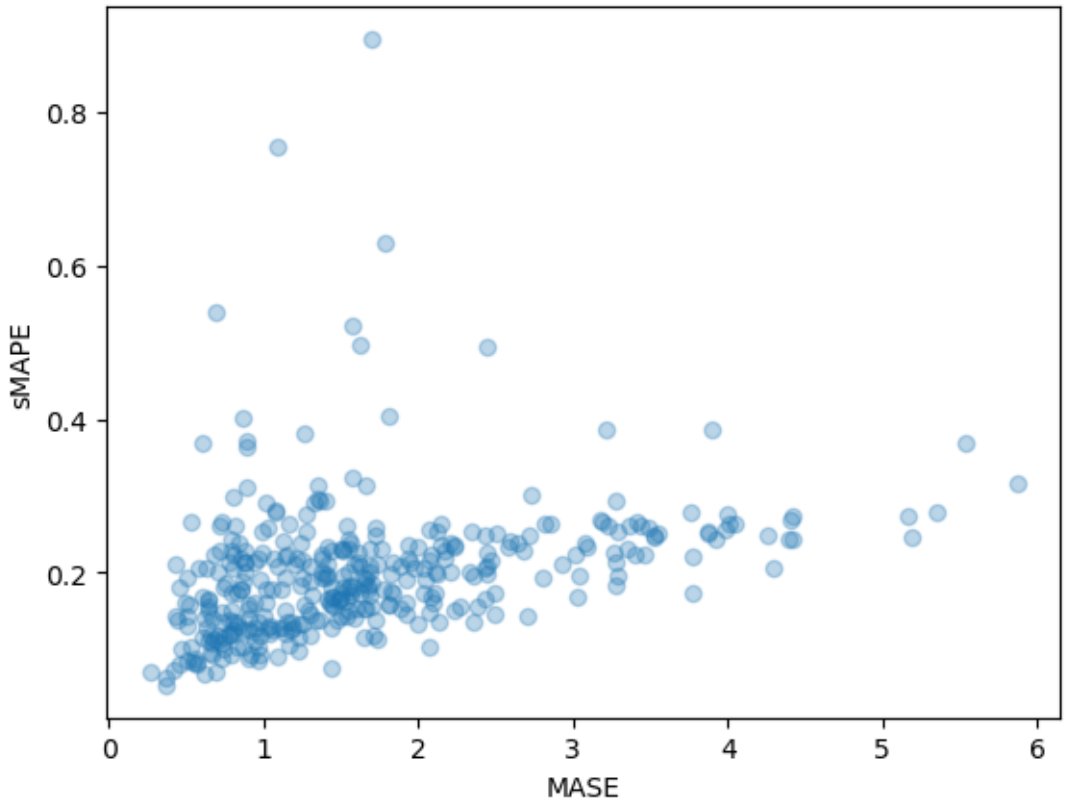
[ ]:
```
print(f"MASE: {np.mean(mase_metrics)}")
```

```
MASE: 1.6625070338437016
```

[ ]:
```
print(f"sMAPE: {np.mean(smape_metrics)}")
```

```
sMAPE: 0.1998331521528693
```

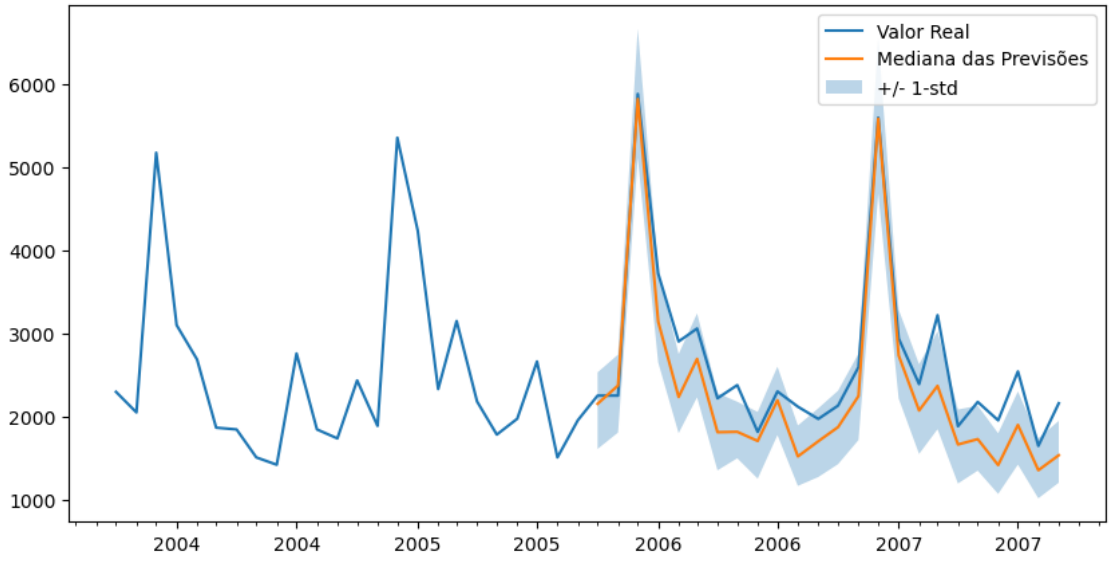
[ ]:
```
# Plot
plt.scatter(mase_metrics, smape_metrics, alpha = 0.3)
plt.xlabel("MASE")
plt.ylabel("sMAPE")
plt.show()
```



[ ]:
```
# Define a função para plotar a série temporal e previsões para um índice␣
 ↪específico
def plot(ts_index):

    # Cria uma figura e um eixo para o plot
    fig, ax = plt.subplots(figsize = (10, 5))

    # Gera o índice de datas para a série temporal a partir dos metadados e do␣
 ↪comprimento do alvo
    index = pd.period_range(start = dsa_dataset_teste[ts_index][FieldName.
 ↪START],
                            periods = len(dsa_dataset_teste[ts_index][FieldName.
 ↪TARGET]),
                            freq = freq).to_timestamp()

    # Configura os locais principais do eixo x para serem nos meses de janeiro␣
 ↪e julho
    ax.xaxis.set_major_locator(mdates.MonthLocator(bymonth=(1, 7)))

    # Configura os locais secundários do eixo x para serem em todos os meses
```

```
    ax.xaxis.set_minor_locator(mdates.MonthLocator())

    # Plota os valores reais da série temporal para os últimos 2 períodos de previsão
    ax.plot(index[-2*prediction_length:],
            dsa_dataset_teste[ts_index]["target"][-2*prediction_length:],
            label="Valor Real")

    # Plota a mediana das previsões para o último período de previsão
    plt.plot(index[-prediction_length:],
             np.median(forecasts[ts_index], axis=0),
             label = "Mediana das Previsões")

    # Preenche a área entre a média menos o desvio padrão e a média mais o desvio padrão das previsões
    plt.fill_between(
        index[-prediction_length:],
        forecasts[ts_index].mean(0) - forecasts[ts_index].std(axis=0),
        forecasts[ts_index].mean(0) + forecasts[ts_index].std(axis=0),
        alpha = 0.3,
        interpolate = True,
        label = "+/- 1-std",
    )

    # Adiciona uma legenda ao plot
    plt.legend()

    # Exibe o plot
    plt.show()
```

[ ]:
```
# Previsão para o índice 334
plot(334)
```



[ ]:
```
%reload_ext watermark
%watermark -a "Data Science Academy"
```

```
Author: Data Science Academy
```

[ ]:
```
#%watermark -v -m
```

[ ]:
```
#%watermark --iversions
```

3 Fim