Ukraine vs Russia War -Copy4

April 18, 2022

0.1 Prevendo perdas russas na Guerra da Ucrânia com ML

1 1. Importando as bibliotecas

[362]:
```
"""
1°) Importação do pandas como pd para trabalhar com dados.
"""
import pandas as pd
"""
2°) Importação do numpy como np para trabalhar com matrizes e tudo mais.
"""
import numpy as np
"""
3°) Importação do matplotlib.pyplot como plt para fazer gráficos.
"""
import matplotlib.pyplot as plt
"""
4°) De matplotlib.ticker vamos importar o AutoMinorLocator e o MaxNLocator para␣
 ↪trabalhar com os "ticks"
    dos gráficos.
"""
import matplotlib.ticker as mticker
from matplotlib.ticker import AutoMinorLocator, MaxNLocator
"""
5°) De matplotlib.font_manager vamos importar FontProperties para criar fontes␣
 ↪de texto.
"""
from matplotlib.font_manager import FontProperties
"""
6°) Importação do seaborn para fazer gráficos
"""
import seaborn as sbn
"""
7°) Importação de pycaret.time_series para trabalhar com séries temporais
"""
from pycaret.time_series import *
"""
8°) Ignorar alguns warnings que não afetam o código
```

```
"""
import warnings
warnings.filterwarnings("ignore")
```

2 2. Importação dos dados

[363]:
```
"""
1°) Importação do dataset que possui as perdas russas em termos de equipamento␣
 ↪bélico
"""
losses_equipment = pd.read_csv("russia_losses_equipment_git.csv")
"""
2°) Importação do dataset que possui as perdas russas em termos de tropas
"""
losses_personnel = pd.read_csv("russia_losses_personnel_git.csv")
```

3 3. Pré-processamento de dados

[364]:
```
"""
Vamos começar concatenando os dois DFs anteriores
"""
"""
1°) Como já vai haver uma coluna de "data" e uma de "dias de guerra" no DF␣
 ↪"losses_equipment",
não faz sentido manter as mesmas no DF "losses_personnel". Além disso, vamos␣
 ↪excluir algumas
colunas que não serão usadas no presente trabalho.
"""
losses_personnel.drop(["date",
                       "day",
                       "personnel*",
                       "POW"], axis = 1, inplace = True)
losses_equipment.drop(["APC",
                       "field artillery",
                       "MRL",
                       "fuel tank",
                       "special equipment",
                       "mobile SRBM system"], axis = 1, inplace = True)
"""
Concatenando os dois DFs anteriores
"""
Dados = pd.concat([losses_equipment, losses_personnel], axis = 1)
"""
Mostrar na tela a parte superior do DF "Dados" com suas 10 colunas
"""
```

```python
pd.set_option("display.max_columns", 10)
Dados.head()
```

[364]:

```
         date  day  aircraft  helicopter  tank  military auto  drone  \
0  2022-02-25    2        10           7    80            100      0
1  2022-02-26    3        27          26   146            130      2
2  2022-02-27    4        27          26   150            130      2
3  2022-02-28    5        29          29   150            291      3
4  2022-03-01    6        29          29   198            305      3

   naval ship  anti-aircraft warfare  personnel
0           2                      0       2800
1           2                      0       4300
2           2                      0       4500
3           2                      5       5300
4           2                      7       5710
```

[365]:

```python
"""
Tranformação da data de object para datetime64[ns]
"""
Dados["date"] = pd.to_datetime(Dados["date"])
```

4 3.1 Dados faltantes

[366]:

```python
"""
Calcula as porcentagens de dados missing em cada coluna do DF
"""
Dados.isnull().sum()/len(Dados["date"])
```

[366]:

```
date                     0.0
day                      0.0
aircraft                 0.0
helicopter               0.0
tank                     0.0
military auto            0.0
drone                    0.0
naval ship               0.0
anti-aircraft warfare    0.0
personnel                0.0
dtype: float64
```

[367]:

```python
Dados.describe()
```

[367]:

```
             day    aircraft  helicopter        tank  military auto  \
count  53.000000   53.000000   53.000000   53.000000      53.000000
mean   28.000000  100.584906  103.622642  495.962264     924.509434
std    15.443445   48.798421   40.976178  200.155897     427.232941
```

```
min      2.000000   10.000000    7.000000   80.000000     100.000000
25%     15.000000   49.000000   81.000000  335.000000     526.000000
50%     28.000000  101.000000  124.000000  517.000000    1008.000000
75%     41.000000  150.000000  134.000000  676.000000    1322.000000
max     54.000000  167.000000  147.000000  790.000000    1487.000000
```

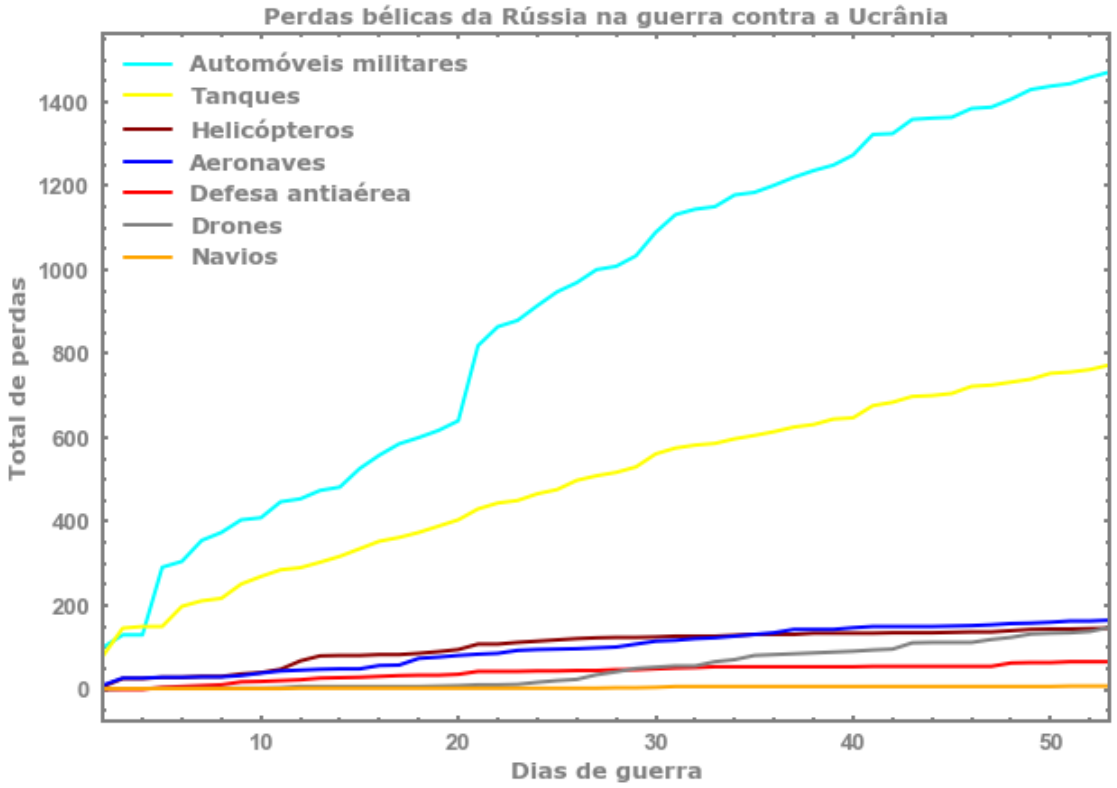
```
            drone  naval ship  anti-aircraft warfare    personnel
count   53.000000   53.000000              53.000000     53.00000
mean    53.867925    4.773585              41.339623  14564.45283
std     50.573267    2.275636              19.206312   4755.46464
min      0.000000    2.000000               0.000000   2800.00000
25%      7.000000    3.000000              29.000000  12000.00000
50%     42.000000    4.000000              47.000000  15600.00000
75%     94.000000    7.000000              55.000000  18500.00000
max    155.000000    8.000000              67.000000  20600.00000
```

5 3.2 Análise de dados

[368]:
```
"""
Criação da primeira fonte de texto para colocar como fonte dos labels
"""
font1 = {"family": "Verdana", "weight": "bold", "color": "gray", "size": 13}
"""
Criação da segunda fonte de texto para colocar como fonte da legenda
"""
font2 = FontProperties(family = "Verdana",
                       weight = "bold",
                       size = 13)
"""
Criando um "local" para alocar a nossa figura
"""
fig, axs = plt.subplots(figsize = (10, 7))
"""
Plot das curvas
"""
axs.plot(Dados["day"], Dados["military auto"], linewidth = 2, label =␣
 ↪"Automóveis militares", color = "cyan")
axs.plot(Dados["day"], Dados["tank"], linewidth = 2, label = "Tanques", color =␣
 ↪"yellow")
axs.plot(Dados["day"], Dados["helicopter"], linewidth = 2, label =␣
 ↪"Helicópteros", color = "darkred")
axs.plot(Dados["day"], Dados["aircraft"], linewidth = 2, label = "Aeronaves",␣
 ↪color = "blue")
axs.plot(Dados["day"], Dados["anti-aircraft warfare"], linewidth = 2, label =␣
 ↪"Defesa antiaérea", color = "red")
```

```
axs.plot(Dados["day"], Dados["drone"], linewidth = 2, label = "Drones", color =␣
 ↪"gray")
axs.plot(Dados["day"], Dados["naval ship"], linewidth = 2, label = "Navios",␣
 ↪color = "orange")
"""
Grid = False
"""
axs.grid(False)
"""
Definindo a "grossura" e a cor do eixos
"""
for axis in ["left", "top", "right", "bottom"]:
    axs.spines[axis].set_linewidth(2)
    axs.spines[axis].set_color("gray")
"""
Trabalha com os ticks do gráfico
"""
axs.xaxis.set_minor_locator(AutoMinorLocator())
axs.yaxis.set_minor_locator(AutoMinorLocator())
axs.tick_params(axis = "both", direction = "in", labelcolor = "gray", labelsize␣
 ↪= 13, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='minor', direction = "in", length=2, color='gray', width␣
 ↪= 2, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='major', direction = "in", color='gray', length=3.4,␣
 ↪width = 2, top = True, right = True, left = True, bottom = True)
"""
Definindo um intervalo para o eixo x do gráfico
"""
plt.xlim(2, 53)
"""
Legenda da figura
"""
plt.legend(frameon = False, prop = font2, labelcolor = "gray")
"""
Tudo em negrito
"""
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"
"""
Labels
"""
axs.set_xlabel("Dias de guerra", fontdict = font1)
axs.set_ylabel("Total de perdas", fontdict = font1)
"""
Fundo branco
"""
fig.patch.set_facecolor("white")
```

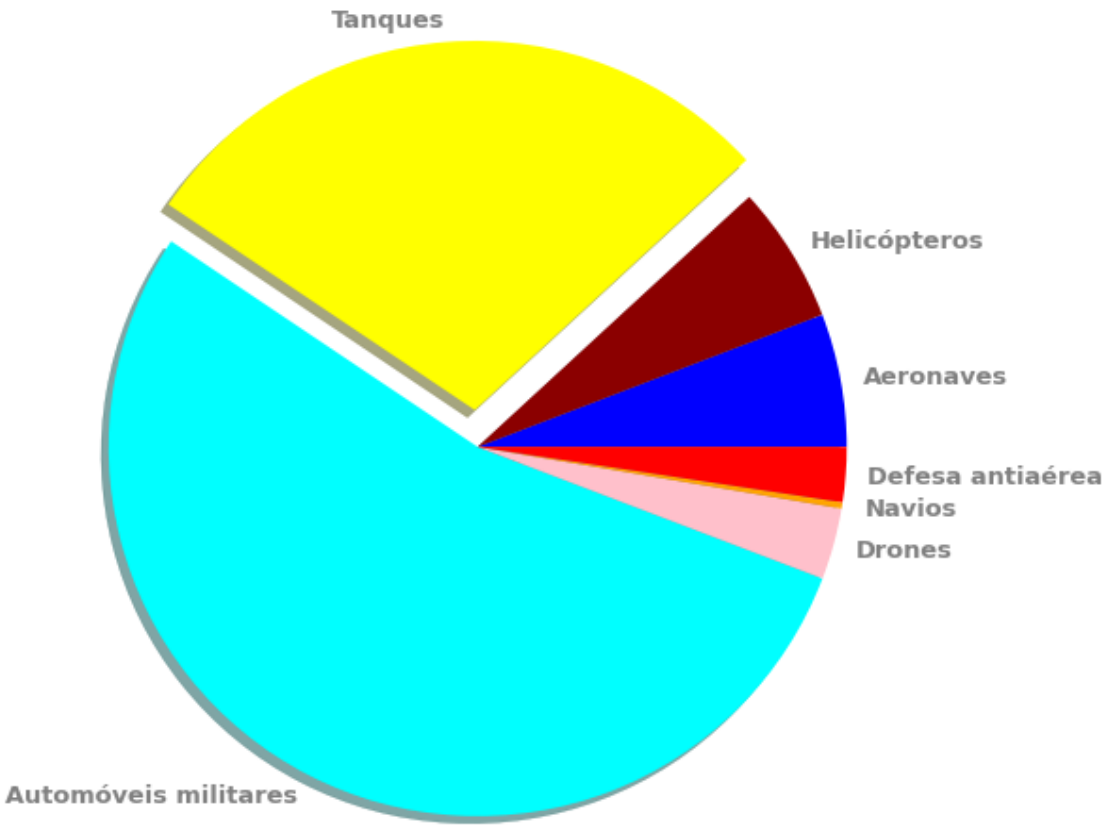
```
"""
Título da figura
"""
axs.set_title("Perdas bélicas da Rússia na guerra contra a Ucrânia", fontdict =␣
 ↪font1)
plt.show()
```



[369]:
```
Lista_de_perdas_para_cada_equipamento = []
Lista_de_equipamentos = ["Aeronaves", "Helicópteros", "Tanques", "Automóveis␣
 ↪militares", "Drones", "Navios", "Defesa antiaérea"]
for c in ["aircraft", "helicopter", "tank", "military auto", "drone", "naval␣
 ↪ship", "anti-aircraft warfare"]:
    S = 0 # Variável soma
    for i in Dados[c]:
        S = S + i
    Lista_de_perdas_para_cada_equipamento.append(S)
```

[370]:
```
fig, axs = plt.subplots(figsize = (10, 9))
"""
Plot de um gráfico do tipo pizza
"""
```

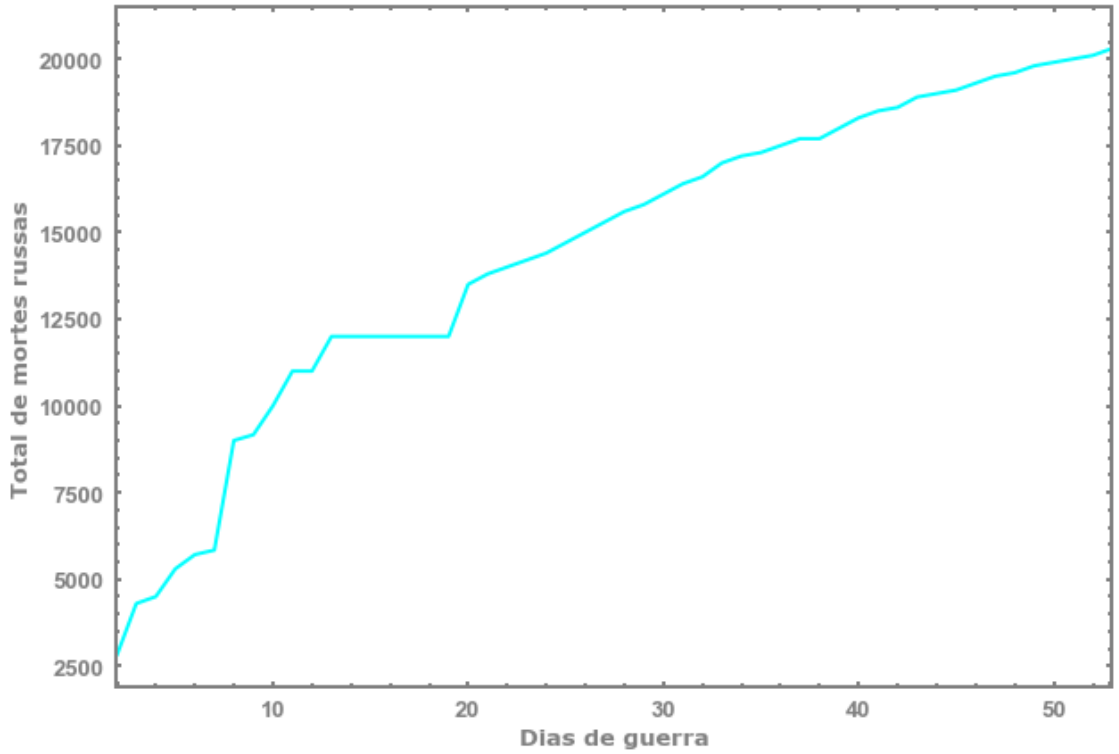
```
axs.pie(x = Lista_de_perdas_para_cada_equipamento,
        labels = Lista_de_equipamentos,
        shadow = True,
        explode = [0, 0, 0.1, 0, 0, 0, 0], # Lista de seprações entre os␣
↪"pedaços da pizza"
        textprops={"family": "verbose",
                   "weight": "bold",
                   "color" :"gray",
                   'fontsize': 13},
        colors = ["blue", "darkred", "yellow", "cyan", "pink", "orange", "red"],
        labeldistance = 1.06) # Distância dos labels ao centro da pizza
fig.patch.set_facecolor("white")
plt.show()
```



Em termos bélicos, o que a Russia mais perde são automóveis militares e tanques de guerra.

[371]:
```
fig, axs = plt.subplots(figsize = (10, 7))
axs.plot(Dados["day"], Dados["personnel"], linewidth = 2, color = "cyan")
```

```
axs.grid(False)
for axis in ["left", "top", "right", "bottom"]:
    axs.spines[axis].set_linewidth(2)
    axs.spines[axis].set_color("gray")
axs.xaxis.set_minor_locator(AutoMinorLocator())
axs.yaxis.set_minor_locator(AutoMinorLocator())
axs.tick_params(axis = "both", direction = "in", labelcolor = "gray", labelsize␣
 ↪= 13, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='minor', direction = "in", length=2, color='gray', width␣
 ↪= 2, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='major', direction = "in", color='gray', length=3.4,␣
 ↪width = 2, top = True, right = True, left = True, bottom = True)
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"
plt.xlim(2, 53)
axs.set_xlabel("Dias de guerra", fontdict = font1)
axs.set_ylabel("Total de mortes russas", fontdict = font1)
fig.patch.set_facecolor("white")
plt.show()
```



Muitos russos foram mortos e, pelo andar crescente do gráfico, vem mais por aí...

6 4. Previsão das séries temporais

Realizaremos previsões para 5 variáveis; Aeronaves, helicópteros, tanques de guerra, automóveis e mortes russas.

[372]:
```
"""
Definindo vários DFs apenas com a data e uma variável.
"""
aeronaves = Dados[["date", "aircraft"]]
helicopteros = Dados[["date", "helicopter"]]
tanques = Dados[["date", "tank"]]
automoveis_militares = Dados[["date", "military auto"]]
pessoas = Dados[["date", "personnel"]]
```

[373]:
```
"""
Transformando a data em índices
"""
aeronaves.set_index("date", drop = True, inplace = True)
helicopteros.set_index("date", drop = True, inplace = True)
tanques.set_index("date", drop = True, inplace = True)
automoveis_militares.set_index("date", drop = True, inplace = True)
pessoas.set_index("date", drop = True, inplace = True)
```

[374]:
```
"""
Criando um setup com a variável
"""
setup(aeronaves, fh = 5, fold = 6, seasonal_period="D", n_jobs = -1, use_gpu =␣
 ↪True)
```

```
<pandas.io.formats.style.Styler at 0x22c1e583760>
```

```
INFO:logs:self.master_model_container: 0
INFO:logs:self.display_container: 1
INFO:logs:Pipeline(memory=None,
         steps=[('dtypes',
                 DataTypes_Auto_infer(categorical_features=[],
                                      display_types=False, features_todrop=[],
                                      float_dtype='float64', id_columns=[],
                                      ml_usecase='regression',
                                      numerical_features=[], target='aircraft',
                                      time_features=[])),
                ('imputer',
                 Simple_Imputer(categorical_strategy='most frequent',
                                fill_value_categorical='not_available',
                                fill_val…
                ('scaling', 'passthrough'), ('P_transform', 'passthrough'),
                ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
                ('cluster_all', 'passthrough'),
```

```
                ('dummy', Dummify(target='aircraft')),
                ('fix_perfect', 'passthrough'),
                ('clean_names', Clean_Colum_Names()),
                ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                ('dfs', 'passthrough'), ('pca', 'passthrough')],
         verbose=False)
INFO:logs:setup() successfully completed…
```

[374]:
```
<pycaret.internal.pycaret_experiment.time_series_experiment.TSForecastingExperim
ent at 0x22c1d8621f0>
```

[375]:
```
"""
Comparando modelos de ajuste
"""
best_model = compare_models()
```

```
<pandas.io.formats.style.Styler at 0x22c1cd34070>
```

```
INFO:logs:master_model_container: 30
INFO:logs:display_container: 2
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:compare_models() successfully
completed…
```

[376]:
```
"""
Definindo o modelo que ganhou na comparação
"""
auto_arima = create_model("auto_arima")
```

```
<pandas.io.formats.style.Styler at 0x22c1e48fa90>
```

```
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:create_model() successfully
completed…
```

[377]:
```
"""
Finalização do modelo
"""
final_aeronaves = finalize_model(auto_arima)
```

```
INFO:logs:Initializing finalize_model()
INFO:logs:finalize_model(self=<pycaret.internal.pycaret_experiment.time_series_e
xperiment.TSForecastingExperiment object at 0x0000022C1D8621F0>,
estimator=AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …), fit_kwargs=None,
groups=None, model_only=True, display=None)
INFO:logs:Finalizing AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:Initializing create_model()
INFO:logs:create_model(self=<pycaret.internal.pycaret_experiment.time_series_exp
eriment.TSForecastingExperiment object at 0x0000022C1D8621F0>,
estimator=AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …), fold=None, round=4,
cross_validation=True, predict=True, fit_kwargs={}, groups=None, refit=True,
probability_threshold=None, verbose=False, system=False,
add_to_model_list=False, metrics=None, display=None, kwargs={})
INFO:logs:Checking exceptions
INFO:logs:Importing libraries
INFO:logs:Copying training dataset
INFO:logs:Defining folds
INFO:logs:Declaring metric variables
INFO:logs:Importing untrained model
INFO:logs:Declaring custom model
INFO:logs:Auto ARIMA Imported successfully
INFO:logs:Starting cross validation
INFO:logs:Cross validating with ExpandingWindowSplitter(fh=array([1, 2, 3, 4,
5]), initial_window=None,
            step_length=5), n_jobs=1
```

```
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
INFO:logs:master_model_container: 31
INFO:logs:display_container: 4
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:create_model() successfully
completed…
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=8048, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:finalize_model() succesfully
completed…
```

[378]:
```
"""
Realizando 5 dias de previsão
"""
pred_aeronaves = predict_model(final_aeronaves, fh = 5)
pred_aeronaves = pd.DataFrame(pred_aeronaves, columns = ["date", "aircraft"])
pred_aeronaves["date"] = pred_aeronaves.index.to_timestamp()
pred_aeronaves = pred_aeronaves.loc[pred_aeronaves["aircraft"] > 0]
pred_aeronaves
```

[378]:

	date	aircraft
2022-04-19	2022-04-19	170.0192
2022-04-20	2022-04-20	173.0385
2022-04-21	2022-04-21	176.0577
2022-04-22	2022-04-22	179.0769
2022-04-23	2022-04-23	182.0962

[379]:
```
setup(helicopteros, fh = 5, fold = 6, seasonal_period = "D", n_jobs = -1,␣
↪use_gpu = True)
```

```
<pandas.io.formats.style.Styler at 0x22c1d74c1c0>
```

```
INFO:logs:self.master_model_container: 0
```

```
INFO:logs:self.display_container: 1
INFO:logs:Pipeline(memory=None,
         steps=[('dtypes',
                 DataTypes_Auto_infer(categorical_features=[],
                                      display_types=False, features_todrop=[],
                                      float_dtype='float64', id_columns=[],
                                      ml_usecase='regression',
                                      numerical_features=[],
                                      target='helicopter', time_features=[])),
                ('imputer',
                 Simple_Imputer(categorical_strategy='most frequent',
                                fill_value_categorical='not_available',
                                fill_v…
                ('scaling', 'passthrough'), ('P_transform', 'passthrough'),
                ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
                ('cluster_all', 'passthrough'),
                ('dummy', Dummify(target='helicopter')),
                ('fix_perfect', 'passthrough'),
                ('clean_names', Clean_Colum_Names()),
                ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                ('dfs', 'passthrough'), ('pca', 'passthrough')],
         verbose=False)
INFO:logs:setup() successfully completed…
```

[379]:
```
<pycaret.internal.pycaret_experiment.time_series_experiment.TSForecastingExperim
ent at 0x22c1e259c40>
```

[380]:
```
best_model = compare_models()
```

```
<pandas.io.formats.style.Styler at 0x22c1de2b580>
```

```
INFO:logs:master_model_container: 30
INFO:logs:display_container: 2
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:compare_models() successfully
completed…
```

[381]:
```
theta = create_model("theta")
```

```
<pandas.io.formats.style.Styler at 0x22c1de014f0>
```

```
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:create_model() successfully
completed…
```

[382]:
```
final_helicopter = finalize_model(theta)
```

```
INFO:logs:Initializing finalize_model()
```

```
INFO:logs:finalize_model(self=<pycaret.internal.pycaret_experiment.time_series_e
xperiment.TSForecastingExperiment object at 0x0000022C1E259C40>,
estimator=ThetaForecaster(deseasonalize=True, initial_level=None, sp=1),
fit_kwargs=None, groups=None, model_only=True, display=None)
INFO:logs:Finalizing ThetaForecaster(deseasonalize=True, initial_level=None,
sp=1)
INFO:logs:Initializing create_model()
INFO:logs:create_model(self=<pycaret.internal.pycaret_experiment.time_series_exp
eriment.TSForecastingExperiment object at 0x0000022C1E259C40>,
estimator=ThetaForecaster(deseasonalize=True, initial_level=None, sp=1),
fold=None, round=4, cross_validation=True, predict=True, fit_kwargs={},
groups=None, refit=True, probability_threshold=None, verbose=False,
system=False, add_to_model_list=False, metrics=None, display=None, kwargs={})
INFO:logs:Checking exceptions
INFO:logs:Importing libraries
INFO:logs:Copying training dataset
INFO:logs:Defining folds
INFO:logs:Declaring metric variables
INFO:logs:Importing untrained model
INFO:logs:Declaring custom model
INFO:logs:Theta Forecaster Imported successfully
INFO:logs:Starting cross validation
INFO:logs:Cross validating with ExpandingWindowSplitter(fh=array([1, 2, 3, 4,
5]), initial_window=None,
            step_length=5), n_jobs=1
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
INFO:logs:master_model_container: 31
INFO:logs:display_container: 4
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:create_model() successfully
completed…
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:finalize_model() succesfully
completed…
```

[383]:
```
pred_helicopter = predict_model(final_helicopter, fh = 5)
pred_helicopter = pd.DataFrame(pred_helicopter, columns = ["date",␣
 ↪"helicopter"])
pred_helicopter["date"] = pred_helicopter.index.to_timestamp()
pred_helicopter = pred_helicopter.loc[pred_helicopter["helicopter"] > 0]
pred_helicopter
```

[383]:
```
                 date  helicopter
2022-04-19 2022-04-19    149.4547
2022-04-20 2022-04-20    150.6820
2022-04-21 2022-04-21    151.9094
2022-04-22 2022-04-22    153.1367
2022-04-23 2022-04-23    154.3641
```

[384]:
```
setup(tanques, fh = 5, fold = 6, seasonal_period = "D", n_jobs = -1, use_gpu =␣
↪True)
```

```
<pandas.io.formats.style.Styler at 0x22c1dddd940>
```

```
INFO:logs:self.master_model_container: 0
INFO:logs:self.display_container: 1
INFO:logs:Pipeline(memory=None,
         steps=[('dtypes',
                 DataTypes_Auto_infer(categorical_features=[],
                                      display_types=False, features_todrop=[],
                                      float_dtype='float64', id_columns=[],
                                      ml_usecase='regression',
                                      numerical_features=[], target='tank',
                                      time_features=[])),
                ('imputer',
                 Simple_Imputer(categorical_strategy='most frequent',
                                fill_value_categorical='not_available',
                                fill_value_n…
                ('scaling', 'passthrough'), ('P_transform', 'passthrough'),
                ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
                ('cluster_all', 'passthrough'),
                ('dummy', Dummify(target='tank')),
                ('fix_perfect', 'passthrough'),
                ('clean_names', Clean_Colum_Names()),
                ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                ('dfs', 'passthrough'), ('pca', 'passthrough')],
         verbose=False)
INFO:logs:setup() successfully completed…
```

[384]:
```
<pycaret.internal.pycaret_experiment.time_series_experiment.TSForecastingExperim
ent at 0x22c1b8c6250>
```

[385]:
```
best_model = compare_models()
```

```
<pandas.io.formats.style.Styler at 0x22c1d77e610>
```

```
INFO:logs:master_model_container: 30
INFO:logs:display_container: 2
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
```

```
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:compare_models() successfully
completed…
```

[386]:
```
auto_arima = create_model("auto_arima")
```

```
<pandas.io.formats.style.Styler at 0x22c19d19f40>
```

```
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:create_model() successfully
completed…
```

[387]:
```
final_tanques = finalize_model(auto_arima)
```

```
INFO:logs:Initializing finalize_model()
INFO:logs:finalize_model(self=<pycaret.internal.pycaret_experiment.time_series_e
xperiment.TSForecastingExperiment object at 0x0000022C1B8C6250>,
estimator=AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …), fit_kwargs=None,
groups=None, model_only=True, display=None)
INFO:logs:Finalizing AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:Initializing create_model()
INFO:logs:create_model(self=<pycaret.internal.pycaret_experiment.time_series_exp
eriment.TSForecastingExperiment object at 0x0000022C1B8C6250>,
estimator=AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
```

```
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …), fold=None, round=4,
cross_validation=True, predict=True, fit_kwargs={}, groups=None, refit=True,
probability_threshold=None, verbose=False, system=False,
add_to_model_list=False, metrics=None, display=None, kwargs={})
INFO:logs:Checking exceptions
INFO:logs:Importing libraries
INFO:logs:Copying training dataset
INFO:logs:Defining folds
INFO:logs:Declaring metric variables
INFO:logs:Importing untrained model
INFO:logs:Declaring custom model
INFO:logs:Auto ARIMA Imported successfully
INFO:logs:Starting cross validation
INFO:logs:Cross validating with ExpandingWindowSplitter(fh=array([1, 2, 3, 4,
5]), initial_window=None,
            step_length=5), n_jobs=1
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
INFO:logs:master_model_container: 31
INFO:logs:display_container: 4
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:create_model() successfully
completed…
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:AutoARIMA(D=None, alpha=0.05, d=None, error_action='warn',
          information_criterion='aic', max_D=1, max_P=2, max_Q=2, max_d=2,
          max_order=5, max_p=5, max_q=5, maxiter=50, method='lbfgs', n_fits=10,
          n_jobs=1, offset_test_args=None, out_of_sample_size=0, random=False,
          random_state=636, scoring='mse', scoring_args=None, seasonal=True,
          seasonal_test='ocsb', seasonal_test_args=None, sp=1, start_P=1,
          start_Q=1, start_p=2, start_params=None, …)
INFO:logs:finalize_model() succesfully
completed…
```

[388]:
```
pred_tanques = predict_model(final_tanques, fh = 5)
pred_tanques = pd.DataFrame(pred_tanques, columns = ["date", "tank"])
pred_tanques["date"] = pred_tanques.index.to_timestamp()
pred_tanques = pred_tanques.loc[pred_tanques["tank"] > 0]
pred_tanques
```

[388]:

	date	tank
2022-04-19	2022-04-19	793.6136
2022-04-20	2022-04-20	800.6231
2022-04-21	2022-04-21	814.3762
2022-04-22	2022-04-22	822.3043
2022-04-23	2022-04-23	829.0610

[389]:
```
setup(automoveis_militares, fh= 5, fold = 6, seasonal_period = "D", n_jobs =␣
↪-1, use_gpu=True)
```

```
<pandas.io.formats.style.Styler at 0x22c1d1b7f10>
```

```
INFO:logs:self.master_model_container: 0
INFO:logs:self.display_container: 1
INFO:logs:Pipeline(memory=None,
         steps=[('dtypes',
                 DataTypes_Auto_infer(categorical_features=[],
                                      display_types=False, features_todrop=[],
                                      float_dtype='float64', id_columns=[],
                                      ml_usecase='regression',
                                      numerical_features=[],
                                      target='military auto',
                                      time_features=[])),
                ('imputer',
                 Simple_Imputer(categorical_strategy='most frequent',
                                fill_value_categorical='not_available',
                                fill…
                ('scaling', 'passthrough'), ('P_transform', 'passthrough'),
                ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
                ('cluster_all', 'passthrough'),
                ('dummy', Dummify(target='military auto')),
                ('fix_perfect', 'passthrough'),
                ('clean_names', Clean_Colum_Names()),
                ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                ('dfs', 'passthrough'), ('pca', 'passthrough')],
         verbose=False)
INFO:logs:setup() successfully completed…
```

[389]:
```
<pycaret.internal.pycaret_experiment.time_series_experiment.TSForecastingExperim
ent at 0x22c1cc745b0>
```

[390]:
```
best_model = compare_models()
```

```
<pandas.io.formats.style.Styler at 0x22c1da1e520>

INFO:logs:master_model_container: 30
INFO:logs:display_container: 2
INFO:logs:BaseCdsDtForecaster(degree=1, deseasonal_model='additive',
                    regressor=HuberRegressor(alpha=0.0001, epsilon=1.35,
                                             fit_intercept=True, max_iter=100,
                                             tol=1e-05, warm_start=False),
                    sp=1, window_length=1)
INFO:logs:compare_models() successfully
completed…
```

[391]:
```
huber_cds_dt = create_model("huber_cds_dt")
```

```
<pandas.io.formats.style.Styler at 0x22c1da2d3a0>

INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:BaseCdsDtForecaster(degree=1, deseasonal_model='additive',
                    regressor=HuberRegressor(alpha=0.0001, epsilon=1.35,
                                             fit_intercept=True, max_iter=100,
                                             tol=1e-05, warm_start=False),
                    sp=1, window_length=1)
INFO:logs:create_model() successfully
completed…
```

[392]:
```
final_automoveis = finalize_model("huber_cds_dt")
```

```
INFO:logs:Initializing finalize_model()
INFO:logs:finalize_model(self=<pycaret.internal.pycaret_experiment.time_series_e
xperiment.TSForecastingExperiment object at 0x0000022C1CC745B0>,
estimator=huber_cds_dt, fit_kwargs=None, groups=None, model_only=True,
display=None)
INFO:logs:Finalizing huber_cds_dt
INFO:logs:Initializing create_model()
INFO:logs:create_model(self=<pycaret.internal.pycaret_experiment.time_series_exp
eriment.TSForecastingExperiment object at 0x0000022C1CC745B0>,
estimator=huber_cds_dt, fold=None, round=4, cross_validation=True, predict=True,
fit_kwargs={}, groups=None, refit=True, probability_threshold=None,
verbose=False, system=False, add_to_model_list=False, metrics=None,
display=None, kwargs={})
INFO:logs:Checking exceptions
INFO:logs:Importing libraries
INFO:logs:Copying training dataset
INFO:logs:Defining folds
INFO:logs:Declaring metric variables
INFO:logs:Importing untrained model
INFO:logs:Huber w/ Cond. Deseasonalize & Detrending Imported successfully
INFO:logs:Starting cross validation
INFO:logs:Cross validating with ExpandingWindowSplitter(fh=array([1, 2, 3, 4,
```

```
5]), initial_window=None,
            step_length=5), n_jobs=1
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
INFO:logs:master_model_container: 31
INFO:logs:display_container: 4
INFO:logs:BaseCdsDtForecaster(degree=1, deseasonal_model='additive',
                    regressor=HuberRegressor(alpha=0.0001, epsilon=1.35,
                                             fit_intercept=True, max_iter=100,
                                             tol=1e-05, warm_start=False),
                    sp=1, window_length=1)
INFO:logs:create_model() successfully
completed…
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:BaseCdsDtForecaster(degree=1, deseasonal_model='additive',
                    regressor=HuberRegressor(alpha=0.0001, epsilon=1.35,
                                             fit_intercept=True, max_iter=100,
                                             tol=1e-05, warm_start=False),
                    sp=1, window_length=1)
INFO:logs:finalize_model() succesfully
completed…
```

[393]:
```
pred_automoveis = predict_model(final_automoveis, fh = 5)
pred_automoveis = pd.DataFrame(pred_automoveis, columns = ["date", "military␣
 ↪auto"])
pred_automoveis["date"] = pred_automoveis.index.to_timestamp()
pred_automoveis = pred_automoveis.loc[pred_automoveis["military auto"] > 0]
pred_automoveis
```

[393]:

	date	military auto
2022-04-19	2022-04-19	1505.2650
2022-04-20	2022-04-20	1523.4064
2022-04-21	2022-04-21	1541.4223
2022-04-22	2022-04-22	1559.3112
2022-04-23	2022-04-23	1577.0712

[394]:
```
setup(pessoas, fh = 5, fold = 6, seasonal_period="D", n_jobs=-1, use_gpu=True)
```

```
<pandas.io.formats.style.Styler at 0x22c1e4bffa0>
```

```
INFO:logs:self.master_model_container: 0
INFO:logs:self.display_container: 1
INFO:logs:Pipeline(memory=None,
         steps=[('dtypes',
                 DataTypes_Auto_infer(categorical_features=[],
                                      display_types=False, features_todrop=[],
```

```
                                    float_dtype='float64', id_columns=[],
                                    ml_usecase='regression',
                                    numerical_features=[], target='personnel',
                                    time_features=[])),
                ('imputer',
                 Simple_Imputer(categorical_strategy='most frequent',
                                fill_value_categorical='not_available',
                                fill_va…
                ('scaling', 'passthrough'), ('P_transform', 'passthrough'),
                ('binn', 'passthrough'), ('rem_outliers', 'passthrough'),
                ('cluster_all', 'passthrough'),
                ('dummy', Dummify(target='personnel')),
                ('fix_perfect', 'passthrough'),
                ('clean_names', Clean_Colum_Names()),
                ('feature_select', 'passthrough'), ('fix_multi', 'passthrough'),
                ('dfs', 'passthrough'), ('pca', 'passthrough')],
         verbose=False)
INFO:logs:setup() successfully completed…
```

[394]:
```
<pycaret.internal.pycaret_experiment.time_series_experiment.TSForecastingExperim
ent at 0x22c1df53760>
```

[395]:
```
best_model = compare_models()
```

```
<pandas.io.formats.style.Styler at 0x22c1ce3bdc0>
```

```
INFO:logs:master_model_container: 30
INFO:logs:display_container: 2
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:compare_models() successfully
completed…
```

[396]:
```
theta = create_model("theta")
```

```
<pandas.io.formats.style.Styler at 0x22c1d73da00>
```

```
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:create_model() successfully
completed…
```

[397]:
```
final_pessoas = finalize_model(theta)
```

```
INFO:logs:Initializing finalize_model()
INFO:logs:finalize_model(self=<pycaret.internal.pycaret_experiment.time_series_e
xperiment.TSForecastingExperiment object at 0x0000022C1DF53760>,
estimator=ThetaForecaster(deseasonalize=True, initial_level=None, sp=1),
fit_kwargs=None, groups=None, model_only=True, display=None)
INFO:logs:Finalizing ThetaForecaster(deseasonalize=True, initial_level=None,
```

```
sp=1)
INFO:logs:Initializing create_model()
INFO:logs:create_model(self=<pycaret.internal.pycaret_experiment.time_series_exp
eriment.TSForecastingExperiment object at 0x0000022C1DF53760>,
estimator=ThetaForecaster(deseasonalize=True, initial_level=None, sp=1),
fold=None, round=4, cross_validation=True, predict=True, fit_kwargs={},
groups=None, refit=True, probability_threshold=None, verbose=False,
system=False, add_to_model_list=False, metrics=None, display=None, kwargs={})
INFO:logs:Checking exceptions
INFO:logs:Importing libraries
INFO:logs:Copying training dataset
INFO:logs:Defining folds
INFO:logs:Declaring metric variables
INFO:logs:Importing untrained model
INFO:logs:Declaring custom model
INFO:logs:Theta Forecaster Imported successfully
INFO:logs:Starting cross validation
INFO:logs:Cross validating with ExpandingWindowSplitter(fh=array([1, 2, 3, 4,
5]), initial_window=None,
            step_length=5), n_jobs=1
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
INFO:logs:master_model_container: 31
INFO:logs:display_container: 4
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:create_model() successfully
completed…
INFO:logs:master_model_container: 31
INFO:logs:display_container: 3
INFO:logs:ThetaForecaster(deseasonalize=True, initial_level=None, sp=1)
INFO:logs:finalize_model() succesfully
completed…
```

[398]:
```
pred_pessoas = predict_model(final_pessoas, fh = 5)
pred_pessoas = pd.DataFrame(pred_pessoas, columns = ["date", "personnel"])
pred_pessoas["date"] = pred_pessoas.index.to_timestamp()
pred_pessoas = pred_pessoas.loc[pred_pessoas["personnel"] > 0]
pred_pessoas
```

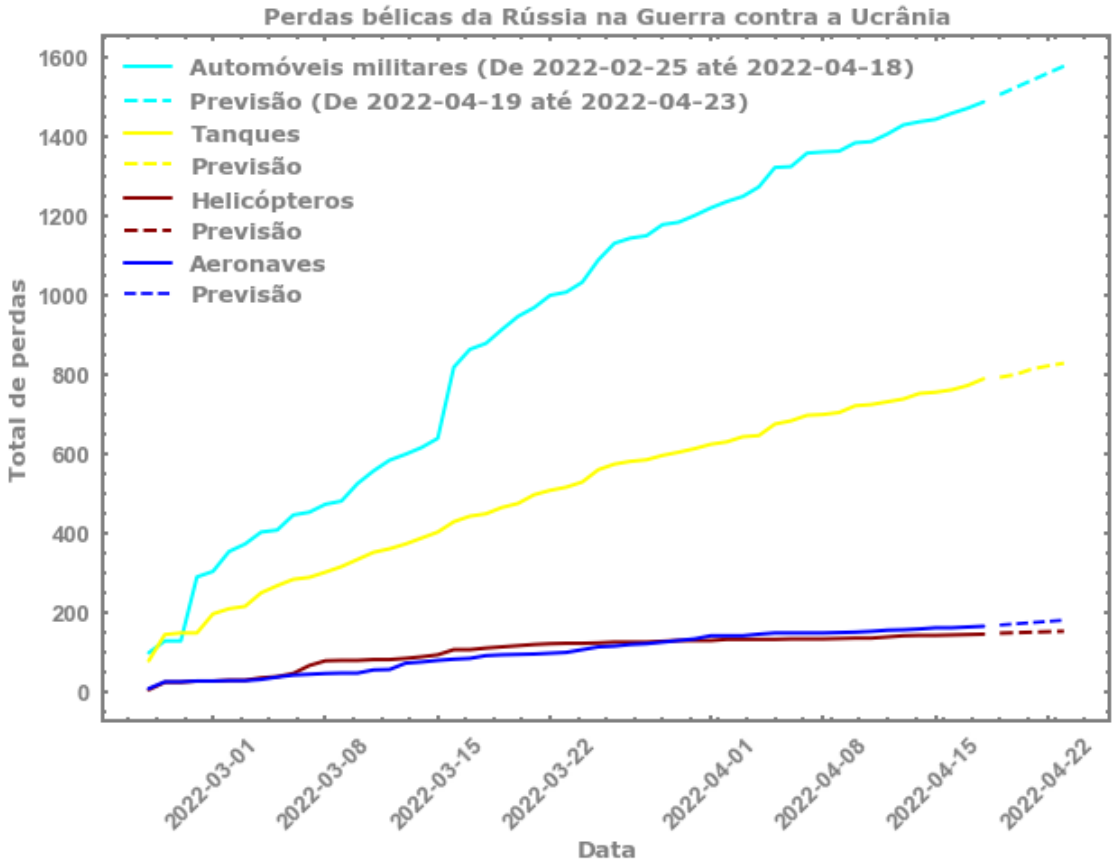
[398]:

	date	personnel
2022-04-19	2022-04-19	20894.2440
2022-04-20	2022-04-20	21041.7478
2022-04-21	2022-04-21	21189.2517
2022-04-22	2022-04-22	21336.7556
2022-04-23	2022-04-23	21484.2594

7 Plotando as previsões

[399]:
```
"""
Criando um "local" para alocar a nossa figura
"""
fig, axs = plt.subplots(figsize = (10, 7))
"""
Plot das curvas
"""
axs.plot(Dados["date"], Dados["military auto"], linewidth = 2, label =␣
↪"Automóveis militares (De 2022-02-25 até 2022-04-18)", color = "cyan")
axs.plot(pred_automoveis["date"], pred_automoveis["military auto"], "--g",␣
↪linewidth = 2, label = "Previsão (De 2022-04-19 até 2022-04-23)", color =␣
↪"cyan")
axs.plot(Dados["date"], Dados["tank"], linewidth = 2, label = "Tanques", color␣
↪= "yellow")
axs.plot(pred_tanques["date"], pred_tanques["tank"], "--g", linewidth = 2,␣
↪label = "Previsão", color = "yellow")
axs.plot(Dados["date"], Dados["helicopter"], linewidth = 2, label =␣
↪"Helicópteros", color = "darkred")
axs.plot(pred_helicopter["date"], pred_helicopter["helicopter"], "--g",␣
↪linewidth = 2, label = "Previsão", color = "darkred")
axs.plot(Dados["date"], Dados["aircraft"], linewidth = 2, label = "Aeronaves",␣
↪color = "blue")
axs.plot(pred_aeronaves["date"], pred_aeronaves["aircraft"], "--g", label =␣
↪"Previsão", color = "blue")
"""
Grid = False
"""
axs.grid(False)
"""
Definindo a "grossura" e a cor do eixos
"""
for axis in ["left", "top", "right", "bottom"]:
    axs.spines[axis].set_linewidth(2)
    axs.spines[axis].set_color("gray")
"""
Trabalha com os ticks do gráfico
"""
axs.xaxis.set_minor_locator(AutoMinorLocator())
axs.yaxis.set_minor_locator(AutoMinorLocator())
axs.tick_params(axis = "both", direction = "in", labelcolor = "gray", labelsize␣
↪= 13, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='minor', direction = "in", length=2, color='gray', width␣
↪= 2, top = True, right = True, left = True, bottom = True)
```

```
axs.tick_params(which='major', direction = "in", color='gray', length=3.4,␣
 ↪width = 2, top = True, right = True, left = True, bottom = True)
"""
Rotacionando o label do eixo x
"""
plt.xticks(rotation=45)
"""
Definindo um intervalo para o eixo x do gráfico
"""
"""
Legenda da figura
"""
plt.legend(frameon = False, prop = font2, labelcolor = "gray")
"""
Tudo em negrito
"""
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"
"""
Labels
"""
axs.set_xlabel("Data", fontdict = font1)
axs.set_ylabel("Total de perdas", fontdict = font1)
"""
Fundo branco
"""
fig.patch.set_facecolor("white")
"""
Título da figura
"""
axs.set_title("Perdas bélicas da Rússia na Guerra contra a Ucrânia", fontdict =␣
 ↪font1)
plt.show()
```



[400]:
```
fig, axs = plt.subplots(figsize = (10, 7))
axs.plot(Dados["date"], Dados["personnel"], linewidth = 2, color = "cyan",␣
 ↪label = "Dados originais (De 2022-02-25 até 2022-04-18)")
axs.plot(pred_pessoas["date"], pred_pessoas["personnel"], "--g", linewidth = 2,␣
 ↪color = "cyan", label = "Previsão (De 2022-04-19 até 2022-04-23)")
axs.grid(False)
for axis in ["left", "top", "right", "bottom"]:
    axs.spines[axis].set_linewidth(2)
    axs.spines[axis].set_color("gray")
axs.xaxis.set_minor_locator(AutoMinorLocator())
axs.yaxis.set_minor_locator(AutoMinorLocator())
axs.tick_params(axis = "both", direction = "in", labelcolor = "gray", labelsize␣
 ↪= 13, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='minor', direction = "in", length=2, color='gray', width␣
 ↪= 2, top = True, right = True, left = True, bottom = True)
axs.tick_params(which='major', direction = "in", color='gray', length=3.4,␣
 ↪width = 2, top = True, right = True, left = True, bottom = True)
plt.xticks(rotation = 45)
plt.rcParams["font.weight"] = "bold"
```

```
plt.rcParams["axes.labelweight"] = "bold"
axs.set_xlabel("Data", fontdict = font1)
axs.set_ylabel("Total de mortes russas", fontdict = font1)
plt.legend(frameon = False, prop = font2, labelcolor = "gray")
fig.patch.set_facecolor("white")
plt.show()
```