PARTE 1

INFORMAÇÕES INICIAIS

INSTALA BIBLIOTECAS

IMPORTA BIBLIOTECAS

DEFINE VARIÁVEIS

```
#
#  Informações
#
#  Os dados recuperados serão desde 01-dezembro-2016 até 31-março-2022
#  Base de treinamento: 01-jan-2017 a 31-dez-2020 (4 anos, 80% dos dados)
#  Base de teste: 01-jan-2021 a 31-dez-2021 (1 ano, 20% dos dados)
#  Base de validação: 01-jan-2022 a 31-mar-2022 (3 meses)
#
#  Colocar com os dados em percentual de variação diária (normalização dos dados)
#  Testar para ver se vai funcionar melhor
#
#  Os dados de índices e da base de Dólar serão extraídos do Yahoo Finance
#  Os dados de médias móveis, RSL, volatilidade e variação % serão calculados no código.
#  Titulo Brasil de 10 anos será extraído do Investing.com
#
```

```
#
#  instalando as bibliotecas
#

!pip install investpy             ## Biblioteca para recuperar dados do site Investing.com
!pip install yfinance             ## Biblioteca para recuperar dados do site Yahoo Finance
!pip install pandas_datareader    ## Biblioteca para coletar dados da web , sites diversos, yaho

!pip install pmdarima
```

```
#
#
#    Importando as bibliotecas
#
#

import pandas as pd
import pandas_datareader.data as web
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import datetime as dt
```

```
# Biblioteca para obter dados do Investing.com
import investpy as inv

# Biblioteca para obter dados do Yahoo Finance
import yfinance as yf
yf.pdr_override()

import math
from sklearn.metrics import mean_squared_error

# Bibiotecas para mmodelo LSTM
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from sklearn.feature_selection import SelectKBest

# Bibliotecas para modelo ARIMA
import pmdarima as pm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.stattools import adfuller
from pmdarima.arima.utils import ndiffs
from pmdarima.metrics import smape

import seaborn as sns
```

PARTE 2

COLETA DE DADOS NA WEB

```
#
#   Os dados históricos estarão no timeframe diário
#
#   Os seguintes dados serão recuperados do site Yahoo Finance:
#       1) Dolar,
#       2) Ibovespa,
#       3) Titulo Publico Norteamericano de 10 anos (USBond 10Y),
#       4) Índice S&P500,
#       5) Índice de Volatilidade do SP500 (Índice VIX),
#       6) Dollar Index (paridade entre o dolar e uma cesta de moedas),
#       7) Futuro do Ouro,
#       8) Petróleo WTI
#
#   Os seguintes dados serão recuperados do site Investing.com:
#       1) Título Público Brasil 10 anos
#
#   Os indicadores de Média Móvel, Volatilidade, Variação % e RSL do Dolar
#   serão calculados no código e inseridos como colunas no dataframe relativo ao Dolar
#



#
#   A base de treinamento tem início em 01/01/2017, porem o histórico de cotação
#   do dólar será recuperado a partir do dia 01/12/2016 para que possamos ter
```

```
#   pelo menos 17 dias de dados antes do início do treinamento para fazer o cálculo
#   da média móvel de 17 dias a partir do dia 01/01/2017.
#
#   Os dados da base de validação são de 01/01/2022 até 31/03/2022, a extração busca o dado até
#
#

data_inicio = dt.datetime(2016,12,1)
data_fim = dt.datetime(2022,4,1)



#
#      Recupera os dados do Yahoo Finance
#

# DOLAR
df_dolar = web.DataReader('BRL=x', data_source='yahoo', start=data_inicio, end=data_fim)

# IBOVESPA
df_ibovespa = web.get_data_yahoo('^BVSP', start=data_inicio, end=data_fim)

# US BOND 10Y (Título Publico norteamericano 10 anos)
df_bondUS = web.get_data_yahoo('^TNX', start=data_inicio, end=data_fim)

# ÍNDICE S&P500
df_sp500 = web.get_data_yahoo('^GSPC', start=data_inicio, end=data_fim)

# ÍNDICE VIX
df_vix = web.get_data_yahoo('^VIX', start=data_inicio, end=data_fim)

# DOLLAR INDEX
df_index = web.get_data_yahoo('DX-Y.NYB', start=data_inicio, end=data_fim)

# OURO FUTURO
df_gold = web.get_data_yahoo('GC=F', start=data_inicio, end=data_fim)

# PETROLEO BRENT
df_brent = web.get_data_yahoo('BZ=F', start=data_inicio, end=data_fim)


# TODOS OS TICKERS MENOS O DOLAR
#tickers = ['^BVSP', '^TNX', '^GSPC', '^VIX', 'DX-Y.NYB', 'GC=F', 'BZ=F']
#df_dados = web.get_data_yahoo(tickers, start=data_inicio, end=data_fim)




#
#      Recupera os dados do Investing.com
#

# Título Público Brasil 10 anos
df_bondBR = inv.get_bond_historical_data('Brazil 10Y', from_date='01/12/2016', to_date='01/04/20
```

PARTE 3

TRATAMENTO DOS DADOS RECUPERADOS

CRIAÇÃO DO DATASET

```
#   a) Primeira verificação dos dados retornados e quantidade de linhas, datas inicial e final
#       dataset

#   b) Verificar se o tipo do campo está correto e na formatação correta, como
#      pontos e vigulas para valores decimais, formato de data, e tipo de campo
#      numérico, inteiro, float, string, etc.;
#       dataset.info()

#   c) Verificar se os dados foram obtidos integralmente, ou seja, se todos os dados
#      foram recuperados com valores numéricos e se há campos em branco ou nulos;
#       dataset.isna().any()

#   d) Calcular e incluir as colunas com os indicadores técnicos do Dolar: Média Móvel, Volatili

#   e) Montar o dataset que será utilizado pelos modelos de Machine Learning com os dados dos ou

#   f) Tratamento das celulas com valores nulos ou branco em função das datas em que não houve n
#      (diferenças de feriados e dias úteis entre Brasil e Estados Unidos, por exemplo)

#   g) Primeira visualização gráfica dos dados do Dolar (preço de fechamento) e indicadores
```

Seção 3.1 - Verificação dos dados carregados (itens a, b, c)

Verificação dos dados carregados

[ ] ↳ *30 células ocultas*

Seção 3.2 - Montagem do dataset modelo (itens d, e, f, g)

Criação dos parâmetros técnicos MM7, MM17, Volatilidade, Variação % e RSL

Inclusão no dataset da coluna ALVO que é o preço alvo do Dolar a ser previsto para o dia seguinte

Montar o dataset agrupando os dados dos ativos

Trata as celulas com valores branco e/ou nulo

Plota o gráfico de dados históricos do Dolar

[ ] ↳ *22 células ocultas*

PARTE 4

ANÁLISE E EXPLORAÇÃO

PRÉ PROCESSAMENTO DOS DADOS

```
#   Definição do tamanho dos dados de teste, treinamento e validação
#   Base de treinamento = anos 2017 a 2020 (4 anos = 80%)
#   Base de testes = ano 2021 (1 ano = 20%)
#   Base de validação = janeiro de 2022 (1 mês)

#   Separação dos dados

ds_dados = ds_modelo.loc['2017-01-01':'2021-12-31']

ds_treino = ds_dados.loc['2017-01-01':'2020-12-31']

ds_teste = ds_dados.loc['2021-01-01':'2021-12-31']

ds_validacao = ds_modelo.loc['2022-01-01':'2022-03-31']



print (len(ds_modelo))
print(len(ds_dados))
print(len(ds_treino))
print(len(ds_teste))
print(len(ds_validacao))


sns.heatmap(ds_dados.drop(['MM7', 'MM17', 'Variação', 'Volatilidade', 'RSL', 'Alvo'], 1).corr(),
#sns.heatmap(ds_dados.drop(['MM7', 'MM17', 'Variação', 'Volatilidade', 'RSL', 'Alvo'], 1).corr()


#   Plota o gráfico dos dados de Volatilidade
#   dados de treinamento e teste (2017-2021)
plt.figure(figsize=(16,8))
plt.plot(ds_dados['Volatilidade'])


#   Plota o gráfico dos dados de Variação %
#   dados de treinamento e teste (2017-2021)
plt.figure(figsize=(16,8))
plt.plot(ds_dados['Variação']*100)


#   Plota o histograma da volatilidade do Dolar durante o período dos
#   dados de treinamento e teste (2017-2021)
plt.figure(figsize=(16,8))
plt.hist(ds_dados['Volatilidade'], bins=100)


#   Plota o histograma da variação percentual do Dolar durante o período dos
#   dados de treinamento e teste (2017-2021)
plt.figure(figsize=(16,8))
plt.hist(ds_dados['Variação'], bins=100)


#   Plota o histograma do valor do Dolar durante o período dos
#   dados de treinamento e teste (2017-2021)
```

```
plt.figure(figsize=(16,8))
plt.hist(ds_dados['Dolar'], bins=100)
```

```
# Separando as features e labels para escolha das melhores
# Escolhendo as melhores features com Kbest

features = ds_dados
labels = ds_dados.Alvo

features_list = ('Dolar', 'MM7','MM17','Volatilidade','Variação','RSL','Ibovespa','Brent','Bond

k_best_features = SelectKBest(k='all')
k_best_features.fit_transform(features, labels)
k_best_features_scores = k_best_features.scores_
raw_pairs = zip(features_list[1:], k_best_features_scores)
ordered_pairs = list(reversed(sorted(raw_pairs, key=lambda x: x[1])))

k_best_features_final = dict(ordered_pairs[:15])
best_features = k_best_features_final.keys()
print ('')
print ("Melhores features:")
print (k_best_features_final)
```

```
#  Separando as features escolhidas
#features = ds_dados.loc[:,['Volatilidade','Brent','Variação']]   ## label=Dolar, Feature com Al

features = ds_dados.loc[:,['MM17','MM7','Volatilidade']]   ## label=Alvo, Feature com Dolar fech
```

```
features
```

PARTE 5

CRIAÇÃO DOS MODELOS DE MACHINE LEARNING

a) LSTM

b) ARIMA

5.a) LSTM

```
dias_previsao = 60
```

```
# Normalização dos dados entre 0 e 1 com o MinMaxScaler

sc = MinMaxScaler(feature_range=(0,1))
```

```
#   Como o modelo usa somente 1 dado de entrada, esta será o valor Dolar que é
#   o preço de fechamento do dia
#   Cria os arrays com os dados em escala
```

```
sc_treino = sc.fit_transform(ds_treino['Dolar'].values.reshape(-1,1))
sc_teste = sc.fit_transform(ds_teste['Dolar'].values.reshape(-1,1))



print(sc_treino.shape)
print(sc_teste.shape)


#       Dados de treinamento

#       X = valor Dolar no dia
#       Y = alvo da previsão
X_treino=[]
Y_treino=[]

for i in range(len(sc_treino)-dias_previsao):
        X_treino.append(sc_treino[i:i+dias_previsao, 0])
        Y_treino.append(sc_treino[i+dias_previsao, 0])

X_treino, Y_treino = np.array(X_treino), np.array(Y_treino)

#       Dados de teste

X_teste = []
Y_teste = []

for i in range(len(sc_teste)-dias_previsao):
        X_teste.append(sc_teste[i:i+dias_previsao, 0])
        Y_teste.append(sc_teste[i+dias_previsao, 0])

X_teste, Y_teste = np.array(X_teste), np.array(Y_teste)



# Reshape input para ser [dados, time steps, features] que é requerido pelo LSTM
X_treino = X_treino.reshape(X_treino.shape[0],X_treino.shape[1] , 1)
X_teste = X_teste.reshape(X_teste.shape[0],X_teste.shape[1] , 1)


X_treino.shape[1]


#   Criação do modelo LSTM

modelo = Sequential()

modelo.add(LSTM(units = 50, return_sequences=True, input_shape=(X_treino.shape[1],1)))
modelo.add(Dropout(0.2))

modelo.add(LSTM(units = 50, return_sequences=True))
modelo.add(Dropout(0.2))

modelo.add(LSTM(units = 50))
modelo.add(Dropout(0.2))

modelo.add(Dense(units=1, activation='relu'))

modelo.summary()
```

```
#   Treinar o modelo com otimizador "adam" e a função de perda "mean squared error"

modelo.compile(optimizer='adam', loss='mean_squared_error')


modelo.fit(X_treino, Y_treino, validation_data=(X_teste,Y_teste), epochs=50, batch_size=64)


perda = modelo.history.history['loss']
plt.plot(perda)
plt.xlabel('Epoch')
plt.ylabel('Perda')
plt.title('Perda do Modelo de Treino')
plt.show()


#   Faz a previsão e verifica métricas de performance
previsao_treino=modelo.predict(X_treino)
previsao_teste=modelo.predict(X_teste)


#   Transformação de volta ao formato original
previsao_treino = sc.inverse_transform(previsao_treino)
previsao_teste = sc.inverse_transform(previsao_teste)


#   Calculate a métrica de performance RMSE para dados de treino e de teste
print(math.sqrt(mean_squared_error(Y_treino, previsao_treino)))
print(math.sqrt(mean_squared_error(Y_teste, previsao_teste)))


#Testes com modelo LSTM:

#units          | dropout | activation | Layers | epochs | batch_size | MSE_treino | MSE_teste |
#50,60,80,120     2,3,4,5       relu            4         100          64              4.88
#50,60,80,120   2,3,4,5     relu          4        50          64
#50,50,50         2,2,2         relu            3          50          64               4.74
#50,50,50         2,2,2         linear        3        25          32              4.90
#50,50            2,2             linear          2        25          32               4.73
#50,50          2,2           relu          2          25          32              4.76
#50,50          2,2           relu          2          25          64              4.72
#50,50,50         2,2,2         relu            3          25          64               4.73
#50,50          2,2           relu          2          50          64              4.74
#50,50          2,2           relu          2          25         128            4.74        4.
#50,50          3,3           relu          2          25          64              4.72
#50,50,50         2,2,2         relu            3          25          32               4.90
#50,50,50         2,2,2         relu            3         100          64           4.73

#optimizer=Adam
#dropout = 0.2
#units = 50
#layers = 3
#activ = relu
#batch = 64
#epoch = 50 (25 e 100 variou pouco)


plt.plot(ds_teste['Dolar'][len(ds_teste)-len(previsao_teste):].values, color='blue')
plt.plot(previsao_teste, color='red')
```

```
plt.title('Previsão Dólar')
plt.xlabel('Dias')
plt.ylabel('Preço')
plt.legend()
plt.show()


#   Faz previsão no dataset de validação, dados ainda desconhecidos do modelo


#ds_validacao
dados_validacao = pd.concat((ds_dados['Dolar'].tail(dias_previsao), ds_validacao['Dolar']), axis
dados_validacao


sc_validacao = sc.fit_transform(dados_validacao.values.reshape(-1,1))


X_valida = []

for i in range (dias_previsao, len(sc_validacao)):
    X_valida.append(sc_validacao[i-dias_previsao:i, 0])



X_valida = np.array(X_valida)
X_valida = X_valida.reshape(X_valida.shape[0], X_valida.shape[1] , 1)


previsao_validacao = modelo.predict(X_valida)
previsao_validacao = sc.inverse_transform(previsao_validacao)


dados_validacao


plt.plot(previsao_validacao, color='red', label='Previsão')
plt.plot(np.array(dados_validacao.tail(len(previsao_validacao))), color='blue', label='Valor Dol
plt.title('Previsão Dólar')
plt.xlabel('Dias')
plt.ylabel('Preço')
plt.legend()
plt.show()


#   Fazer previsão para ao longo de 60 dias ao invés de 1 dia de cada vez
#   Validar na base de validação (01/01/2022 a 31/03/2022)

print(len(ds_validacao))


#   Usar os últimos 100 dias da base de testes (timestep=100), pois base de validação começa a p
#   Usar 100 desta vez no timesetp para não confundir com os 60 dias de previsão

fut_inp = sc_teste[len(ds_teste)-100:]
fut_inp = fut_inp.reshape(1,-1)
tmp_inp = list(fut_inp)


fut_inp.shape
```

```
#   Cria lista dos últimos 100 dados -> é o timestep
tmp_inp = tmp_inp[0].tolist()


#   Previsão para os próximos 60 dias usando os dados correntes (gerados pela previsão)

lst_output=[]
n_steps=100
i=0
while(i<60):       # este 60 aqui é a qtde de dia para frente que quero prever

    if(len(tmp_inp)>100):
        fut_inp = np.array(tmp_inp[1:])
        fut_inp=fut_inp.reshape(1,-1)
        fut_inp = fut_inp.reshape((1, n_steps, 1))
        yhat = modelo.predict(fut_inp, verbose=0)
        tmp_inp.extend(yhat[0].tolist())
        tmp_inp = tmp_inp[1:]
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        fut_inp = fut_inp.reshape((1, n_steps,1))
        yhat = modelo.predict(fut_inp, verbose=0)
        tmp_inp.extend(yhat[0].tolist())
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)


len(sc_teste)


#   Plota a previsão ao longo de 60 dias como continuação dos preços

plot_real=np.arange(1,60)
plot_pred=np.arange(60,120)

dado_real= sc.inverse_transform(sc_validacao[1:60])
dado_previsto = sc.inverse_transform(lst_output)
plt.plot(plot_real, dado_real, label='Real')
plt.plot(plot_pred, dado_previsto, label='Previsto')
plt.xlabel('Dias')
plt.ylabel('Preço')
plt.legend()
plt.show()


#   Plota a previsão com a continuidade do preço real

plot_real=np.arange(1,120)
plot_pred=np.arange(60,120)

dado_real= sc.inverse_transform(sc_validacao[:119])
dado_previsto = sc.inverse_transform(lst_output)
plt.plot(plot_real, dado_real, label='Real')
plt.plot(plot_pred, dado_previsto, label='Previsto')
plt.xlabel('Dias')
```

```
plt.ylabel('Preço')
plt.legend()
plt.show()


#   Plotar o gráfico um sobre o outro
#   Plota somente a previsão ao longo de 60 dias sobre o preço real
#   Comparando com a base de validação (jan a mar de 2022)

dado_real= sc.inverse_transform(sc_validacao[60:])
dado_previsto = sc.inverse_transform(lst_output)
plt.plot(dado_real, label='Real')
plt.plot(dado_previsto, label='Previsto')
plt.xlabel('Dias')
plt.ylabel('Preço')
plt.legend()
plt.show()
```

5.b) ARIMA

O modelo ARIMA é caracterizado pelos 3 termos (p, d, q):

p = número de time lags do modelo auto-regressivo (AR)

d = grau de diferenciação, número de diferenciações requeridas para tornar o modelo estacionário;

q = ordem do modelo de média-móvel (MA)

Como podemos ver pelos parâmetros requeridos pelo modelo, qualquer série temporal estacionária pode ser modelada com ARIMA.

Estacionariedade

Subtrair os valores anteriores do valor atual (diferença). Se apenas diferenciarmos uma vez, podemos não obter uma série estacionária, então fazemos isso várias vezes. E o número mínimo de operações de diferenciação necessárias para tornar a série estacionária será inserida em nosso modelo ARIMA.

ADF test

Usaremos o Augumented Dickey Fuller (ADF) para testar se a série de preços é estacionária. A hipótese nula do teste ADF diz que a série não é estacionária. Então, se o p-level do teste for menor que o nível de significância (0.05) podemos rejeitar a hipótese nula e inferir que a série é de fato estacionária.

Neste caso, se o p-value > 0.05 precisaremos encontrar seu valor d.

```
x_treino = ds_treino.Dolar
y_treino = ds_treino.Alvo
x_teste = ds_teste.Dolar
y_teste = ds_teste.Alvo
```

```
x_valida = ds_validacao.Dolar
y_valida = ds_validacao.Alvo
```

```
#     Primeira forma de obter valor de 'd'


#   Verifica se a série é estacionária

resultado = adfuller(ds_dados.Dolar.dropna())
print(f"Estatística ADF: {resultado[0]}")
print(f"p-value: {resultado[1]}")
```

Autocorrelation Function (ACF)

```
fig, (ax1,ax2) = plt.subplots(1, 2, figsize=(16, 4))

ax1.plot(ds_modelo.Dolar)
ax1.set_title("Dados Originais")
#   acrescentar o ; ao final da linha para não plotar duplicado
plot_acf(ds_dados.Dolar, ax=ax2);


#   Uma forma de obter o valor de 'd' é fazer as diferenciações
diff = ds_dados.Dolar.diff().dropna()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 4))

ax1.plot(diff)
ax1.set_title("Difference once")
plot_acf(diff, ax=ax2);


diff = ds_dados.Dolar.diff().dropna()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 4))

ax1.plot(diff)
ax1.set_title("Difference twice")
plot_acf(diff, ax=ax2);
```

Como praticamente não há diferença entre o "difference once" e o "difference twice", necessitou de apenas 1 diferenciação.

```
#   confirmação do valor de 'd'
ndiffs(x_treino, test="adf")


diff.values


#   Autocorrelação parcial

diff = ds_dados.Dolar.diff().dropna()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 4))
```

```
ax1.plot(diff)
ax1.set_title("Difference once")
ax2.set_ylim(0, 1)
plot_pacf(diff, ax=ax2);

# O método auto_arima trabalha com multiplas combinações de p, d e q
# para encontrar os melhores valores
modelo = auto_arima(    ds_dados.Dolar, start_p=1, start_q=1,
    test="adf", max_p=6, max_q=6, m=1,  # frequencia da serie
    d=None,  # determina 'd'
    seasonal=False, trace=True, stepwise=True,
)


#   ARIMA Model

modelo = ARIMA(diff, order=(1, 1, 0))
result = modelo.fit()

print(result.summary())


result.plot_diagnostics(figsize=(16, 8));




#####################################################
#   Documentação do site oficial do pacote ARIMA
#####################################################


#   Estimador do valor de 'd'
kpss_diffs = ndiffs(y_treino, alpha=0.05, test='kpss', max_d=6)
adf_diffs = ndiffs(y_treino, alpha=0.05, test='adf', max_d=6)
n_diffs = max(adf_diffs, kpss_diffs)

print(f"Valor de d estimado: {n_diffs}")


# O método auto_arima trabalha com multiplas combinações de p, d e q
# para encontrar os melhores valores
modelo_arima = pm.auto_arima(y_treino, d=n_diffs, seasonal=False, stepwise=True,
                      suppress_warnings=True, error_action="ignore", max_p=6,
                      max_order=None, trace=True)


print(modelo_arima.order)


def previsao_um_dia():
    fc, conf_int = modelo_arima.predict(n_periods=1, return_conf_int=True)
    return (
        fc.tolist()[0],
        np.asarray(conf_int).tolist()[0])

forecasts = []
intervalo_confianca = []

for nova_obs in y_teste:
```

```
    fc, conf = previsao_um_dia()
    forecasts.append(fc)
    intervalo_confianca.append(conf)

    # Atualiza o modelo com um pequena quantidade de 'MLE steps'
    modelo_arima.update(nova_obs)


print(f"Mean squared error: {mean_squared_error(y_teste, forecasts)}")
print(f"SMAPE: {smape(y_teste, forecasts)}")


fig, axes = plt.subplots(2, 1, figsize=(12, 12))

# --------------------- Actual vs. Predicted --------------------------
axes[0].plot(x_treino, color='blue', label='Dados de Treino')
axes[0].plot(x_teste.index, forecasts, color='green', marker='o',
             label='Preço Previsto')

axes[0].plot(y_teste.index, y_teste, color='red', label='Preço Atual')
axes[0].set_xlabel('Datas')
axes[0].set_ylabel('Preços')

axes[0].legend()

# ------------------ Predicted with confidence intervals ----------------
axes[1].plot(y_treino, color='blue', label='Dados de Treino')
axes[1].plot(x_teste.index, forecasts, color='green',
             label='Preço Previsto')

axes[1].set_title('Preços Previstos & Intervalos de Confiança')
axes[1].set_xlabel('Datas')
axes[1].set_ylabel('Preço')

conf_int = np.asarray(intervalo_confianca)
axes[1].fill_between(x_teste.index,
                     conf_int[:, 0], conf_int[:, 1],
                     alpha=0.9, color='orange',
                     label="Intervalo de Confiança")

axes[1].legend()


fig, axes = plt.subplots(2, 1, figsize=(12, 12))

# --------------------- Actual vs. Predicted --------------------------
#axes[0].plot(x_treino, color='blue', label='Dados de Treino')
axes[0].plot(x_teste.index, forecasts, color='green', marker='o',
             label='Preço Previsto')

axes[0].plot(y_teste.index, y_teste, color='red', label='Preço Atual')
axes[0].set_xlabel('Datas')
axes[0].set_ylabel('Preços')

axes[0].legend()

# ------------------ Predicted with confidence intervals ----------------
#axes[1].plot(y_treino, color='blue', label='Dados de Treino')
```

```
axes[1].plot(x_teste.index, forecasts, color='green',
             label='Preço Previsto')

axes[1].set_title('Preços Previstos & Intervalos de Confiança')
axes[1].set_xlabel('Datas')
axes[1].set_ylabel('Preço')

conf_int = np.asarray(intervalo_confianca)
axes[1].fill_between(x_teste.index,
                     conf_int[:, 0], conf_int[:, 1],
                     alpha=0.9, color='orange',
                     label="Intervalo de Confiança")

axes[1].legend()
```