Prova Final

September 28, 2022

1 Prova Final

1.1 Portfólio com Ações Do Mercado De Criptoativos

[73]:
```
import math
import copy
from datetime import date, timedelta, datetime

import yfinance
import requests
import numpy as np
import pandas as pd
from loguru import logger

import seaborn as sns
import plotly.express as px
import plotly.offline as pyo
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

from pypfopt import plotting
from pypfopt import risk_models
from pypfopt import expected_returns
from pypfopt.risk_models import CovarianceShrinkage
from pypfopt.efficient_frontier import EfficientFrontier

import scipy.stats as stats
import statsmodels.api as sm

from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```python
from sklearn.preprocessing import RobustScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

pyo.init_notebook_mode()
```

Utilizou-se a base de dados do yahoo finance bara buscar criptoativos, isto é, ativos do mercado de criptomoedas. O Yahoo Finance, no entanto, não conta com dados pareados dos ativos em reais, assim, foi utilizada a moeda fiduciária *Dolár Americano*. Como índice para a bolsa, por se tratar de um mercado não regulado e altamente volátil, como indicador do mercado foi utilizado o ativo *USDC* que é uma *stablecoin*.

Stablecoins, também chamadas de moedas estáveis, são criptomoedas pareadas em algum ativo estável ou cesta de ativos, de modo a controlar a volatilidade. Neste caso o ativo *USDC* tem seu valor pareado ao *Dolár Americano*.

O período analisado diz respeito ao ínicio de 2019 até os dias atuais. A data de início escolhida tem ligação com a criação do ativo *USDC* que começou a ser negociado no final de 2018.

[3]:
```python
acoes = ["BTC-USD", "ADA-USD", "LTC-USD", "ETH-USD", "USDC-USD"]
data_inicio = "2019-01-01"
data_fim = date.today().strftime("%Y-%m-%d")

acoes_df = yfinance.download(acoes, data_inicio, data_fim)['Close']
```

```
[*********************100%***********************]  5 of 5 completed
```

[4]:
```python
acoes_df.head()
```

[4]:

Date	ADA-USD	BTC-USD	ETH-USD	LTC-USD	USDC-USD
2019-01-01	0.042547	3843.520020	140.819412	31.979931	1.013301
2019-01-02	0.045258	3943.409424	155.047684	33.433681	1.018173
2019-01-03	0.042682	3836.741211	149.135010	32.026699	1.013577
2019-01-04	0.043812	3857.717529	154.581940	32.404167	1.008160
2019-01-05	0.044701	3845.194580	155.638596	34.936867	1.011010

[5]:
```python
acoes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1366 entries, 2019-01-01 to 2022-09-27
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ADA-USD   1366 non-null   float64
 1   BTC-USD   1366 non-null   float64
 2   ETH-USD   1366 non-null   float64
 3   LTC-USD   1366 non-null   float64
 4   USDC-USD  1366 non-null   float64
```

```
dtypes: float64(5)
memory usage: 64.0 KB
```

1.1.1 Preço Individual dos Ativos

[6]:
```python
preco_individual = px.line(acoes_df, title="Preço Individual dos Ativos")
preco_individual.show()
```

Como visto acima, o Bitcoin parece dominar a escala do gráfico, pois o preço absoluto da ação é muito alto. Os gráficos de todas as outras ações são achatados. Um gráfico como este não é muito útil para comparar o desempenho relativo das ações.

1.2 Otimização do Portfólio

1.2.1 Otimizando o Índice de Sharpe

A fronteira eficiente é o conjunto de carteiras ótimas que oferecem o maior retorno esperado para um nível definido de risco (volatilidade) ou o menor risco (volatilidade) para um determinado nível de retorno esperado. É representado por uma linha no gráfico Retorno vs Volatilidade. A carteira do índice max Sharpe encontra-se na fronteira eficiente.

Para representar tudo visualmente, o código abaixo gera 10000 portfólios de nossas ações com pesos aleatórios e plota seus retornos e volatilidade. A fronteira eficiente e a carteira de razão máxima de Sharpe também são plotadas no gráfico.

[7]:
```python
# calculate the mean and variance
# mu = expected_returns.ema_historical_return(acoes_df, frequency=365)
mu = expected_returns.mean_historical_return(acoes_df, compounding=True,␣
 ↪frequency=1363)
sigma = risk_models.exp_cov(acoes_df, frequency=1363)
```

[8]:
```python
sigma
```

[8]:

	ADA-USD	BTC-USD	ETH-USD	LTC-USD	USDC-USD
ADA-USD	2.968467	1.703432	2.367813	2.207867	-0.000058
BTC-USD	1.703432	1.629830	1.959061	1.636192	0.000079
ETH-USD	2.367813	1.959061	3.205307	2.378586	-0.000436
LTC-USD	2.207867	1.636192	2.378586	2.640843	0.000711
USDC-USD	-0.000058	0.000079	-0.000436	0.000711	0.000121

[9]:
```python
# get the efficient frontier
ef = EfficientFrontier(mu, sigma)
```

[10]:
```python
sharpe_weights = ef.max_sharpe()
ef.clean_weights()
```

[10]:
```
OrderedDict([('ADA-USD', 0.78015),
             ('BTC-USD', 0.0),
             ('ETH-USD', 0.21985),
```

```
                ('LTC-USD', 0.0),
                ('USDC-USD', 0.0)])
```

[11]:
```
ef.portfolio_performance(verbose=True)
```

```
Expected annual return: 914.5%
Annual volatility: 166.5%
Sharpe Ratio: 5.48
```

[11]:
```
(9.14463927257169, 1.6654935716831423, 5.478639742422038)
```

[12]:
```
texto = f"O portfólio ideal que maximiza o Sharpe Ratio é investir em ADA-USD␣
↪({ef.clean_weights().get('ADA-USD') * 100}) e em ETH ({ef.clean_weights().
↪get('ETH-USD') * 100})."
```

[13]:
```
print(texto)
```

```
O portfólio ideal que maximiza o Sharpe Ratio é investir em ADA-USD (78.015) e
em ETH (21.985).
```

1.2.2 Agora, desejamos uma carteira com os ativos de menor volatilidade.

[14]:
```
mu_min_v = expected_returns.mean_historical_return(acoes_df, compounding=True,␣
↪frequency=1363)
sigma_min_v = risk_models.exp_cov(acoes_df, frequency=1363)
ef_min_v = EfficientFrontier(mu, sigma)
```

[15]:
```
pesos = ef_min_v.min_volatility()
```

[16]:
```
pesos
```

[16]:
```
OrderedDict([('ADA-USD', 0.0),
             ('BTC-USD', 0.0),
             ('ETH-USD', 0.0001736030025511),
             ('LTC-USD', 0.0),
             ('USDC-USD', 0.999826396997449)])
```

Para um portfólio com a menor volatilidade devemos investir 99.99% em USDC e 0.001% em ETH.

[17]:
```
ef_min_v.portfolio_performance(verbose=True)
```

```
Expected annual return: -1.2%
Annual volatility: 1.1%
Sharpe Ratio: -2.87
```

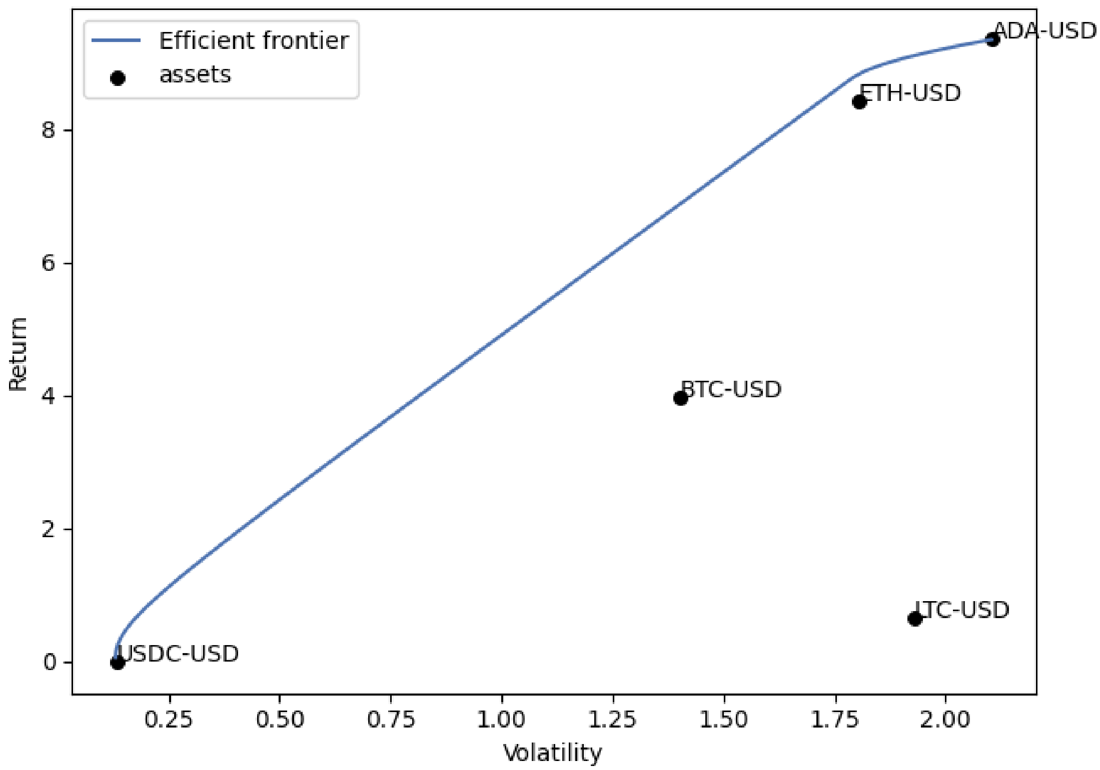
[17]:
```
(-0.011523026034443254, 0.010975853208269794, -2.872034222423103)
```

1.2.3 Plot da Fronteira Eficiente

[18]:
```
mu_frontier = expected_returns.mean_historical_return(acoes_df,␣
 ↪compounding=True, frequency=1363)
sigma_frontier = risk_models.sample_cov(acoes_df, frequency=1363)
```

[19]:
```
ef_frontier = EfficientFrontier(mu_frontier, sigma_frontier)
```

[20]:
```
fig, ax = plt.subplots()
plotting.plot_efficient_frontier(ef_frontier, ax=ax, show_assets=True)

for i, asset in enumerate(ef_frontier.tickers):
    ax.annotate(asset, ((np.diag(ef_frontier.cov_matrix) ** (1/2))[i],␣
 ↪ef_frontier.expected_returns[i]))
```



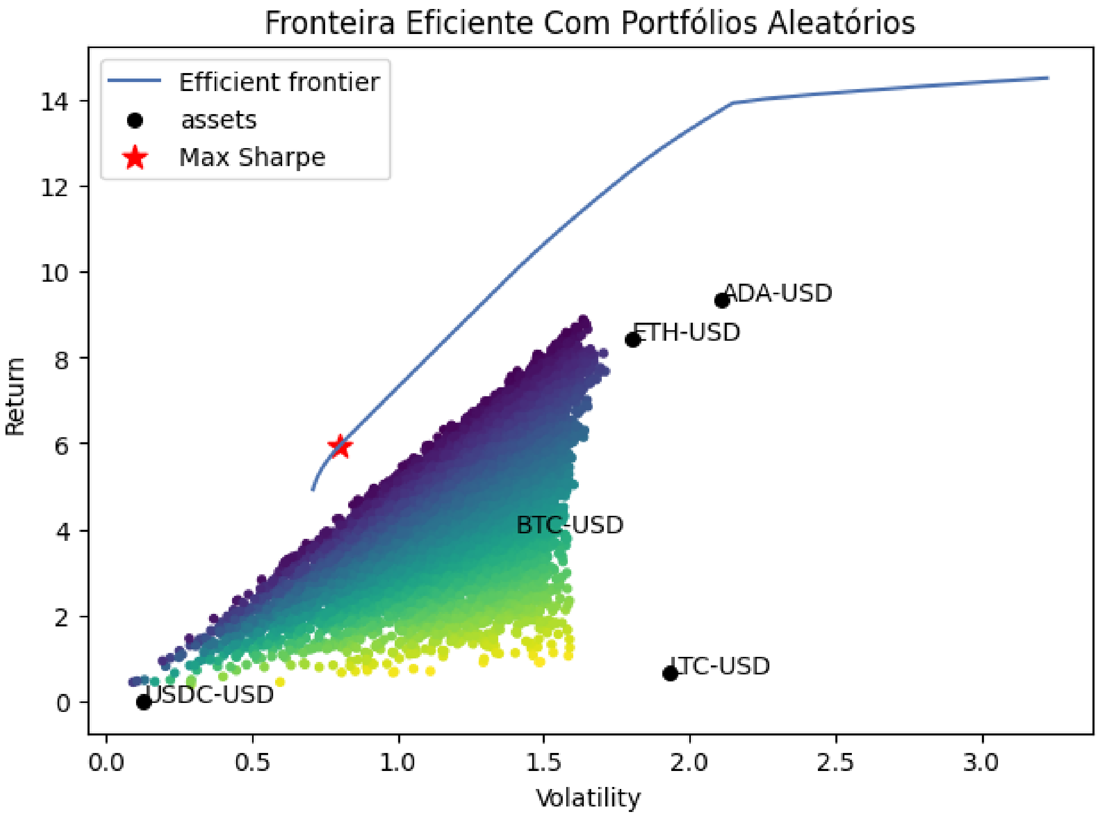
Com o gráfico da fronteira eficiente é possível inferir que o ativo *ADA-USD* apresenta o maior retorno dentro do período medido em contrapartida o ativo *LTC-USD* apresenta quase o mesmo risco, no entanto, com um retorno muito inferior.

O ativo *USDC-USD* apresenta a menor volatilidade e também o menor retorno no período observado. Os resultados são esperados uma vez que o ativo é uma *stablecoin* e tem seu valor lastreado no dólar americano.

1.3 Com Portfólios Aleatórios

[21]:
```python
ef_frontier = EfficientFrontier(mu_frontier, sigma_frontier,␣
 ↪weight_bounds=(None, None))
ef_frontier.add_constraint(lambda w: w[0] >= 0.2)
ef_frontier.add_constraint(lambda w: w[2] == 0.15)
ef_frontier.add_constraint(lambda w: w[3] + w[4] <= 0.10)
```

[22]:
```python
fig, ax = plt.subplots()
ef_max_sharpe = copy.deepcopy(ef_frontier)
plotting.plot_efficient_frontier(ef_frontier, ax=ax, show_assets=True)

for i, asset in enumerate(ef_frontier.tickers):
    ax.annotate(asset, ((np.diag(ef_frontier.cov_matrix) ** (1/2))[i],␣
 ↪ef_frontier.expected_returns[i]))

# Find the tangency portfolio
ef_max_sharpe.max_sharpe()
ret_tangent, std_tangent, _ = ef_max_sharpe.portfolio_performance()
ax.scatter(std_tangent, ret_tangent, marker="*", s=100, c="r", label="Max␣
 ↪Sharpe")

# Generate random portfolios
n_samples = 10_000
w = np.random.dirichlet(np.ones(ef.n_assets), n_samples)
rets = w.dot(ef.expected_returns)
stds = np.sqrt(np.diag(w @ ef.cov_matrix @ w.T))
sharpes = rets / stds
ax.scatter(stds, rets, marker=".", c=sharpes, cmap="viridis_r")

# Output
ax.set_title("Fronteira Eficiente Com Portfólios Aleatórios")
ax.legend()
plt.tight_layout()
plt.show()
```



1.4 Análise Descritiva dos Dados

1.4.1 Retornos Diários

O retorno diário de uma ação é o ganho fracionário (ou perda) em um determinado dia em relação ao dia anterior, é dado por (preço de fechamento do dia atual - preço de fechamento do dia anterior) / (preço de fechamento do dia anterior). Por ser um valor relativo, proporciona uma comparação mais justa entre os retornos das ações, independentemente dos preços absolutos das ações. O método pct_change() pode ser usado para obter os retornos diários de forma eficiente.

[23]:
```
retornos = acoes_df.pct_change()
retornos.head()
```

[23]:

	ADA-USD	BTC-USD	ETH-USD	LTC-USD	USDC-USD
Date					
2019-01-01	NaN	NaN	NaN	NaN	NaN
2019-01-02	0.063718	0.025989	0.101039	0.045458	0.004808
2019-01-03	-0.056918	-0.027050	-0.038135	-0.042083	-0.004514
2019-01-04	0.026475	0.005467	0.036523	0.011786	-0.005344
2019-01-05	0.020291	-0.003246	0.006836	0.078160	0.002827

Agora, traçando os retornos diários das ações ADA-USD, BTC-USD, ETH-USD, USDC-USD, durante todo o período medido. Notavelmente, as flutuações são muito maiores durante um período de alta volatilidade (ou seja, durante o crash do Covid em março de 2020).

[24]:
```
fig = px.line(retornos[["BTC-USD", "ADA-USD", "LTC-USD", "ETH-USD",␣
 ↪"USDC-USD"]], title='Retornos Diários')
fig.show()
```

1.4.2 Média dos Retornos Diários

[25]:
```
retornos.mean()
```

[25]:
```
ADA-USD     0.003332
BTC-USD     0.001910
ETH-USD     0.002866
LTC-USD     0.001753
USDC-USD   -0.000003
dtype: float64
```

1.4.3 Desvio Padrão dos Retornos Diários

[26]:
```
retornos.std()
```

[26]:
```
ADA-USD     0.057005
BTC-USD     0.037983
ETH-USD     0.048832
LTC-USD     0.052249
USDC-USD    0.003583
dtype: float64
```

1.4.4 Matriz de Covariância

[27]:
```
retornos.cov()
```

[27]:

	ADA-USD	BTC-USD	ETH-USD	LTC-USD	USDC-USD
ADA-USD	0.003250	0.001444	0.002047	0.002145	-0.000012
BTC-USD	0.001444	0.001443	0.001512	0.001562	-0.000009
ETH-USD	0.002047	0.001512	0.002385	0.002091	-0.000012
LTC-USD	0.002145	0.001562	0.002091	0.002730	-0.000008
USDC-USD	-0.000012	-0.000009	-0.000012	-0.000008	0.000013

1.4.5 Mapa de Calor a Partir da Matriz de Correlação dos Ativos

A matriz de correlação nos dá os coeficientes de correlação entre cada par de ações. Os coeficientes de correlação são indicadores da força da relação linear entre duas variáveis diferentes. É um valor de 0 a 1, com 1 indicando a relação mais forte. Se for um valor negativo, então as duas variáveis estão inversamente relacionadas.

[28]:
```python
def most_correlated(dataframe):
    """
    Returns a dataframe that contains the most correlated features

    dataframe: dataframe that gives the column names and their correlation value
    """
    corr_values = dataframe.abs().unstack()
    sorted_values = pd.DataFrame(corr_values.sort_values(kind="quicksort"),␣
 ↪index= None)
    sorted_values = sorted_values[(sorted_values[0] > 0.6) & (sorted_values[0]␣
 ↪< 1)]

    return sorted_values.drop_duplicates()
```

[29]:
```python
corr_df = acoes_df.corr().round(2) # round to 2 decimal places
fig_corr = px.imshow(corr_df, title = 'Correlação Entre Criptoativos')
fig_corr.show()
```

[30]:
```python
most_correlated(corr_df)
```

[30]:

		0
LTC-USD	ETH-USD	0.75
ADA-USD	LTC-USD	0.79
BTC-USD	LTC-USD	0.86
ADA-USD	BTC-USD	0.89
ETH-USD	ADA-USD	0.91
BTC-USD	ETH-USD	0.92

1.5 ANOVA e Testes de Hipóteses

Para o teste de ANOVA é necessário que algumas suposições se provem verdadeiras:

- Normalidade: Cada amostra é de uma população normalmente distribuída.
- Indepedência: As amostras são independentes.
- Homocedasticidade: Os desvios padrão da população dos grupos são todos iguais.

Serão coletados dados dos ativos ao longo de um ano.

[31]:
```python
acoes = ["BTC-USD", "ADA-USD", "LTC-USD", "ETH-USD", "USDC-USD"]
data_inicio = "2019-01-01"
data_fim = "2019-12-31"

cripto_df = yfinance.download(acoes, data_inicio, data_fim)['Close']
```

```
[*********************100%***********************]  5 of 5 completed
```

[32]:
```python
cripto_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2019-01-01 to 2019-12-31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ADA-USD   365 non-null    float64
 1   BTC-USD   365 non-null    float64
 2   ETH-USD   365 non-null    float64
 3   LTC-USD   365 non-null    float64
 4   USDC-USD  365 non-null    float64
dtypes: float64(5)
memory usage: 17.1 KB
```

1.5.1 Teste de Shapiro-Wilk - Normalidade

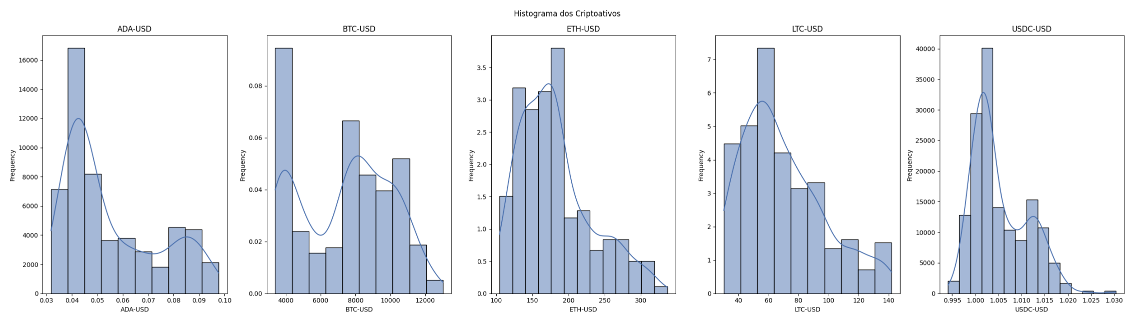
O teste de Shapiro-Wilk é um teste de normalidade. É usado para determinar se uma amostra vem ou não de uma distribuição normal.

Para realizar um teste Shapiro-Wilk em Python podemos usar a função scipy.stats.shapiro()

[33]:
```
shapiro_df = pd.DataFrame(columns=[k for k in cripto_df.keys()],␣
 ↪index=["estatistica", "p_valor"])

for coluna in shapiro_df:
    shapiro_df[coluna] = stats.shapiro(cripto_df[coluna])

shapiro_df
```

[33]:
```
                  ADA-USD       BTC-USD       ETH-USD       LTC-USD  \
estatistica  8.611770e-01  9.289624e-01  9.320979e-01  9.371803e-01
p_valor      1.611809e-17  3.673350e-12  7.715647e-12  2.700180e-11

                 USDC-USD
estatistica  9.196935e-01
p_valor      4.621984e-13
```

[34]:
```
# histograma
fig, axis = plt.subplots(ncols=len(cripto_df.columns), figsize=(25, 7))
fig.suptitle("Histograma dos Criptoativos")


for col, ax in zip(cripto_df, axis):
    sns.histplot(data=col, x=cripto_df[col], stat="frequency", ax=ax, kde=True)
    ax.set_title(col)

fig.tight_layout()
plt.show()
```



1.5.2 Teste de Levene - Homocedasticidade

[35]:
```python
levene_df = pd.DataFrame(columns=[k for k in cripto_df.keys() if k != "USDC-USD"], index=["estatistica_F", "p_valor"])

for coluna in levene_df:
    levene_df[coluna] = stats.levene(acoes_df["USDC-USD"], cripto_df[coluna])

levene_df
```

[35]:

	ADA-USD	BTC-USD	ETH-USD	LTC-USD
estatistica_F	6.693143e+02	3182.561485	1.809551e+03	1.902920e+03
p_valor	5.063106e-125	0.000000	3.487421e-271	5.742226e-281

Como o valor de p é menor que 0.05 nos testes de Shapiro-Wilk e Levene, rejeitamos a hipótese nula. Temos evidências suficientes para dizer que os dados da amostra não vêm de uma distribuição normal.

1.5.3 ANOVA

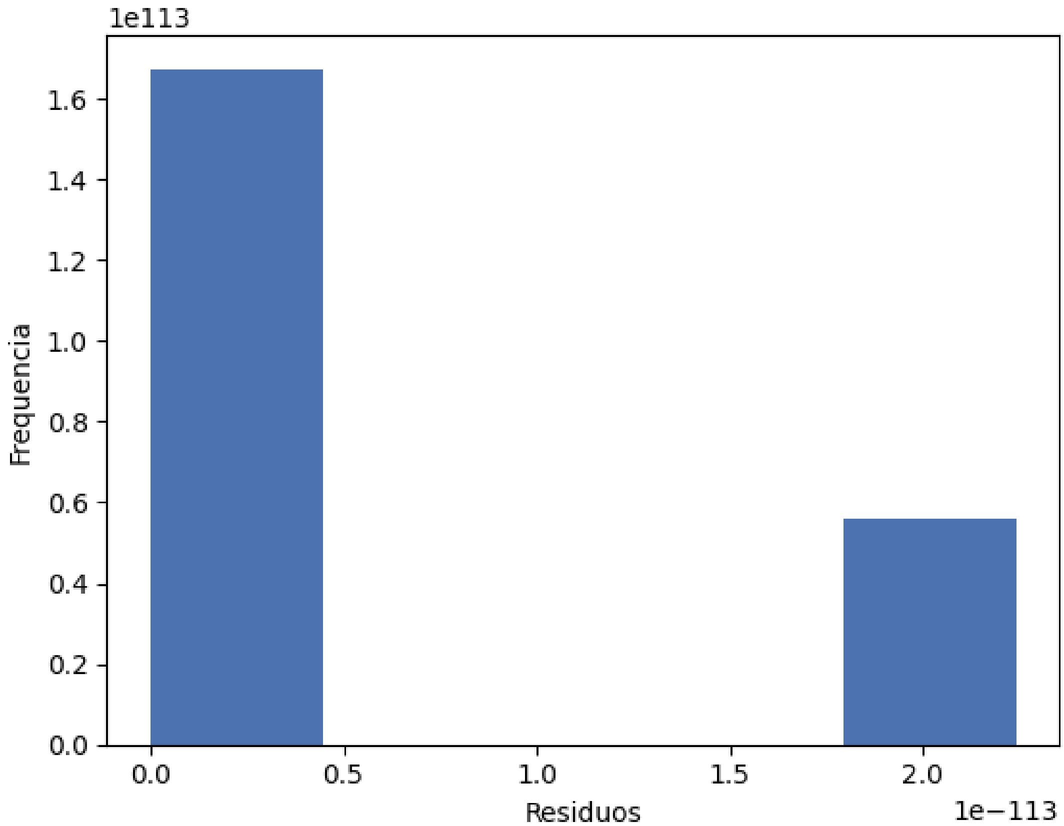
[36]:
```python
anova_df = pd.DataFrame(columns=[k for k in cripto_df.keys() if k != "USDC-USD"], index=["f_valor", "p_valor"])

for coluna in anova_df:
    anova_df[coluna] = stats.f_oneway(cripto_df["USDC-USD"], cripto_df[coluna])

anova_df
```

[36]:

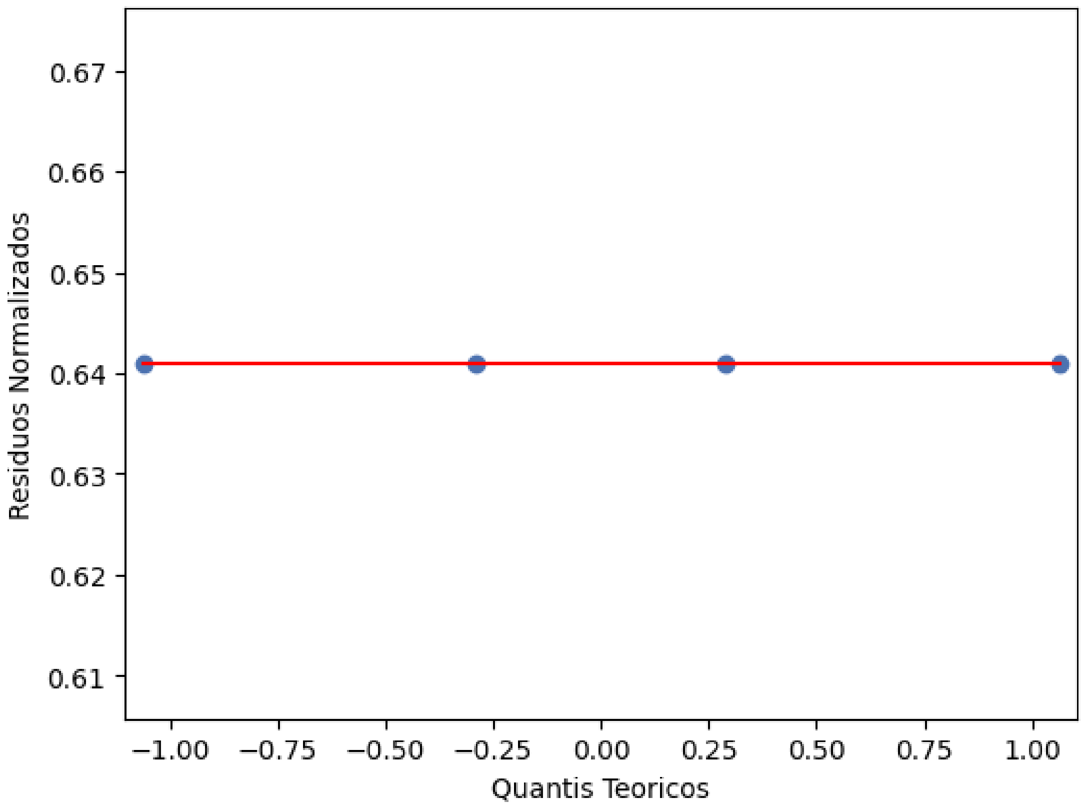
	ADA-USD	BTC-USD	ETH-USD	LTC-USD
f_valor	873502.484407	2.866299e+03	4.698495e+03	2.271495e+03
p_valor	0.000000	1.235932e-254	8.967291e-320	5.025830e-226

[37]:
```python
# histograma
plt.hist(np.sqrt(anova_df.iloc[1]), bins="doane", histtype="bar", density=True)
plt.xlabel("Residuos")
```

```
plt.ylabel("Frequencia")
plt.show()
```



[38]:
```
sm.qqplot(np.sqrt(anova_df.iloc[1]), dist=stats.t, line="s", fit=True)
plt.xlabel("Quantis Teoricos")
plt.ylabel("Residuos Normalizados")
plt.show()
```



1.6 Regressão Linear

[39]:
```
cripto_df.corr()
```

[39]:

	ADA-USD	BTC-USD	ETH-USD	LTC-USD	USDC-USD
ADA-USD	1.000000	0.187901	0.704564	0.853517	-0.366252
BTC-USD	0.187901	1.000000	0.753659	0.602965	-0.728380
ETH-USD	0.704564	0.753659	1.000000	0.918237	-0.644323
LTC-USD	0.853517	0.602965	0.918237	1.000000	-0.573592
USDC-USD	-0.366252	-0.728380	-0.644323	-0.573592	1.000000

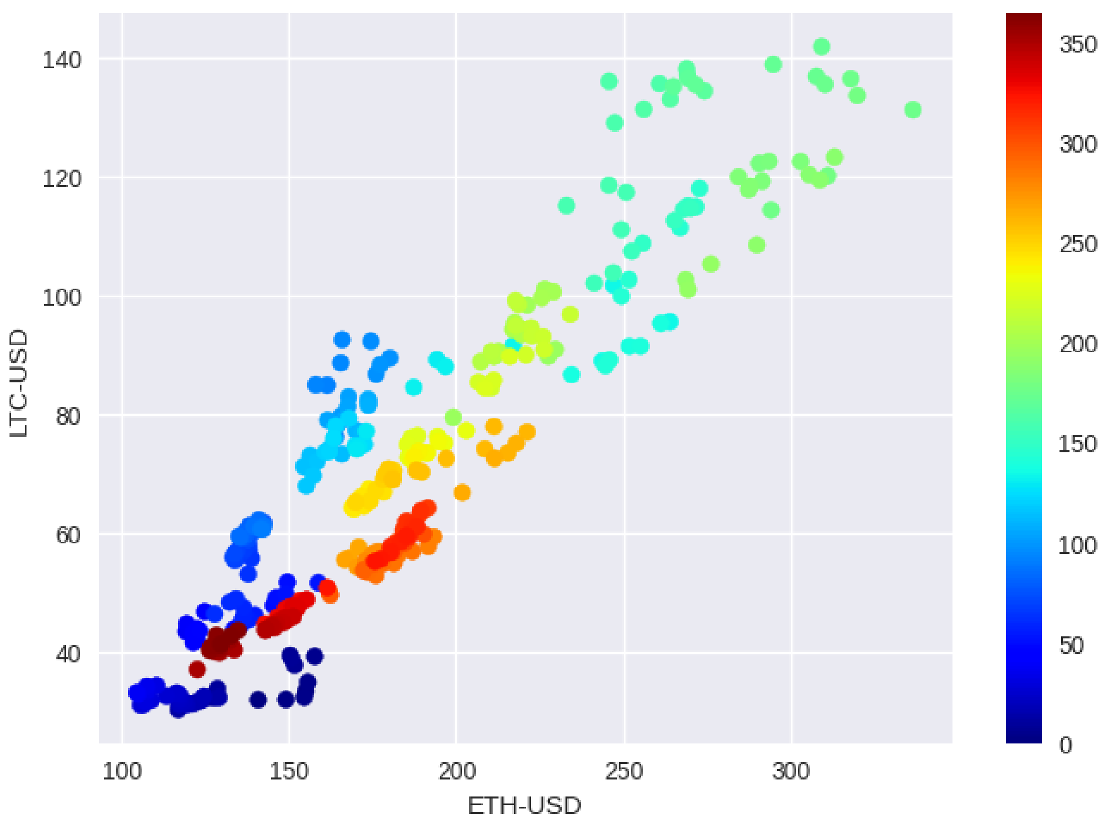
Escolheremos o ativo *LTC-USD* e como índice o ativo *ETH-USD*. Pois, são os dois ativos que possuem maior correlação dentro do período observado dentro da atividade proposta. Ademais, será possível prever o valor de um a partir do outro.

1.6.1 Gráfico de Dispersão

Queremos saber se existe de fato uma relação linear entre os dois ativos. Para isto, faremos um gráfico de dispersão dos dois ativos.

[40]:
```
plt.style.use("seaborn")
plt.scatter(
    x=cripto_df["ETH-USD"],
    y=cripto_df["LTC-USD"],
    c=np.array(range(0, len(cripto_df["USDC-USD"]))),
    cmap="jet",
)
plt.xlabel("ETH-USD")
plt.ylabel("LTC-USD")
plt.colorbar()
plt.show()
```

```
/tmp/ipykernel_51731/1375303785.py:1: MatplotlibDeprecationWarning:

The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no
longer correspond to the styles shipped by seaborn. However, they will remain
available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API
instead.
```



1.6.2 Gráfico da Tendência dos Dados em Relação a Média

Agora, queremos analisar visualmente uma linha que atravesse os dois dados na média e apresente o comportamento dos dados em relação a esta tendência.

[41]:
```
fig, ax = plt.subplots()
plt.scatter(
    x=cripto_df["ETH-USD"],
    y=cripto_df["LTC-USD"],
    c=np.array(range(0, len(cripto_df["USDC-USD"]))),
    cmap="jet",
)
ax.plot(
    np.unique(cripto_df["ETH-USD"]),
    np.poly1d(np.polyfit(cripto_df["ETH-USD"], cripto_df["LTC-USD"], 1))(
        np.unique(cripto_df["ETH-USD"])
    ),
    color="black",
)
```

[41]: [<matplotlib.lines.Line2D at 0x7f99ac6e4bb0>]

1.6.3 Regressão Linear Com Sklearn

[42]:
```
X = cripto_df["ETH-USD"].values.reshape(-1, 1)
y = cripto_df["LTC-USD"].values.reshape(-1, 1)
```

[43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[44]:
```
regressor = LinearRegression()
```

[45]:
```
regressor.fit(X_train, y_train)
```

[45]:
```
LinearRegression()
```

[46]:
```
regressor.intercept_
```

[46]:
```
array([-22.83377267])
```

[47]:
```
regressor.coef_
```

[47]:
```
array([[0.50937123]])
```

Com o resultado podemos criar a seguinte equação:

$$LTC = 1.66858034 * ETH + 1.66858034 \tag{1}$$

Vamos testar a equação e checar o quanto o modelo se aproximou do resultado real

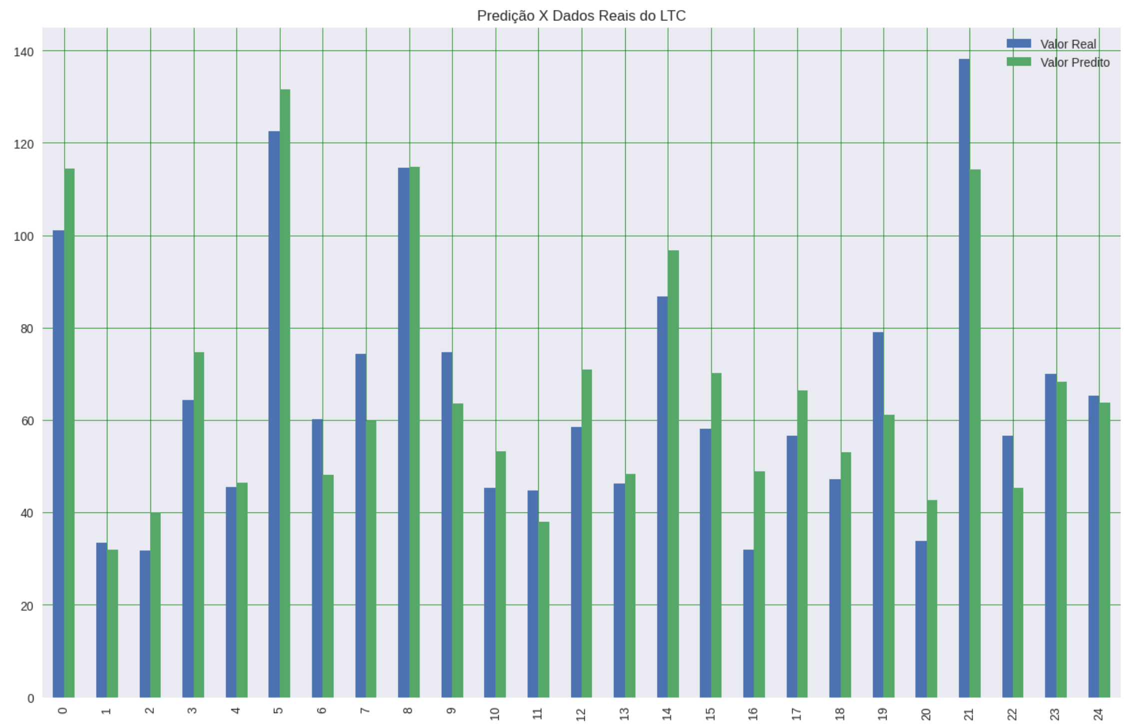
[48]:
```
y_pred = regressor.predict(X_test)
```

[49]:
```
df = pd.DataFrame({"Valor Real": y_test.flatten(), "Valor Predito": y_pred.
↪flatten()})
df
```

[49]:
```
    Valor Real  Valor Predito
0   101.024796     114.420788
1    33.439255      31.919900
2    31.823418      40.097548
3    64.268745      74.758618
4    45.581425      46.504875
..         …              …
68   90.622627      85.524012
69   60.223877      46.943425
70   88.677864      61.474728
71  105.300056     117.893631
72   47.615536      53.708511
```

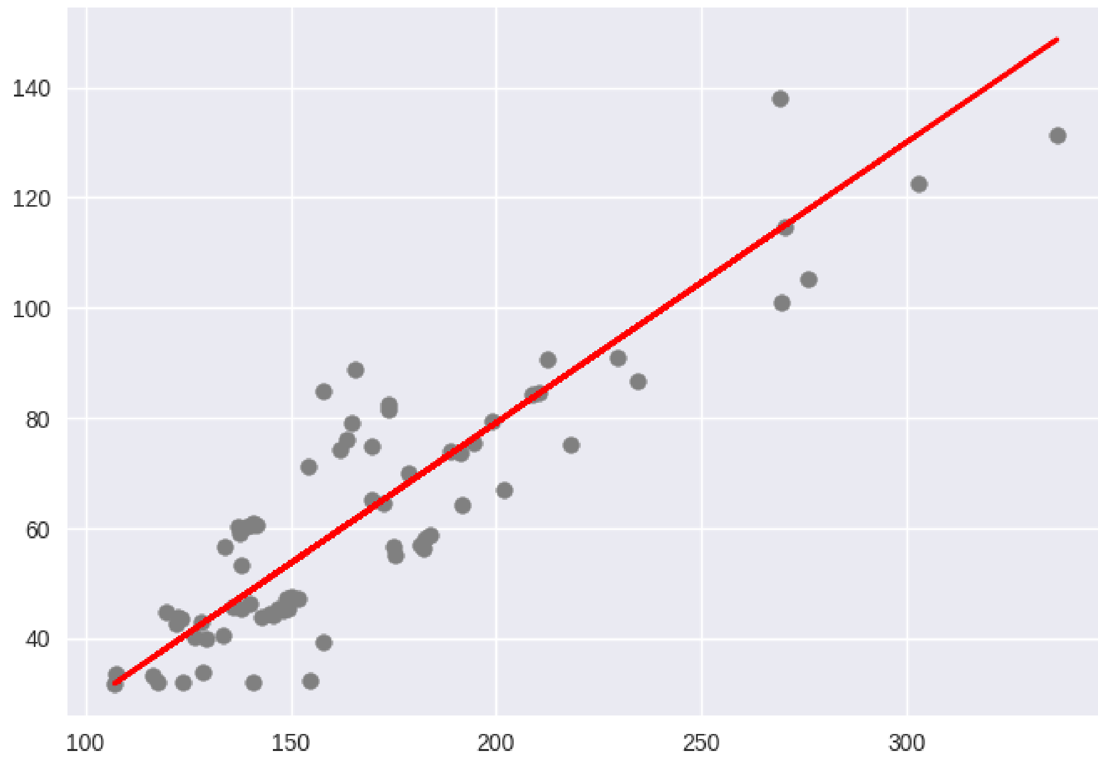
```
[73 rows x 2 columns]
```

[50]:
```
df1 = df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.title("Predição X Dados Reais do LTC")
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



A linha de regressão do modelo treinado

[51]:
```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



O quanto o modelo errou, usando *Mean Absolute Error*, *Mean Squared Error* e *Root Mean Squared Error*.

[52]:
```
print("Mean Absolute Error:", metrics.mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", metrics.mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(metrics.mean_squared_error(y_test,␣
↪y_pred)))
```

```
Mean Absolute Error: 8.806731177763458
Mean Squared Error: 121.66276892190167
Root Mean Squared Error: 11.030084719615786
```

1.7 Machine Learning

Coletaremos os dados de operação do ativo *Bitcoin* do ano (2013) em que ele começou a ser operado na plataforma Mercado Bitcoin até os dias atuais.

[53]:
```
sns.set_style('white', { 'axes.spines.right': False, 'axes.spines.top': False})

# check the tensorflow version and the number of available GPUs
print('Tensorflow Version: ' + tf.__version__)
physical_devices = tf.config.list_physical_devices('GPU')
print("Num GPUs:", len(physical_devices))
```

```python
# Setting the timeframe for the data extraction
end_date =  date.today().strftime("%Y-%m-%d")
tomorrow =  (date.today() + timedelta(days=1)).strftime("%Y-%m-%d")
```

```
Tensorflow Version: 2.10.0
Num GPUs: 1
```

```
2022-09-28 07:46:36.197861: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-09-28 07:46:36.201643: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-09-28 07:46:36.202162: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
```

[54]:
```python
days = range(1, 32)
months = range(1, 13)
years = range(2013, datetime.now().year + 1)
day_summary = "https://www.mercadobitcoin.net/api/{coin}/day-summary/{year}/
 ↪{month}/{day}/"
coin = "BTC"
coin_info = []
sample_json = {
    "date": "2022-09-02",
    "opening": 105024.1362104,
    "closing": 103517.33893317,
    "lowest": 102759.09207443,
    "highest": 106234.45856376,
    "volume": "4252248.79130990",
    "quantity": "40.62196075",
    "amount": 2933,
    "avg_price": 104678.57072383,
}
```

1.7.1 Coleta de Dados

[55]:
```python
for year in years:
    for month in months:

        if year == datetime.now().year and month > datetime.now().month:
            continue
```

```
        logger.info(f"{year}-{month}")
        for day in days:

            if month == datetime.now().month and day >= datetime.now().day: #␣
↪we cant look up for values in the future
                continue

            url = day_summary.format(coin=coin, year=year, month=month, day=day)
            response = requests.get(url=url)

            if response.status_code != 200:
                continue

            coin_info.append(response.json())
```

```
2022-09-28 07:46:36.211 | INFO     | __main__:<module>:6 - 2013-1
2022-09-28 07:46:44.339 | INFO     | __main__:<module>:6 - 2013-2
2022-09-28 07:46:52.421 | INFO     | __main__:<module>:6 - 2013-3
2022-09-28 07:47:00.297 | INFO     | __main__:<module>:6 - 2013-4
2022-09-28 07:47:08.644 | INFO     | __main__:<module>:6 - 2013-5
2022-09-28 07:47:17.220 | INFO     | __main__:<module>:6 - 2013-6
2022-09-28 07:47:25.459 | INFO     | __main__:<module>:6 - 2013-7
2022-09-28 07:47:33.571 | INFO     | __main__:<module>:6 - 2013-8
2022-09-28 07:47:41.703 | INFO     | __main__:<module>:6 - 2013-9
2022-09-28 07:47:48.658 | INFO     | __main__:<module>:6 - 2013-10
2022-09-28 07:47:56.611 | INFO     | __main__:<module>:6 - 2013-11
2022-09-28 07:48:04.807 | INFO     | __main__:<module>:6 - 2013-12
2022-09-28 07:48:23.732 | INFO     | __main__:<module>:6 - 2014-1
2022-09-28 07:48:37.085 | INFO     | __main__:<module>:6 - 2014-2
2022-09-28 07:48:46.388 | INFO     | __main__:<module>:6 - 2014-3
2022-09-28 07:48:54.508 | INFO     | __main__:<module>:6 - 2014-4
2022-09-28 07:49:02.664 | INFO     | __main__:<module>:6 - 2014-5
2022-09-28 07:49:10.943 | INFO     | __main__:<module>:6 - 2014-6
2022-09-28 07:49:19.059 | INFO     | __main__:<module>:6 - 2014-7
2022-09-28 07:49:27.089 | INFO     | __main__:<module>:6 - 2014-8
2022-09-28 07:49:35.160 | INFO     | __main__:<module>:6 - 2014-9
2022-09-28 07:49:42.134 | INFO     | __main__:<module>:6 - 2014-10
2022-09-28 07:49:50.567 | INFO     | __main__:<module>:6 - 2014-11
2022-09-28 07:49:58.937 | INFO     | __main__:<module>:6 - 2014-12
2022-09-28 07:50:06.977 | INFO     | __main__:<module>:6 - 2015-1
2022-09-28 07:50:15.413 | INFO     | __main__:<module>:6 - 2015-2
2022-09-28 07:50:23.553 | INFO     | __main__:<module>:6 - 2015-3
2022-09-28 07:50:32.202 | INFO     | __main__:<module>:6 - 2015-4
2022-09-28 07:50:40.194 | INFO     | __main__:<module>:6 - 2015-5
2022-09-28 07:50:48.240 | INFO     | __main__:<module>:6 - 2015-6
2022-09-28 07:50:56.369 | INFO     | __main__:<module>:6 - 2015-7
2022-09-28 07:51:04.526 | INFO     | __main__:<module>:6 - 2015-8
```

```
2022-09-28 07:51:12.590 | INFO     | __main__:<module>:6 - 2015-9
2022-09-28 07:51:19.480 | INFO     | __main__:<module>:6 - 2015-10
2022-09-28 07:51:27.815 | INFO     | __main__:<module>:6 - 2015-11
2022-09-28 07:51:35.814 | INFO     | __main__:<module>:6 - 2015-12
2022-09-28 07:51:43.945 | INFO     | __main__:<module>:6 - 2016-1
2022-09-28 07:51:52.225 | INFO     | __main__:<module>:6 - 2016-2
2022-09-28 07:52:00.203 | INFO     | __main__:<module>:6 - 2016-3
2022-09-28 07:52:08.237 | INFO     | __main__:<module>:6 - 2016-4
2022-09-28 07:52:16.296 | INFO     | __main__:<module>:6 - 2016-5
2022-09-28 07:52:24.296 | INFO     | __main__:<module>:6 - 2016-6
2022-09-28 07:52:32.692 | INFO     | __main__:<module>:6 - 2016-7
2022-09-28 07:52:40.840 | INFO     | __main__:<module>:6 - 2016-8
2022-09-28 07:52:48.873 | INFO     | __main__:<module>:6 - 2016-9
2022-09-28 07:52:56.390 | INFO     | __main__:<module>:6 - 2016-10
2022-09-28 07:53:04.408 | INFO     | __main__:<module>:6 - 2016-11
2022-09-28 07:53:12.541 | INFO     | __main__:<module>:6 - 2016-12
2022-09-28 07:53:21.162 | INFO     | __main__:<module>:6 - 2017-1
2022-09-28 07:53:29.185 | INFO     | __main__:<module>:6 - 2017-2
2022-09-28 07:53:37.270 | INFO     | __main__:<module>:6 - 2017-3
2022-09-28 07:53:45.847 | INFO     | __main__:<module>:6 - 2017-4
2022-09-28 07:53:54.431 | INFO     | __main__:<module>:6 - 2017-5
2022-09-28 07:54:02.544 | INFO     | __main__:<module>:6 - 2017-6
2022-09-28 07:54:10.527 | INFO     | __main__:<module>:6 - 2017-7
2022-09-28 07:54:18.486 | INFO     | __main__:<module>:6 - 2017-8
2022-09-28 07:54:26.581 | INFO     | __main__:<module>:6 - 2017-9
2022-09-28 07:54:33.505 | INFO     | __main__:<module>:6 - 2017-10
2022-09-28 07:54:41.493 | INFO     | __main__:<module>:6 - 2017-11
2022-09-28 07:54:49.501 | INFO     | __main__:<module>:6 - 2017-12
2022-09-28 07:54:57.631 | INFO     | __main__:<module>:6 - 2018-1
2022-09-28 07:55:05.760 | INFO     | __main__:<module>:6 - 2018-2
2022-09-28 07:55:13.810 | INFO     | __main__:<module>:6 - 2018-3
2022-09-28 07:55:22.124 | INFO     | __main__:<module>:6 - 2018-4
2022-09-28 07:55:30.126 | INFO     | __main__:<module>:6 - 2018-5
2022-09-28 07:55:38.239 | INFO     | __main__:<module>:6 - 2018-6
2022-09-28 07:55:46.661 | INFO     | __main__:<module>:6 - 2018-7
2022-09-28 07:55:54.744 | INFO     | __main__:<module>:6 - 2018-8
2022-09-28 07:56:02.708 | INFO     | __main__:<module>:6 - 2018-9
2022-09-28 07:56:09.553 | INFO     | __main__:<module>:6 - 2018-10
2022-09-28 07:56:17.644 | INFO     | __main__:<module>:6 - 2018-11
2022-09-28 07:56:25.640 | INFO     | __main__:<module>:6 - 2018-12
2022-09-28 07:56:33.710 | INFO     | __main__:<module>:6 - 2019-1
2022-09-28 07:56:41.823 | INFO     | __main__:<module>:6 - 2019-2
2022-09-28 07:56:49.978 | INFO     | __main__:<module>:6 - 2019-3
2022-09-28 07:56:58.060 | INFO     | __main__:<module>:6 - 2019-4
2022-09-28 07:57:06.100 | INFO     | __main__:<module>:6 - 2019-5
2022-09-28 07:57:14.151 | INFO     | __main__:<module>:6 - 2019-6
2022-09-28 07:57:22.205 | INFO     | __main__:<module>:6 - 2019-7
2022-09-28 07:57:30.354 | INFO     | __main__:<module>:6 - 2019-8
```

```
2022-09-28 07:57:38.485 | INFO     | __main__:<module>:6 - 2019-9
2022-09-28 07:57:45.850 | INFO     | __main__:<module>:6 - 2019-10
2022-09-28 07:57:54.029 | INFO     | __main__:<module>:6 - 2019-11
2022-09-28 07:58:02.015 | INFO     | __main__:<module>:6 - 2019-12
2022-09-28 07:58:10.027 | INFO     | __main__:<module>:6 - 2020-1
2022-09-28 07:58:18.189 | INFO     | __main__:<module>:6 - 2020-2
2022-09-28 07:58:26.342 | INFO     | __main__:<module>:6 - 2020-3
2022-09-28 07:58:34.320 | INFO     | __main__:<module>:6 - 2020-4
2022-09-28 07:58:42.352 | INFO     | __main__:<module>:6 - 2020-5
2022-09-28 07:58:50.310 | INFO     | __main__:<module>:6 - 2020-6
2022-09-28 07:58:58.340 | INFO     | __main__:<module>:6 - 2020-7
2022-09-28 07:59:06.414 | INFO     | __main__:<module>:6 - 2020-8
2022-09-28 07:59:14.381 | INFO     | __main__:<module>:6 - 2020-9
2022-09-28 07:59:21.445 | INFO     | __main__:<module>:6 - 2020-10
2022-09-28 07:59:29.439 | INFO     | __main__:<module>:6 - 2020-11
2022-09-28 07:59:37.519 | INFO     | __main__:<module>:6 - 2020-12
2022-09-28 07:59:45.541 | INFO     | __main__:<module>:6 - 2021-1
2022-09-28 07:59:54.022 | INFO     | __main__:<module>:6 - 2021-2
2022-09-28 08:00:02.771 | INFO     | __main__:<module>:6 - 2021-3
2022-09-28 08:00:10.874 | INFO     | __main__:<module>:6 - 2021-4
2022-09-28 08:00:19.187 | INFO     | __main__:<module>:6 - 2021-5
2022-09-28 08:00:27.700 | INFO     | __main__:<module>:6 - 2021-6
2022-09-28 08:00:35.742 | INFO     | __main__:<module>:6 - 2021-7
2022-09-28 08:00:43.598 | INFO     | __main__:<module>:6 - 2021-8
2022-09-28 08:00:51.692 | INFO     | __main__:<module>:6 - 2021-9
2022-09-28 08:00:58.971 | INFO     | __main__:<module>:6 - 2021-10
2022-09-28 08:01:06.895 | INFO     | __main__:<module>:6 - 2021-11
2022-09-28 08:01:15.681 | INFO     | __main__:<module>:6 - 2021-12
2022-09-28 08:01:23.733 | INFO     | __main__:<module>:6 - 2022-1
2022-09-28 08:01:31.619 | INFO     | __main__:<module>:6 - 2022-2
2022-09-28 08:01:39.593 | INFO     | __main__:<module>:6 - 2022-3
2022-09-28 08:01:48.146 | INFO     | __main__:<module>:6 - 2022-4
2022-09-28 08:01:56.195 | INFO     | __main__:<module>:6 - 2022-5
2022-09-28 08:02:04.334 | INFO     | __main__:<module>:6 - 2022-6
2022-09-28 08:02:13.152 | INFO     | __main__:<module>:6 - 2022-7
2022-09-28 08:02:21.341 | INFO     | __main__:<module>:6 - 2022-8
2022-09-28 08:02:29.474 | INFO     | __main__:<module>:6 - 2022-9
```

1.7.2 Construindo o DataFrame

[56]:
```
btc_time_series_df = pd.DataFrame(coin_info, columns=[c for c in sample_json.
↪keys()])
btc_time_series_df['date'] = pd.to_datetime(btc_time_series_df['date'])

btc_time_series_df.set_index('date', inplace=True)

btc_time_series_df['volume'] = pd.to_numeric(btc_time_series_df['volume'])
```

```
btc_time_series_df['quantity'] = pd.to_numeric(btc_time_series_df['quantity'])
```

[57]:
```
btc_time_series_df.head()
```

[57]:
```
            opening  closing  lowest  highest       volume   quantity  amount  \
date
2013-06-12   249.00   265.00  249.00   275.00  2799.690778  10.916965      11
2013-06-13   265.00   269.00  259.00   269.00  2830.406722  10.624724      16
2013-06-14   267.00   250.00  245.00   268.00  8694.710569  34.040328      35
2013-06-15   250.00   246.01  246.01   259.99  4481.405612  17.445940       8
2013-06-16   246.01   252.00  246.01   257.43   427.690102   1.669200      14

             avg_price
date
2013-06-12  256.453225
2013-06-13  266.398141
2013-06-14  255.423818
2013-06-15  256.873845
2013-06-16  256.224600
```

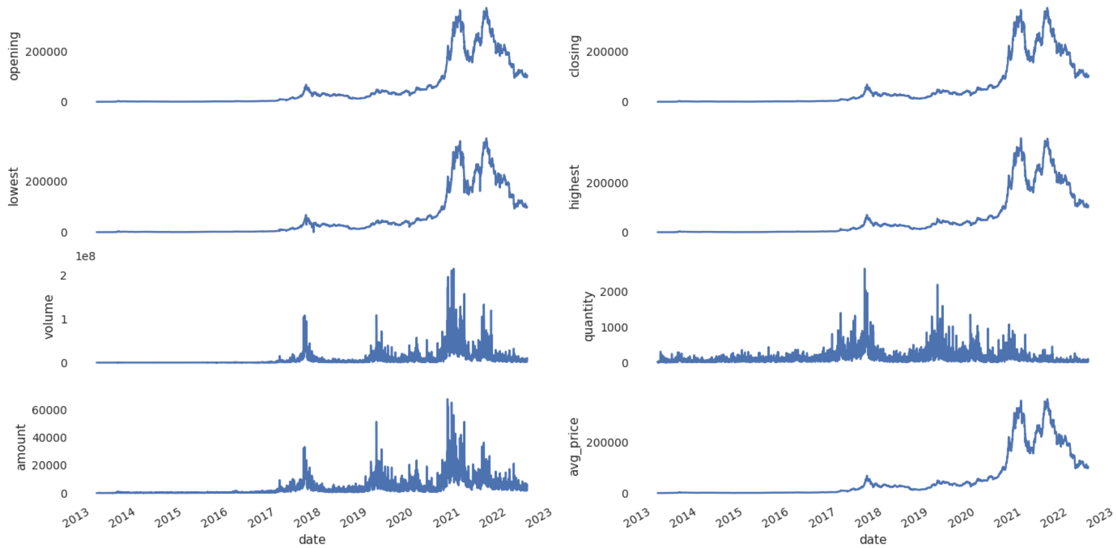
[58]:
```
btc_time_series_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3421 entries, 2013-06-12 to 2022-09-27
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   opening    3421 non-null   float64
 1   closing    3421 non-null   float64
 2   lowest     3421 non-null   float64
 3   highest    3421 non-null   float64
 4   volume     3421 non-null   float64
 5   quantity   3421 non-null   float64
 6   amount     3421 non-null   int64
 7   avg_price  3421 non-null   float64
dtypes: float64(7), int64(1)
memory usage: 240.5 KB
```

1.7.3 Explorando os Dados

[61]:
```
# Plot line charts
df_plot = btc_time_series_df.copy()

ncols = 2
nrows = int(round(df_plot.shape[1] / ncols, 0))
fig, ax = plt.subplots(nrows=nrows, ncols=ncols, sharex=True, figsize=(14, 7))
```

```
for i, ax in enumerate(fig.axes):
    sns.lineplot(data = df_plot.iloc[:, i], ax=ax)
    ax.tick_params(axis="x", rotation=30, labelsize=10, length=0)
    ax.xaxis.set_major_locator(mdates.AutoDateLocator())

fig.tight_layout()
plt.show()
```



1.7.4 Pré-Processamento e Escolha de Features

[62]:
```
# Indexing Batches
train_df = btc_time_series_df.sort_values(by=['date']).copy()

# List of considered Features
features = ['highest', 'lowest', 'opening', 'closing', 'volume']

print("FEATURE LIST:", features)

# Create the dataset with features and filter the data to the list of features
data = pd.DataFrame(train_df)
data_filtered = data[features]

# We add a prediction column and set dummy values to prepare the data for␣
↪scaling
data_filtered_ext = data_filtered.copy()
data_filtered_ext['prediction'] = data_filtered_ext['closing']

# Print the tail of the dataframe
```

```
data_filtered_ext.tail()
```

```
FEATURE LIST: ['highest', 'lowest', 'opening', 'closing', 'volume']
```

[62]:

```
                  highest         lowest        opening        closing  \
date
2022-09-23  102000.000000  97055.455946   99337.50000  100426.310000
2022-09-24  100793.455339  98928.750000  100480.00001   99205.893436
2022-09-25  100341.000000  98062.940000   99190.01000   99063.947493
2022-09-26  105950.000000  98334.070014   98957.15990  105950.000000
2022-09-27  108975.000000  99250.000000  105950.00000  100010.558789

                  volume     prediction
date
2022-09-23  5.201004e+06  100426.310000
2022-09-24  2.055391e+06   99205.893436
2022-09-25  1.827535e+06   99063.947493
2022-09-26  5.461422e+06  105950.000000
2022-09-27  1.041158e+07  100010.558789
```

[63]:

```
# Get the number of rows in the data
nrows = data_filtered.shape[0]

# Convert the data to numpy values
np_data_unscaled = np.array(data_filtered)
np_data = np.reshape(np_data_unscaled, (nrows, -1))
print(np_data.shape)

# Transform the data by scaling each feature to a range between 0 and 1
scaler = MinMaxScaler()
np_data_scaled = scaler.fit_transform(np_data_unscaled)

# Creating a separate scaler that works on a single column for scaling␣
 ↪predictions
scaler_pred = MinMaxScaler()
df_Close = pd.DataFrame(data_filtered_ext['closing'])
np_Close_scaled = scaler_pred.fit_transform(df_Close)
```

```
(3421, 5)
```

[66]:

```
# Set the sequence length - this is the timeframe used to make a single␣
 ↪prediction
sequence_length = 50

# Prediction Index
index_Close = data.columns.get_loc("closing")
```

```
# Split the training data into train and train data sets
# As a first step, we get the number of rows to train the model on 80% of the␣
↪data
train_data_len = math.ceil(np_data_scaled.shape[0] * 0.8)

# Create the training and test data
train_data = np_data_scaled[0:train_data_len, :]
test_data = np_data_scaled[train_data_len - sequence_length:, :]

# The RNN needs data with the format of [samples, time steps, features]
# Here, we create N samples, sequence_length time steps per sample, and 6␣
↪features
def partition_dataset(sequence_length, data):
    x, y = [], []
    data_len = data.shape[0]
    for i in range(sequence_length, data_len):
        x.append(data[i-sequence_length:i,:]) #contains sequence_length values␣
↪0-sequence_length * columsn
        y.append(data[i, index_Close]) #contains the prediction values for␣
↪validation,  for single-step prediction

    # Convert the x and y to numpy arrays
    x = np.array(x)
    y = np.array(y)
    return x, y

# Generate training data and test data
x_train, y_train = partition_dataset(sequence_length, train_data)
x_test, y_test = partition_dataset(sequence_length, test_data)

# Print the shapes: the result is: (rows, training_sequence, features)␣
↪(prediction value, )
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)

# Validate that the prediction value and the input match up
# The last close price of the second input sample should equal the first␣
↪prediction value
print(x_train[1][sequence_length-1][index_Close])
print(y_train[0])
```

```
(2687, 50, 5) (2687,)
(684, 50, 5) (684,)
0.0006284135835351043
0.0006284135835351043
```

1.7.5 Treinamento do Modelo

[68]:
```
# Configure the neural network model
model = Sequential()

# Model with n_neurons = inputshape Timestamps, each with x_train.shape[2]␣
↪variables
n_neurons = x_train.shape[1] * x_train.shape[2]
print(n_neurons, x_train.shape[1], x_train.shape[2])
model.add(LSTM(n_neurons, return_sequences=True, input_shape=(x_train.shape[1],␣
↪x_train.shape[2])))
model.add(LSTM(n_neurons, return_sequences=True, input_shape=(x_train.shape[1],␣
↪x_train.shape[2])))
model.add(LSTM(n_neurons, return_sequences=False))
model.add(Dense(5))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mse')
```

```
250 50 5
```

[70]:
```
# Training the model
epochs = 50
batch_size = 16
early_stop = EarlyStopping(monitor='loss', patience=5, verbose=1)
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_test, y_test)
                   )
```

```
Epoch 1/50
168/168 [==============================] - 2s 13ms/step - loss: 6.7232e-05 -
val_loss: 0.0025
Epoch 2/50
168/168 [==============================] - 2s 9ms/step - loss: 5.5785e-05 -
val_loss: 0.0021
Epoch 3/50
168/168 [==============================] - 2s 9ms/step - loss: 3.4063e-05 -
val_loss: 0.0049
Epoch 4/50
168/168 [==============================] - 2s 9ms/step - loss: 2.9751e-05 -
val_loss: 0.0035
Epoch 5/50
168/168 [==============================] - 2s 9ms/step - loss: 2.8009e-05 -
val_loss: 0.0025
Epoch 6/50
```

```
168/168 [==============================] - 2s 9ms/step - loss: 2.6494e-05 -
val_loss: 0.0023
Epoch 7/50
168/168 [==============================] - 2s 9ms/step - loss: 2.8500e-05 -
val_loss: 0.0015
Epoch 8/50
168/168 [==============================] - 2s 9ms/step - loss: 2.3462e-05 -
val_loss: 0.0015
Epoch 9/50
168/168 [==============================] - 2s 9ms/step - loss: 2.4675e-05 -
val_loss: 8.3607e-04
Epoch 10/50
168/168 [==============================] - 2s 9ms/step - loss: 1.8992e-05 -
val_loss: 0.0010
Epoch 11/50
168/168 [==============================] - 2s 9ms/step - loss: 2.0263e-05 -
val_loss: 7.9106e-04
Epoch 12/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6047e-05 -
val_loss: 5.8794e-04
Epoch 13/50
168/168 [==============================] - 2s 9ms/step - loss: 2.2565e-05 -
val_loss: 0.0025
Epoch 14/50
168/168 [==============================] - 2s 9ms/step - loss: 1.9561e-05 -
val_loss: 6.0082e-04
Epoch 15/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7878e-05 -
val_loss: 0.0045
Epoch 16/50
168/168 [==============================] - 2s 9ms/step - loss: 1.8359e-05 -
val_loss: 8.1962e-04
Epoch 17/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6328e-05 -
val_loss: 0.0017
Epoch 18/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7466e-05 -
val_loss: 9.3486e-04
Epoch 19/50
168/168 [==============================] - 2s 9ms/step - loss: 2.1576e-05 -
val_loss: 8.2825e-04
Epoch 20/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7184e-05 -
val_loss: 0.0015
Epoch 21/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6625e-05 -
val_loss: 7.9800e-04
Epoch 22/50
```

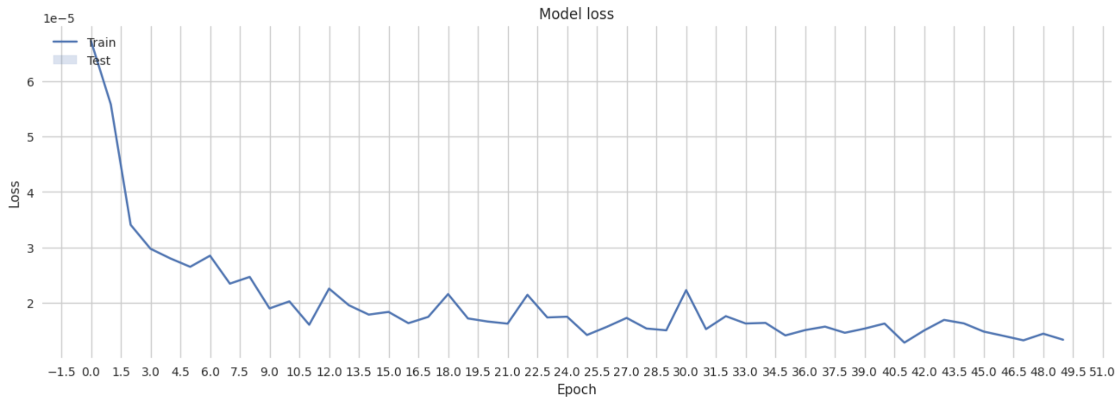
```
168/168 [==============================] - 2s 9ms/step - loss: 1.6246e-05 -
val_loss: 5.9921e-04
Epoch 23/50
168/168 [==============================] - 2s 9ms/step - loss: 2.1444e-05 -
val_loss: 8.2576e-04
Epoch 24/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7363e-05 -
val_loss: 0.0030
Epoch 25/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7497e-05 -
val_loss: 0.0015
Epoch 26/50
168/168 [==============================] - 2s 9ms/step - loss: 1.4200e-05 -
val_loss: 0.0033
Epoch 27/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5668e-05 -
val_loss: 5.3791e-04
Epoch 28/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7280e-05 -
val_loss: 0.0016
Epoch 29/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5363e-05 -
val_loss: 8.3880e-04
Epoch 30/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5047e-05 -
val_loss: 6.4881e-04
Epoch 31/50
168/168 [==============================] - 2s 9ms/step - loss: 2.2290e-05 -
val_loss: 0.0014
Epoch 32/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5255e-05 -
val_loss: 0.0020
Epoch 33/50
168/168 [==============================] - 2s 9ms/step - loss: 1.7593e-05 -
val_loss: 4.5250e-04
Epoch 34/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6274e-05 -
val_loss: 4.9595e-04
Epoch 35/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6379e-05 -
val_loss: 4.6611e-04
Epoch 36/50
168/168 [==============================] - 2s 9ms/step - loss: 1.4120e-05 -
val_loss: 5.3798e-04
Epoch 37/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5085e-05 -
val_loss: 9.6395e-04
Epoch 38/50
```

```
168/168 [==============================] - 2s 9ms/step - loss: 1.5706e-05 -
val_loss: 0.0013
Epoch 39/50
168/168 [==============================] - 2s 9ms/step - loss: 1.4597e-05 -
val_loss: 0.0012
Epoch 40/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5355e-05 -
val_loss: 4.8376e-04
Epoch 41/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6263e-05 -
val_loss: 5.8874e-04
Epoch 42/50
168/168 [==============================] - 2s 9ms/step - loss: 1.2822e-05 -
val_loss: 4.9071e-04
Epoch 43/50
168/168 [==============================] - 2s 9ms/step - loss: 1.5026e-05 -
val_loss: 5.3729e-04
Epoch 44/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6922e-05 -
val_loss: 4.6224e-04
Epoch 45/50
168/168 [==============================] - 2s 9ms/step - loss: 1.6290e-05 -
val_loss: 5.1633e-04
Epoch 46/50
168/168 [==============================] - 2s 9ms/step - loss: 1.4815e-05 -
val_loss: 5.6788e-04
Epoch 47/50
168/168 [==============================] - 2s 9ms/step - loss: 1.4037e-05 -
val_loss: 0.0011
Epoch 48/50
168/168 [==============================] - 2s 9ms/step - loss: 1.3237e-05 -
val_loss: 7.1813e-04
Epoch 49/50
168/168 [==============================] - 2s 9ms/step - loss: 1.4430e-05 -
val_loss: 4.9384e-04
Epoch 50/50
168/168 [==============================] - 2s 9ms/step - loss: 1.3347e-05 -
val_loss: 5.4244e-04
```

[71]:

```
# Plot training & validation loss values
fig, ax = plt.subplots(figsize=(16, 5), sharex=True)
sns.lineplot(data=history.history["loss"])
plt.title("Model loss")
plt.ylabel("Loss")
plt.xlabel("Epoch")
ax.xaxis.set_major_locator(plt.MaxNLocator(epochs))
plt.legend(["Train", "Test"], loc="upper left")
```

```
plt.grid()
plt.show()
```



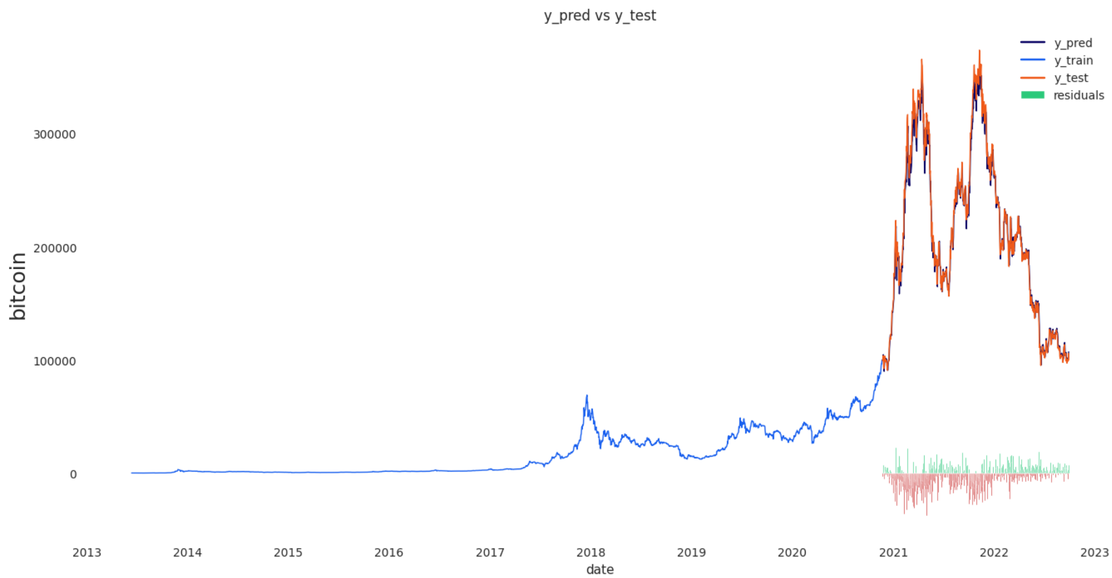
1.7.6 Avaliação da Performance do Modelo

[74]:
```
# Get the predicted values
y_pred_scaled = model.predict(x_test)

# Unscale the predicted values
y_pred = scaler_pred.inverse_transform(y_pred_scaled)
y_test_unscaled = scaler_pred.inverse_transform(y_test.reshape(-1, 1))

# Mean Absolute Error (MAE)
MAE = mean_absolute_error(y_test_unscaled, y_pred)
print(f"Median Absolute Error (MAE): {np.round(MAE, 2)}")

# Mean Absolute Percentage Error (MAPE)
MAPE = np.mean((np.abs(np.subtract(y_test_unscaled, y_pred)/ y_test_unscaled)))␣
↪* 100
print(f"Mean Absolute Percentage Error (MAPE): {np.round(MAPE, 2)} %")

# Median Absolute Percentage Error (MDAPE)
MDAPE = np.median((np.abs(np.subtract(y_test_unscaled, y_pred)/␣
↪y_test_unscaled)) ) * 100
print(f"Median Absolute Percentage Error (MDAPE): {np.round(MDAPE, 2)} %")
```

```
22/22 [==============================] - 0s 4ms/step
Median Absolute Error (MAE): 5690.19
Mean Absolute Percentage Error (MAPE): 2.63 %
Median Absolute Percentage Error (MDAPE): 1.82 %
```

[75]:

```
# The date from which on the date is displayed
display_start_date = "2013-06-12"

# Add the difference between the valid and predicted prices
train = pd.DataFrame(data_filtered_ext['closing'][:train_data_len + 1]).
  ↪rename(columns={'closing': 'y_train'})
valid = pd.DataFrame(data_filtered_ext['closing'][train_data_len:]).
  ↪rename(columns={'closing': 'y_test'})
valid.insert(1, "y_pred", y_pred, True)
valid.insert(1, "residuals", valid["y_pred"] - valid["y_test"], True)
df_union = pd.concat([train, valid])

# Zoom in to a closer timeframe
df_union_zoom = df_union[df_union.index > display_start_date]

# Create the lineplot
fig, ax1 = plt.subplots(figsize=(16, 8))
plt.title("y_pred vs y_test")
plt.ylabel("bitcoin", fontsize=18)
sns.set_palette(["#090364", "#1960EF", "#EF5919"])
sns.lineplot(data=df_union_zoom[['y_pred', 'y_train', 'y_test']], linewidth=1.
  ↪0, dashes=False, ax=ax1)

# Create the bar plot with the differences
df_sub = ["#2BC97A" if x > 0 else "#C92B2B" for x in df_union_zoom["residuals"].
  ↪dropna()]
ax1.bar(height=df_union_zoom['residuals'].dropna(),␣
  ↪x=df_union_zoom['residuals'].dropna().index, width=3, label='residuals',␣
  ↪color=df_sub)
plt.legend()
plt.show()
```



1.7.7 Prevendo o Valor do Ativo no Próximo Dia

[76]:
```
df_temp = btc_time_series_df[-sequence_length:]
new_df = df_temp.filter(features)

N = sequence_length

# Get the last N day closing price values and scale the data to be values␣
↪between 0 and 1
last_N_days = new_df[-sequence_length:].values
last_N_days_scaled = scaler.transform(last_N_days)

# Create an empty list and Append past N days
X_test_new = []
X_test_new.append(last_N_days_scaled)

# Convert the X_test data set to a numpy array and reshape the data
pred_price_scaled = model.predict(np.array(X_test_new))
pred_price_unscaled = scaler_pred.inverse_transform(pred_price_scaled.
↪reshape(-1, 1))

# Print last price and predicted price for the next day
price_today = np.round(new_df['closing'][-1], 2)
predicted_price = np.round(pred_price_unscaled.ravel()[0], 2)
change_percent = np.round(100 - (price_today * 100)/predicted_price, 2)

plus = '+'; minus = ''
print(f"The close price for bitcoin at {end_date} was {price_today}")
```

```
print(f"The predicted close price is {predicted_price} ({plus if change_percent␣
↪> 0 else minus}{change_percent}%)")
```

```
1/1 [==============================] - 0s 15ms/step
The close price for bitcoin at 2022-09-28 was 100010.56
The predicted close price is 101747.5234375 (+1.71%)
```

[ ]:

```

```