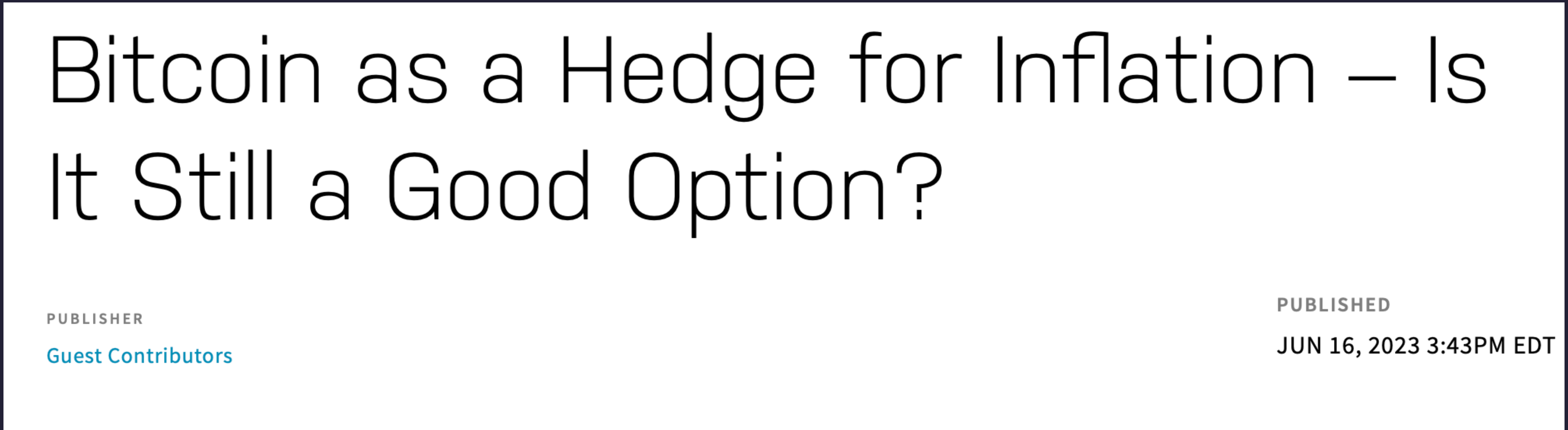
Advanced Pandas: Data Exploration using Pandas



Bitcoin as a Hedge for Inflation – Is It Still a Good Option?

PUBLISHER

Guest Contributors

PUBLISHED

JUN 16, 2023 3:43PM EDT

https://www.nasdaq.com/articles/bitcoin-as-a-hedge-for-inflation-is-it-still-a-good-option#:~:text=Bitcoin has potential as an,add a level of risk.

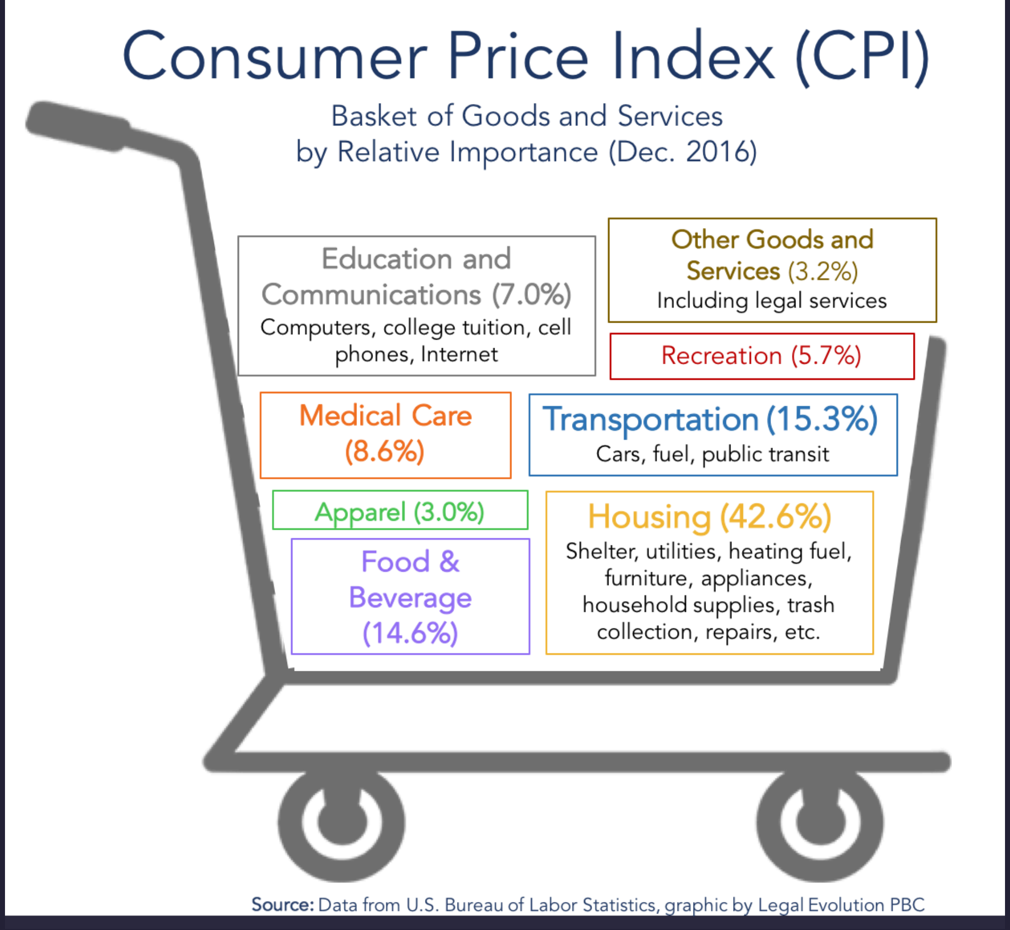
Bitcoin as a hedge for inflation?

Data pipeline:

- Bitcoin price data
- Inflation data
  - Consumer Price Index (CPI)
  - Global Price Index of all commodities (GPI)

Compare bitcoin price with the economic indicators

Price Index



Consumer Price Index (CPI)

Basket of Goods and Services
by Relative Importance (Dec. 2016)

Education and Communications (7.0%)
Computers, college tuition, cell phones, Internet

Other Goods and Services (3.2%)
Including legal services

Recreation (5.7%)

Medical Care (8.6%)

Transportation (15.3%)
Cars, fuel, public transit

Apparel (3.0%)

Food & Beverage (14.6%)

Housing (42.6%)
Shelter, utilities, heating fuel, furniture, appliances, household supplies, trash collection, repairs, etc.

Source: Data from U.S. Bureau of Labor Statistics, graphic by Legal Evolution PBC

1. **Define data requirements and database schema**
2. **Extract data from a source**
    i. identify endpoint, path, and query parameters from API documentation
    ii. request and get response from API
3. **Transform data**
    i. select relevant values from response
    ii. transform data into a format that can be loaded into a database
4. **Load data into database**
    i. create table
    ii. insert data into table

Create additional tables in the `coins.db` database

Table: `cpi`

Column	Type
date	VARCHAR(10)
cpi	FLOAT

Table: `gpi`

Column	Type
date	VARCHAR(10)
gpi	FLOAT

Extract CPI and GPI from Alpha Vantage

https://www.alphavantage.co/documentation/#economic-indicators

- Endpoint
- Path
- Query parameters

Docstrings and Type hinting

```python
def indicators_etl(indicator_type: str)->None:
    """Extract, transform, and load economic indicators from Alpha Vantage API
    Args:
        indicator_type (str): economic indicator
    Returns:
        None
    """

    response = extract_indicators(indicator_type)
    data = transform_indicators(response)
    load_indicators(data)
```

```python
# ETL pipeline for CPI
indicators_etl("CPI")

# ETL pipeline for GPI
indicators_etl("ALL_COMMODITIES")
```

`extract_indicators`

`transform_indicators`

`load_indicators`

Read Bitcoin price, CPI, GPI from database as DataFrame

Aggregation

Conditional assignment

Missing data handling

Data visualization

More aggregate functions

- `last` (`first`): last (first) value in a group
- `nth`: nth value in a group
- `diff`: difference from the previous value
- `pct_change`: percentage change from the previous value
- `nunique`: number of unique values in a group

…

---

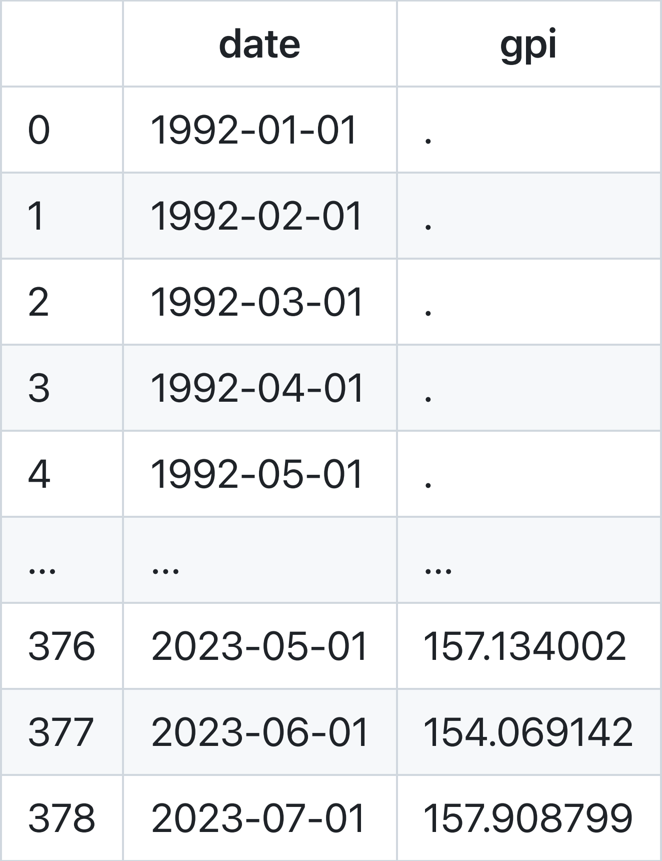
https://pandas.pydata.org/pandas-docs/stable/reference/groupby.html#aggregation

Aggregate daily Bitcoin price to monthly

Calculate Bitcoin monthly return

Calculate monthly inflation rate (CPI)

GPI data



	date	gpi
0	1992-01-01	.
1	1992-02-01	.
2	1992-03-01	.
3	1992-04-01	.
4	1992-05-01	.
...	...	...
376	2023-05-01	157.134002
377	2023-06-01	154.069142
378	2023-07-01	157.908799

```
# error
# can't calculate pct_change with a dot
rate = gpi_df.agg({"gpi": "pct_change"})

# gpi data type is object, not float
gpi_df['gpi'].dtype
# dtype('O')

cpi_df['cpi'].dtype
# dtype('float64')
```

Data types for missing values

Types:

- `None`: Python's built-in missing value
- `pd.NA`: Pandas's missing value
- `np.nan`: Numpy's missing value

Advantages:

- Compatible with numerical data types
- Numerical operations supported
- Easy missing value detection and handling

```
df = pd.DataFrame({
    'a': [1, 2, None],
    'b': [1, 2, pd.NA],
    'c': [1, 2, np.nan]
})
# outputs
#     a     b    c
# 0  1.0  1    1.0
# 1  2.0  2    2.0
# 2  NaN  <NA> NaN

df['a'].dtype
# dtype('float64')
```

Replace `.` with missing value

If a row has a value of `.`, replace it with a missing value

```
# error: The truth value of a Series is ambiguous
if gpi_df["gpi"] == ".":
    gpi_df["gpi"] = None
```

Two options:

- `apply()`: loop through the rows to update
- `loc[]`: select rows and update

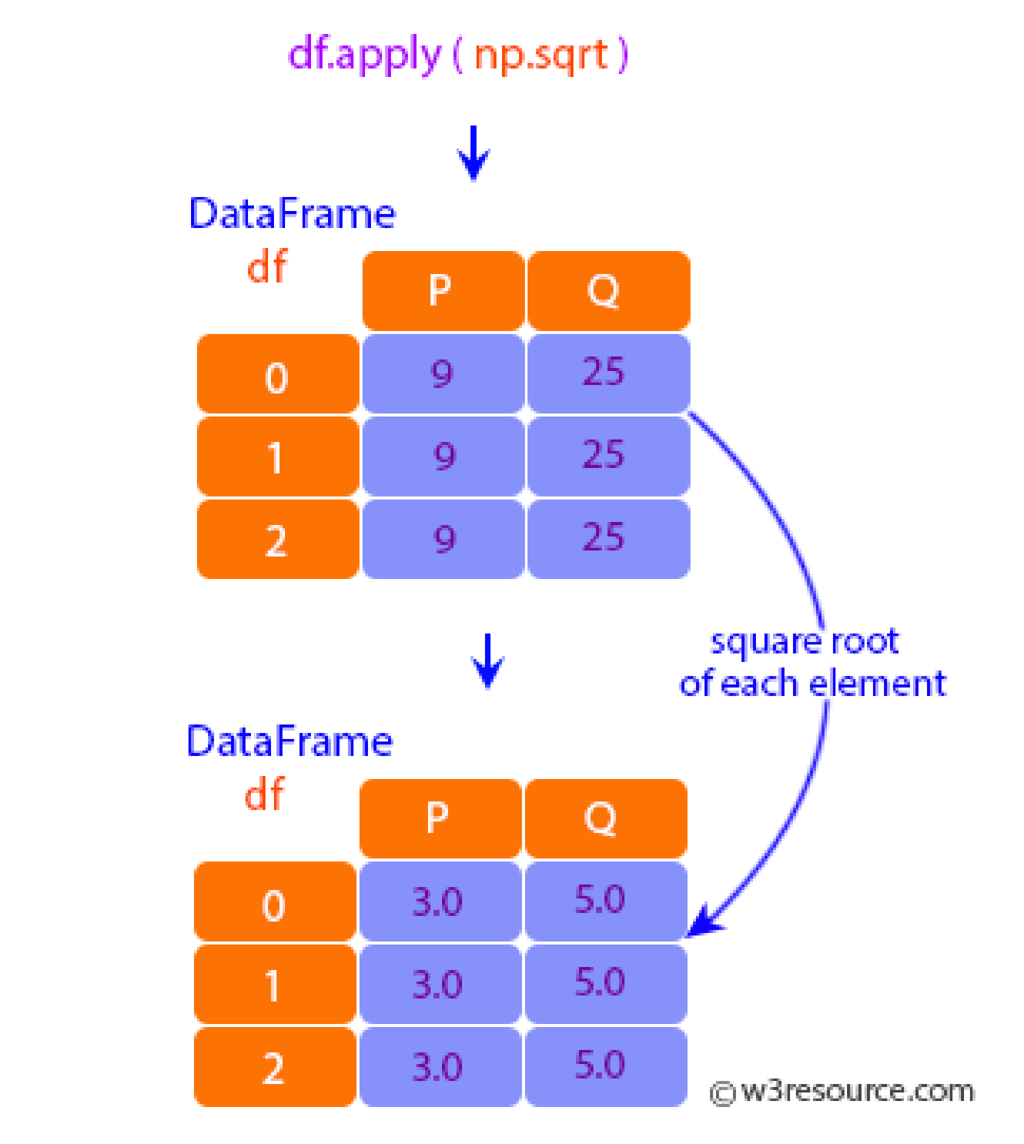
Loop through the rows

```python
def replace_dot(value):
    return None if value == "." else value

for value in gpi_df["gpi"]:
    gpi_df["gpi"] = replace_dot(value)
```

`apply` a function to each row or column

- method that applies **a function** along **an axis** of the DataFrame.
- `axis=0` applies function to each column (default)
- `axis=1` applies function to each row



df.apply (np.sqrt)

DataFrame
df

	P	Q
0	9	25
1	9	25
2	9	25

DataFrame
df

	P	Q
0	3.0	5.0
1	3.0	5.0
2	3.0	5.0

©w3resource.com

Use `apply` to replace `.` with missing value

`apply` `replace_dot` to the `gpi` column

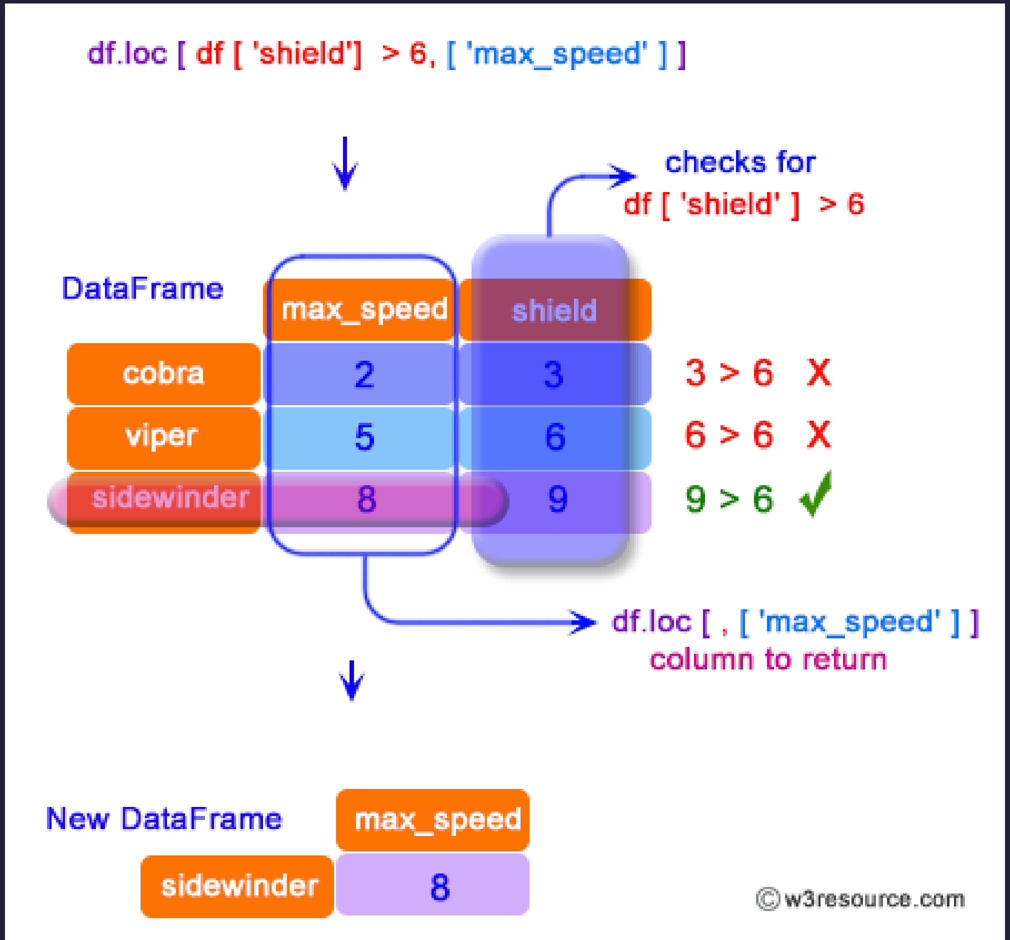
```python
def replace_dot(value):
    return None if value == "." else value

# Loop through gpi column
for value in gpi_df["gpi"]:
    gpi_df["gpi"] = replace_dot(value)

# Select the column and apply the function
gpi_df["gpi"] = gpi_df["gpi"].apply(replace_dot)

# Using lambda function
gpi_df["gpi"] = gpi_df.apply(lambda x: replace_dot(x["gpi"]), axis=1)
```

`loc` for filtering and updating



df.loc [df ['shield'] > 6, ['max_speed']]

checks for
df ['shield'] > 6

DataFrame

	max_speed	shield
cobra	2	3
viper	5	6
sidewinder	8	9

df.loc [, ['max_speed']]
column to return

New DataFrame

	max_speed
sidewinder	8

©w3resource.com

`query` for filtering

```
df.query('age > 25')
df.query('age > 25 and house == "Gryffindor"')
df.query('age > 25') = df['age'] * 2            # error
```

`loc` for filtering and updating

```
df.loc[df['age'] > 25]
df.loc[(df['age'] > 25) & (df['house'] == 'Gryffindor')]
df.loc[df['age'] > 25, 'age'] = df['age'] * 2   # ok
```

loc[row index, column index]

```
# select row 0
df.loc[0]

# error: no row named 'age'
df.loc['age']

# select column 'age' (all rows)
df.loc[:, 'age']

# select rows 0 to 3, columns 'age' and 'name'
df.loc[0:3, ['age', 'name']]

# select rows where age > 25, columns 'age' and 'name'
df.loc[df['age'] > 25, ['age', 'name']]
```

Use `loc` to replace `.` with missing value

```
gpi_df.loc[gpi_df["gpi"] == ".", "gpi"] = None
```

🖥️ Conditional assignment

Create a new column `positive` that is `True` if the monthly BTC return is positive, and `False` otherwise

- **Q1.** `apply`
  - Write a function `is_positive` that takes a value and returns `True` if the value is positive, and `False` otherwise
  - Use `apply` to apply the function to the `return` column
  - Assign the result to a new column `positive`
- **Q2.** `loc`
  - Use `loc` to select rows where `return` is positive
  - Assign `True` to the `positive` column in the selected rows

GPI data after replacing `.` with missing value

	date	gpi
0	1992-01-01	NaN
1	1992-02-01	NaN
2	1992-03-01	NaN
3	1992-04-01	NaN
4	1992-05-01	NaN
…	…	…
376	2023-05-01	157.134002
377	2023-06-01	154.069142
378	2023-07-01	157.908799

Handling missing data

- `isna()` : returns `True` if the value is missing, `False` otherwise
- `notna()` : returns `True` if the value is not missing, `False` otherwise
- `dropna()` : **drop rows with missing data**

```
# Count missing data in each column
gpi_df.isna().agg('sum')

# Count non-missing data in each column
gpi_df.notna().agg('sum')

# Drop rows with missing data and assign to gpi_df
gpi_df = gpi_df.dropna()

# Drop rows with missing data and assign to gpi_df
gpi_df = gpi_df.dropna(subset=["gpi"])
```

Missing data imputation

Year	Firm ID	Stock Price	Revenue	Earnings	Total Assets
2015	XYZ	85.50	1000	120	5000
2016	XYZ	90.00	1050	NaN	5200
2017	XYZ	NaN	1075	125	NaN
2018	XYZ	NaN	1100	130	5400
2019	XYZ	80.25	1150	NaN	5600
2020	XYZ	100.00	NaN	140	5800

Missing data imputation

```
# Fill missing data with 0
gpi_df = gpi_df.fillna(0)

# Fill missing data with the previous value (forward fill)
gpi_df = gpi_df.fillna(method="ffill")

# Fill missing data with the next value (backward fill)
gpi_df = gpi_df.fillna(method="bfill")

# Fill missing data with linear interpolation
gpi_df = gpi_df.interpolate(method="linear")
```

Calculate inflation rate (GPI)

`merge` Bitcoin price and economic indicators

Chaining Pandas methods

```
df = coins_monthly_df.merge(cpi_df, on="month").merge(gpi_df, on="month").dropna()[cols]
```

```
df = (
    coins_monthly_df                 # coins_monthly_df
    .merge(cpi_df, on="month")  # merge with cpi_df
    .merge(gpi_df, on="month")  # merge with gpi_df
    .dropna()                        # drop rows with missing data
    [cols]                           # select columns
)
```

Data Visualization

- Matplotlib
- Seaborn
- Bokeh
- Altair
- **Plotly**

Plotly Express Syntax

```python
import plotly.express as px

fig = px.scatter(df, x="age", y="height")
fig.show()
```

https://plotly.com/python/plotly-express/
https://plotly.com/python/px-arguments/

Line plot

```
# line plot for monthly return
fig = px.line(df, x="month", y="return")
fig.show()
```

```
# line plot for monthly return and inflation rate
fig = px.line(df, x="month", y=["return", "cpi_change"])
fig.show()
```

Moving average (`rolling`)

```python
# calculate 3-month moving average
df["return_ma"] = df["return"].rolling(3).mean()

# line plot for monthly return and 3-month moving average
fig = px.line(df, x="month", y=["return", "return_ma"])
fig.show()
```

Heatmap for correlation

```python
# correlation matrix
corr = df.corr()

# heatmap
fig = px.imshow(corr, color_continuous_scale="Redor")
fig.show()
```

color scale: https://plotly.com/python/builtin-colorscales/

Scatter plot with regression line

```python
fig = px.scatter(df, x="inflation", y="return", trendline="ols")
fig.show()
```

Regression using `statsmodels`

```python
import statsmodels.api as sm

X = df["cpi_change"]
y = df["return"]
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
model.summary()
```

https://www.statsmodels.org/stable/index.html

Add buttons to switch between charts

```
my_buttons = [
    {'label': 'Monthly Returns', 'method': 'update', 'args': [{'visible': [True, True, True]}]},
    {'label': 'Bitcoin', 'method': 'update', 'args': [{'visible': [True, False, False]}]},
    {'label': 'CPI', 'method': 'update', 'args': [{'visible': [False, True, False]}]},
    {'label': 'GPI', 'method': 'update', 'args': [{'visible': [False, False, True]}]},
]
layout = {
    'updatemenus': [{
        'type': 'buttons',
        'direction': 'down',
        'active': 0,
        'x': 1.2, 'y': 0.5,
        'buttons': my_buttons
    }]
}
fig = px.line(
    df,
    x='month',
    y=['return_ma', 'cpi_change_ma', 'gpi_change_ma'],
    title='Monthly Returns (Moving Average)'
)
fig.update_layout(layout)
fig.show()
```