# ETL and Data Exploration with Pandas

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

PUBLISHER

Guest Contributors

PUBLISHED

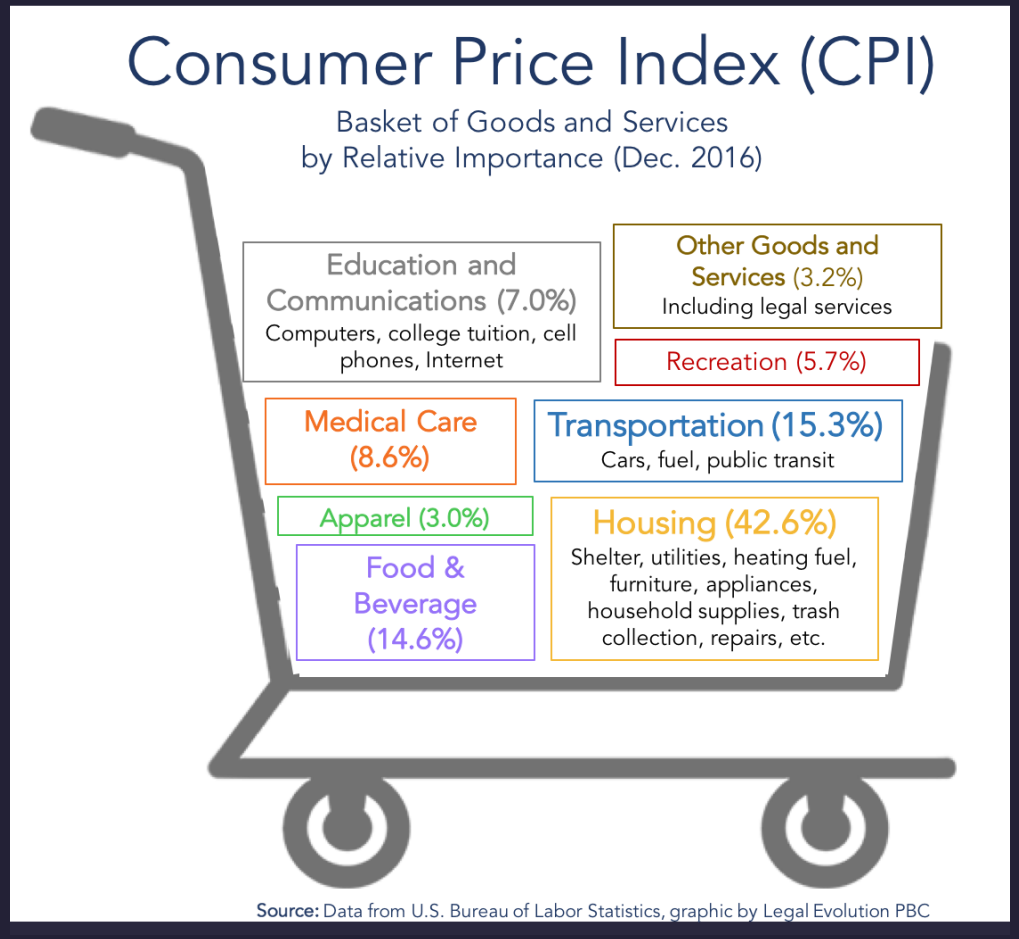
JUN 16, 2023 3:43PM EDT

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?

- Bitcoin price data
- Economic indicators for inflation
  - Consumer Price Index (CPI)
  - Global Price Index of all commodities (GPI)
- **Compare** bitcoin price with 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%)
Housing (42.6%)
Shelter, utilities, heating fuel, furniture, appliances, household supplies, trash collection, repairs, etc.
Food & Beverage (14.6%)
Source: Data from U.S. Bureau of Labor Statistics, graphic by Legal Evolution PBC

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

# Create new tables in the `coins` 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/

- Endpoint
- Path
- Query parameters

# Pseudocode

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

    response = extract_indicators(indicator)
    data = transform_indicators(response)
    load_indicators(data)

indicators_etl("CPI")
```

`extract_indicators`

# Accessing items in a list/tuple/dict

- list/tuple: `[index]`
- dict: `[key]`

# How to access the following items?

```
cities = [
    {"name": "Montreal", "state": "QC", "country": "CA"},
    {"name": "Toronto", "state": "ON", "country": "CA"},
    {"name": "Vancouver", "state": "BC", "country": "CA"},
    {"name": "Detroit", "state": "MI", "country": "US"}
]
```

- `{"name": "Vancouver", "state": "BC", "country": "CA"}`
- `"Vancouver"`
- `"Montreal"`

```
cities = {
    "name": ["Montreal", "Toronto", "Vancouver", "Detroit"],
    "state": ["QC", "ON", "BC", "MI"],
    "country": ["CA", "CA", "CA", "US"]
}
```

- `["Montreal", "Toronto", "Vancouver", "Detroit"]`
- `"Montreal"`

```
cities = {
    "Montreal": {"state": "QC", "country": "CA"},
    "Toronto": {"state": "ON", "country": "CA"},
    "Vancouver": {"state": "BC", "country": "CA"},
    "Detroit": {"state": "MI", "country": "US"}
}
```

- `{"state": "QC", "country": "CA"}`
- `"QC"`

```
cities = {
    "location": {
        "Montreal": {"state": "QC", "country": "CA"},
        "Toronto": {"state": "ON", "country": "CA"},
        ...
    },
    "stats": {
        "Montreal": [
            {"year": 2013, "population": 2000000, "area": 431.5},
            {"year": 2014, "population": 1980000, "area": 431.5}
            ],
        "Toronto": [
            {"year": 2013, "population": 2800000, "area": 630.2},
            ],
        ...
    }
}
```

- `{"year": 2013, "population": 2000000, "area": 431.5}`
- `"QC"`
- `2000000`

```
response = {
    "Meta Data": {
        "1. Information": "Daily Prices and Volumes for Digital Currency",
        "2. Digital Currency Code": "BTC",
        ...
    },
    "Time Series (Digital Currency Daily)": {
        "2023-10-30": {
            ...
            "4a. close (CNY)": "252122.24151800",
            "4b. close (USD)": "34456.58000000",
            ...
        },
        "2023-10-30": {...},
    }
}
```

- `"BTC"`
- `"34456.58000000"`

# Select CPI from response

```
response = {
    "name": "Consumer Price Index for all Urban Consumers",
    "interval": "monthly",
    "unit": "index 1982-1984=100",
    "data": [
        {
            "date": "2023-09-01",
            "value": "307.789"
        },
        {
            "date": "2023-08-01",
            "value": "307.026"
        },
        ...
    ]
}
```

`transform_indicators`

`load_indicators`

# Read Bitcoin price, CPI, and GPI from database as DataFrame

# Aggregate Bitcoin price by month

# 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

# 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 |

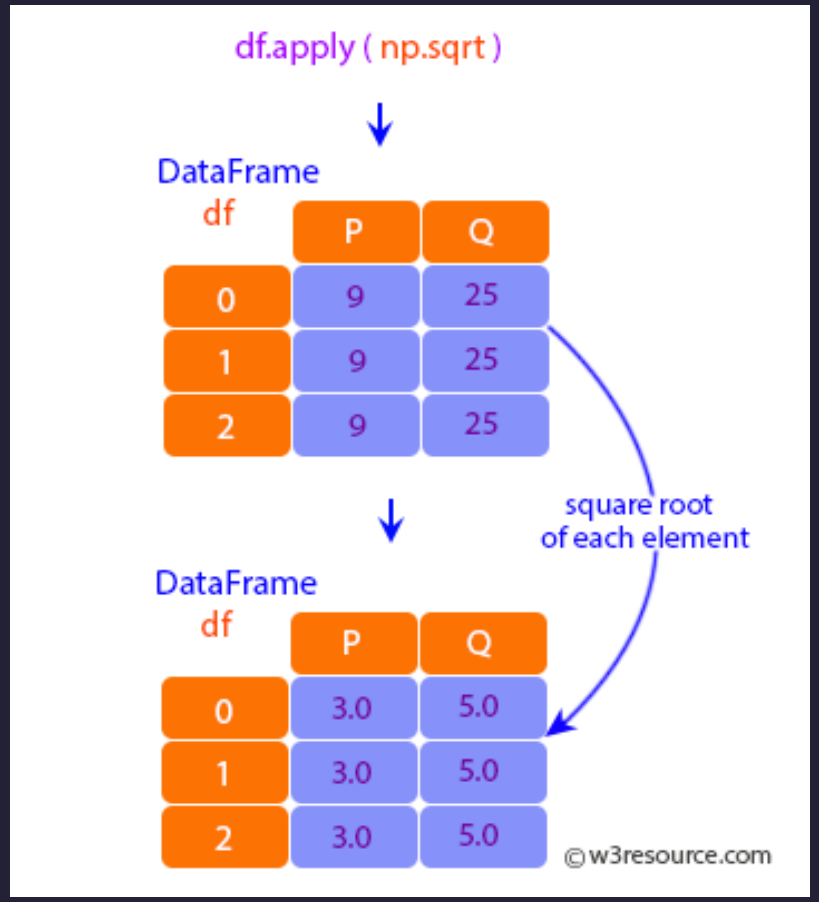
# Replace `.` with missing value

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

Missing values:

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

# apply()


df.apply ( np.sqrt )
DataFrame
df
P
Q
0
9
25
1
9
25
2
9
25
square root
of each element
DataFrame
df
P
Q
0
3.0
5.0
1
3.0
5.0
2
3.0
5.0
©w3resource.com

# Conditional column creation using `apply`

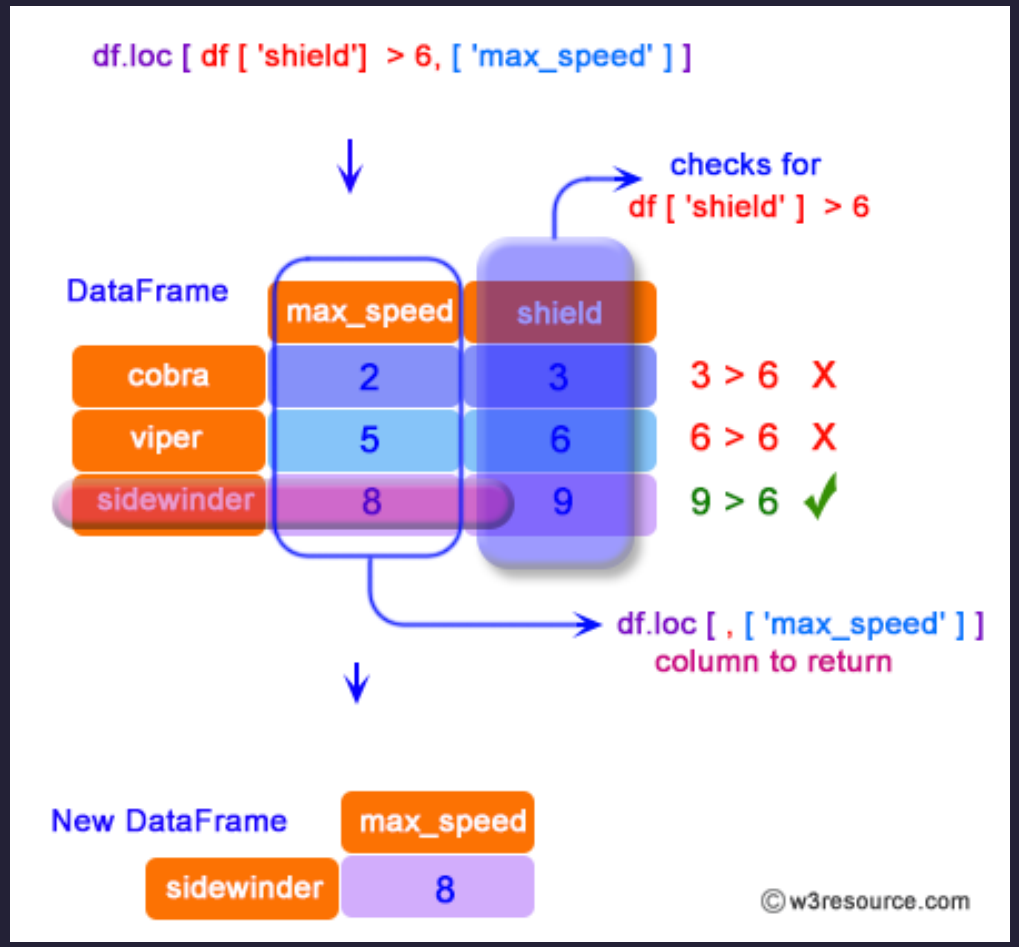
`apply`

- 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

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

gpi_df["gpi"] = gpi_df["gpi"].apply(replace_dot)
```

# loc[index, column]



## `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[index, column]

```
# 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']]
```

# Conditional column creation using `loc`

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

# 💻 Conditional column creation

**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

| | date | gpi |
|---|---|---|
| 0 | 1992-01-01 | None |
| 1 | 1992-02-01 | None |
| 2 | 1992-03-01 | None |
| 3 | 1992-04-01 | None |
| 4 | 1992-05-01 | None |
| ... | ... | ... |
| 376 | 2023-05-01 | 157.134002 |

# 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 | None | 5200 |
| 2017 | XYZ | None | 1075 | 125 | None |
| 2018 | XYZ | None | 1100 | 130 | 5400 |
| 2019 | XYZ | 80.25 | 1150 | None | 5600 |
| 2020 | XYZ | 100.00 | None | 140 | 5800 |

# Missing data imputation

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

# Fill missing data with the previous value and assign to gpi_df
gpi_df = gpi_df.fillna(method="ffill")

# Fill missing data with the next value and assign to gpi_df
gpi_df = gpi_df.fillna(method="bfill")

# Interpolate missing data with a linear method and assign to gpi_df
gpi_df = gpi_df.interpolate(method="linear")
```

# Calculate inflation rate (GPI)

# `merge` Bitcoin price and economic indicators

# Chaining 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

```
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`)

```
# 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

```
# 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

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

# Regression using `statsmodels`

```
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()
```