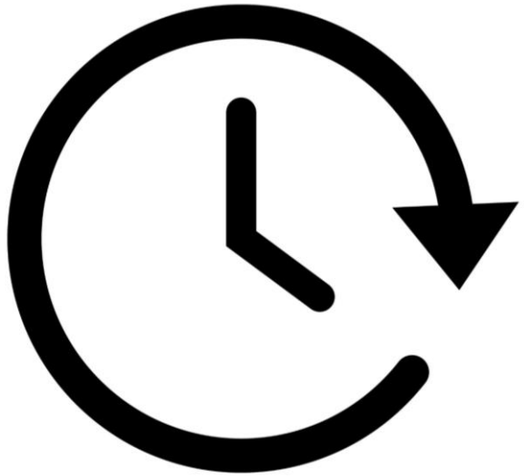
Datetime e Regex



Reg[ular]

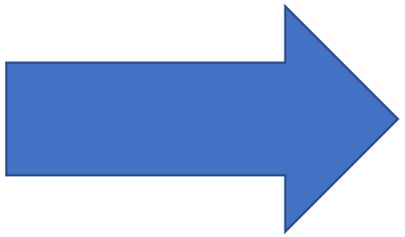
Ex[pression]

Professor: Alex Pereira

Resetar o índice de um dataframe

- df.reset_index()

	0
2018-01-01 00:00:00	0
2018-01-01 01:00:00	1
2018-01-01 02:00:00	2
2018-01-01 03:00:00	3
2018-01-01 04:00:00	4
2018-01-01 05:00:00	5
2018-01-01 06:00:00	6
2018-01-01 07:00:00	7
2018-01-01 08:00:00	8
2018-01-01 09:00:00	9



	index	0
0	2018-01-01 00:00:00	0
1	2018-01-01 01:00:00	1
2	2018-01-01 02:00:00	2
3	2018-01-01 03:00:00	3
4	2018-01-01 04:00:00	4
5	2018-01-01 05:00:00	5
6	2018-01-01 06:00:00	6
7	2018-01-01 07:00:00	7
8	2018-01-01 08:00:00	8
9	2018-01-01 09:00:00	9

Manipulação de Datas em Python

Biblioteca Datetime do Python

- Importar a biblioteca datetime
  - `from datetime import datetime`
- Criar data a partir dos números do ano, dia e mês
  - `datetime(2020, 5, 17)`
- Converter string para data
  - `datetime.strptime('18/09/19 01:55:19', '%d/%m/%y %H:%M:%S')`
    - ✓ Tabela de códigos (diretivas)
      - https://docs.python.org/3.7/library/datetime.html#strftime-and-strptime-behavior
- Converter datetime para string
  - `d.strftime('%Y-%m-%d')`
    - ✓ # d é um objeto do tipo datetime
- Date time de agora
  - `datetime.now()`

Biblioteca Datetime do Python

- Alguns Atributos interessantes de um objeto datetime
  - `d.day, d.month, d.year, d.hour, d.minute`
- Dia da semana
  - `d.weekday()`
- timedelta (intervalo de tempo)
  - `from datetime import timedelta`
  - `d = timedelta(weeks=2)`
    - ✓ Um interval de 2 semanas
- Aritmética de datas
  - `datetime.datetime.now() + timedelta(days=15)`
  - Subtração
    - ✓ `date1 = datetime(2017, 6, 21, 18, 25, 30)`
    - ✓ `date2 = datetime(2017, 5, 16, 8, 21, 10)`
    - ✓ `date1 - date2`

Biblioteca Datetime do Python

- Locale (configurações específicas de língua)
  - Útil para exibir nomes (dias, meses) em português
    - ✓ https://wiki.archlinux.org/index.php/Locale_(Portugu%C3%AAs)
  - No colab:

```
import locale
locale.getlocale()
```

```
('en_US', 'UTF-8')
```

- No Jupyter notebook é possível ajustar o locale

```
import locale
locale.setlocale(locale.LC_ALL, 'pt_BR.UTF-8')
```

- No Google Colab não é possível alterar o locale

Manipulação de Datas com Pandas

- pd.date_range
  - Amostras de datas em Intervalos fixos

```
1  pd.date_range("2018-01-01", periods=3, freq="H")
```

```
DatetimeIndex(['2018-01-01 00:00:00', '2018-01-01 01:00:00',
               '2018-01-01 02:00:00'],
              dtype='datetime64[ns]', freq='H')
```

- DatetimeIndex
  - Immutable ndarray-like of datetime64 data.
    - ✓ Um (parecido com) array imutável de dados do tipo datetime64
- Outros métodos
  - https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html

Reamostrar uma Série temporal

- pd.Series.resample

```
3 idx = pd.date_range("2018-01-01", periods=5, freq="H")
4 ts = pd.Series(range(len(idx)), index=idx)
5 ts
```

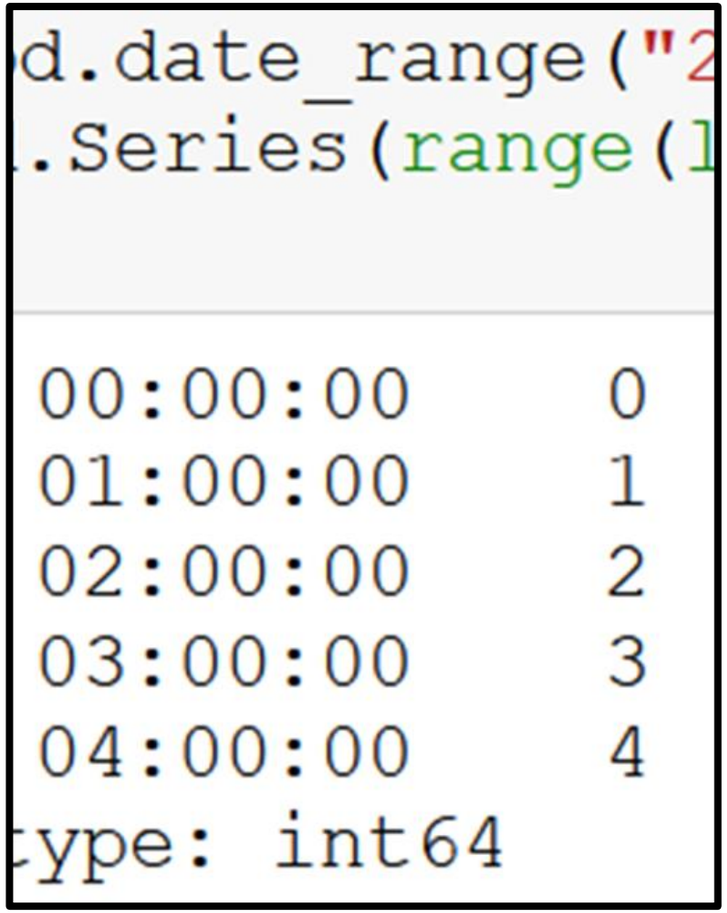
```
2018-01-01 00:00:00    0
2018-01-01 01:00:00    1
2018-01-01 02:00:00    2
2018-01-01 03:00:00    3
2018-01-01 04:00:00    4
Freq: H, dtype: int64
```

Reamostrar uma Série temporal (2)

- pd.Series.resample
  - Reamostrar os valores num intervalo de 2h

```
3  ts.resample("2H").mean()
```

```
2018-01-01 00:00:00    0.5
2018-01-01 02:00:00    2.5
2018-01-01 04:00:00    4.0
Freq: 2H, dtype: float64
```



String para Datetime com Pandas

- pd.to_datetime
  - O argumento de entrada (arg) pode ser dos tipos
    - ✓ int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like
- Converter uma string

```
2  pd.to_datetime("2010/11/12", format="%Y/%m/%d")
```

```
Timestamp('2010-11-12 00:00:00')
```

- Converter uma serie

```
2  s = pd.Series(['3/11/2000', '3/12/2000', '3/13/2000'])
3  pd.to_datetime(s, infer_datetime_format=True)
```

```
0   2000-03-11
1   2000-03-12
2   2000-03-13
dtype: datetime64[ns]
```

Regex em Python

Regex em Python

- Regex = Regular Expression
  - Uma expressão regular representa um conjunto de expressões/sentenças
    - ✓ Que seguem uma regra de construção
- Sintaxe das regex (Resumo)
  - [a-g] qualquer caractere entre a & g
  - \w \d \s palavra, dígito, espaço em branco
  - ^abc$ início / fim de uma string
  - a* a+ a? 0 ou mais, 1 ou mais 0 ou 1
  - a{5} a{2,} exatamente cinco, dois ou mais
  - a{1,3} entre um & três
  - ab|cd encontrar ab ou cd
  - \. * \\ caracteres especiais escapados

Exemplos de Regex

- CPF: 245.986.748-56
  - \d{3}\.\d{3}\.\d{3}-\d{2}
    - ✓ Mais conciso ?
    - ✓ E dessas maneiras 245986748-56 24598674856 ?
- CNPJ: 01.984.199/0001-07
  - \d{2}(\.\d{3}){2}\/\d{4}-\d{2}
- CPF ou CNPJ
  - \b(\d{3}\.?){2}\d{3}-?\d{2}\b|\b\d{2}\.?(\d{3}\.?){2}\/?\d{4}-?\d{2}\b
- Regex de número de telefone
  - https://medium.com/@igorrozani/criando-uma-express%C3%A3o-regular-para-telefone-fef7a8f98828

Testes online: https://regexr.com/

Regex com Python

```
1 import re
```

```
1 # Encontrar todos os termos de uma string que são compostos por letras
2 # ou números
3 g = re.findall('[a-zA-Z0-9]+',  'ABCDE2 2Fab.(cdef1 23 450 345#aaa')
4 g
```

```
['ABCDE2', '2Fab', 'cdef1', '23', '450', '345', 'aaa']
```

```
1 result1 = re.search('^\w+',  'teste 1')
2 print(result1)
```

```
<_sre.SRE_Match object; span=(0, 5), match='teste'>
```

```
1 result2 = re.search('^\w+',  '#teste 2')
2 print(result2)
```

```
None
```

Teste de Regex com Python

```
1  # Criar uma função pra tes
2  def test_regex(s):
3      pat = '^\w+'
4      if re.search(pat, s):
5          return True
6      else:
7          return False
8
9  test_regex('#teste 2')
```

Replace com Regex no Pandas

```
3  pd.Series(['foo', 'fuz', np.nan]).str.replace('f.', 'ba', regex=True)
```

```
0    bao
1    baz
2    NaN
```

```
3  pd.Series(['f.o', 'fuz', np.nan]).str.replace('f.', 'ba', regex=False)
```

```
0    bao
1    fuz
2    NaN
```

```
6  pd.Series(['foo 123', 'bar baz', np.nan]).str.replace('[a-z]+', repl)
```

```
0    oof 123
1    rab zab
2        NaN
```

Encontrar os caracteres que simbolizam NA

- Objetivo: identificar caracteres que não sejam
  - Números (0 a 9), vírgula e ponto
    - ✓ -?[0-9]+(.|,)?[0-9]*
  - df_gini = pd.read_csv(path_gini, sep=';', skiprows=2, skipfooter=2, encoding='utf8', engine='python', decimal=',', dtype={"1991": "str"})

	Município	1991	2000	2010
0	110001 Alta Floresta D'Oeste	0,5983	0,5868	0,5893
1	110037 Alto Alegre dos Parecis	...	0,508	0,5491
2	110040 Alto Paraíso	...	0,6256	0,5417
3	110034 Alvorada D'Oeste	0,569	0,6534	0,5355
4	110002 Ariquemes	0,5827	0,5927	0,5496

```
1 df_gini.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5565 entries, 0 to 5564
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Município   5565 non-null   object
 1   1991        5565 non-null   object
 2   2000        5565 non-null   object
 3   2010        5565 non-null   float64
dtypes: float64(1), object(3)
memory usage: 174.0+ KB
```

Encontrar os caracteres que simbolizam NA

- Solução
  - result = df_gini['1991'].apply(
    lambda x:
    x if not re.search('(-?(([0-9]+(.|,)?)+[0-9]*))', x) else np.nan
)

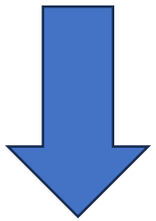
```
1  result.unique()
```

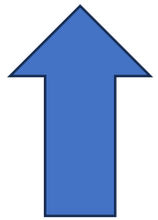
```
array([nan, '...'], dtype=object)
```

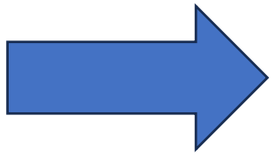
Apache
Airflow

Orquestrador de workflows e pipelines

Dashboard do Preço do Bitcoin

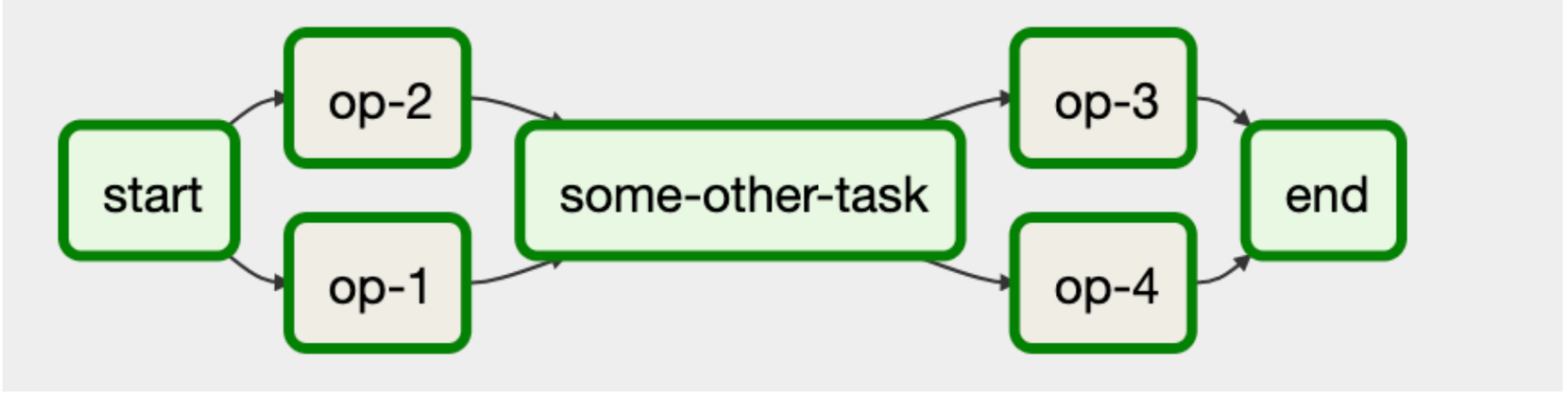






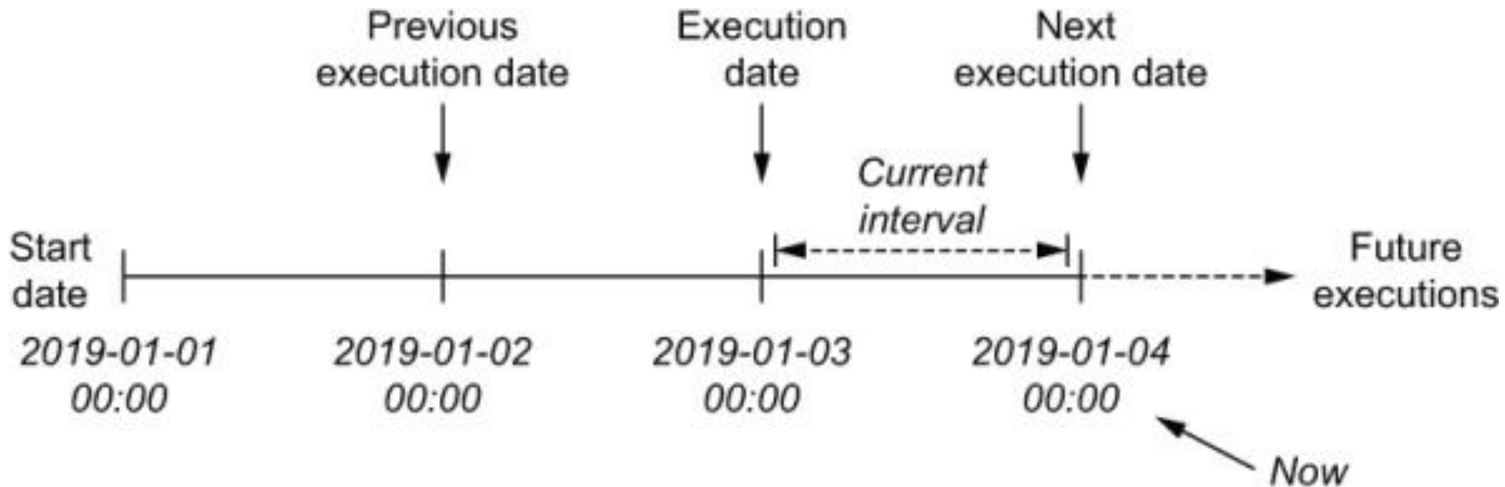
DAG (Directed Acyclic Graph)

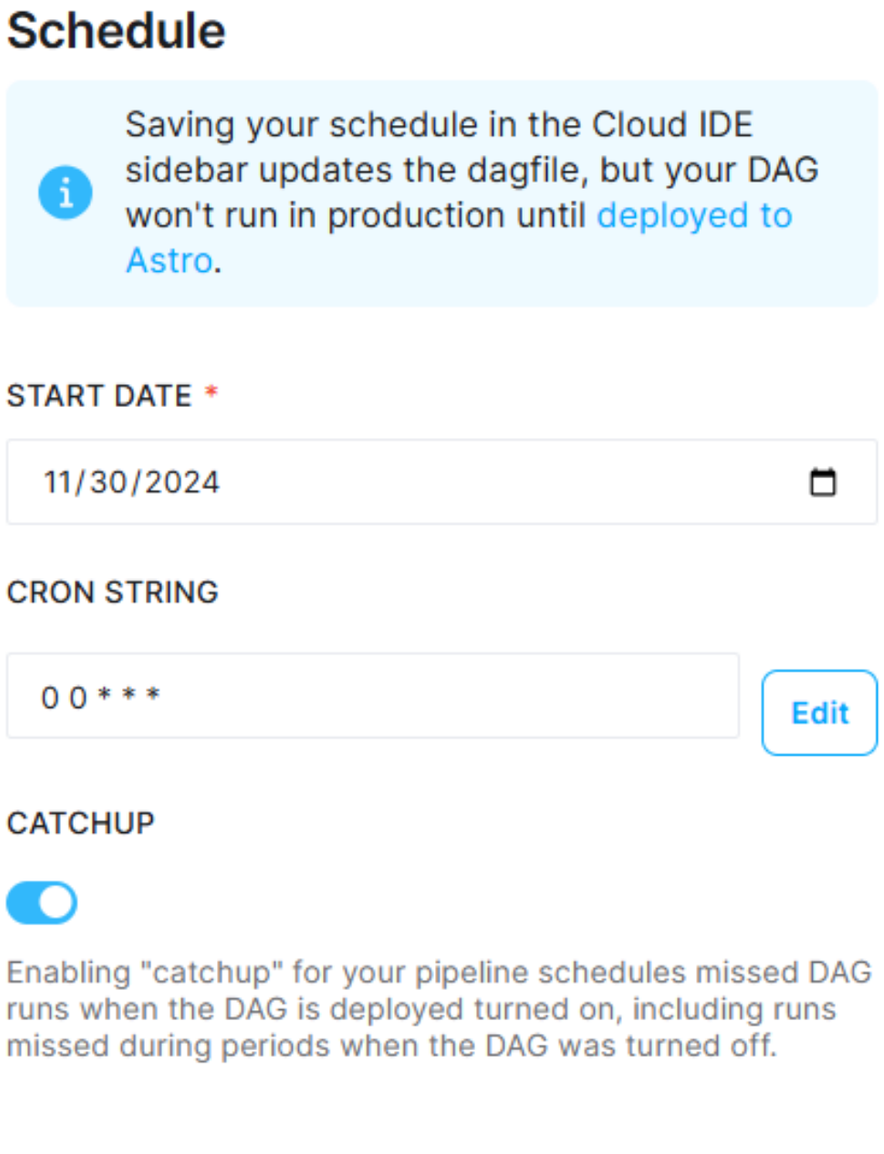
- Grafo acíclico direto
  - Sem laços
    - ✓ Tarefas sequenciais e paralelizáveis
- Operators
  - Building blocks do Airflow
    - ✓ Contém a lógica / implementação dos requisitos; e
    - ✓ Templates prontos pra configurar e usar.



Agendamento de Execução

- start_date
  - Data a partir da qual a CRON STRING
    - ✓ Passa a valer
- CRON STRING
  - Agendamento periódico
    - ✓ Ex.: todo dia, toda semana, todo mês, etc
- Catchup
  - Executa os agendamentos passados
    - ✓ Antes de hoje e depois de start_date
- execution_date
  - Data lógica para referência na DAG





Agendamento com cron

“At 04:05.”

next at 2024-12-05 04:05:00 random

5 4 * * *

minute	hour	day (month)	month	day (week)
		*	any value	
		,	value list separator	
		-	range of values	
		/	step values	
@yearly			(non-standard)	
@annually			(non-standard)	
@monthly			(non-standard)	
@weekly			(non-standard)	
@daily			(non-standard)	
@hourly			(non-standard)	

Coleta de dados do preço do bitcoin (coincap.io)

```python
from airflow.operators.python import get_current_context
context = get_current_context()
start_date = context['dag_run'].execution_date
print(f"DAG start date: {start_date}")

# Define the time range for yesterday
end_time = start_date.replace(hour=0, minute=0, second=0, microsec
start_time = end_time - timedelta(days=1)

# Convert to Unix timestamps in milliseconds
start_timestamp = int(start_time.timestamp() * 1000)
end_timestamp = int(end_time.timestamp() * 1000)

# CoinCap API endpoint for Bitcoin historical data
url = 'https://api.coincap.io/v2/assets/bitcoin/history'

# Parameters for the API request
params = {
    'interval': 'd1',  # diario
    'start': start_timestamp,
    'end': end_timestamp}
```

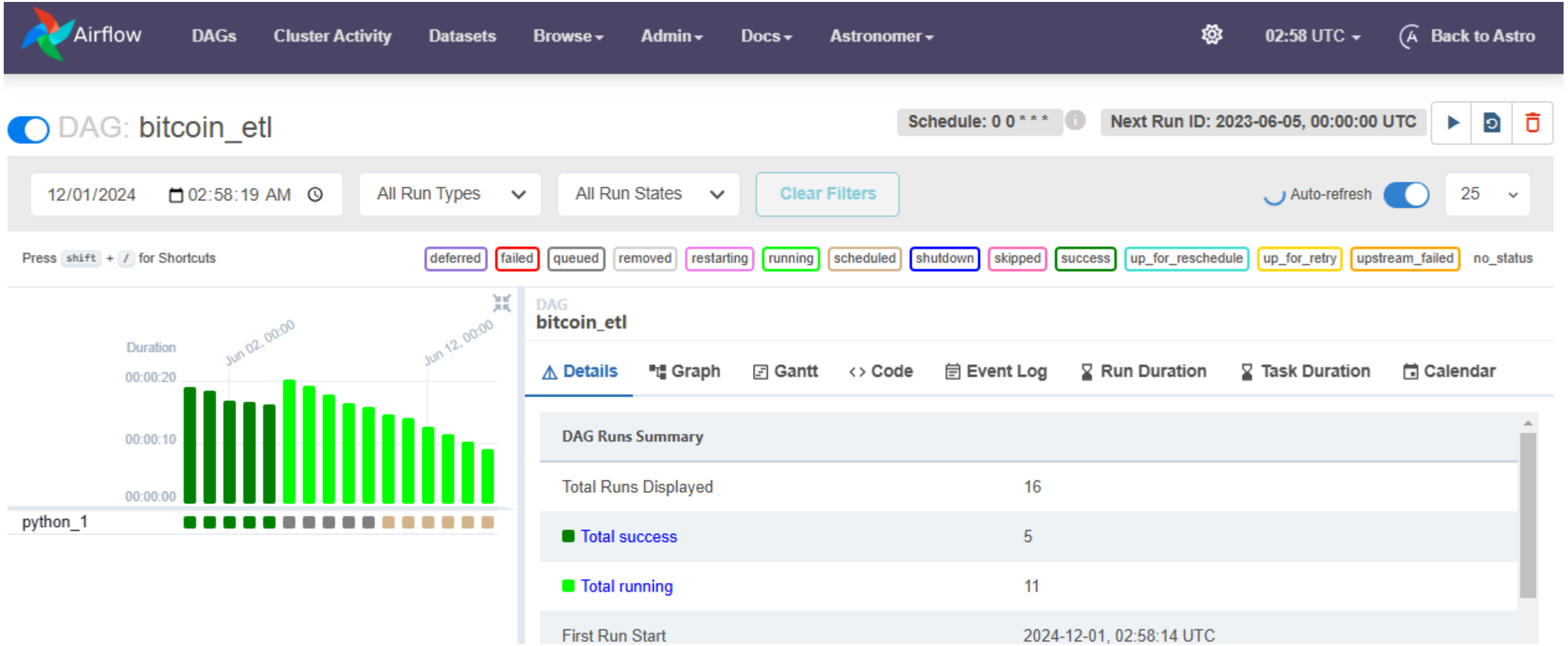
Coleta de dados do preço do bitcoin (coincap.io)

```
response = requests.get(url, params=params)
data = response.json()

# Check if data is available
if 'data' in data:
    # Convert data to pandas DataFrame
    df = pd.DataFrame(data['data'])
    # Convert time column to datetime
    df['time'] = pd.to_datetime(df['time'], unit='ms')
    # Set time as index
    df.set_index('time', inplace=True)
    # Display the DataFrame
    print(df)
else:
    print("No data available for the specified date range.")

#### Load
pg_hook = PostgresHook(postgres_conn_id='postgres_default')
engine = pg_hook.get_sqlalchemy_engine()
df.to_sql('bitcoin_history', con=engine, if_exists='append', index=
```

Coleta de 1 ano em andamento



- A TI pode fazer as coisas ficarem pior, mais rápido

Dashboard do Preço do Bitcoin

Astronomer.io e Airflow

- Tutorial sobre como https://youtu.be/sw-hATdQrBU
  - Configurar o Airflow no Astronomer;
  - Integrar com o github para fazer deploy automatizado; e
  - Geração de código de DAGs.

Exercício 9.5 (Vale 10 pontos)

- Replique o experimento descrito no tutorial em vídeo do Airflow
- Construa um dashboard com dados diários de 6 meses do preço do bitcoin
- Compartilhe neste formulário
  - Um print (screenshot) da sua tela mostrando a sua instância do Apache Airflow, as Tasks executadas e o log da execução.
  - A URL do github do código da sua DAG
  - A URL do seu dashboard **público** no Looker Studio

Prática no Google Colab

- Faça os exercícios da aula.
  - ✓ Datetime e Regex