CARRERA: ESPECIALIZACIÓN EN CIENCIA DE DATOS

Seminario de Tópicos Avanzados en Datos Complejos

C2020

TRABAJO PRÁCTICO FINAL

Integrantes:

Ariel Belossi

Álvaro Vanina

Tópicos:

Procesamiento Natural del Lenguaje

Superset

Airflow

1. Introducción

Para la realización del presente trabajo hemos utilizado como base la plataforma configurada durante la realización del seminario, la cual fue clonada y actualizada en el siguiente repositorio de github:

https://github.com/alvarov0907/tp_seminario.git

El trabajo consta principalmente de 3 secciones:

1. **Modelo de aprendizaje automático (ML):** para el cual hemos utilizado un dataset obtenido de kaggle y realizamos 2 modelos predictivos utilizando regresión logística y Random Forest

2. **Superset:** la notebook utilizada para generar los modelos de aprendizaje automático hacen una copia del dataset utilizado en una base de datos porstgres, sobre la cual se definieron un conjunto de gráficos en superset y luego confeccionamos un “Dashboard” de visualización.

3. **Airflow:** creamos 3 dags para obtener información pares de cotizaciones de monedas virtuales del exchange Binance.

2. Modelo de aprendizaje automático

Para llevar a cabo el modelo de aprendizaje automático efectuado los siguientes pasos:

Selección del dataset

Obtuvimos del sitio web “https://www.kaggle.com/” un dataset que contiene información de solicitudes de empleos de diferentes países. Dicho dataset contiene los siguientes campos:

- **job_id:** id del job
- **title:** título de la solicitud de empleo
- **location:** Ubicación del empleo (País, estado/provincia, localidad)
- **department:** Departamento / gerencia del puesto de trabajo
- **salary_range:** rango de salario
- **company_profile:** perfil de la compañía
- **description:** descripción del puesto de trabajo
- **requirements:** requerimientos necesarios para cubrir el puesto
- **benefits:** beneficios ofrecidos
- **telecommuting:** permite trabajo remoto (0:no, 1:si)
- **has_company_logo:** la compañía tiene logo (0:no, 1:si)
- **has_questions:** preguntas asociadas al puesto (0:no, 1:si)
- **employment_type:** tipo de empleo
- **required_experience:** experiencia requerida
- **required_education:** nivel de educación requerido.
- **industry:** industria de la empresa
- **function:** función asociada al puesto

- **fraudulent:** indica si la solicitud es verdadera o fraudulenta (0: verdadera, 1: fraudulenta). **Es variable a predecir.**

Url para descarga del dataset:

https://raw.githubusercontent.com/alvarov0907/tp_seminario/master/dataset/fake_job_postings.csv

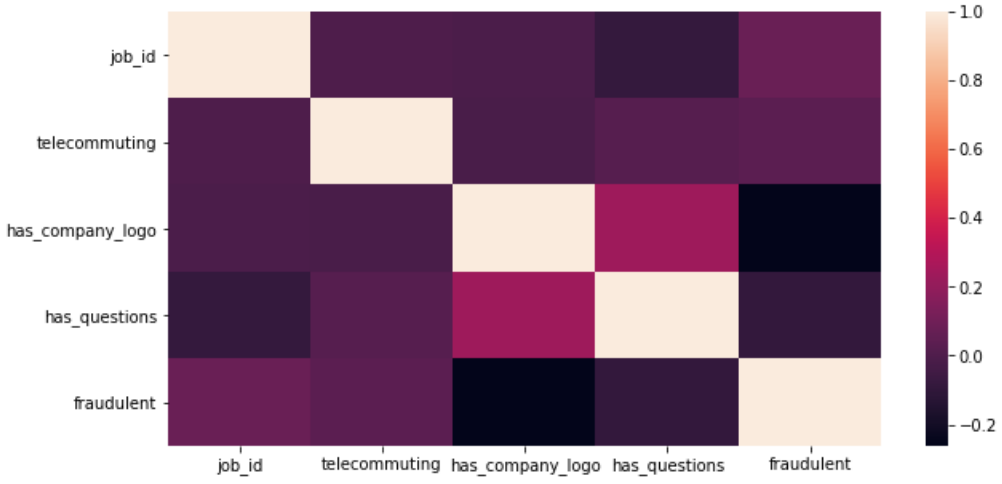
Instalación de paquetes adicionales:

La notebook creada tiene un primer apartado donde se instalan paquetes adicionales que no se encuentran cargados en la plataforma utilizada como base. Ellos son:

- Librerías de ML:
  - scikit-learn
  - sklearn_pandas
  - nltk
  - Pillow (versión 6.2.0)
  - wordcloud
  - imblearn
- Librerías para interactuar con base de datos:
  - psycopg2 (versión 2.7.6)
  - sqlalchemy

Análisis exploratorio:

Ejecutamos distintos comandos y gráficos para conocer los datos incluidos en el dataset. Por ejemplo, un mapa de calor que indica la correlación entre las variables.

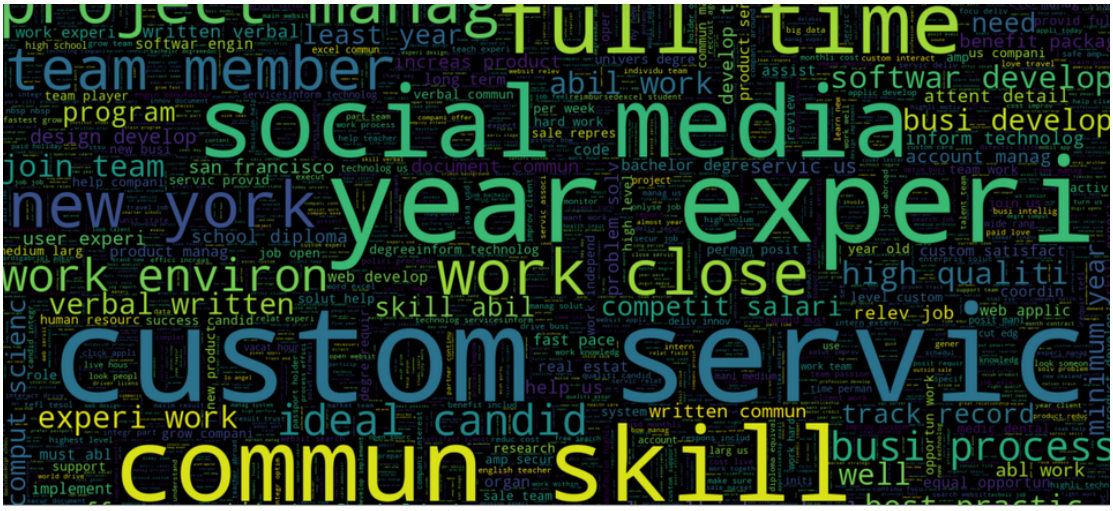


feature engineering:

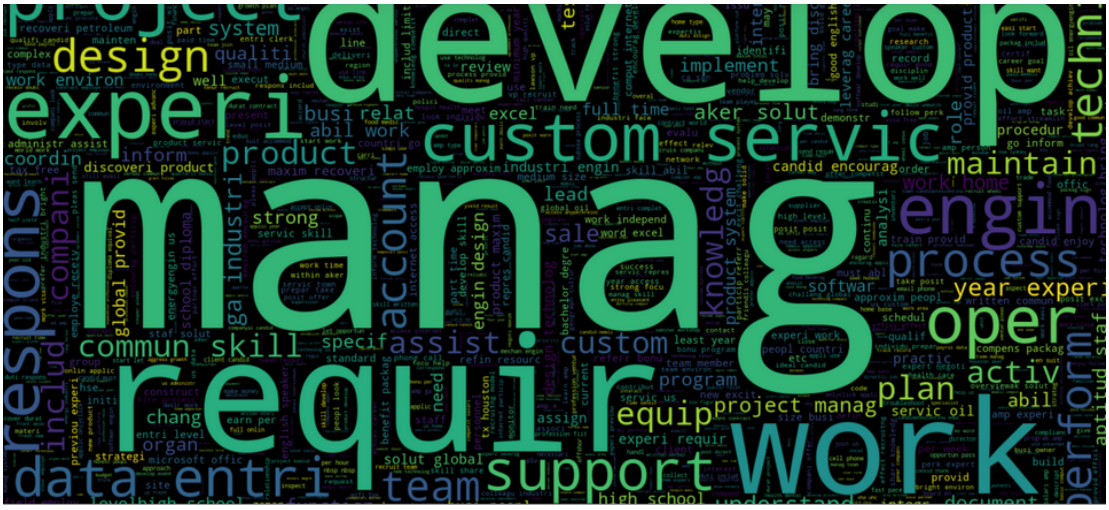
Para realizar el entrenamiento del modelo y la posterior verificación generamos un nuevo campo que contiene la suma (concatenación de todos los campos de texto del dataset).

Una vez generado el nuevo campo efectuamos una nube de palabras para evaluar el comportamiento de los empleos verdaderos y falsos.

Verdaderos:



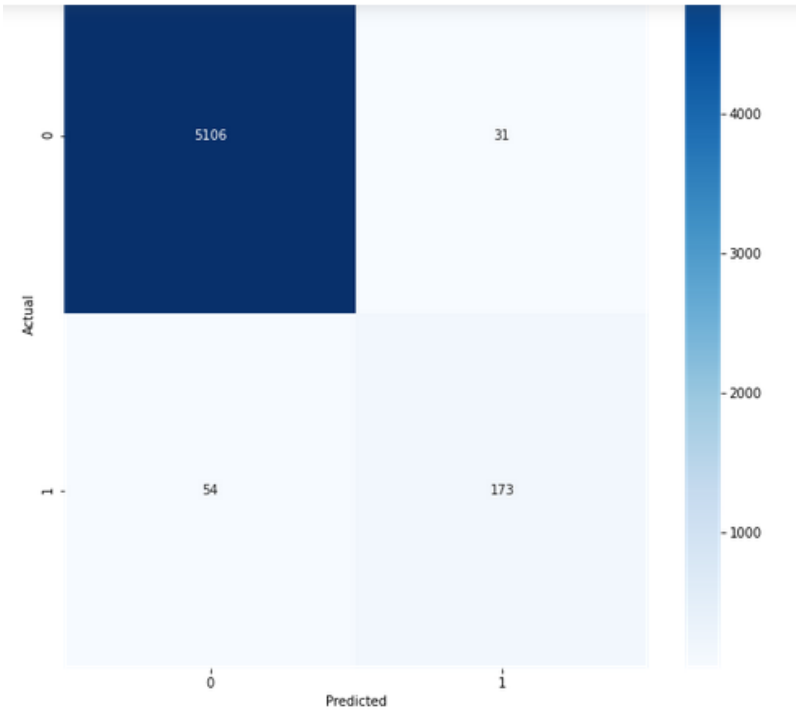
Falsos



Modelo Regresión Logística

Para efectuar el modelado de regresión logística utilizamos una función “search” que busca los mejores hiperparametros para el set de datos seleccionado.

La matriz de confusión del modelo realizado arroja los siguientes valores:

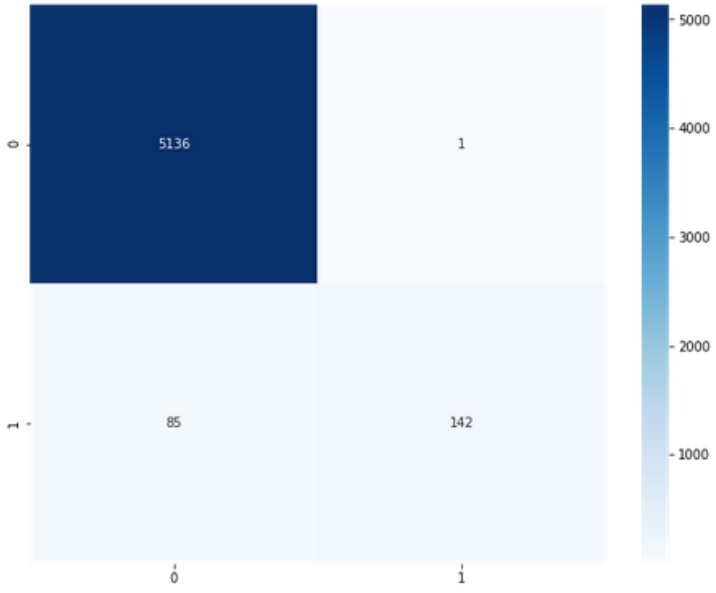


accuracy: 0.98

Modelo Random Forest

si bien existe una función de busca los mejores hiperparametros, utilizamos un set de parámetros predefinidos ya que dicha función tarda más de 24 hs en ejecutar.

La matriz de confusión del modelo realizado arroja los siguientes valores:



Accuracy: 0.98

Conclusión

Si bien ambos modelos tienen el mismo accuracy el modelo de Random Forest tiene mejor clasificación de requerimientos falsos que fueron determinados como positivos. En cambio el modelo de regresión logística

tiene casi la misma cantidad de diferencias pero más distribuidos entre entre positivos falsos y falsos positivos.

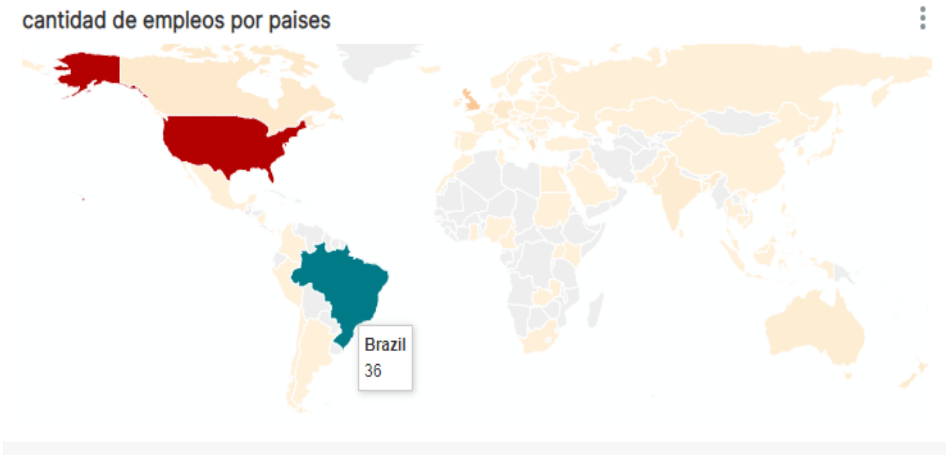
url notebook:

https://github.com/alvarov0907/tp_seminario/blob/master/jupyter/notebook/tp_ariel_alvaro/tp_ml_fake_job_postings.ipynb

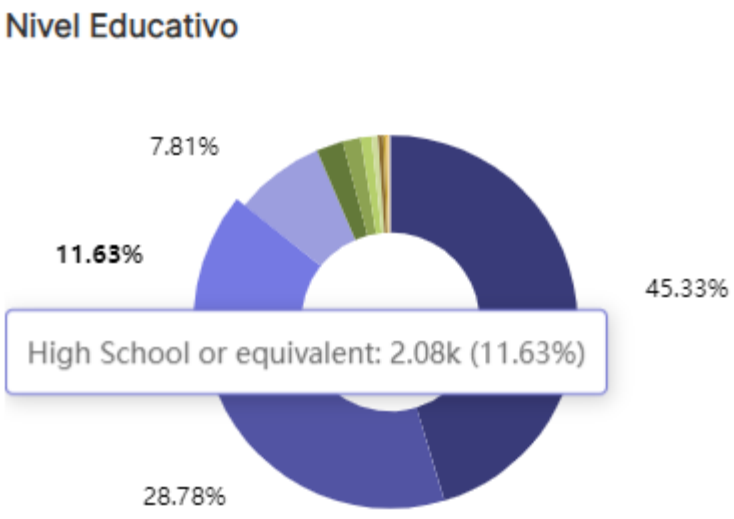
3. Visualizaciones en Superset

A partir del dataset guardado en la base de datos postrgres (Base de datos:“workshop” tabla: “jobs”) generamos un dashboard con los siguientes gráficos:

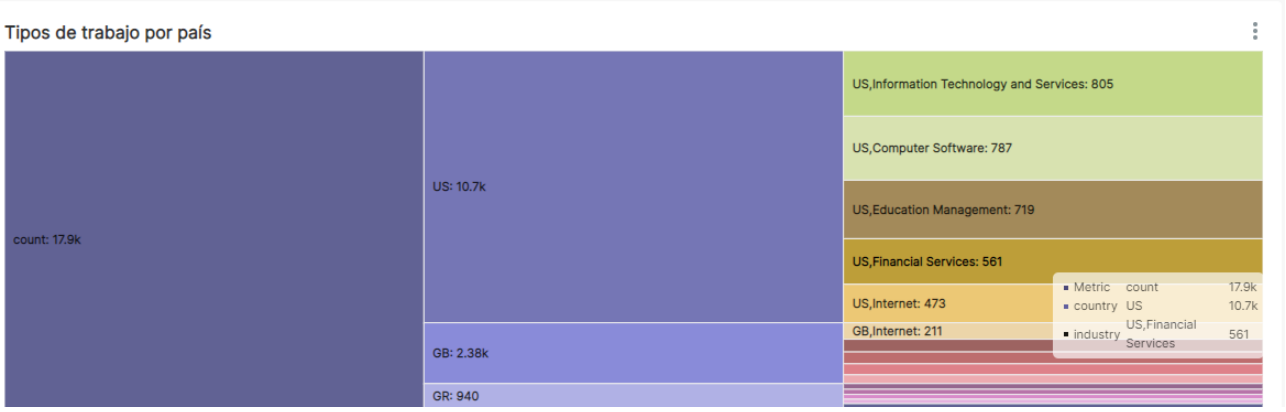
Cantidad de empleos por países



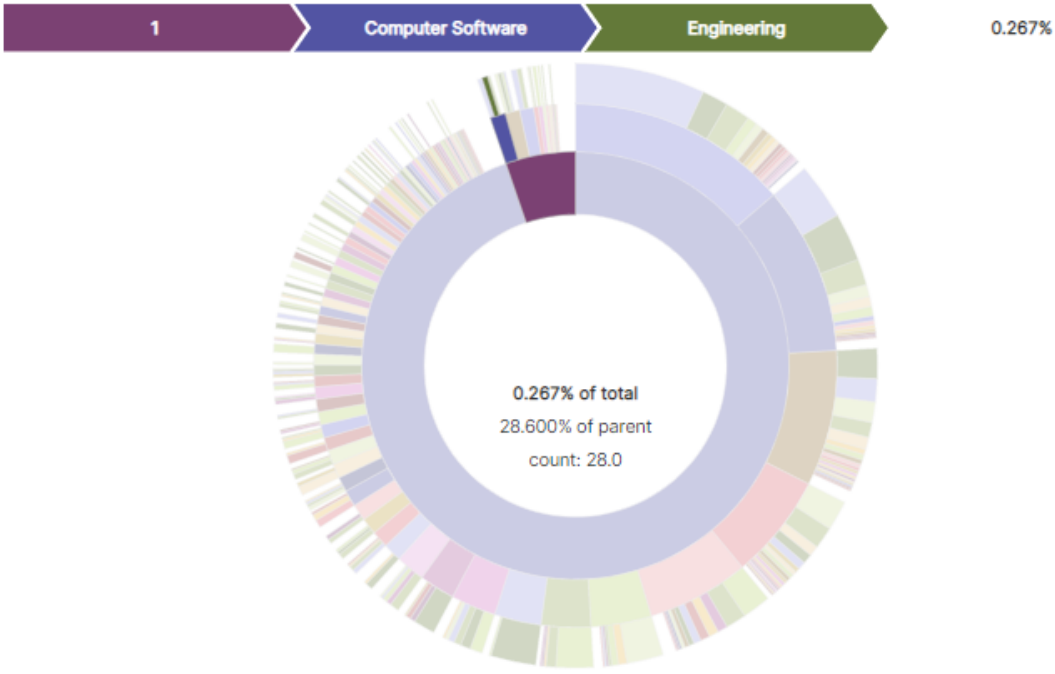
Empleos agrupados por niveles educativos requeridos



Solicitudes agrupadas por país y tipo de trabajo



Trabajos remotos y presenciales agrupados por industria y función



Airflow.

Se crearon tres dags en airflow :

params.py

```
#!/usr/bin/python3
from sqlalchemy import create_engine
from datetime import datetime, timedelta

SYMBOLS = [
   'BTCUSDT','ETHUSDT','LTCUSDT','LINKUSDT','DOTUSDT','BNBUSDT','SHIBUSDT','MANAUSDT',
   'SANDUSDT','XRPUSDT','GALAUSDT','SOLUSDT','TRXUSDT','LTCUSDT','FILUSDT','ADAUSDT',
   'DOGEUSDT','AVAXUSDT','LINKUSDT','SXPUSDT','PORTOUSDT','FTMUSDT','CHZUSDT','OMGUSDT',
   'IOXTXUSDT','LRCUSDT','WAPXUSDT','MATICUSDT','WINUSDT','ETCUSDT','CHRUSDT','EOSUSDT',
   'ALGOUSDT','NEARUSDT','MITHUSDT','SLPUSDT','ATOMUSDT','ICPUSDT'
   ]

user = "workshop"
password = "w0rkzh0p"
host = "postgres"
port = "5432"
database = "workshop"
schema = "workshop"
engine = create_engine(f"postgresql://{user}:{password}@{host}:{port}/{database}")

default_args = {
   "depends_on_past": False,
   "max_active_runs": 1,
   "owner": "TP_SEMINARIO",
   "retries": 3,
   "retry_delay": timedelta(minutes=1),
   "wait_for_downstream": False,
}
```

binance_market_dag_hourly.py

```
"""
TP SEMINARIO.
Alumnos: Alvaro Vanina y Ariel Belossi

Get all Market-Binance values and ingest in postgres.
---------------------------------

1) Create table if not exists binance_market

2) Get all values of Binance Markets Symbols.
```

```
3) Update new values each hour searching in binance_market and make the ingestion in
table in postgres.

P.D. We are taking some params from params.py like the keys and
    the connection to access postgress and the default_args for the dags.
---------------------------------
"""

#!/usr/bin/python3
from time import sleep
import requests
import urllib.parse
import json
import pandas as pd
import sys
import subprocess
import logging
from datetime import timedelta
from datetime import datetime
from pandas.io.json import json_normalize
from airflow.models import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator
from airflow.operators.postgres_operator import PostgresOperator
from pathlib import Path


from params import (
   SYMBOLS,
   schema,
   engine,
   default_args,
)

STORE_DIR = Path(__file__).resolve().parent / 'tmp-files' / 'random-num'
Path.mkdir(STORE_DIR, exist_ok=True, parents=True)

SQL_CREATE_TABLE_BINANCE_MARKET = f""" CREATE TABLE IF NOT EXISTS binance_market(
symbol TEXT,
priceChange TEXT,
priceChangePercent TEXT,
weightedAvgPrice TEXT,
prevClosePrice TEXT,
lastPrice TEXT,
lastQty TEXT,
bidPrice TEXT,
bidQty TEXT,
askPrice TEXT,
askQty TEXT,
```

```
openPrice TEXT,
highPrice TEXT,
lowPrice TEXT,
volume TEXT,
quoteVolume TEXT,
openTime INT,
closeTime INT,
firstId INT,
lastId  INT,
count INT
) ;
"""

def _ingestion_binance_market():
    r = requests.get('https://api.binance.com/api/v1/ticker/24hr')
    if r.status_code == 200:
        result = r.json()
        result = json_normalize(result)
        df = pd.DataFrame(result)
    #print(str(STORE_DIR)+str('/binance.csv'))
    df.to_csv(str(STORE_DIR)+str('/binance.csv'),index=False) #Escribo dataframe a csv en
la carpeta tmp-files/random-run
    with engine.begin():
        try:
            df.to_sql("binance_market", con=engine, schema=schema, if_exists="replace",
index=False)
        except Exception as e:
            pass
    logging.info(f"It was wrote to postgres binance_market succesfully.")

# dag definition
with DAG(
    dag_id="tp_seminario_binance_market_hourly",
    description="Get Binance Market hourly.",
    start_date=datetime(2021, 11, 16),
    schedule_interval="@hourly", # “At every minute.”
    catchup=False,
    default_args=default_args,
    tags=["Binance", "Market" , "hourly"],
    dagrun_timeout=timedelta(seconds=3600),
    doc_md=__doc__
) as dag:
    # tasks
    start_execution = DummyOperator(task_id="start_execution")
    end_execution = DummyOperator(task_id="end_execution")

    create_table_binance_market = PostgresOperator(
        task_id='create_table_binance_market',
        sql=SQL_CREATE_TABLE_BINANCE_MARKET,
        postgres_conn_id='my_postgres_connection_id',
```

```
    )

    ingestion_binance_market = PythonOperator(
        task_id="ingestion_binance_market",
        python_callable=_ingestion_binance_market,
    )


    start_execution >> create_table_binance_market >> ingestion_binance_market >>
end_execution
```

binance_scalping_dag_minute.py

```
"""
TP SEMINARIO.
Alumnos: Alvaro Vanina y Ariel Belossi

Get all Market-Binance values and ingest in postgres.
---------------------------------

1) Create table if not exists binance_market

2) Get all values of Binance Markets Symbols.

3) Update new values each hour searching in binance_market and make the ingestion in
table in postgres.

P.D. We are taking some params from params.py like the keys and
    the connection to access postgress and the default_args for the dags.
---------------------------------
"""

#!/usr/bin/python3
from time import sleep
import requests
import urllib.parse
import json
import pandas as pd
import sys
import subprocess
import logging
from datetime import timedelta
from datetime import datetime
from pandas.io.json import json_normalize
from airflow.models import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator
```

```
from airflow.operators.postgres_operator import PostgresOperator
from pathlib import Path


from params import (
   SYMBOLS,
   schema,
   engine,
   default_args,
)

STORE_DIR = Path(__file__).resolve().parent / 'tmp-files' / 'random-num'
Path.mkdir(STORE_DIR, exist_ok=True, parents=True)

SQL_CREATE_TABLE_BINANCE_MARKET = f""" CREATE TABLE IF NOT EXISTS binance_market(
symbol TEXT,
priceChange TEXT,
priceChangePercent TEXT,
weightedAvgPrice TEXT,
prevClosePrice TEXT,
lastPrice TEXT,
lastQty TEXT,
bidPrice TEXT,
bidQty TEXT,
askPrice TEXT,
askQty TEXT,
openPrice TEXT,
highPrice TEXT,
lowPrice TEXT,
volume TEXT,
quoteVolume TEXT,
openTime INT,
closeTime INT,
firstId INT,
lastId  INT,
count INT
) ;
"""

def _ingestion_binance_market():
   r = requests.get('https://api.binance.com/api/v1/ticker/24hr')
   if r.status_code == 200:
       result = r.json()
       result = json_normalize(result)
       df = pd.DataFrame(result)
   #print(str(STORE_DIR)+str('/binance.csv'))
   df.to_csv(str(STORE_DIR)+str('/binance.csv'),index=False) #Escribo dataframe a csv en
la carpeta tmp-files/random-run
   with engine.begin():
       try:
```

```
            df.to_sql("binance_market", con=engine, schema=schema, if_exists="replace",
index=False)
        except Exception as e:
            pass
    logging.info(f"It was wrote to postgres binance_market succesfully.")

# dag definition
with DAG(
    dag_id="tp_seminario_binance_market_hourly",
    description="Get Binance Market hourly.",
    start_date=datetime(2021, 11, 16),
    schedule_interval="@hourly", # “At every minute.”
    catchup=False,
    default_args=default_args,
    tags=["Binance", "Market" , "hourly"],
    dagrun_timeout=timedelta(seconds=3600),
    doc_md=__doc__
) as dag:
    # tasks
    start_execution = DummyOperator(task_id="start_execution")
    end_execution = DummyOperator(task_id="end_execution")

    create_table_binance_market = PostgresOperator(
        task_id='create_table_binance_market',
        sql=SQL_CREATE_TABLE_BINANCE_MARKET,
        postgres_conn_id='my_postgres_connection_id',
    )

    ingestion_binance_market = PythonOperator(
        task_id="ingestion_binance_market",
        python_callable=_ingestion_binance_market,
    )


    start_execution >> create_table_binance_market >> ingestion_binance_market >>
end_execution
```

Descripción de cada uno de los scripts.

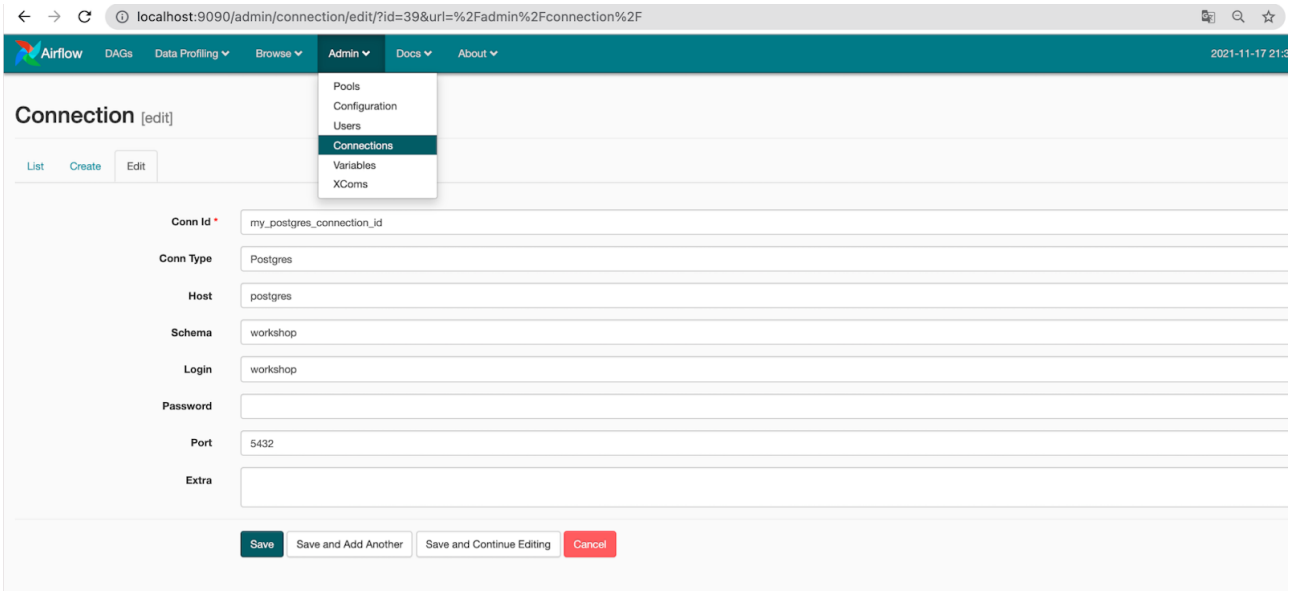
params.py

El script params.py guarda los parámetros a usar en los otros dos scripts.

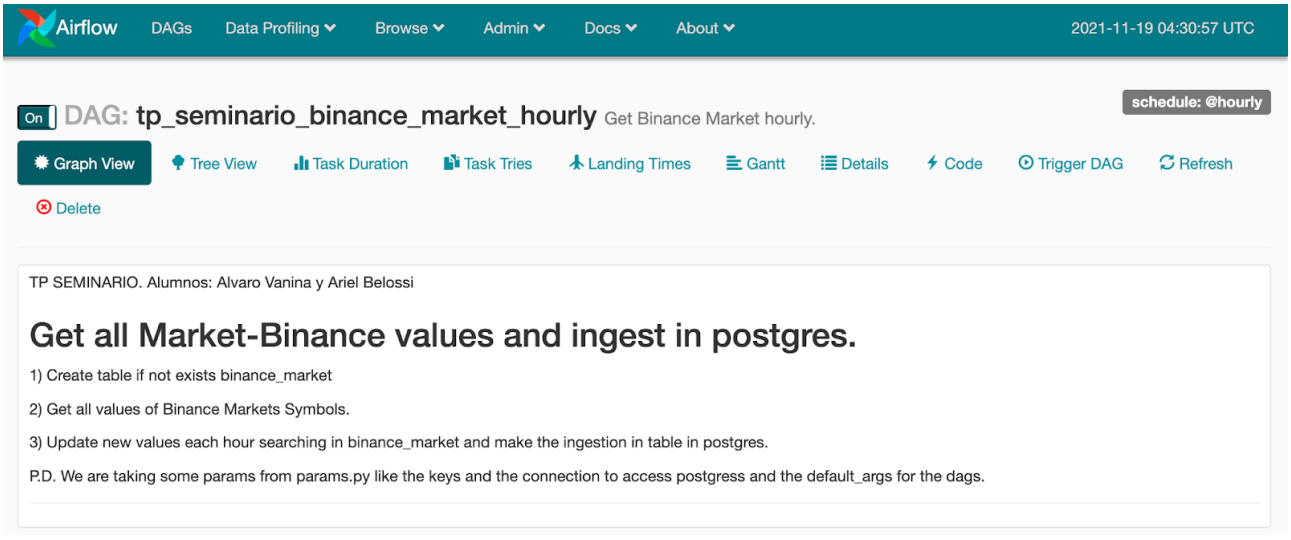
1. Por ejemplo, tiene una lista llamada SYMBOLS con los pares de criptomonedas más importantes en transacciones de volumen en las últimas 24 horas en sus pares USDT para el exchange Binance. Por ejemplo tenemos el par BTCUSDT, ETHUSDT, LTCUSDT y así sucesivamente.
2. Además están las credenciales para la conexión al postgres del docker via engine.
3. Además están los default_args que utilizaremos en los otros dos dags.

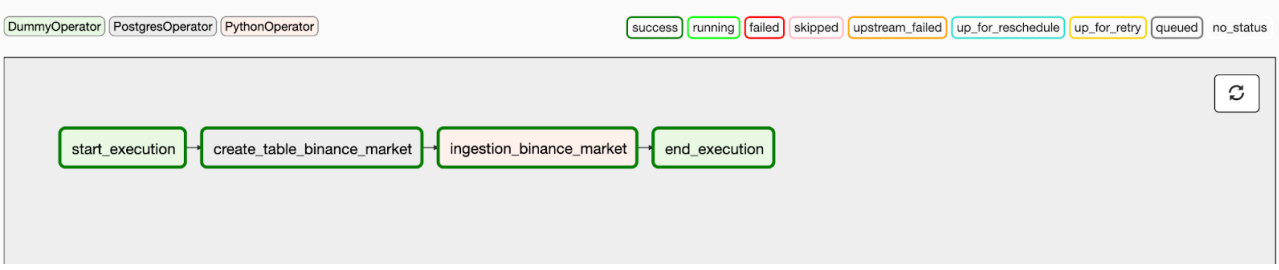
binance_market_dag_hourly.py

Este dag se ejecuta cada una hora y lo que hace es realizar un requests a la siguiente url https://api.binance.com/api/v1/ticker/24hr y así traerse todas los valores de monedas digitales del exchange Binance y lo guarda en un csv binance.csv en la carpeta tmp-files/random-num y además escribe todos estos valores en una tabla llamada binance_market en el schema workshop de la base de datos workshop que utilizamos en el taller. Para este dag fue necesario crear en Airflow una nueva conneción desde la web en admin --> Connections que la llamamos ***my_postgres_connection_id*** con las credenciales del postgres que usamos en el taller como muestra la siguiente imagen.



Además al comienzo de cada dag se escribió un descriptivo para que aparezca en el dag como muestra la siguiente imagen.





Para este dag usamos los operadores **DummyOperator** , **PostgresOperator** y **PythonOperator**

Se utiliza el operador **DummyOperator** para generar dos tareas llamadas start_execution y end_execution que son el comienzo y el fin de todas las tareas. Luego utiliza un operador **PostgresOperator** para crear la tabla binance_market en caso que no exista y luego utiliza el operado**r PythonOperator** que es quien realiza el requests , escribe en un csv y hace la ingesta en la tabla binance_market. Este Dag corre cada hora.

binance_scalping_dag_minute.py

Este dag tiene de interesante que consulta para los pares más importantes en volúmenes de transacciones de las últimas 24 hs en el exchange Binance y calcula para cada symbol (moneda en su par USDT) en temporalidad de 15 minutos:

- Su volumen de transacción actual en USDT
- La media móvil de los últimos 13 períodos
- La media móvil de los últimos 34 períodos
- Volumen promedio de los últimos 10 volúmenes.

Con estos datos utiliza una antiquísima regla de trading que dice:

Si la Ema 13 cruza hacia arriba a la Ema 34 ...

- Ema13 > Ema 34 y Volumen Actual > Volumen promedio → Señal Alcista = Comprar.

Si la Ema 13 cruza hacia abajo a la Ema 34 ...

- Ema13 < Ema 34 y Volumen Actual > Volumen promedio → Señal Bajista = Vender.

El script toma todos estos datos calculados y los va appendiando en una tabla llamada binance_scalping donde se escriben tres booleanos que son volumen_mayor_media , cruce_alcista, cruce_bajista que nos ayudarán en la decisión de comprar o vender una moneda de la siguiente manera.

Si el volumen_mayor_media = True y cruce_alcista = True entonces Comprar.
Si el volumen_mayor_media = True y cruce_bajista = True entonces Vender.

Aclaración: cruce_alcista cuando ema13 cruza hacia arriba a la ema34 y cruce_bajista cuando ema13 cruce hacia abajo a la ema34.

Observemos el siguiente ejemplo para el par GALA/USDT.

En la base de datos está escrito en UTC 0 y en la gráfica de Binance está en UTC -3 . Entre la hora 7.15 am a 7.30 am del 18/11/2021 se produce el cruce de la ema 13 por arriba de la ema 34 y el volumen en ese momento es mayor al volumen promedio de los últimos 10 volúmenes.

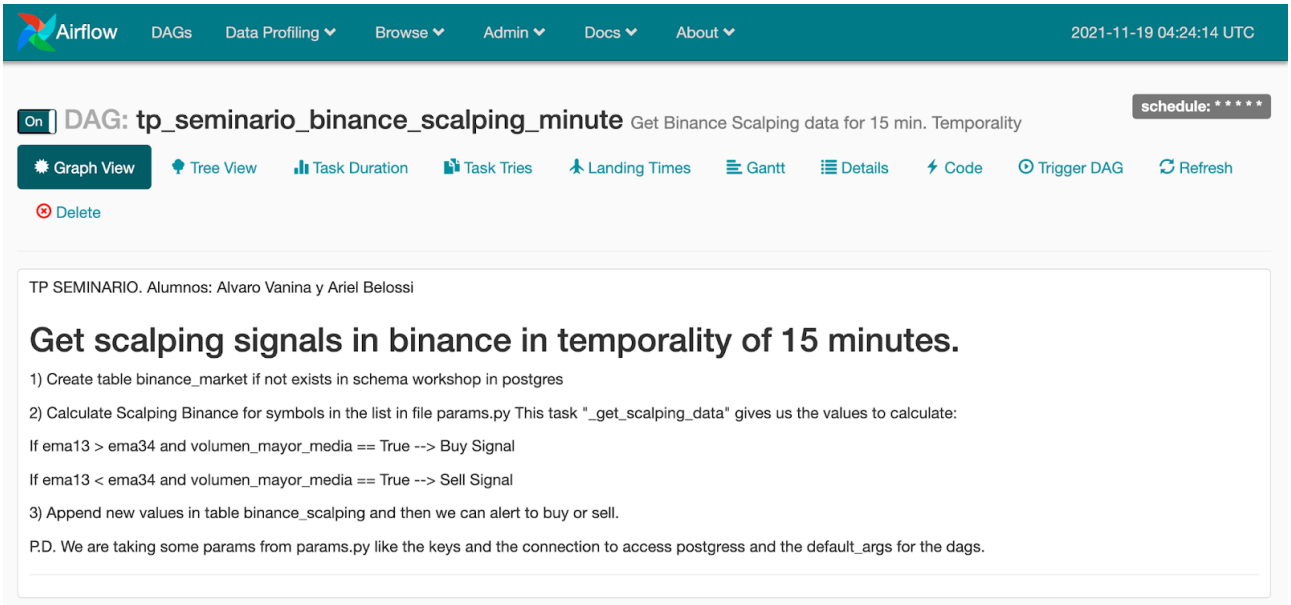
Dentro de ese intervalo de tiempo de 15 minutos el script (dag) detecta el cruce de la ema13 sobre la ema34 (cruce_alcista) con un volumen_mayor_media donde ambos valores booleanos son verdaderos, produciendo una señal de tipo alcista y como vemos en el gráfico de Binance posterior a ese cruce existió un aumento del valor de GALAUSDT en sus próximas 2 horas a dicho cruce hasta las 9.15 am.

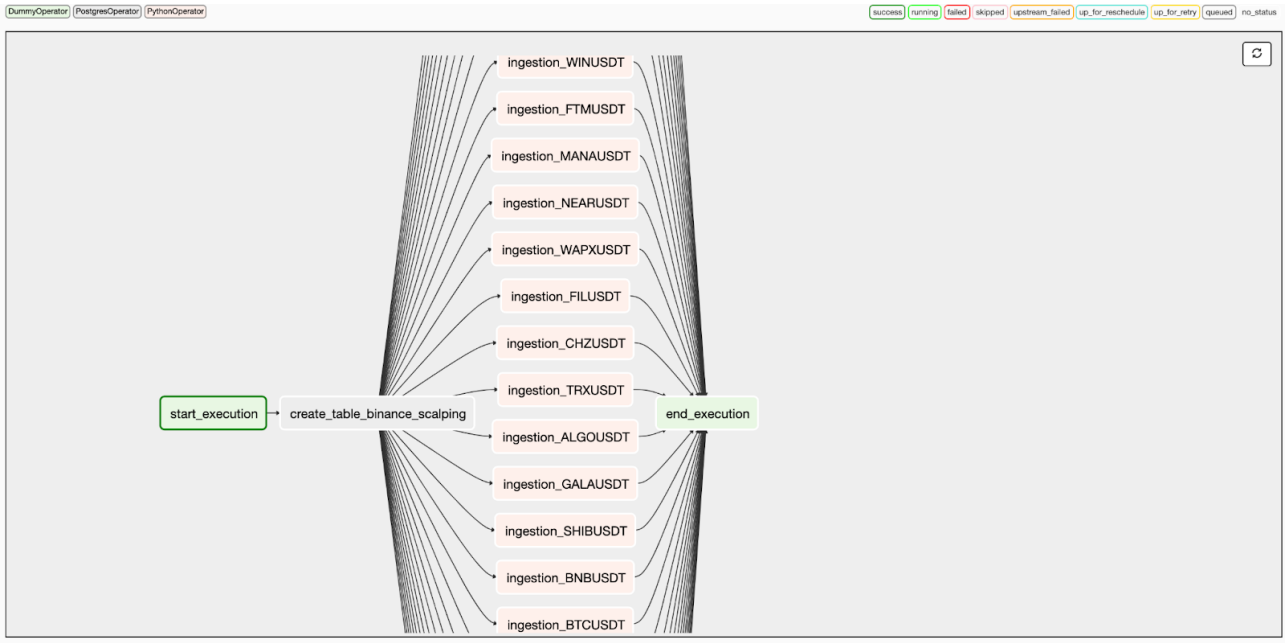
symbol	ema13	ema34	ema13_1	ema34_1	volumen	volumen_mayor_media	cruce_alcista	cruce_bajista	lastprice	datetime	execution_date_dag
GALAUSDT	0.217326782025474	0.217165195620657	0.215124579029719	0.216354601415849	52376599	t	t	f	0.23054	11/18/2021, 10:26:27	2021-11-18
GALAUSDT	0.217525353454045	0.217244624192086	0.215124579029719	0.216354601415849	61949942	t	t	f	0.23193	11/18/2021, 10:27:44	2021-11-18
GALAUSDT	0.217676782025474	0.217305195620657	0.215124579029719	0.216354601415849	70906233	t	t	f	0.23299	11/18/2021, 10:28:47	2021-11-18
GALAUSDT	0.217746782025474	0.217333195620657	0.215124579029719	0.216354601415849	77435307	t	t	f	0.23348	11/18/2021, 10:29:46	2021-11-18

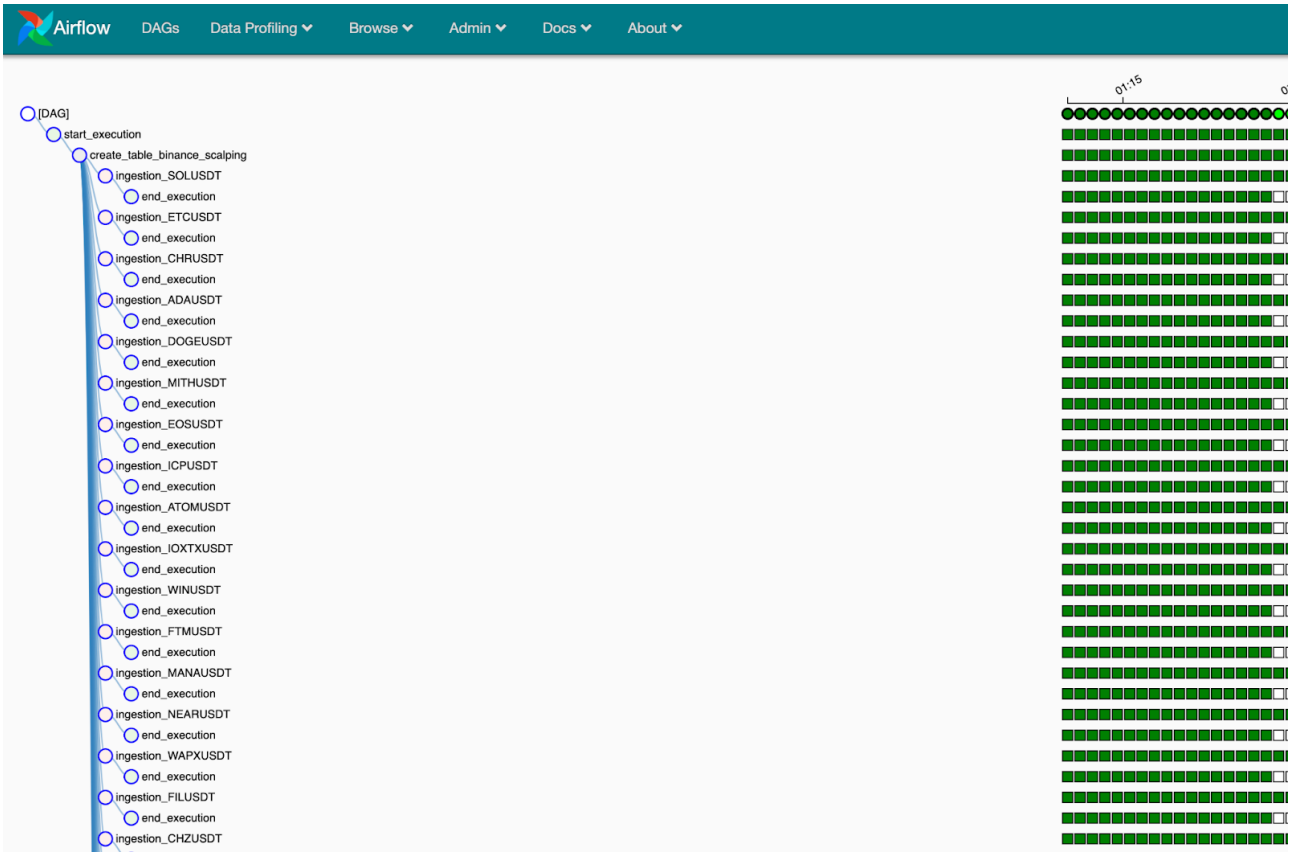
(4 rows)

Para la construcción de este dag utilizamos el operador **DummyOperator** para el inicio y fin de la ejecución. Luego se usó el operador **PostgresOperator** para la creación de la tabla binance_scalping en caso que no exista. Luego se utiliza el operador de **PythonOperator** que llama a la función que calcula todo lo referente a scalping. Este dag corre cada minuto.

Nota: Todos estos dags están disponibles en github en la carpeta dags dentro de airflow, en caso que necesiten verlos o correrlos dentro de su entorno.




DummyOperator
PostgresOperator
PythonOperator
success
running
failed
skipped
upstream_failed
up_for_reschedule
up_for_retry
queued
no_status
start_execution
create_table_binance_scalping
ingestion_WINUSDT
ingestion_FTMUSDT
ingestion_MANAUSDT
ingestion_NEARUSDT
ingestion_WAPXUSDT
ingestion_FILUSDT
ingestion_CHZUSDT
ingestion_TRXUSDT
ingestion_ALGOUSDT
ingestion_GALAUSDT
ingestion_SHIBUSDT
ingestion_BNBUSDT
ingestion_BTCUSDT
end_execution


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
01:15
[DAG]
start_execution
create_table_binance_scalping
ingestion_SOLUSDT
end_execution
ingestion_ETCUSDT
end_execution
ingestion_CHRUSDT
end_execution
ingestion_ADAUSDT
end_execution
ingestion_DOGEUSDT
end_execution
ingestion_MITHUSDT
end_execution
ingestion_EOSUSDT
end_execution
ingestion_ICPUSDT
end_execution
ingestion_ATOMUSDT
end_execution
ingestion_IOXTXUSDT
end_execution
ingestion_WINUSDT
end_execution
ingestion_FTMUSDT
end_execution
ingestion_MANAUSDT
end_execution
ingestion_NEARUSDT
end_execution
ingestion_WAPXUSDT
end_execution
ingestion_FILUSDT
end_execution
ingestion_CHZUSDT