Урок 8. Специфика применения ETL в различных предметных сферах

1. Используя материалы семинара и s8dag.py нужно доработать задачу в части записи данных в mysq по погоде яндекса и open weather (поля - метка текущего времени и температура).

2. Создать еще одну задачу по отправке данных в телеграм. За основу взять данные таблиц платежей из 4-го семинара (все 360 периодов), конвертировать их в текстовый формат и отправить их в telegram.

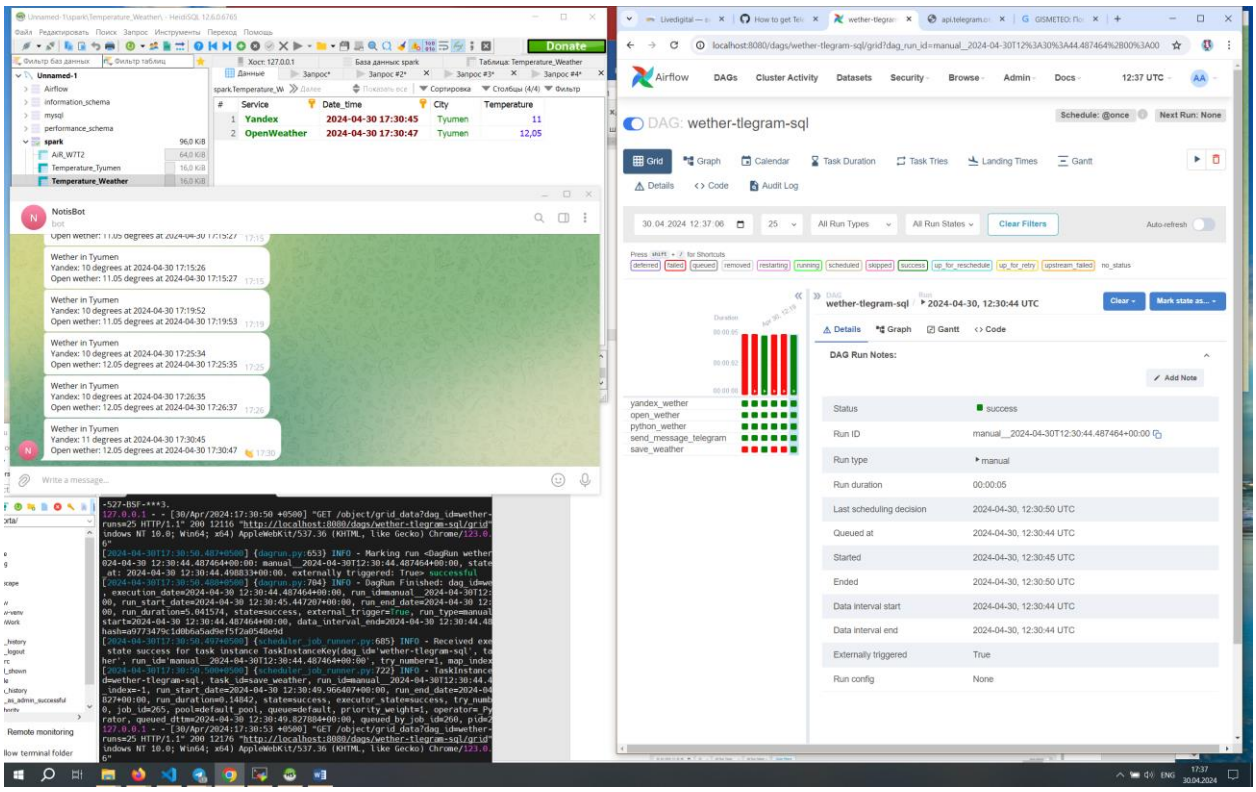
3. Рассмотрите возможность применения разметки html либо markdown. Нужно выслать одну основную таблицу. Есть есть лимит по сообщениям, можно ограничить количество строк таблицы. Можете использовать функцию limit в sql запросе.

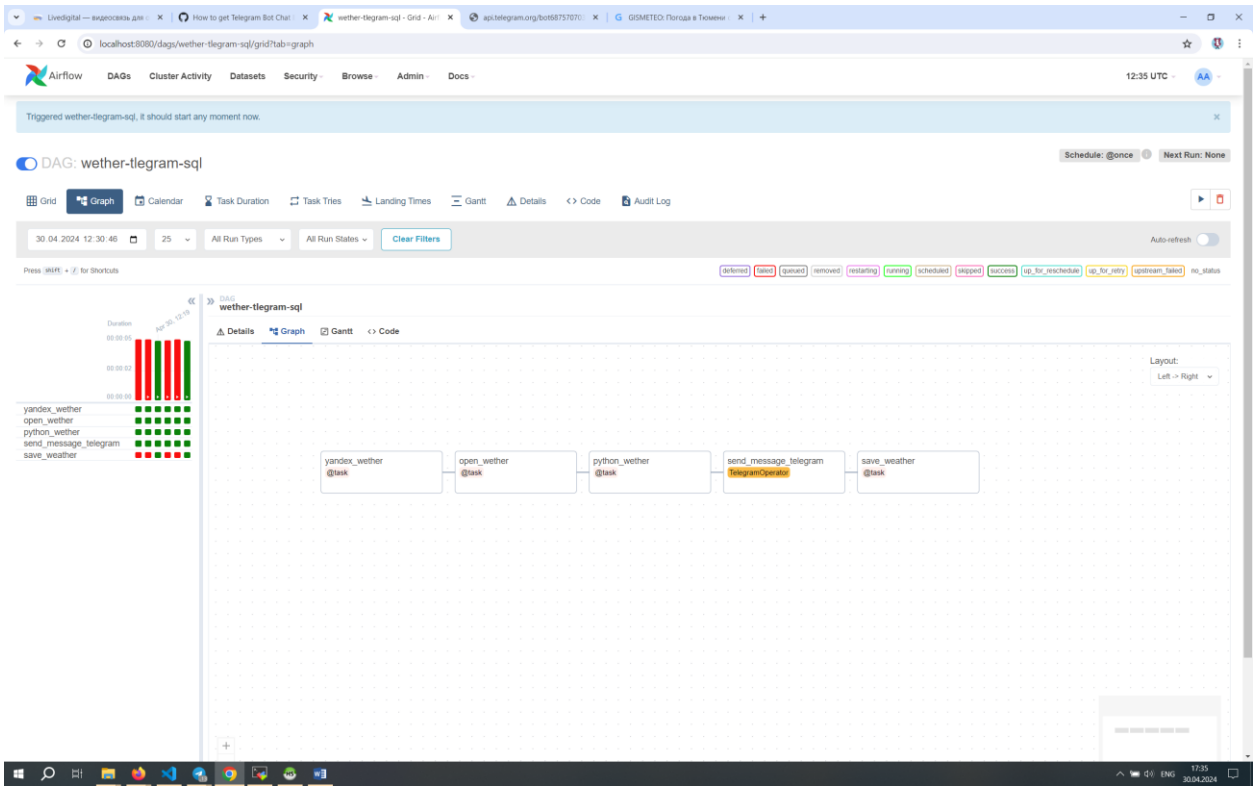
4. К ДЗ приложите код и скриншоты отработанных задач аирфлоу, а также отправленный слепкок из базы данных в вашем чаботе.

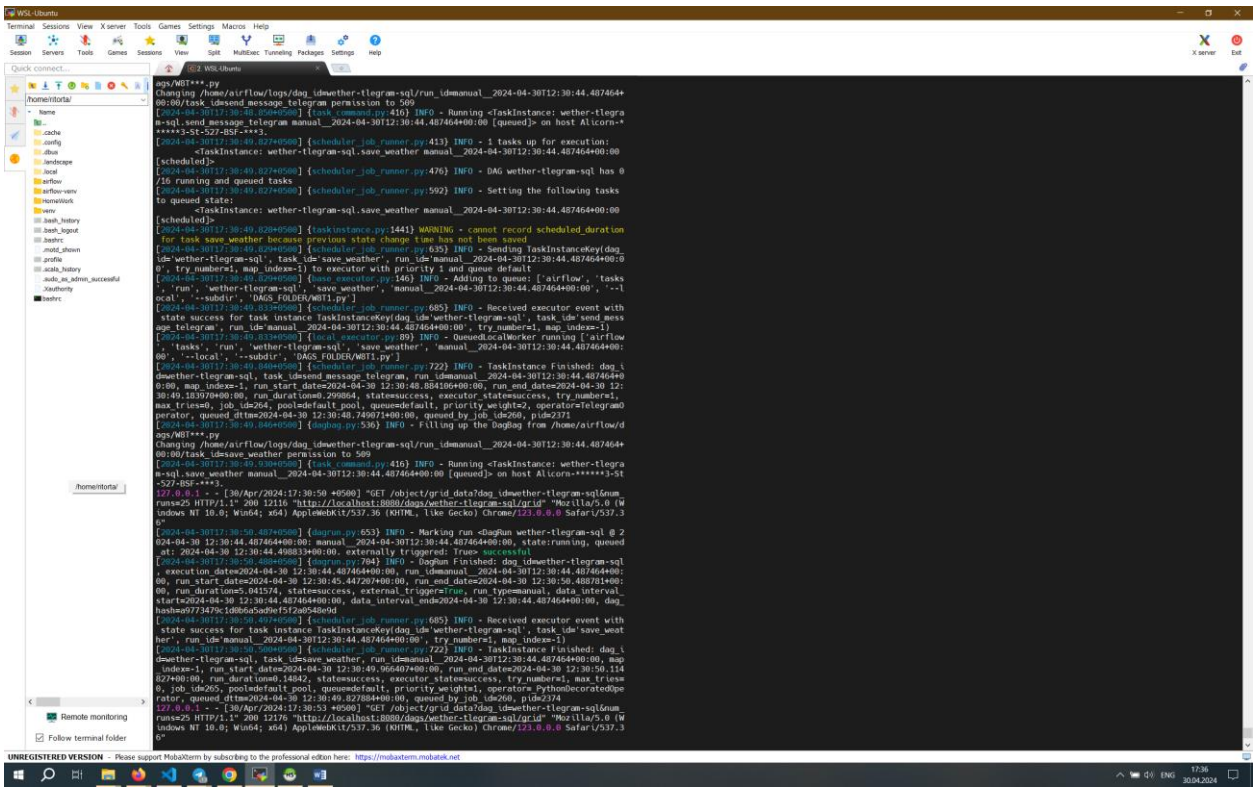
```
import datetime
import os
import requests
import pendulum
from airflow.decorators import dag, task
from airflow.providers.telegram.operators.telegram import TelegramOperator
from sqlalchemy import create_engine

os.environ["no_proxy"]="*"

@dag(
    dag_id="wether-tlegram-sql",
    schedule="@once",
    start_date=pendulum.datetime(2024, 4, 30, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
)


def WetherETL():

    send_message_telegram_task = TelegramOperator(
        task_id='send_message_telegram',
        telegram_conn_id='telegram_default',
        token='6875707033:AAG2TaDKrrLUwlUmcX9LIVA1uCm6S43pya0',
        chat_id='547504860',
        text='Wether in Tyumen \nYandex: ' + "{{
ti.xcom_pull(task_ids=['yandex_wether'],key='wether')[0]['temperature']}}" + "
degrees at " + "{{
ti.xcom_pull(task_ids=['yandex_wether'],key='wether')[0]['datetime']}}" +
    "\nOpen wether: " + "{{
ti.xcom_pull(task_ids=['open_wether'],key='open_wether')[0]['temperature']}}" + "
degrees at " + "{{
ti.xcom_pull(task_ids=['open_wether'],key='open_wether')[0]['datetime']}}",
```

```
    )


    @task(task_id='yandex_wether')
    def get_yandex_wether(**kwargs):
        ti = kwargs['ti']
        url =
"https://api.weather.yandex.ru/v2/informers/?lat=57.152985&lon=65.527168"

        payload={}
        headers = {
        'X-Yandex-API-Key': '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
        }
        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        temperature = response.json()['fact']['temp']
        current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        a = response.json()['fact']['temp']
        print(a)

        ti.xcom_push(key='wether', value={'temperature': temperature, 'datetime':
current_datetime})


    @task(task_id='open_wether')
    def get_open_wether(**kwargs):
        ti = kwargs['ti']
        url =
"https://api.openweathermap.org/data/2.5/weather?lat=57.152985&lon=65.527168&appi
d=2cd78e55c423fc81cebc1487134a6300"

        payload={}
        headers = {}

        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        temperature = round(float(response.json()['main']['temp']) - 273.15, 2)
        current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        a = round(float(response.json()['main']['temp']) - 273.15, 2)
        print(a)

        ti.xcom_push(key='open_wether', value={'temperature': temperature,
'datetime': current_datetime})


    @task(task_id='save_weather')
    def get_save_weather(**kwargs):
        yandex_data = kwargs['ti'].xcom_pull(task_ids='yandex_wether',
key='wether')
        open_weather_data = kwargs['ti'].xcom_pull(task_ids='open_wether',
key='open_wether')
```

```
        temperature_yandex = yandex_data['temperature']
        datetime_yandex = yandex_data['datetime']
        service_yandex = 'Yandex'

        temperature_open_weather = open_weather_data['temperature']
        datetime_open_weather = open_weather_data['datetime']
        service_open_weather = 'OpenWeather'

        engine = create_engine("mysql://root:1@localhost:33061/spark")

        with engine.connect() as connection:
            connection.execute("""DROP TABLE IF EXISTS
spark.`Temperature_Weather`""")
            connection.execute("""CREATE TABLE IF NOT EXISTS
spark.`Temperature_Weather` (
                Service VARCHAR(255),
                Date_time TIMESTAMP,
                City VARCHAR(255),
                Temperature FLOAT,
                PRIMARY KEY (Date_time, Service)
            )COLLATE='utf8mb4_general_ci' ENGINE=InnoDB""")
            connection.execute(f"""INSERT INTO spark.`Temperature_Weather`
(Date_time, City, Temperature, Service) VALUES ('{datetime_yandex}', 'Tyumen',
{temperature_yandex}, '{service_yandex}')""")
            connection.execute(f"""INSERT INTO spark.`Temperature_Weather`
(Date_time, City, Temperature, Service) VALUES ('{datetime_open_weather}',
'Tyumen', {temperature_open_weather}, '{service_open_weather}')""")


    @task(task_id='python_wether')
    def get_wether(**kwargs):
        print("Yandex
"+str(kwargs['ti'].xcom_pull(task_ids=['yandex_wether'],key='wether')[0])+" Open
"+str(kwargs['ti'].xcom_pull(task_ids=['open_wether'],key='open_wether')[0]))


    yandex_wether = get_yandex_wether()
    open_wether = get_open_wether()
    python_wether = get_wether()
    save_weather = get_save_weather()

    yandex_wether >> open_wether >> python_wether >> send_message_telegram_task
>> save_weather

dag = WetherETL()
```


Donate
NotisBot
Wether in Tyumen
Yandex: 10 degrees at 2024-04-30 17:15:26
Open wether: 11.05 degrees at 2024-04-30 17:15:27
Wether in Tyumen
Yandex: 10 degrees at 2024-04-30 17:19:52
Open wether: 11.05 degrees at 2024-04-30 17:19:53
Wether in Tyumen
Yandex: 10 degrees at 2024-04-30 17:25:34
Open wether: 12.05 degrees at 2024-04-30 17:25:35
Wether in Tyumen
Yandex: 10 degrees at 2024-04-30 17:26:35
Open wether: 12.05 degrees at 2024-04-30 17:26:37
Wether in Tyumen
Yandex: 11 degrees at 2024-04-30 17:30:45
Open wether: 12.05 degrees at 2024-04-30 17:30:47
Write a message...
Remote monitoring
DAG: wether-tlegram-sql
Schedule: @once
Next Run: None
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
Clear Filters
wether-tlegram-sql
2024-04-30, 12:30:44 UTC
Clear
Mark state as...
DAG Run Notes:
Add Note
Status
success
Run ID
manual__2024-04-30T12:30:44.487464+00:00
Run type
manual
Run duration
00:00:05
Last scheduling decision
2024-04-30, 12:30:50 UTC
Queued at
2024-04-30, 12:30:44 UTC
Started
2024-04-30, 12:30:45 UTC
Ended
2024-04-30, 12:30:50 UTC
Data interval start
2024-04-30, 12:30:44 UTC
Data interval end
2024-04-30, 12:30:44 UTC
Externally triggered
True
Run config
None
yandex_wether
open_wether
python_wether
send_message_telegram
save_weather


Triggered wether-tlegram-sql, it should start any moment now.
DAG: wether-tlegram-sql
Schedule: @once
Next Run: None
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Audit Log
Clear Filters
wether-tlegram-sql
Details
Graph
Gantt
Code
Layout:
Left -> Right
yandex_wether
@task
open_wether
@task
python_wether
@task
send_message_telegram
TelegramOperator
save_weather
@task



Второе задание разделено на несколько вариантов кода первый, здесь несмотря на что в Телеграмм выглядит таблица не отформатированной при копировании сообщения, таблица оказывается полностью ровной и это лишь проблема отображения телеграмма в виду маленьких окон сообщений, код:

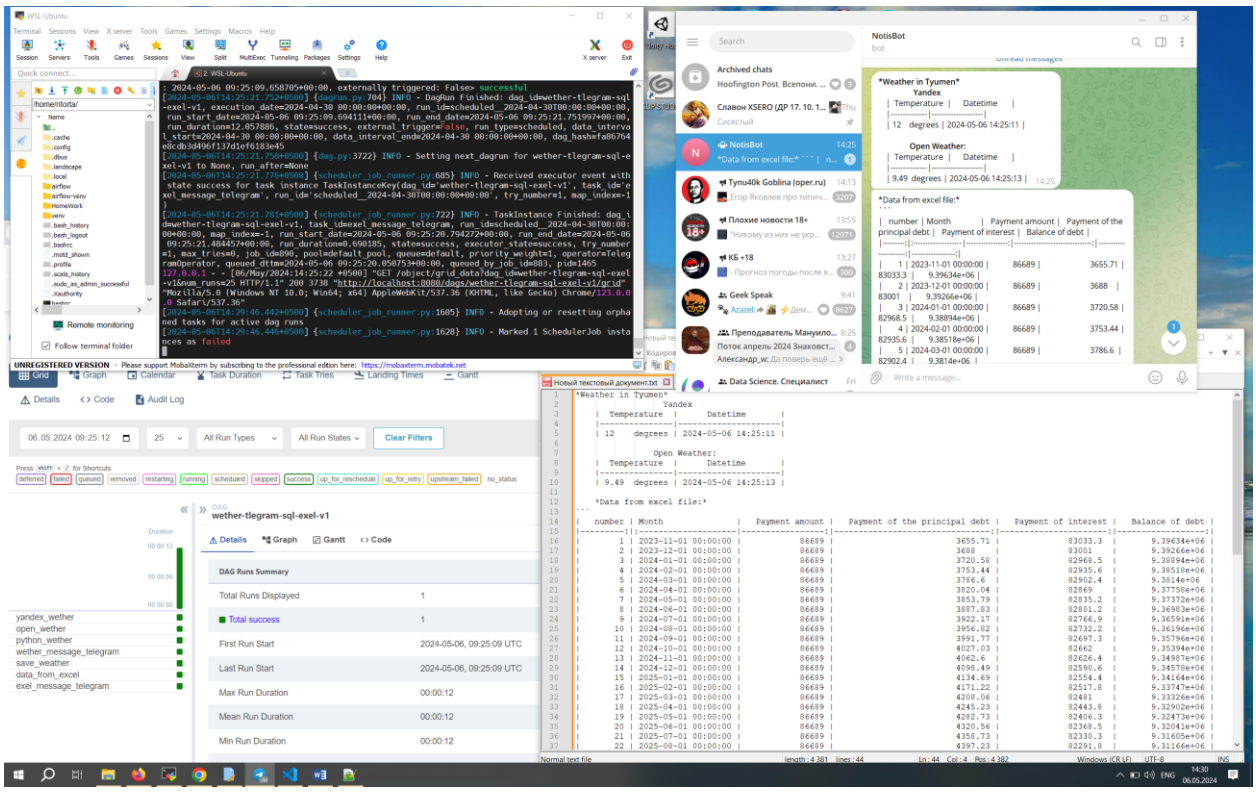
```
import datetime
import os
import requests
import pendulum
from airflow.decorators import dag, task
from airflow.providers.telegram.operators.telegram import TelegramOperator
from sqlalchemy import create_engine
import pandas as pd

os.environ["no_proxy"]="*"

@dag(
    dag_id="wether-tlegram-sql-exel-v1",
    schedule="@once",
    start_date=pendulum.datetime(2024, 4, 30, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
)


def WetherETL():

    first_message_telegram = TelegramOperator(
```

```
        task_id='wether_message_telegram',
        telegram_conn_id='telegram_default',
        token='6875707033:AAG2TaDKrrLUwlUmcX9LIVA1uCm6S43pya0',
        chat_id='547504860',
        text='''\
                  <b>*Weather in Tyumen*</b>
                  <b>Yandex</b>
    |  Temperature  |      Datetime       |
    |---------------|---------------------|
    | {{ '{:<5}'.format(ti.xcom_pull(task_ids=["yandex_wether"],
key="wether")[0]["temperature"]) }} degrees | {{
ti.xcom_pull(task_ids=["yandex_wether"], key="wether")[0]["datetime"] }} |

                <b>Open Weather:</b>
    |  Temperature  |      Datetime       |
    |---------------|---------------------|
    | {{ '{:<5}'.format(ti.xcom_pull(task_ids=["open_wether"],
key="open_wether")[0]["temperature"]) }} degrees | {{
ti.xcom_pull(task_ids=["open_wether"], key="open_wether")[0]["datetime"] }} |
    '''
    )


    second_message_telegram = TelegramOperator(
        task_id='exel_message_telegram',
        telegram_conn_id='telegram_default',
        token='6875707033:AAG2TaDKrrLUwlUmcX9LIVA1uCm6S43pya0',
        chat_id='547504860',
        text='*Data from excel file:*\n```\n{{
task_instance.xcom_pull(task_ids="data_from_excel") }}\n```',
    )


    @task(task_id='yandex_wether')
    def get_yandex_wether(**kwargs):
        ti = kwargs['ti']
        url =
"https://api.weather.yandex.ru/v2/informers/?lat=57.152985&lon=65.527168"

        payload={}
        headers = {
        'X-Yandex-API-Key': '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
        }
        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        temperature = response.json()['fact']['temp']
        current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        a = response.json()['fact']['temp']
        print(a)
```

```
        ti.xcom_push(key='wether', value={'temperature': temperature, 'datetime':
current_datetime})


    @task(task_id='open_wether')
    def get_open_wether(**kwargs):
        ti = kwargs['ti']
        url =
"https://api.openweathermap.org/data/2.5/weather?lat=57.152985&lon=65.527168&appi
d=2cd78e55c423fc81cebc1487134a6300"

        payload={}
        headers = {}

        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        temperature = round(float(response.json()['main']['temp']) - 273.15, 2)
        current_datetime = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        a = round(float(response.json()['main']['temp']) - 273.15, 2)
        print(a)

        ti.xcom_push(key='open_wether', value={'temperature': temperature,
'datetime': current_datetime})



    @task(task_id='save_weather')
    def get_save_weather(**kwargs):
        yandex_data = kwargs['ti'].xcom_pull(task_ids='yandex_wether',
key='wether')
        open_weather_data = kwargs['ti'].xcom_pull(task_ids='open_wether',
key='open_wether')

        temperature_yandex = yandex_data['temperature']
        datetime_yandex = yandex_data['datetime']
        service_yandex = 'Yandex'

        temperature_open_weather = open_weather_data['temperature']
        datetime_open_weather = open_weather_data['datetime']
        service_open_weather = 'OpenWeather'

        engine = create_engine("mysql://root:1@localhost:33061/spark")

        with engine.connect() as connection:
            connection.execute("""DROP TABLE IF EXISTS
spark.`Temperature_Weather`""")
            connection.execute("""CREATE TABLE IF NOT EXISTS
spark.`Temperature_Weather` (
                Service VARCHAR(255),
                Date_time TIMESTAMP,
                City VARCHAR(255),
```

```
                Temperature FLOAT,
                PRIMARY KEY (Date_time, Service)
            )COLLATE='utf8mb4_general_ci' ENGINE=InnoDB""")
            connection.execute(f"""INSERT INTO spark.`Temperature_Weather`
(Date_time, City, Temperature, Service) VALUES ('{datetime_yandex}', 'Tyumen',
{temperature_yandex}, '{service_yandex}')""")
            connection.execute(f"""INSERT INTO spark.`Temperature_Weather`
(Date_time, City, Temperature, Service) VALUES ('{datetime_open_weather}',
'Tyumen', {temperature_open_weather}, '{service_open_weather}')""")


    @task(task_id='python_wether')
    def get_wether(**kwargs):
        print("Yandex
"+str(kwargs['ti'].xcom_pull(task_ids=['yandex_wether'],key='wether')[0])+" Open
"+str(kwargs['ti'].xcom_pull(task_ids=['open_wether'],key='open_wether')[0]))


    def truncate_message(message, max_length=3959):
        if len(message) > max_length:
            return message[:max_length]
        return message


    @task(task_id='data_from_excel')
    def get_data_from_excel():
        excel_file_path = "/home/ritorta/HomeWork/W8/TeL_W8T2.xlsx"
        df = pd.read_excel(excel_file_path)
        table_text = df.to_markdown(index=False)
        truncated_text = truncate_message(table_text)

        return truncated_text

    yandex_wether = get_yandex_wether()
    open_wether = get_open_wether()
    python_wether = get_wether()
    save_weather = get_save_weather()
    data_from_excel = get_data_from_excel()

    yandex_wether >> open_wether >> python_wether >> first_message_telegram >>
save_weather >> data_from_excel >> second_message_telegram

dag = WetherETL()
```

Ещё раз повторюсь, если скопировать сообщения из телеграмма в блокнот или документ word/google при сохранении форматирования а не просто вставки как текст, текст будет выглядеть как отформатированная таблица.



Вариант кода два загружает из баз данных SQL

```
import datetime
import os
import requests
import pendulum
from airflow.decorators import dag, task
from airflow.providers.telegram.operators.telegram import TelegramOperator
from sqlalchemy import create_engine
import pandas as pd
from tabulate import tabulate

os.environ["no_proxy"]="*"

@dag(
    dag_id="wether-tlegram-sql-exel-V2",
    schedule="@once",
    start_date=pendulum.datetime(2024, 4, 30, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
)


def WetherETL():

    first_message_telegram = TelegramOperator(
        task_id='weather_sql_message_telegram',
        telegram_conn_id='telegram_default',
```

```
        token='6875707033:AAG2TaDKrrLUwlUmcX9LIVA1uCm6S43pya0',
        chat_id='547504860',
        text='Wether in Tyumen \n\n' + "<pre>{{
ti.xcom_pull(task_ids=['get_sql_save_weather'],key='table_wether')[0]}}</pre>"
    )

    second_message_telegram = TelegramOperator(
        task_id='exel_sql_message_telegram',
        telegram_conn_id='telegram_default',
        token='6875707033:AAG2TaDKrrLUwlUmcX9LIVA1uCm6S43pya0',
        chat_id='547504860',
        text="<pre>{{
ti.xcom_pull(task_ids=['exel_get_sql'],key='table_payments')[0]}}</pre>"
    )


    @task(task_id='yandex_wether')
    def get_yandex_wether(**kwargs):
        ti = kwargs['ti']
        url =
"https://api.weather.yandex.ru/v2/informers/?lat=57.152985&lon=65.527168"

        payload={}
        headers = {
        'X-Yandex-API-Key': '33f45b91-bcd4-46e4-adc2-33cfdbbdd88e'
        }
        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        temperature = response.json()['fact']['temp']

        a = response.json()['fact']['temp']
        print(a)

        ti.xcom_push(key='wether', value=temperature)


    @task(task_id='open_wether')
    def get_open_wether(**kwargs):
        ti = kwargs['ti']
        url =
"https://api.openweathermap.org/data/2.5/weather?lat=57.152985&lon=65.527168&appi
d=2cd78e55c423fc81cebc1487134a6300"

        payload={}
        headers = {}

        response = requests.request("GET", url, headers=headers, data=payload)
        print("test")
        temperature = round(float(response.json()['main']['temp']) - 273.15, 2)

        a = round(float(response.json()['main']['temp']) - 273.15, 2)
```

```
        print(a)

        ti.xcom_push(key='open_wether', value=temperature)


    @task(task_id='get_sql_save_weather')
    def save_weather_get_sql(**kwargs):
        ti = kwargs['ti']
        yandex_data = str(kwargs['ti'].xcom_pull(task_ids=['yandex_wether'],
key='wether')[0])
        open_weather_data =
str(kwargs['ti'].xcom_pull(task_ids=['open_wether'],key='open_wether')[0])
        engine = create_engine("mysql://root:1@localhost:33061/spark")
        city='Tyumen'

        df = pd.DataFrame  ({
            'City': [city],
            'Yandex': [yandex_data],
            'Open Weather': [open_weather_data],
            'date_time': [pd.Timestamp.now().round(freq='s')]
        })

        df.to_sql(name='test_8_3', con=engine, schema='spark',
if_exists='append', index=False)
        ti.xcom_push(key='table_wether', value=tabulate(df, headers='keys',
tablefmt='psql'))


    @task(task_id='python_wether')
    def get_wether(**kwargs):
        print("Yandex
"+str(kwargs['ti'].xcom_pull(task_ids=['yandex_wether'],key='wether')[0])+" Open
"+str(kwargs['ti'].xcom_pull(task_ids=['open_wether'],key='open_wether')[0]))


    @task(task_id='exel_get_sql')
    def get_sql_exel(**kwargs):
        ti = kwargs['ti']
        engine = create_engine("mysql://root:1@localhost:33061/spark")

        df = pd.read_sql(sql='AiR_W7T2', con=engine)
        df['Month'] = df['Month'].dt.date
        df.drop('Balance of debt', inplace=True, axis=1)
        ti.xcom_push(key='table_payments', value=tabulate(df.head(20),
headers='keys', tablefmt='psql'))


    yandex_wether = get_yandex_wether()
    open_wether = get_open_wether()
    python_wether = get_wether()
    save_weather = save_weather_get_sql()
```

```
    sql_exel = get_sql_exel()

    yandex_wether >> open_wether >> python_wether >> save_weather >>
first_message_telegram >> sql_exel >> second_message_telegram

dag = WetherETL()
```