ALLEANDRA ASYRAFIAN NURSANI – MID AI ENGINEER

Analytical database: Clickhouse (Scale plan, trial)

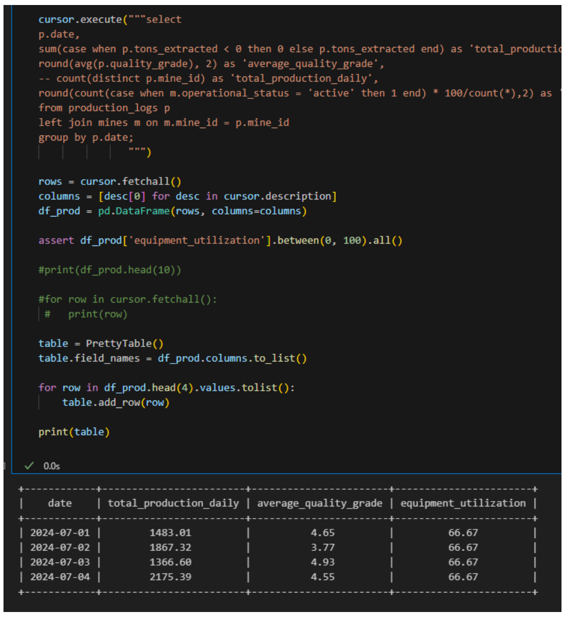
Editor: Visual Studio Code, MySQL Workbench

Language: MySQL, python(requests, pandas, mysql-connector-python, prettytables, loggings, matplotlib)

1. DATA PIPELINE

a. Extraction

- Retrieve daily production data from the production_logs

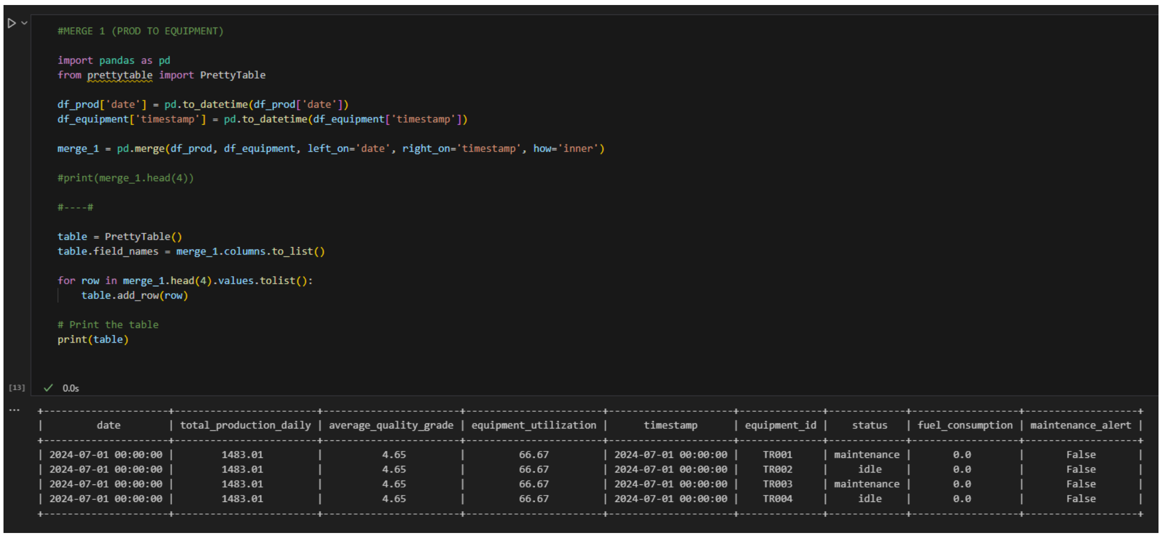


```
cursor.execute("""select
p.date,
sum(case when p.tons_extracted < 0 then 0 else p.tons_extracted end) as 'total_productic
round(avg(p.quality_grade), 2) as 'average_quality_grade',
-- count(distinct p.mine_id) as 'total_production_daily',
round(count(case when m.operational_status = 'active' then 1 end) * 100/count(*),2) as '
from production_logs p
left join mines m on m.mine_id = p.mine_id
group by p.date;
                """)

rows = cursor.fetchall()
columns = [desc[0] for desc in cursor.description]
df_prod = pd.DataFrame(rows, columns=columns)

assert df_prod['equipment_utilization'].between(0, 100).all()

#print(df_prod.head(10))

#for row in cursor.fetchall():
#   print(row)

table = PrettyTable()
table.field_names = df_prod.columns.to_list()

for row in df_prod.head(4).values.tolist():
    table.add_row(row)

print(table)
```

✓ 0.0s

```
+------------+------------------------+-----------------------+-----------------------+
|    date    | total_production_daily | average_quality_grade | equipment_utilization |
+------------+------------------------+-----------------------+-----------------------+
| 2024-07-01 |        1483.01         |          4.65         |         66.67         |
| 2024-07-02 |        1867.32         |          3.77         |         66.67         |
| 2024-07-03 |        1366.60         |          4.93         |         66.67         |
| 2024-07-04 |        2175.39         |          4.55         |         66.67         |
+------------+------------------------+-----------------------+-----------------------+
```

- Read the equipment_sensors.csv

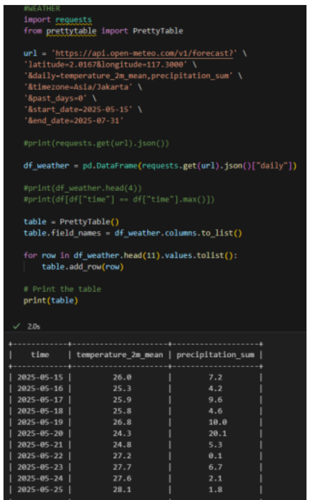


```
#MERGE 1 (PROD TO EQUIPMENT)

import pandas as pd
from prettytable import PrettyTable

df_prod['date'] = pd.to_datetime(df_prod['date'])
df_equipment['timestamp'] = pd.to_datetime(df_equipment['timestamp'])

merge_1 = pd.merge(df_prod, df_equipment, left_on='date', right_on='timestamp', how='inner')

#print(merge_1.head(4))

#----#

table = PrettyTable()
table.field_names = merge_1.columns.to_list()

for row in merge_1.head(4).values.tolist():
    table.add_row(row)

# Print the table
print(table)
```

[13] ✓ 0.0s

```
+---------------------+------------------------+-----------------------+-----------------------+---------------------+--------------+-------------+------------------+-------------------+
|         date        | total_production_daily | average_quality_grade | equipment_utilization |      timestamp      | equipment_id |    status   | fuel_consumption | maintenance_alert |
+---------------------+------------------------+-----------------------+-----------------------+---------------------+--------------+-------------+------------------+-------------------+
| 2024-07-01 00:00:00 |        1483.01         |          4.65         |         66.67         | 2024-07-01 00:00:00 |    TR001     | maintenance |       0.0        |       False       |
| 2024-07-01 00:00:00 |        1483.01         |          4.65         |         66.67         | 2024-07-01 00:00:00 |    TR002     |     idle    |       0.0        |       False       |
| 2024-07-01 00:00:00 |        1483.01         |          4.65         |         66.67         | 2024-07-01 00:00:00 |    TR003     | maintenance |       0.0        |       False       |
| 2024-07-01 00:00:00 |        1483.01         |          4.65         |         66.67         | 2024-07-01 00:00:00 |    TR004     |     idle    |       0.0        |       False       |
+---------------------+------------------------+-----------------------+-----------------------+---------------------+--------------+-------------+------------------+-------------------+
```

- Call weather API



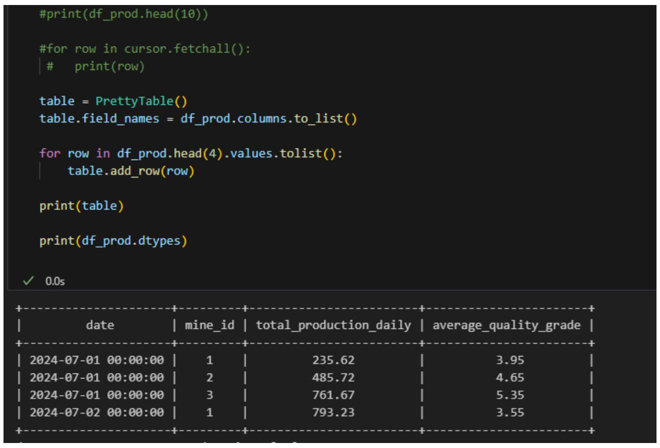
```
#WEATHER
import requests
from prettytable import PrettyTable

url = 'https://api.open-meteo.com/v1/forecast?' \
'latitude=2.0167&longitude=117.3000' \
'&daily=temperature_2m_mean,precipitation_sum' \
'&timezone=Asia/Jakarta' \
'&past_days=0' \
'&start_date=2025-05-15' \
'&end_date=2025-07-31'

#print(requests.get(url).json())

df_weather = pd.DataFrame(requests.get(url).json()["daily"])

#print(df_weather.head(4))
#print(df[df["time"] == df["time"].max()])

table = PrettyTable()
table.field_names = df_weather.columns.to_list()

for row in df_weather.head(11).values.tolist():
    table.add_row(row)

# Print the table
print(table)
```

✓ 2.0s

```
+------------+--------------------+-------------------+
|    time    | temperature_2m_mean | precipitation_sum |
+------------+--------------------+-------------------+
| 2025-05-15 |        26.0        |        7.2        |
| 2025-05-16 |        25.3        |        4.2        |
| 2025-05-17 |        25.9        |        9.6        |
| 2025-05-18 |        25.8        |        4.6        |
| 2025-05-19 |        26.8        |        10.0       |
| 2025-05-20 |        24.3        |        20.1       |
| 2025-05-21 |        24.8        |        5.3        |
| 2025-05-22 |        27.2        |        0.1        |
| 2025-05-23 |        27.7        |        6.7        |
| 2025-05-24 |        27.6        |        2.1        |
| 2025-05-25 |        28.1        |        1.8        |
```

b. Transformation

- Total_production_daily
- Average_quality_grade

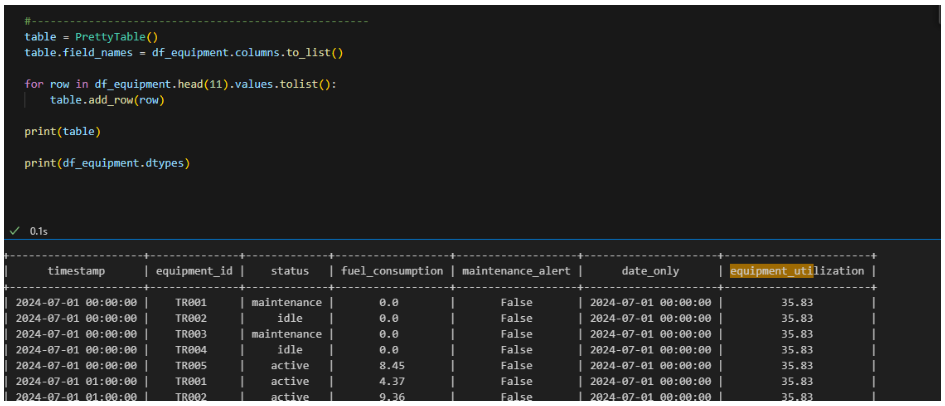


```
#print(df_prod.head(10))

#for row in cursor.fetchall():
#    print(row)

table = PrettyTable()
table.field_names = df_prod.columns.to_list()

for row in df_prod.head(4).values.tolist():
    table.add_row(row)

print(table)

print(df_prod.dtypes)
```

✓ 0.0s

```
+---------------------+---------+---------------------+----------------------+
|         date        | mine_id | total_production_daily | average_quality_grade |
+---------------------+---------+---------------------+----------------------+
| 2024-07-01 00:00:00 |    1    |        235.62       |         3.95         |
| 2024-07-01 00:00:00 |    2    |        485.72       |         4.65         |
| 2024-07-01 00:00:00 |    3    |        761.67       |         5.35         |
| 2024-07-02 00:00:00 |    1    |        793.23       |         3.55         |
+---------------------+---------+---------------------+----------------------+
```

- equipment_utilization:

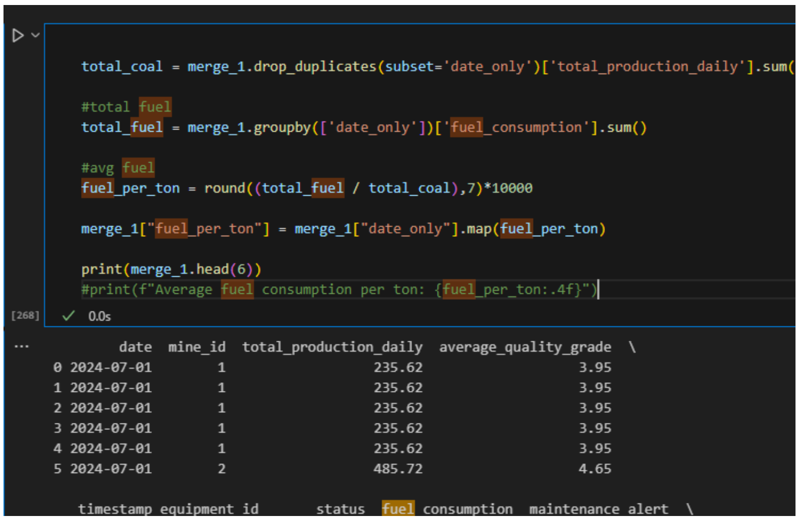


```
#--------------------------------------------------
table = PrettyTable()
table.field_names = df_equipment.columns.to_list()

for row in df_equipment.head(11).values.tolist():
    table.add_row(row)

print(table)

print(df_equipment.dtypes)
```

✓ 0.1s

```
+---------------------+--------------+-------------+------------------+-------------------+---------------------+-----------------------+
|      timestamp      | equipment_id |    status   | fuel_consumption | maintenance_alert |      date_only      | equipment_utilization |
+---------------------+--------------+-------------+------------------+-------------------+---------------------+-----------------------+
| 2024-07-01 00:00:00 |    TR001     | maintenance |       0.0        |       False       | 2024-07-01 00:00:00 |         35.83         |
| 2024-07-01 00:00:00 |    TR002     |     idle    |       0.0        |       False       | 2024-07-01 00:00:00 |         35.83         |
| 2024-07-01 00:00:00 |    TR003     | maintenance |       0.0        |       False       | 2024-07-01 00:00:00 |         35.83         |
| 2024-07-01 00:00:00 |    TR004     |     idle    |       0.0        |       False       | 2024-07-01 00:00:00 |         35.83         |
| 2024-07-01 00:00:00 |    TR005     |    active   |       8.45       |       False       | 2024-07-01 00:00:00 |         35.83         |
| 2024-07-01 01:00:00 |    TR001     |    active   |       4.37       |       False       | 2024-07-01 00:00:00 |         35.83         |
| 2024-07-01 01:00:00 |    TR002     |    active   |       9.36       |       False       | 2024-07-01 00:00:00 |         35.83         |
```

- fuel_efficiency

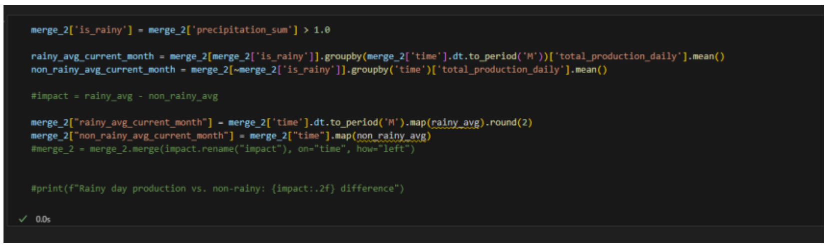


```
total_coal = merge_1.drop_duplicates(subset='date_only')['total_production_daily'].sum(

#total fuel
total_fuel = merge_1.groupby(['date_only'])['fuel_consumption'].sum()

#avg fuel
fuel_per_ton = round((total_fuel / total_coal),7)*10000

merge_1["fuel_per_ton"] = merge_1["date_only"].map(fuel_per_ton)

print(merge_1.head(6))
#print(f"Average fuel consumption per ton: {fuel_per_ton:.4f}")
```

[268] ✓ 0.0s

```
...          date  mine_id  total_production_daily  average_quality_grade  \
    0 2024-07-01        1                  235.62                   3.95
    1 2024-07-01        1                  235.62                   3.95
    2 2024-07-01        1                  235.62                   3.95
    3 2024-07-01        1                  235.62                   3.95
    4 2024-07-01        1                  235.62                   3.95
    5 2024-07-01        2                  485.72                   4.65

          timestamp equipment_id        status  fuel_consumption  maintenance_alert  \
```

- weather_impact

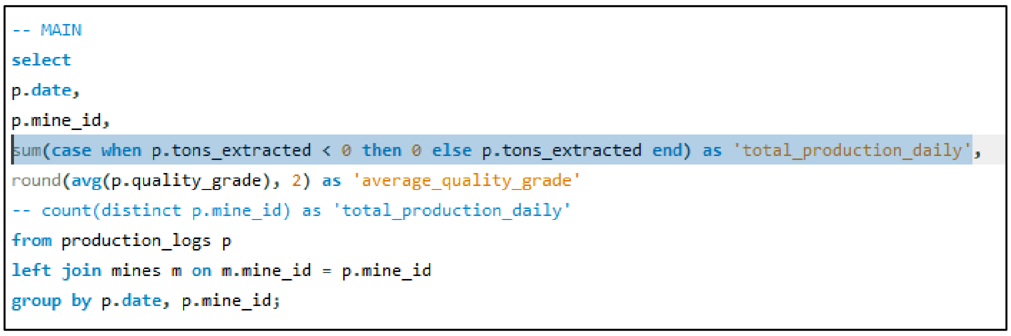


```
merge_2['is_rainy'] = merge_2['precipitation_sum'] > 1.0

rainy_avg_current_month = merge_2[merge_2['is_rainy']].groupby(merge_2['time'].dt.to_period('M'))['total_production_daily'].mean()
non_rainy_avg_current_month = merge_2[~merge_2['is_rainy']].groupby('time')['total_production_daily'].mean()

#impact = rainy_avg - non_rainy_avg

merge_2["rainy_avg_current_month"] = merge_2['time'].dt.to_period('M').map(rainy_avg).round(2)
merge_2["non_rainy_avg_current_month"] = merge_2["time"].map(non_rainy_avg)
#merge_2 = merge_2.merge(impact.rename("impact"), on="time", how="left")


#print(f"Rainy day production vs. non-rainy: {impact:.2f} difference")
```

✓ 0.0s

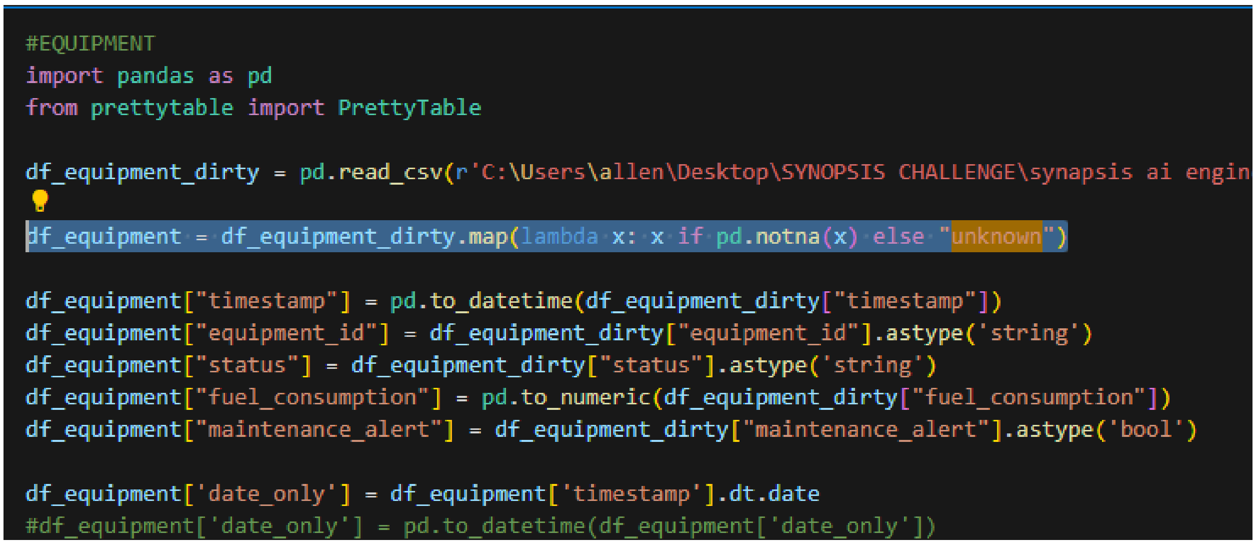
c. **handling**

- **avoid tons_extracted values that are negative**



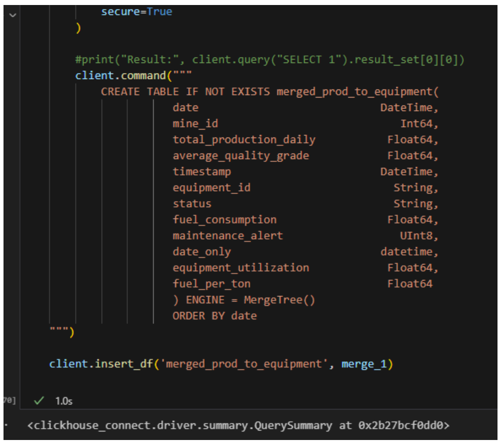
```sql
-- MAIN
select
p.date,
p.mine_id,
sum(case when p.tons_extracted < 0 then 0 else p.tons_extracted end) as 'total_production_daily',
round(avg(p.quality_grade), 2) as 'average_quality_grade'
-- count(distinct p.mine_id) as 'total_production_daily'
from production_logs p
left join mines m on m.mine_id = p.mine_id
group by p.date, p.mine_id;
```

- **unknown flagging for missing sensor data**



```python
#EQUIPMENT
import pandas as pd
from prettytable import PrettyTable

df_equipment_dirty = pd.read_csv(r'C:\Users\allen\Desktop\SYNOPSIS CHALLENGE\synapsis ai engin

df_equipment = df_equipment_dirty.map(lambda x: x if pd.notna(x) else "unknown")

df_equipment["timestamp"] = pd.to_datetime(df_equipment_dirty["timestamp"])
df_equipment["equipment_id"] = df_equipment_dirty["equipment_id"].astype('string')
df_equipment["status"] = df_equipment_dirty["status"].astype('string')
df_equipment["fuel_consumption"] = pd.to_numeric(df_equipment_dirty["fuel_consumption"])
df_equipment["maintenance_alert"] = df_equipment_dirty["maintenance_alert"].astype('bool')

df_equipment['date_only'] = df_equipment['timestamp'].dt.date
#df_equipment['date_only'] = pd.to_datetime(df_equipment['date_only'])
```

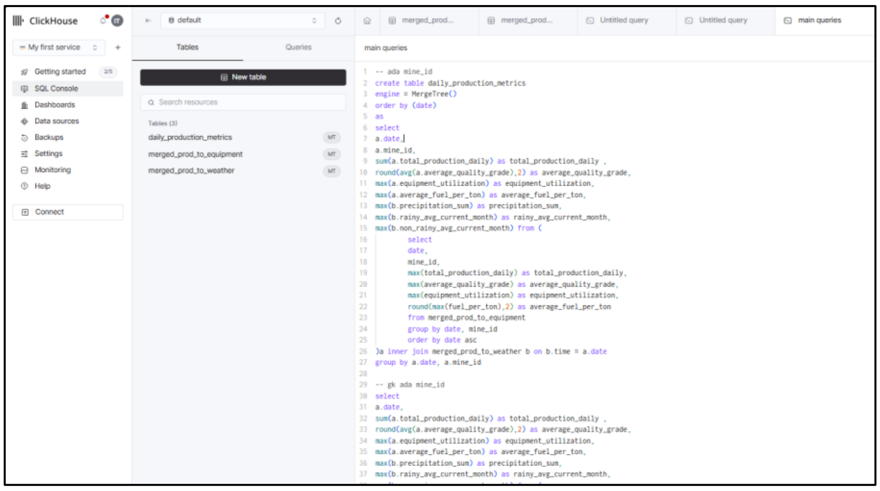
2. IMPLEMENT ETL SCRIPT



```python
        secure=True
    )

    #print("Result:", client.query("SELECT 1").result_set[0][0])
    client.command("""
        CREATE TABLE IF NOT EXISTS merged_prod_to_equipment(
                    date                                DateTime,
                    mine_id                                Int64,
                    total_production_daily               Float64,
                    average_quality_grade                Float64,
                    timestamp                           DateTime,
                    equipment_id                          String,
                    status                                String,
                    fuel_consumption                     Float64,
                    maintenance_alert                      UInt8,
                    date_only                           datetime,
                    equipment_utilization                Float64,
                    fuel_per_ton                         Float64
                    ) ENGINE = MergeTree()
                    ORDER BY date
""")

client.insert_df('merged_prod_to_equipment', merge_1)
```

```
✓ 1.0s
<clickhouse_connect.driver.summary.QuerySummary at 0x2b27bcf0dd0>
```

Python



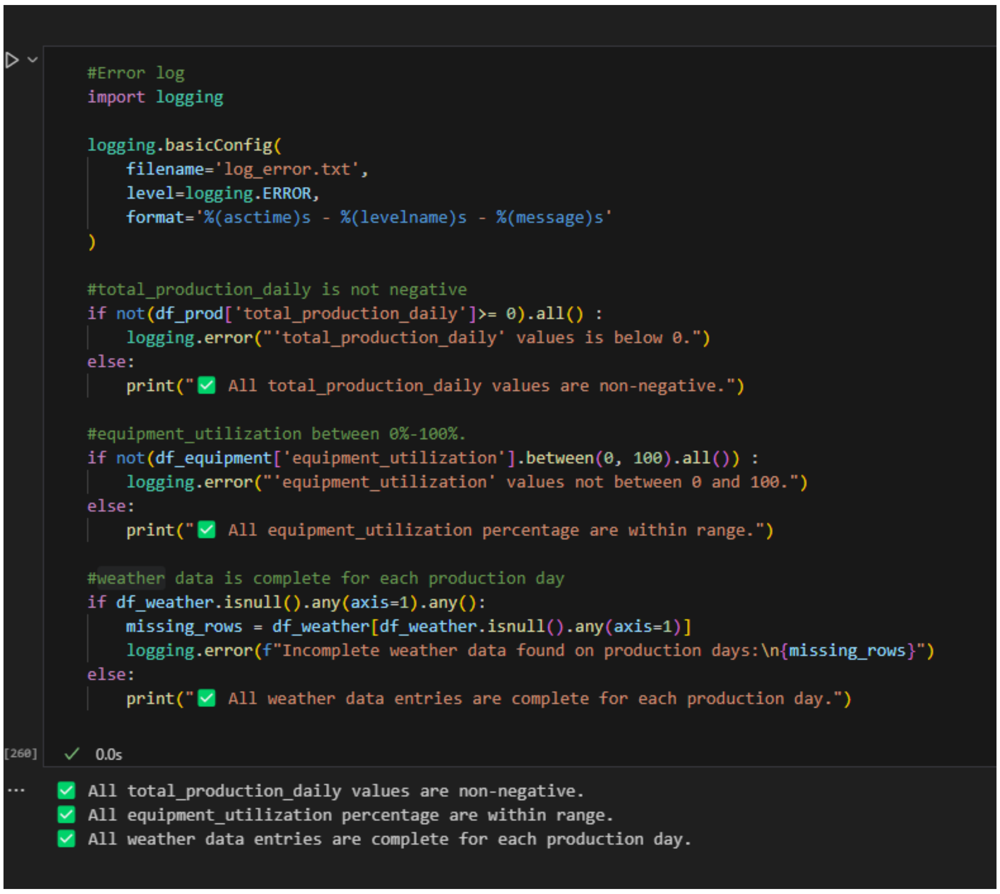
```sql
-- ada mine_id
create table daily_production_metrics
engine = MergeTree()
order by (date)
as
select
a.date,
a.mine_id,
sum(a.total_production_daily) as total_production_daily ,
round(avg(a.average_quality_grade),2) as average_quality_grade,
max(a.equipment_utilization) as equipment_utilization,
max(a.average_fuel_per_ton) as average_fuel_per_ton,
max(b.precipitation_sum) as precipitation_sum,
max(b.rainy_avg_current_month) as rainy_avg_current_month,
max(b.non_rainy_avg_current_month) from (
        select
        date,
        mine_id,
        max(total_production_daily) as total_production_daily,
        max(average_quality_grade) as average_quality_grade,
        max(equipment_utilization) as equipment_utilization,
        round(max(fuel_per_ton),2) as average_fuel_per_ton
        from merged_prod_to_equipment
        group by date, mine_id
        order by date asc
)a inner join merged_prod_to_weather b on b.time = a.date
group by a.date, a.mine_id

-- gk ada mine_id
select
a.date,
sum(a.total_production_daily) as total_production_daily ,
round(avg(a.average_quality_grade),2) as average_quality_grade,
max(a.equipment_utilization) as equipment_utilization,
max(a.average_fuel_per_ton) as average_fuel_per_ton,
max(b.precipitation_sum) as precipitation_sum,
max(b.rainy_avg_current_month) as rainy_avg_current_month,
```

Clickhouse

3. VALIDATE DATA

a. Implement checks

- Total_production_daily non-negative
- equipment_utilization is between 0 and 100%.
- Confirm weather data is complete for each production day.

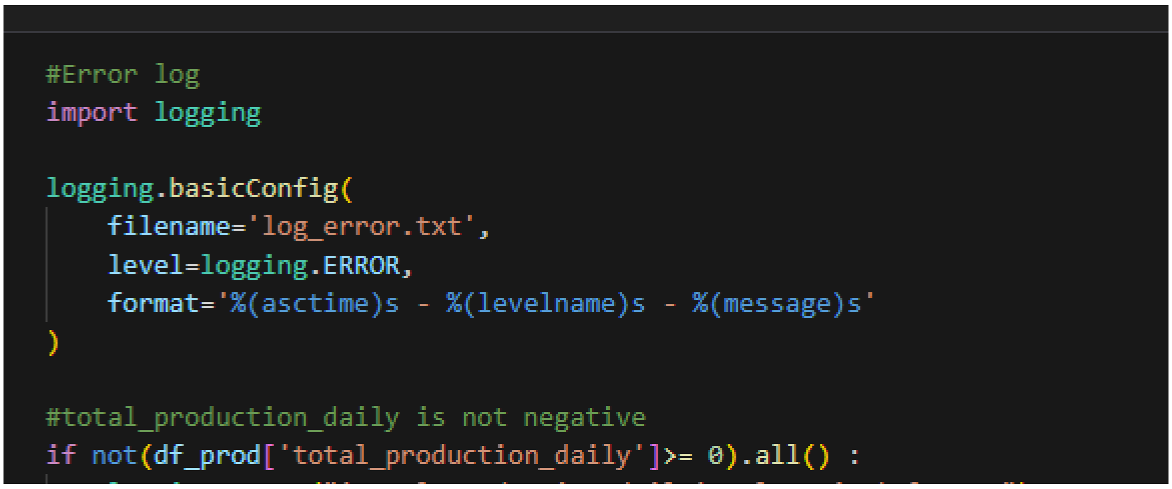


```python
#Error log
import logging

logging.basicConfig(
    filename='log_error.txt',
    level=logging.ERROR,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

#total_production_daily is not negative
if not(df_prod['total_production_daily']>= 0).all() :
    logging.error("'total_production_daily' values is below 0.")
else:
    print("✅ All total_production_daily values are non-negative.")

#equipment_utilization between 0%-100%.
if not(df_equipment['equipment_utilization'].between(0, 100).all()) :
    logging.error("'equipment_utilization' values not between 0 and 100.")
else:
    print("✅ All equipment_utilization percentage are within range.")

#weather data is complete for each production day
if df_weather.isnull().any(axis=1).any():
    missing_rows = df_weather[df_weather.isnull().any(axis=1)]
    logging.error(f"Incomplete weather data found on production days:\n{missing_rows}")
else:
    print("✅ All weather data entries are complete for each production day.")
```

```
[260]  ✓ 0.0s
```

```
✅ All total_production_daily values are non-negative.
✅ All equipment_utilization percentage are within range.
✅ All weather data entries are complete for each production day.
```

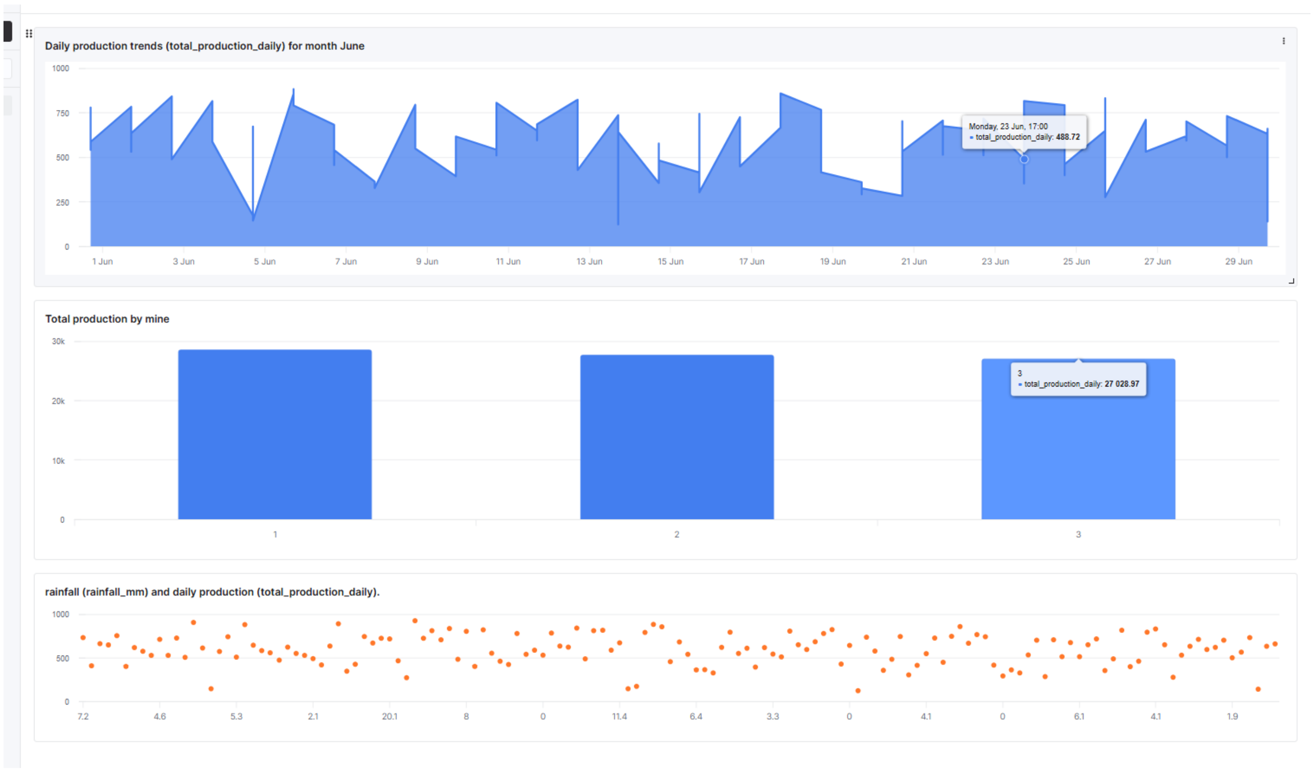
b. Handle anomalies



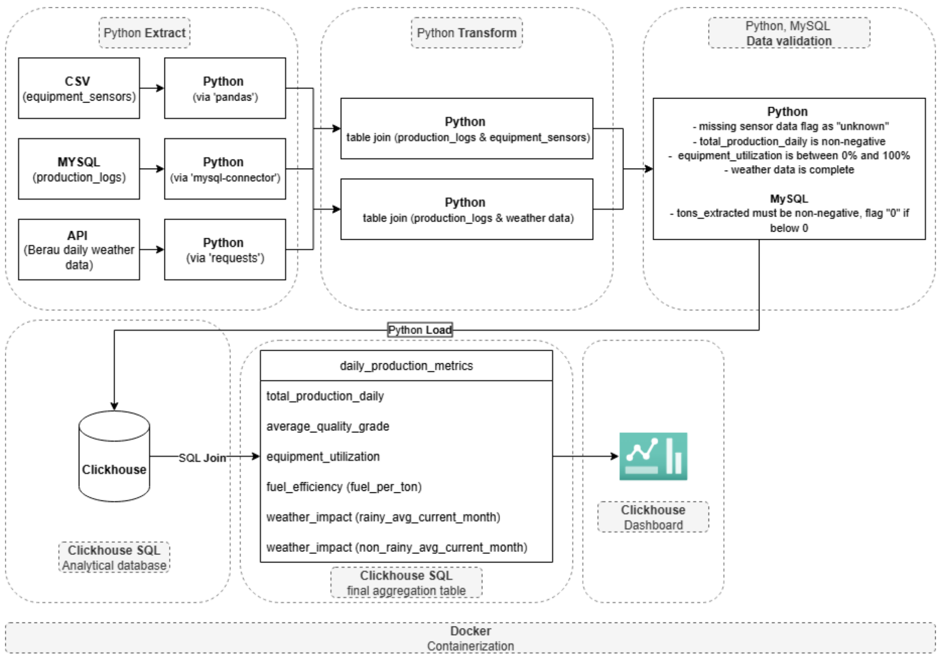
```python
#Error log
import logging

logging.basicConfig(
    filename='log_error.txt',
    level=logging.ERROR,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

#total_production_daily is not negative
if not(df_prod['total_production_daily']>= 0).all() :
```

4. DASHBOARD

***note: I'm most comfortable with Power BI, and can make great charts with PBI, however, because I used Clickhouse on an AWS VM (using Singapore server), I could not get the host url for PBI, so I ended up making basic charts in Clickhouse. I hope this is alright. Thank you!**



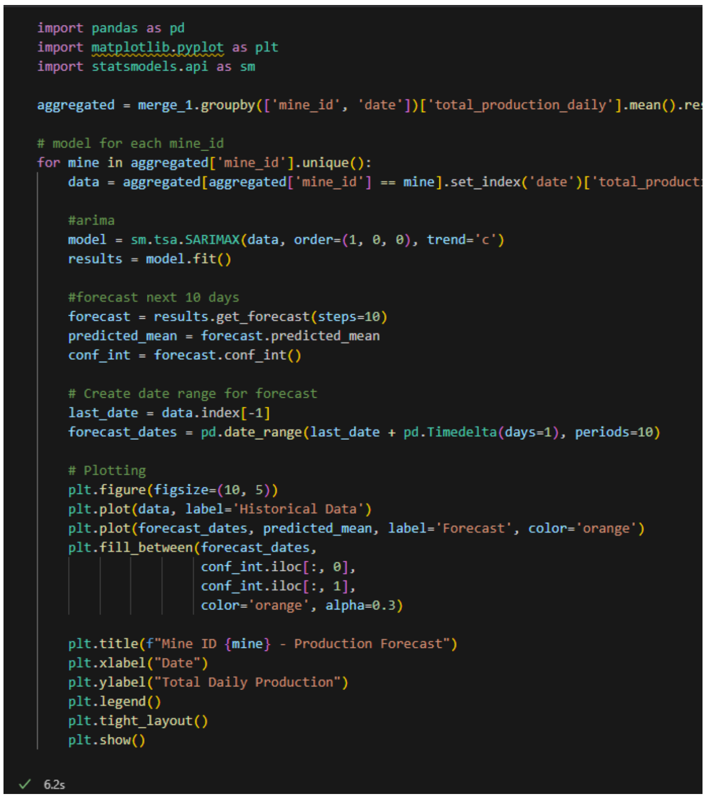
5. DOCUMENT



Python Extract
CSV (equipment_sensors)
Python (via 'pandas')
MYSQL (production_logs)
Python (via 'mysql-connector')
API (Berau daily weather data)
Python (via 'requests')
Python Transform
Python
table join (production_logs & equipment_sensors)
Python
table join (production_logs & weather data)
Python, MySQL Data validation
Python
- missing sensor data flag as "unknown"
- total_production_daily is non-negative
- equipment_utilization is between 0% and 100%
- weather data is complete
MySQL
- tons_extracted must be non-negative, flag "0" if below 0
Python Load
Clickhouse
SQL Join
Clickhouse SQL Analytical database
daily_production_metrics
total_production_daily
average_quality_grade
equipment_utilization
fuel_efficiency (fuel_per_ton)
weather_impact (rainy_avg_current_month)
weather_impact (non_rainy_avg_current_month)
Clickhouse SQL final aggregation table
Clickhouse Dashboard
Docker Containerization

6. PREDICICTIVE MODEL GO TO NEXT PAGE

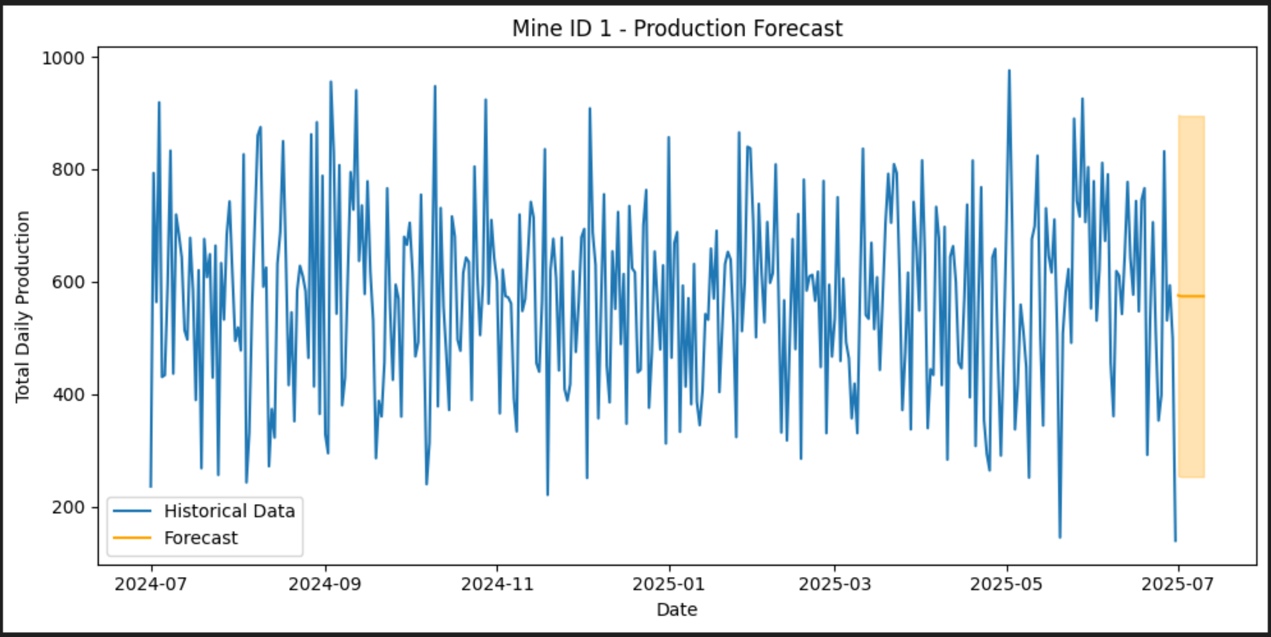
PREDCTIVE MODEL TO FORECAST THE NEXT 2 DAYS for mine_id = 1



```python
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

aggregated = merge_1.groupby(['mine_id', 'date'])['total_production_daily'].mean().res

# model for each mine_id
for mine in aggregated['mine_id'].unique():
    data = aggregated[aggregated['mine_id'] == mine].set_index('date')['total_product

    #arima
    model = sm.tsa.SARIMAX(data, order=(1, 0, 0), trend='c')
    results = model.fit()

    #forecast next 10 days
    forecast = results.get_forecast(steps=10)
    predicted_mean = forecast.predicted_mean
    conf_int = forecast.conf_int()

    # Create date range for forecast
    last_date = data.index[-1]
    forecast_dates = pd.date_range(last_date + pd.Timedelta(days=1), periods=10)

    # Plotting
    plt.figure(figsize=(10, 5))
    plt.plot(data, label='Historical Data')
    plt.plot(forecast_dates, predicted_mean, label='Forecast', color='orange')
    plt.fill_between(forecast_dates,
                     conf_int.iloc[:, 0],
                     conf_int.iloc[:, 1],
                     color='orange', alpha=0.3)

    plt.title(f"Mine ID {mine} - Production Forecast")
    plt.xlabel("Date")
    plt.ylabel("Total Daily Production")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

✓ 6.2s

```
self._init_dates(dates, freq)
```



-- Thank you! --