```python
import datetime
import time

import sqlalchemy
from sqlalchemy.ext.automap import automap_base
from sqlalchemy.orm import Session
from sqlalchemy import create_engine, select, func
import inspect

DMIS_RECORDINGS_DB = 'postgresql://dmis_dbuser:dmis_dbpassword@dmis_db-container/dmis_recordings_db'


def retrieve_name(var):
    callers_local_vars = inspect.currentframe().f_back.f_locals.items()
    return [var_name for var_name, var_val in callers_local_vars if var_val is var][0]


Base = automap_base()

# engine, suppose it has two tables 'user' and 'address' set up

new_adress = DMIS_RECORDINGS_DB.replace('dmis_db-container', 'energy.uni-passau.de:6543')

engine = create_engine(new_adress)

# reflect the tables
Base.prepare(engine, reflect=True)


def sensor_generator():
    metadata_object = sqlalchemy.MetaData()
    metadata_object.reflect(bind=engine)
    i = 1
    for table in metadata_object.sorted_tables:
        yield table, i, metadata_object
        i += 1
```

```

def interval_end_generator(session: Session, table: sqlalchemy.table, interval_length: int, **kwargs):
    interval_column = getattr(table._columns, 'Unixtime Reply')
    interval_begin = kwargs.get('interval_begin', None)
    interval_termination = kwargs.get('interval_termination', time.time() * 1000)
    if interval_begin is None:
        interval_begin = session.execute(select(func.min(interval_column))).first()[0]
    value_in_db = session.execute(select(interval_column).where(interval_column >= interval_begin).limit(1)).all()
    while value_in_db.__len__() > 0 and interval_begin < interval_termination:
        interval_end = interval_begin + interval_length
        yield interval_begin, interval_end, datetime.datetime.fromtimestamp(interval_begin)
        interval_begin = interval_end
        value_in_db = session.execute(select(interval_column).where(interval_column >= interval_begin).limit(1)).all()


def compute_average_power(session: Session, table: sqlalchemy.table, interval_begin: int, interval_end: int):
    interval_column = getattr(table._columns, 'Unixtime Reply')
    power_column = getattr(table._columns, 'Leistung')
    stmnt = select(func.avg(power_column), func.count(power_column)).filter(interval_column >= interval_begin * 1000).filter(interval_column < interval_end * 1000)
    q_result = session.execute(stmnt)
    return_tuple = q_result.first()
    return return_tuple[0], return_tuple[1]
```

```


def create_table_and_insert_values(session: Session, metadata: sqlalchemy.MetaData, table_name: str, values: list):

    if not sqlalchemy.inspect(engine).has_table(f'{table_name}'):
        meta_table = sqlalchemy.Table(f'{table_name}', Base.metadata,
                                      sqlalchemy.Column('timestamp_end', sqlalchemy.Integer, primary_key=True,
                                                        autoincrement=False),
                                      sqlalchemy.Column('avg_power', sqlalchemy.Float, nullable=True),
                                      sqlalchemy.Column('measurement_count', sqlalchemy.Integer, nullable=True))
        meta_table.create(bind=engine)
        metadata_object._add_table(meta_table.name, None, meta_table)
        print(f'table {meta_table.name} created')
    else:
        meta_table = metadata.tables[f'{table_name}']
        print(f'table already exists')
    print('Inserting values ...')
    session.execute(meta_table.insert(), values)
    print('Insert finished')


start_time = datetime.datetime.now()
print(start_time)
session = Session(engine)


def determine_if_table_finished(i: int, metadata_object: sqlalchemy.MetaData):
    table_name = f"Sensor_{i}_meta"
    meta_table = metadata_object.tables.get(f'{
```

```python
table_name}')
    if meta_table is None:
        return False
    qvalue = session.query(meta_table).first()

    if qvalue is None:
        return False
    else:
        return True


for sensor_table, i, metadata_object in sensor_generator():
    if "meta" in sensor_table.name:
        print("meta tables reached, terminating process ...")
        break
    sensor_starttime = datetime.datetime.now()
    if determine_if_table_finished(i, metadata_object):
        print(f'table Sensor{i} is finished, skipping ...')
        continue
    average_power_dict_list = []  # [{timestamp_end : INT , avg_power: INT} ...]
    insert_flush_iterator = 0
    for interval_timestamp_begin, interval_timestamp_end, interval_datetime in interval_end_generator(session, sensor_table,
                                                                                                          900,
                                                                                                          interval_begin=1651356000,
                                                                                                          interval_termination=1656626400):
        average_power, measurement_count = compute_average_power(session, sensor_table, interval_timestamp_begin, interval_timestamp_end)
        average_power_dict_list.append({'timestamp_end': interval_timestamp_end, 'avg_power': average_power,
                                        'measurement_count': measurement_count})
        if insert_flush_iterator >= 999:
            create_table_and_insert_values(session, metadata_object, f'Sensor_{i}_meta', average_power_dict_list)
            average_power_dict_list = []
            insert_flush_iterator = 0
        else:
            insert_flush_iterator += 1

    create_table_and_insert_values(session, metadata_object, f'Sensor_{i}_meta', average_power_dict_list)
    end_time = datetime.datetime.now()
    print(f'table Sensor_{i} completion time: {end_time - sensor_starttime}')
    session.commit()

end_time = datetime.datetime.now()
print(f'completion time: {end_time - start_time}')

```

```python
from sqlalchemy.orm import declarative_base

Base = declarative_base()
```

```python
from datetime import datetime

import calmap
import matplotlib.pyplot as plt
import pandas
import pandas as pd


def create_per_day_plot():
    csv_file_wholeday = 'per_day_report_wholeday_plot.csv'
    values_wholeday = pd.read_csv(csv_file_wholeday, sep=';').squeeze()
    values_wholeday.index = pandas_weekdays
    # create workday series from csv file
    csv_file_workday = 'per_day_report_workday_plot.csv'
    values_workday = pd.read_csv(csv_file_workday, sep=';').squeeze()
    values_workday.index = pandas_weekdays
    # plot the data
    calmap.calendarplot(values_wholeday, cmap='RdYlGn')
    plt.savefig('wholeday_cal.png', dpi=600)
    calmap.calendarplot(values_workday, cmap='coolwarm')
    plt.savefig('workday_cal.png', dpi=600)


def create_per_device_plot():
    csv_file_device = 'per_day_per_sensor_report.csv'
    values_device = pd.read_csv(csv_file_device, sep=';')
    per_sensor_matrizes = {}
    for sensor_id in range(1, 38):
        matrix = values_device[values_device['sensorID'] == sensor_id]
        matrix.set_index('day', inplace=True)
        matrix.index = pandas.to_datetime(matrix.index, format='%d-%m-%Y', dayfirst=True)
        per_sensor_matrizes[sensor_id] = matrix[['
wholeday_avg', 'workday_avg']]

    per_user_cals(per_sensor_matrizes)

    printer_cals(per_sensor_matrizes)

    per_devicetype_cals(per_sensor_matrizes)


def printer_cals(per_sensor_matrizes):
    printer_sensors_matrix = per_sensor_matrizes[22].add(per_sensor_matrizes[12])
    print('Printer:')
    print(printer_sensors_matrix)
    fig, axes = calmap.calendarplot(printer_sensors_matrix[['wholeday_avg']].squeeze(), cmap='coolwarm')
    plt.savefig('printer_wholeday_cal.png', dpi=600)
    plt.clf()
    plt.close(fig)
    fig, axes = calmap.calendarplot(printer_sensors_matrix[['workday_avg']].squeeze(), cmap='coolwarm')
    plt.savefig('printer_workday_cal.png', dpi=600)
    plt.clf()
    plt.close(fig)


def per_user_cals(per_sensor_matrizes):
    user_to_sensor_map = {1: [per_sensor_matrizes[1], per_sensor_matrizes[2], per_sensor_matrizes[3]],
                          2: [per_sensor_matrizes[4], per_sensor_matrizes[5], per_sensor_matrizes[6]],
                          3: [per_sensor_matrizes[7], per_sensor_matrizes[8], per_sensor_matrizes[9]],
                          4: [per_sensor_matrizes[10], per_sensor_matrizes[11], per_sensor_matrizes[12]],
                          5: [per_sensor_matrizes[14], per_sensor_matrizes[15]],
                          6: [per_sensor_matrizes[16], per_sensor_matrizes[17], per_sensor_matrizes[18]],
                          7: [per_sensor_matrizes[19], per_sensor_matrizes[20], per_sensor_matrizes[21]],
                          8: [per_sensor_matrizes[23], per_sensor_matrizes[24], per_sensor_matrizes[25]],
                          9: [per_sensor_matrizes[26], per_sensor_matrizes[27], per_sensor_matrizes[28],
                              per_sensor_matrizes[29]],
                          10: [per_sensor_matrizes[30], per_sensor_matrizes[31]],
                          11: [per_sensor_matrizes[33]],
                          12: [per_sensor_matrizes[34]],
                          13: [per_sensor_matrizes[35]],
                          14: [per_sensor_matrizes[36]],
                          15: [per_sensor_matrizes[37]]
                          }
    for user in user_to_sensor_map:
        matrix_list = user_to_sensor_map[user]
        sensor_matrix = matrix_list[0]
        for i in range(1, len(matrix_list)):
            sensor_matrix = sensor_matrix.add(matrix_list[i], fill_value=0)

        print(f'User {str(user)}:')
        print(sensor_matrix)
        fig, axes = calmap.calendarplot(sensor_matrix[['wholeday_avg']].squeeze(), cmap='coolwarm')
        plt.savefig(f'user_{str(user)}_wholeday_cal.png', dpi=600)
        fig.clear()
        plt.clf()
        plt.close(fig)
        fig, axes = calmap.calendarplot(
```

```python
sensor_matrix[['workday_avg']].squeeze(), cmap='coolwarm')
        fig.savefig(f'user_{str(user)}_workday_cal.png', dpi=600)
        fig.clear()
        plt.clf()
        plt.close(fig)


def per_devicetype_cals(per_sensor_matrizes):
    sensor_csv_file = 'sensor_to_workplace.csv'
    sensor_to_workplace_matrix = pd.read_csv(sensor_csv_file, sep=';')
    sensor_to_devicetype_map = {}
    for entry in sensor_to_workplace_matrix.iterrows():
        sensor_id = entry[1]['sensor_id']
        devicetype = entry[1]['device_type']
        if devicetype not in sensor_to_devicetype_map:
            sensor_to_devicetype_map[devicetype] = [per_sensor_matrizes[sensor_id]]
        else:
            sensor_to_devicetype_map[devicetype].append(per_sensor_matrizes[sensor_id])

    for devicetype in sensor_to_devicetype_map:

        matrix_list = sensor_to_devicetype_map[devicetype]

        if devicetype != 'Multiple':
            sensor_matrix = matrix_list[0]
            for i in range(1, len(matrix_list)):
                sensor_matrix = sensor_matrix.add(matrix_list[i], fill_value=0)
            print(f'Devicetype {str(devicetype)}:')
            print(sensor_matrix)
            fig, axes = calmap.calendarplot(sensor_matrix[['wholeday_avg']].squeeze(), cmap='coolwarm')
            plt.savefig(f'devicetype_{str(devicetype)}_wholeday_cal.png', dpi=600)
            fig.clear()
            plt.clf()
            plt.close(fig)
            fig, axes = calmap.calendarplot(sensor_matrix[['workday_avg']].squeeze(), cmap='coolwarm')
            fig.savefig(f'devicetype_{str(devicetype)}_workday_cal.png', dpi=600)
            fig.clear()
            plt.clf()
            plt.close(fig)
        else:
            numbering = 1
            for sensor in matrix_list:
                print(f'user workplace {numbering}')
                print(sensor)
                fig, axes = calmap.calendarplot(sensor[['wholeday_avg']].squeeze(), cmap='coolwarm')
                plt.savefig(f'user_workplace_{numbering}_wholeday_cal.png', dpi=600)
                fig.clear()
                plt.clf()
                plt.close(fig)
                fig, axes = calmap.calendarplot(sensor[['workday_avg']].squeeze(), cmap='coolwarm')
                fig.savefig(f'user_workplace_{numbering}_workday_cal.png', dpi=600)
                fig.clear()
                plt.clf()
                plt.close(fig)
                numbering += 1


if __name__ == '__main__':
    bdate_start = datetime.fromisoformat('2022-06-01')
    bdate_end = datetime.fromisoformat('2022-07-29')
    pandas_weekdays = pd.bdate_range(start=bdate_start, end=bdate_end, freq='B')
    # create wholeday series from csv file
    create_per_day_plot()
    # create per device series from csv file
    create_per_device_plot()

```

```
from base import Base
from sqlalchemy.types import String, Integer
from sqlalchemy import Column, ForeignKey


PC, MONITOR, UTILITY, PRINTER, MULTIPLE = range(5)

class DeviceType(Base):
    __tablename__ = "deviceType"
    id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(String)


class Device(Base):
    __tablename__ = "device"
    id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(String)
    type = Column(Integer, ForeignKey('deviceType.id'))
    sensor_id = Column(Integer, ForeignKey('sensor.id'))
    workplace = Column(Integer, ForeignKey("workplace.id"))

def get_device_types():
    return [
        DeviceType(id=PC, description="PC"),
        DeviceType(id=MONITOR, description="Monitor"),
        DeviceType(id=UTILITY, description="Utility"),
        DeviceType(id=PRINTER, description="Printer"),
        DeviceType(id=MULTIPLE, description="Multiple")
    ]


def get_devices():
    return [
    Device(id=1, description="PC",type=PC, sensor_id=1, workplace=0),
    Device(id=2, description="PC",type=MONITOR, sensor_id=2, workplace=0),
    Device(id=3, description="PC",type=MONITOR, sensor_id=3, workplace=0),
    Device(id=4, description="PC",type=PC, sensor_id=4, workplace=1),
    Device(id=5, description="PC",type=MONITOR, sensor_id=5, workplace=1),
    Device(id=6, description="PC",type=MONITOR, sensor_id=6, workplace=1),
    Device(id=7, description="PC",type=PC, sensor_id=7, workplace=1),
    Device(id=8, description="PC",type=MONITOR, sensor_id=8, workplace=1),
    Device(id=9, description="PC",type=MONITOR, sensor_id=9, workplace=1),
    Device(id=10, description="PC",type=PC, sensor_id=10, workplace=2),
    Device(id=11, description="PC",type=MONITOR, sensor_id=11, workplace=2),
    Device(id=12, description="PC",type=PRINTER, sensor_id=12, workplace=2),
    Device(id=13, description="PC",type=UTILITY, sensor_id=13, workplace=2),
    Device(id=14, description="PC",type=PC, sensor_id=14, workplace=3),
    Device(id=15, description="PC",type=MONITOR, sensor_id=15, workplace=3),
    Device(id=16, description="PC",type=PC, sensor_id=16, workplace=3),
    Device(id=17, description="PC",type=MONITOR, sensor_id=17, workplace=3),
    Device(id=18, description="PC",type=MONITOR, sensor_id=18, workplace=3),
    Device(id=19, description="PC",type=PC, sensor_id=19, workplace=3),
    Device(id=20, description="PC",type=MONITOR, sensor_id=20, workplace=3),
    Device(id=21, description="PC",type=MONITOR, sensor_id=21, workplace=3),
    Device(id=22, description="PC",type=PRINTER, sensor_id=22, workplace=3),
    Device(id=23, description="PC",type=PC, sensor_id=23, workplace=4),
    Device(id=24, description="PC",type=MONITOR, sensor_id=24, workplace=4),
    Device(id=25, description="PC",type=MONITOR, sensor_id=25, workplace=4),
    Device(id=26, description="PC",type=PC, sensor_id=26, workplace=4),
    Device(id=27, description="PC",type=MONITOR, sensor_id=27, workplace=4),
    Device(id=28, description="PC",type=MONITOR, sensor_id=28, workplace=4),
    Device(id=29, description="PC",type=MONITOR, sensor_id=29, workplace=4),
    Device(id=30, description="PC",type=PC, sensor_id=30, workplace=4),
    Device(id=31, description="PC",type=MONITOR, sensor_id=31, workplace=4),
    Device(id=32, description="PC",type=UTILITY, sensor_id=32),
    Device(id=33, description="PC",type=MULTIPLE, sensor_id=33, workplace=5),
    Device(id=34, description="PC",type=MULTIPLE, sensor_id=34, workplace=6),
    Device(id=35, description="PC",type=MULTIPLE, sensor_id=35, workplace=6),
    Device(id=36, description="PC",type=MULTIPLE, sensor_id=36, workplace=7),
    Device(id=37, description="PC",type=MULTIPLE, sensor_id=37, workplace=7)
]
```

```python
from datetime import datetime
from typing import Tuple, Dict

import numpy
import pandas as pandas
import sqlalchemy.orm
from sqlalchemy import Column, MetaData, ForeignKey, create_engine, String, func, Table
from sqlalchemy.orm import Session
from sqlalchemy.types import Integer

from base import Base
from devices import Device, DeviceType, get_devices, get_device_types
from feedback import Feedbackpost
from sensors import Sensor, get_sensors
from workplaces import Workplace, WorkplaceType, get_workplace_types, get_workplaces

MILLISECONDS_IN_18_HOURS = 64800000
SECONDS_IN_18_HOURS = 64800

MILLISECONDS_IN_8_HOURS = 28800000
SECONDS_IN_8_HOURS = 28800

MILLISECONDS_IN_DAY = 86400000
SECONDS_IN_DAY = 86400

DMIS_RECORDINGS_DB_PATH = "postgresql://dmis_dbuser:dmis_dbpassword@energy.uni-passau.de:6543/dmis_recordings_db"
WORKPLACE_ENUM_TABLE_NAME = "workplaceType"
SENSOR_TO_WORKPLACE_TABLE_NAME = "sensorToWorkplace"
SENSOR_TO_DEVICETYPE_TABLE_NAME = "sensorToDeviceType"


class SensorToWorkplace(Base):
    __tablename__ = SENSOR_TO_WORKPLACE_TABLE_NAME
    sensorID = Column(Integer, ForeignKey('sensor.id'), primary_key=True)
    workplaceID = Column(Integer, ForeignKey('workplace.id'), primary_key=True)


class SensorToDeviceGroup(Base):
    __tablename__ = SENSOR_TO_DEVICETYPE_TABLE_NAME
    sensorID = Column(Integer, ForeignKey('sensor_id'), primary_key=True)
    deviceTypeID = Column(Integer, ForeignKey('deviceType_id'), primary_key=True)


class SensorToDbTable(Base):
    __tablename__ = "sensorToDbTable"
    id = Column(Integer, primary_key=True, autoincrement=True)
    measurement_table = Column(String)
    meta_table = Column(String)


metadata = MetaData()


def insert_information():
    # insert sensors
    sensor_table = Sensor.__table__
    workplace_type_table = WorkplaceType.__table__
    workplace_table = Workplace.__table__
    feedback_table = Feedbackpost.__table__
    device_type_table = DeviceType.__table__
    device_table = Device.__table__
    metadata.create_all(engine,
                        tables=[sensor_table, workplace_type_table, workplace_table, feedback_table, device_type_table,
                                device_table])
    session.add_all(get_sensors())
    session.commit()
    session.add_all(get_workplace_types())
    session.commit()
    session.add_all(get_workplaces())
    session.commit()
    session.add_all(get_device_types())
    session.commit()  # commit to avoid foreign key violation
    session.add_all(get_devices())
    session.commit()


def min_max_avg_consumption_per_device_type(measurement_table_map: Dict, device: Device, dbsession: Session,
                                            column_name: str) -> Tuple[
    float, float, float]:
    min_consumption_query = func.min(getattr(measurement_table_map[device.sensor_id]._columns, column_name))
    max_consumption_query = func.max(getattr(measurement_table_map[device.sensor_id]._columns, column_name))
    avg_consumption_query = func.avg(getattr(measurement_table_map[device.sensor_id]._columns, column_name))

    return dbsession.execute(min_consumption_query).scalar(), dbsession.execute(
        max_consumption_query).scalar(), dbsession.execute(avg_consumption_query).scalar()


def compute_results(metavalues=True):
    report = ""
    metareport = ""

    # map devices to sensors
    measurement_table_map = {}
    meta_table_map = {}
    for sensor in session.query(Sensor).all():
        if sensor.name != "":
            measurement_table_map[sensor.id] = Table(sensor.name, metadata, autoload_with=engine)
            meta_table_map[sensor.id] = Table(sensor.meta_name, metadata, autoload_with=engine)

    # min, max and average consumption per device type
    metareport, report = device_and_type_report(measurement_table_map, meta_table_map, metareport, metavalues, report)

    # min, max and average consumption per workplace type
    metareport, report = workplace_and_type_report(measurement_table_map, meta_table_map, metareport, metavalues,
                                                   report)

    # min, max and average consumption per workplace on weekdays
    metareport, report = days_and_workdays_report(measurement_table_map, meta_table_map, metareport, metavalues, report)

    with open("results.txt", "w") as f:
        f.write(report)
    if metavalues:
        with open("meta_results.txt", "w") as f:
            f.write(metareport)


def device_and_type_report(measurement_table_map, meta_table_map, metareport, metavalues, report):
    for device_type in session.query(DeviceType).all():
        report += f"Device type: {device_type.description}\n"
        mins_per_device_type, maxs_per_device_type, avgs_per_device_type, meta_mins_per_device_type, meta_maxs_per_device_type, meta_avgs_per_device_type = (
        [] for i in range(6))
        if metavalues:
            metareport += f"Device type: {device_type.description}\n"
        for device in session.query(Device).filter(Device.type == device_type.id, Sensor.id == Device.sensor_id):
            if device.sensor_id in measurement_table_map.keys():
                min_report, max_report, avg_report = min_max_avg_consumption_per_device_type(measurement_table_map,
                                                                                             device, session,
                                                                                             "Leistung")
                mins_per_device_type.append(min_report)
                maxs_per_device_type.append(max_report)
                avgs_per_device_type.append(avg_report)
                report += f"  Device: {str(device.id)} {device_type.description}\n"
                report += f"    Min consumption: {str(min_report)}\n"
                report += f"    Max consumption: {str(max_report)}\n"
                report += f"    Avg consumption: {str(avg_report)}\n"
            if metavalues and device.sensor_id in meta_table_map.keys():
                min_meta, max_meta, avg_meta = min_max_avg_consumption_per_device_type(meta_table_map, device, session,
                                                                                       "avg_power")
                meta_mins_per_device_type.append(min_meta)
                meta_maxs_per_device_type.append(max_meta)
                meta_avgs_per_device_type.append(avg_meta)
                metareport += f"  Sensor: {str(device.id)} {device_type.description}\n"
                metareport += f"    Min consumption : {str(min_meta)}\n"
                metareport += f"    Max consumption : {str(max_meta)}\n"
                metareport += f"    Avg consumption : {str(avg_meta)}\n"
        report += f" Total min consumption: {str(min(mins_per_device_type))}\n"
        report += f" Total max consumption: {str(max(maxs_per_device_type))}\n"
        report += f" Total avg consumption: {str(sum(avgs_per_device_type) / len(avgs_per_device_type))}\n\n"
        if metavalues:
            metareport += f" Total min consumption: {str(min(meta_mins_per_device_type))}\n"
            metareport += f" Total max consumption: {str(max(meta_maxs_per_device_type))}\n"
            metareport += f" Total avg consumption: {str(sum(meta_avgs_per_device_type) / len(meta_avgs_per_device_type))}\n\n"
    return metareport, report


def workplace_and_type_report(measurement_table_map, meta_table_map, metareport, metavalues, report):
    for workplace_type in session.query(WorkplaceType).all():
        report += f"Workplace type: {workplace_type.description}\n"
        mins_per_workplace_type, maxs_per_workplace_type, avgs_per_workplace_type, meta_mins_per_workplace_type, meta_maxs_per_workplace_type, meta_avgs_per_workplace_type = (
        [] for i in range(6))
        if metavalues:
            metareport += f"Workplace type: {workplace_type.description}\n"
        for workplace in session.query(Workplace).filter(workplace_type.id == Workplace.type):
            report += f"  Workplace: {str(workplace.id)} {workplace_type.description}\n"
            mins_per_workplace, maxs_per_workplace, avgs_per_workplace, meta_mins_per_workplace, meta_maxs_per_workplace, meta_avgs_per_workplace = (
            [] for i in range(6))
            if metavalues:
                metareport += f"  Workplace: {str(workplace.id)} {workplace_type.description}\n"
            for device in session.query(Device).filter(Device.workplace == workplace.id):
                report += f"    Device: {str(device.id)} {device.description}\n"
                if device.sensor_id in measurement_table_map.keys():
                    min_report, max_report, avg_report = min_max_avg_consumption_per_device_type(measurement_table_map,
                                                                                                  device,
                                                                                                  session, "Leistung")
                    mins_per_workplace.append(min_report)
                    maxs_per_workplace.append(max_report)
                    avgs_per_workplace.append(avg_report)

                    report += f"      Min consumption: {str(min_report)}\n"
                    report += f"      Max consumption: {str(max_report)}\n"
                    report += f"      Avg consumption: {str(avg_report)}\n"
                    if metavalues and device.
```

```
sensor_id in meta_table_map.keys():
                        min_meta, max_meta, avg_meta = min_max_avg_consumption_per_device_type(meta_table_map, device,
                                                                                                session, "avg_power")
                        meta_mins_per_workplace.append(min_meta)
                        meta_maxs_per_workplace.append(max_meta)
                        meta_avgs_per_workplace.append(avg_meta)
                        metareport += f"    Device: {str(device.id)} {device.description}\n"
                        metareport += f"      Min consumption: {str(min_meta)}\n"
                        metareport += f"      Max consumption: {str(max_meta)}\n"
                        metareport += f"      Avg consumption: {str(avg_meta)}\n"
            meta_workplace_min = min(mins_per_workplace)
            meta_workplace_max = max(maxs_per_workplace)
            meta_workplace_avg = sum(avgs_per_workplace) / len(avgs_per_workplace)
            report += f"   Total min consumption: {str(meta_workplace_min)}\n"
            report += f"   Total max consumption: {str(meta_workplace_max)}\n"
            report += f"   Total avg consumption: {str(meta_workplace_avg)}\n\n"
            mins_per_workplace_type.append(meta_workplace_min)
            maxs_per_workplace_type.append(meta_workplace_max)
            avgs_per_workplace_type.append(meta_workplace_avg)
            if metavalues:
                meta_workplace_min = min(
```

```python
meta_mins_per_workplace)
                meta_workplace_max = max(meta_maxs_per_workplace)
                meta_workplace_avg = sum(meta_avgs_per_workplace) / len(meta_avgs_per_workplace)
                metareport += f"   Total min consumption: {str(meta_workplace_min)}\n"
                metareport += f"   Total max consumption: {str(meta_workplace_max)}\n"
                metareport += f"   Total avg consumption: {str(meta_workplace_avg)}\n"
                meta_mins_per_workplace_type.append(meta_workplace_min)
                meta_maxs_per_workplace_type.append(meta_workplace_max)
                meta_avgs_per_workplace_type.append(meta_workplace_avg)
    return metareport, report


def days_and_workdays_report(measurement_table_map, meta_table_map, metareport, metavalues, report,
                             report_per_day=True):
    per_day_per_sensor_report = "sensorID;day;wholeday_avg;workday_avg\n"
    per_day_avg_values_dict, per_workday_avg_values_dict = {}, {}
    meta_result_list_whole_day, meta_result_list_work_day = [], []
    per_day_report = "day;wholeday_avg;workday_avg\n"
    bdate_start = datetime.fromisoformat('2022-06-01')
    bdate_end = datetime.now()
    pandas_weekdays = pandas.bdate_range(start=bdate_start, end=bdate_end, freq='B').astype(numpy.int64)
    report += f"Workplace consumption on weekdays from {bdate_start} until {bdate_end} or data becomes unavailable\n"
    if metavalues:
        metareport += f"Workplace consumption on weekdays from {bdate_start} until {bdate_end} or data becomes unavailable\n"
    for (sensor_id, table, meta_table) in zip(measurement_table_map.keys(), measurement_table_map.values_wholeday(),
                                              meta_table_map.values_wholeday()):
        result_list_whole_day, result_list_work_day = [], []
        for weekday_ns in pandas_weekdays:
            weekday_ms = int(weekday_ns / 1000000)
            weekday_s = int(weekday_ms / 1000)
            power_column = getattr(table._columns, "Leistung")
            timestamp_column = getattr(table._columns, "Unixtime Reply")
            whole_weekday_data = session.query(func.avg(power_column)).filter(
                timestamp_column.between(weekday_ms, weekday_ms + MILLISECONDS_IN_DAY)).scalar()
            # workday from 08:00 to 18:00
            workday_data = session.query(func.avg(power_column)).filter(
                timestamp_column.between(weekday_ms + MILLISECONDS_IN_8_HOURS,
                                         weekday_ms + MILLISECONDS_IN_18_HOURS)).scalar()
            if whole_weekday_data is not None and workday_data is not None:
                result_list_whole_day.append(whole_weekday_data)
                result_list_work_day.append(workday_data)
                if report_per_day:
                    currently_processed_day_iso = datetime.utcfromtimestamp(weekday_s).strftime('%D/%M/%Y')
                    if currently_processed_day_iso not in per_day_avg_values_dict.keys():
                        per_day_avg_values_dict[currently_processed_day_iso] = whole_weekday_data
                        per_workday_avg_values_dict[currently_processed_day_iso] = workday_data
                    else:
                        per_day_avg_values_dict[currently_processed_day_iso] += workday_data
                        per_workday_avg_values_dict[currently_processed_day_iso] += workday_data
                    per_day_per_sensor_report += f"{sensor_id};{currently_processed_day_iso};{whole_weekday_data};{workday_data}\n"
            if metavalues:
                avg_power_column = getattr(meta_table._columns, "avg_power")
                timestamp_end_column = getattr(meta_table._columns, "timestamp_end")
                whole_weekday_meta_data = session.query(func.avg(avg_power_column)).filter(
                    timestamp_end_column.between(weekday_s, weekday_s + SECONDS_IN_DAY)).scalar()
                workday_meta_data = session.query(func.avg(avg_power_column)).filter(
                    timestamp_end_column.between(weekday_s + SECONDS_IN_8_HOURS,
                                                 weekday_s + SECONDS_IN_18_HOURS)).scalar()
                if whole_weekday_meta_data is not None and workday_meta_data is not None:
                    meta_result_list_whole_day.append(whole_weekday_meta_data)
                    meta_result_list_work_day.append(workday_meta_data)
        nr_of_rec_wholedays = len(result_list_whole_day)
        wholedays_avg_value = sum(result_list_whole_day) / nr_of_rec_wholedays
        report += f"AVG consumption per weekday for Sensor{sensor_id} in {nr_of_rec_wholedays} days: {str(wholedays_avg_value)}\n"
        nr_of_rec_workdays = len(
result_list_work_day)
        workdays_avg_value = sum(
result_list_work_day) / nr_of_rec_workdays
        report += f"AVG consumption per workday for
Sensor{sensor_id} in {nr_of_rec_workdays} workdays:
{str(workdays_avg_value)}\n"
        if metavalues:
            nr_of_recorded_wholedays = len(
meta_result_list_whole_day)
            wholedays_meta_avg_value = sum(
meta_result_list_whole_day) /
nr_of_recorded_wholedays
            metareport += f"AVG consumption per
weekday for Sensor{sensor_id} in {
nr_of_recorded_wholedays} days: {str(
wholedays_meta_avg_value)}\n"
            nr_of_recorded_workdays = len(
meta_result_list_work_day)
            workdays_meta_avg_value = sum(
meta_result_list_work_day) / nr_of_recorded_workdays
            metareport += f"AVG consumption per
workday for Sensor{sensor_id} in {
nr_of_recorded_workdays} workdays: {str(
workdays_meta_avg_value)}\n"
    if report_per_day:
        per_day_report += f"Overview of the sum of
all sensors per day and workday\ndate;wholeday_avg;
workday_avg\n"
        for entry in per_day_avg_values_dict.keys():
            per_day_report += f"{entry};{
per_day_avg_values_dict[entry]};{
per_workday_avg_values_dict[entry]}\n"
        with open('per_day_report.csv', 'w') as f:
            f.write(per_day_report)
        with open('per_day_per_sensor_report.csv', '
w') as f:
            f.write(per_day_per_sensor_report)
    return metareport, report


if __name__ == '__main__':
    engine = create_engine(
        DMIS_RECORDINGS_DB_PATH,
        echo=False)

    Session = sqlalchemy.orm.sessionmaker(bind=engine)

    session = Session()

    insert_information()
    start = datetime.now()
    compute_results()
    end = datetime.now()
    print("Duration: " + str(end - start))

```

```python
from base import Base
from sqlalchemy.types import String, Integer
from sqlalchemy import Column


class Sensor(Base):
    __tablename__ = 'sensor'
    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
    meta_name = Column(String)

def get_sensors():
    return [
    Sensor(id=1, name="BLADL_00_001", meta_name="Sensor_1_meta"),
    Sensor(id=2, name="BLADL_00_002", meta_name="Sensor_2_meta"),
    Sensor(id=3, name="BLADL_00_003", meta_name="Sensor_3_meta"),
    Sensor(id=4, name="BLADL_00_004", meta_name="Sensor_4_meta"),
    Sensor(id=5, name="BLADL_00_005", meta_name="Sensor_5_meta"),
    Sensor(id=6, name="BLADL_00_006", meta_name="Sensor_6_meta"),
    Sensor(id=7, name="BLADL_00_007", meta_name="Sensor_7_meta"),
    Sensor(id=8, name="BLADL_00_008", meta_name="Sensor_8_meta"),
    Sensor(id=9, name="BLADL_00_009", meta_name="Sensor_9_meta"),
    Sensor(id=10, name="BLADL_00_010", meta_name="Sensor_10_meta"),
    Sensor(id=11, name="BLADL_00_011", meta_name="Sensor_11_meta"),
    Sensor(id=12, name="BLADL_00_012", meta_name="Sensor_12_meta"),
    Sensor(id=13, name="BLADL_00_013", meta_name="Sensor_13_meta"),
    Sensor(id=14, name="BLADL_00_014", meta_name="
Sensor_14_meta"),
    Sensor(id=15, name="BLADL_00_015", meta_name="Sensor_15_meta"),
    Sensor(id=16, name="BLADL_00_016", meta_name="Sensor_16_meta"),
    Sensor(id=17, name="BLADL_00_017", meta_name="Sensor_17_meta"),
    Sensor(id=18, name="BLADL_00_018", meta_name="Sensor_18_meta"),
    Sensor(id=19, name="BLADL_00_019", meta_name="Sensor_19_meta"),
    Sensor(id=20, name="BLADL_00_020", meta_name="Sensor_20_meta"),
    Sensor(id=21, name="BLADL_00_021", meta_name="Sensor_21_meta"),
    Sensor(id=22, name="BLADL_00_022", meta_name="Sensor_22_meta"),
    Sensor(id=23, name="BLADL_00_023", meta_name="Sensor_23_meta"),
    Sensor(id=24, name="BLADL_00_024", meta_name="Sensor_24_meta"),
    Sensor(id=25, name="BLADL_00_025", meta_name="Sensor_25_meta"),
    Sensor(id=26, name="BLADL_00_026", meta_name="Sensor_26_meta"),
    Sensor(id=27, name="BLADL_00_027", meta_name="Sensor_27_meta"),
    Sensor(id=28, name="BLADL_00_028", meta_name="Sensor_28_meta"),
    Sensor(id=29, name="BLADL_00_029", meta_name="Sensor_29_meta"),
    Sensor(id=30, name="BLADL_00_030", meta_name="Sensor_30_meta"),
    Sensor(id=31, name="BLADL_00_031", meta_name="Sensor_31_meta"),
    Sensor(id=32, name="BLADL_00_032", meta_name="Sensor_32_meta"),
    Sensor(id=33, name="BLADL_00_033", meta_name="Sensor_33_meta"),
    Sensor(id=34, name="BLADL_00_034", meta_name="Sensor_34_meta"),
```

```python
    Sensor(id=35, name="BLADL_00_035", meta_name="Sensor_35_meta"),
    Sensor(id=36, name="BLADL_00_036", meta_name="Sensor_36_meta"),
    Sensor(id=37, name="BLADL_00_037", meta_name="Sensor_37_meta")
]

```

```python
from sqlalchemy.orm import relationship
from sqlalchemy.types import String, Integer
from sqlalchemy import ForeignKey
from sqlalchemy import Column
from base import Base


class WorkplaceType(Base):
    __tablename__ = 'workplaceType'
    id = Column(Integer, primary_key=True, autoincrement=True)
    description = Column(String)

class Workplace(Base):
    __tablename__ = 'workplace'
    id = Column(Integer, primary_key=True, autoincrement=True)
    type = Column(Integer, ForeignKey('workplaceType.id'))


def get_workplace_types():
    return [
    WorkplaceType(id=0, description="Single"),
    WorkplaceType(id=1, description="Dual"),
    WorkplaceType(id=2, description="Multi")
]

def get_workplaces():
    return [
    Workplace(id=0, type=0), # Sensor 1-3
    Workplace(id=1, type=1), # Sensor 4-9
    Workplace(id=2, type=0), # Sensor 10-13
    Workplace(id=3, type=0), # Sensor 14
    Workplace(id=4, type=2), # Sensor 15 -23
    Workplace(id=5, type=2), # Sensor 24 - 32
    Workplace(id=6, type=1), # Sensor 34 - 37
    Workplace(id=7, type=1) # Sensor 38 - 39
]

```