Database script

```
import pyodbc
import logging
from sqlalchemy import create_engine


class DatabaseConnection:
    """
    Connection to MS SQL Server (or other) database
    """

    # Configuring file for logs
    logging.basicConfig(filename='app.log',
                        format='%(asctime)s - %(levelname)s - %(message)s',
                        level=logging.INFO,
                        datefmt='%d.%m.%Y %H:%M:%S')

    # Credentials for database
    __username_database = 'dbi338283_psvproject'
    __password_database = 'datastic123'
    __dsn = 'MYMSSQL'

    # All messages (info, logs, etc.) would start with
    __info_message_start = '[DATABASE]'

    def __init__(self):
        """
        This function initializes the database connection.
        """

        try:
            # Format of string:
            # {Database Type}+{Database
Connector}://{login}:{password}@{host}:{port}/{Database}?driver={Driver
with spaces replaced with +}

            # Initializing the database
            self.engine = create_engine(

f'mssql+pyodbc://{self.__username_database}:{self.__password_database}@{sel
f.__dsn}')
            self.conn = pyodbc.connect(

f'DSN={self.__dsn};UID={self.__username_database};PWD={self.__password_data
base}')

            with self.conn:
                self.cursor = self.conn.cursor()

            print(f'{self.__info_message_start} Connection to the database
was established!')
            logging.info(f'{self.__info_message_start} Connection to the
database was established!')

        except pyodbc.ProgrammingError:
            print(f'{self.__info_message_start} Wrong credentials for
database!')
            logging.exception()
        except pyodbc.InterfaceError:
            print(f'{self.__info_message_start} Wrong DNS or driver!')
```

```
            logging.exception(f'{self.__info_message_start} Wrong DNS or
driver!')
        except pyodbc.OperationalError:
            print(f'{self.__info_message_start} Unable to connect to the
database!')
            logging.exception(f'{self.__info_message_start} Unable to
connect to the database!')
        except Exception as e:
            print(f'{self.__info_message_start} Error while connecting to
the database: {e}.')
            logging.exception(f'{self.__info_message_start} Error while
connecting to the database: {e}.')

    def close_connection(self):
        """
        This function closes the database connection.
        :return: None
        """
        # noinspection PyInterpreter
        try:
            self.cursor.close()
        except Exception as e:
            print(f'{self.__info_message_start} Error while closing the
connection: {e}.')
            logging.exception(f'{self.__info_message_start} Error while
closing the connection: {e}.')
```

Main script:

```
from coosto import CoostoAPI
from ortec import OrtecAPI
from database import DatabaseConnection


if __name__ == '__main__':
    credentials_coosto = {'username': 'j.vandermeulen@psv.nl', 'password':
'b4jfb4G%3@s8'}
    results_path = 'export/'

    try:
        # Creating instances of API and database connection
        dbc = DatabaseConnection()
        ortec_api = OrtecAPI(results_path, 'OrtecSDFTest', 'DF1@ORTEC',
match_id=75201, db_conn=dbc)
        coosto_api = CoostoAPI(results_path, **credentials_coosto,
db_conn=dbc)

        # Export Ortec data
        ortec_api.players_stats_export(output='csv')
        ortec_api.teams_stats_export(output='csv')
        ortec_api.player_stats_meta_export(output='csv')
        ortec_api.team_stats_meta_export(output='csv')
        ortec_api.positions_meta_export(output='csv')
        ortec_api.venues_meta_export(output='csv')
        ortec_api.match_info_export(output='csv')  # For now 'replace' mode
is used in order to eliminate errors with PK
        ortec_api.persons_export(output='csv', team_id=8326)
        ortec_api.teams_export(output='csv')
        ortec_api.export_team_lineups(output='csv', team_id=8326)
```

```
        # Set PK and FK for Ortec data
        ortec_api.set_pk_and_fk()

        #ortec_api.clean_database()

        # Export Coosto data
        coosto_api.export_all(output='csv')

        # Close database connection
        dbc.close_connection()

    except Exception as e:
        print(f'Error: {e.with_traceback()}')

    # Add file with settings for sql server

    # ADD LOGGING

    # COMMENT UNNECESSARY COLUMNS

    # LOGIC OF RUNNING SCRIPT (SCHEMA WITH UPDATE INTERVALS)

    # CONNECT INTERNAL DATA SOURCES


    # CREATE README.MD

    # First tables need to be created, then add PK and FK. Otherwise, some
of tables may not be created yet
    # and FK could not be added
```

Ortec Script

```
import requests
import json
import pandas as pd
import os
import logging
from sqlalchemy import inspect


class OrtecAPI:
    """
    API for players' statistics provider (Ortec)
    """

    # Configuring file for logs
    logging.basicConfig(filename='app.log',
                        format='%(asctime)s - %(levelname)s - %(message)s',
                        level=logging.INFO,
                        datefmt='%d.%m.%Y %H:%M:%S')

    # Link to the main api
    api_url = 'https://sports.ortec-hosting.com/EIADataFeedApi/'
```

```
    # Change table names (values in dict) only here . Table names further
in a script would change automatically
    __tables = {'table_teams': 'Teams',
                'table_persons': 'Persons',
                'table_playerstats': 'PlayersStats',
                'table_playerstatsmeta': 'PlayerStatsMeta',
                'table_teamstats': 'TeamsStats',
                'table_teamstatsmeta': 'TeamStatsMeta',
                'table_positionsmeta': 'PositionsMeta',
                'table_venuesmeta': 'VenuesMeta',
                'table_matches': 'Matches',
                'table_teamslineup': 'TeamsLineUp'}

    # Defining possible options for output formats and SQL modes
    __possible_outputs = ['csv', 'sql', 'csv-sql']
    __possible_modes = ['append', 'replace']

    # All messages (info, logs, etc.) would start with
    __info_message_start = '[ORTEC]'

    def __init__(self, path, username, password, match_id, db_conn):
        """
        This function initializes the Ortec API connection.
        :param path: Path to folder where the results should be.
        :param username: Username for API
        :param password: Password for API
        :param match_id: ID of match the information should be extracted
about
        :param db_conn: Database connection instance where the results
should be exported
        """
        self.path = path
        self.username = username
        self.password = password
        self.match_id = match_id
        self.__db = db_conn

        # Credentials for authorizing
        credentials = {'username': username, 'password': password}

        try:
            # Received token: is needed to access the data
            __token = requests.post(self.api_url + 'api/token',
data=credentials).text

            # Create authorization header (removing quotes symbols from
response)
            self.auth = f'Session {__token[1:-1]}'

            # Get players' stats for match with 'match_id'
            self.match_stats_api = requests.get(self.api_url +
f'api/Registration/{match_id}/Statistics',
                                                headers={'Authorization':
self.auth})
            self.match_stats = json.loads(self.match_stats_api.text)

        except Exception as e:
            print(f'{self.__info_message_start} Error while connecting to
the API: {e}.')
            logging.exception(f'{self.__info_message_start} Error while
```

```
connecting to the API: {e}.')
            return

        print(f'{self.__info_message_start} Connection to the API was
established!')
        logging.info(f'{self.__info_message_start} Connection to the API
was established!')

        # Path to the folder with all result files
        self.path_to_results = path + '/ortec_data/'
        if not os.path.exists(self.path_to_results):
            os.makedirs(self.path_to_results)

    def __export(self, df, table, output, mode):
        """
        This function takes a dataframe and then exports it to CSV file
or/and MS SQL Server.
        :param df: DataFrame to export (what to export).
        :param table: Table in SQL database to put the result in.
        :param output: Where to export:
                        csv - only in CSV file
                        sql - only in MS SQL Server Database
                        csv-sql - both CSV and database
        :param mode: How to insert data? Only applicable if output contains
sql
                        append - add data to the previous rows
                        replace - remove all previous data from table, then
add new
        :return: None
        """
        try:
            if output == 'csv':
                df.to_csv(self.path_to_results + self.__tables[table] +
'.csv')
            elif output == 'sql':
                df.to_sql(self.__tables[table], self.__db.engine,
if_exists=mode, index=False)
            elif output == 'csv-sql':
                df.to_csv(self.path_to_results + self.__tables[table] +
'.csv')
                df.to_sql(self.__tables[table], self.__db.engine,
if_exists=mode, index=False)

        except FileNotFoundError:
            print(f'{self.__info_message_start} Wrong path to the folder or
file: {self.path_to_results}.')
            logging.exception(f'{self.__info_message_start} Wrong path to
the folder or file: {self.path_to_results}.')
            return
        except Exception as e:
            print(f'{self.__info_message_start} Error while exporting table
<{self.__tables[table]}>: {e}.')
            logging.exception(f'{self.__info_message_start} Error while
exporting table <{self.__tables[table]}>: {e}.')
            return

        print(
            f'{self.__info_message_start} Table <{self.__tables[table]}>
was successfully exported to {output}!')
        logging.info(
            f'{self.__info_message_start} Table <{self.__tables[table]}>
```

```
was successfully exported to {output}!')

    def __check_output_and_mode(self, output, mode, table):
        """
        This function checks if provided output format and/or SQL mode are
in a list of available options.
        :param output: Provided output format. Should be "csv", "sql" or
"csv-sql".
        :param mode: Provided SQL mode. Should be "append" or "replace".
        :param table: Table in SQL database to put the result in.
        :return: True or False depending on the result of check.
        """

        if output not in self.__possible_outputs or mode not in
self.__possible_modes:
            print(f'{self.__info_message_start} Wrong output format or SQL
mode for table <{self.__tables[table]}>!')
            logging.exception(f'{self.__info_message_start} Wrong output
format or SQL mode for table <{self.__tables[table]}>!')
            return False
        else:
            return True

    def set_pk_and_fk(self):
        """
        Sets the primary and foreign keys for tables (making relations
between tables).
        :return: None
        """

        # List of tuples in format: (TableName, PrimaryKey,
ListOfForeignKeys(FK, ReferenceTable, ReferenceTablePK))
        tables_pk_fk = [(self.__tables['table_teams'], 'Id'),
                        (self.__tables['table_matches'], 'MatchID',
[('HomeTeamId', self.__tables['table_teams'], 'Id'),

('AwayTeamId', self.__tables['table_teams'], 'Id'),

('VenueId', self.__tables['table_venuesmeta'],

'Id')]),
                        (self.__tables['table_persons'], 'PersonId',
                         [('DefaultPosition',
self.__tables['table_positionsmeta'], 'Id')]),
                        (self.__tables['table_venuesmeta'], 'Id'),
                        (self.__tables['table_playerstatsmeta'], 'Id'),
                        (self.__tables['table_teamstatsmeta'], 'Id'),
                        (self.__tables['table_positionsmeta'], 'Id'),
                        (self.__tables['table_playerstats'], None,
                         [('MatchID', self.__tables['table_matches'],
'MatchID'),
                          ('PersonID', self.__tables['table_persons'],
'PersonID'),
                          ('StatisticID',
self.__tables['table_playerstatsmeta'],
                           'Id')]),
                        (self.__tables['table_teamstats'], None,
[('MatchID', self.__tables['table_matches'], 'MatchID'),
                                                                ('TeamID',
self.__tables['table_teams'], 'Id'),
```

```
('StatisticID', self.__tables['table_teamstatsmeta'],
                                                                'Id')])]

        # Iterating through every table to set PK
        for table in tables_pk_fk:
            try:
                query_pk = "SELECT Col.Column_Name FROM \
                            INFORMATION_SCHEMA.TABLE_CONSTRAINTS Tab, \
                            INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE Col
\
                            WHERE Col.Constraint_Name = Tab.Constraint_Name
\
                            AND Col.Table_Name = Tab.Table_Name \
                            AND Constraint_Type = 'PRIMARY KEY' \
                            AND Col.Table_Name = ?;"

                self.__db.cursor.execute(query_pk, table[0])
                checker_pk = len(self.__db.cursor.fetchall()) == 0  #
Checking if PK already exists

                # If PK was not set before and table has a PK
                if checker_pk and table[1] is not None:
                    self.__db.cursor.execute(f'ALTER TABLE {table[0]} ALTER
COLUMN {table[1]} INT NOT NULL')
                    self.__db.cursor.execute(f'ALTER TABLE {table[0]} ADD
PRIMARY KEY ({table[1]});')
                    self.__db.conn.commit()

            except Exception as e:
                print(f'{self.__info_message_start} Error while setting PK
<{table[1]}> for table <{table[0]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error while
setting PK <{table[1]}> for table <{table[0]}>: {e}.')
                continue

            print(f'{self.__info_message_start} PK <{table[1]}> for table
<{table[0]}> was successfully added!')
            logging.info(f'{self.__info_message_start} PK <{table[1]}> for
table <{table[0]}> was successfully added!')

        # Iterating through every table to set FK
        for table in tables_pk_fk:
            query_fk = "SELECT Col.Column_Name FROM \

INFORMATION_SCHEMA.TABLE_CONSTRAINTS Tab, \

INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE Col \
                                        WHERE Col.Constraint_Name =
Tab.Constraint_Name \
                                        AND Col.Table_Name = Tab.Table_Name
\
                                        AND Constraint_Type = 'FOREIGN KEY'
\
                                        AND Col.Table_Name = ?;"

            # If table has FK
            if len(table) == 3:
                try:
                    self.__db.cursor.execute(query_fk, table[0])
                    result = self.__db.cursor.fetchall()
                except Exception as e:
```

```
                    print(f'{self.__info_message_start} Error while setting
FK for table <{table[0]}>: {e}.')
                    logging.exception(f'{self.__info_message_start} Error
while setting FK for table <{table[0]}>: {e}.')
                    continue

                # Iterating through each FK
                for fk_reference in table[2]:
                    try:
                        checker_fk = (fk_reference[0],) not in result  #
Checking if FK already exists

                        if checker_fk:
                                self.__db.cursor.execute(
                                    f'ALTER TABLE {table[0]} ALTER COLUMN
{fk_reference[0]} INT')  # PK and FK should be the same data type
                                self.__db.cursor.execute(
                                    f'ALTER TABLE {table[0]} WITH NOCHECK
ADD FOREIGN KEY ({fk_reference[0]}) \
                                    REFERENCES {fk_reference[1]}
({fk_reference[2]}) ON DELETE NO ACTION;')  # NOCHECK allows to avoid error
with FK (when you don't have a corresponding record in parent table)
                                self.__db.conn.commit()

                    except Exception as e:
                        print(f'{self.__info_message_start} Error while
setting FK <{fk_reference[0]}> for table <{table[0]}>: {e}.')
                        logging.exception(f'{self.__info_message_start}
Error while setting FK <{fk_reference[0]}> for table <{table[0]}>: {e}.')
                        continue

                    print(f'{self.__info_message_start} FK
<{fk_reference[0]}> for table <{table[0]}> was successfully added!')
                    logging.info(f'{self.__info_message_start} FK
<{fk_reference[0]}> for table <{table[0]}> was successfully added!')

    def drop_pk_and_fk(self):
        """
        Drops all primary and foreign keys for tables (deleting relations
between tables)
        :return: None
        """

        # SQL query for dropping FK
        query_fk = "SELECT 'alter table ' + SCHEMA_NAME(Schema_id)+'.'+
object_name(parent_object_id) \
                + '  DROP CONSTRAINT  ' +  NAME FROM sys.foreign_keys f1 \
                WHERE object_name(parent_object_id) IN (?, ?, ?, ?, ?, ?,
?, ?, ?, ?);"

        try:
            self.__db.cursor.execute(query_fk,
self.__tables['table_teams'],
                                     self.__tables['table_persons'],
                                     self.__tables['table_playerstats'],

self.__tables['table_playerstatsmeta'],
                                     self.__tables['table_teamstats'],
                                     self.__tables['table_teamstatsmeta'],
                                     self.__tables['table_positionsmeta'],
                                     self.__tables['table_venuesmeta'],
```

```
                                    self.__tables['table_matches'],
                                    self.__tables['table_teamslineup'])

            result_fk = self.__db.cursor.fetchall()

            for elem in result_fk:
                self.__db.cursor.execute(elem[0])

            self.__db.conn.commit()

        except Exception as e:
            print(f'{self.__info_message_start} Error while dropping FK:
{e}.')
            logging.exception(f'{self.__info_message_start} Error while
dropping FK: {e}.')
            return

        print(f'{self.__info_message_start} All FK were successfully
dropped!')
        logging.info(f'{self.__info_message_start} All FK were successfully
dropped!')

        # SQL query for dropping PK
        query_pk = "SELECT 'alter table ' + SCHEMA_NAME(Schema_id)+'.'+
object_name(parent_object_id) \
                + '  DROP CONSTRAINT  ' +  NAME FROM sys.key_constraints
f1 \
                WHERE object_name(parent_object_id) IN (?, ?, ?, ?, ?, ?,
?, ?, ?);"

        try:
            self.__db.cursor.execute(query_pk,
self.__tables['table_teams'],
                                    self.__tables['table_persons'],
                                    self.__tables['table_playerstats'],

self.__tables['table_playerstatsmeta'],
                                    self.__tables['table_teamstats'],
                                    self.__tables['table_teamstatsmeta'],
                                    self.__tables['table_positionsmeta'],
                                    self.__tables['table_venuesmeta'],
                                    self.__tables['table_matches'])

            result_pk = self.__db.cursor.fetchall()

            for elem in result_pk:
                self.__db.cursor.execute(elem[0])

            self.__db.conn.commit()

        except Exception as e:
            print(f'{self.__info_message_start} Error while dropping PK:
{e}.')
            logging.exception(f'{self.__info_message_start} Error while
dropping PK: {e}.')
            return

        print(f'{self.__info_message_start} All PK were successfully
dropped!')
        logging.info(f'{self.__info_message_start} All PK were successfully
dropped!')
```

```
    def players_stats_export(self, output='csv', mode='replace'):
        """
        This function takes string with attribute for players' stats and
exports data.
        Possible attributes: HomePlayerStatistics, AwayPlayerStatistics,
                             HomeKeeperStatistics, AwayKeeperStatistics

        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """
        players_stats_attributes = ['HomePlayerStatistics',
'HomeKeeperStatistics',
                                    'AwayPlayerStatistics',
'AwayKeeperStatistics']

        table = 'table_playerstats'

        if self.__check_output_and_mode(output, mode, table):
            try:
                df = pd.DataFrame(columns=['MatchID', 'PersonID',
'StatisticID', 'StatisticValue'])

                for attribute in players_stats_attributes:
                    get_data = self.match_stats[attribute]
                    for i in range(len(get_data)):
                        person_id = get_data[i]['PersonId']
                        for j in range(len(get_data[i]['Statistics'])):
                            statistic_id =
get_data[i]['Statistics'][j]['Statistic']
                            statistic_value =
get_data[i]['Statistics'][j]['Value']
                            df = df.append({'MatchID': self.match_id,
                                            'PersonID': person_id,
                                            'StatisticID': statistic_id,
                                            'StatisticValue':
statistic_value}, ignore_index=True)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
table <{self.__tables[table]}: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for table <{self.__tables[table]}: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def teams_stats_export(self, output='csv', mode='replace'):
        """
        This function takes string with attribute for teams' stats (Total,
1st half, 2nd half, over time) and exports data.
        Possible attributes: HomeTeamStats, AwayTeamStats
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
```

```
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """
        teams_stats_attributes = {'HomeTeamStats':
self.match_stats['HomeTeam']['Id'],
                                  'AwayTeamStats':
self.match_stats['AwayTeam']['Id']}

        table = 'table_teamstats'

        if self.__check_output_and_mode(output, mode, table):
            try:
                df = pd.DataFrame(columns=['MatchID', 'TeamID',
'StatisticID', 'Total', 'FirstHalf', 'SecondHalf',
                                           'FirstOverTime',
'SecondOverTime'])
                for attribute, team_id in teams_stats_attributes.items():
                    get_data = self.match_stats[attribute]

                    for i in range(len(get_data)):
                        statistic_id = get_data['Total'][i]['Statistic']
                        value_total = get_data['Total'][i]['Value']
                        value_first_half =
get_data['FirstHalf'][i]['Value']
                        value_second_half =
get_data['SecondHalf'][i]['Value']
                        value_first_over_time =
get_data['FirstOverTime'][i][
                            'Value'] if 'FirstOverTime' in get_data.keys()
else ''
                        value_second_over_time =
get_data['SecondOverTime'][i][
                            'Value'] if 'SecondOverTime' in get_data.keys()
else ''
                        df = df.append({'MatchID': self.match_id,
                                        'TeamID': team_id,
                                        'StatisticID': statistic_id,
                                        'Total': value_total,
                                        'FirstHalf': value_first_half,
                                        'SecondHalf': value_second_half,
                                        'FirstOverTime':
value_first_over_time if value_first_over_time else '',
                                        'SecondOverTime':
value_second_over_time if value_second_over_time else ''},
                                       ignore_index=True)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
table <{self.__tables[table]}: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for table <{self.__tables[table]}: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')
```

```
    def persons_export(self, team_id, output='csv', mode='replace'):
        """
        This function exports current players from provided team.
        :param team_id: ID of team to export the players from
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_persons'

        if self.__check_output_and_mode(output, mode, table):
            try:
                persons_data = requests.get(self.api_url +
f'api/selections/persons/{team_id}',
                                            headers={'Authorization':
self.auth})
                persons = json.loads(persons_data.text)

                df = pd.DataFrame(columns=['TeamID', 'PersonID',
'FirstName', 'SurNamePrefix', 'SurName',
                                           'ActiveSelection', 'NickName',
'NationalityCode',
                                           'DateOfBirth',
'DefaultPosition', 'Role', 'DefaultShirtNumber',
                                           'PreferredFoot', 'Height',
'Weight'])

                for person in persons:
                    df = df.append({'TeamID': team_id,
                                    'PersonID': person['Id'],
                                    'FirstName': person['FirstName'],
                                    'SurNamePrefix': person[
                                        'SurNamePrefix'] if 'SurNamePrefix'
in person.keys() else '',
                                    'SurName': person['SurName'],
                                    'ActiveSelection':
person['ActiveSelection'],
                                    'NickName': person['NickName'] if
'NickName' in person.keys() else '',
                                    'NationalityCode':
person['NationalityCode'],
                                    'DateOfBirth': person['DateOfBirth'],
                                    'DefaultPosition':
person['DefaultPosition'],
                                    'Role': person['Role'],
                                    'DefaultShirtNumber':
person['DefaultShirtNumber'],
                                    'PreferredFoot':
person['PreferredFoot'],
                                    'Height': person['Height'] if 'Height'
in person.keys() else '',
                                    # Some of attributes may miss values
                                    'Weight': person['Weight'] if 'Weight'
in person.keys() else ''}, ignore_index=True)

                # Exporting the results
                self.__export(df, table, output, mode)
```

```
            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
table <{self.__tables[table]}: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for table <{self.__tables[table]}: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def team_stats_meta_export(self, output='csv', mode='replace'):
        """
        This function exports meta information about teams' statistics
(e.g. statistic name).
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_teamstatsmeta'

        if self.__check_output_and_mode(output, mode, table):
            try:
                team_statistics_meta_api = requests.get(self.api_url +
'api/metadata/TeamStatistics/',

headers={'Authorization': self.auth})
                team_statistics_meta =
json.loads(team_statistics_meta_api.text)
                df = pd.DataFrame(team_statistics_meta)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute table <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def player_stats_meta_export(self, output='csv', mode='replace'):
        """
        This function exports meta information about players' statistics
(e.g. statistic name).
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_playerstatsmeta'
```

```
        if self.__check_output_and_mode(output, mode, table):
            try:
                player_statistics_meta_api = requests.get(self.api_url +
'api/metadata/PlayerStatistics/',

headers={'Authorization': self.auth})
                player_statistics_meta =
json.loads(player_statistics_meta_api.text)
                df = pd.DataFrame(player_statistics_meta)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute table <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def positions_meta_export(self, output='csv', mode='replace'):
        """
        This function exports meta information about positions (e.g.
position name).
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_positionsmeta'

        if self.__check_output_and_mode(output, mode, table):
            try:
                positions_meta_api = requests.get(self.api_url +
'api/metadata/positions/',
                                                  headers={'Authorization':
self.auth})
                positions_meta = json.loads(positions_meta_api.text)
                df = pd.DataFrame(positions_meta)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
table <{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for table <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')
```

```
    def venues_meta_export(self, output='csv', mode='replace'):
        """
        This function exports meta information about stadiums (e.g. stadium
name, city, country).
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_venuesmeta'

        if self.__check_output_and_mode(output, mode, table):
            try:
                venues_meta_api = requests.get(self.api_url +
'api/metadata/Venues/',
                                               headers={'Authorization':
self.auth})
                venues_meta = json.loads(venues_meta_api.text)
                df = pd.DataFrame(venues_meta)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
table <{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for table <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def match_info_export(self, output='csv', mode='replace'):
        """
        This function exports information about match (e.g. stadium name,
round, home team, away team etc.).
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_matches'

        if self.__check_output_and_mode(output, mode, table):
            try:
                df = pd.DataFrame(columns=['MatchID', 'HomeTeamID',
'AwayTeamID', 'DateTime',
                                           'Round', 'LastChanged',
                                           'MatchStatus', 'VenueId'])

                df = df.append({'MatchID': self.match_stats['Id'],
                                'HomeTeamID':
self.match_stats['HomeTeam']['Id'],
                                'AwayTeamID':
self.match_stats['AwayTeam']['Id'],
```

```
                                'DateTime': self.match_stats['DateTime'],
                                'Round': self.match_stats['Round'],
                                # 'AnalysisFinished':
self.match_stats['AnalysisFinished'],
                                'LastChanged':
self.match_stats['LastChanged'],
                                'MatchStatus': self.match_stats[
                                    'MatchStatus'] if 'MatchStatus' in
self.match_stats.keys() else '',
                                'VenueId': self.match_stats['VenueId']},
ignore_index=True)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
table <{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for table <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def teams_export(self, output='csv', mode='replace'):
        """
        This function exports information about teams.
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_teams'

        if self.__check_output_and_mode(output, mode, table):
            try:
                teams_api = requests.get(self.api_url +
'api/selections/all/', headers={'Authorization': self.auth})
                teams = json.loads(teams_api.text)
                df = pd.DataFrame(teams)
                df.drop('SelectionType', axis=1, inplace=True)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def export_team_lineups(self, team_id, output='csv', mode='append'):
```

```
        """
        This function exports information about teams' lineups.
        :param team_id: ID of team to export the players from
        :param output: Output format. Should be "csv", "sql" or "csv-sql".
        :param mode: SQL mode. Should be "append" or "replace".
                                        append - add data to previous
                                        replace - replace previous data
        :return: None
        """

        table = 'table_teamslineup'

        try:
            inspector = inspect(self.__db.engine)

            # Checking if Persons and Teams tables exist and TeamsLineUp
table doesn't
            check = all(
                [inspector.has_table(self.__tables['table_persons']),
inspector.has_table(self.__tables['table_teams']),
                 not inspector.has_table(self.__tables[table])])
        except Exception as e:
            print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
            logging.exception(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
            return

        if check:
            try:
                # Select PK and FK from tables Persons and Teams
                query_pk = "SELECT ccu.COLUMN_NAME \
                           FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS tc
\
                           JOIN INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE
AS ccu ON ccu.CONSTRAINT_NAME = tc.CONSTRAINT_NAME \
                           WHERE tc.TABLE_NAME in (?, ?) \
                           AND tc.CONSTRAINT_TYPE in ('PRIMARY KEY',
'FOREIGN KEY');"

                self.__db.cursor.execute(query_pk,
self.__tables['table_persons'], self.__tables['table_teams'])

                # Check if needed PK and FK are in a result
                checker_pk = {'DefaultPosition', 'PersonID',
'Id'}.issubset(
                    set([x[0] for x in self.__db.cursor.fetchall()]))  #
Checking if PK are assigned

                if checker_pk:
                    query = f"CREATE TABLE {self.__tables[table]}( \
                             TeamID int NOT NULL FOREIGN KEY REFERENCES
{self.__tables['table_teams']} (Id), \
                             PersonID int NOT NULL FOREIGN KEY REFERENCES
{self.__tables['table_persons']} (PersonId) \
                             );"

                    self.__db.cursor.execute(query)
                    self.__db.conn.commit()
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
```

```
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')
                return

        if self.__check_output_and_mode(output, mode, table):
            try:
                persons_data = requests.get(self.api_url +
f'api/selections/persons/{team_id}',
                                            headers={'Authorization':
self.auth})
                persons = json.loads(persons_data.text)

                df = pd.DataFrame(columns=['TeamID', 'PersonID'])

                for person in persons:
                    df = df.append({'TeamID': team_id,
                                    'PersonID': person['Id']},
ignore_index=True)

                # Exporting the results
                self.__export(df, table, output, mode)

            except KeyError as e:
                print(f'{self.__info_message_start} Wrong attribute for
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Wrong
attribute for <{self.__tables[table]}>: {e}.')
            except Exception as e:
                print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error for
table <{self.__tables[table]}>: {e}.')

    def export_all(self, team_id, output='csv', mode='append',
incl_meta=False):
        """
        This functions exports all available data at once.
        :param team_id: ID of Team to export information about.
        :param incl_meta: True if tables with meta information should be
exported.
        :param output: What format of output file is expected?
                               csv - result is only exported in CSV file
(default value)
                               sql - result is only exported in SQL database
                               csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                               append - add data to the previous rows (default
value)
                               replace - remove all previous data from table,
then add new
        :return: None
        """

        if output not in self.__possible_outputs or mode not in
self.__possible_modes:
            print(f'{self.__info_message_start} Wrong output format or SQL
mode!')
            logging.exception(f'{self.__info_message_start} Wrong output
```

```
format or SQL mode!')

        else:
            self.players_stats_export(output=output, mode=mode)
            self.teams_stats_export(output=output, mode=mode)
            self.persons_export(team_id=team_id, output=output, mode=mode)
            self.match_info_export(output=output, mode=mode)

            if incl_meta:
                self.team_stats_meta_export(output=output, mode=mode)
                self.player_stats_meta_export(output=output, mode=mode)
                self.positions_meta_export(output=output, mode=mode)
                self.venues_meta_export(output=output, mode=mode)
                self.teams_export(output=output, mode=mode)

    def clean_database(self, delete_tables=True):
        """
        This functions cleans the MS SQL Server database
        :param delete_tables: True if tables should be deleted as well as
its' content
        :return: None
        """
        self.drop_pk_and_fk()

        for table in self.__tables.values():
            if delete_tables:
                query = f"DROP TABLE {table}"
            else:
                query = f"DELETE FROM {table}"

            try:
                self.__db.cursor.execute(query)
                self.__db.conn.commit()

            except Exception as e:
                print(f'{self.__info_message_start} Error while deleting
table <{table}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error while
deleting table <{table}>: {e}.')
                continue

            print(f'{self.__info_message_start} Table <{table}> was
successfully deleted!')
            logging.info(f'{self.__info_message_start} Table <{table}> was
successfully deleted!')
```

coosto Script:

```
import pandas as pd
import requests as r
import json
import os
import logging
from datetime import datetime as dt


class CoostoAPI:
    """
    API for social media data provider (Coosto)
    """
```

```
    # Configuring file for logs
    logging.basicConfig(filename='app.log',
                        format='%(asctime)s - %(levelname)s - %(message)s',
                        level=logging.INFO,
                        datefmt='%d.%m.%Y %H:%M:%S')

    # Link to the main API
    api_url = 'https://in.coosto.com/api/1/'

    # Change table names (values in dict) only here. Table names further in
a script would change automatically
    __tables = {'table_projects': 'projects',
                'table_topics': 'trending_topics',
                'table_sources': 'sources',
                'table_sentiment': 'sentiment',
                'table_authors': 'authors'}

    # Defining possible options for output formats and SQL modes
    __possible_outputs = ['csv', 'sql', 'csv-sql']
    __possible_modes = ['append', 'replace']

    # All messages (info, logs, etc.) would start with
    __info_message_start = '[COOSTO]'

    def __init__(self, path, username, password, db_conn):
        """
        This function initializes the Coosto API connection.
        :param path: Path to folder where the results should be.
        :param username: Username for API
        :param password: Password for API
        :param db_conn: Database connection instance where the results
should be exported
        :return: None
        """

        self.path = path
        self.username = username
        self.password = password
        self.__db = db_conn

        # Setting credentials for API
        credentials = {'username': username, 'password': password}

        # Logging into API and getting session id
        try:
            self.login_api = r.get(self.api_url + 'users/login',
params=credentials, stream=True)
            if self.login_api.status_code == 200:
                self.__session_id =
json.loads(self.login_api.text)['data']['sessionid']
            else:
                print(f'{self.__info_message_start} Wrong credentials for
API!')
                logging.exception(f'{self.__info_message_start} Wrong
credentials for API!')

            print(f'{self.__info_message_start} Connection to the API was
established!')
            logging.info(f'{self.__info_message_start} Connection to the
API was established!')
```

```
        except Exception as e:
            print(f'{self.__info_message_start} Error while connecting to
the API: {e}.')
            logging.exception(f'{self.__info_message_start} Error while
connecting to the API: {e}.')

        # Path to results
        self.path_to_results = path + '/coosto_data/'
        if not os.path.exists(self.path_to_results):
            os.makedirs(self.path_to_results)

    def __export(self, df, table, output, mode):
        """
        This function takes a dataframe and then exports it to CSV file
or/and MS SQL Server.
        :param df: DataFrame to export (what to export).
        :param table: Table in SQL database to put the result in.
        :param output: Where to export:
                        csv - only in CSV file
                        sql - only in MS SQL Server Database
                        csv-sql - both CSV and database
        :param mode: How to insert data? Only applicable if output contains
sql
                        append - add data to the previous rows
                        replace - remove all previous data from table, then
add new
        :return: None
        """
        try:
            if output == 'csv':
                df.to_csv(self.path_to_results + self.__tables[table] +
'.csv')
            elif output == 'sql':
                df.to_sql(self.__tables[table], self.__db.engine,
if_exists=mode, index=False)
            elif output == 'csv-sql':
                df.to_csv(self.path_to_results + self.__tables[table] +
'.csv')
                df.to_sql(self.__tables[table], self.__db.engine,
if_exists=mode, index=False)

        except FileNotFoundError:
            print(f'{self.__info_message_start} Wrong path to the folder or
file: {self.path_to_results}.')
            logging.exception(f'{self.__info_message_start} Wrong path to
the folder or file: {self.path_to_results}.')
            return
        except Exception as e:
            print(f'{self.__info_message_start} Error while exporting table
<{self.__tables[table]}>: {e}.')
            logging.exception(f'{self.__info_message_start} Error while
exporting table <{self.__tables[table]}>: {e}.')
            return

        print(f'{self.__info_message_start} Table <{self.__tables[table]}>
was successfully exported to {output}!')
        logging.info(f'{self.__info_message_start} Table
<{self.__tables[table]}> was successfully exported to {output}!')

    def __check_output_and_mode(self, output, mode, table):
```

```
        """
        This function checks if provided output format and/or SQL mode are
in a list of available options.
        :param output: Provided output format. Should be "csv", "sql" or
"csv-sql".
        :param mode: Provided SQL mode. Should be "append" or "replace".
        :param table: Table in SQL database to put the result in.
        :return: True or False depending on the result of check.
        """

        if output not in self.__possible_outputs or mode not in
self.__possible_modes:
            print(
                f'{self.__info_message_start} Wrong output format or SQL
mode for table <{self.__tables[table]}>!')
            logging.exception(
                f'{self.__info_message_start} Wrong output format or SQL
mode for table <{self.__tables[table]}>!')
            return False
        else:
            return True

    def export_saved_queries(self, output='csv', mode='replace'):
        """
        This function exports projects (saved queries). Exported ID of
query is used later.
        :param output: What format of output file is expected?
                               csv - result is only exported in CSV file
(default value)
                               sql - result is only exported in SQL database
                               csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                               append - add data to the previous rows (default
value)
                               replace - remove all previous data from table,
then add new
        :return: None
        """

        table = 'table_projects'

        if self.__check_output_and_mode(output, mode, table):
            with self.login_api:
                try:
                    # Getting response
                    saved_queries_api = json.loads(
                        r.get(self.api_url + 'savedqueries/get_all/',
params={'sessionid': self.__session_id}).text)

                    saved_queries = {x['name']: x['id'] for x in
saved_queries_api['data']}

                    # Saving result to dataframe
                    projects_df = pd.DataFrame(list(saved_queries.items()),
columns=['Name', 'ID'])

                    # Exporting the results
                    self.__export(projects_df, table, output, mode)
```

```
                except KeyError:
                    print(f'{self.__info_message_start} Error while loading
<{self.__tables[table]}>: check API link or parameters!')
                    logging.exception(f'{self.__info_message_start} Error
while loading <{self.__tables[table]}>: check API link or parameters!')
                except Exception as e:
                    print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                    logging.exception(f'{self.__info_message_start} Error
for table <{self.__tables[table]}>: {e}.')

    def export_trending_topics(self, output='csv', mode='replace'):
        """
        This function exports trending topics.
        :param output: What format of output file is expected?
                            csv - result is only exported in CSV file
(default value)
                            sql - result is only exported in SQL database
                            csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                            append - add data to the previous rows (default
value)
                            replace - remove all previous data from table,
then add new
        :return: None
        """

        table = 'table_topics'

        if self.__check_output_and_mode(output, mode, table):
            with self.login_api:
                try:
                    # Getting response
                    trending_api = json.loads(
                        r.get(self.api_url + 'query/trending',
params={'sessionid': self.__session_id, 'qid': 131755}).text)

                    topics = [x['topic'] for x in trending_api['data'][0]]
                    scores = [x['score'] for x in trending_api['data'][0]]
                    trending_final_list = zip(topics, scores)

                    # Saving result to DataFrame
                    trending_df = pd.DataFrame(trending_final_list,
columns=['Topic', 'Score'])

                    # Exporting the results
                    self.__export(trending_df, table, output, mode)

                except KeyError:
                    print(f'{self.__info_message_start} Error while loading
<{self.__tables[table]}>: check API link or parameters!')
                    logging.exception(f'{self.__info_message_start} Error
while loading <{self.__tables[table]}>: check API link or parameters!')
                except Exception as e:
                    print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                    logging.exception(f'{self.__info_message_start} Error
for table <{self.__tables[table]}>: {e}.')
```

```
    def export_source_types(self, output='csv', mode='replace'):
        """
        This function exports source types (e.g. Twitter, blog, news etc.).
        :param output: What format of output file is expected?
                               csv - result is only exported in CSV file
(default value)
                               sql - result is only exported in SQL database
                               csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                               append - add data to the previous rows (default
value)
                               replace - remove all previous data from table,
then add new
        :return: None
        """

        table = 'table_sources'

        if self.__check_output_and_mode(output, mode, table):
            with self.login_api:
                try:
                    # Getting response
                    sources_api = json.loads(
                        r.get(self.api_url + 'query/sourcetypes',
params={'sessionid': self.__session_id, 'qid': 131755}).text)

                    names = [x['sourcetype'] for x in
sources_api['data'][0]]
                    freq = [x['freq'] for x in sources_api['data'][0]]
                    sent = [x['sent'] for x in sources_api['data'][0]]
                    sentp = [x['sentp'] for x in sources_api['data'][0]]
                    sentn = [x['sentn'] for x in sources_api['data'][0]]
                    sent0 = [x['sent0'] for x in sources_api['data'][0]]
                    source_types_final_list = zip(names, freq, sent, sentp,
sentn, sent0)

                    # Saving result to dataframe
                    source_types_df = pd.DataFrame(source_types_final_list,
                                                   columns=['Name',
'Frequency', 'Overall_sentiment', 'Positive_sentiment',

'Negative_sentiment', 'Neutral_sentiment'])

                    # Exporting the results
                    self.__export(source_types_df, table, output, mode)

                except KeyError:
                    print(f'{self.__info_message_start} Error while loading
<{self.__tables[table]}>: check API link or parameters!')
                    logging.exception(f'{self.__info_message_start} Error
while loading <{self.__tables[table]}>: check API link or parameters!')
                except Exception as e:
                    print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}')
                    logging.exception(f'{self.__info_message_start} Error
for table <{self.__tables[table]}>: {e}')

    def export_sentiment_per_day(self, output='csv', mode='append'):
        """
```

```
        This function exports sentiment analysis per day.
        :param output: What format of output file is expected?
                                csv - result is only exported in CSV file
(default value)
                                sql - result is only exported in SQL database
                                csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                                append - add data to the previous rows (default
value)
                                replace - remove all previous data from table,
then add new
        :return: None
        """

        table = 'table_sentiment'

        if self.__check_output_and_mode(output, mode, table):
            with self.login_api:
                try:
                    # Getting response
                    sentiment_api = json.loads(
                        r.get(self.api_url + 'query/days',
params={'sessionid': self.__session_id, 'qid': 131755}).text)

                    days = [dt.utcfromtimestamp(ts['time']).strftime('%Y-
%m-%d') for ts in sentiment_api['data'][0]]
                    freq = [x['freq'] for x in sentiment_api['data'][0]]
                    senttot = [x['sent'] for x in sentiment_api['data'][0]]
                    sentp = [x['sentp'] for x in sentiment_api['data'][0]]
                    sentn = [x['sentn'] for x in sentiment_api['data'][0]]
                    sent0 = [x['sent0'] for x in sentiment_api['data'][0]]
                    sent_final_list = zip(days, freq, senttot, sentp,
sentn, sent0)

                    # Saving result to dataframe
                    sent_df = pd.DataFrame(sent_final_list,
                                           columns=['Date', 'Frequency',
'Overall_sentiment', 'Positive_sentiment',
                                                    'Negative_sentiment',
'Neutral_sentiment'])

                    # Exporting the results
                    self.__export(sent_df, table, output, mode)

                except KeyError:
                    print(f'{self.__info_message_start} Error while loading
<{self.__tables[table]}>: check API link or parameters!')
                    logging.exception(f'{self.__info_message_start} Error
while loading <{self.__tables[table]}>: check API link or parameters!')
                except Exception as e:
                    print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                    logging.exception(f'{self.__info_message_start} Error
for table <{self.__tables[table]}>: {e}.')

    def export_authors(self, output='csv', mode='replace'):
        """
        This function exports most popular authors.
        :param output: What format of output file is expected?
```

```
                            csv - result is only exported in CSV file
(default value)
                            sql - result is only exported in SQL database
                            csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                            append - add data to the previous rows (default
value)
                            replace - remove all previous data from table,
then add new
        :return: None
        """

        table = 'table_authors'

        if self.__check_output_and_mode(output, mode, table):
            with self.login_api:
                try:
                    # Getting response
                    authors_api = json.loads(
                        r.get(self.api_url + 'query/authors',
params={'sessionid': self.__session_id, 'qid': 131755}).text)

                    authors = [x['author'] for x in authors_api['data'][0]]
                    freq = [x['freq'] for x in authors_api['data'][0]]
                    sent = [x['sent'] for x in authors_api['data'][0]]
                    sentp = [x['sentp'] for x in authors_api['data'][0]]
                    sentn = [x['sentn'] for x in authors_api['data'][0]]
                    sent0 = [x['sent0'] for x in authors_api['data'][0]]
                    influence = [x['influence'] for x in
authors_api['data'][0]]
                    gender = [x['gender'] for x in authors_api['data'][0]]
                    followers = [x['followers'] for x in
authors_api['data'][0]]
                    reactions = [x['reactions'] for x in
authors_api['data'][0]]
                    authors_final_list = zip(authors, freq, sent, sentp,
sentn, sent0,
                                             influence, gender, followers,
reactions)

                    # Saving result to dataframe
                    authors_df = pd.DataFrame(authors_final_list,
                                              columns=['Author', 'Freq',
'Sent', 'SentP', 'SentN', 'Sent0',
                                                       'Influence',
'Gender', 'Followers', 'Reactions'])

                    # Exporting the results
                    self.__export(authors_df, table, output, mode)

                except KeyError:
                    print(f'{self.__info_message_start} Error while loading
<{self.__tables[table]}>: check API link or parameters!')
                    logging.exception(f'{self.__info_message_start} Error
while loading <{self.__tables[table]}>: check API link or parameters!')
                except Exception as e:
                    print(f'{self.__info_message_start} Error for table
<{self.__tables[table]}>: {e}.')
                    logging.exception(f'{self.__info_message_start} Error
```

```
for table <{self.__tables[table]}>: {e}.')

    def export_all(self, output='csv', mode='replace'):
        """
        This functions exports all available data at once.
        :param output: What format of output file is expected?
                               csv - result is only exported in CSV file
(default value)
                               sql - result is only exported in SQL database
                               csv-sql - result is exported both in CSV file
and SQL database
        :param mode: Only used if output format contains SQL. How to insert
data in SQL database?
                               append - add data to the previous rows (default
value)
                               replace - remove all previous data from table,
then add new
        :return: None
        """

        if output not in self.__possible_outputs or mode not in
self.__possible_modes:
            print(f'{self.__info_message_start} Wrong output format or SQL
mode!')
            logging.exception(f'{self.__info_message_start} Wrong output
format or SQL mode!')

        else:
            self.export_saved_queries(output=output, mode=mode)
            self.export_trending_topics(output=output, mode=mode)
            self.export_source_types(output=output, mode=mode)
            self.export_sentiment_per_day(output=output, mode='append')
            self.export_authors(output=output, mode=mode)

    def clean_database(self, delete_tables=True):
        """
        This functions cleans the MS SQL Server database
        :param delete_tables: True if tables should be deleted as well as
its' content
        :return: None
        """

        for table in self.__tables.values():
            if delete_tables:
                query = f"DROP TABLE {table}"
            else:
                query = f"DELETE FROM {table}"

            try:
                self.__db.cursor.execute(query)
                self.__db.conn.commit()

            except Exception as e:
                print(f'{self.__info_message_start} Error while deleting
table <{table}>: {e}.')
                logging.exception(f'{self.__info_message_start} Error while
deleting table <{table}>: {e}.')
                continue

            print(f'{self.__info_message_start} Table <{table}> was
successfully deleted!')
```

```
            logging.info(f'{self.__info_message_start} Table <{table}> was
successfully deleted!')
```