spark as sparksession

spark.read.csv()

Self.spark_session = SparkSession.build()

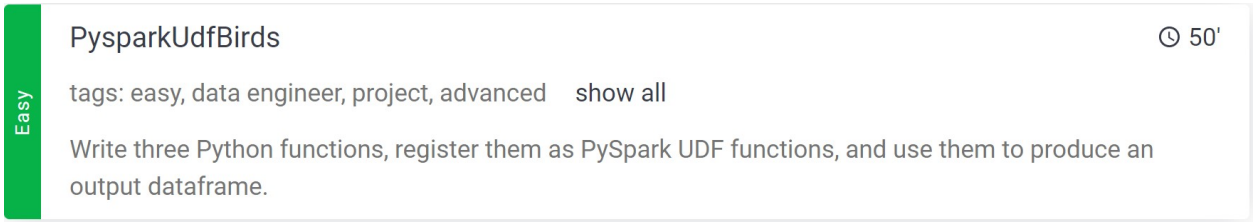
self.input _directory=”data”

---------- path definition

self.input_directory+”/direct/a.csv”

Dont change the Function name and prepare the return values as same as the arguments requirement.

Pyspark udf



To create udf in pyspark :

from pyspark.sql.functions import udf

 # create the function

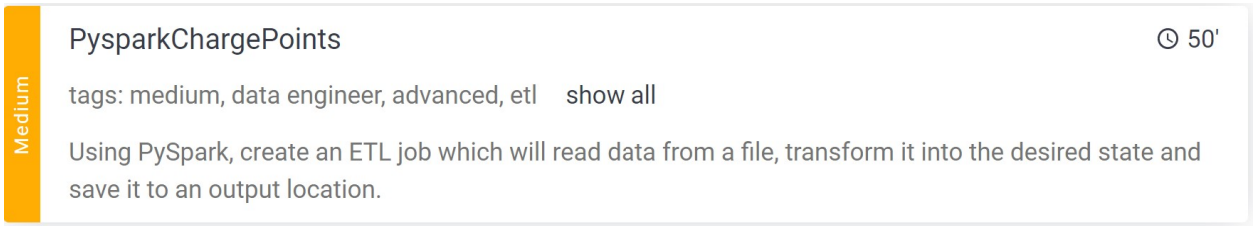
def add_twoval():

 # To register function as udf

Add_twoval_udf = udf(add_twoval, returntype())

 # Call udf in the process of dataframe :

Pyspark ETL



Extract : reading data from the given source directory

Create a function to create dataframes from the given source files in the input directory

Notes ; actual path , check column count , required column for the next task

Transform : create dataframe with best method available in spark
Decide the join condition based on the required output . return output
If possible apply sort

Run function :

Consolidation of all your functions (extract / transform).

==========

```
import sys
from operator import add
from pyspark.sql import SparkSession
from pyspark.sql.functions import lit
from functools import reduce
from pyspark.sql import DataFrame

class CouncilsJob:
    def __init__(self):

        self.spark=(SparkSession.builder
```

```
                                .master("local[*]")
                                .appName("EnglandCouncilsJob")
                                .getOrCreate())
        self.input_directory="file:///home/ak/Datasets/csv_files/"

    def extract_councils(self):

districtdf=self.spark.read.option("header","true").option("inferSchema","true").csv(self.input_dire
ctory+"england_councils/district_councils.csv")

londondf=self.spark.read.option("header","true").option("inferSchema","true").csv(self.input_dire
ctory+"england_councils/lo/ndon_boroughs.csv")

metropolitandf=self.spark.read.option("header","true").option("inferSchema","true").csv(self.input
_directory+"england_councils/metropolitan_districts.csv")

unitarydf=self.spark.read.option("header","true").option("inferSchema","true").csv(self.input_dire
ctory+"england_councils/unitary_authorities.csv")
        df1=districtdf.withColumn("council_type",lit('district councils'))
        df2=londondf.withColumn("council_type",lit('london boroughs'))
        df3=metropolitandf.withColumn("council_type",lit('metropolitan_districts'))
        df4=unitarydf.withColumn("council_type",lit('unitary_authorities'))
        dfs = [df1, df2, df3, df4]
        councils_df = reduce(DataFrame.unionAll, dfs)
        return councils_df


    def extract_avg_price(self):
        avg_price_df=
self.spark.read.option("header","true").option("inferSchema","true").csv(self.input_directory+"dat
a/property_avg_price.csv")
        return avg_price_df


    def extract_sales_volume(self):
        sales_volume_df=
self.spark.read.option("header","true").option("inferSchema","true").csv(self.input_directory+"dat
a/property_sales_volume.csv")
        return sales_volume_df


    def transform(self,councils_df,avg_price_df,sales_volume_df):

join1=councils_df.join(avg_price_df,councils_df.council==avg_price_df.local_authority,"left")
```

```
        df_complete1=join1.select("council","county","council_type","avg_price_nov_2019")

join2=df_complete1.join(sales_volume_df,df_complete1.council==sales_volume_df.local_author
ity,"left")

DataFrame=join2.select("council","county","council_type","avg_price_nov_2019","sales_volume
_sep_2019")
        return DataFrame

    def run(self):

final_dataframe=self.transform(self.extract_councils(),self.extract_avg_price(),self.extract_sales
_volume())
        return final_dataframe
```

==========Flight case study ============

Datasets :
Trips : origin , destination , internal_flight_id

Flights : **internal_flight_id**

- **public_flight_number**

```
trips = pd.DataFrame({

   "origin": [

       "PMI",

       "ATH",
```

```
        "JFK",
        "HND"
    ],
    "destination": [
        "OPO",
        "BCN",
        "MAD",
        "LAX"
    ],
    "internal_flight_ids": [
        [2, 1],
        [3],
```

```
        [5, 4, 6],

        [8, 9, 7, 0]

    ]

})
```

```
trips = spark.createDataFrame(trips)
```

```
flights = pd.DataFrame({
    "internal_flight_id": [
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    ],
    "public_flight_number": [
        "FR5763", "UT9586", "B4325", "RW35675", "LP656",
        "NB4321", "CX4599", "AZ8844", "KH8851", "OP8777"
    ]
})
flights = spark.createDataFrame(flights)
```

URL for reference :

https://github.com/tirthajyoti/Spark-with-Python

https://towardsdatascience.com/six-spark-exercises-to-rule-them-all-242445b24565

https://medium.com/bluekiri/check-your-pyspark-abilities-by-solving-this-quick-challenge-86f563a343dd

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as f
from pyspark.sql.types import DoubleType


class ChargePointsETLJob:
    input_path = 'data/input/electric-chargepoints-2017.csv'
    output_path = 'data/output/chargepoints-2017-analysis'

    def __init__(self):
        self.spark_session = (SparkSession.builder
                              .master("local[*]")
                              .appName("ElectricChargePointsETLJob")
                              .getOrCreate())

    def extract(self):
        return self.spark_session.read\
        .option('header', 'true')\
        .option('inferSchema', 'true')\
        .csv(self.input_path)

    def transform(self, df):
        return df.groupBy('CPID')\
                .agg(
                f.format_number(f.max(f.col('PluginDuration')),
2).cast(DoubleType()).alias('max_duration'),
                f.format_number(f.mean(f.col('PluginDuration')),
2).cast(DoubleType()).alias('avg_duration')
            ).withColumnRenamed('CPID', 'chargepoint_id')

    def load(self, df):
        df.write.parquet(self.output_path)

    def run(self):
        self.load(self.transform(self.extract()))
```

==========

Electric charge question

```
---------------------------------------------------------------------------------------
from pyspark.sql import SparkSession
from pyspark.sql import *
from pyspark.sql.functions import col
from pyspark.sql.functions import bround

class ChargePointsETLJob:
    input_path = 'data/input/electric-chargepoints-2017.csv'
    output_path = 'data/output/chargepoints-2017-analysis'

    def __init__(self):
        self.spark_session = (SparkSession.builder
                              .master("local[*]")
                              .appName("ElectricChargePointsETLJob")
                              .getOrCreate())

    def extract(self):

inputdf=self.spark_session.read.option("header","true").option("inferSchema","true").csv('data/in
put/electric-chargepoints-2017.csv')
        return inputdf

    def transform(self, df):
        new_df=df.select("CPID", "PluginDuration").withColumn("PluginDuration_duplicate",
df.PluginDuration)
        newer_df=new_df.groupBy("CPID").agg({"PluginDuration" : "max" ,
"PluginDuration_duplicate" : "avg" })

final=newer_df.withColumnRenamed("CPID","chargepoint_id").withColumnRenamed("max(Plug
inDuration)", "max_duration").withColumnRenamed("avg(PluginDuration_duplicate)",
"avg_duration")
        final2=final.select(round(col('max_duration',2)),round(col('avg_duration',2)))
        return final2

    def load(self, df):
        df.write.parquet('data/output/chargepoints-2017-analysis')

    def run(self):
        self.load(self.transform(self.extract()))
---------------------------------------------------------------------------------------------------------------
```

```
from pyspark.sql.functions import *



diff_secs_col = col("endtime").cast("long") - col("starttime").cast("long")

df2 = df1.withColumn( "diff_mins", diff_secs_col/ 60 )






new_df=df.select("CPID", "PluginDuration").withColumn("PluginDuration_duplicate",
df.PluginDuration)
      newer_df=new_df.groupBy("CPID").agg({"PluginDuration" : "max" ,
"PluginDuration_duplicate" : "avg" })

final=newer_df.withColumnRenamed("CPID","chargepoint_id").withColumnRenamed("max(Plug
inDuration)", "max_duration").withColumnRenamed("avg(PluginDuration_duplicate)",
"avg_duration")

      return final

df.write.parquet('data/output/chargepoints-2017-analysis')

================================================================================
==============

for birds



def get_start_year(period):

   split_col = pyspark.sql.functions.split(period['Period'], '-')
   year = period.withColumn('collected_from_year', split_col.getItem(0))
   year = year.select("collected_from_year")
   return year

---------------------- udf.py
---------------------- udf.py
```

```
from pyspark.sql.functions import *
from pyspark.sql.types import *

def get_english_name(species):
    return species.split('(')[0].rstrip()


def get_start_year(period):
    start_year = period.split('-')[0]
    start_year = start_year[1:]
    syear = int(start_year)
    return syear


def get_trend(annual_percentage_change):
    if annual_percentage_change <= -3.00:
        return 'strong decline'
    elif annual_percentage_change > -3.00 and annual_percentage_change <= -0.50:
        return 'weak decline'
    elif annual_percentage_change > -0.50 and annual_percentage_change < 0.50:
        return 'no change'
    elif annual_percentage_change >= 0.50 and annual_percentage_change <= 3.00:
        return 'weak increase'
    elif annual_percentage_change > 3.00:
        return 'strong increase'

get_english_name = udf(get_english_name,StringType())
get_start_year = udf(get_start_year,IntegerType())
get_trend = udf(get_trend,StringType())


----------job.py
from pyspark.sql.functions import *

def transform(self, df):
        mod_df =
df.withColumn("Species",get_english_name(df.Species)).withColumn("Period",get_start_year(df
.Period)).withColumn("trend",get_trend(col("Annual Percentage
Change"))).withColumnRenamed("Period","collected_from_year").withColumnRenamed("Specie
s","species").withColumnRenamed("Annual Percentage
Change","annual_percentage_change").withColumnRenamed("Category","category")

        return mod_df
```

```
================================================england councils
================================================================================
====================
import sys
from operator import add
from pyspark.sql import SparkSession
from pyspark.sql.functions import lit
from functools import reduce
from pyspark.sql import DataFrame

class CouncilsJob:
        def __init__(self):
                self.spark_session =
(SparkSession.builder.master("local[*]").appName("EnglandCouncilsJob").getOrCreate())
                self.input_directory = "data"

        def extract_councils(self):
                district_councils = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/district_councils.csv")
                london_boroughs = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/london_boroughs.csv")
```

```
                metropolitan_districts = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/metropolitan_districts.csv")
                unitary_authorities = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/unitary_authorities.csv")

                district_councils = district_councils.withColumn("council_type", lit("District
Councils"))
                london_boroughs = london_boroughs.withColumn("council_type", lit("London
Boroughs"))
                metropolitan_districts = metropolitan_districts.withColumn("council_type",
lit("Metropolitan Districts"))
                unitary_authorities = unitary_authorities.withColumn("council_type", lit("Unitary
Authorities"))

                councils_df = district_councils.union(london_boroughs)
                councils_df = councils_df.union(metropolitan_districts)
                councils_df = councils_df.union(unitary_authorities)

                return councils_df

        def extract_avg_price(self):
                property_avg_value = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory + "/property_avg_price.csv")
                avg_prices_df = property_avg_value.select(col("local_authority").alias("council"),
col("avg_price_nov_2019"))
                return avg_prices_df

        def extract_sales_volume(self):
                property_sales_volume = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory + "/property_sales_volume.csv")
                sales_volumes_df =
property_sales_volume.select(col("local_authority").alias("council"),
col("sales_volume_sep_2019"))
                return sales_volumes_df

        def transform(self, councils_df, avg_prices_df, sales_volumes_df):
                temp_join_df = councils.join(avg_prices_df, on=['council'], how='left')
                transformed_df = temp_join_df.join(sales_volumes_df, on=['council'], how='left')
                return transformed_df

        def run(self):
```

```
                return self.transform(self.extract_councils(), self.extract_avg_price(),
self.extract_sales_volume())

================================


from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.functions import round
from pyspark.sql.types import *


class ChargePointsETLJob:
    input_path = 'data/input/electric-chargepoints-2017.csv'
    output_path = 'data/output/chargepoints-2017-analysis'

    def __init__(self):
        self.spark_session = (SparkSession.builder
                              .master("local[*]")
                              .appName("ElectricChargePointsETLJob")
                              .getOrCreate())

    def extract(self):
        df =
self.spark_session.read.option("header","true").option("inferSchema","true").csv(self.input_path)
        return df
    def transform(self, df):
        df = df.groupBy("CPID").agg(max("PluginDuration").alias("max_duration"),
mean("PluginDuration").alias("avg_duration"))
        df = df.select( col("CPID").alias("chargepoint_id"),
round("max_duration",2).alias("max_duration"), round("avg_duration",2).alias("avg_duration"))



        return df

    def load(self, df):
        df.write.parquet(self.output_path)
        return df

    def run(self):
        self.load(self.transform(self.extract()))
================================================
```

Final solutions for birds and electric

```
from pyspark.sql.functions import *
from pyspark.sql.types import *

def get_english_name(species):
    return species.split('(')[0].rstrip()

def get_start_year(period):
    return int(period.split("-")[0][1:])

def get_trend(annual_percentage_change):
    if annual_percentage_change <= -3.00:
        return 'strong decline'
    elif annual_percentage_change > -3.00 and annual_percentage_change <= -0.50:
        return 'weak decline'
    elif annual_percentage_change > -0.50 and annual_percentage_change < 0.50:
        return 'no change'
    elif annual_percentage_change >= 0.50 and annual_percentage_change <= 3.00:
        return 'weak increase'
    elif annual_percentage_change > 3.00:
        return 'strong increase'

get_english_name = udf(get_english_name,StringType())
get_start_year = udf(get_start_year,IntegerType())
get_trend = udf(get_trend,StringType())

          df = df.withColumn("Species", get_english_name(df.Species)).withColumn("Period",
get_start_year(df.Period)).withColumn("trend", get_trend(col("Annual percentage
change"))).withColumnRenamed("Species", "species").withColumnRenamed("Category",
"category").withColumnRenamed("Period",
"collected_from_year").withColumnRenamed("Annual percentage change",
"annual_percentage_change")
       return df
```

Electric charge question

```
from pyspark.sql import SparkSession
from pyspark.sql import *
from pyspark.sql.functions import col
from pyspark.sql.functions import bround
```

```
from pyspark.sql.types import DoubleType
import pyspark.sql.functions as f

class ChargePointsETLJob:
    input_path = 'data/input/electric-chargepoints-2017.csv'
    output_path = 'data/output/chargepoints-2017-analysis'

    def __init__(self):
        self.spark_session = (SparkSession.builder
                              .master("local[*]")
                              .appName("ElectricChargePointsETLJob")
                              .getOrCreate())

    def extract(self):

inputdf=self.spark_session.read.option("header","true").option("inferSchema","true").csv('data/in
put/electric-chargepoints-2017.csv')
        return inputdf

    def transform(self, df):
       return df.groupBy('CPID')\
                 .agg(
                 f.format_number(f.max(f.col('PluginDuration')),
2).cast(DoubleType()).alias('max_duration'),
                 f.format_number(f.mean(f.col('PluginDuration')),
2).cast(DoubleType()).alias('avg_duration')
              ).withColumnRenamed('CPID', 'chargepoint_id')

    def load(self, df):
        df.write.parquet('data/output/chargepoints-2017-analysis')

    def run(self):
-----------------------------------------------------------------------------------------------------------
from pyspark.sql.functions import *



diff_secs_col = col("endtime").cast("long") - col("starttime").cast("long")

df2 = df1.withColumn( "diff_mins", diff_secs_col/ 60 )
```

```
new_df=df.select("CPID", "PluginDuration").withColumn("PluginDuration_duplicate",
df.PluginDuration)
        newer_df=new_df.groupBy("CPID").agg({"PluginDuration" : "max" ,
"PluginDuration_duplicate" : "avg" })

final=newer_df.withColumnRenamed("CPID","chargepoint_id").withColumnRenamed("max(Plug
inDuration)", "max_duration").withColumnRenamed("avg(PluginDuration_duplicate)",
"avg_duration")

        return final

df.write.parquet('data/output/chargepoints-2017-analysis')

==========================================================================
==========================================================================
==============


for birds

---------------------- udf.py
from pyspark.sql.functions import *
from pyspark.sql.types import *

def get_english_name(species):
    return species.split('(')[0].rstrip()


def get_start_year(period):
    start_year = period.split('-')[0]
    start_year = start_year[1:]
    syear = int(start_year)
    return syear


def get_trend(annual_percentage_change):
    if annual_percentage_change <= -3.00:
        return 'strong decline'
    elif annual_percentage_change > -3.00 and annual_percentage_change <= -0.50:
        return 'weak decline'
    elif annual_percentage_change > -0.50 and annual_percentage_change < 0.50:
        return 'no change'
```

```
    elif annual_percentage_change >= 0.50 and annual_percentage_change <= 3.00:
        return 'weak increase'
    elif annual_percentage_change > 3.00:
        return 'strong increase'

get_english_name = udf(get_english_name,StringType())
get_start_year = udf(get_start_year,IntegerType())
get_trend = udf(get_trend,StringType())


----------job.py
from pyspark.sql.functions import *

def transform(self, df):
        mod_df =
df.withColumn("Species",get_english_name(df.Species)).withColumn("Period",get_start_year(df
.Period)).withColumn("trend",get_trend(col("Annual Percentage
Change"))).withColumnRenamed("Period","collected_from_year").withColumnRenamed("Specie
s","species").withColumnRenamed("Annual Percentage
Change","annual_percentage_change").withColumnRenamed("Category","category")

        return mod_df
```

```
==============================================england councils
================================================================================
====================
import sys
from operator import add
from pyspark.sql import SparkSession
from pyspark.sql.functions import lit
from functools import reduce
from pyspark.sql import DataFrame

class CouncilsJob:
        def __init__(self):
                self.spark_session =
(SparkSession.builder.master("local[*]").appName("EnglandCouncilsJob").getOrCreate())
                self.input_directory = "data"

        def extract_councils(self):
                district_councils = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/district_councils.csv")
                london_boroughs = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/london_boroughs.csv")
                metropolitan_districts = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/metropolitan_districts.csv")
                unitary_authorities = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory +
"/england_councils/unitary_authorities.csv")

                district_councils = district_councils.withColumn("council_type", lit("District
Councils"))
                london_boroughs = london_boroughs.withColumn("council_type", lit("London
Boroughs"))
                metropolitan_districts = metropolitan_districts.withColumn("council_type",
lit("Metropolitan Districts"))
                unitary_authorities = unitary_authorities.withColumn("council_type", lit("Unitary
Authorities"))

                councils_df = district_councils.union(london_boroughs)
                councils_df = councils_df.union(metropolitan_districts)
                councils_df = councils_df.union(unitary_authorities)

                return councils_df
```

```
        def extract_avg_price(self):
                property_avg_value = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory + "/property_avg_price.csv")
                avg_prices_df = property_avg_value.select(col("local_authority").alias("council"),
col("avg_price_nov_2019"))
                return avg_prices_df

        def extract_sales_volume(self):
                property_sales_volume = self.spark_session.read.option("inferSchema",
"true").option("header", "true").csv(self.input_directory + "/property_sales_volume.csv")
                sales_volumes_df =
property_sales_volume.select(col("local_authority").alias("council"),
col("sales_volume_sep_2019"))
                return sales_volumes_df

        def transform(self, councils_df, avg_prices_df, sales_volumes_df):
                temp_join_df = councils.join(avg_prices_df, on=['council'], how='left')
                transformed_df = temp_join_df.join(sales_volumes_df, on=['council'], how='left')
                return transformed_df

        def run(self):
                return self.transform(self.extract_councils(), self.extract_avg_price(),
self.extract_sales_volume())
```

==========

Birds ETL

```
//job.py
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType
from solution.udfs import get_english_name, get_start_year, get_trend
from pyspark.sql.functions import *


class BirdsETLJob:
   input_path = 'data/input/birds.csv'

   def __init__(self):
      self.spark_session = (SparkSession.builder
                            .master("local[1]")
                            .appName("BirdsETLJob")
```

```
                                .getOrCreate())

    def extract(self):
        input_schema = StructType([StructField("Species", StringType()),
                                   StructField("Category", StringType()),
                                   StructField("Period", StringType()),
                                   StructField("Annual percentage change", DoubleType())
                                   ])
        df = self.spark_session.read.option('header', 'true').option('inferSchema',
'true').schema(input_schema).csv(self.input_path)
        return df

    def transform(self, df):
        df = df.withColumn("Species", get_english_name(df.Species)).withColumn("Period",
get_start_year(df.Period)).withColumn("trend", get_trend(col("Annual percentage
change"))).withColumnRenamed("Species", "species").withColumnRenamed("Category",
"category").withColumnRenamed("Period",
"collected_from_year").withColumnRenamed("Annual percentage change",
"annual_percentage_change")
        return df
        pass

    def run(self):
        return self.transform(self.extract())

//udfs.py

from pyspark.sql.functions import *
from pyspark.sql.types import *

def get_english_name(species):
    return species.split('(')[0].rstrip()
    pass


def get_start_year(period):
    return int(period.split("-")[0][1:])
    pass


def get_trend(annual_percentage_change):
    if annual_percentage_change < -3.0:
        return "strong decline"
    elif annual_percentage_change >= -3.0 and annual_percentage_change <= -0.5:
```

```
        return "weak decline"
    elif annual_percentage_change > -0.5 and annual_percentage_change < 0.5:
        return "no change"
    elif annual_percentage_change >= 0.5 and annual_percentage_change <= 3.0:
        return "weak increase"
    elif annual_percentage_change > 3.0:
        return "strong increase"
    pass

get_english_name = udf(get_english_name,StringType())
get_start_year = udf(get_start_year,IntegerType())
get_trend = udf(get_trend,StringType())
================================================================================
================================
from pyspark.sql import SparkSession
import sys
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql import *

class ChargePointsETLJob:
    input_path = 'data/input/electric-chargepoints-2017.csv'
    output_path = 'data/output/chargepoints-2017-analysis'

    def __init__(self):
        self.spark_session = (SparkSession.builder
                              .master("local[*]")
                              .appName("ElectricChargePointsETLJob")
                              .getOrCreate())

    def extract(self):
        df = self.spark_session.read.option("header", "true").option("inferSchema",
"true").csv(self.input_path)
        return df
        #pass

    def transform(self, df):
        #df = df.groupBy(df.CPID).agg(round(max("PluginDuration"),2).alias("max_duration"),
round(avg("PluginDuration"),2).alias("avg_duration"))

        df = df.groupBy("CPID").agg(max("PluginDuration").alias("max_duration"),
mean("PluginDuration").alias("avg_duration"))
        df = df.select( col("CPID").alias("chargepoint_id"),
round("max_duration",2).alias("max_duration"), round("avg_duration",2).alias("avg_duration"))
```

```
        return df

        #pass

    def load(self, df):
        df.write.parquet(self.output_path)
        #pass

    def run(self):
        self.load(self.transform(self.extract()))
```

---

charge:

```
import sys
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql import *
from pyspark.sql import SparkSession
import pyspark.sql.functions as f
from pyspark.sql.types import DoubleType


class ChargePointsETLJob:
    input_path = 'data/input/electric-chargepoints-2017.csv'
    output_path = 'data/output/chargepoints-2017-analysis'

    def __init__(self):
        self.spark_session = (SparkSession.builder
                              .master("local[*]")
                              .appName("ElectricChargePointsETLJob")
                              .getOrCreate())

    def extract(self):
        df = self.spark_session.read.option("header", "true").option("inferSchema",
"true").csv(self.input_path)
        return df
        pass


    def transform(self, df):
```

```
        return df.groupBy('CPID').agg(f.format_number(f.max(f.col('PluginDuration')),
2).cast(DoubleType()).alias('max_duration'),f.format_number(f.mean(f.col('PluginDuration')),
2).cast(DoubleType()).alias('avg_duration')).withColumnRenamed('CPID', 'chargepoint_id')


    def load(self, df):
        df.write.parquet(self.output_path)
        pass

    def run(self):
        self.load(self.transform(self.extract()))
```