Here is the Solution for Exercise 4: Building a Full Pipeline

This is the solution code for exercise4.py

```
import datetime

from airflow import DAG

from airflow.operators import (
    FactsCalculatorOperator,
    HasRowsOperator,
    S3ToRedshiftOperator
)

#
# The following DAG performs the following functions:
#
#       1. Loads Trip data from S3 to RedShift
#       2. Performs a data quality check on the Trips table in RedShift
#       3. Uses the FactsCalculatorOperator to create a Facts table in Redshift
#           a. **NOTE**: to complete this step you must complete the FactsCalcuatorOper
#              skeleton defined in plugins/operators/facts_calculator.py
#
dag = DAG("lesson3.exercise4", start_date=datetime.datetime.utcnow())

#
# The following code will load trips data from S3 to RedShift. Use the s3_key
#       "data-pipelines/divvy/unpartitioned/divvy_trips_2018.csv"
#       and the s3_bucket "udacity-dend"
#
copy_trips_task = S3ToRedshiftOperator(
    task_id="load_trips_from_s3_to_redshift",
    dag=dag,
    table="trips",
    redshift_conn_id="redshift",
    aws_credentials_id="aws_credentials",
    s3_bucket="udacity-dend",
    s3_key="data-pipelines/divvy/unpartitioned/divvy_trips_2018.csv"
)

#
#  Data quality check on the Trips table
#
check_trips = HasRowsOperator(
    task_id="check_trips_data",
    dag=dag,
    redshift_conn_id="redshift",
    table="trips"
)

#
# We use the FactsCalculatorOperator to create a Facts table in RedShift. The fact colu
#  `tripduration` and the groupby_column is `bikeid`
#
calculate_facts = FactsCalculatorOperator(
    task_id="calculate_facts_trips",
    dag=dag,
    redshift_conn_id="redshift"
```

```
    redshift_conn_id="redshift",
    origin_table="trips",
    destination_table="trips_facts",
    fact_column="tripduration",
    groupby_column="bikeid"
)

#
# Task ordering for the DAG tasks
#
copy_trips_task >> check_trips
check_trips >> calculate_facts
```

This is the solution code for the Custom Operator: `facts_calculator`

```
import logging

from airflow.hooks.postgres_hook import PostgresHook
from airflow.models import BaseOperator
from airflow.utils.decorators import apply_defaults


class FactsCalculatorOperator(BaseOperator):
    facts_sql_template = """
    DROP TABLE IF EXISTS {destination_table};
    CREATE TABLE {destination_table} AS
    SELECT
        {groupby_column},
        MAX({fact_column}) AS max_{fact_column},
        MIN({fact_column}) AS min_{fact_column},
        AVG({fact_column}) AS average_{fact_column}
    FROM {origin_table}
    GROUP BY {groupby_column};
    """

    @apply_defaults
    def __init__(self,
                 redshift_conn_id="",
                 origin_table="",
                 destination_table="",
                 fact_column="",
                 groupby_column="",
                 *args, **kwargs):

        super(FactsCalculatorOperator, self).__init__(*args, **kwargs)
        self.redshift_conn_id = redshift_conn_id
        self.origin_table = origin_table
        self.destination_table = destination_table
        self.fact_column = fact_column
        self.groupby_column = groupby_column

    def execute(self, context):
        redshift = PostgresHook(postgres_conn_id=self.redshift_conn_id)
        facts_sql = FactsCalculatorOperator.facts_sql_template.format(
            origin_table=self.origin_table,
            destination_table=self.destination_table,
            fact_column=self.fact_column,
            groupby_column=self.groupby_column
        )
        redshift.run(facts_sql)
```