Network Operating System

Developing market-worthy models using cloud developement

Paweł Pozorski, Zuzanna Sieńko

2024-12-23

Abstract

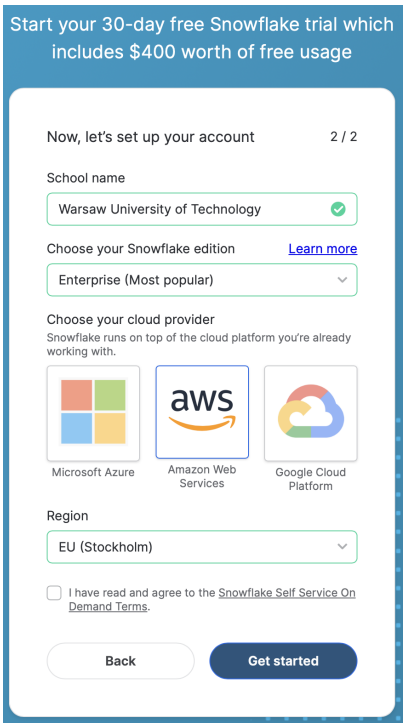
In following document we share our route for developement of the project infrastructure.

Contents

1 Deploying model using Snowpark, python and VS Code

1.1 Creating trial Snowflake Account

For purpose of the project we'll use free, trial Snowflake account that is avaliable for public for up to 30 days / 400 USD of credits. As our cloud provider, we'll choose AWS as it was introduced on the labs.



Registration form.

Using snowsight, we create basic setup for snowflake:

```
CREATE WAREHOUSE IF NOT EXISTS ml_warehouse
  WITH WAREHOUSE_SIZE = 'XSMALL'
  AUTO_SUSPEND = 60  -- Automatically suspends after 60 seconds of inactivity
  AUTO_RESUME = TRUE;

CREATE ROLE IF NOT EXISTS DATA_SCIENTIST;
GRANT ROLE DATA_SCIENTIST TO ROLE SYSADMIN;
GRANT ALL PRIVILEGES ON WAREHOUSE ml_warehouse TO ROLE DATA_SCIENTIST;

CREATE DATABASE ml_database;
CREATE SCHEMA datasets;

GRANT OWNERSHIP ON DATABASE ml_database TO ROLE DATA_SCIENTIST;
GRANT OWNERSHIP ON SCHEMA datasets TO ROLE DATA_SCIENTIST;
```

Now let's try to access it from snowpark. For installation we've followed these instructions[1]. After providing required connection details described here[2], we create simple code to validate if it works correctly:

```
from snowflake.snowpark import Session
from snowflake.ml.utils.connection_params import SnowflakeLoginOptions
```

[1] https://docs.snowflake.com/en/developer-guide/snowpark/python/setup
[2] https://docs.snowflake.com/en/developer-guide/snowflake-cli/connecting/configure-connections

```
options = SnowflakeLoginOptions(login_file=".snowsql/config", connection_name="ml")

sp_session = Session.builder.configs(options).create()

sp_session.sql("DESCRIBE DATABASE ml_database;").collect()
```

Recieving:

```
[Row(created_on=datetime.datetime(2024, 12, 24, 13, 6, 41, 881000, tzinfo=<DstTzInfo 'America/Los_Angel
 Row(created_on=datetime.datetime(2024, 12, 24, 13, 34, 5, 265000, tzinfo=<DstTzInfo 'America/Los_Angel
```

1.2 Loading data

Here we've downloaded our dataset of choice ([source](https://archive.ics.uci.edu/static/public/9/auto+mpg.zip “)), unzipped it and made some adjustments to its structure (those normally wouldn't be neccessary, but this dataset is very old and it does not follow normal csv encoding standards, therefore we've used slow regex convertion to fix them). Next steps were again cloud-related:

- create stage on snowflake - for those unfamilliar with that technology, stage is place on our cloud where we can put our files. In our case, it will use azure cloud file system to accomplish that.
- put the file into the stage
- scheck if it is present there

name	size	md5	last_modified
auto_mpg_stage/auto-mpg.data	30288	b26b22a6...	Sat, 4 Jan 2025 15:01:04 GMT

  Here we can also see that snowflake automatically stores some metadata on our file - its modification date, size and md5 hash. Thats usefull for production usage because whenever we overwrite such file and try to load it into table again, snowflake will check if it's hash has changed. If not, it will skip data loading to avoid repetitions (providing we won't force load it)

- create sql table to store data from our file - below is final table structure (after some transformations described later)

  ```
  root
  |-- "MPG": DoubleType() (nullable = True)
  |-- "CYLINDERS": LongType() (nullable = True)
  |-- "DISPLACEMENT": DoubleType() (nullable = True)
  |-- "HORSEPOWER": DoubleType() (nullable = True)
  |-- "WEIGHT": DoubleType() (nullable = True)
  |-- "ACCELERATION": DoubleType() (nullable = True)
  |-- "MODEL_YEAR": LongType() (nullable = True)
  |-- "ORIGIN": LongType() (nullable = True)
  ```

- create file format - or those unfamilliar with that technology, file format is abstract object that tells snowsql how to load data - here we can define its type, endoding, how to treat null values and many more.
- load data into table

file	status	rows_parsed	rows_loaded	error_limit
auto_mpg_stage/cleaned-auto-mpg.data	LOADED	398	398	1

errors_seen	first_error	first_error_line	first_error_character	first_error_column_name
0	NULL	NULL	NULL	NULL

Here we can also debug our file format - if any error occurs, we will be able to address that. For production usage we can also set in our file format how many errors are acceptable and more.

1.3 EDA

Before we bagin, this part aims to show power of cloud resources not to be a compleate data science project. Therefore, below we show some basic calculations performed in snowpark. Each havy-calculation part is performed natively on cloud, so this code can be used on very large datasets as well.

Let's take peak at our data:

COLUMN	ROW_1	ROW_2	ROW_3
MPG	18.0	15.0	18.0
CYLINDERS	8	8	8
DISPLACEMENT	307.0	350.0	318.0
HORSEPOWER	130.0	165.0	150.0
WEIGHT	3504.0	3693.0	3436.0
ACCELERATION	12.0	11.5	11.0
MODEL_YEAR	70	70	70
ORIGIN	1	1	1
CAR_NAME	chevrolet,chevelle,malibu	buick,skylark,320	plymouth,satellite

We've got 398 rows. Car name will be irrelevant for our target - MPG, so we drop it. Let's see their statistics:

COLUMN	count	mean	stddev	min	max
MPG	398.0	23.51	7.82	9.0	46.6
CYLINDERS	398.0	5.45	1.7	3.0	8.0
DISPLACEMENT	398.0	193.43	104.27	68.0	455.0
HORSEPOWER	392.0	104.47	38.49	46.0	230.0
WEIGHT	398.0	2970.42	846.84	1613.0	5140.0
ACCELERATION	398.0	15.57	2.76	8.0	24.8
MODEL_YEAR	398.0	76.01	3.7	70.0	82.0
ORIGIN	398.0	1.57	0.8	1.0	3.0

Which columns are categorical?

COL_NAME	NO_DISTINCT	VAL_DISTINCT
MPG	129	18, 15, 16, 17, 14
CYLINDERS	5	8, 4, 6, 3, 5
DISPLACEMENT	82	307, 350, 304, 302, 429
HORSEPOWER	93	130, 165, 150, 198, 220
WEIGHT	351	3504, 3693, 3436, 3433, 3449
ACCELERATION	95	12, 11.5, 11, 10.5, 10
MODEL_YEAR	13	70, 71, 72, 73, 74
ORIGIN	3	1, 3, 2
CAR_NAME	305	chevrolet,chevelle,malibu, buick,skylark

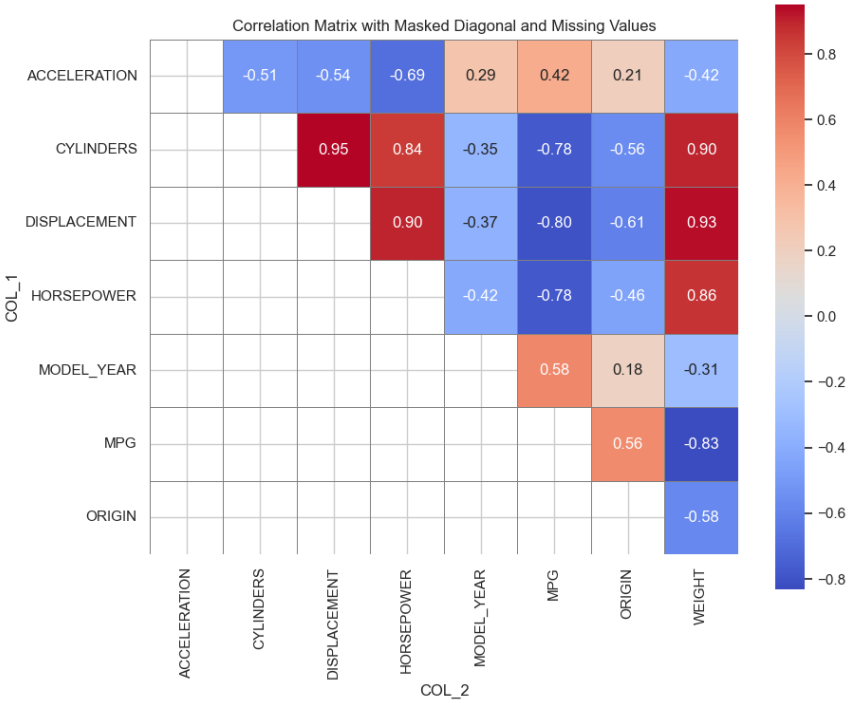


Figure 1: Correlation matrix.

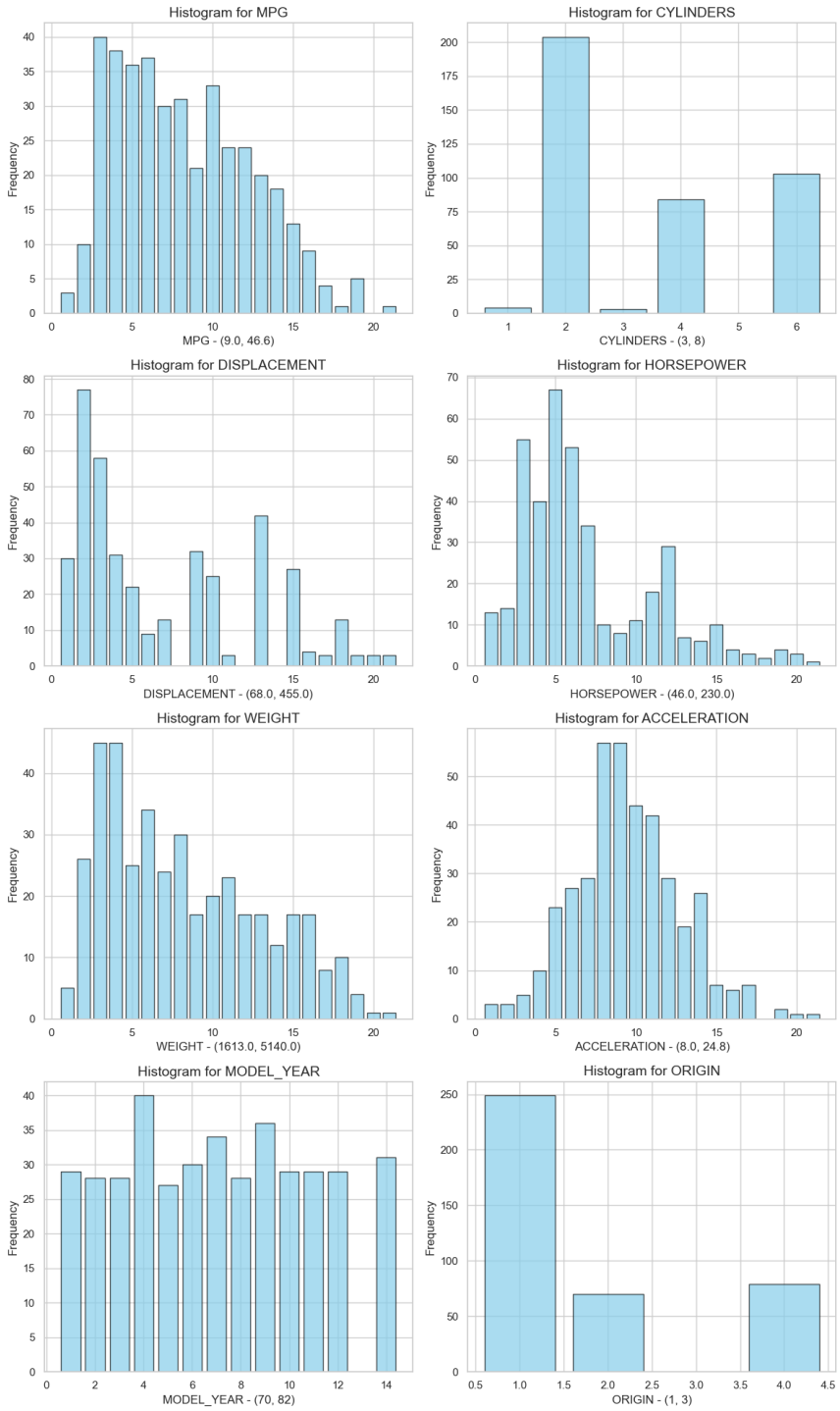


Figure 2: Histograms.