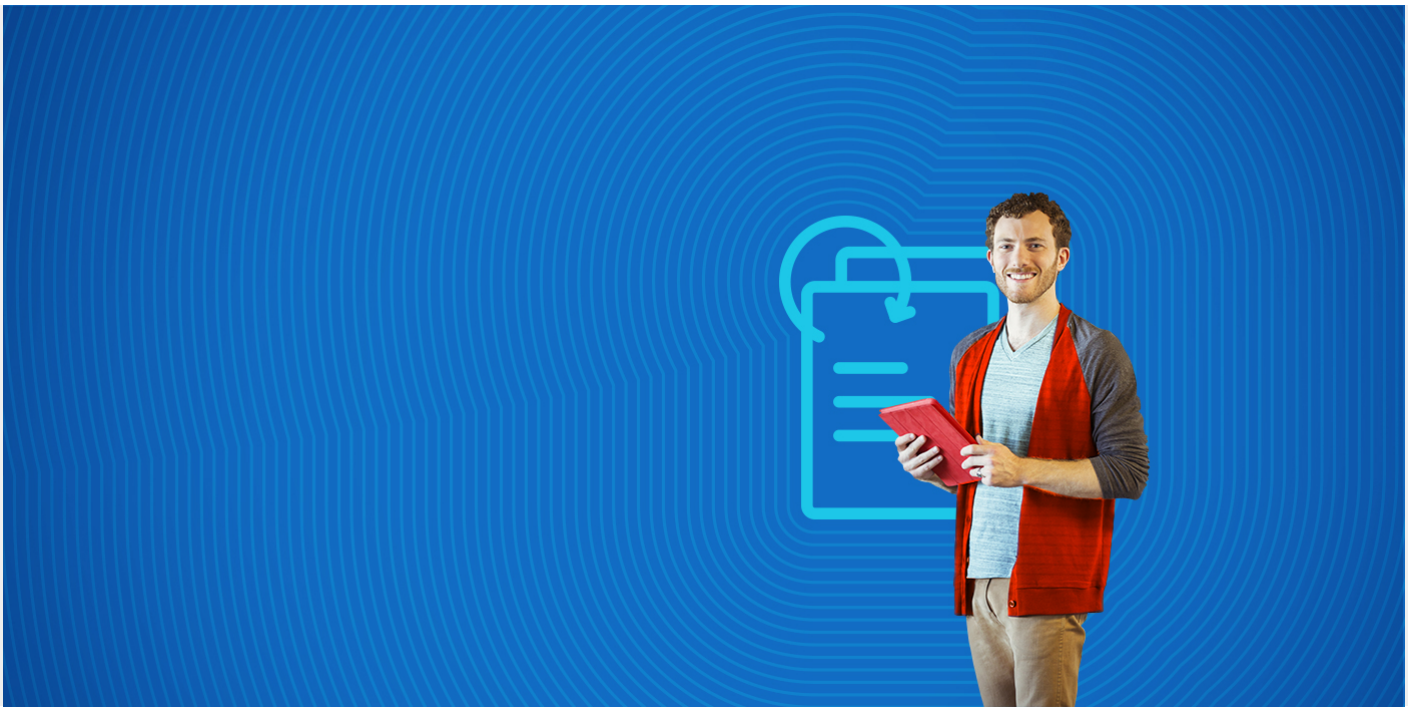
SNOWFLAKE: USING ANALYTICS & STATISTICAL FUNCTIONS



Snowflake does not do machine learning. It only has simple linear regression and basic statistical functions. (If you want to do machine learning with Snowflake, you need to put the data into Spark or another third-party product.)

You can, however, do analytics in Snowflake, armed with some knowledge of mathematics and aggregate functions and windows functions. Basic analytics is all you need in most situations, and it is the first step towards more elaborate analysis.

In this tutorial, we show you how to use Snowflake statistical functions with some examples. We will demonstrate these functions:

- Covariance
- Average, maximum, minimum, standard deviation
- Correlation
- Rank

First, let's set up our work.

(This article is part of our Snowflake Guide. Use the right-hand menu to navigate.)

Sample data

You need some data to work through this example. Download 1,000 hourly weather records from here.

(We purchased 20 years of weather data for Paphos, Cyprus from OpenWeather. This is just a small

subset of 1,000 records, converted to CSV format. We have worked with this data in other tutorials, including Loading CSV Files from S3 to Snowflake.)

Upload this data to Amazon S3 like this:

```
aws s3 cp paphosWeather.csv s3://gluebmcwalkerrowe/paphosWeather.csv
```

Create table in Snowflake

Unfortunately, Snowflake does not read the header record and create the table for you. (Presumably that is because they would prefer that you define the column data types and number precision and size yourself.)

```
create table weather(
dt integer,
temp decimal(6,2),
temp_min decimal(6,2),
temp_max decimal(6,2),
pressure int,
humidity int,
speed decimal(6,2),
deg int,
main varchar(50),
description varchar(50))
```

Copy data file to Snowflake stage area

Then create a Snowflake stage area like this.

(It does not matter what your Snowflake credentials are: put your Amazon AWS credentials. In fact, there is no place to look up your Snowflake credentials since they are the same as your Amazon IAM credentials even if you are running it on a Snowflake instance, i.e., you did not install Snowflake locally on your EC2 servers.)

```
create or replace stage paphosweather
url='s3://gluebmcwalkerrowe/paphosWeather.csv'
  credentials=(aws_key_id='xxxxxxx' aws_secret_key='xxxxxxx')
```

Now copy the data into the table you created above.

```
copy into weather
  from 's3://gluebmcwalkerrowe/paphosWeather.csv'
  credentials=(aws_key_id='xxxxxxx' aws_secret_key='xxxxxxxx')
  file_format = (type = csv field_delimiter = ',' skip_header = 1);
```

Covariance

Covariance c of the variables x and y is > 0 when an increase in x results in an increase in y. It's a measure of how two variables are related to each other. Let's show how to do this in Snowflake.

Each of the queries below wraps an inner query in an outer query. This is necessary because:

- Inner query runs some calculations making a smaller set that contains the result of those calculations.
- The outer query runs next. It uses the columns passed to it from the inner query to make the query process a two-step process. In other words, in Step 1 we create some set A. In Step 2, we produce either another set, call it B, or a scalar, which means a single number.

The data we have is hourly weather data from one location: Paphos, Cyprus. Meteorologists say that falling barometric pressure results in an increase in air speed. So, let's measure that by calculating the covariance between the decrease in air pressure and the increase in wind speed.

(Remember that we ran the tutorial below using 117,000 hourly weather data records from OpenWeather for a single city, Paphos, Cyprus, over 20 years. You can download 1,000 records, 41 days, from here. Or, purchase data for your location to investigate historical conditions there.)

We calculate:

1. The change in air pressure using the lag() windows function to look at the pressure at two points in time. A window function runs a calculation over adjacent rows in query results. So, we use **lag(pressure,8)** to calculate the change in air pressure over the previous 8 hours, i.e. 8 rows behind the current row. Each record in the database is 1 hour.
2. Then we calculate the change in wind speed over the same 8 hours using **lag()** as well.

Use simple subtraction to calculate the change. We order the data by the date column **dt** as we are working with time so the dates must be in order.

```
select to_timestamp(dt)
pressure, main,
lag(pressure, 1)over (order by dt)  as pressure1,
lag(pressure, 8) over (order by dt) as pressure8,
pressure - pressure8 as pressurechange,
 speed ,
lag(speed, 1) over (order by dt) as speed1,
lag(speed, 8) over (order by dt) as speed8,
speed8 -  speed1 as windchange
from weather
order by dt desc
```

Now we wrap the inner function inside the outer function. Note that:

- Since the inner function gave the name to the calculations **pressurechange** and **windchange** we simply write **covar_pop(pressurechange, windchange)**.
- We add the column **main**, which is **cloudy**, **rainy**, thunderstorm. etc. to show the covariance for each weather condition.

```
select  main, covar_pop(pressurechange, windchange) as covariance from
(select to_timestamp(dt)
pressure, main,
lag(pressure, 1)over (order by dt)  as pressure1,
lag(pressure, 8) over (order by dt)as pressure8,
pressure - pressure8 as pressurechange,
```

```
 speed ,
lag(speed, 1) over (order by dt) as speed1,
lag(speed, 8) over (order by dt) as speed8,
speed8 -  speed1 as windchange
from weather
order by dt desc)
group by main
```

Here we see that a decrease in barometric pressure results in an increase in wind speed, as we would expect. That effect is strongest in thunderstorms.

MAIN	COVARIANCE
Clouds'	2.682560523
Clear'	2.578284619
Thunderstorm'	3.578449916
Rain'	3.328280913
Dust'	1.697657949
Squall'	2.921487603
Mist'	2.513627372
Fog'	2.9256
Haze'	3.565528153
Tornado'	-1.425

Average, maximum, minimum, standard deviation

The average, maximum, minimum, and standard deviation are aggregate functions. That means they expect a **group by** clause. These are the most basic statistics. They give you an idea of how the average and average variation in some metrics over time. Note that:

- **to_timestamp()** converts the epoch time (which is seconds since 1970/01/01) to **yyyy-mm-dd hh:mm:ss**.
- We use **date_part()** to pull out the month and year.

```
select  round(avg(temp),2) as average, round(stddev(temp),2) as std,
max(temp) as maxtemp,
min(temp) as mintemp, date_part(year, to_timestamp(dt)) as year,
date_part(month,to_timestamp(dt)) as month
from weather
where to_timestamp(dt) > '2017-01-01'
group by date_part(year, to_timestamp(dt)), date_part(month,to_timestamp(dt))
order by date_part(year, to_timestamp(dt)), date_part(month,to_timestamp(dt))
```

Here are the results:

AVERAGE	STD	MAXTEMP	MINTEMP	YEAR	MONTH
53.4	5.57	64.08	37.42	2017	1

54.4	6.37	72.84	40.44	2017	2
59.17	5.1	70.84	45.91	2017	3
63.79	5.36	76.78	51.67	2017	4
79.28	0.1	79.39	79.21	2017	8
78.35	4.9	88.5	67.21	2017	9
71.86	4.86	82.2	62.13	2017	10
64.08	5.29	78.31	50.61	2017	11
59.55	5.69	69.87	45.84	2017	12
56.5	4.99	66.42	44.31	2018	1
58.81	4.47	68.04	47.88	2018	2
61.63	5.27	77.61	50.56	2018	3
65.91	6.07	78.75	51.94	2018	4
75.88	1.43	77.54	75.04	2018	8
79.21	5.01	88.81	66.67	2018	9
73.28	5.97	85.15	58.28	2018	10

We can get a year-by-year comparison by sorting the results by month first and then year:

```
select  round(avg(temp),2) as average, round(stddev(temp),2) as std,
max(temp) as maxtemp,
min(temp) as mintemp, date_part(year, to_timestamp(dt)) as year,
date_part(month,to_timestamp(dt)) as month
from weather
group by date_part(year, to_timestamp(dt)), date_part(month,to_timestamp(dt))
order by date_part(month,to_timestamp(dt)) ,date_part(year, to_timestamp(dt))
```

Correlation

Obviously the months correlated with the temperature as it gets hotter in summer and colder in winter. How strong is that correlation? You would think it would be close to 100%. Here, it's closed to 70% in the mild climate of Cyprus.

This results in a scalar statistic since it returns one value, **corr(month, average)** instead of rows.

```
select corr(month, average) from
(select  round(avg(temp),2) as average, round(stddev(temp),2) as std,
max(temp) as maxtemp,
min(temp) as mintemp, date_part(year, to_timestamp(dt)) as year,
date_part(month,to_timestamp(dt)) as month
from weather
where to_timestamp(dt) > '2017-01-01'
group by date_part(year, to_timestamp(dt)), date_part(month,to_timestamp(dt))
order by date_part(year, to_timestamp(dt)),
date_part(month,to_timestamp(dt)))
```

Results in this scalar value. So, they are positively correlated, but less than we would imagine.

```
CORR(MONTH, AVERAGE)
0.6127760999
```

Rank

Now let's calculate the hottest average months since 2017. We could do that with a regular SQL function, but if we use the **rank()** windows function we can number each row in the resulting set. That number is the rank.

This is a two-step function, too, because we need to convert hourly weather records to monthly so that we don't have too many lines:

1. Calculate average temperature for each month.
2. Select the metrics that we want ranked. This is given by putting the column **average** in the **(order by average desc)** statement.

```
select year, month, average, rank() over (order by average desc) as hottest
from
(select  round(avg(temp),2) as average, round(stddev(temp),2) as std,
max(temp) as maxtemp,
min(temp) as mintemp, date_part(year, to_timestamp(dt)) as year,
date_part(month,to_timestamp(dt)) as month
from weather
where to_timestamp(dt) > '2017-01-01'
group by date_part(year, to_timestamp(dt)),
date_part(month,to_timestamp(dt)))
```

Here are the results (in Fahrenheit).

YEAR	MONTH	AVERAGE	HOTTEST
2020	9	81.73	1
2019	8	79.36	2
2017	8	79.28	3
2018	9	79.21	4
2017	9	78.35	5
2019	9	77.74	6
2020	8	76.48	7
2018	8	75.88	8
2019	10	73.79	9
2018	10	73.28	10
2017	10	71.86	11
2019	11	68.13	12
2018	11	66.25	13
2018	4	65.91	14

Additional resources

For more tutorials like this, explore these resources:

- BMC Machine Learning & Big Data Blog
- How To Import Amazon S3 Data to Snowflake
- How To Query JSON Data in Snowflake
- AWS Guide, with 15 articles and tutorials
- Amazon Braket Quantum Computing: How To Get Started