| Transformation | SQL | Python |
|---|---|---|
| **Extracting Parts of a Date** | | |
| **Year** | SELECT YEAR(date_column) AS year_part FROM table_name; | df.select(year("date_column").alias("year_part")) |
| | | |
| **Month** | SELECT MONTH(date_column) AS month_part FROM table_name; | df.select(month("date_column").alias("month_part")) |
| | | |
| **Day** | SELECT DAY(date_column) AS day_part FROM table_name; | df.select(dayofmonth("date_column").alias("day_part")) |
| | | |
| **Hour** | SELECT HOUR(timestamp_column) AS hour_part FROM table_name; | df.select(hour("timestamp_column").alias("hour_part")) |
| | | |
| **Date Formatting** | | |
| **Convert a date to a string with a specific format** | SELECT DATE_FORMAT(date_column, 'yyyy-MM-dd') AS formatted_date FROM table_name; | df.select(date_format("date_column", "yyyy-MM-dd").alias("formatted_date")) |
| | | |
| **Date Arithmetic** | | |
| **Add** | SELECT DATE_ADD(date_column, 5) AS date_plus_5_days FROM table_name; | df.select(date_add("date_column", 5).alias("date_plus_5_days")) |
| | | |
| **Subtract** | SELECT DATE_SUB(date_column, 5) AS date_minus_5_days FROM table_name; | df.select(date_sub("date_column", 5).alias("date_minus_5_days")) |
| | | |
| **Difference Between Dates** | SELECT DATEDIFF(end_date_column, start_date_column) AS date_diff FROM | df.select(datediff("end_date_column", "start_date_column").alias("date_diff")) |
| | | |
| **Truncating Dates** | | |
| **Truncate to First Day of Month** | SELECT TRUNC(date_column, 'MM') AS first_day_of_month FROM table_name; | df.select(trunc("date_column", "MM").alias("first_day_of_month")) |
| | | |
| **Truncate to First Day of Year** | SELECT TRUNC(date_column, 'YYYY') AS first_day_of_year FROM table_name; | df.select(trunc("date_column", "YYYY").alias("first_day_of_year")) |

| Transformation | SQL | Python |
|---|---|---|
| | | |
| **Changing Timezones** | | |
| **Convert from one timezone to another** | SELECT FROM_UTC_TIMESTAMP(timestamp_column, 'America/New_York') AS ny_time FROM table_name; | df.select(from_utc_timestamp("timestamp_column", "America/New_York").alias("ny_time")) |
| | | |
| **Parsing Strings into Dates** | | |
| **Convert a string into a date** | SELECT TO_DATE(string_column, 'yyyy-MM-dd') AS date_column FROM table_name; | df.select(to_date("string_column", "yyyy-MM-dd").alias("date_column")) |
| | | |
| **Current Date and Time** | | |
| **Current Date** | SELECT CURRENT_DATE() AS current_date FROM table_name; | df.select(current_date().alias("current_date")) |
| | | |
| **Date Comparison** | | |
| **Compare two dates** | SELECT * FROM table_name WHERE date_column > '2023-01-01'; | df.filter(df["date_column"] > "2023-01-01") |
| | | |
| **Extracting Week and Quarter** | | |
| **Week of Year** | SELECT WEEKOFYEAR(date_column) AS week_of_year FROM table_name; | df.select(weekofyear("date_column").alias("week_of_year")) |
| | | |
| **Quarter of Year** | SELECT QUARTER(date_column) AS quarter_of_year FROM table_name; | df.select(quarter("date_column").alias("quarter_of_year")) |
| | | |
| **Adding/Subtracting Months and Years** | | |
| **Adding Months** | SELECT ADD_MONTHS(date_column, 3) AS date_plus_3_months FROM table_name; | df.select(add_months("date_column", 3).alias("date_plus_3_months")) |
| | | |
| **Subtracting Years** | SELECT ADD_MONTHS(date_column, -12) AS date_minus_1_year FROM table_name; | df.select(add_months("date_column", -12).alias("date_minus_1_year")) |

| Transformation | SQL | Python |
|---|---|---|
| | | |
| **Finding the Next/Previous Day of the Week** | | |
| **Next Monday** | SELECT NEXT_DAY(date_column, 'Monday') AS next_monday FROM table_name; | df.select(next_day("date_column", "Monday").alias("next_monday")) |
| | | |
| **Previous Sunday** | SELECT DATE_SUB(NEXT_DAY(date_column, 'Sunday'), 7) AS previous_sunday FROM | df.select(date_sub(next_day("date_column", "Sunday"), 7).alias("previous_sunday")) |
| | | |
| **Month** | | |
| **Last Day of the Current Month** | SELECT LAST_DAY(date_column) AS last_day_of_month FROM table_name; | df.select(last_day("date_column").alias("last_day_of_month")) |
| | | |
| **Converting Between Date and Timestamp** | | |
| **Date to Timestamp** | SELECT TO_TIMESTAMP(date_column, 'yyyy-MM-dd') AS timestamp_column FROM table_name; | df.select(to_timestamp("date_column", "yyyy-MM-dd").alias("timestamp_column")) |
| | | |
| **Timestamp to Date** | SELECT TO_DATE(timestamp_column) AS date_column FROM table_name; | df.select(to_date("timestamp_column").alias("date_column")) |
| | | |
| **Time Difference in Hours, Minutes, or Seconds** | | |
| **Difference in Hours** | SELECT ROUND((UNIX_TIMESTAMP(end_timestamp) - UNIX_TIMESTAMP(start_timestamp)) / 3600) AS | df.select((unix_timestamp("end_timestamp") - unix_timestamp("start_timestamp")) / 3600).alias("hours_diff") |
| | | |
| **Difference in Minutes** | SELECT ROUND((UNIX_TIMESTAMP(end_timestamp) - UNIX_TIMESTAMP(start_timestamp)) / 60) AS | df.select((unix_timestamp("end_timestamp") - unix_timestamp("start_timestamp")) / 60).alias("minutes_diff") |
| | | |

| Transformation | SQL | Python |
|---|---|---|
| **Difference in Seconds** | SELECT (UNIX_TIMESTAMP(end_timestamp) - UNIX_TIMESTAMP(start_timestamp)) AS seconds_diff FROM table_name; | df.select((unix_timestamp("end_timestamp") - unix_timestamp("start_timestamp")).alias("seconds_diff")) |
| | | |
| **Formatting Timestamp with Milliseconds** | | |
| **Extracting Milliseconds** | SELECT DATE_FORMAT(timestamp_column, 'yyyy-MM-dd HH:mm:ss.SSS') AS timestamp_with_ms FROM table_name; | df.select(date_format("timestamp_column", "yyyy-MM-dd HH:mm:ss.SSS").alias("timestamp_with_ms")) |
| | | |
| **Date Difference in Months** | | |
| **Difference in Months** | SELECT MONTHS_BETWEEN(end_date, start_date) AS months_diff FROM table_name; | df.select(months_between("end_date", "start_date").alias("months_diff")) |
| | | |
| **Extracting Day of Week** | | |
| **Day of Week (1=Monday, 7=Sunday)** | SELECT DAYOFWEEK(date_column) AS day_of_week FROM table_name; | df.select(dayofweek("date_column").alias("day_of_week")) |
| | | |
| **Getting the First Day of the Month** | | |
| **First Day of the Month** | SELECT DATE_TRUNC('MM', date_column) AS first_day_of_month FROM table_name; | df.select(date_trunc("MM", "date_column").alias("first_day_of_month")) |
| | | |
| **Rounding a Timestamp to the Nearest Interval** | | |
| **Round to Nearest Hour** | SELECT DATE_TRUNC('HOUR', timestamp_column) AS rounded_hour FROM table_name; | df.select(date_trunc("HOUR", "timestamp_column").alias("rounded_hour")) |
| **Round to Nearest Minute** | SELECT DATE_TRUNC('MINUTE', timestamp_column) AS rounded_minute FROM table_name; | df.select(date_trunc("MINUTE", "timestamp_column").alias("rounded_minute")) |
| **Shifting Time by Timezone** | | |

| Transformation | SQL | Python |
|---|---|---|
| **onvert from One Timezone to Another and Adjust for Daylight Saving** | SELECT FROM_UTC_TIMESTAMP(timestamp_column, 'America/Los_Angeles') AS la_time FROM table_name; | df.select(from_utc_timestamp("timestamp_column", "America/Los_Angeles").alias("la_time")) |
| | | |
| **Calculating the End of the Quarter** | | |
| **End of the Quarter** | SELECT LAST_DAY(ADD_MONTHS(DATE_TRUNC('QUARTER', date_column), 2)) AS end_of_quarter FROM table_name; | df.select(last_day(add_months(date_trunc("QUARTER", "date_column"), 2)).alias("end_of_quarter")) |
| **Extracting ISO Week Date Components** | | |
| **ISO Week Number** | SELECT WEEKOFYEAR_ISO(date_column) AS iso_week_number FROM table_name; | df.select(weekofyear("date_column").alias("iso_week_number")) |
| **ISO Year** | SELECT YEAROFWEEKISO(date_column) AS iso_year FROM table_name; | df.select(year("date_column").alias("iso_year")) |
| | | |
| **Handling Epoch/Unix Timestamps** | | |
| **Convert Unix Epoch to Timestamp** | SELECT FROM_UNIXTIME(unix_timestamp_column) AS timestamp_column FROM table_name; | df.select(from_unixtime("unix_timestamp_column").alias("timestamp_column")) |
| **Convert Timestamp to Unix Epoch** | SELECT UNIX_TIMESTAMP(timestamp_column) AS unix_timestamp FROM table_name; | df.select(unix_timestamp("timestamp_column").alias("unix_timestamp")) |
| | | |
| **Working with Leap Years** | | |
| **Checking if a Year is a Leap Year** | SELECT CASE WHEN (MOD(YEAR(date_column), 4) = 0 AND MOD(YEAR(date_column), 100) != 0) OR MOD(YEAR(date_column), 400) = 0 THEN TRUE ELSE FALSE END AS is_leap_year FROM table_name; | df.select((year("date_column") % 4 == 0).cast("boolean").alias("is_leap_year")) |
| | | |

| Transformation | SQL | Python |
|---|---|---|
| **Custom Date Parsing** | | |
| **Parsing Custom Date Formats**: | SELECT TO_DATE(CAST(UNIX_TIMESTAMP(string_column, 'MM-dd-yyyy') AS TIMESTAMP)) AS parsed_date FROM table_name; | df.select(to_date(unix_timestamp("string_column", "MM-dd-yyyy").cast("timestamp")).alias("parsed_date")) |
| | | |
| **Calculating Time Since a Specific Event** | | |
| **Time Passed in Days** | SELECT DATEDIFF(CURRENT_DATE(), event_date) AS days_since_event FROM table_name; | df.select(datediff(current_date(), "event_date").alias("days_since_event")) |
| **Time Passed in Months** | SELECT MONTHS_BETWEEN(CURRENT_DATE(), event_date) AS months_since_event FROM table_name; | df.select(months_between(current_date(), "event_date").alias("months_since_event")) |
| | | |
| **Interval Addition** | | |
| **Add Interval to a Timestamp** | SELECT TIMESTAMPADD(DAY, 10, timestamp_column) AS new_timestamp FROM table_name; | df.select(expr("timestamp_column + interval 10 days").alias("new_timestamp")) |
| | | |
| **Shifting Time by Specific Intervals** | | |
| **Add Hours to a Timestamp** | SELECT TIMESTAMPADD(HOUR, 3, timestamp_column) AS new_timestamp FROM table_name; | df.select(expr("timestamp_column + interval 3 hours").alias("new_timestamp")) |
| **Add Minutes to a Timestamp** | SELECT TIMESTAMPADD(MINUTE, 45, timestamp_column) AS new_timestamp FROM table_name; | df.select(expr("timestamp_column + interval 45 minutes").alias("new_timestamp")) |
| **Subtract Seconds from a Timestamp** | SELECT TIMESTAMPADD(SECOND, -30, timestamp_column) AS new_timestamp FROM table_name; | df.select(expr("timestamp_column - interval 30 seconds").alias("new_timestamp")) |
| | | |

| Transformation | SQL | Python |
|---|---|---|
| **Calculating the Number of Weekdays Between Two Dates** | | |
| **Weekdays Difference** | SELECT SIZE(SEQUENCE(<br>DATE_TRUNC('DAY', start_date),<br>DATE_TRUNC('DAY', end_date),<br>INTERVAL 1 DAY<br>)) - 2 * (DATEDIFF(end_date, start_date) / 7) AS weekday_diff<br>FROM table_name; | from pyspark.sql.functions import sequence, expr, size<br><br>df.select(<br>(size(sequence(<br>expr("date_trunc('DAY', start_date)"),<br>expr("date_trunc('DAY', end_date)"),<br>expr("INTERVAL 1 DAY")<br>)) - 2 * (expr("datediff(end_date, start_date) / 7"))).alias("weekday_diff")<br>) |
| | | |
| **Calculating Age Based on Birthdate** | | |
| **Age Calculation** | SELECT<br>FLOOR(MONTHS_BETWEEN(CURRENT_DATE(), birthdate_column) / 12) AS age FROM table_name; | df.select(floor(months_between(current_date(), "birthdate_column") / 12).alias("age")) |
| | | |
| **Identifying the Last Day of the Previous Month** | | |
| **Last Day of the Previous Month** | SELECT LAST_DAY(ADD_MONTHS(date_column, -1)) AS last_day_prev_month FROM table_name; | df.select(last_day(add_months("date_column", -1)).alias("last_day_prev_month")) |
| | | |
| **Calculating the Number of Days Until the End of the Year** | | |

| Transformation | SQL | Python |
|---|---|---|
| **Days Until End of Year** | SELECT DATEDIFF(LAST_DAY(DATE_TRUNC('YEAR', date_column), 'YYYY'), date_column) AS days_until_end_of_year FROM table_name; | df.select(datediff(last_day(date_trunc("YEAR", "date_column")), "date_column").alias("days_until_end_of_year")) |
| | | |
| **Handling Weekends in Date Calculations** | | |
| **Add Days Skipping Weekends** | SELECT CASE<br>WHEN DAYOFWEEK(date_column) IN (7, 1) THEN DATE_ADD(date_column, 2)<br>ELSE DATE_ADD(date_column, 1)<br>END AS next_business_day<br>FROM table_name; | df.select(<br>expr("""<br>CASE<br>WHEN DAYOFWEEK(date_column) IN (7, 1) THEN DATE_ADD(date_column, 2)<br>ELSE DATE_ADD(date_column, 1)<br>END<br>""").alias("next_business_day")<br>) |
| | | |
| **Calculating the Midpoint Between Two Dates** | | |
| **Midpoint Date** | SELECT DATE_ADD(start_date, DATEDIFF(end_date, start_date) / 2) AS midpoint_date FROM table_name; | df.select(date_add("start_date", (datediff("end_date", "start_date") / 2)).alias("midpoint_date")) |
| | | |
| **Extracting Fiscal Year and Quarter** | | |

| Transformation | SQL | Python |
|---|---|---|
| **Fiscal Year** | SELECT CASE<br>WHEN MONTH(date_column) >= 10 THEN YEAR(date_column) + 1<br>ELSE YEAR(date_column)<br>END AS fiscal_year<br>FROM table_name; | df.select(<br>expr("""<br>CASE<br>WHEN MONTH(date_column) >= 10 THEN YEAR(date_column) + 1<br>ELSE YEAR(date_column)<br>END<br>""").alias("fiscal_year")<br>) |
| **Fiscal Quarter** | SELECT CASE<br>WHEN MONTH(date_column) IN (10, 11, 12) THEN 1<br>WHEN MONTH(date_column) IN (1, 2, 3) THEN 2<br>WHEN MONTH(date_column) IN (4, 5, 6) THEN 3<br>ELSE 4<br>END AS fiscal_quarter<br>FROM table_name; | df.select(<br>expr("""<br>CASE<br>WHEN MONTH(date_column) IN (10, 11, 12) THEN 1<br>WHEN MONTH(date_column) IN (1, 2, 3) THEN 2<br>WHEN MONTH(date_column) IN (4, 5, 6) THEN 3<br>ELSE 4<br>END<br>""").alias("fiscal_quarter")<br>) |
| | | |
| **Converting to a Specific Date Format** | | |
| **Custom Date Format** | SELECT DATE_FORMAT(date_column, 'dd-MMM-yyyy') AS formatted_date FROM table_name; | df.select(date_format("date_column", "dd-MMM-yyyy").alias("formatted_date")) |
| | | |
| **Parsing Dates with Different Time Zones** | | |
| **Parse Date with Time Zone** | SELECT TO_TIMESTAMP_TZ('2023-01-01 10:00:00 America/New_York', 'yyyy-MM-dd HH:mm:ss z') AS timestamp_with_timezone FROM table_name; | df.select(to_timestamp("2023-01-01 10:00:00 America/New_York", "yyyy-MM-dd HH:mm:ss z").alias("timestamp_with_timezone")) |
| | | |

| Transformation | SQL | Python |
|---|---|---|
| **Calculating the Number of Full Weeks Between Two Dates** | | |
| **Weeks Difference** | SELECT FLOOR(DATEDIFF(end_date, start_date) / 7) AS weeks_diff FROM table_name; | df.select(floor(datediff("end_date", "start_date") / 7).alias("weeks_diff")) |
| | | |
| **Identifying the Start of the Week (Monday)** | | |
| **Start of the Week** | SELECT DATE_SUB(date_column, DAYOFWEEK(date_column) - 2) AS start_of_week FROM table_name; | df.select(date_sub("date_column", dayofweek("date_column") - 2).alias("start_of_week")) |
| | | |
| **Extracting Date Components in Different Formats** | | |
| **Day of Year** | SELECT DAYOFYEAR(date_column) AS day_of_year FROM table_name; | df.select(dayofyear("date_column").alias("day_of_year")) |
| **Week of Month** | SELECT WEEKOFMONTH(date_column) AS week_of_month FROM table_name; | df.select(expr("WEEKOFMONTH(date_column)").alias("week_of_month")) |
| | | |
| **Calculating the First Day of the Quarter** | | |
| **First Day of the Quarter** | SELECT DATE_TRUNC('QUARTER', date_column) AS first_day_of_quarter FROM table_name; | df.select(date_trunc("QUARTER", "date_column").alias("first_day_of_quarter")) |
| | | |
| **Creating a Date from Individual Components** | | |
| **Construct Date from Year, Month, and Day** | SELECT MAKE_DATE(year_column, month_column, day_column) AS constructed_date FROM table_name; | from pyspark.sql.functions import make_date<br>df.select(make_date("year_column", "month_column", "day_column").alias("constructed_date")) |
| | | |
| **Handling Null Dates** | | |

| Transformation | SQL | Python |
|---|---|---|
| **Replace Null Date with a Default Date** | SELECT COALESCE(date_column, '2023-01-01') AS non_null_date FROM table_name; | df.select(coalesce("date_column", lit("2023-01-01")).alias("non_null_date")) |
| | | |
| **Adding/Subtracting Date Parts** | | |
| **Add/Subtract Date Parts Using INTERVAL** | SELECT date_column + INTERVAL 2 MONTH - INTERVAL 1 DAY AS new_date FROM table_name; | df.select(expr("date_column + interval 2 months - interval 1 day").alias("new_date")) |
| | | |
| **Calculating Business Days Between Two Dates** | | |
| **Business Days Difference** | SELECT COUNT(CASE<br>WHEN DAYOFWEEK(DATE_ADD(start_date, seq)) NOT IN (7, 1)<br>THEN 1 END) AS business_days_diff<br>FROM table_name<br>LATERAL VIEW POSEXPLODE(SEQUENCE(0, DATEDIFF(end_date, start_date))) AS seq_idx, seq; | from pyspark.sql.functions import sequence, posexplode, expr<br>df.select(<br>expr("COUNT(CASE WHEN DAYOFWEEK(DATE_ADD(start_date, seq)) NOT IN (7, 1) THEN 1 END)").alias("business_days_diff")<br>).withColumn("seq", posexplode(sequence(expr("0"), datediff("end_date", "start_date")))) |
| | | |
| **Handling Different Date Formats in a Single Column** | | |
| **Parse Multiple Date Formats** | SELECT COALESCE(<br>TO_DATE(date_column, 'yyyy-MM-dd'),<br>TO_DATE(date_column, 'MM/dd/yyyy'),<br>TO_DATE(date_column, 'dd-MMM-yyyy')<br>) AS parsed_date<br>FROM table_name; | df.select(coalesce(<br>to_date("date_column", "yyyy-MM-dd"),<br>to_date("date_column", "MM/dd/yyyy"),<br>to_date("date_column", "dd-MMM-yyyy")<br>).alias("parsed_date")) |
| | | |
| **Calculating the First Monday After a Given Date** | | |

| Transformation | SQL | Python |
|---|---|---|
| **First Monday After Date** | SELECT NEXT_DAY(date_column, 'Monday') AS first_monday_after FROM table_name; | df.select(next_day("date_column", "Monday").alias("first_monday_after")) |
| | | |
| **Extracting the Decade from a Date** | | |
| **Extract Decade** | SELECT FLOOR(YEAR(date_column) / 10) * 10 AS decade FROM table_name; | df.select((floor(year("date_column") / 10) * 10).alias("decade")) |
| | | |
| **Calculating the Number of Weekends Between Two Dates** | | |
| **Weekends Count** | SELECT SIZE(FILTER(SEQUENCE(<br>start_date,<br>end_date,<br>INTERVAL 1 DAY<br>),<br>x -> DAYOFWEEK(x) IN (1, 7)<br>)) AS weekends_count FROM table_name; | df.select(<br>size(<br>filter(<br>sequence("start_date", "end_date", expr("INTERVAL 1 DAY")),<br>lambda x: dayofweek(x).isin([1, 7])<br>)<br>).alias("weekends_count")<br>) |
| | | |
| **Finding the Next Occurrence of a Specific Day of the Month** | | |

| Transformation | SQL | Python |
|---|---|---|
| **Next 15th of the Month** | SELECT CASE<br>WHEN DAY(date_column) > 15 THEN DATE_ADD(DATE_TRUNC('MM', DATE_ADD(date_column, INTERVAL 1 MONTH)), 14)<br>ELSE DATE_TRUNC('MM', date_column) + INTERVAL 14 DAY<br>END AS next_15th_of_month<br>FROM table_name; | df.select(expr("""<br>CASE<br>WHEN DAY(date_column) > 15 THEN DATE_ADD(DATE_TRUNC('MM', DATE_ADD(date_column, INTERVAL 1 MONTH)), 14)<br>ELSE DATE_TRUNC('MM', date_column) + INTERVAL 14 DAY<br>END<br>""").alias("next_15th_of_month")) |
| | | |
| **Calculating the Last Business Day of the Month** | | |
| **Last Business Day of the Month** | SELECT<br>CASE<br>WHEN DAYOFWEEK(last_day(date_column)) IN (1, 7)<br>THEN NEXT_DAY(LAST_DAY(date_column), 'FRIDAY')<br>ELSE LAST_DAY(date_column)<br>END AS last_business_day_of_month<br>FROM table_name; | df.select(expr("""<br>CASE<br>WHEN DAYOFWEEK(LAST_DAY(date_column)) IN (1, 7)<br>THEN NEXT_DAY(LAST_DAY(date_column), 'FRIDAY')<br>ELSE LAST_DAY(date_column)<br>END<br>""").alias("last_business_day_of_month")) |
| | | |
| **Calculating the Number of Holidays Between Two Dates** | | |
| **Counting Specific Holidays** | SELECT COUNT(*) AS holiday_count<br>FROM (SELECT EXPLODE(SEQUENCE(start_date, end_date, INTERVAL 1 DAY)) AS date_sequence) ds<br>WHERE DATE_FORMAT(date_sequence, 'MM-dd') IN ('12-25', '01-01'); | df.withColumn("date_sequence", explode(sequence("start_date", "end_date", expr("INTERVAL 1 DAY"))))\<br>.filter(date_format("date_sequence", "MM-dd").isin("12-25", "01-01"))\<br>.agg(count("*").alias("holiday_count")) |
| | | |
| **Rolling Date Calculation** | | |

| Transformation | SQL | Python |
|---|---|---|
| **Calculate Rolling 7-Day Average** | SELECT date_column, AVG(value_column) OVER (ORDER BY date_column ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_avg FROM table_name; | from pyspark.sql.window import Window<br>windowSpec = Window.orderBy("date_column").rowsBetween(-6, 0)<br>df.select("date_column", avg("value_column").over(windowSpec).alias("rolling_avg")) |
| | | |
| **Calculating the Duration Between Two Timestamps in Different Units** | | |
| **Duration in Hours, Minutes, and Seconds** | SELECT<br>TIMESTAMPDIFF(HOUR, start_timestamp, end_timestamp) AS hours_diff,<br>TIMESTAMPDIFF(MINUTE, start_timestamp, end_timestamp) % 60 AS minutes_diff,<br>TIMESTAMPDIFF(SECOND, start_timestamp, end_timestamp) % 60 AS seconds_diff<br>FROM table_name; | df.select(<br>expr("TIMESTAMPDIFF(HOUR, start_timestamp, end_timestamp)").alias("hours_diff"),<br>expr("TIMESTAMPDIFF(MINUTE, start_timestamp, end_timestamp) % 60").alias("minutes_diff"),<br>expr("TIMESTAMPDIFF(SECOND, start_timestamp, end_timestamp) % 60").alias("seconds_diff")<br>) |
| | | |
| **Identifying the Last Monday Before a Given Date** | | |
| **Last Monday Before Date** | SELECT DATE_SUB(date_column, MOD(DAYOFWEEK(date_column) + 5, 7)) AS last_monday FROM table_name; | df.select(date_sub("date_column", expr("MOD(DAYOFWEEK(date_column) + 5, 7)")).alias("last_monday")) |
| | | |
| **Handling Quarter-Based Calculations** | | |

| Transformation | SQL | Python |
|---|---|---|
| **Calculate Quarter Progress** | SELECT<br>(DAYOFYEAR(date_column) - DAYOFYEAR(DATE_TRUNC('QUARTER', date_column))) /<br><br>(DAYOFYEAR(ADD_MONTHS(DATE_TRUNC('QUARTER', date_column), 3)) - DAYOFYEAR(DATE_TRUNC('QUARTER', date_column))) AS quarter_progress<br>FROM table_name; | df.select(<br>((dayofyear("date_column") - dayofyear(date_trunc("QUARTER", "date_column"))) /<br>(dayofyear(add_months(date_trunc("QUARTER", "date_column"), 3)) - dayofyear(date_trunc("QUARTER", "date_column")))).alias("quarter_progress")<br>) |
| | | |
| **Calculating the Week Number Starting from a Specific Date** | | |
| **Custom Week Number Calculation** | SELECT FLOOR(DATEDIFF(date_column, '2024-01-01') / 7) + 1 AS custom_week_number FROM table_name; | df.select((floor(datediff("date_column", lit("2024-01-01")) / 7) + 1).alias("custom_week_number")) |
| | | |
| **Calculating the Time Until a Specific Time of Day** | | |
| **Time Until 5 PM** | SELECT TIMESTAMPDIFF(SECOND, CURRENT_TIMESTAMP,<br><br>TIMESTAMP(CONCAT(DATE(date_column), ' 17:00:00'))) AS seconds_until_5pm<br>FROM table_name; | df.select(expr("TIMESTAMPDIFF(SECOND, CURRENT_TIMESTAMP, TIMESTAMP(CONCAT(DATE(date_column), ' 17:00:00')))").alias("seconds_until_5pm")) |
| | | |
| **Extracting the Week Number and Year Together** | | |
| **Week Number with Year** | SELECT CONCAT(YEAR(date_column), '-', LPAD(WEEKOFYEAR(date_column), 2, '0')) AS year_week<br>FROM table_name; | df.select(expr("CONCAT(YEAR(date_column), '-', LPAD(WEEKOFYEAR(date_column), 2, '0'))").alias("year_week")) |

| Transformation | SQL | Python |
|---|---|---|
| | | |
| **Calculating the Next Occurrence of a Specific Day of the Week** | | |
| **Next Friday** | SELECT NEXT_DAY(date_column, 'FRIDAY') AS next_friday FROM table_name; | df.select(next_day("date_column", "FRIDAY").alias("next_friday")) |