```
/* ---------- 2: RegExp  ---------- */

SELECT
    extract_string_column,

    /* Method 1 -  Split creates and array of values
      and SAFE_ORDINAL picks up array elements by position */

    extract_string_column_array[SAFE_ORDINAL(1)] AS contact_category_array_element_1,
    extract_string_column_array[SAFE_ORDINAL(2)] AS contact_reason_array_element_2,
    extract_string_column_array[SAFE_ORDINAL(3)] AS contact_subreason_array_element_3,

    /* Method 2 - Regular expression way. Matching alpha characters with an optional
underscore character.
      Importantly, specifying the position and occurrence with last two parameters */

    REGEXP_EXTRACT(extract_string_column,"[a-zA-Z]+_*[a-zA-Z]+",1,1) AS
contact_category_regexp,
    REGEXP_EXTRACT(extract_string_column,"[a-zA-Z]+_*[a-zA-Z]+",1,2) AS
contact_reason_regexp,
    REGEXP_EXTRACT(extract_string_column,"[a-zA-Z]+_*[a-zA-Z]+",1,3) AS
contact_subreason_regexp
FROM
(
SELECT
     SPLIT('order_change_~_sizing_~_too_big',"_~_") AS extract_string_column_array,
     'order_change_~_sizing_~_too_big'            AS extract_string_column
 ) AS a       ;

/* --------- 3: SQL Bool Logic ---------- */

SELECT
     1 = 1              AS condition_1, -- Condition 1 /* This evaluates to TRUE */
     1 = 0              AS condition_2, -- Condition 2 /* This evaluates to FALSE */
     1 = 1 AND 1 = 0       AS condition_3, -- Condition 3 /* First condition evaluates to TRUE
and Second evaluates to FALSE (TRUE AND FALSE = FALSE) */
     NOT (1 = 1 OR 1 = 0)   AS condition_4, -- Condition 4 /* First and Second conditions
evaluate to (TRUE OR FALSE = TRUE), however the outer NOT makes the whole condition
FALSE
     TRUE <> (NOT (1 != 0)) AS condition_5, -- Condition 5 /* Second condition evaluates to
FALSE, therefore TRUE <> FALSE = TRUE */
```

```
      (CURRENT_DATE() > CURRENT_TIMESTAMP()) AS condition_6 -- Condition 6 /* This condition results in an "ERROR" using Google Bigquery syntax. Since, one is DATE and other is DATE_TIMESTAMP (type mismatch) */
                                                            /* Generally, DATE is always smaller than DATE_TIMESTAMP (given the database engine takes the syntax correctly) */
                                                            /* Therefore, it will evaluate to FALSE (Only if the database engine parses it correctly. For Bigquery its an "ERROR" to due type mismatch */
;

/* ------------- Bonus -------------- */

 NOT (CURRENT_DATE() <= DATE(CURRENT_TIMESTAMP()) /* The inner DATE comparison results in TRUE, however the outer NOT makes it FALSE */
 AND NOT EXISTS ( SELECT 1
                  FROM sample_table st
                  WHERE DATE(st.created_at) > CURRENT_DATE()) /* Inner Subquery - Record creation date in future? */
                                                    /* The outer EXISTS returns true if the subquery returns any row. */
                                                    /* Final condition reads as anything in "TODAY" and "FUTURE" is not needed */



/* --------- 4: Bigquery (Nested) Data ------------- */

 /* The biggest dilemma here is what columns gets updated whenever a new record is appended for a given ticket

    The key to building this query is to look at the very first date for the given ticket so that we know when the ticket was originally created in case of updates.
    Also, to rank the given ticket row by DESC order of the updated at to figure out the most recently updated record.

    I made assumptions that ticket form id does not change across records for a given ticket id in case of multiple updates.

    Status might update, therefore, I used it in the 2nd part of the WITH subquery.

    The mean and median resolution time is measured in days from the earliest created at to most recent updated at for a given ticket

For Bonus #1, I assumed the dataset of “wrong_products” since it was not clearly indicated as to what was applicable there.
```

CSAT calculation is (# of “good” tickets / total number of of tickets)

```
 */

 WITH customer_contact_events_part_1 AS (
   SELECT
         ticket_id,
         created_at,
         updated_at,
         customer_id,
         tikcet_form_id,
         status,
         satisfaction_rating.score
AS satisfaction_rating_score,
         FIRST_VALUE(created_at) OVER(PARTITION BY ticket_id ORDER BY created_at ASC
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)  AS
earliest_created_at,
         RANK()           OVER(PARTITION BY ticket_id ORDER BY updated_at DESC)
AS rank_row_per_ticket_id
  FROM customer_contatcs CROSS JOIN UNNEST(custom_fields) AS cf -- unnest cross join is
from repeated field.
  WHERE ticket_form_id =172834 /* Spanish Order Form */
    AND cf.value = 'wrong_product'
  ),  customer_contact_events_part_2 AS ( SELECT
                                      ticket_id,
                                      created_at,
                                      updated_at,
                                      customer_id,
                                      ticket_form_id,
                                      status,
                                      earliest_created_at,
                                      rank_row_per_ticket_id,
                                      satisfaction_rating_score,
                                      /* Calculating the resolution time in "Days".
                                         For only status='solved'.
                                         In all likelihood, its safe to use the status column based on the
most recent updated row
                                      */
                                      CASE WHEN status='solved' THEN
TIMESTAMP_DIFF(updated_at,earliest_created_at,DAY)
                                                   ELSE NULL
                                      END AS resolution_time_in_days
                                    FROM customer_contact_events_part_1
```

```
                                WHERE rank_row_per_ticket_id=1 /* Picking up the most recent
updated record for the ticket */
                                 /*  Looking at anything created last month,
                                    based on the earliest date creation of the ticket. */
                                 AND earliest_created_at BETWEEN
DATE_SUB(DATE_TRUNC(CURRENT_DATE(),MONTH),INTERVAL 1 MONTH)
                                                        AND
DATE_SUB(DATE_TRUNC(CURRENT_DATE(),MONTH),INTERVAL 1 DAY)
                                ),
                               customer_contact_events_part_3 AS (
                                                    SELECT
                                                        ticket_id,
                                                        satisfaction_rating_score,
                                                        resolution_time_in_days,
                                                        /* Calculating MEDIAN and MEAN
resolution times (in days) for the entire result set */

PERCENTILE_DISC(resolution_time_in_days,0.5) OVER() AS median_resolution_time,
                                                        AVG(resolution_time_in_days)
OVER() AS mean_resolution_time
                                                    FROM customer_contact_events_part_2
                                                   )
 SELECT
       ROUND(median_resolution_time,2) AS median_resolution_time,
       ROUND(mean_resolution_time,2)   AS mean_resolution_time,

       /* Positive CSAT %s for Bonus # 2 Question - see my explanation in the starting comment
*/
       ROUND(100*SAFE_DIVIDE(total_positive_csat_solved_within_median_resolution_time,
                        total_ticket_volume),2
           )                    AS percent_positive_csat_within_median_time,

ROUND(100*SAFE_DIVIDE(total_positive_csat_solved_within_mean_resolution_time_resolutio
n_time,
                        total_ticket_volume),2
           )                    AS percent_positive_csat_within_mean_time
 FROM
   (
    SELECT
        COUNT(DISTINCT ticket_id)        AS total_ticket_volume,
        COUNT(DISTINCT CASE WHEN satisfaction_rating_score='good'
                        AND resolution_time_in_days <= median_resolution_time THEN ticket_id
                                                        ELSE NULL
                    END
```

```
            )                       AS total_positive_csat_solved_within_median_resolution_time,
        COUNT(DISTINCT CASE WHEN satisfaction_rating_score='good'
                        AND resolution_time_in_days <= mean_resolution_time THEN ticket_id
                                                                            ELSE NULL
                    END
            )                       AS
total_positive_csat_solved_within_mean_resolution_time_resolution_time,
        /* These two are gonna be repeating across rows so using ANY_VALUE to avoid the
GROUP BY clause */
        ANY_VALUE(median_resolution_time) AS median_resolution_time,
        ANY_VALUE(mean_resolution_time)   AS mean_resolution_time
    FROM customer_contact_events_part_3
  ) AS main_query;


/* ------------- Bonus -------------- */

 /* #1 Mean and Median depict the distribution of the data, and are helpful to measure central
tendency in data.
      It would be interesting to see the if mean and median resolution depict a positive skewed
distribution.
      In the case of +ve skewness, the mean will be higher than the median. Converse is true in
the negative skewness.

      If the data has a large skewness, then it would be better off using the Median to gauge the
central tendency in data.
 */


/* --------- 1: First response time -------------- */

 /*
   Since, the bonus for Part 4 and and Part 1 are somewhat related,
   I would stay from the mean FRT, simply because of the outliers (bunch of high or low values
skewing the data distribution).

   My tendencies always fall toward the Median FRT.

   Even better, I would like to construct a "Whisker Bar Plot" to specifically look at the mean,
median, 25th, 50th and 75th data.

It will give me a vantage point view of my FRT data distribution and possible skewness in data.

   In addition to using the FRT, First Contact Resolution (FCR) could be used.
```

As customer loyalty is often the most talked about topic, FCR performance indicator gives more insights into how good are the agents at understanding and addressing the issues at hand thereby eliminating the need to have multiple interactions.

*/