1. Analyzing the Effectiveness of Moving the Personal Health Number (PHN) Question in the BC Demographic Survey

The BC Demographic Survey initially placed a question regarding respondents' Personal Health Numbers (PHN) in the first part of the survey. Observations in the first two weeks revealed a reluctance from participants to provide their PHN, leading to early drop-offs from the survey. We decided to move the PHN question to the final section of the survey to reduce early drop-offs and improve PHN completion rates. Below are the SQL queries I built to investigate if that change was effective.

1.1 PHN Completion Rate Analysis

I developed SQL queries to compare the PHN question's completion rates before and after it was relocated to the survey's end. This analysis aimed to quantify the change's impact, specifically looking at the total number of completed surveys and the responses to the PHN question. The completion rate was calculated as the number of respondents who answered the PHN question divided by the total number of completed surveys, segmented into periods before and after June 29, 2023.

```
-- Compare the PHN completion rate before and after the change
SELECT
    1 AS 'Order',
       'Before June 29' AS Period,
    COUNT(*) AS 'Total Completed',
    SUM(CASE WHEN Q9VERI = 1 THEN 1 ELSE 0 END) AS 'Responses to Q9VERI',
    CAST(SUM(CASE WHEN Q9VERI = 1 THEN 1 ELSE 0 END) AS FLOAT) / COUNT(*) AS 'PHN
Completion Rate'
FROM
    [BCDemoSurveyData].[dbo].[View_Response_Reporting_2]
WHERE
    QCOMP2 = 1 AND
    ASTART_DATE < '2023-06-29'
UNION ALL
SELECT
    2 AS 'Order',
       'After June 29' AS Period,
    COUNT(*) AS 'Total Completed',
    SUM(CASE WHEN Q9VERI_P3 = 1 THEN 1 ELSE 0 END) AS 'Responses to Q9VERI',
    CAST(SUM(CASE WHEN Q9VERI_P3 = 1 THEN 1 ELSE 0 END) AS FLOAT) / COUNT(*) AS 'PHN
Completion Rate'
FROM
    [BCDemoSurveyData].[dbo].[View_Response_Reporting_2]
WHERE
    QCOMP2 = 1 AND
    ASTART_DATE >= '2023-06-29'
```

1.2 Detailed Drop-off Rate Analysis

To further understand the dynamics of survey participation, I also analyzed the drop-off rates in more detail, focusing on the specific points within the survey where participants were more likely to exit before completion. This analysis provided insights into how the relocation of the PHN question influenced participant behavior across different sections of the survey. The query I constructed segmented drop-off counts by the last completed question, comparing periods before and after the change.

This analysis revealed that while participants still tended to drop off at the PHN question, the overall response rate to other questions increased due to this strategic relocation. This outcome underscored the effectiveness of moving the PHN question to the survey's end, as it allowed us to collect more comprehensive data across other survey sections, enriching our dataset and enhancing the survey's overall success.

```
-- Compare drop off rate before and after the PHN question was moved, detailed by last
dropped off question
WITH DropOffs AS (
  SELECT
    CASE
      WHEN ASTART_DATE < '2023-06-29' THEN 'Before June 29'
      ELSE 'After June 29'
    END AS Period,
    CASE
      WHEN ASTART_DATE < '2023-06-29' THEN ALASTSCREEN
      WHEN ASTART_DATE >= '2023-06-29' AND ALASTSCREEN_P2 IS NOT NULL THEN ALASTSCREEN_P2
      WHEN ASTART_DATE >= '2023-06-29' THEN ALASTSCREEN_P3
      ELSE NULL
    END AS LastScreen,
    1 AS DropOffCount
  FROM
    [BCDemoSurveyData].[dbo].[View_Response_Reporting_0]
  WHERE
    (ALASTSCREEN IS NOT NULL AND ASTART_DATE < '2023-06-29') OR
    (ALASTSCREEN_P2 IS NOT NULL AND ASTART_DATE >= '2023-06-29') OR
    (ALASTSCREEN_P3 IS NOT NULL AND ASTART_DATE >= '2023-06-29')
)
SELECT
  Period,
  LastScreen,
  COUNT(*) AS DropOffCount
FROM
  DropOffs
GROUP BY
  Period, LastScreen
ORDER BY
         Period, COUNT(*) DESC;
```

2. De-identifying and Preparing Survey Data for NLP Analysis and Coding

In my capacity as a data analyst for the BC Demographic Survey, I encountered the challenge of dealing with a variety of open-ended responses. Our objective was twofold: first, to categorize a substantial volume of open responses into predefined categories using NLP models; and second, to manually code the remaining responses with the help of both internal and external coders. To facilitate this process, I established a comprehensive procedure for de-identifying the survey data, exporting it to a coding database in the required format, and ensuring that the newly coded data could be seamlessly integrated back into our main database.

2.1 Data De-identification and Export

I initiated the process by generating unique identifiers for each survey response to ensure anonymity. This was achieved through the creation of a UniqueIdentifier table, which mapped each respondent's Telkey to a new, anonymized ID.

```
USE BCDemoSurveyData;
GO
-- Generate unique identifier and create a table --
CREATE TABLE dbo.UniqueIdentifier (
    IDGenerate INT IDENTITY(1,1) NOT NULL,
       ID AS RIGHT('0000000' + CAST(IDGenerate AS VARCHAR(7)), 7) PERSISTED,
    Telkey VARCHAR(255),
    PRIMARY KEY (IDGenerate)
);
```

Subsequently, I populated this table with Telkeys from completed surveys, sorted by their start date, and established a method for appending new Telkeys as additional surveys were completed.

```
-- Populate the table with telkeys of completed surveys and ordered by start date --
INSERT INTO dbo.UniqueIdentifier (Telkey)
SELECT _telkey
FROM View_Response_Reporting_2
WHERE QCOMP2 = 1
ORDER BY ASTART_DATE;

-- Append new telkeys to the table to generate new unique identifiers when there's new
records --
USE BCDemoSurveyData;
GO
-- Identify the new telkeys that have not yet been added to the UniqueIdentifier table --
DECLARE @NewTelkeys TABLE (Telkey VARCHAR(255));

INSERT INTO @NewTelkeys (Telkey)
SELECT _telkey
FROM View_Response_Reporting_2 AS VRR
WHERE VRR.QCOMP2 = 1
AND NOT EXISTS (
    SELECT 1
    FROM dbo.UniqueIdentifier AS UI
    WHERE UI.Telkey = VRR._telkey
);
-- Append the new telkeys to the UniqueIdentifier table --
INSERT INTO dbo.UniqueIdentifier (Telkey)
SELECT Telkey
FROM @NewTelkeys;
```

2.2 Data Preparation for Coding

For the coding of open-ended responses, I developed a view tailored to the specific column structure needed by our coding team and data scientists. This view is designed to refresh automatically with the arrival of new data, facilitating the subsequent manual inclusion of these new entries at a later phase.

```
-- Create a view that matches the column structures in Access --
USE BCDemoSurveyData;
GO
```

```sql
CREATE VIEW dbo.Q32_open_ends
AS
SELECT
    ID.ID,
    CW.Q32RACE,
    CW.AQ32RACE,
    NULL AS AQ32RACE_Cleaned,
    NULL AS Coding_Comment,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 1)) AS Q32RACE_C01,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 2)) AS Q32RACE_C02,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 3)) AS Q32RACE_C03,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 4)) AS Q32RACE_C04,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 5)) AS Q32RACE_C05,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 6)) AS Q32RACE_C06,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 7)) AS Q32RACE_C07,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 8)) AS Q32RACE_C08,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 9)) AS Q32RACE_C09,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 10)) AS Q32RACE_C10,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 11)) AS Q32RACE_C11,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 12)) AS Q32RACE_C12,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 13)) AS Q32RACE_C13,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 14)) AS Q32RACE_C14,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 15)) AS Q32RACE_C15,
    REVERSE(PARSENAME(REPLACE(REVERSE(CW.Q32RACE), 'µ', '.'), 16)) AS Q32RACE_C16,
    ID.Cycle
FROM
    dbo.UniqueIdentifier AS ID
INNER JOIN
    callweb2023_bcds_p1.callweb AS CW
ON
    ID.Telkey = CW._telkey
WHERE
    CW.Q32RACE LIKE '%97%';

-- Crete a table copy of the view in BCDemoSurveyCoding
USE BCDemoSurveyCoding;
GO

SELECT * INTO dbo.Q32RACE
FROM BCDemoSurveyData.dbo.Q32_open_ends;

-- Manually append records that are not in the coding table
USE BCDemoSurveyCoding;
GO

INSERT INTO dbo.AQ32RACE
SELECT *
FROM BCDemoSurveyData.dbo.Q32_open_ends AS V
WHERE NOT EXISTS (
    SELECT 1
    FROM dbo.AQ32RACE AS T
    WHERE T.ID = V.ID
);
```

2.3 Integration of Coded Responses

Once coding was completed, I established another process to integrate the newly coded data back into our archive table, ensuring the original codes were replaced with the updated values. This involved complex SQL operations, such as splitting concatenated values into separate rows and updating the archive table accordingly.

```
-- Sending finished code back to Archive table
-- Q34CULTURE
-- split values to different rows so one column has all the codes --
WITH SplitValues AS (
    SELECT
        c.ID,
        value,
        ROW_NUMBER() OVER (PARTITION BY c.ID ORDER BY (SELECT NULL)) AS SeqNum
    FROM
        [BCDemoSurveyCoding].[dbo].[AQ34CULTURE_Concatenated] c
        CROSS APPLY STRING_SPLIT(c.ConcatenatedColumn, ',')
),
-- creates a single row for each ID with separate columns --
PivotedValues AS (
    SELECT
        ID,
        MAX(CASE WHEN SeqNum = 1 THEN value END) AS C01,
        MAX(CASE WHEN SeqNum = 2 THEN value END) AS C02,
        MAX(CASE WHEN SeqNum = 3 THEN value END) AS C03,
        MAX(CASE WHEN SeqNum = 4 THEN value END) AS C04,
        MAX(CASE WHEN SeqNum = 5 THEN value END) AS C05,
        MAX(CASE WHEN SeqNum = 6 THEN value END) AS C06,
        MAX(CASE WHEN SeqNum = 7 THEN value END) AS C07,
        MAX(CASE WHEN SeqNum = 8 THEN value END) AS C08,
        MAX(CASE WHEN SeqNum = 9 THEN value END) AS C09,
        MAX(CASE WHEN SeqNum = 10 THEN value END) AS C10
    FROM SplitValues
    GROUP BY ID
)
-- update the archive table --
UPDATE a
SET
    Q34CULTURE_A = COALESCE(p.C01, a.Q34CULTURE_A),
    Q34CULTURE_B = COALESCE(p.C02, a.Q34CULTURE_B),
    Q34CULTURE_C = COALESCE(p.C03, a.Q34CULTURE_C),
    Q34CULTURE_D = COALESCE(p.C04, a.Q34CULTURE_D),
    Q34CULTURE_E = COALESCE(p.C05, a.Q34CULTURE_E),
    Q34CULTURE_F = COALESCE(p.C06, a.Q34CULTURE_F),
    Q34CULTURE_G = COALESCE(p.C07, a.Q34CULTURE_G),
    Q34CULTURE_H = COALESCE(p.C08, a.Q34CULTURE_H),
    Q34CULTURE_I = COALESCE(p.C09, a.Q34CULTURE_I),
    Q34CULTURE_J = COALESCE(p.C10, a.Q34CULTURE_J)
FROM
    [BCDemoSurveyData].[dbo].[Archive_Table] a
INNER JOIN
    [BCDemoSurveyData].dbo.Disposition d ON a._telkey = d.Telkey
INNER JOIN
    PivotedValues p ON d.ID = p.id;
```

3. Data validation for Duplicate Entries

After the survey concluded, our data validation efforts focused on identifying and eliminating duplicate and invalid entries. I developed all necessary queries for this purpose. Below is a query I designed to detect duplicates by name within the same postal code, subsequently marking the older records for exclusion based on their unique identifier.

```
-- Append older telkeys to Telkeys_To_Exclude Table --
-- Selecting all entries from All_name_h1 where there's a corresponding survey response
WITH
PartialName AS (
    SELECT n.*
    FROM dbo.All_name_h1 n
    WHERE EXISTS (
        SELECT 1
        FROM dbo.View_Response_Reporting_2 r
        WHERE n.ACHILDTELKEY = r._telkey
        AND r.Q32RACE <> ''
    )
),
-- Identifying duplicate entries based on postal code and name, ordered by the end date
DuplicateData AS (
    SELECT
        a.ADDR_MPOSTALCODE,
        a.FirstName,
        a.LastName,
        DENSE_RANK() OVER (PARTITION BY a.ADDR_MPOSTALCODE, a.FirstName, a.LastName ORDER
BY a.AEND_DATE_TIME DESC) as RankNum,
        a.ACHILDTELKEY as DuplicateTelKey,
        a.AEND_DATE_TIME
    FROM PartialName a
    JOIN PartialName b
        ON a.ADDR_MPOSTALCODE = b.ADDR_MPOSTALCODE
        AND a.FirstName = b.FirstName
        AND a.LastName = b.LastName
        AND a.ACHILDTELKEY <> b.ACHILDTELKEY
),
-- Aggregating duplicate data to count duplicates and identify older records
AggregatedData AS (
    SELECT
        ADDR_MPOSTALCODE,
        FirstName,
        LastName,
        COUNT(DISTINCT DuplicateTelKey) as DuplicateCount,
        MAX(CASE WHEN RankNum = 1 THEN DuplicateTelKey ELSE NULL END) as TelKey1,
        MAX(CASE WHEN RankNum = 2 THEN DuplicateTelKey ELSE NULL END) as TelKey2,
        MAX(CASE WHEN RankNum = 3 THEN DuplicateTelKey ELSE NULL END) as TelKey3,
        MAX(CASE WHEN RankNum = 4 THEN DuplicateTelKey ELSE NULL END) as TelKey4,
        MAX(CASE WHEN RankNum = 5 THEN DuplicateTelKey ELSE NULL END) as TelKey5,
        MAX(CASE WHEN RankNum = 6 THEN DuplicateTelKey ELSE NULL END) as TelKey6,
        MAX(CASE WHEN RankNum = 7 THEN DuplicateTelKey ELSE NULL END) as TelKey7
    FROM DuplicateData
    GROUP BY ADDR_MPOSTALCODE, FirstName, LastName
    HAVING COUNT(DISTINCT DuplicateTelKey) > 1 AND ADDR_MPOSTALCODE <> ''
)
```

```
-- Process to append older duplicate records' telkeys to the exclusion list
INSERT INTO dbo.Telkeys_To_Exclude (Telkey_To_Exclude)
SELECT DuplicateTelKey
FROM AggregatedData
CROSS APPLY (VALUES
    (TelKey2),
    (TelKey3),
    (TelKey4),
    (TelKey5),
    (TelKey6),
    (TelKey7)
) AS X(DuplicateTelKey)
WHERE DuplicateTelKey IS NOT NULL
AND NOT EXISTS (
       -- Ensuring the telkey isn't already marked for exclusion
    SELECT 1
    FROM dbo.Telkeys_To_Exclude
    WHERE Telkey_To_Exclude = DuplicateTelKey
)
```

4. Non-response Analysis

For our survey, participants had the option to select "Prefer Not to Answer" for any question. I created a dynamic SQL script to iterate over each question, quantifying the instances of non-response. This approach efficiently aggregates non-responses across all survey questions, highlighting areas with higher instances of participant reluctance or privacy concerns for future survey design considerations.

```
DECLARE @sql NVARCHAR(MAX) = '';

-- Initialize a temporary table to hold non-response counts
CREATE TABLE #Temp (
    ColumnName NVARCHAR(128),
    Num_of_responses INT,
    Num_of_99 INT
);

-- Dynamically generate SQL to tally non-responses for each question
SELECT @sql += '
INSERT INTO #Temp (ColumnName, Num_of_responses, Num_of_99)
SELECT
    ''' + COLUMN_NAME + ''' AS ColumnName,
    COUNT(NULLIF(' + QUOTENAME(COLUMN_NAME) + ', '''')) AS Num_of_responses,
    SUM(CASE
        WHEN ''' + COLUMN_NAME + ''' IN (''Q49INC'', ''Q51INC'', ''Q52INC'',
''Q54DISAB'') AND ' + QUOTENAME(COLUMN_NAME) + ' = ''999'' THEN 1
        WHEN ''' + COLUMN_NAME + ''' LIKE ''Q34CULTURE_C%'' AND ' +
QUOTENAME(COLUMN_NAME) + ' = ''99999'' THEN 1
        WHEN ''' + COLUMN_NAME + ''' NOT IN (''Q49INC'', ''Q51INC'', ''Q52INC'',
''Q54DISAB'') AND ''' + COLUMN_NAME + ''' NOT LIKE ''Q34CULTURE_C%'' AND ' +
QUOTENAME(COLUMN_NAME) + ' = ''99'' THEN 1
        ELSE 0
    END) AS Num_of_99
FROM
    [BCDemoSurveyData].[dbo].[BC_Demographic_Survey_Jan03_Second_Ingestion_Final];'
FROM
```

```
    INFORMATION_SCHEMA.COLUMNS
WHERE
    TABLE_NAME = 'BC_Demographic_Survey_Jan03_Second_Ingestion_Final' AND TABLE_SCHEMA =
'dbo';

-- Execute the dynamic SQL
EXEC sp_executesql @sql;

-- Select the results from the temporary table
SELECT * FROM #Temp;

-- Drop the temporary table
-- DROP TABLE #Temp;
```