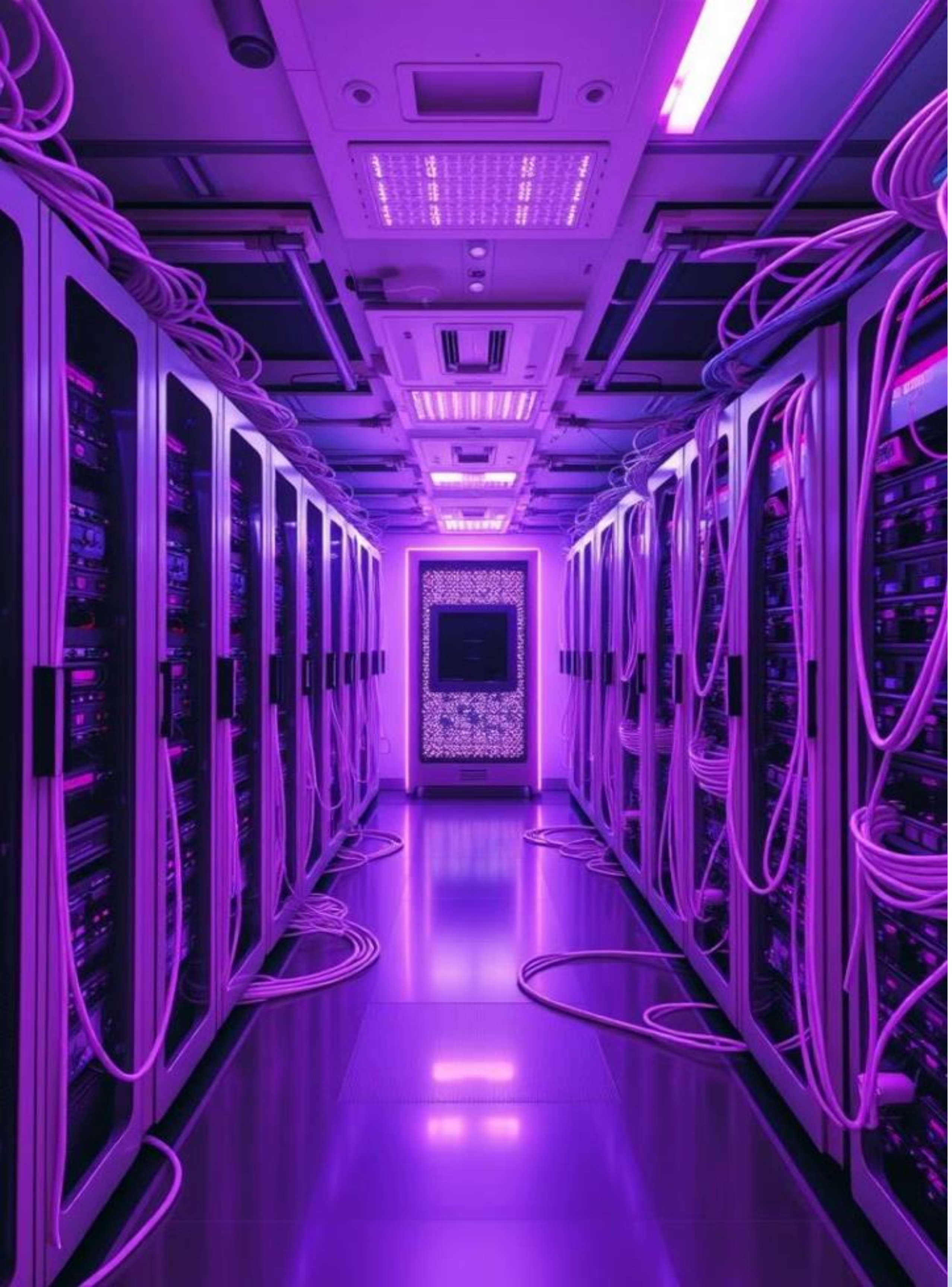


User Analytics for Strategic Business Acquisition

In this project, I analyzed telecom user behaviour, network performance, and customer satisfaction to uncover actionable insights for improving the user experience. The dataset, containing over 150,000 records, was thoroughly cleaned by handling duplicates, filling missing values, and addressing extreme values. The data preparation set the stage for detailed analysis, where I explored user engagement, application preferences, and performance metrics like download speeds and retransmission rates.

To provide deeper insights, I used machine learning techniques, including K-Means Clustering for user segmentation and Random Forest Regression to predict satisfaction scores. I saved the trained model using `pickle`, enabling seamless reusability for future predictions. Additionally, I created a visually engaging Streamlit dashboard, providing an interactive interface for stakeholders to explore key findings and visualizations in real-time.

The results were further structured by exporting critical insights and aggregated scores to an SQL Server database. This required configuring the database, defining table structures, and automating data insertion with Python. By combining data engineering, analytics, and predictive modelling with intuitive tools like Streamlit and SQL Server, this project delivered a comprehensive solution for understanding and improving telecom user experience.

by Debasis Baidya

Importing of the necessary Libraries

01
Numpy
It helped me perform numerical computations and handle arrays.

02
Pandas
It helped me manipulate and analyze data efficiently.

03
Matplotlib
It helped me create visualizations and plots.

04
Seaborn
It helped me generate attractive statistical graphics.

05
Scikit-learn
It helped me scale features for machine learning.

06
Pickle
This module helped me serialize and deserialize Python objects.

07
SQLAlchemy
It helped me connect to SQL SSMS and execute SQL queries.

08
Streamlit
It helped me create interactive web applications for model interaction.

Loading Data & Checking Info

	Bearer Id	Start	Start ms	End	End ms	Dur. (ms)	IMSI	MSISDN/Number	IMEI
0	13114483460844900352	2019-04-04 12:01:18	770	2019-04-25 14:35:31	662	1823652	208201448079117	33664962239	35521209507511
1	13114483482878900224	2019-04-09 13:04:04	235	2019-04-25 08:15:48	606	1365104	208201909211140	33681854413	35794009006359
2	13114483484080500736	2019-04-09 17:42:11	1	2019-04-25 11:58:13	652	1361762	208200314458056	33760627129	35281510359387
3	13114483485442799616	2019-04-10 00:31:25	486	2019-04-25 07:36:35	171	1321509	208201402342131	33750343200	35356610164913
4	13114483499480700928	2019-04-12 20:10:23	565	2019-04-25 10:40:32	954	1089009	208201401415120	33699795932	35407009745539

```
# DataFrame info
print("\nDataFrame Info:")
data.info()
```

```
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150001 entries, 0 to 150000
Data columns (total 55 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   Bearer Id                       150001 non-null  object
 1   Start                           150000 non-null  datetime64[ns]
 2   Start ms                        150000 non-null  float64
 3   End                             150000 non-null  datetime64[ns]
 4   End ms                          150000 non-null  float64
 5   Dur. (ms)                       150000 non-null  float64
 6   IMSI                            149431 non-null  float64
 7   MSISDN/Number                   148935 non-null  float64
 8   IMEI                            149429 non-null  float64
 9   Last Location Name              148848 non-null  object
 10  Avg RTT DL (ms)                 122172 non-null  float64
 11  Avg RTT UL (ms)                 122189 non-null  float64
 12  Avg Bearer TP DL (kbps)         150000 non-null  float64
 13  Avg Bearer TP UL (kbps)         150000 non-null  float64
 14  TCP DL Retrans. Vol (Bytes)     61855 non-null   float64
 15  TCP UL Retrans. Vol (Bytes)     53352 non-null   float64
 16  DL TP < 50 Kbps (%)             149247 non-null  float64
 17  50 Kbps < DL TP < 250 Kbps (%)  149247 non-null  float64
 18  250 Kbps < DL TP < 1 Mbps (%)   149247 non-null  float64
 19  DL TP > 1 Mbps (%)              149247 non-null  float64
 20  UL TP < 10 Kbps (%)             149209 non-null  float64
 21  10 Kbps < UL TP < 50 Kbps (%)   149209 non-null  float64
 22  50 Kbps < UL TP < 300 Kbps (%)  149209 non-null  float64
 23  UL TP > 300 Kbps (%)            149209 non-null  float64
 24  HTTP DL (Bytes)                 68527 non-null   float64
 25  HTTP UL (Bytes)                 68191 non-null   float64
 26  Activity Duration DL (ms)       150000 non-null  float64
 27  Activity Duration UL (ms)       150000 non-null  float64
 28  Dur. (ms).1                     150000 non-null  float64
 29  Handset Manufacturer            149429 non-null  object
 30  Handset Type                    149429 non-null  object
 31  Nb of sec with 125000B < Vol DL 52463 non-null   float64
```

1

Loading Data

Reading the Excel file using pandas i.e. ‘pd.read_excel’

2

Displaying the Max Columns in the DataFrame

Displaying the max columns using pd.set_option('display.max_columns', None) method.

3

Checking Data Information

Using info() to view Column Types, data counts, Data Types.

Task 1 - User Overview Analysis

Task 1.1 - User Behaviour Overview

Task 1.1 - User Behavior Overview

```python
def user_behavior_overview(data):
    """Aggregates user behavior information."""
    user_agg = data.groupby('MSISDN/Number').agg(
        num_sessions=('Bearer Id', 'count'),
        total_duration=('Dur. (ms)', 'sum'),
        total_dl_data=('Total DL (Bytes)', 'sum'),
        total_ul_data=('Total UL (Bytes)', 'sum')
    )

    # Create a dictionary to store aggregated data per application
    app_data = {}
    app_columns = ['Social Media DL (Bytes)', 'Social Media UL (Bytes)',
                   'Youtube DL (Bytes)', 'Youtube UL (Bytes)',
                   'Netflix DL (Bytes)', 'Netflix UL (Bytes)',
                   'Google DL (Bytes)', 'Google UL (Bytes)',
                   'Email DL (Bytes)', 'Email UL (Bytes)',
                   'Gaming DL (Bytes)', 'Gaming UL (Bytes)',
                   'Other DL (Bytes)', 'Other UL (Bytes)']

    for col in app_columns:
        if 'DL' in col:
            app_name = col.replace(' DL (Bytes)', '')
            dl_col = col
            ul_col = col.replace('DL', 'UL')
            app_data[app_name] = data.groupby('MSISDN/Number')[[dl_col, ul_col]].sum()

            # Sum DL and UL columns for a single total data volume per application
            app_data[app_name]['total_data_volume'] = app_data[app_name][dl_col] + app_data[app_name][ul_col]

            # Rename column for easy access
            app_data[app_name] = app_data[app_name].rename(columns={'total_data_volume': f'{app_name}_total_data_volume'})

            # Remove the individual DL/UL columns before merging
            app_data[app_name] = app_data[app_name].drop(columns=[dl_col, ul_col])

    # Merge application specific data into user aggregated data
    for app_name, app_df in app_data.items():
        user_agg = user_agg.merge(app_df, on='MSISDN/Number', how='left')

    user_agg = user_agg.fillna(0) # Replace NaNs with 0
    return user_agg

# Call the user_behavior_overview function
user_agg_data = user_behavior_overview(data)

# Display the result
user_agg_data
```

MSISDN/Number	num_sessions	total_duration	total_dl_data	total_ul_data	Social Media_total_data_volume	Youtube_total_data_volume	Netflix_total_data_volume	Goc
33601001722	1	116720	842637466	36053108	2232135	21624548	27180981	
33601001754	1	181230	120755184	36104459	2660565	12432223	11221763	
33601002511	1	134969	556659663	39306820	3195623	21333570	19353900	
33601007832	1	49878	401993172	20327526	280294	6977321	1942092	
33601008617	2	37104	1363130417	94280527	2912542	41533002	49201724	
...	...	...	...	...	...	...	...	
33789996170	1	8810	687925212	26716429	300183	26647843	14902538	
33789997247	1	140988	444575092	35732243	498569	19851572	8531060	
3197020876596	1	877385	194828056	37295915	715224	11959905	26592300	
337000037000919	1	253030	539634985	56652839	521566	36734940	30905042	
882397108489451	1	869844	78697597	60456049	1546088	40940710	28846230	

106856 rows × 11 columns

1

Function Creation

Developed a function to aggregate user behaviour data. Generated a DataFrame (user_agg_data) containing aggregated user metrics, enabling detailed analysis of user engagement across various applications.

2

Data Aggregation

- Grouped by MSISDN/Number to calculate:
- Number of Sessions: Count of Bearer Id.
- Total Duration: Sum of Dur. (ms).
- Total Download Data: Sum of Total DL (Bytes).
- Total Upload Data: Sum of Total UL (Bytes).

3

Application-Specific Aggregation

- Created a dictionary for applications (e.g., Social Media, YouTube).
- Summed download (DL) and upload (UL) data.
- Calculated Total Data Volume for each application and merged it into the main dataset.

4

Data Cleaning

- Replaced NaN values with 0 for clarity.

Task 1 - User Overview Analysis

Task 1.2 - Exploratory Data Analysis (EDA)

STEP 01

Data Type Description

Identified data types for each column.

STEP 02

Basic Metrics

Generated descriptive statistics.

STEP 03

Missing Values Handling

Filled missing values with mean (numeric) and mode (categorical).

STEP 04

Univariate Analysis

Calculated dispersion parameters (std, variance, skewness).

STEP 05

Graphical Univariate Analysis

Histograms: Showed distribution shapes (e.g., right-skewed for total download data), indicating most users have lower download volumes.

STEP 06

Bivariate Analysis

Scatter Plots: Examined relationships (e.g., Social Media vs. total download data) showing positive correlations, indicating increased usage leads to higher data consumption.

STEP 07

Variable Transformations

Segmented session durations into deciles, revealing patterns in data usage.

STEP 08

Correlation Analysis

Created a correlation matrix showing strong relationships (e.g., Social Media and YouTube), useful for targeting marketing strategies.

STEP 09

Dimensionality Reduction (PCA)

Analyzed variance explained by components, indicating key factors in user behaviour.

STEP 10

Handset Analysis

Identified top handsets and manufacturers & Successfully saved the cleaned data for future use.

Task 2: User Engagement Analysis

Loading the Pre-Processed data saved as Excel File in Task 1

```python
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Step 2: Loading the Data saved in Task 1
df = pd.read_excel(r"pre-processed_data.xlsx")

# Show all columns
pd.set_option('display.max_columns', None)
# Set to allow auto-fit
pd.set_option('display.max_colwidth', None)
# Prevent scientific notation
pd.set_option('display.float_format', '{:.0f}'.format)

# Displaying first few rows
df.head()
```

	Bearer Id	Start	Start ms	End	End ms	Dur. (ms)	IMSI	MSISDN/Number	IMEI	Last Location Name	Avg RTT DL (ms)	Avg RTT UL (ms)	Avg Bearer TP DL (kbps)
0	13114483460844900352	2019-04-04 12:01:18	770	2019-04-25 14:35:31	662	1823652	208201448079117	33664962239	35521209507511	9164566995485190	42	5	23
1	13114483482878900224	2019-04-09 13:04:04	235	2019-04-25 08:15:48	606	1365104	208201909211140	33681854413	35794009006359	L77566A	65	5	16
2	13114483484080500736	2019-04-09 17:42:11	1	2019-04-25 11:58:13	652	1361762	208200314458056	33760627129	35281510359387	D42335A	110	18	6
3	13114483485442799616	2019-04-10 00:31:25	486	2019-04-25 07:36:35	171	1321509	208201402342131	33750343200	35356610164913	T21824A	110	18	44
4	13114483499480700928	2019-04-12 20:10:23	565	2019-04-25 10:40:32	954	1089009	208201401415120	33699795932	35407009745539	D88865A	110	18	6

01

Loading the Data

Used pd.read_excel(r"pre-processed_data.xlsx") to load the dataset.

02

Data Display Settings

pd.set_option('display.max_columns', None) to show all columns.

pd.set_option('display.max_colwidth', None) for auto-fit.

pd.set_option('display.float_format', '{:.0f}'.format) to prevent scientific notation.

03

Initial Data Exploration

- Displayed first few rows with df.head().
- Checked data types using df.dtypes.

04

Data Overview

Data Types: Confirmed the structure, including key metrics such as:

Avg Bearer TP DL (kbps): float64

Avg Bearer TP UL (kbps): float64

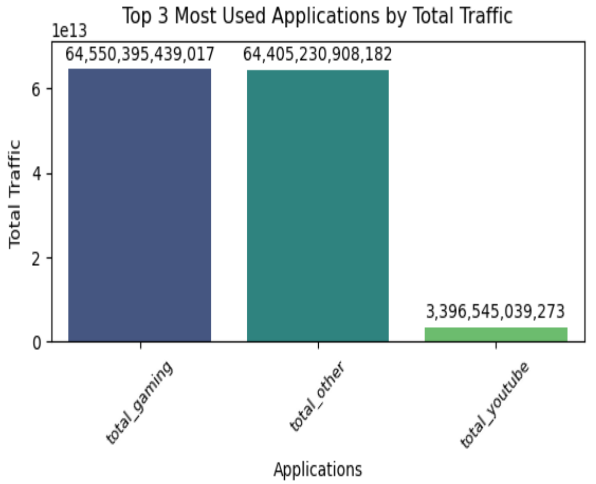
TCP DL Retrans. Vol (Bytes): float64

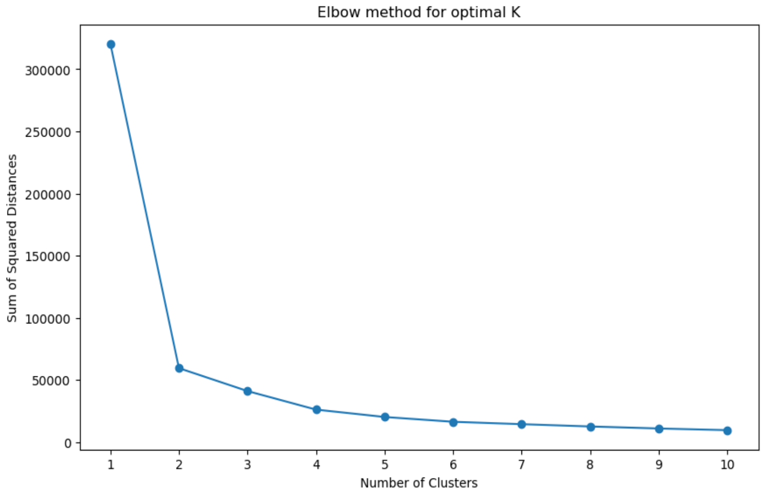
Task 2: User Engagement Analysis

Task 2.1 – Engagement Metrics Analysis

Aggregate metrics per customer ID (MSISDN) and report the top 10 customers per engagement metric

```python
import pandas as pd

def engagement_metrics_analysis(df):
    """Analyzes user engagement metrics."""

    # Aggregate user data
    user_engagement_data = df.groupby('MSISDN/Number').agg(
        session_frequency=('Bearer Id', 'count'),
        session_duration=('Dur. (ms)', 'sum'),
        total_dl=('Total DL (Bytes)', 'sum'),
        total_ul=('Total UL (Bytes)', 'sum'),
        total_social_media_dl=('Social Media DL (Bytes)', 'sum'),
        total_social_media_ul=('Social Media UL (Bytes)', 'sum'),
        total_youtube_dl=('Youtube DL (Bytes)', 'sum'),
        total_youtube_ul=('Youtube UL (Bytes)', 'sum'),
        total_netflix_dl=('Netflix DL (Bytes)', 'sum'),
        total_netflix_ul=('Netflix UL (Bytes)', 'sum'),
        total_google_dl=('Google DL (Bytes)', 'sum'),
        total_google_ul=('Google UL (Bytes)', 'sum'),
        total_email_dl=('Email DL (Bytes)', 'sum'),
        total_email_ul=('Email UL (Bytes)', 'sum'),
        total_gaming_dl=('Gaming DL (Bytes)', 'sum'),
        total_gaming_ul=('Gaming UL (Bytes)', 'sum'),
        total_other_dl=('Other DL (Bytes)', 'sum'),
        total_other_ul=('Other UL (Bytes)', 'sum')
    ).reset_index()

    # Calculate total traffic and other metrics
    user_engagement_data['total_traffic'] = user_engagement_data['total_dl'] + user_engagement_data['total_ul']
    user_engagement_data['total_social_media'] = user_engagement_data['total_social_media_dl'] + user_engagement_data['total
    user_engagement_data['total_youtube'] = user_engagement_data['total_youtube_dl'] + user_engagement_data['total_youtube_u
    user_engagement_data['total_netflix'] = user_engagement_data['total_netflix_dl'] + user_engagement_data['total_netflix_u
    user_engagement_data['total_google'] = user_engagement_data['total_google_dl'] + user_engagement_data['total_google_ul']
    user_engagement_data['total_email'] = user_engagement_data['total_email_dl'] + user_engagement_data['total_email_ul']
    user_engagement_data['total_gaming'] = user_engagement_data['total_gaming_dl'] + user_engagement_data['total_gaming_ul']
    user_engagement_data['total_other'] = user_engagement_data['total_other_dl'] + user_engagement_data['total_other_ul']

    # Top 10 users per metric
    print("Top 10 Customers for Engagement Metrics:")
    for metric in ['session_frequency', 'session_duration', 'total_traffic']:
        top_10 = user_engagement_data[metric].nlargest(10)
        print(f"\nTop 10 by {metric}:\n", top_10)

    return user_engagement_data  # Return the DataFrame for further processing

user_engagement_data = engagement_metrics_analysis(df)
```





01

User Groups

I grouped users into three categories based on how they use the platform:

- **Low Engagement:** Some users hardly use the platform.
- **Highly Engaged:** A small group of users is very active, with up to 1066 sessions.
- **Moderate Engagement:** Most users fall somewhere in between.

02

Traffic by Application

I looked at how much data different applications use and found:

- **Gaming:** This app uses about 64 trillion bytes of data.
- **Other Applications:** Also around 64 trillion bytes.
- **YouTube:** This one uses about 3.4 trillion bytes.

03

Most Used Applications

The top three applications that users engage with the most are:

- **Total Gaming**
- **Total Other**
- **Total YouTube**

Gaming is clearly the favourite among users.

04

Optimal Clusters

To figure out how many user groups to create, I used a method called the elbow method. It showed that three groups make the most sense, as there's not much difference when trying to add more groups.

05

Graphical Insights

Engagement Clusters - Scatter Plot: I created a scatter plot that shows how total session time relates to total data usage. It clearly shows the three user groups I identified.

Top Applications - Bar Chart: I made a bar chart that displays the total data used by the most popular applications. It highlights how much more gaming and other applications are used compared to YouTube.

Elbow Method - Line Graph: I used a line graph to show the sum of squared distances for different numbers of user groups. It visually confirms that three groups is the best choice.

Task 3: Experience Analytics

Task 3.1 - Aggregate Experience Metrics | Task 3.2 - Top and Bottom Metrics | Task 3.3 - Distribution Analysis | Task 3.4 - Clustering User Experiences

01 Cleaned the Data

- Fixed missing information by filling in gaps with average values for numbers and the most common type for names.
- Found and limited extreme values to keep the data accurate.

02 Combined Data

- Grouped information by customer to get average performance metrics like data retransmission and speed.

03 Analyzed Metrics

- Looked at the best and worst performance numbers, as well as the most common values.

04 Created Visuals

- Made bar charts to show average speeds and data retransmission for the top 10 phone types, making it easier to see differences.

05 Grouped Users

- Used a simple method to categorize users based on their performance metrics, helping to identify different user experiences.

06 Saved the Data

- Exported the cleaned and combined data into CSV files for future use.

07 Average Throughput Graphical Insights

- Top Performer: Huawei B715S-23C: 89,088.18
- General Range: Most handsets around 80,000.
- Low Performers: Several models (e.g., Spa Condor) near 60,000.

08 Average TCP Retransmission Insights

- Highest: Huawei B715S-23C: 53,486,938.00
- High Counts: Other models (Huawei Pele, Y9 2019) also show significant retransmissions.
- Lowest: Zyxel Communicat. Sbg3600: 44,508.00.

Task 4: Satisfaction Analysis

Task 4.1 - Assign Scores | Task 4.2 - Satisfaction Score

	MSISDN/Number	avg_tcp_retransmission	avg_rtt	handset_type	avg_throughput	experience_cluster
0	33601001722	21569573	46	Huawei P20 Lite Huawei Nova 3E	76	0
1	33601001754	21569573	31	Apple iPhone 7 (A1778)	99	0
2	33601002511	21569573	127	undefined	97	0
3	33601007832	760725	84	Apple iPhone 5S (A1457)	248	1
4	33601008617	15470202	60	Apple iPhone Se (A1723)	26152	1
5	33601010682	11165996	76	Samsung Galaxy A8 (2018)	3954	0
6	33601011634	10839902	26	Huawei Mate 10 Pro Porsche Design Huawei Mate 10	21256	1
7	33601011959	759937	52	Samsung Galaxy S8 Plus (Sm-G955F)	1247	1
8	33601014694	21569573	127	undefined	94	0
9	33601020306	20811208	62	Apple iPhone X (A1865)	146	0

```
Engagement DataFrame:
   MSISDN/Number  session_frequency  session_duration  total_traffic  cluster
0    33601001722                  1            116720      878690574        0
1    33601001754                  1            181230      156859643        0
2    33601002511                  1            134969      595966483        0
3    33601007832                  1             49878      422320698        0
4    33601008617                  2             37104     1457410944        0

Experience DataFrame:
   MSISDN/Number  avg_tcp_retransmission  avg_rtt  avg_throughput  experience_cluster
0    33601001722                21569573       46              76                   0
1    33601001754                21569573       31              99                   0
2    33601002511                21569573      127              97                   0
3    33601007832                  760725       84             248                   1
4    33601008617                15470202       60           26152                   1
```

```
          MSISDN/Number  engagement_score  experience_score
0           33601001722         203595411            523472
1           33601001754         518235523            523472
2           33601002511          79128681            523472
3           33601007832         252774480          21332315
4           33601008617         782315786           6622880
...                 ...               ...               ...
106852      33789997247         194787829            523472
106853      41882819545      531069619018          12077814
106854    3197020876596         442971815            523473
106855  337000037000919          78807429            523473
106856  882397108489451         535942022            523473

[106857 rows x 3 columns]
```

```
Top 10 Satisfied Customers:
        MSISDN/Number  satisfaction_score
106853    41882819545        265540848416
6437      33614892860          4092661688
92923     33760536639          3922548365
13180     33625779332          3915608881
13526     33626320676          3652505970
76363     33675877202          3613476583
37052     33659725664          3523629391
63028     33666464084          3321795709
92577     33760413819          3234885129
57241     33664712899          3104326093
```

Data Preparation

Created df_engagement: Extracted relevant columns from the engagement data.

Created df_experience: Extracted relevant columns from the experience data.

Function Definition

- Defined assign_scores Function
- Identified Clusters
- Calculated Averages
- Assigned Scores
- Merged Scores

Function Execution

- Called assign_scores and printed the resulting DataFrame, handling any errors that occurred.
- Error Handling: Included error handling to catch and print any issues during execution.

Calculating Satisfaction Score

Added a new column, satisfaction_score, using the formula: satisfaction_score = (engagement_score + experience_score) / 2

Identifying Top Customers

Extracted the top 10 customers with the highest satisfaction scores.

Task 4: Satisfaction Analysis

Task 4.3 - Predicting Satisfaction

Task 4.3 - Predicting Satisfaction

```
# Check the columns in the DataFrame
print("Columns in clustered_df:", updated_df.columns)

# Strip any leading or trailing whitespace from column names
updated_df.columns = updated_df.columns.str.strip()

# Re-check the columns after stripping
print("Columns after stripping:", updated_df.columns)
```

```
Columns in clustered_df: Index(['MSISDN/Number', 'engagement_score', 'experience_score', 'satisfaction_score'], dtype='object')
Columns after stripping: Index(['MSISDN/Number', 'engagement_score', 'experience_score', 'satisfaction_score'], dtype='object')
```

```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor

# Define features and target
X = updated_df[['engagement_score', 'experience_score']]
y = updated_df['satisfaction_score']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and fit the model
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"\nModel RMSE: {rmse:.2f}")

# Cross-validation scores
cv_scores = cross_val_score(model, X, y, cv=5, scoring='neg_root_mean_squared_error')
print(f"\nCross-validated RMSE: {-cv_scores.mean():.2f}")

# Inspect predictions
comparison_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred,
    'Difference': y_test - y_pred,
    'Absolute Difference': abs(y_test - y_pred)
})
print(comparison_df)
```

```
Model RMSE: 758662.03

Cross-validated RMSE: 359021237.21
          Actual  Predicted  Difference  Absolute Difference
21377   65473889   65522767      -48878                48878
22356  280635546  280609343       26203                26203
42414   19178307   19091523       86784                86784
95160  435144978  434311450      833528               833528
61074   40292442   40272947       19494                19494
...          ...        ...         ...                  ...
57023  279638926  279820399     -181474               181474
39594  279988165  279988894        -728                  728
36906  279070593  279071959       -1367                 1367
16701  627913549  628337911     -424361               424361
92311   24415385   24416816       -1431                 1431

[21372 rows x 4 columns]
```

```
from sklearn.metrics import r2_score

# Calculate R² score
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2:.2f}")
```

```
R² Score: 1.00
```

Saving the Model to a Pickle File for Future use

```
import pickle

# Save the model to a pickle file
with open('random_forest_model.pkl', 'wb') as file:
    pickle.dump(model, file)

print("Model saved to random_forest_model.pkl")
```

```
Model saved to random_forest_model.pkl
```

01 **Data Preparation**

Checked and stripped whitespace from the DataFrame column names.

02 **Feature and Target Definition**

Defined features (X) as engagement_score & experience_score.

Defined the target variable (y) as satisfaction_score.

03 **Data Splitting**

Split the data into training and testing sets (80% train, 20% test).

04 **Model Creation and Training**

Created a Random Forest Regressor model and fitted it to the training data.

05 **Predictions**

Made predictions on the test set.

Model RMSE: 758,662.03 | **Cross-Validated RMSE:** 359,021,237.21

06 **Model Evaluation**

Calculated RMSE, Cross-validated through positive branding.

Generated a comparison DataFrame of actual vs. predicted values.

07 **R^2 Score Calculation**

Calculated the R^2 score to assess the model's performance.

R^2 Score: 1.00 (indicating perfect fit)

08 **Model Saving**

Saved the trained model to a pickle file for future use.

Task 4: Satisfaction Analysis

Task 4.4 - Clustering Satisfaction and Experience | Task 4.5 - Aggregate Scores per Cluster

Task 4.4 - Clustering Satisfaction and Experience

```python
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

def cluster_satisfaction_experience(df):
    """Clusters users based on satisfaction and experience scores."""

    # Clustering based on k-means using engagement and experience
    scaler = StandardScaler()
    scaled_scores = scaler.fit_transform(df[['engagement_score', 'experience_score']])

    kmeans = KMeans(n_clusters=2, random_state=42)
    df['satisfaction_cluster'] = kmeans.fit_predict(scaled_scores)

    # Print the first 20 cluster values
    print(f"\nCluster values:\n {df[['MSISDN/Number', 'satisfaction_cluster']].head(20)}")

    return df

# Now call the clustering function
clustered_df = cluster_satisfaction_experience(updated_df)
```

```
Cluster values:
     MSISDN/Number  satisfaction_cluster
0      33601001722                     0
1      33601001754                     0
2      33601002511                     0
3      33601007832                     0
4      33601008617                     0
5      33601010682                     0
6      33601011634                     0
7      33601011959                     0
8      33601014694                     0
9      33601020306                     0
10     33601021045                     0
11     33601021217                     0
12     33601022743                     0
13     33601024291                     0
14     33601025738                     0
15     33601026147                     0
16     33601027208                     0
17     33601031129                     0
18     33601032846                     0
19     33601032987                     0
```

```python
print(updated_df.head())
```

	MSISDN/Number	engagement_score	experience_score	satisfaction_score	satisfaction_cluster
0	33601001722	203595411	523472	102059442	0
1	33601001754	518235523	523472	259379498	0
2	33601002511	79128681	523472	39826077	0
3	33601007832	252774480	21332315	137053398	0
4	33601008617	782315786	6622880	394469333	0

Task 4.5 - Aggregate Scores per Cluster

```python
def aggregate_scores_per_cluster(df):
    """Aggregates average satisfaction and experience scores per cluster."""

    # Group by satisfaction cluster and calculate mean scores
    cluster_scores = df.groupby('satisfaction_cluster')[['satisfaction_score', 'experience_score']].mean()

    # Print the average scores per cluster
    print("\nAverage satisfaction and experience scores per cluster:\n\n", cluster_scores)

    return cluster_scores

# Now call the aggregation function
average_scores = aggregate_scores_per_cluster(clustered_df)
```

Average satisfaction and experience scores per cluster:

satisfaction_cluster	satisfaction_score	experience_score
0	174476438	8365504
1	265540848416	12077814

Function Creation

Developed a function to perform clustering.

Data Standardization

Used StandardScaler to normalize the engagement_score and experience_score for better accuracy.

K-Means Application

Implemented K-Means with 2 clusters to categorize users.

Function Development

Created a function to aggregate scores by cluster.

Aggregation

Used groupby to calculate the mean satisfaction and experience scores for each cluster.

Task 4: Satisfaction Analysis

Task 4.6 - Export to MySQL

```python
from sqlalchemy import create_engine, text

# Define SQL Server connection details
server = 'DEB\\SQLEXPRESS'
database = 'Debais_Baidya'
username = 'Debasis'
password = '12345'

# Create a connection string
connection_url = f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server"

# Create an engine
engine = create_engine(connection_url)

# Create Table
create_table_query = text("""
IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='user_scores' AND xtype='U')
BEGIN
CREATE TABLE user_scores (
    [MSISDN/Number] VARCHAR(255) PRIMARY KEY,
    engagement_score FLOAT,
    experience_score FLOAT,
    satisfaction_score FLOAT,
    satisfaction_cluster INT
)
END
""")
```

```python
# Assuming you have a connection object created from the engine
with engine.connect() as connection:
    connection.execute(create_table_query)

    # Prepare insert query
    insert_query = text("""
    MERGE INTO user_scores AS target
    USING (SELECT :msisdn AS [MSISDN/Number], :engagement AS engagement_score,
                  :experience AS experience_score, :satisfaction AS satisfaction_score,
                  :cluster AS satisfaction_cluster) AS source
    ON target.[MSISDN/Number] = source.[MSISDN/Number]
    WHEN MATCHED THEN
        UPDATE SET
            target.engagement_score = source.engagement_score,
            target.experience_score = source.experience_score,
            target.satisfaction_score = source.satisfaction_score,
            target.satisfaction_cluster = source.satisfaction_cluster
    WHEN NOT MATCHED THEN
        INSERT ([MSISDN/Number], engagement_score, experience_score, satisfaction_score, satisfaction_cluster)
        VALUES (source.[MSISDN/Number], source.engagement_score, source.experience_score, source.satisfaction_score, source.satisfaction_cluster);
    """)
```

```python
    # Insert data using MERGE
    for _, row in updated_df.iterrows():
        # Execute the insert query with parameters as a dictionary
        connection.execute(insert_query, {
            'msisdn': str(row['MSISDN/Number']),  # Ensure it's a string
            'engagement': row['engagement_score'],
            'experience': row['experience_score'],
            'satisfaction': row['satisfaction_score'],
            'cluster': row['satisfaction_cluster']
        })

    connection.commit()  # Commit once after all inserts

print("Data exported to SQL Server successfully.")

# Dispose the engine
engine.dispose()
```

```
Data exported to SQL Server successfully.
```

```python
# Check if the table exists
check_table_query = text("""
SELECT *
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME = 'user_scores';
""")

with engine.connect() as connection:
    result = connection.execute(check_table_query)
    table_info = result.fetchall()

if table_info:
    print("Table 'user_scores' exists.")
else:
    print("Table 'user_scores' does not exist.")
```

```
Table 'user_scores' exists.
```

```python
# Describe the table structure
describe_table_query = text("""
SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'user_scores';
""")

with engine.connect() as connection:
    result = connection.execute(describe_table_query)
    columns_info = result.fetchall()

print("Table 'user_scores' structure:")
for column in columns_info:
    print(f"Column: {column.COLUMN_NAME}, Type: {column.DATA_TYPE}, Length: {column.CHARACTER_MAXIMUM_LENGTH}")
```

```
Table 'user_scores' structure:
Column: MSISDN/Number, Type: varchar, Length: 255
Column: engagement_score, Type: float, Length: None
Column: experience_score, Type: float, Length: None
Column: satisfaction_score, Type: float, Length: None
Column: satisfaction_cluster, Type: int, Length: None
```

01 **Install Required Libraries**

Utilized sqlalchemy and pyodbc for database connectivity.

02 **Connection Setup**

Defined connection details for SQL Server and created a connection string.

03 **Table Creation**

Executed a query to create the user_scores table, checked if it did not already exist.

04 **Data Insertion**

Used a MERGE statement to insert or update user scores based on the existing records in the table.

04 **user_scores table in SQL Server**

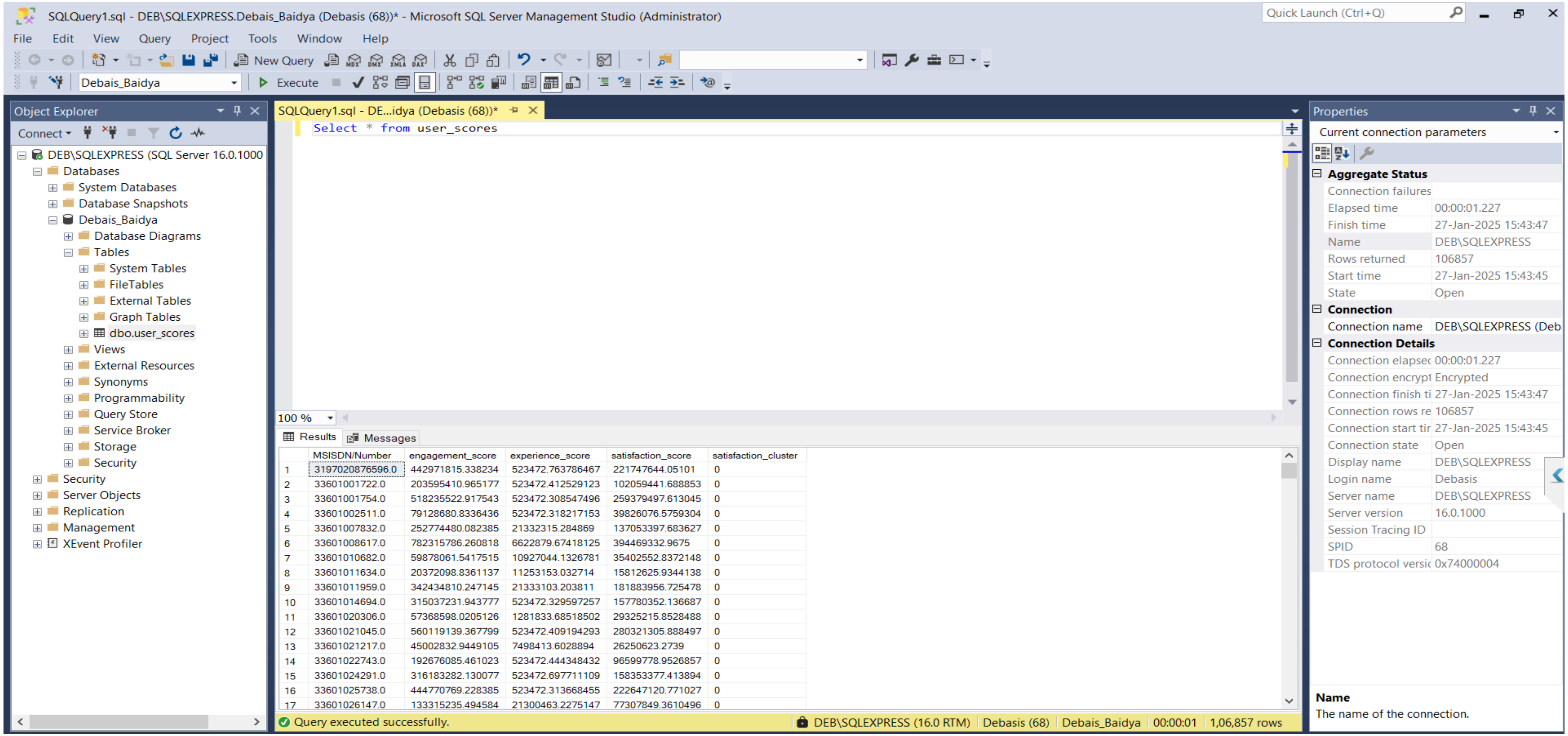
Successfully exported data to SQL Server.

04 **Verified Table Existence**

Confirmed that the user_scores table exists.

04 **Table Structure**

Retrieved and displayed the column names, data types, and lengths, ensuring the table structure is as expected.



1 Executed Query in SQL SSMS

Executed "Select * from user_scores" table to check if it populates correctly.

2 Results Appeared in Output Pane

Data in 'users_scores' table appeared with All the columns present in updated_df DataFrame

Streamlit Setting Up & App Testing

1

Streamlit.py File Execution with Pickle

Used Pickle module to load the model random_forest_model.pkl & set it up for Streamlit Satisfaction Prediction Dashboard

1

Executed Streamlit Run Command from Terminal

First copied path of my python file & then entered cd “C\Users\DEB\Project 5” to open the directory & Executed the ‘steamlit run streamlit.py’ in terminal window.

2

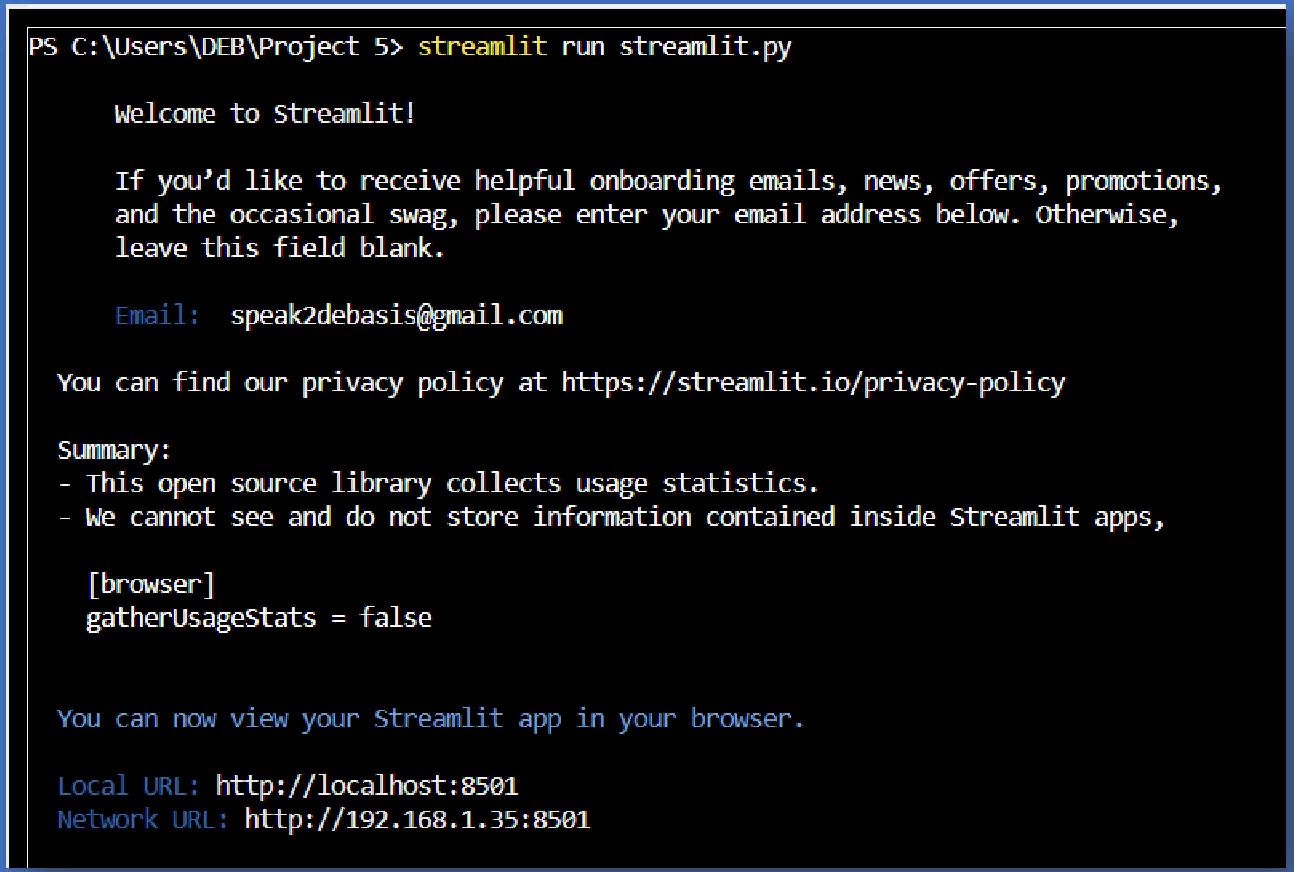
Local & Network URL to view Streamlit App

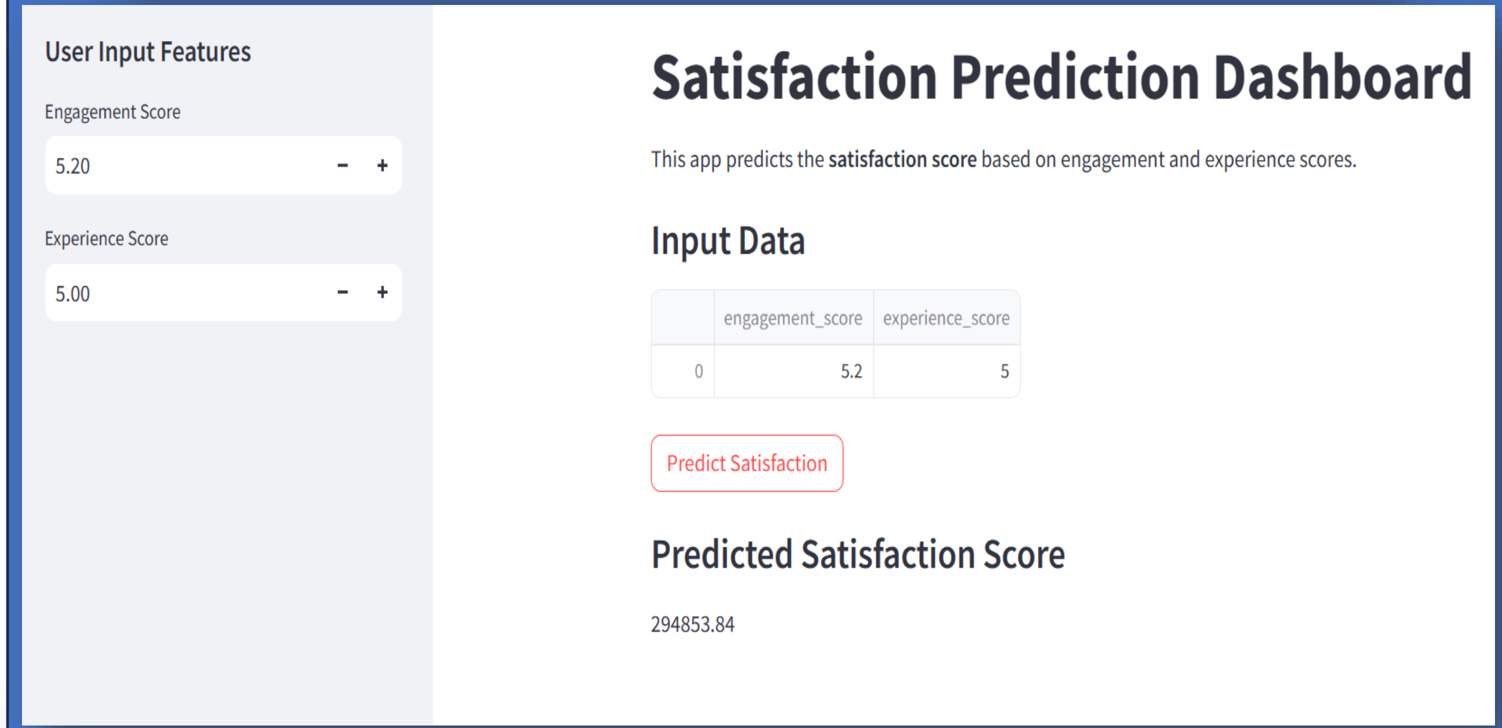
Got Local URL: http://localhost:8501 &
Network URL: http://192.168.1.35:8501 to go to Streamlit App in Web Browser.

3

User Input & Predict Satisfaction Button Checked

Gave 5.20 as ‘Engagement Score’ & 5.00 as Experience Score & pressed the button Predict Satisfaction to check the ‘Predicted Satisfaction Score’.





I hope you found this presentation helpful and informative.

Please don't hesitate to contact me with any questions or feedback you may have.

I am more than eager to hear your thoughts and suggestions.