GORGIAS USER ENGAGEMENT ANALYSIS

PYTHON SCRIPTS

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:
```
df_user_engagement = pd.read_csv('/Users/pierremora/Desktop/Gorgias/user_eng
```

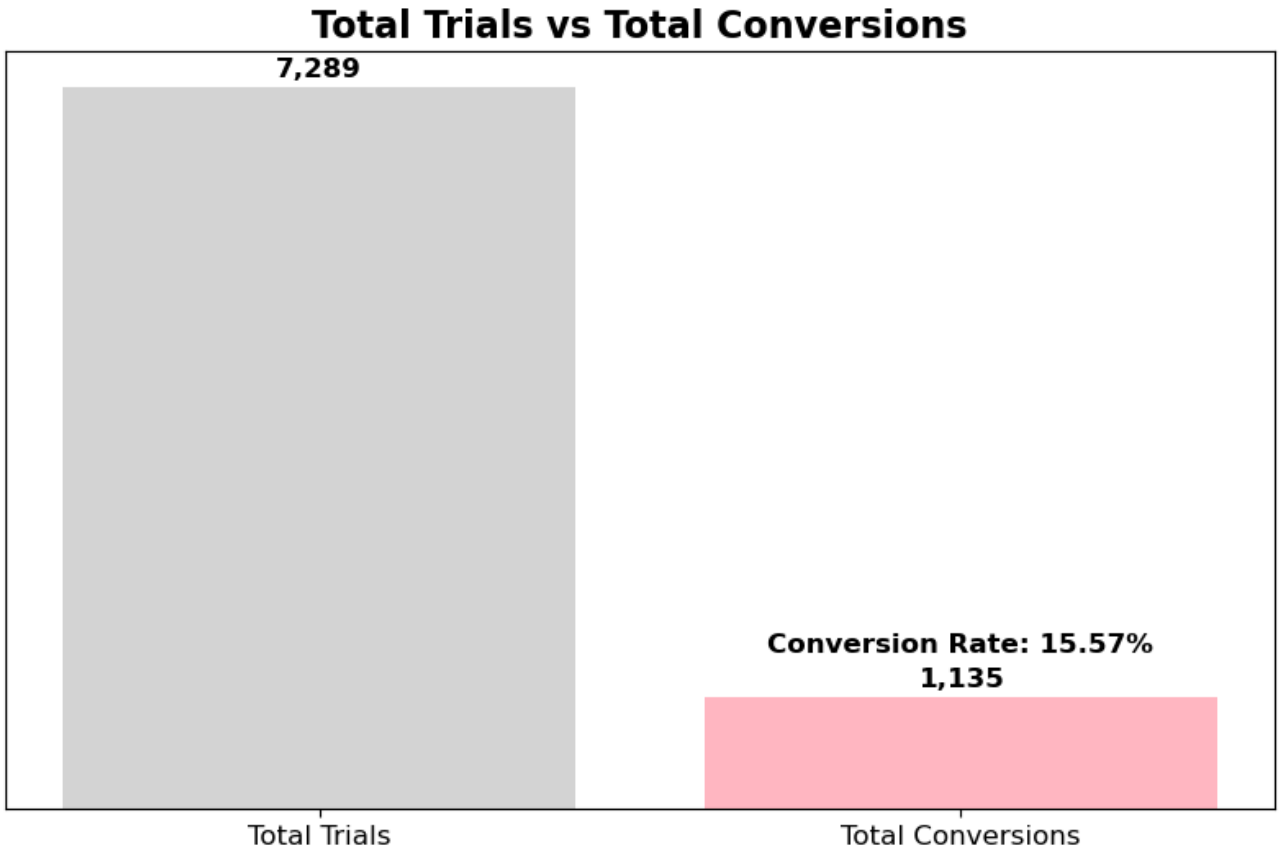
In [4]:
```
df_daily_engagement = pd.read_csv('/Users/pierremora/Desktop/Gorgias/daily_e
```

In [128…
```
df_total_conversions = pd.read_csv('/Users/pierremora/Desktop/Gorgias/total_
```

Creating a bar chart showing the difference between total trials and total conversions

In [135…
```
total_trials = df_total_conversions['total_trials'].values[0]
total_conversions = df_total_conversions['total_conversions'].values[0]
conversion_rate_percentage = df_total_conversions['conversion_rate_percentag

# Data for the bar chart
categories = ['Total Trials', 'Total Conversions']
values = [total_trials, total_conversions]

# Custom colors
grey_color = '#D3D3D3'  # Light grey for total trials
light_pink_color = '#FFB6C1'  # Light pink for total conversions
black_color = '#000000'  # Black for text

# Creating the bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(categories, values, color=[grey_color, light_pink_color])

# Annotate the bars with the actual numbers
plt.text(0, total_trials + 50, f'{total_trials:,}', ha='center', va='bottom'
plt.text(1, total_conversions + 50, f'{total_conversions:,}', ha='center', v

# Annotate with conversion rate even further above the total conversions num
plt.text(1, total_conversions + 400, f'Conversion Rate: {conversion_rate_per
         ha='center', va='bottom', fontsize=12, color=black_color, fontweigh

# Adding titles and labels
plt.title('Total Trials vs Total Conversions', fontsize=16, color=black_colo
plt.xticks(fontsize=12, color=black_color)
```

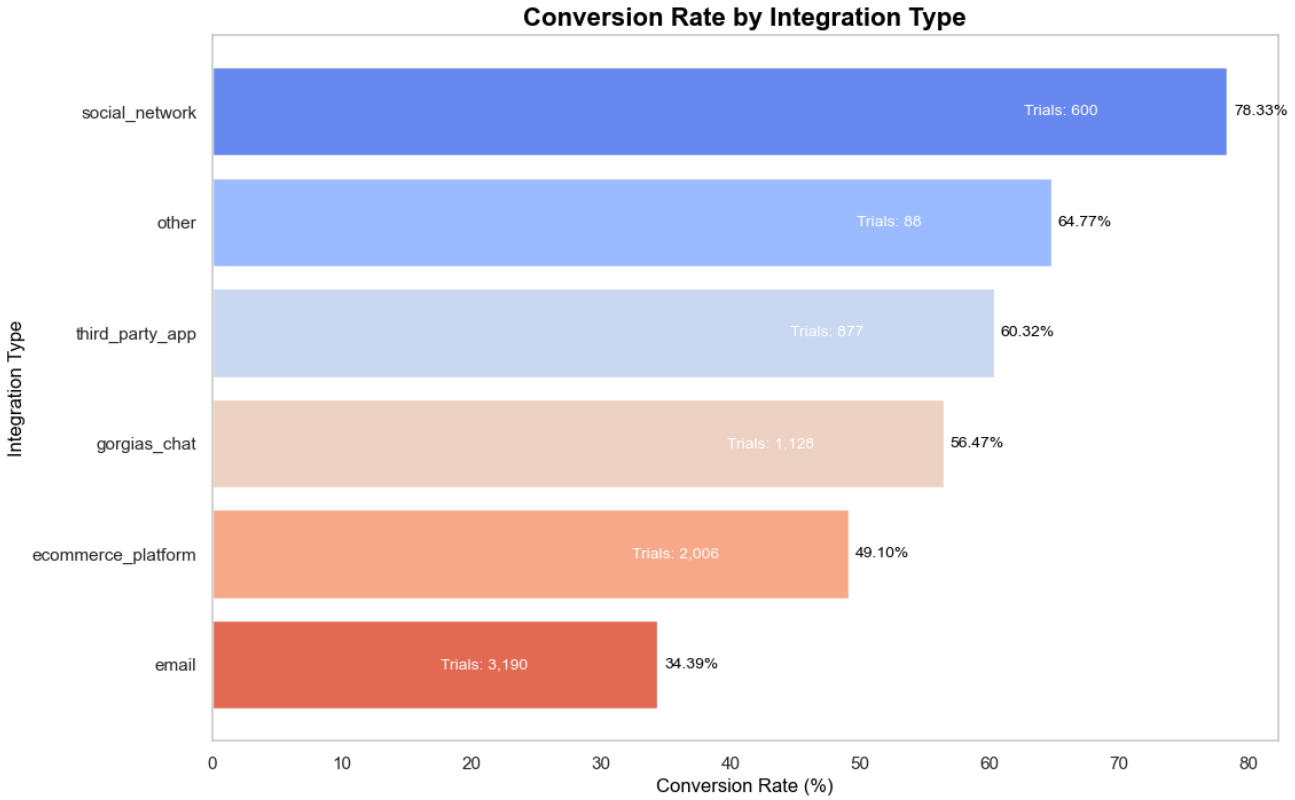
```
plt.yticks([])  # Remove the count numbers on the left

# Removing grid lines
plt.grid(False)

plt.show()
```



In [136…
```
df_integration_conversion = pd.read_csv('/Users/pierremora/Desktop/Gorgias/i
```

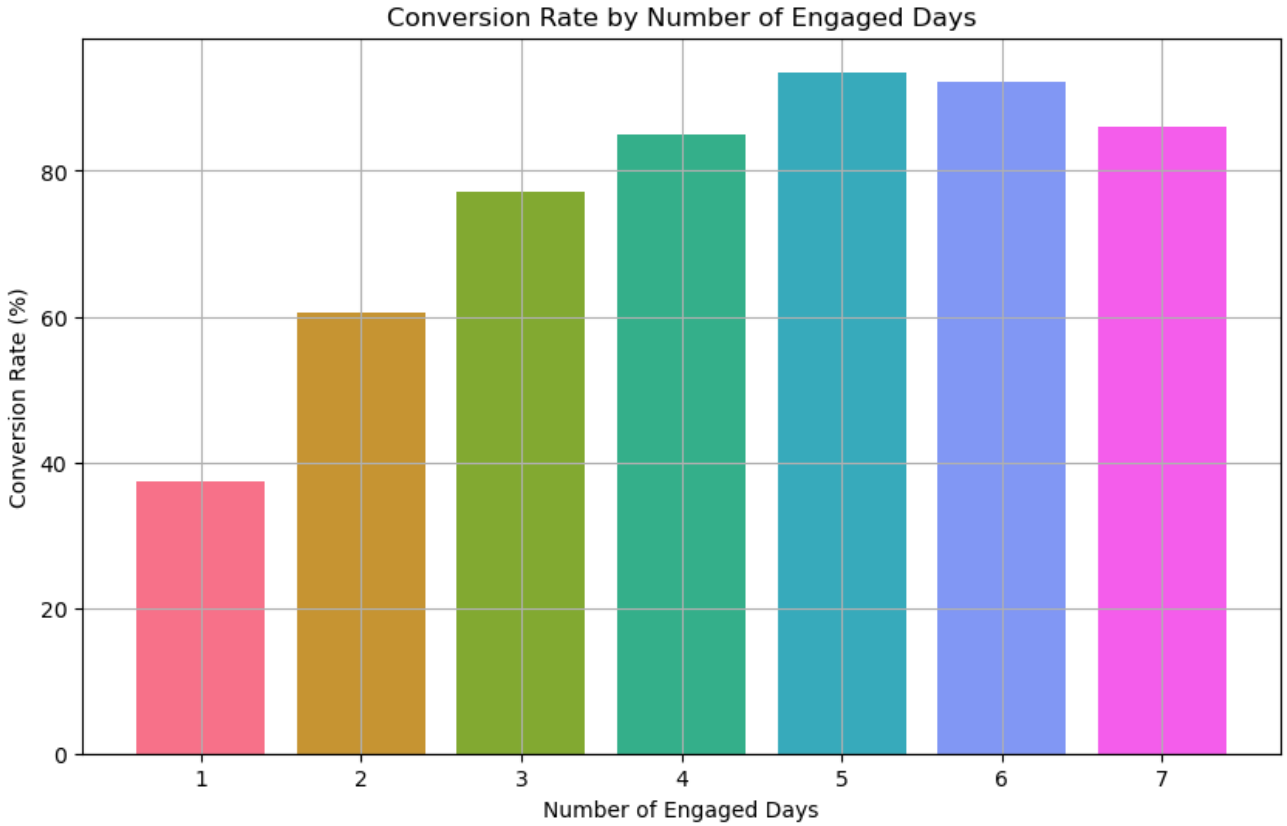
In [137…
```
# Sorting by conversion rate for better visualization
df_integration_conversion = df_integration_conversion.sort_values(by='conver

# Define the figure size and style
plt.figure(figsize=(12, 8))
sns.set(style="whitegrid")

# Create a horizontal bar chart
bars = plt.barh(df_integration_conversion['integration_type'], df_integratic
                color=sns.color_palette("coolwarm", len(df_integration_conve

# Add the conversion rate as annotations on the bars
for i in range(len(df_integration_conversion)):
    plt.text(df_integration_conversion['conversion_rate_with_integration'].v
             i,
             f"{df_integration_conversion['conversion_rate_with_integration'
```

```
            va='center',
            fontsize=10,
            color='black')

# Add total trials annotations on the bars
for i in range(len(df_integration_conversion)):
    plt.text(df_integration_conversion['conversion_rate_with_integration'].v
            i,
            f"Trials: {df_integration_conversion['total_trials_with_integra
            va='center',
            ha='right',
            fontsize=10,
            color='white')

# Add titles and labels
plt.title('Conversion Rate by Integration Type', fontsize=16, color='black',
plt.xlabel('Conversion Rate (%)', fontsize=12, color='black')
plt.ylabel('Integration Type', fontsize=12, color='black')

# Invert y-axis to have the highest rate at the top
plt.gca().invert_yaxis()

# Remove grid lines
plt.grid(False)

# Show the plot
plt.show()
```



In [7]:
```
# Analyzing conversion rate based on engaged days
conversion_by_engaged_days = df_user_engagement.groupby('engaged_days')['cor
```

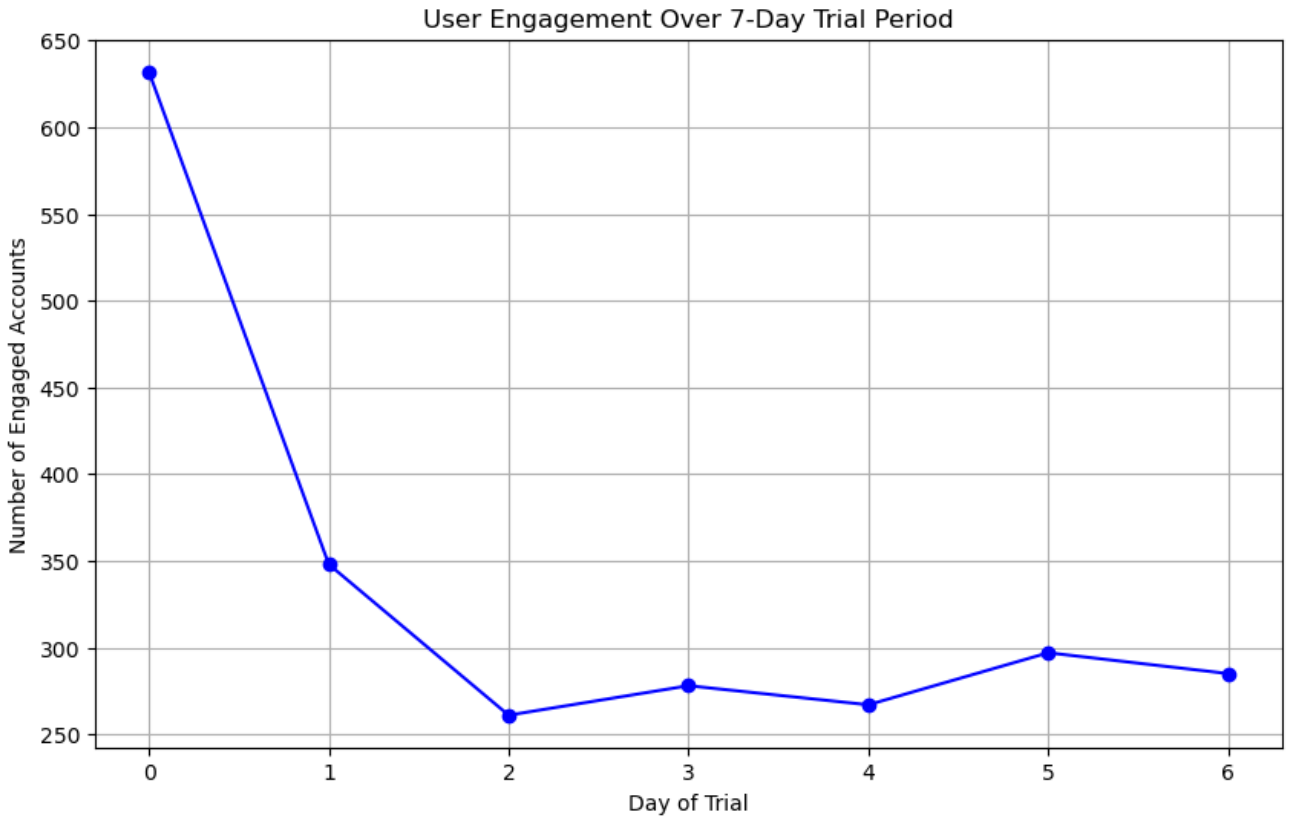
In [13]:
```
import seaborn as sns

colors = sns.color_palette('husl', len(conversion_by_engaged_days))  # Gener

#Visualizing conversion rate by number of engaged days with custom colors
plt.figure(figsize=(10, 6))
plt.bar(conversion_by_engaged_days['engaged_days'], conversion_by_engaged_da
plt.title('Conversion Rate by Number of Engaged Days')
plt.xlabel('Number of Engaged Days')
plt.ylabel('Conversion Rate (%)')
plt.xticks(conversion_by_engaged_days['engaged_days'])
plt.grid(True)
plt.show()
plt.savefig('/Users/pierremora/Desktop/Gorgias/conversion_rate_by_engaged_da
```




```
<Figure size 640x480 with 0 Axes>
```

In [14]:
```
# Visualizing user engagement over time (already done previously)
plt.figure(figsize=(10, 6))
plt.plot(df_daily_engagement['day_of_trial'], df_daily_engagement['engaged_a
plt.title('User Engagement Over 7-Day Trial Period')
plt.xlabel('Day of Trial')
plt.ylabel('Number of Engaged Accounts')
```

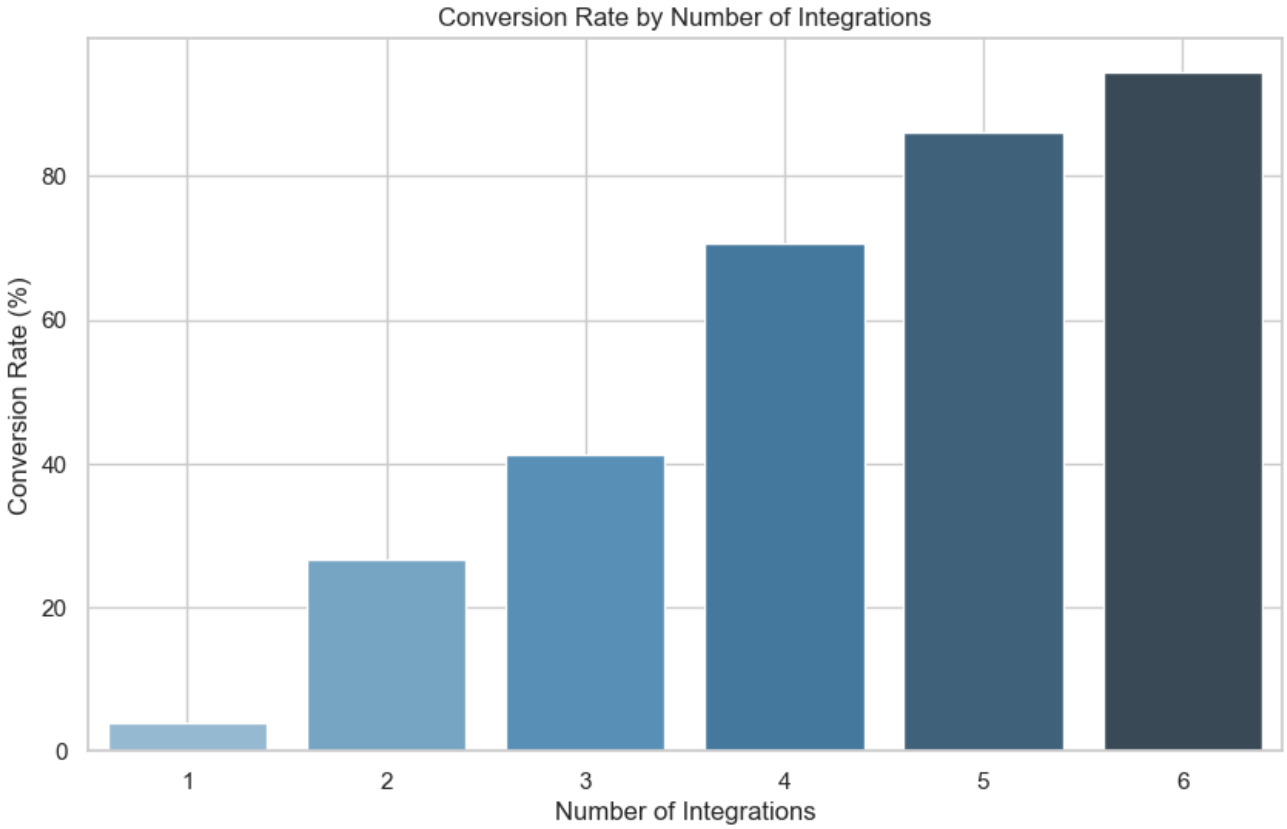
```
plt.grid(True)
plt.show()

plt.savefig('/Users/pierremora/Desktop/Gorgias/user_engagement_over_time.png
```




```
<Figure size 640x480 with 0 Axes>
```

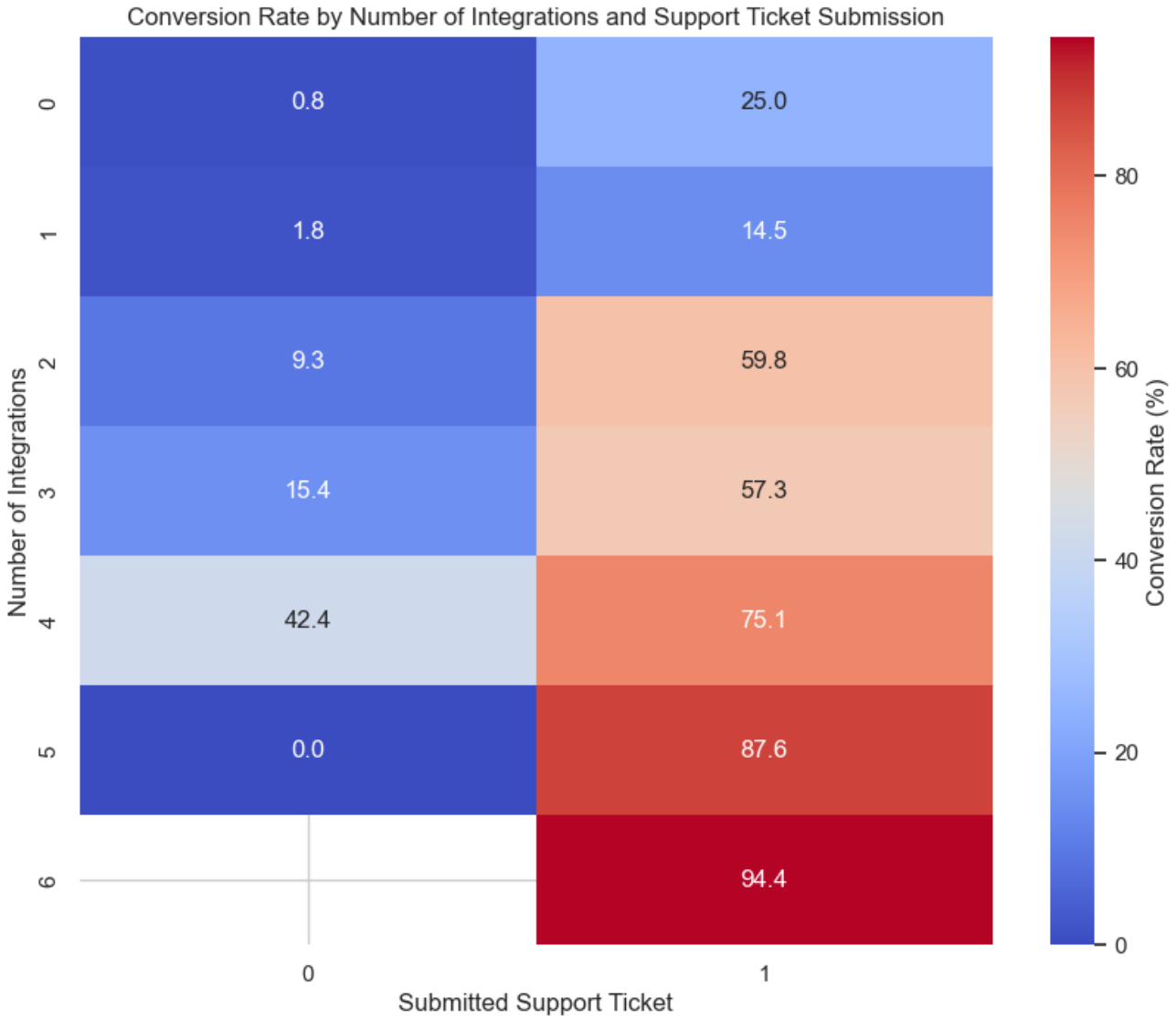
In [138…

```
df_integration_count = pd.read_csv('/Users/pierremora/Desktop/Gorgias/integr

# Calculating conversion rate based on the number of unique integrations set
conversion_by_integration_count = df_integration_count.groupby('number_of_ir

# Multiplying conversion rate by 100 to convert to percentage
conversion_by_integration_count['converted'] *= 100

# Plotting the conversion rate by number of integrations using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(x='number_of_integrations', y='converted', data=conversion_by_ir
plt.title('Conversion Rate by Number of Integrations')
plt.xlabel('Number of Integrations')
plt.ylabel('Conversion Rate (%)')
plt.grid(True)
plt.show()

plt.show()
```



In [140…

```
df_feature_usage_conversion = pd.read_csv('/Users/pierremora/Desktop/Gorgias

pivot_table = df_feature_usage_conversion.pivot_table(
    index='number_of_integrations',
    columns='submitted_support_ticket',
    values='converted',
    aggfunc='mean'
)

# Creating the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(pivot_table * 100, annot=True, fmt=".1f", cmap='coolwarm', cbar_
plt.title('Conversion Rate by Number of Integrations and Support Ticket Subm
plt.xlabel('Submitted Support Ticket')
plt.ylabel('Number of Integrations')
plt.show()
```



Section 2: How Does App Usage Impact Resolution Time, First Response Time, and Satisfaction Score?

In [92]:
```
df_actions = pd.read_csv('/Users/pierremora/Desktop/Gorgias/Question 2/actic
```

In [93]:
```
df_actions.dtypes
```

Out[93]:
```
id                                 object
status                             object
vitally_first_paid_date            object
churned_at                         object
plan                               object
arr                               float64
helpdesk_arr                      float64
voice_arr                         float64
sms_arr                           float64
automate_arr                      float64
convert_arr                       float64
billing_interval                   object
shopify_plan                       object
gmv_bucket                        float64
yearly_gross_merchandise_value    float64
industry_category                  object
enabled_apps                       object
support_performance_30d           float64
avg_first_response_time_30d       float64
avg_resolution_time_30d           float64
avg_satisfaction_score_30d        float64
dtype: object
```

In [94]:
```
# Inspect the first few rows of the enabled_apps column
print(df_actions['enabled_apps'].head(10))
```

```
0    []
1    []
2    []
3    []
4    []
5    []
6    []
7    []
8    []
9    []
Name: enabled_apps, dtype: object
```

In [95]:
```
import ast
import re

# Function to fix the formatting of the enabled_apps strings
def fix_formatting(apps_str):
    # Add quotes around each app name correctly without over-quoting
    fixed_str = re.sub(r'(\w[\w-]*)', r'"\1"', apps_str)
    return fixed_str

# Applyw   the formatting fix to the enabled_apps column
df_actions['enabled_apps'] = df_actions['enabled_apps'].apply(lambda x: fix_
```

```
# Now safely evaluate the strings to convert them to lists
df_actions['enabled_apps'] = df_actions['enabled_apps'].apply(lambda x: eval

# Verify the conversion
print(df_actions['enabled_apps'].head())
```

```
0    []
1    []
2    []
3    []
4    []
Name: enabled_apps, dtype: object
```

In [96]:
```
# Recalculating the num_enabled_apps feature
df_actions['num_enabled_apps'] = df_actions['enabled_apps'].apply(len)

# Verifying the recalculation
print(df_actions[['enabled_apps', 'num_enabled_apps']].head(10))
```

```
  enabled_apps  num_enabled_apps
0           []                 0
1           []                 0
2           []                 0
3           []                 0
4           []                 0
5           []                 0
6           []                 0
7           []                 0
8           []                 0
9           []                 0
```

In [97]:
```
# Filtering and display rows where enabled_apps is not an empty list
non_empty_apps = df_actions[df_actions['enabled_apps'].apply(lambda x: isins

# Displaying the first few rows of the non-empty apps
print(non_empty_apps[['enabled_apps', 'num_enabled_apps']].head(10))
```

```
                                          enabled_apps  num_enabled_apps
599                    [gmail-native, recharge-native]                 2
605                                     [smile-native]                 1
606  [channelreply, facebook-native, gmail-native, ...                 4
607  [conduit-1, facebook-native, gmail-native, gor...                 5
609                  [facebook-native, shopify-native]                 2
611          [facebook-native, optizo, shopify-native]                 3
612      [shopify-native, tiktok-shop-2, yotpo-native]                 3
613    [facebook-native, gmail-native, shopify-native]                 3
614                     [gmail-native, shopify-native]                 2
615                        [gmail-native, woocommerce]                 2
```

Handling Missing Values

First step is identifying the missing values in the dataset and then decide how to handle missing values

In [98]:
```
# Calculating the total number of missing values per column
missing_values = df_actions.isnull().sum()

# Calculating the percentage of missing values per column
missing_percentage = (missing_values / len(df_actions)) * 100

# Displaying the missing values and their percentages
missing_data = pd.DataFrame({'Missing Values': missing_values, 'Percentage':
print(missing_data.sort_values(by='Percentage', ascending=False))
```

```
                                Missing Values  Percentage
convert_arr                              23368   97.399133
sms_arr                                  20135   83.923808
voice_arr                                20117   83.848783
avg_satisfaction_score_30d               17149   71.477993
churned_at                               14287   59.549016
automate_arr                             11822   49.274758
avg_resolution_time_30d                  11326   47.207402
avg_first_response_time_30d              11267   46.961487
support_performance_30d                  10997   45.836112
shopify_plan                             10479   43.677059
yearly_gross_merchandise_value            9447   39.375625
industry_category                         6018   25.083361
gmv_bucket                                5904   24.608203
helpdesk_arr                              3284   13.687896
id                                           0    0.000000
enabled_apps                                 0    0.000000
billing_interval                             0    0.000000
status                                       0    0.000000
arr                                          0    0.000000
plan                                         0    0.000000
vitally_first_paid_date                      0    0.000000
num_enabled_apps                             0    0.000000
```

In [100…
```
# Selecting potential predictors
potential_predictors = [
    'plan', 'num_enabled_apps', 'industry_category',
    'support_performance_30d', 'arr', 'helpdesk_arr',
    'billing_interval'
]

# Encoding categorical variables to check correlations
df_encodedd = pd.get_dummies(df_actions[potential_predictors + ['avg_first_r

# Checking correlations
correlationss = df_encodedd.corr()['avg_first_response_time_30d'].sort_value
print(correlationss)
```

```
avg_first_response_time_30d                              1.000000
plan_starter                                             0.134905
plan_basic                                               0.065460
industry_category_Computer Software                      0.024780
industry_category_Entertainment                          0.021026
industry_category_Hospital & Health Care                 0.019000
industry_category_Design                                 0.016368
industry_category_Consumer Services                      0.014725
industry_category_Textiles                               0.008281
industry_category_Luxury Goods & Jewelry                 0.006912
industry_category_Electrical & Electronic Manufacturing  0.006460
industry_category_Information Technology & Services      0.006016
industry_category_Renewables & Environment               0.005607
industry_category_Non-profit Organization Management     0.005227
industry_category_Building Materials                     0.004687
industry_category_Hospitality                            0.001826
industry_category_Food & Beverages                       0.001506
industry_category_Furniture                              0.001436
industry_category_Music                                  0.001385
industry_category_Wholesale                              0.001319
industry_category_Retail                                 0.000928
industry_category_Health, Wellness & Fitness             0.000837
industry_category_Media Production                       0.000179
industry_category_Sporting Goods                        -0.000525
industry_category_Sports                                -0.000777
industry_category_Transportation/Trucking/Railroad      -0.001744
industry_category_Publishing                            -0.002671
plan_free                                               -0.004987
industry_category_Arts & Crafts                         -0.005039
industry_category_Professional Training & Coaching      -0.005829
industry_category_Internet                              -0.006128
industry_category_Medical Device                        -0.006189
industry_category_Food Production                       -0.006431
industry_category_Restaurants                           -0.007487
industry_category_Printing                              -0.009293
industry_category_Consumer Electronics                  -0.010222
industry_category_Marketing & Advertising               -0.014148
industry_category_Management Consulting                 -0.018175
industry_category_Veterinary                            -0.021238
industry_category_Automotive                            -0.026684
industry_category_Consumer Goods                        -0.030266
plan_enterprise                                         -0.046718
billing_interval_yearly                                 -0.075080
helpdesk_arr                                            -0.087922
plan_pro                                                -0.091547
arr                                                     -0.093734
num_enabled_apps                                        -0.140945
support_performance_30d                                 -0.360422
industry_category_Real Estate                                 NaN
Name: avg_first_response_time_30d, dtype: float64
```

Imputation: avg_first_response_time_30d

I will start with the imputation process for avg_first_response_time_30d using features like num_enabled_apps, plan, arr, and other available data.

In [102…
```
# Encoding categorical variables like 'plan' and 'billing_interval'
df_encoded = pd.get_dummies(df_actions, columns=['plan', 'billing_interval'])

# List the columns to verify
print(df_encoded.columns)
```

```
Index(['id', 'status', 'vitally_first_paid_date', 'churned_at', 'arr',
       'helpdesk_arr', 'voice_arr', 'sms_arr', 'automate_arr', 'convert_ar
r',
       'shopify_plan', 'gmv_bucket', 'yearly_gross_merchandise_value',
       'industry_category', 'enabled_apps', 'support_performance_30d',
       'avg_first_response_time_30d', 'avg_resolution_time_30d',
       'avg_satisfaction_score_30d', 'num_enabled_apps', 'plan_basic',
       'plan_enterprise', 'plan_free', 'plan_pro', 'plan_starter',
       'billing_interval_yearly'],
      dtype='object')
```

In [104…
```
# Check for missing values in the features used for imputation
missing_in_features = df_encoded[features_for_imputation].isnull().sum()
print(missing_in_features)
```

```
num_enabled_apps              0
arr                           0
helpdesk_arr               3284
plan_starter                  0
plan_basic                    0
plan_enterprise               0
billing_interval_yearly       0
dtype: int64
```

In [106…
```
# Correcting the typo in the variable name
features_for_imputation = [
    'num_enabled_apps', 'arr',
    'plan_starter', 'plan_basic', 'plan_enterprise',
    'billing_interval_yearly'
]

# Ensure you remove 'helpdesk_arr' if it's still in the list
if 'helpdesk_arr' in features_for_imputation:
    features_for_imputation.remove('helpdesk_arr')

# Proceed with the imputation process
from sklearn.linear_model import LinearRegression

# Filter the data to include only rows where avg_first_response_time_30d is
```

```
df_complete = df_encoded.dropna(subset=['avg_first_response_time_30d'])

X = df_complete[features_for_imputation]
y = df_complete['avg_first_response_time_30d']

# Train the regression model
model = LinearRegression()
model.fit(X, y)

# Predict the missing values
missing_idx = df_encoded[df_encoded['avg_first_response_time_30d'].isnull()]
df_encoded.loc[missing_idx, 'avg_first_response_time_30d'] = model.predict(d

# Verify the imputation
print(df_encoded['avg_first_response_time_30d'].isnull().sum())
```

```
0
```

Imputation: avg_response_time_30d

In [107…

```
# Defining the features to use for imputing avg_resolution_time_30d
features_for_imputation = [
    'num_enabled_apps', 'arr',
    'plan_starter', 'plan_basic', 'plan_enterprise',
    'billing_interval_yearly', 'avg_first_response_time_30d'
]

# Filtering the data to include only rows where avg_resolution_time_30d is n
df_complete = df_encoded.dropna(subset=['avg_resolution_time_30d'])

X = df_complete[features_for_imputation]
y = df_complete['avg_resolution_time_30d']

# Train the regression model
model = LinearRegression()
model.fit(X, y)

# Predicting the missing values
missing_idx = df_encoded[df_encoded['avg_resolution_time_30d'].isnull()].inc
df_encoded.loc[missing_idx, 'avg_resolution_time_30d'] = model.predict(df_er

# Verifying the imputation
print(df_encoded['avg_resolution_time_30d'].isnull().sum())
```

```
0
```

Imputing: support_performance_30d

Given that support_performance_30d is influenced by

avg_first_response_time and avg_resolution_time_30d, i focused on these two columns as main predictors for replacing missing values

In [109…
```
import numpy as np
# Defining the features to use for imputing support_performance_30d
features_for_imputation = [
    'avg_first_response_time_30d', 'avg_resolution_time_30d'
]

# Filtering the data to include only rows where support_performance_30d is n
df_complete = df_encoded.dropna(subset=['support_performance_30d'])

X = df_complete[features_for_imputation]
y = df_complete['support_performance_30d']

# Train the regression model
model = LinearRegression()
model.fit(X, y)

# Predicting the missing values
missing_idx = df_encoded[df_encoded['support_performance_30d'].isnull()].inc
predicted_values = model.predict(df_encoded.loc[missing_idx, features_for_im

# Clipping the predicted values to ensure they are between 1 and 5
predicted_values_clipped = np.clip(predicted_values, 1, 5)

# Imputing the clipped values back into the dataframe
df_encoded.loc[missing_idx, 'support_performance_30d'] = predicted_values_cl

# Verifying the imputation
print(df_encoded['support_performance_30d'].isnull().sum())
```

0

Dealing with satisfaction_score_30d

Given that satisfaction_score_30d column has 71% missing values, and i have already imputed the key variables, i needed an thoughtful approach on how to handle this imputation effectively.

In [111…
```
# Filter the DataFrame to include only numeric columns
numeric_df = df_encoded.select_dtypes(include=[np.number])

# Calculating the correlation matrix
corr_matrix = numeric_df.corr()

# Extract the correlations of avg_satisfaction_score_30d with other features
satisfaction_corr = corr_matrix['avg_satisfaction_score_30d'].sort_values(as
print(satisfaction_corr)
```

```
avg_satisfaction_score_30d        1.000000
support_performance_30d           0.069663
num_enabled_apps                  0.060152
yearly_gross_merchandise_value   -0.013385
voice_arr                        -0.024471
helpdesk_arr                     -0.044691
arr                              -0.044740
automate_arr                     -0.046949
gmv_bucket                       -0.048887
sms_arr                          -0.050966
avg_resolution_time_30d          -0.087646
convert_arr                      -0.096639
avg_first_response_time_30d      -0.161266
Name: avg_satisfaction_score_30d, dtype: float64
```

Note: Based on the correlation matrix, the following features show some level of corelation:

In [112…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import numpy as np

# Define the features for imputing avg_satisfaction_score_30d
features_for_imputation = [
    'support_performance_30d', 'num_enabled_apps',
    'avg_first_response_time_30d', 'avg_resolution_time_30d'
]

# Filter the data to include only rows where avg_satisfaction_score_30d is r
df_complete = df_encoded.dropna(subset=['avg_satisfaction_score_30d'])

X = df_complete[features_for_imputation]
y = df_complete['avg_satisfaction_score_30d']

# Split the data into training and test sets to validate the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Train the RandomForestRegressor
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Predict the missing values
missing_idx = df_encoded[df_encoded['avg_satisfaction_score_30d'].isnull()].
predicted_values = model.predict(df_encoded.loc[missing_idx, features_for_im

# Clip the predicted values to ensure they are within a valid range (typical
predicted_values_clipped = np.clip(predicted_values, 1, 5)

# Impute the clipped values back into the dataframe
df_encoded.loc[missing_idx, 'avg_satisfaction_score_30d'] = predicted_values
```

```
# Verify the imputation
print(df_encoded['avg_satisfaction_score_30d'].isnull().sum())
```

```
0
```

In [114…

```
from collections import defaultdict

# Initializing dictionaries to store the sum and count of metrics for each a
app_performance = defaultdict(lambda: {'resolution_time_sum': 0, 'response_t

# Iterating over each row and accumulate metrics by app
for _, row in df_encoded.iterrows():
    for app in row['enabled_apps']:
        app_performance[app]['resolution_time_sum'] += row['avg_resolution_t
        app_performance[app]['response_time_sum'] += row['avg_first_response
        app_performance[app]['satisfaction_sum'] += row['avg_satisfaction_sc
        app_performance[app]['count'] += 1

# Calculating average metrics for each app
app_avg_metrics = {app: {
                        'avg_resolution_time': values['resolution_time_sum']
                        'avg_first_response_time': values['response_time_sum
                        'avg_satisfaction_score': values['satisfaction_sum']
                    } for app, values in app_performance.items()}

# Converting to DataFrame for easier analysis
app_metrics_df = pd.DataFrame.from_dict(app_avg_metrics, orient='index').res
app_metrics_df.columns = ['app', 'avg_resolution_time', 'avg_first_response_

# Displaying the top apps by average satisfaction score
top_apps_by_satisfaction = app_metrics_df.sort_values(by='avg_satisfaction_s
print(top_apps_by_satisfaction.head(10))
```

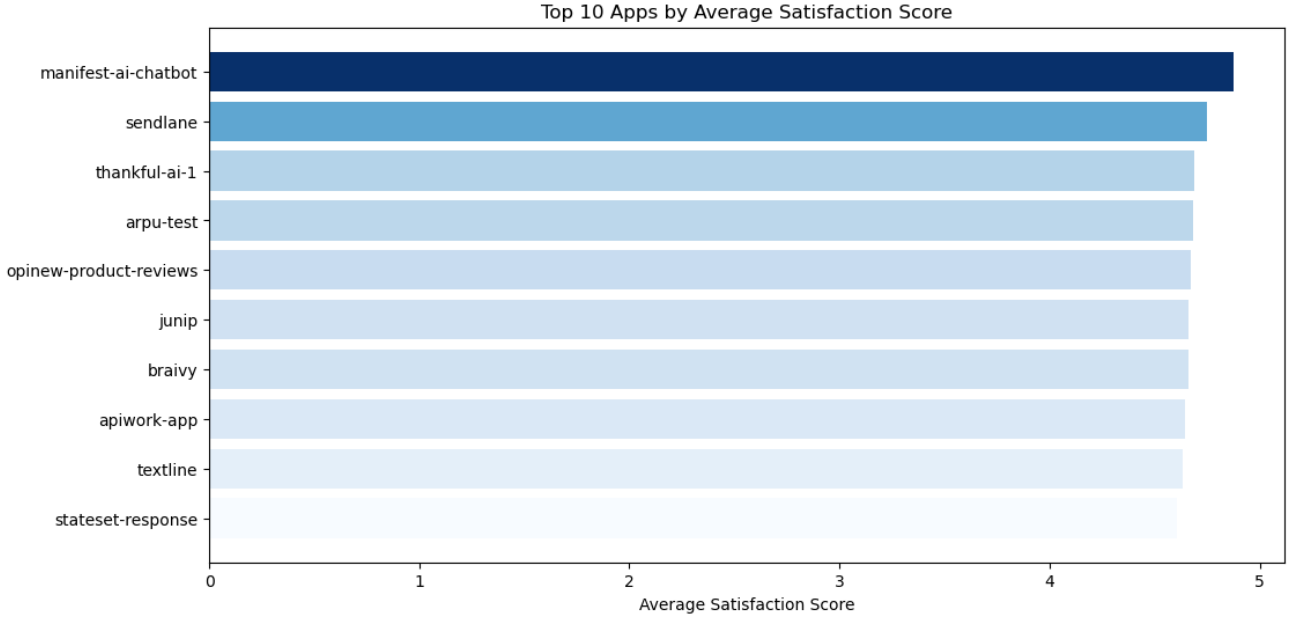
```
                        app  avg_resolution_time  avg_first_response_time  \
122     manifest-ai-chatbot             11.975000                12.550000
81                  sendlane             15.947857                 7.620714
96            thankful-ai-1             11.211429                 3.402857
67                arpu-test             13.288261                 7.767826
117  opinew-product-reviews              6.113333                 4.126667
49                     junip             15.860435                10.326304
126                  braivy             13.270000                 7.560000
100             apiwork-app             18.865000                13.115000
97                  textline              9.001111                 3.492222
87        stateset-response             16.412308                 8.315385

     avg_satisfaction_score
122                 4.875000
81                  4.750257
96                  4.688600
67                  4.682009
117                 4.670267
49                  4.660478
126                 4.659200
100                 4.645000
97                  4.631278
87                  4.605815
```

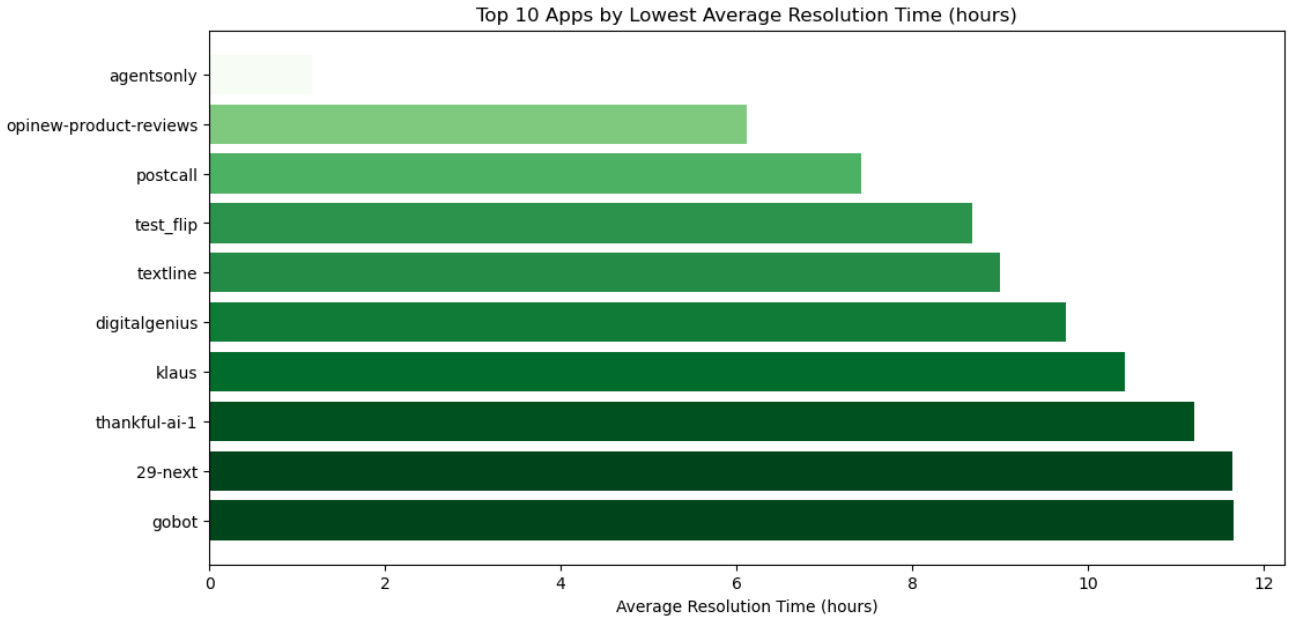
In [121…

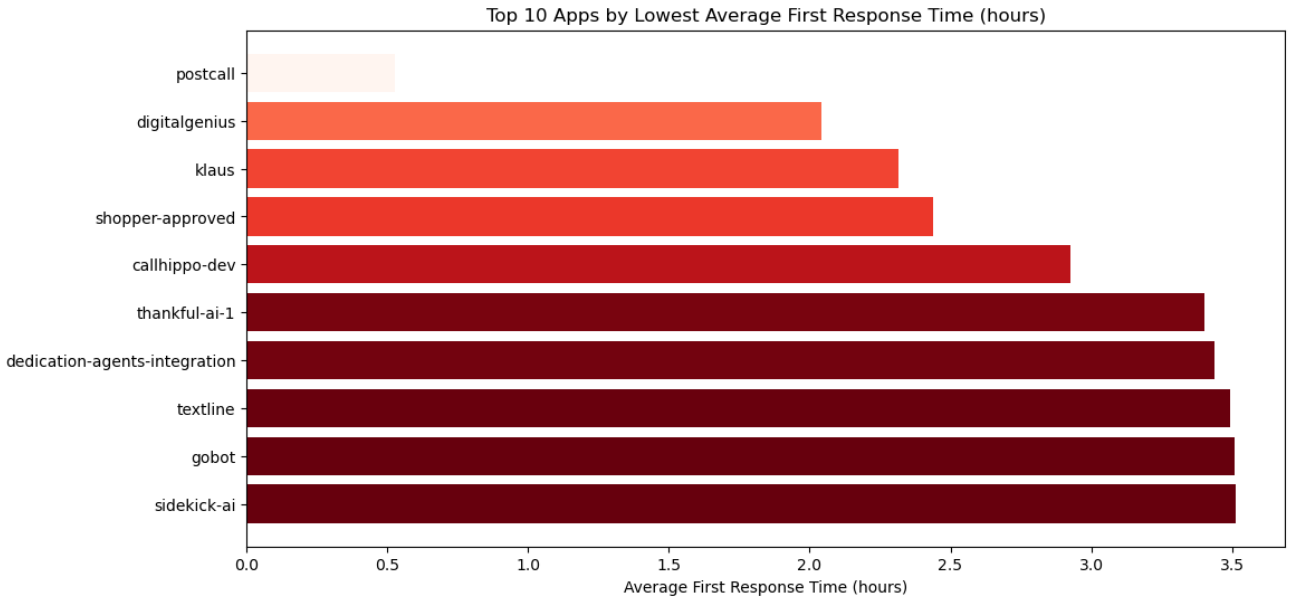
```python
# Function to apply gradient color based on values
def gradient_color_barh(ax, bars, cmap_name):
    # Extract the values (widths) of the bars
    bar_values = [bar.get_width() for bar in bars]
    min_value, max_value = min(bar_values), max(bar_values)
    norm = plt.Normalize(min_value, max_value)  # Normalize the values to ra

    for bar in bars:
        bar.set_facecolor(plt.cm.get_cmap(cmap_name)(norm(bar.get_width())))

# Top 10 apps by average satisfaction score
top_apps_by_satisfaction = app_metrics_df.sort_values(by='avg_satisfaction_s
bars = top_apps_by_satisfaction['avg_satisfaction_score'].values

plt.figure(figsize=(12, 6))
bars_plot = plt.barh(top_apps_by_satisfaction['app'], bars, color='skyblue')
gradient_color_barh(plt.gca(), bars_plot, 'Blues')
plt.xlabel('Average Satisfaction Score')
plt.title('Top 10 Apps by Average Satisfaction Score')
plt.gca().invert_yaxis()
plt.show()

# Top 10 apps by lowest average resolution time (in hours)
top_apps_by_resolution = app_metrics_df.sort_values(by='avg_resolution_time'
bars = top_apps_by_resolution['avg_resolution_time'].values
```

```
plt.figure(figsize=(12, 6))
bars_plot = plt.barh(top_apps_by_resolution['app'], bars, color='lightgreen'
gradient_color_barh(plt.gca(), bars_plot, 'Greens')
plt.xlabel('Average Resolution Time (hours)')
plt.title('Top 10 Apps by Lowest Average Resolution Time (hours)')
plt.gca().invert_yaxis()
plt.show()

# Top 10 apps by lowest average first response time (in hours)
top_apps_by_response = app_metrics_df.sort_values(by='avg_first_response_tim
bars = top_apps_by_response['avg_first_response_time'].values

plt.figure(figsize=(12, 6))
bars_plot = plt.barh(top_apps_by_response['app'], bars, color='lightcoral')
gradient_color_barh(plt.gca(), bars_plot, 'Reds')
plt.xlabel('Average First Response Time (hours)')
plt.title('Top 10 Apps by Lowest Average First Response Time (hours)')
plt.gca().invert_yaxis()
plt.show()

plt.savefig('/Users/pierremora/Desktop/Gorgias/top_apps_by_first_response_ti
plt.close()
```

```
/var/folders/fs/0r3n69nx483gkfslj205nybc0000gn/T/ipykernel_22754/3682675430.
py:9: MatplotlibDeprecationWarning: The get_cmap function was deprecated in
Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotli
b.colormaps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  bar.set_facecolor(plt.cm.get_cmap(cmap_name)(norm(bar.get_width())))
```







In [122…
```
# Segment the data into two groups: with enabled apps and without enabled ap
with_apps = df_actions[df_actions['enabled_apps'].apply(lambda x: len(x) > 0
without_apps = df_actions[df_actions['enabled_apps'].apply(lambda x: len(x)

# Checking the size of each segment
print(f"Number of observations with enabled apps: {with_apps.shape[0]}")
print(f"Number of observations without enabled apps: {without_apps.shape[0]}
```

```
Number of observations with enabled apps: 14974
Number of observations without enabled apps: 9018
```

In [125…
```
# Segmenting the data into groups based on the number of enabled apps
segment_1 = df_actions[df_actions['num_enabled_apps'] == 1]
segment_2_to_4 = df_actions[(df_actions['num_enabled_apps'] >= 2) & (df_acti
segment_5_to_6 = df_actions[(df_actions['num_enabled_apps'] >= 5) & (df_acti
```

```
segment_7_or_more = df_actions[df_actions['num_enabled_apps'] >= 7]

# Checking the size of each segment
print(f"Segment 1 Enabled App: {segment_1.shape[0]} rows")
print(f"Segment 2 to 4 Enabled Apps: {segment_2_to_4.shape[0]} rows")
print(f"Segment 5 to 6 Enabled Apps: {segment_5_to_6.shape[0]} rows")
print(f"Segment 7 or More Enabled Apps: {segment_7_or_more.shape[0]} rows")
```

```
Segment 1 Enabled App: 1416 rows
Segment 2 to 4 Enabled Apps: 8209 rows
Segment 5 to 6 Enabled Apps: 3316 rows
Segment 7 or More Enabled Apps: 2033 rows
```

In [126…

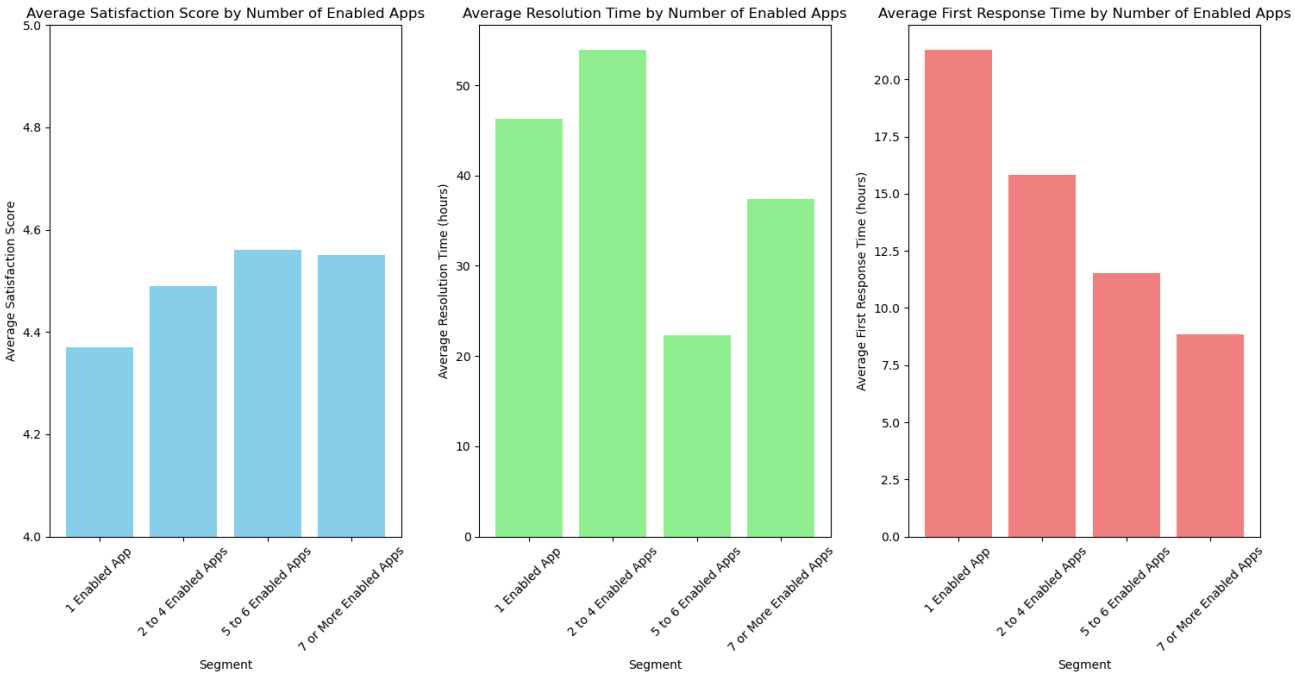
```
# Calculating average metrics for each segment
metrics_by_segment = {
    'Segment 1 Enabled App': {
        'avg_satisfaction_score': segment_1['avg_satisfaction_score_30d'].me
        'avg_resolution_time': segment_1['avg_resolution_time_30d'].mean(),
        'avg_first_response_time': segment_1['avg_first_response_time_30d'].
    },
    'Segment 2 to 4 Enabled Apps': {
        'avg_satisfaction_score': segment_2_to_4['avg_satisfaction_score_30d
        'avg_resolution_time': segment_2_to_4['avg_resolution_time_30d'].mea
        'avg_first_response_time': segment_2_to_4['avg_first_response_time_3
    },
    'Segment 5 to 6 Enabled Apps': {
        'avg_satisfaction_score': segment_5_to_6['avg_satisfaction_score_30d
        'avg_resolution_time': segment_5_to_6['avg_resolution_time_30d'].mea
        'avg_first_response_time': segment_5_to_6['avg_first_response_time_3
    },
    'Segment 7 or More Enabled Apps': {
        'avg_satisfaction_score': segment_7_or_more['avg_satisfaction_score_
        'avg_resolution_time': segment_7_or_more['avg_resolution_time_30d'].
        'avg_first_response_time': segment_7_or_more['avg_first_response_tim
    }
}

# Displaying the results
for segment, metrics in metrics_by_segment.items():
    print(f"\n{segment}:")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.2f}")
```

```
Segment 1 Enabled App:
avg_satisfaction_score: 4.37
avg_resolution_time: 46.24
avg_first_response_time: 21.31

Segment 2 to 4 Enabled Apps:
avg_satisfaction_score: 4.49
avg_resolution_time: 53.96
avg_first_response_time: 15.81

Segment 5 to 6 Enabled Apps:
avg_satisfaction_score: 4.56
avg_resolution_time: 22.29
avg_first_response_time: 11.54

Segment 7 or More Enabled Apps:
avg_satisfaction_score: 4.55
avg_resolution_time: 37.44
avg_first_response_time: 8.83
```

In [127…

```python
import matplotlib.pyplot as plt

# Segments and their corresponding metrics
segments = ['1 Enabled App', '2 to 4 Enabled Apps', '5 to 6 Enabled Apps', '
satisfaction_scores = [4.37, 4.49, 4.56, 4.55]
resolution_times = [46.24, 53.96, 22.29, 37.44]
response_times = [21.31, 15.81, 11.54, 8.83]

# Create a figure for the three subplots
plt.figure(figsize=(15, 8))

# Plotting Satisfaction Scores
plt.subplot(1, 3, 1)
bars = plt.bar(segments, satisfaction_scores, color='skyblue')
plt.title('Average Satisfaction Score by Number of Enabled Apps')
plt.xlabel('Segment')
plt.ylabel('Average Satisfaction Score')
plt.ylim(4.0, 5.0)
plt.xticks(rotation=45)

# Plotting Resolution Times
plt.subplot(1, 3, 2)
bars = plt.bar(segments, resolution_times, color='lightgreen')
plt.title('Average Resolution Time by Number of Enabled Apps')
plt.xlabel('Segment')
plt.ylabel('Average Resolution Time (hours)')
plt.xticks(rotation=45)

# Plotting First Response Times
plt.subplot(1, 3, 3)
```

```
bars = plt.bar(segments, response_times, color='lightcoral')
plt.title('Average First Response Time by Number of Enabled Apps')
plt.xlabel('Segment')
plt.ylabel('Average First Response Time (hours)')
plt.xticks(rotation=45)

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the plot
plt.show()
```



In [ ]:
```
# Inspect the first few rows of the enabled_apps column
print(df_actions['enabled_apps'].head(10))
```