notebook

September 8, 2025

1 Import Libraries

[1]:
```python
import functools
import tracemalloc
import psutil
import os
from pathlib import Path
import time

import httpx
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

2 Utilities

[2]:
```python
def profiler(func):
    """Decorator to measure memory usage and execution time of a function."""
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        process = psutil.Process(os.getpid())

        # Start memory + time tracking
        start_mem = process.memory_info().rss / 1024**2
        tracemalloc.start()
        start_time = time.time()

        result = func(*args, **kwargs)  # run target function

        # After execution
        current, peak = tracemalloc.get_traced_memory()
        end_mem = process.memory_info().rss / 1024**2
        end_time = time.time()
        tracemalloc.stop()

        print(f"\n--- Memory Profile for `{func.__name__}` ---")
        print(f"Start memory   : {start_mem:.2f} MB")
        print(f"End memory     : {end_mem:.2f} MB")
        print(f"Peak (tracked) : {peak / 1024**2:.2f} MB")
        print(f"Execution time : {end_time - start_time:.2f} sec")
        print("------------------------------------------\n")

        return result
    return wrapper
```

[3]:
```
# Constants

ROOT_PATH = Path(os.getcwd())
DATASET_URL = "https://drive.usercontent.google.com/download?
 ↪id=1N1xoxgcw2K3d-49tlchXAWw4wuxLj7EV&export=download"
DATASET_OUTPUT_PATH = ROOT_PATH / "dataset.csv"
```

[4]:
```
# Utilities
@profiler
def download_data(url: str, output_path: Path) -> None:
    with httpx.stream("GET", url) as response:
        response.raise_for_status()  # check for HTTP errors
        with open(output_path, "wb") as f:
            for chunk in response.iter_bytes():
                f.write(chunk)

    print(f"Downloaded to {output_path}")

@profiler
def load_data(file_path: Path, **kwargs) -> pd.DataFrame:
    return pd.read_csv(file_path, **kwargs)
```

3 EDA

3.1 Data Loading

[ ]:
```
# Data Loading
download_data(DATASET_URL, DATASET_OUTPUT_PATH)
```

[13]:
```
# Data Parsing
df = load_data(DATASET_OUTPUT_PATH, index_col=0)
```

```
--- Memory Profile for `load_data` ---
Start memory   : 193.36 MB
End memory     : 304.31 MB
Peak (tracked) : 56.09 MB
Execution time : 1.75 sec
------------------------------------------
```

3.2 Data Understanding

[6]:
```
df.shape
```

[6]:
```
(100000, 11)
```

[7]:
```
df.head()
```

[7]:
```
            Customer Id First Name   Last Name                    Company  \
Index
1       ffeCAb7AbcB0f07      Jared     Jarvis            Sanchez-Fletcher
2       b687FfC4F1600eC      Marie     Malone                   Mckay PLC
3       9FF9ACbc69dcF9c     Elijah    Barrera              Marks and Sons
4       b49edDB1295FF6E     Sheryl  Montgomery  Kirby, Vaughn and Sanders
5       3dcCbFEB17CCf2E     Jeremy    Houston              Lester-Manning

                   City                                       Country  \
Index
1         Hatfieldshire                                       Eritrea
2       Robertsonburgh                                       Botswana
3               Kimbury                                      Barbados
4           Briannaview  Antarctica (the territory South of 60 deg S)
5         South Brianna                                    Micronesia

                      Phone 1                Phone 2  \
Index
1         274.188.8773x41185  001-215-760-4642x969
2               283-236-9529   (189)129-8356x63741
3                 8252703789     459-916-7241x0909
4               425.475.3586         (392)819-9063
5       +1-223-666-5313x4530      252-488-3850x692

                                  Email Subscription Date  \
Index
1         gabriellehartman@benjamin.com        2021-11-11
2                  kstafford@sexton.com        2021-05-14
3               jeanettecross@brown.com        2021-03-17
4             thomassierra@barrett.com        2020-09-23
5       rubenwatkins@jacobs-wallace.info        2020-09-18

                         Website
Index
1         https://www.mccarthy.info/
2           http://www.reynolds.com/
3                  https://neal.com/
```

```
4      https://www.powell-bryan.com/
5          https://www.carrillo.com/
```

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 100000 entries, 1 to 100000
Data columns (total 11 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Customer Id        100000 non-null  object
 1   First Name         100000 non-null  object
 2   Last Name          100000 non-null  object
 3   Company            100000 non-null  object
 4   City               100000 non-null  object
 5   Country            100000 non-null  object
 6   Phone 1            100000 non-null  object
 7   Phone 2            100000 non-null  object
 8   Email              100000 non-null  object
 9   Subscription Date  100000 non-null  object
 10  Website            100000 non-null  object
dtypes: object(11)
memory usage: 9.2+ MB
```

[16]:
```
# Checking missing values
df.isnull().sum()
```

[16]:
```
Customer Id          0
First Name           0
Last Name            0
Company              0
City                 0
Country              0
Phone 1              0
Phone 2              0
Email                0
Subscription Date    0
Website              0
dtype: int64
```

3.3 Data Cleaning and Preparation

[15]:
```
# Parsing to date
@profiler
def parse_date(df: pd.DataFrame, col: str):
    df[col] = pd.to_datetime(df[col])

parse_date(df, "Subscription Date")
```

```
--- Memory Profile for `parse_date` ---
Start memory    : 242.02 MB
End memory      : 242.78 MB
Peak (tracked) : 1.53 MB
Execution time : 0.10 sec
------------------------------------------
```

[11]:
```
df.nunique()
```

[11]:
```
Customer Id          100000
First Name              690
Last Name              1000
Company               71994
City                  49154
Country                 243
Phone 1              100000
Phone 2              100000
Email                 99995
Subscription Date       880
Website               50471
dtype: int64
```

3.4 Descriptive Analysis

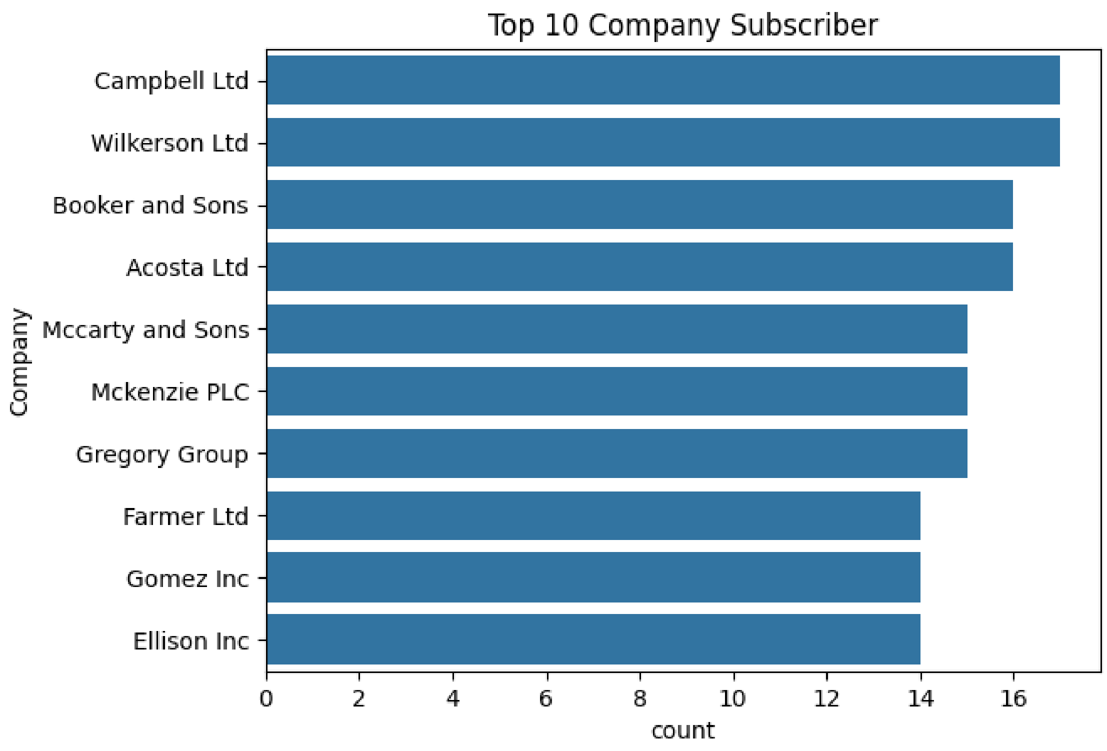
[130]:
```
@profiler
def plot_top_n(src, col: str = None, title: str = None, top: int = 10):
    if isinstance(src, pd.DataFrame):
        if col is None:
            raise ValueError("When src is a DataFrame, you must provide a␣
↪column name.")
        series = src[col]
    elif isinstance(src, pd.Series):
        series = src
    else:
        raise TypeError("src must be a pandas DataFrame or Series")

    ax = sns.countplot(
        y=series,
        order=series.value_counts().head(top).index
    )
    ax.set_title(title)
```

3.4.1 Company

```
[131]: plot_top_n(df, 'Company', "Top 10 Company Subscriber")
```

```
--- Memory Profile for `plot_top_n` ---
Start memory   : 377.68 MB
End memory     : 385.97 MB
Peak (tracked) : 11.51 MB
Execution time : 0.73 sec
------------------------------------------
```



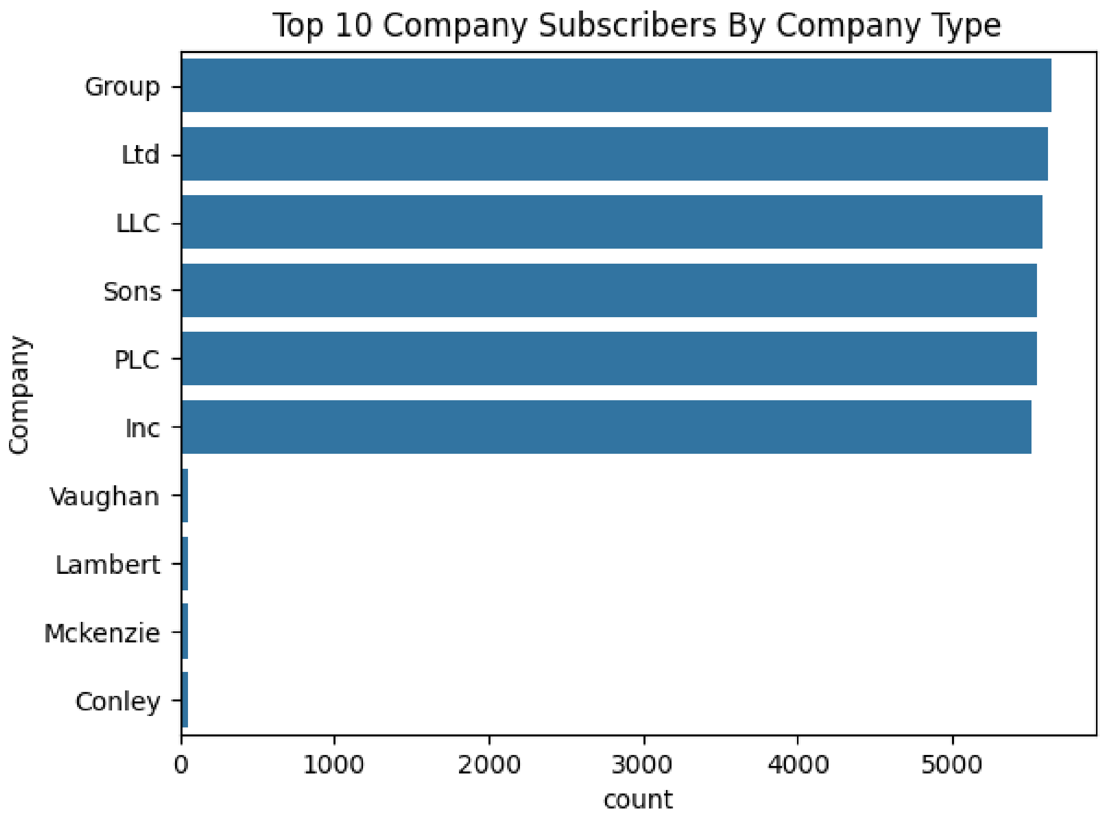
```
[93]: @profiler
def plot_top_company_by_type(src):
    company_suffix = src["Company"].str.split().apply(lambda x: x[-1])
    plot_top_n(company_suffix, title="Top 10 Company Subscribers By Company␣
 ↪Type")

plot_top_company_by_type(df)
```

```
--- Memory Profile for `plot_top_10` ---
Start memory   : 384.72 MB
```

```
End memory     : 392.74 MB
Peak (tracked) : 30.62 MB
Execution time : 1.07 sec
------------------------------------------


--- Memory Profile for `plot_top_company_by_type` ---
Start memory   : 377.88 MB
End memory     : 358.63 MB
Peak (tracked) : 0.00 MB
Execution time : 3.04 sec
------------------------------------------
```

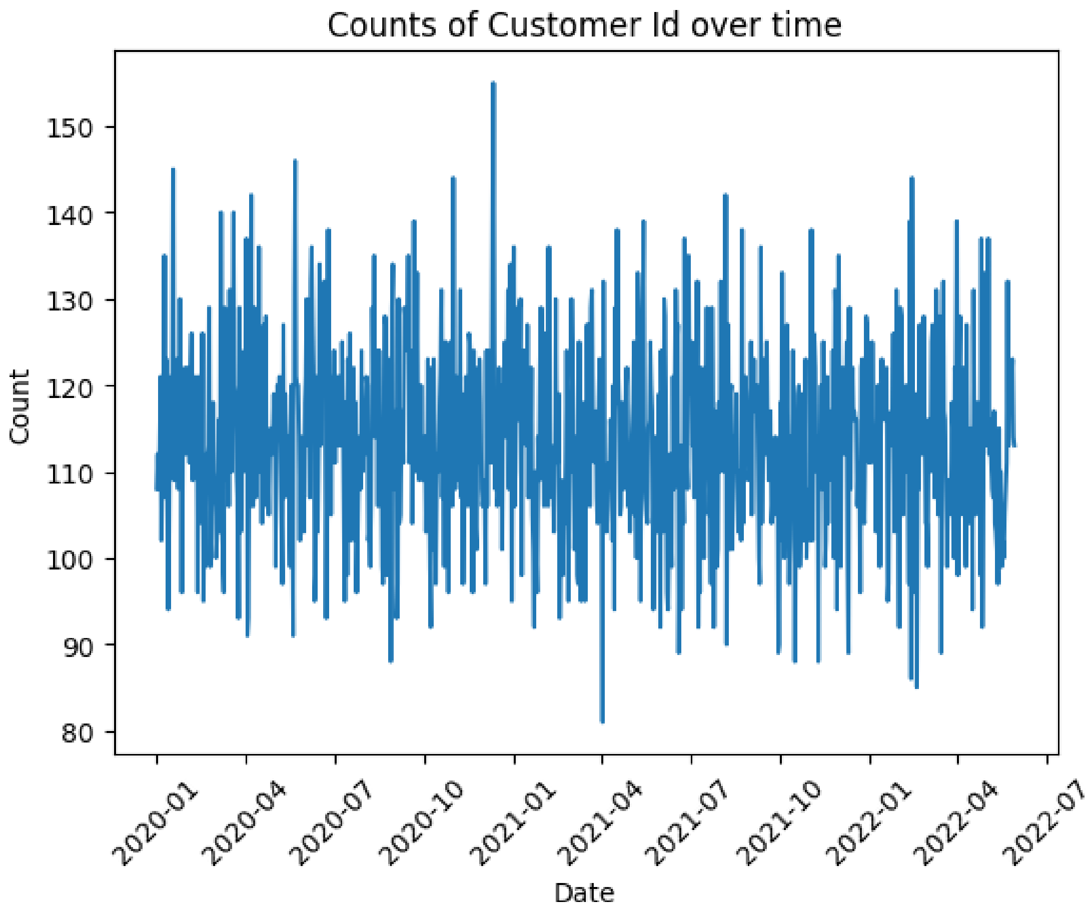


3.4.2 Subscription Date

[94]:
```python
@profiler
def plot_counts_over_time(df: pd.DataFrame, date_col: str, count_col: str):
    # Ensure datetime
    df[date_col] = pd.to_datetime(df[date_col])
    # Aggregate counts per date
    df_counts = df.groupby(date_col)[count_col].count().reset_index()

    # Plot
    sns.lineplot(data=df_counts, x=date_col, y=count_col)
    plt.xticks(rotation=45)
    plt.xlabel("Date")
    plt.ylabel("Count")
    plt.title(f"Counts of {count_col} over time")
    plt.show()

plot_counts_over_time(df, "Subscription Date", "Customer Id")
```



```
--- Memory Profile for `plot_counts_over_time` ---
Start memory    : 360.22 MB
End memory      : 365.00 MB
Peak (tracked) : 3.57 MB
```

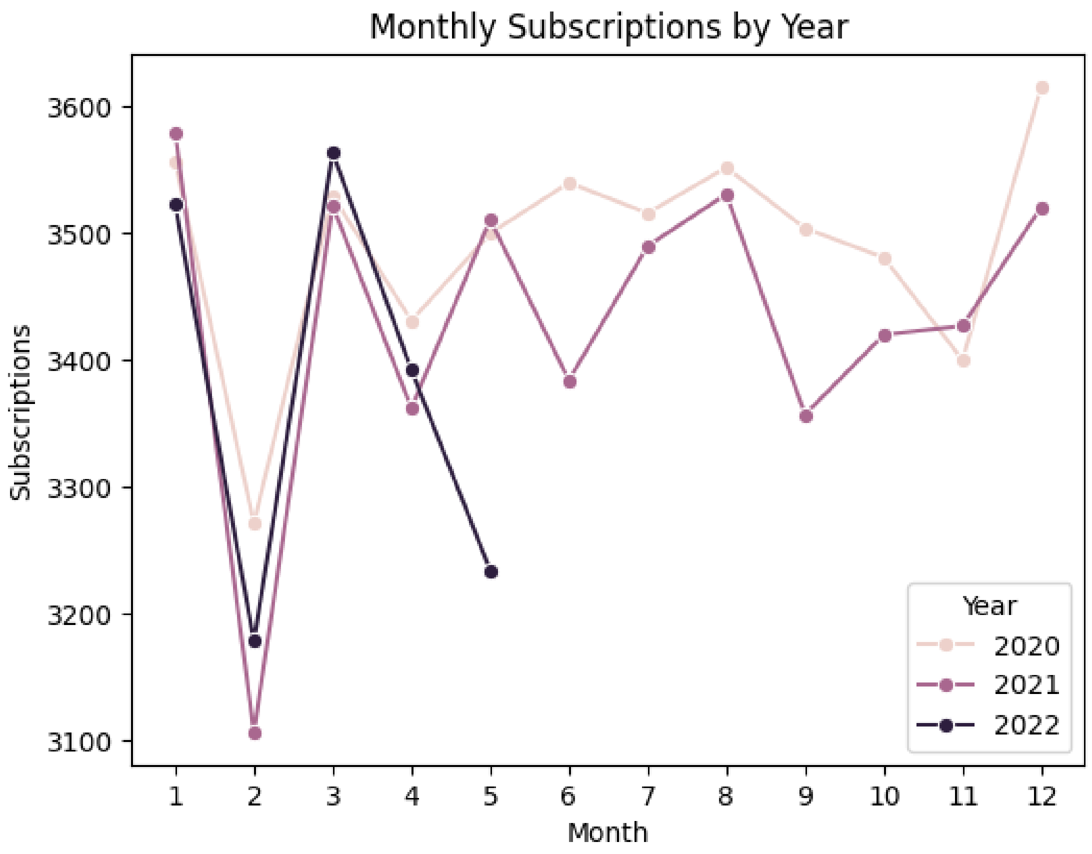
```
Execution time : 2.38 sec
------------------------------------------
```

[147]:
```
@profiler
def plot_subscriptions_monthly(df: pd.DataFrame, date_col: str):
    df["Year"] = df[date_col].dt.year
    df["Month"] = df[date_col].dt.month

    monthly_counts = df.groupby(["Year", "Month"]).size().
 ↪reset_index(name="Count")
    sns.lineplot(
        data=monthly_counts,
        x="Month",
        y="Count",
        hue="Year",       # separate line per year
        marker="o"        # dots on points (optional)
    )

    plt.xticks(range(1, 13))  # months 1-12
    plt.xlabel("Month")
    plt.ylabel("Subscriptions")
    plt.title("Monthly Subscriptions by Year")
    plt.show()

plot_subscriptions_monthly(df, "Subscription Date")
```

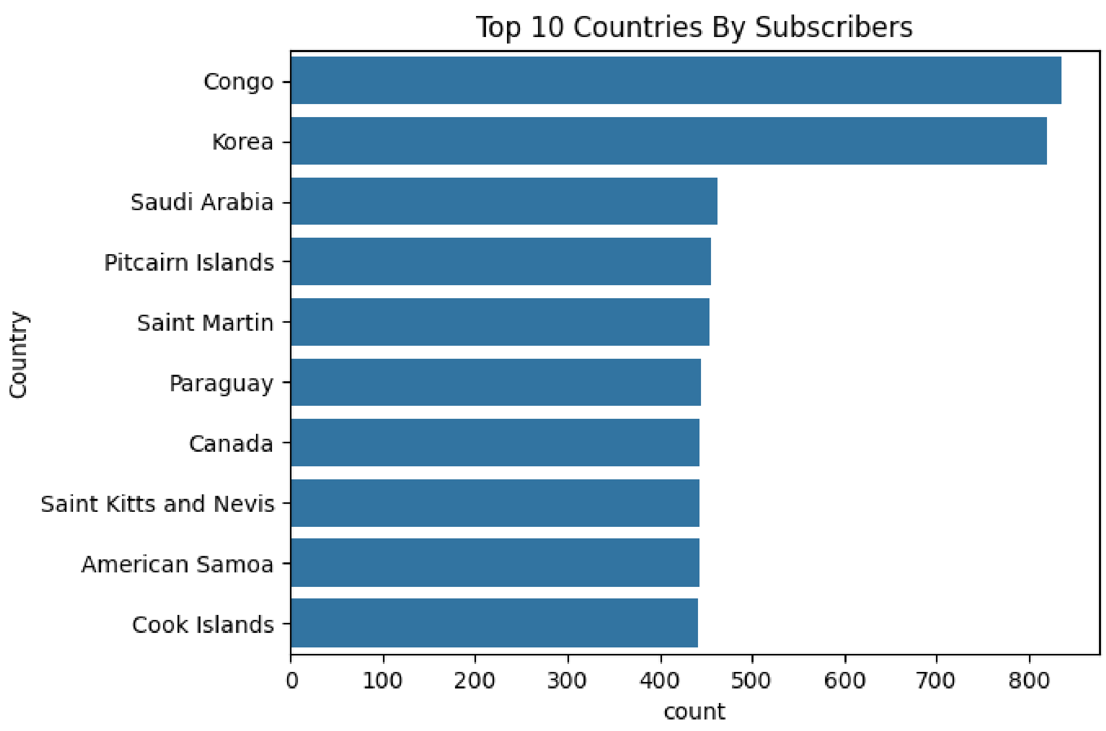


```
--- Memory Profile for `plot_subscriptions_monthly` ---
Start memory   : 424.00 MB
End memory     : 426.63 MB
Peak (tracked) : 6.70 MB
Execution time : 1.54 sec
------------------------------------------
```

3.4.3 Country

[97]:
```
plot_top_10(df, "Country", title="Top 10 Countries By Subscribers")
```

```
--- Memory Profile for `plot_top_10` ---
Start memory   : 368.26 MB
End memory     : 371.79 MB
Peak (tracked) : 10.96 MB
Execution time : 0.74 sec
------------------------------------------
```

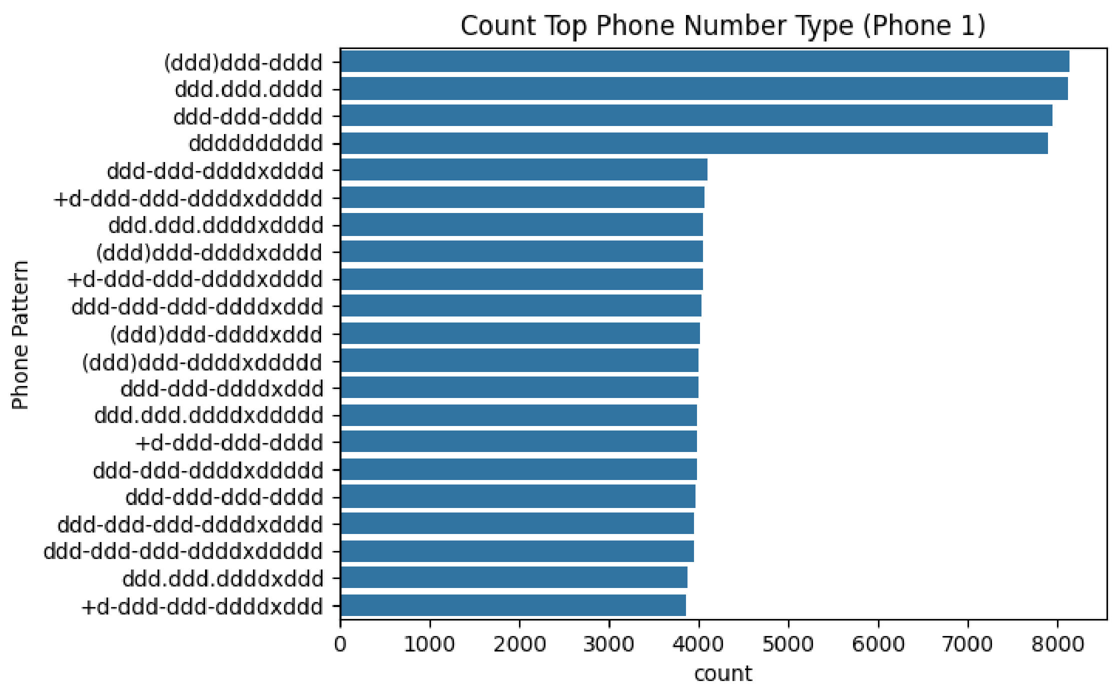


3.4.4 Phone Number

[141]:
```
def plot_phone_number_pattern(df: pd.DataFrame, col: str):
    df['Phone Pattern'] = df[col].str.replace(r"\d", "d", regex=True)
    n = df['Phone Pattern'].nunique()
    plot_top_n(df, 'Phone Pattern', title=f"Count Top Phone Number Type␣
 ↪({col})", top=n)
```

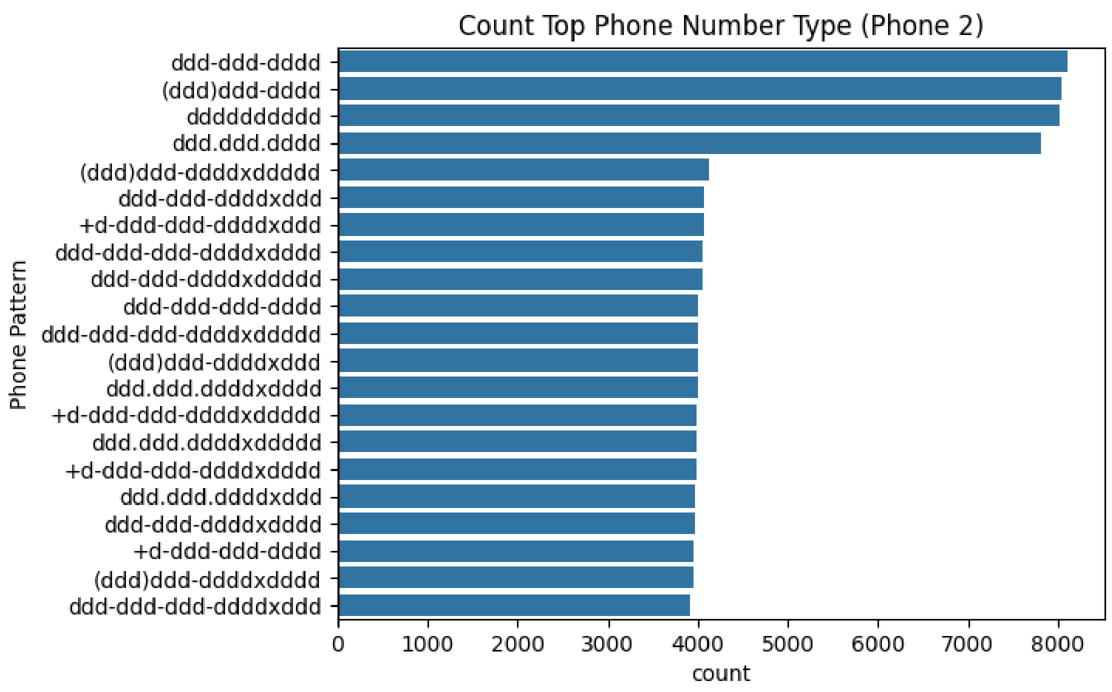
[143]:
```
plot_phone_number_pattern(df, "Phone 1")
```

```
--- Memory Profile for `plot_top_n` ---
Start memory    : 405.56 MB
End memory      : 409.86 MB
Peak (tracked) : 11.81 MB
Execution time : 1.09 sec
------------------------------------------
```



```
[144]: plot_phone_number_pattern(df, "Phone 2")
```

```
--- Memory Profile for `plot_top_n` ---
Start memory   : 416.24 MB
End memory     : 423.22 MB
Peak (tracked) : 11.81 MB
Execution time : 1.07 sec
------------------------------------------
```



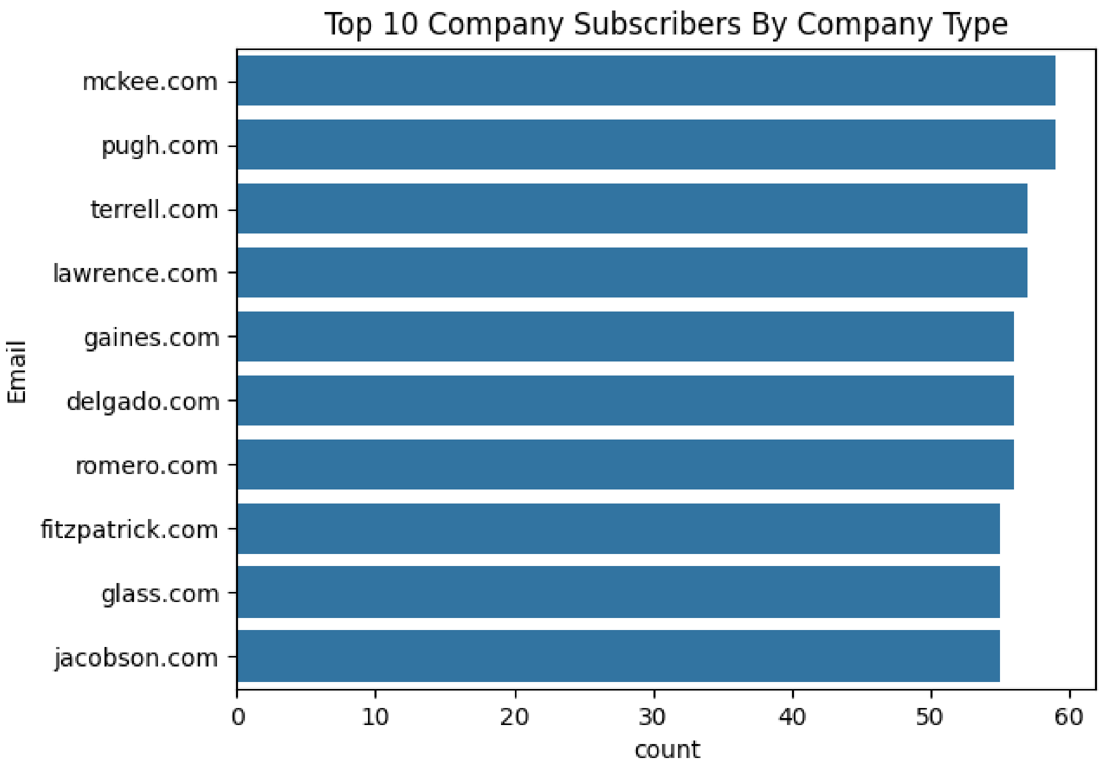
3.4.5 Email

```
[157]: email_domain = df["Email"].str.split("@").apply(lambda x: x[-1])
       email_domain.nunique()
```

```
[157]: 38322
```

```
[154]: plot_top_n(email_domain, title="Top 10 Company Subscribers By Company Type")
```

```
--- Memory Profile for `plot_top_n` ---
Start memory    : 385.48 MB
End memory      : 395.90 MB
Peak (tracked) : 11.25 MB
Execution time : 0.77 sec
------------------------------------------
```


Top 10 Company Subscribers By Company Type
mckee.com
pugh.com
terrell.com
lawrence.com
gaines.com
delgado.com
romero.com
fitzpatrick.com
glass.com
jacobson.com
Email
0
10
20
30
40
50
60
count