

RGR Case Study - 14122019

December 15, 2019

1 Deliveroo “Rider Get Rider” Case Study

1.0.1 By Edd Webster

Notebook last updated: 14/12/2019

```
In [1]: from IPython.display import Image
        Image(url= "./images/deliveroologo.png", width=250)
```

```
Out[1]: <IPython.core.display.Image object>
```

Click Section ?? to jump straight to the Exploratory Data Analysis section and skip the Section ?? and Section ?? sections. Or you can click Section ?? to jump straight to the Conclusion.

1.1 Introduction

This notebook is an Exploratory Data Analysis (EDA) of Rider data for [Deliveroo](#), analysing the performance of the ‘Rider Get Rider’ (RGR) scheme, an incentivised referral scheme targeting current Deliveroo Riders, determining whether or not it is successful. The dataset provided is explored and the findings summarised.

For more information about this notebook and the author, I’m available through all the following channels: * [EddWebster.com](#), * edd.j.webster@gmail.com, * [LinkedIn.com/in/eddwebster](#), and * [GitHub/eddwebster](#).

The accompanying GitHub repository for this notebook can be found [here](#) and a static version of this notebook can be found [here](#).

1.2 Notebook Contents

1. Section ??
2. Section ??
3. Section ??
4. Section ??
5. Section ??
6. Section ??
7. Section ??

1.3 1. Notebook Setup

First we will import all the required libraries for this notebook. These include: * [IPython](#) * [NumPy](#) * [Pandas](#) * [Seaborn](#)

In [2]: *# Import modules*

```
# Python 3.5 (ideally)
import sys
assert sys.version_info >= (3, 5)

# Data preprocessing
import numpy as np
import pandas as pd
import os

# Plotting figures
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno    # pip install missingno if you don't have it

# Display in Jupyter
from IPython.display import Image
from IPython.core.display import HTML

# Ignore warnings
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")

print("Setup Complete")
```

Setup Complete

1.4 2. Task Brief

The following brief has been copied and pasted from the Word document provided and is included in this Jupyter notebook for reference. Click Section ?? to skip to the next section where the proper coding begins.

1.4.1 Background

At [Deliveroo](#), we want to understand our customers, rider and restaurants in as much detail as possible. As we attempt to build and maintain the optimal rider fleet we utilise a series of marketing channels and schemes to attract new Riders. One of these channels is referred to as 'Rider Get Rider' or 'RGR', which is an incentivised referral scheme targeting current Deliveroo Riders.

1.4.2 The Task

Imagine a senior manager has asked you to analyse the performance of the RGR scheme and determine if it is successful. They have asked you to take them through your findings later that day. Analyse the dataset provided and summarise your findings.

The overarching question you should answer with your analysis is: “Is the RGR scheme successful, and why have you reached that conclusion?” * You should define what success is
* How has RGR performed compared with other channels? * How does rider performance / behaviour vary between the channels?

Summarise your findings * Your summary could be a short document, one tab of a spreadsheet or a notebook * Please do not spend time creating a presentation / slide deck * It should be logically set out so the reader can follow your main findings and any conclusions you have arrived at
* It should make appropriate use of visualisations to convey your findings

Note - We have provided a clean data set and you do not need to do any data cleaning.

1.5 3. Data Source

1.5.1 Data Import

First, let's import the CSV file provided, 'rgr_take_home_v3_dataset.csv', to this Jupyter notebook as a Pandas DataFrame.

```
In [3]: # Import CSV to a Pandas DataFrame
        filepath = "data/rgr_take_home_v3_dataset.csv"
        df_raw = pd.read_csv(filepath)
```

1.5.2 Initial Data Handling

Let's quality of the dataset by looking first and last rows, using the `head()` and `tail()` methods.

```
In [4]: # Display the first 5 rows of the raw DataFrame, df_raw
        df_raw.head()
```

```
Out[4]:
```

	RIDER_ID	LOCATION	APPLICATION_DATE	APPLICATION_APPROVED_DATE	\
0	1864	Roo York	05/03/2017	31/03/2017	
1	1864	Roo York	05/03/2017	31/03/2017	
2	1864	Roo York	05/03/2017	31/03/2017	
3	1864	Roo York	05/03/2017	31/03/2017	
4	1864	Roo York	05/03/2017	31/03/2017	

	FIRST_WORK_DATE	ACQUISITION_CHANNEL	VEHICLE_TYPE	\
0	03/04/2017	Digital	Scooter / Motorcycle	
1	03/04/2017	Digital	Scooter / Motorcycle	
2	03/04/2017	Digital	Scooter / Motorcycle	
3	03/04/2017	Digital	Scooter / Motorcycle	
4	03/04/2017	Digital	Scooter / Motorcycle	

	WEEKS_SINCE_FIRST_WORK	RIDER_ACTIVE	HOURS_WORKED	\
0	0	True	18.918334	

1	1	True	33.625274
2	2	True	10.539444
3	3	True	10.268887
4	4	True	17.463331

	HOURS_WORKED_CUMULATIVE	ORDERS_DELIVERED	ORDERS_DELIVERED_CUMULATIVE \
0	18.918334	44.0	44.0
1	52.543608	78.0	122.0
2	63.083052	30.0	152.0
3	73.351939	38.0	190.0
4	90.815270	58.0	248.0

	THROUGHPUT	THROUGHPUT_CUMULATIVE	REFERRALS	REFERRALS_CUMULATIVE \
0	2.325786	2.325786	NaN	NaN
1	2.319684	2.321881	NaN	NaN
2	2.846450	2.409522	NaN	NaN
3	3.700498	2.590252	NaN	NaN
4	3.321245	2.730818	NaN	NaN

	SUCCESSFUL_REFERRALS	SUCCESSFUL_REFERRALS_CUMULATIVE
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN

```
In [5]: # Display the last 5 rows of the raw DataFrame, df_raw
df_raw.tail()
```

```
Out [5]:
```

	RIDER_ID	LOCATION	APPLICATION_DATE	APPLICATION_APPROVED_DATE \
113520	103776	Roo York	28/03/2017	07/04/2017
113521	103776	Roo York	28/03/2017	07/04/2017
113522	103776	Roo York	28/03/2017	07/04/2017
113523	103776	Roo York	28/03/2017	07/04/2017
113524	103776	Roo York	28/03/2017	07/04/2017

	FIRST_WORK_DATE	ACQUISITION_CHANNEL	VEHICLE_TYPE \
113520	08/04/2017	Referral	Scooter / Motorcycle
113521	08/04/2017	Referral	Scooter / Motorcycle
113522	08/04/2017	Referral	Scooter / Motorcycle
113523	08/04/2017	Referral	Scooter / Motorcycle
113524	08/04/2017	Referral	Scooter / Motorcycle

	WEEKS_SINCE_FIRST_WORK	RIDER_ACTIVE	HOURS_WORKED \
113520	20	False	NaN
113521	21	True	2.080833
113522	22	True	0.664722
113523	23	False	0.000000

113524	24	False	NaN
--------	----	-------	-----

	HOURS_WORKED_CUMULATIVE	ORDERS_DELIVERED	\
113520	283.251111	NaN	
113521	285.331944	4.0	
113522	285.996666	2.0	
113523	285.996666	NaN	
113524	285.996666	NaN	

	ORDERS_DELIVERED_CUMULATIVE	THROUGHPUT	THROUGHPUT_CUMULATIVE	\
113520	638.0	NaN	2.252418	
113521	642.0	1.922307	2.250011	
113522	644.0	3.008777	2.251775	
113523	644.0	NaN	2.251775	
113524	644.0	NaN	2.251775	

	REFERRALS	REFERRALS_CUMULATIVE	SUCCESSFUL_REFERRALS	\
113520	NaN	NaN	NaN	
113521	NaN	NaN	NaN	
113522	NaN	NaN	NaN	
113523	NaN	NaN	NaN	
113524	NaN	NaN	NaN	

	SUCCESSFUL_REFERRALS_CUMULATIVE
113520	NaN
113521	NaN
113522	NaN
113523	NaN
113524	NaN

```
In [6]: # Print the Shape of the raw DataFrame, df_raw
print(df_raw.shape)
```

```
(113525, 19)
```

The raw DataFrame has: * 113,525 observations (rows), each observation represents one Rider per week, and * 19 attributes (columns).

```
In [7]: # Features (column names) of the raw DataFrame, df_raw
df_raw.columns
```

```
Out[7]: Index(['RIDER_ID', 'LOCATION', 'APPLICATION_DATE', 'APPLICATION_APPROVED_DATE',
              'FIRST_WORK_DATE', 'ACQUISITION_CHANNEL', 'VEHICLE_TYPE',
              'WEEKS_SINCE_FIRST_WORK', 'RIDER_ACTIVE', 'HOURS_WORKED',
              'HOURS_WORKED_CUMULATIVE', 'ORDERS_DELIVERED',
              'ORDERS_DELIVERED_CUMULATIVE', 'THROUGHPUT', 'THROUGHPUT_CUMULATIVE',
              'REFERRALS', 'REFERRALS_CUMULATIVE', 'SUCCESSFUL_REFERRALS',
              'SUCCESSFUL_REFERRALS_CUMULATIVE'],
              dtype='object')
```

The Dataset has nineteen features (columns): * RIDER_ID: a unique rider identifier * LOCATION: the city the rider applied in * APPLICATION_DATE: the date the rider applied to work for Deliveroo * APPLICATION_APPROVED_DATE: * FIRST_WORK_DATE: the date that the rider application was approved * ACQUISITION_CHANNEL: the last touch marketing channel recorded for the rider/applicant * VEHICLE_TYPE: the type of vehicle the rider uses * WEEKS_SINCE_FIRST_WORK: the number of weeks since the first_work_date, from 0-24 * RIDER_ACTIVE: Boolean flag if the rider worked in that week or not * HOURS_WORKED: the number of hours worked by the rider that week * HOURS_WORKED_CUMULATIVE: the cumulative number of hours worked by the rider to date * ORDERS_DELIVERED: the number of orders delivered by the rider that week * ORDERS_DELIVERED_CUMULATIVE: the cumulative number of orders delivered by the rider to date * THROUGHPUT: the orders per hour (orders / hours) of the rider that week * THROUGHPUT_CUMULATIVE_REFERRALS: the cumulative throughput of the rider to date * REFERRALS: the number of referrals made by that rider that week * REFERRALS_CUMULATIVE: the cumulative number of referrals made by that rider to date * SUCCESSFUL_REFERRALS: the number of successful (approved) referrals made by the rider in that week * SUCCESSFUL_REFERRALS_CUMULATIVE: the cumulative number of successful (approved) referrals made by the rider to date

The `dtypes` method returns the data types of each attribute in the DataFrame.

```
In [8]: # dtypes of the features of the raw DataFrame, df_raw
df_raw.dtypes
```

```
Out [8]: RIDER_ID                int64
LOCATION                object
APPLICATION_DATE        object
APPLICATION_APPROVED_DATE  object
FIRST_WORK_DATE         object
ACQUISITION_CHANNEL     object
VEHICLE_TYPE           object
WEEKS_SINCE_FIRST_WORK  int64
RIDER_ACTIVE            bool
HOURS_WORKED           float64
HOURS_WORKED_CUMULATIVE float64
ORDERS_DELIVERED       float64
ORDERS_DELIVERED_CUMULATIVE float64
THROUGHPUT             float64
THROUGHPUT_CUMULATIVE  float64
REFERRALS              float64
REFERRALS_CUMULATIVE   float64
SUCCESSFUL_REFERRALS    float64
SUCCESSFUL_REFERRALS_CUMULATIVE float64
dtype: object
```

The `info` method to get a quick description of the data, in particular the total number of rows, and each attribute's type and number of non-null values.

```
In [9]: # Info for the raw DataFrame, df_raw
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113525 entries, 0 to 113524
```

Data columns (total 19 columns):

RIDER_ID	113525 non-null int64
LOCATION	113525 non-null object
APPLICATION_DATE	113525 non-null object
APPLICATION_APPROVED_DATE	113525 non-null object
FIRST_WORK_DATE	113525 non-null object
ACQUISITION_CHANNEL	113525 non-null object
VEHICLE_TYPE	113525 non-null object
WEEKS_SINCE_FIRST_WORK	113525 non-null int64
RIDER_ACTIVE	113525 non-null bool
HOURS_WORKED	55001 non-null float64
HOURS_WORKED_CUMULATIVE	113525 non-null float64
ORDERS_DELIVERED	51774 non-null float64
ORDERS_DELIVERED_CUMULATIVE	111362 non-null float64
THROUGHPUT	51774 non-null float64
THROUGHPUT_CUMULATIVE	111362 non-null float64
REFERRALS	2409 non-null float64
REFERRALS_CUMULATIVE	17068 non-null float64
SUCCESSFUL_REFERRALS	2409 non-null float64
SUCCESSFUL_REFERRALS_CUMULATIVE	17068 non-null float64

dtypes: bool(1), float64(10), int64(2), object(6)
memory usage: 15.7+ MB

The `describe` method to show some useful statistics for each numerical column in the DataFrame.

```
In [10]: # Description of the raw DataFrame, df_raw, showing some summary statistics for each column
df_raw.describe()
```

```
Out[10]:
```

	RIDER_ID	WEEKS_SINCE_FIRST_WORK	HOURS_WORKED	\
count	113525.000000	113525.000000	55001.000000	
mean	80873.171989	12.000000	16.333247	
std	12798.378932	7.211134	14.652287	
min	1864.000000	0.000000	0.000000	
25%	70838.000000	6.000000	5.484444	
50%	77621.000000	12.000000	11.976390	
75%	94220.000000	18.000000	22.985834	
max	103776.000000	24.000000	122.296665	

	HOURS_WORKED_CUMULATIVE	ORDERS_DELIVERED	ORDERS_DELIVERED_CUMULATIVE	\
count	113525.000000	51774.000000	111362.000000	
mean	117.987786	37.995287	252.875532	
std	166.645993	35.692802	389.403744	
min	0.167500	1.000000	1.000000	
25%	19.801390	12.000000	35.000000	
50%	55.344720	26.000000	104.000000	
75%	143.990280	53.000000	294.000000	

max		2075.802227	284.000000	5046.000000
-----	--	-------------	------------	-------------

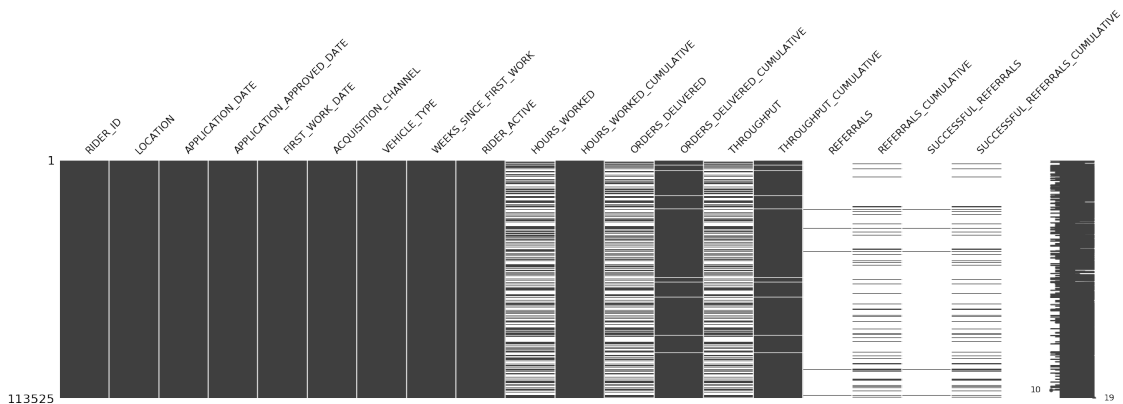
	THROUGHPUT	THROUGHPUT_CUMULATIVE	REFERRALS	REFERRALS_CUMULATIVE \
count	51774.000000	111362.000000	2409.000000	17068.000000
mean	2.185034	1.945347	1.077210	2.067495
std	0.781561	0.663847	1.717319	5.127942
min	0.023303	0.025417	0.000000	0.000000
25%	1.696575	1.521931	1.000000	1.000000
50%	2.210061	1.993766	1.000000	1.000000
75%	2.686744	2.403426	1.000000	2.000000
max	30.000300	6.326892	37.000000	285.000000

	SUCCESSFUL_REFERRALS	SUCCESSFUL_REFERRALS_CUMULATIVE
count	2409.000000	17068.000000
mean	0.233707	0.396649
std	0.476797	0.801456
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	1.000000
max	5.000000	28.000000

Next, we will look at the number of NULL values in the dataset. This can be plotted nicely using the [missingno](#) library (pip install missingno).

```
In [11]: # Plot visualisation of the missing values for each feature of the raw DataFrame, df_
missingno.matrix(df_raw, figsize = (30, 7))
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x11a6658fd68>
```



```
In [12]: # Counts of missing values
null_value_stats = df_raw.isnull().sum(axis=0)
null_value_stats[null_value_stats != 0]
```



```
Out [12]: HOURS_WORKED                58524
ORDERS_DELIVERED                61751
ORDERS_DELIVERED_CUMULATIVE      2163
THROUGHPUT                      61751
THROUGHPUT_CUMULATIVE           2163
REFERRALS                      111116
REFERRALS_CUMULATIVE            96457
SUCCESSFUL_REFERRALS            111116
SUCCESSFUL_REFERRALS_CUMULATIVE  96457
dtype: int64
```

Even though it is good practice to look at the null values in a new dataset, as stated in the brief, the dataset is clean and no data cleaning is required. We therefore will leave null values here.

1.5.3 Data Wrangling

As stated in the brief, the dataset has 113,525 records in which one each row refers to one rider per week i.e. each of the 4,541 Riders has 25 rows of data (including week 0 as the week they joined).

For my analysis, I want to separate the latest version of each Rider's data, i.e. week 24, from the previous week's data, to analyse only their total performance of the Riders over the 24 week period. This will leave a dataset of 4,541 records, one per Rider, each with the total value of the following data points: WEEKS_SINCE_FIRST_WORK, HOURS_WORKED_CUMULATIVE, ORDERS_DELIVERED_CUMULATIVE, THROUGHPUT_CUMULATIVE, REFERRALS_CUMULATIVE, and SUCCESSFUL_REFERRALS_CUMULATIVE.

To do this, we can take the rows only where WEEKS_SINCE_FIRST_WORK is equal to 24 i.e. the final week and create the new Dataframe that we'll be working with, using the standard naming convention df.

```
In [13]: # Week 24 only of each Rider
df = df_raw.loc[df_raw['WEEKS_SINCE_FIRST_WORK'] == 24]
```

Again we will perform some of the data handling checks that we performed with the raw dataset.

```
In [14]: df.head()
```

```
Out [14]:
```

	RIDER_ID	LOCATION	APPLICATION_DATE	APPLICATION_APPROVED_DATE	\
24	1864	Roo York	05/03/2017	31/03/2017	
49	3062	Roo York	03/10/2016	22/10/2016	
74	5276	Roo York	24/03/2017	31/03/2017	
99	9510	Roo York	05/10/2016	22/10/2016	
124	16977	Roo York	18/03/2017	20/03/2017	

	FIRST_WORK_DATE	ACQUISITION_CHANNEL	VEHICLE_TYPE	\
24	03/04/2017	Digital	Scooter / Motorcycle	
49	30/10/2016	Offline	Bicycle	
74	01/04/2017	Organic	Scooter / Motorcycle	
99	28/10/2016	Digital	Bicycle	
124	24/03/2017	Referral	Bicycle	

	WEEKS_SINCE_FIRST_WORK	RIDER_ACTIVE	HOURS_WORKED \
24	24	True	35.087779
49	24	False	NaN
74	24	False	NaN
99	24	False	NaN
124	24	False	NaN

	HOURS_WORKED_CUMULATIVE	ORDERS_DELIVERED	ORDERS_DELIVERED_CUMULATIVE \
24	820.883044	70.0	1977.0
49	15.555832	NaN	26.0
74	23.326944	NaN	70.0
99	74.400552	NaN	97.0
124	23.710833	NaN	67.0

	THROUGHPUT	THROUGHPUT_CUMULATIVE	REFERRALS	REFERRALS_CUMULATIVE \
24	1.994997	2.408382	NaN	1.0
49	NaN	1.671399	NaN	NaN
74	NaN	3.000822	NaN	1.0
99	NaN	1.303754	NaN	NaN
124	NaN	2.825713	NaN	NaN

	SUCCESSFUL_REFERRALS	SUCCESSFUL_REFERRALS_CUMULATIVE
24	NaN	1.0
49	NaN	NaN
74	NaN	0.0
99	NaN	NaN
124	NaN	NaN

In [15]: df.tail()

Out[15]:

	RIDER_ID	LOCATION	APPLICATION_DATE	APPLICATION_APPROVED_DATE \
113424	103762	Roo York	31/03/2017	07/04/2017
113449	103764	Roo York	20/03/2017	07/04/2017
113474	103774	Roo York	26/02/2017	07/04/2017
113499	103775	Roo York	25/03/2017	07/04/2017
113524	103776	Roo York	28/03/2017	07/04/2017

	FIRST_WORK_DATE	ACQUISITION_CHANNEL	VEHICLE_TYPE \
113424	08/04/2017	Referral	Scooter / Motorcycle
113449	07/04/2017	Referral	Scooter / Motorcycle
113474	07/04/2017	Organic	Scooter / Motorcycle
113499	07/04/2017	Referral	Scooter / Motorcycle
113524	08/04/2017	Referral	Scooter / Motorcycle

	WEEKS_SINCE_FIRST_WORK	RIDER_ACTIVE	HOURS_WORKED \
113424	24	True	19.114722
113449	24	False	NaN

113474	24	False	NaN
113499	24	True	23.698056
113524	24	False	NaN

	HOURS_WORKED_CUMULATIVE	ORDERS_DELIVERED \
113424	903.356102	59.0
113449	216.875004	NaN
113474	278.383333	NaN
113499	338.583337	83.0
113524	285.996666	NaN

	ORDERS_DELIVERED_CUMULATIVE	THROUGHPUT	THROUGHPUT_CUMULATIVE \
113424	2259.0	3.086626	2.500675
113449	321.0	NaN	1.480115
113474	783.0	NaN	2.812668
113499	1107.0	3.502397	3.269505
113524	644.0	NaN	2.251775

	REFERRALS	REFERRALS_CUMULATIVE	SUCCESSFUL_REFERRALS \
113424	NaN	NaN	NaN
113449	NaN	4.0	NaN
113474	NaN	NaN	NaN
113499	NaN	5.0	NaN
113524	NaN	NaN	NaN

	SUCCESSFUL_REFERRALS_CUMULATIVE
113424	NaN
113449	2.0
113474	NaN
113499	1.0
113524	NaN

In [16]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4541 entries, 24 to 113524
Data columns (total 19 columns):
RIDER_ID                4541 non-null int64
LOCATION                  4541 non-null object
APPLICATION_DATE         4541 non-null object
APPLICATION_APPROVED_DATE 4541 non-null object
FIRST_WORK_DATE          4541 non-null object
ACQUISITION_CHANNEL      4541 non-null object
VEHICLE_TYPE             4541 non-null object
WEEKS_SINCE_FIRST_WORK   4541 non-null int64
RIDER_ACTIVE             4541 non-null bool
HOURS_WORKED             1319 non-null float64
HOURS_WORKED_CUMULATIVE  4541 non-null float64
```

```

ORDERS_DELIVERED          1223 non-null float64
ORDERS_DELIVERED_CUMULATIVE 4464 non-null float64
THROUGHPUT                1223 non-null float64
THROUGHPUT_CUMULATIVE     4464 non-null float64
REFERRALS                 74 non-null float64
REFERRALS_CUMULATIVE      1002 non-null float64
SUCCESSFUL_REFERRALS       74 non-null float64
SUCCESSFUL_REFERRALS_CUMULATIVE 1002 non-null float64
dtypes: bool(1), float64(10), int64(2), object(6)
memory usage: 678.5+ KB

```

```
In [17]: df.describe()
```

```

Out[17]:
      count  RIDER_ID  WEEKS_SINCE_FIRST_WORK  HOURS_WORKED  \
count      4541.000000          4541.0    1319.000000
mean       80873.171989           24.0     19.372840
std        12799.731993           0.0     16.926589
min         1864.000000           24.0     0.000000
25%         70838.000000           24.0     5.693751
50%         77621.000000           24.0    14.942779
75%         94220.000000           24.0    30.296527
max        103776.000000           24.0    89.775555

      count  HOURS_WORKED_CUMULATIVE  ORDERS_DELIVERED  ORDERS_DELIVERED_CUMULATIVE  \
count      4541.000000          1223.000000          4464.000000
mean       197.829756           50.528209           440.673835
std        259.929947           43.825879           621.260672
min          0.178888            1.000000            1.000000
25%         29.188333           15.000000           53.000000
50%         90.675278           38.000000          172.000000
75%        262.693056           75.000000          573.250000
max        2075.802227          251.000000         5046.000000

      count  THROUGHPUT  THROUGHPUT_CUMULATIVE  REFERRALS  REFERRALS_CUMULATIVE  \
count      1223.000000          4464.000000       74.000000          1002.000000
mean         2.403633           1.990289       1.270270           2.589820
std          0.717144           0.667197       2.216361           9.369239
min          0.056955           0.025861       0.000000           0.000000
25%          1.965016           1.568525       1.000000           1.000000
50%          2.440885           2.051430       1.000000           1.000000
75%          2.888611           2.461074       1.000000           3.000000
max          5.396876           5.191057      19.000000          285.000000

      count  SUCCESSFUL_REFERRALS  SUCCESSFUL_REFERRALS_CUMULATIVE
count         74.000000          1002.000000
mean          0.310811           0.561876
std           0.547114           1.215396

```

min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	1.000000	1.000000
max	2.000000	28.000000

The new dataset, `df`, is now only 1/25 of it's original size, containing only the final row for each Rider and their total performance in the recorded metrics. The original dataset is still accessible using the DataFrame `df_raw`.

Next we will explore the new DataFrame and plot the key metrics for further analysis.

1.6 4. Exploratory Data Analysis

The exploratory data analysis aims to answer the question, **“Is the RGR scheme successful, and if so, how have you reached that conclusion?”**.

As per the project brief, I will separate my answer to this section into the following three sub-sections: * Defining what success is (Section ??) * How has RGR performed compared with other channels? (Section ??) * How does rider performance / behaviour vary between the channels? (Section ??)

We will be using the [Pandas](#) library for DataFrames and the [Seaborn](#) library for plotting.

1.6.1 4.1. Definition of a successful RGR scheme

Some gig economy startups can have a turnover of staff as high as 500% per year ([source 1](#)) and ([source 2](#)). It is therefore important for these companies to focus on recruitment, how it can be done smarter and at less cost.

It is difficult to define what success for an RGR scheme would be, I would consider the following three components: * a system whereby current staff recommend a sufficient number potential employees to the application process, * of those candidates that apply, a significant percentage meet the standards to be eligible to work for the company, and * of these potential employees, those that are hired perform go on to meet the required standards and expected commitment to the role.

These three requirements are analysed in much further detail in the following sections Section ?? and Section ??.

To throw out some rough numbers, if for every two members of staff, one potential employee can be referred and then of those candidates, 1/4 are successful, I would say that is quite a successful recruitment process.

Let's take a look at the rough numbers for our dataset:

```
In [18]: total_riders = df['RIDER_ID'].count()
total_referrals = df['REFERRALS_CUMULATIVE'].sum()
total_successful_referrals = df['SUCCESSFUL_REFERRALS_CUMULATIVE'].sum()

print("Total Riders: " + str(total_riders))
print("Total Referrals: " + str(total_referrals))
print("Total Successful Referrals: " + str(total_successful_referrals))

print("Rate of Successful Referrals to Referrals (%): " + str(round((total_successful_referrals / total_referrals) * 100)))
print("Rate of Successful Referrals to Riders (%): " + str(round((total_successful_referrals / total_riders) * 100)))
```

Total Riders: 4541
Total Referrals: 2595.0
Total Successful Referrals: 563.0
Rate of Successful Referrals to Referrals (%): 21.7%
Rate of Successful Referrals to Riders (%): 12.4%

Looking at the overall dataset, of the 4,531 Riders, 2,595 referrals were made and of these referrals, 563 were successful. That means that for every candidate, 0.573 candidates are referred and 0.124 successful candidates are referred. Of the candidates that are referred, the success rate is about 21.7% i.e. 1 in 5. Looking at these rough numbers, I would suggest that Deliveroo's RGR scheme is extremely productive. We will however dive much deeper into these numbers in the following sections.

To make better comments on defining what is a successful RGR scheme, I would like to know what the turnover rate of Riders for Deliveroo and the average length of employment of Riders with the company, for example.

1.6.2 4.2. How has RGR performed compared with other channels?

Let's plot the data for Riders by Acquisition Channel to see the volumes of Riders recruited by each different channel. These are: * Digital * Job Platforms * Offline * Organic * Referral

Plot the data

```
In [19]: # Set the width and height of the figure
plt.figure(figsize=(8,5))

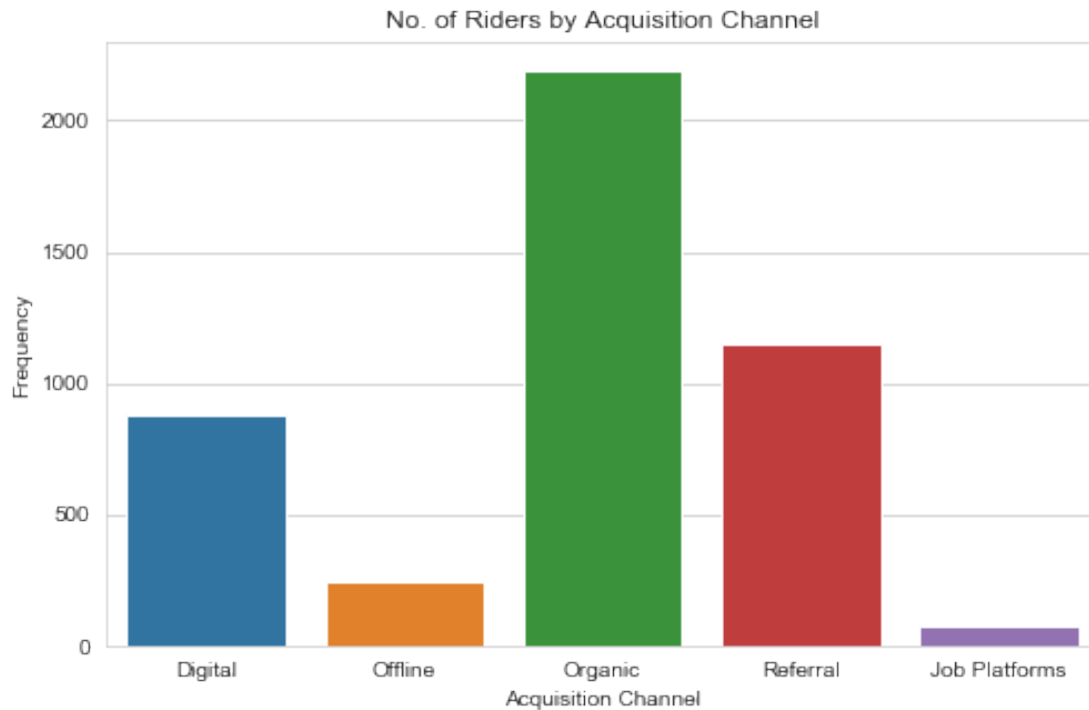
# Plot style
sns.set_style("whitegrid")

# Add title
plt.title("No. of Riders by Acquisition Channel")

# Bar chart showing
sns_plot_4_2 = sns.countplot(df['ACQUISITION_CHANNEL'])

# Add labels for axes
plt.xlabel("Acquisition Channel")
plt.ylabel("Frequency")

# Save plot
fig_4_2 = sns_plot_4_2.get_figure()
fig_4_2.savefig("./figures/4_2.png")
```



```
In [20]: print(df['ACQUISITION_CHANNEL'].describe())
```

```
count      4541
unique         5
top      Organic
freq       2190
Name: ACQUISITION_CHANNEL, dtype: object
```

```
In [21]: print(df.groupby('ACQUISITION_CHANNEL')[['RIDER_ID']].count())
```

	RIDER_ID
ACQUISITION_CHANNEL	
Digital	882
Job Platforms	77
Offline	242
Organic	2190
Referral	1150

Analysis The data shows that of the 4,541 total Riders in the dataset, 1,150 joined Deliveroo through the Referral channel, making up just over a quarter (25.3%) of the total workforce. This is the second most popular Acquisition Channel after Organic, which makes up 2,190 of the Riders in the dataset (48.2%).

The key thing to notice here is that even though the Referral scheme is the second best Acquisition Measure, it is actually the best performing channel in which Deliveroo is actively involved in when driving recruitment. Half of the Riders in the dataset came are said to have come to Deliveroo Organically which presumably means that Deliveroo had no active participation in finding the Riders before their application.

When comparing channel to Job Platforms, presumably sites like [LinkedIn](#), [Indeed](#), [Monster](#), [Reed](#) etc., this channel recruited only 77 Riders (1.70% of the Riders in the dataset). The next best Acquisition Channel is Digital, making up a considerable number of the Riders with 882 (19.4% of the dataset). Offline makes up the remaining 242 Riders (5.33%).

Considering quarter of the Riders came to Deliveroo through Referrals and it is the best performing active measure, I would say I would say that the RGR scheme is performing very well and is a key component for Deliveroo maintaining its staff and high performance levels regarding the delivery of the food.

1.6.3 4.3. How does Rider performance / behaviour vary between the channels?

Wrangle the data For this section, before we can make any plots, we need to wrangle the data to a form that's more suited to our needs. We need to take the existing DataFrame of Riders data for the 24th week, `df`, and use the `groupby` and `aggregate` methods to create a new DataFrame, `df_channel`, that counts the number of Riders in each channel by unique rider id and the total sum for key performance metrics, defined below.

For this exercise, the first performance metrics are: Hours Worked, Cumulative Orders Delivered, Orders Delivered Cumulative, and Cumulative Throughput. We will also next look at Cumulative Referrals, and Cumulative Successful Referrals to see the 'performance' of the Riders regarding the referring new candidates to the business.

Using this new DataFrame, we can produce calibrated metrics (discussed further in next steps) taking averages per Rider and use these create plots that determine the performance and behaviour of the Riders by channel.

The code for this is as follows:

```
In [22]: # Created groupedby DataFrame, by channel
df_channel = df.groupby(
    ['ACQUISITION_CHANNEL']
).agg(
    {
        'RIDER_ID': 'count',           # the total number of Riders
        'HOURS_WORKED_CUMULATIVE': 'sum', # the sum total of cumulative hours
        'ORDERS_DELIVERED_CUMULATIVE': 'sum', # the sum total of cumulative deliveries
        'THROUGHPUT_CUMULATIVE': 'sum', # the sum total of cumulative throughput
        'REFERRALS_CUMULATIVE': 'sum', # the sum total of cumulative referrals
        'SUCCESSFUL_REFERRALS_CUMULATIVE': 'sum' # the sum total of cumulative successful referrals
    }
).reset_index()

# Rename the RIDER_ID column to TOTAL_RIDERS
df_channel = df_channel.rename(columns={'RIDER_ID': 'TOTAL_RIDERS'})

df_channel
```



```

Out [22]:  ACQUISITION_CHANNEL  TOTAL_RIDERS  HOURS_WORKED_CUMULATIVE  \
0           Digital          882          140313.938052
1      Job Platforms          77          14923.873923
2           Offline         242          37820.421062
3           Organic        2190          407886.645300
4           Referral        1150          297400.045266

          ORDERS_DELIVERED_CUMULATIVE  THROUGHPUT_CUMULATIVE  REFERRALS_CUMULATIVE  \
0                292481.0                1625.277691                533.0
1                32032.0                139.768841                 6.0
2                79032.0                434.241888                113.0
3               924513.0               4453.435329                962.0
4               639110.0               2231.928393                981.0

          SUCCESSFUL_REFERRALS_CUMULATIVE
0                      75.0
1                      3.0
2                     17.0
3                     193.0
4                     275.0

```

```
In [23]: df_channel.dtypes
```

```

Out [23]: ACQUISITION_CHANNEL      object
TOTAL_RIDERS                      int64
HOURS_WORKED_CUMULATIVE          float64
ORDERS_DELIVERED_CUMULATIVE      float64
THROUGHPUT_CUMULATIVE           float64
REFERRALS_CUMULATIVE            float64
SUCCESSFUL_REFERRALS_CUMULATIVE  float64
dtype: object

```

As previously discussed in Section ?? and again in the grouped table, `df_channel`, we have different numbers of Riders for each of the Acquisition Channels. We will therefore calibrate the metrics in question regarding performance, these being: `HOURS_WORKED_CUMULATIVE`, `ORDERS_DELIVERED_CUMULATIVE`, `THROUGHPUT_CUMULATIVE`, `REFERRALS_CUMULATIVE`, and `SUCCESSFUL_REFERRALS_CUMULATIVE`. These will be calibrated by taken an average per rider i.e. dividing the sum total by the number of Riders. These new metrics can be identified with the suffix `'_PR'` (per rider) e.g. `HOURS_WORKED_CUMULATIVE_PR` is the cumulative hours worked per rider.

```

In [24]: # Create per Rider (PR) columns
df_channel['HOURS_WORKED_CUMULATIVE_PR'] = df_channel['HOURS_WORKED_CUMULATIVE'] / df_channel['TOTAL_RIDERS']
df_channel['ORDERS_DELIVERED_CUMULATIVE_PR'] = df_channel['ORDERS_DELIVERED_CUMULATIVE'] / df_channel['TOTAL_RIDERS']
df_channel['THROUGHPUT_CUMULATIVE_PR'] = df_channel['THROUGHPUT_CUMULATIVE'] / df_channel['TOTAL_RIDERS']
df_channel['REFERRALS_CUMULATIVE_PR'] = df_channel['REFERRALS_CUMULATIVE'] / df_channel['TOTAL_RIDERS']
df_channel['SUCCESSFUL_REFERRALS_CUMULATIVE_PR'] = df_channel['SUCCESSFUL_REFERRALS_CUMULATIVE'] / df_channel['TOTAL_RIDERS']

```

```
# Drop unnamed channel (created 2nd index)
df_channel = df_channel.loc[:, ~df_channel.columns.str.contains('^Unnamed')]
```

```
In [25]: df_channel.head()
```

```
Out [25]:
```

	ACQUISITION_CHANNEL	TOTAL_RIDERS	HOURS_WORKED_CUMULATIVE	\
0	Digital	882	140313.938052	
1	Job Platforms	77	14923.873923	
2	Offline	242	37820.421062	
3	Organic	2190	407886.645300	
4	Referral	1150	297400.045266	

	ORDERS_DELIVERED_CUMULATIVE	THROUGHPUT_CUMULATIVE	REFERRALS_CUMULATIVE	\
0	292481.0	1625.277691	533.0	
1	32032.0	139.768841	6.0	
2	79032.0	434.241888	113.0	
3	924513.0	4453.435329	962.0	
4	639110.0	2231.928393	981.0	

	SUCCESSFUL_REFERRALS_CUMULATIVE	HOURS_WORKED_CUMULATIVE_PR	\
0	75.0	159.086098	
1	3.0	193.816544	
2	17.0	156.282732	
3	193.0	186.249610	
4	275.0	258.608735	

	ORDERS_DELIVERED_CUMULATIVE_PR	THROUGHPUT_CUMULATIVE_PR	\
0	331.611111	1.842718	
1	416.000000	1.815180	
2	326.578512	1.794388	
3	422.152055	2.033532	
4	555.747826	1.940807	

	REFERRALS_CUMULATIVE_PR	SUCCESSFUL_REFERRALS_CUMULATIVE_PR
0	0.604308	0.085034
1	0.077922	0.038961
2	0.466942	0.070248
3	0.439269	0.088128
4	0.853043	0.239130

You can now see the new DataFrame with the new per Rider metrics, including HOURS_WORKED_CUMULATIVE_PR, ORDERS_DELIVERED_CUMULATIVE_PR, THROUGHPUT_CUMULATIVE_PR, REFERRALS_CUMULATIVE_PR, SUCCESSFUL_REFERRALS_CUMULATIVE_PR, and ORDERS_DELIVERED_PER_HOUR_PR.

Before plotting this data, let's export the grouped and aggregated dataset in case we, or other analysts, would like to use it in future projects/analysis.

```
In [26]: df_channel.to_csv(r'./data/rgr_take_home_v3_dataset_grouped.csv')
```

Plot the data

Plot 1: Average No. of Hours Worked per Rider by Acquisition Channel after 24 weeks

```
In [27]: # Set the width and height of the figure
plt.figure(figsize=(8,5))

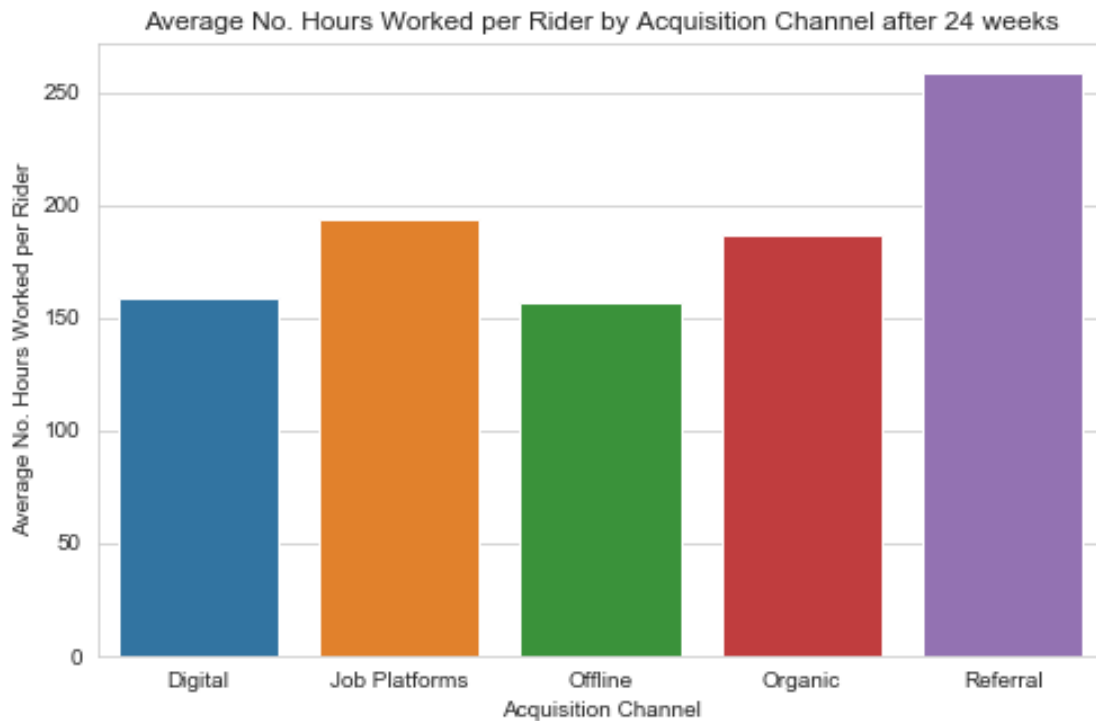
# Plot style
sns.set_style("whitegrid")

# Add title
plt.title("Average No. Hours Worked per Rider by Acquisition Channel after 24 weeks")

# Bar chart showing the Average No. Hours Worked per Rider by Acquisition Channel aft
sns_plot_4_3a = sns.barplot(x=df_channel['ACQUISITION_CHANNEL'], y=df_channel['HOURS_W

# Add labels for axes
plt.xlabel("Acquisition Channel")
plt.ylabel("Average No. Hours Worked per Rider")

# Save plot
fig_4_3a = sns_plot_4_3a.get_figure()
fig_4_3a.savefig("./figures/4_3a.png")
```



Plot 2: Average No. of Orders Delivered per Rider by Acquisition Channel after 24 weeks

```
In [28]: # Set the width and height of the figure
plt.figure(figsize=(8,5))

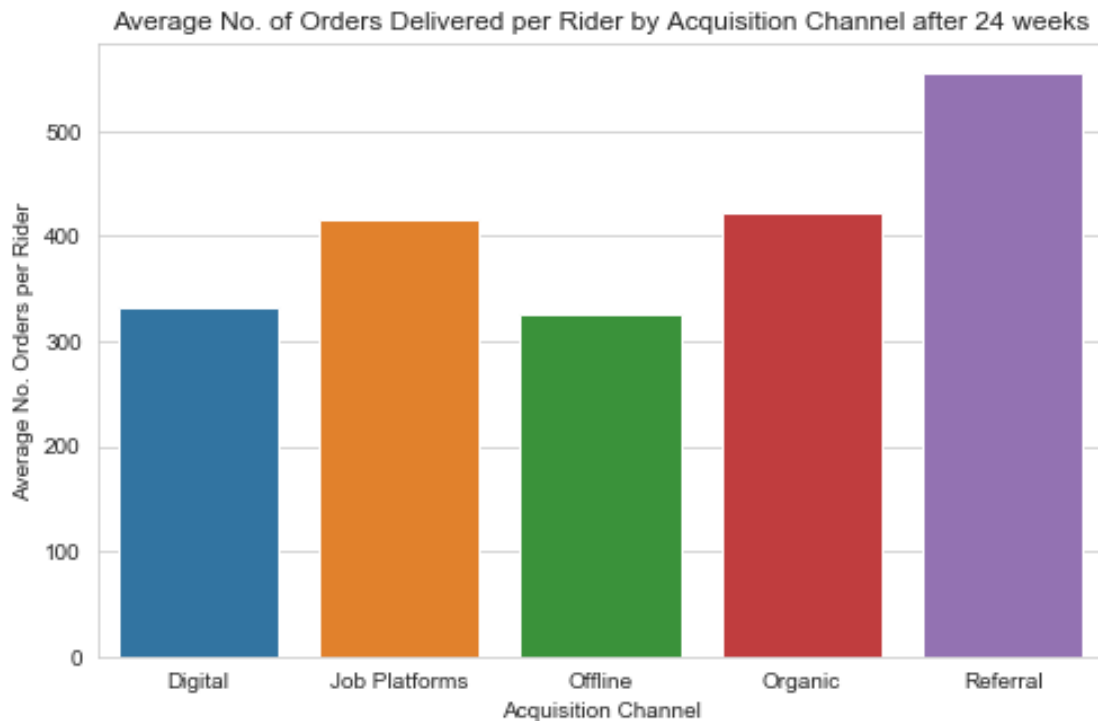
# Plot style
sns.set_style("whitegrid")

# Add title
plt.title("Average No. of Orders Delivered per Rider by Acquisition Channel after 24 weeks")

# Bar chart showing the Average No. of Orders Delivered per Rider by Acquisition Channel
sns_plot_4_3b = sns.barplot(x=df_channel['ACQUISITION_CHANNEL'], y=df_channel['ORDERS_DELIVERED'])

# Add labels for axes
plt.xlabel("Acquisition Channel")
plt.ylabel("Average No. Orders per Rider")

# Save plot
fig_4_3b = sns_plot_4_3b.get_figure()
fig_4_3b.savefig("./figures/4_3b.png")
```



Plot 3: Average Throughput per Rider by Acquisition Channel after 24 weeks

```

In [29]: # Set the width and height of the figure
plt.figure(figsize=(8,5))

# Plot style
sns.set_style("whitegrid")

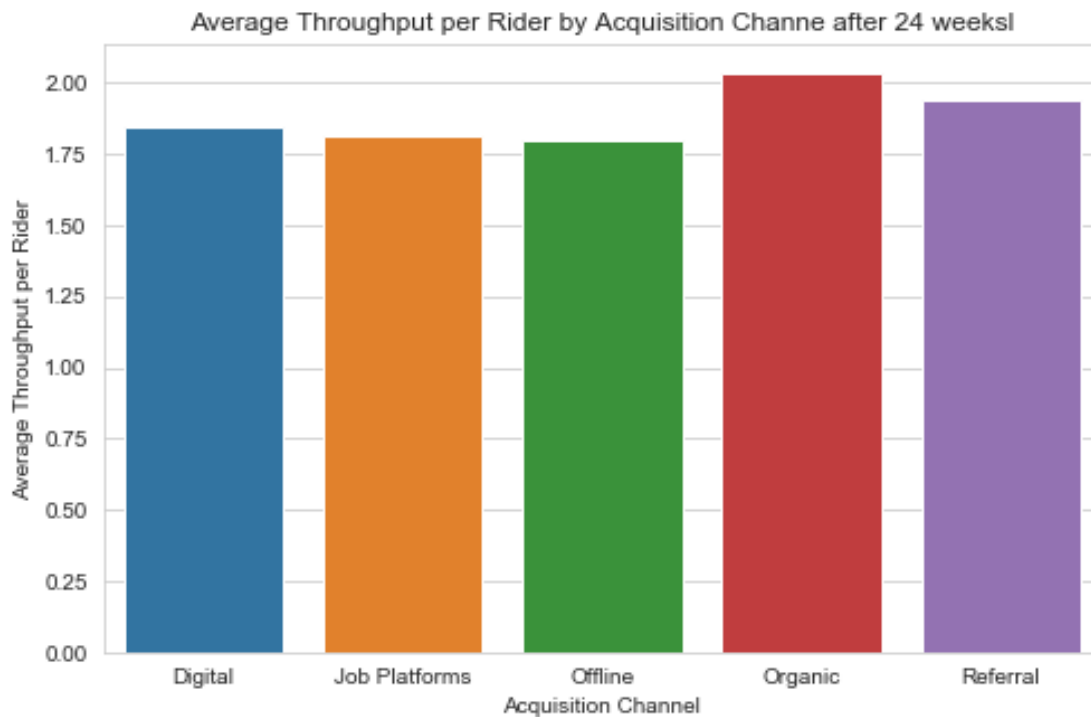
# Add title
plt.title("Average Throughput per Rider by Acquisition Channel after 24 weeks1")

# Bar chart showing the Average Throughput per Rider by Acquisition Channel after 24 weeks1
sns_plot_4_3c = sns.barplot(x=df_channel['ACQUISITION_CHANNEL'], y=df_channel['THROUGHPUT'])

# Add labels for axes
plt.xlabel("Acquisition Channel")
plt.ylabel("Average Throughput per Rider")

# Save plot
fig_4_3c = sns_plot_4_3c.get_figure()
fig_4_3c.savefig("./figures/4_3c.png")

```



Plot 4: Average No. of Referrals per Rider by Acquisition Channel after 24 weeks

```

In [30]: # Set the width and height of the figure
plt.figure(figsize=(8,5))

```

```

# Plot style
sns.set_style("whitegrid")

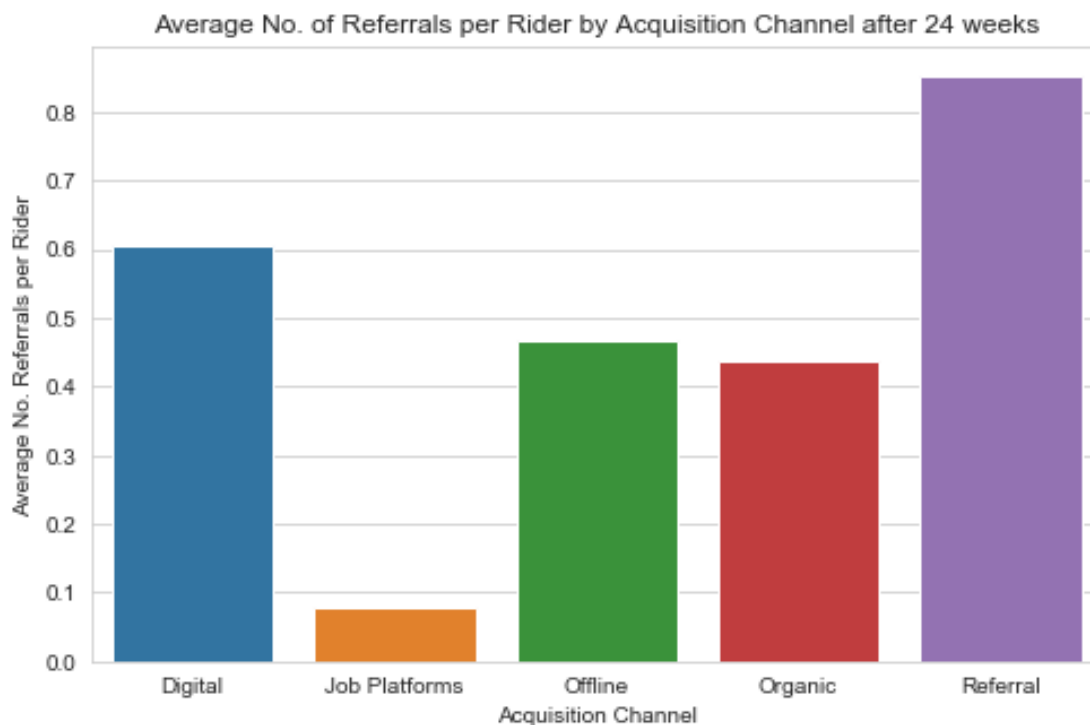
# Add title
plt.title("Average No. of Referrals per Rider by Acquisition Channel after 24 weeks")

# Bar chart showing the Average No. of Referrals per Rider by Acquisition Channel aft
sns_plot_4_3d = sns.barplot(x=df_channel['ACQUISITION_CHANNEL'], y=df_channel['REFERR

# Add labels for axes
plt.xlabel("Acquisition Channel")
plt.ylabel("Average No. Referrals per Rider")

# Save plot
fig_4_3d = sns_plot_4_3d.get_figure()
fig_4_3d.savefig("./figures/4_3d.png")

```



Plot 5: Average No. of Successful Referrals per Rider by Acquisition Channel after 24 weeks

```

In [31]: # Set the width and height of the figure
plt.figure(figsize=(8,5))

# Plot style

```

```

sns.set_style("whitegrid")

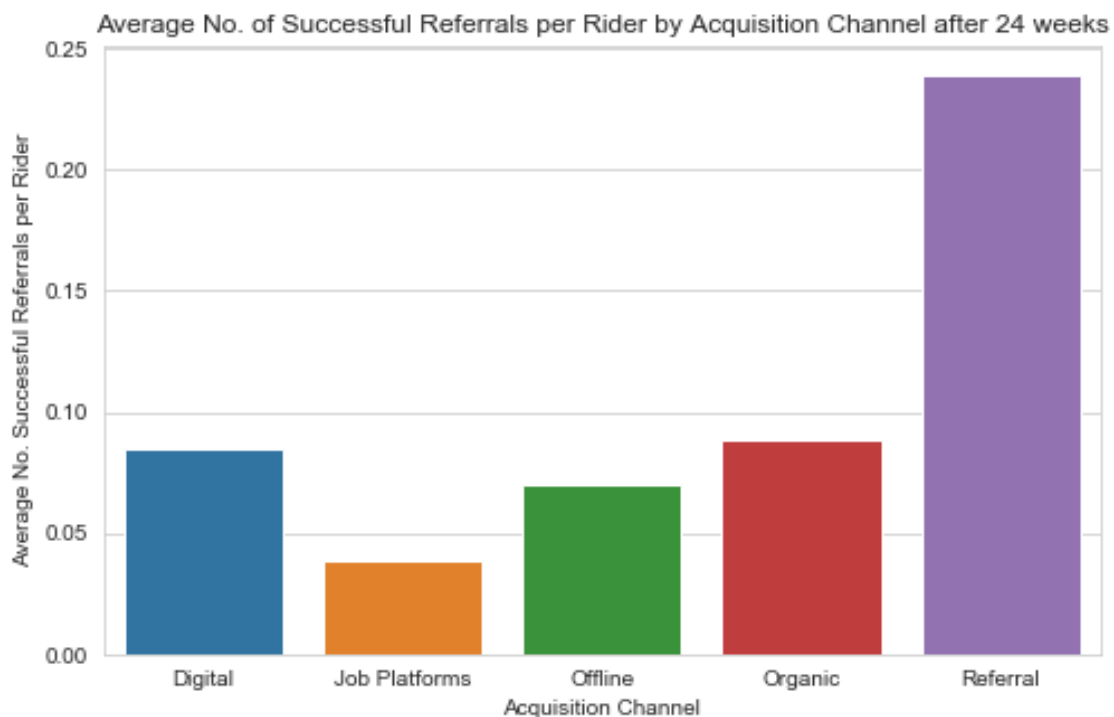
# Add title
plt.title("Average No. of Successful Referrals per Rider by Acquisition Channel after 24 weeks")

# Bar chart showing the Average No. of Successful Referrals per Rider by Acquisition Channel
sns_plot_4_3e = sns.barplot(x=df_channel['ACQUISITION_CHANNEL'], y=df_channel['SUCCESSFUL_REFERRALS'])

# Add labels for axes
plt.xlabel("Acquisition Channel")
plt.ylabel("Average No. Successful Referrals per Rider")

# Save plot
fig_4_3e = sns_plot_4_3e.get_figure()
fig_4_3e.savefig("./figures/4_3e.png")

```



Analysis All analysis focuses on the Referral Acquisition Channel, comparing this to the other channels. Each of the five previous plots has a written paragraph, interpreting the results.

Plot 1: Average No. of Hours Worked per Rider by Acquisition Channel For the first plot, the Average No. of Hours Worked per Rider by Acquisition Channel, the data shows that Riders who joined Deliveroo by Referral came out on top, with an average no. of 258.6 total hours worked per Rider after 24 weeks, averaging at 10.8 hours a week. For reference, Riders work on average 15

hours a week in the UK ([source](#)). Of the total Hours Worked by all Riders in the dataset, this makes up 27.1% of total Hours Worked, suggesting that the Referral Riders are the hardest working and most committed demographic. The four other platform's share of the total percentage ranges between 16.7% and 20.3%, the next best channel being by Job Platform, with an average hours worked of 193.8 (20.3%).

Plot 2: Average No. of Orders Delivered per Rider by Acquisition Channel For the second plot, the Average No. of Orders Delivered per Rider by Acquisition Channel, the data shows that Riders who joined Deliveroo by Referral came out on top again, with an average no. of 555.7 orders delivered per Rider after 24 weeks. Of the total Orders Delivered by all Riders in the dataset, this makes up 27.1% of the total Orders Delivered, suggesting that the Referral Riders are the most productive demographic. The four other platform's share of the total percentage ranges between 15.9% and 20.6%, the next best channel being the Organic joiners, with an average Orders Delivered of 422.0 (20.6%).

Plot 3: Throughput per Rider by Acquisition Channel As a quick reminder before discussing the third plot, the Cumulative Throughput is the cumulative orders per hour (orders / hours) of the Rider.

As we can see from the data, the Average Throughput per Rider by Acquisition Channel of Riders who joined Deliveroo by Referral again came out very strongly, overall placing second with an average Throughput per Rider of 2.03. Of the total sum of Throughput per Rider in the dataset, this makes up 20.6%, suggesting that the Referral Riders are the some of the most product Riders and are very efficient in delivering the most orders per hour, as possible. The other platform's share of the total percentage are all very close, ranging between 19.0% (Offline) and 21.6% (Organic).

My instinct with this observation is that the Vehicle Type may be a more interesting indicator to use when looking at the Throughput of the Riders. However, I believe it's interesting analysis and has some value being in this report.

Plot 4: Average No. of Referrals per Rider by Acquisition Channel For the forth plot, the Average No. of Referrals per Rider by Acquisition Channel, the data shows that Riders who joined Deliveroo by Referral came out on top again, with an average no. of 0.853 Referrals per Rider after 24 weeks. Of the total Referrals by all Riders in the dataset, this makes up 34.9% of the total Referrals, suggesting that the Referral Riders are far more likely to Refer new Riders to the company. The four other platform's share of the total percentage ranges between 3.19% (Job Platforms) and 24.8% (Digital).

Plot 5: Average No. of Successful Referrals per Rider by Acquisition Channel For the fifth plot, the Average No. of Successful Referrals per Rider by Acquisition Channel, the data shows that Riders who joined Deliveroo by Referral continued to observe the previous trends, again leading the channels with an average no. of 0.239 Successful per Rider after 24 weeks. Of the total Successful Referrals by all Riders in the dataset, this makes up 45.9% of the total Successful Referrals, suggesting that not only are the Referral Riders are far more likely to Refer other Riders to the company (as seen in plot 4), but they are also far more likely to refer successful candidates. The four other platform's share of the total percentage ranges between 7.47% (Job Platforms) and 16.9% (Organic).

1.7 5. Summary and Conclusion

To answer the question “Is the RGR scheme successful and why?” using our limited dataset and restricted time for analysis, I believe that the answer is a resounding Yes!

Our analysis was divided into three sections: 1) We defined what a successful RGR scheme is; 2) We looked at how the RGR scheme performed compared with other channels; and 3) We analysed how Rider’s performance and behaviour depending on their channel of recruitment.

We first looked at the 4,541 Riders and took a high level view to see through which channel they had come through to the company. The data showed that 1,150 employees joined Deliveroo through the Referral channel, making up just over a quarter (25.3%) of the workforce in the dataset, second only to Organic (48.2%), a demographic which is inherently out of Deliveroo’s control. This therefore suggests that Referrals is Deliveroo’s best active form of recruitment.

We next took our DataFrame and calibrated the key performance metrics of the Riders to determine average ‘per Rider’ values over the 24 week period for: Hours Worked, Orders Delivered, Orders Delivered, Throughput, Referrals, and Successful Referrals. Our plots show that in all but one of the categories, Referred Riders outperform other Riders in regarding these metrics, except for Throughput where Referrals placed second and as discussed in section 4.3, all channels performed quite similarly and I have suspicious that it’s the form of transport that would have greater effect on improving this number i.e. using a car and not a bike.

To conclude, not only is ‘Rider Get Rider’ scheme a great way for Deliveroo to find new employees, the data suggests that these employees are some of the most productive, hardest working and efficient Riders in the business. The data also suggests that Riders who joined the business through the Referral scheme are more likely to go on to refer more future Riders and with a greater success rate.

1.8 6. Next Steps

This workbook was a short exploratory data analysis for Deliveroo Riders.

Some of the metrics and avenues not touched but may be of interest include: * Further address our definition of success using additional data of Riders including the turnover rate of Rider, the average length of employment of Riders with the company, for example. * Further analysis of the Rider’s VEHICLE_TYPE, especially when considering it’s effect on THROUGHPUT i.e. do car drivers deliver more orders per hours than cyclists. * Further analysis of the Rider LOCATION. We could maybe create a heatmap of Location vs. Acquisition Channel and see the hot spots. * Further analysis of the Rider’s RIDER_ACTIVITY i.e. TRUE or FALSE. Does further inspection show any patterns?

1.9 7. Bibliography

1. <https://www.businessinsider.com/uber-lyft-drivers-livable-wage-complaints-2019-5?r=US&IR=T>
2. <https://www.td.org/insights/has-the-gig-economy-jumped-the-shark>
3. <https://roocommunity.com/tech-round-up-statistics/>

Visit my website EddWebster.com or my [GitHub Repository](#) for more projects. If you’d like to get in contact, my email is: edd.j.webster@gmail.com.

Section ??