2.0 EDA - Customer Entity

August 12, 2021

1 Exploratory Data Analysis - Customer Entity

This notebook documents the Exploratory Data Analysis steps for this project. Specifically, I will work on the `Customer` entity as to study the behavior of the customer base to verify some hypothesis regarding which customers we would like to include in one of the goals of this project: *creating a fidelity program* and identifying *similarities* between customers.

[9]:
```
!pip install seaborn >> ../configs/package_installation.txt
```

[6]:
```
%load_ext autoreload
%load_ext lab_black
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
The lab_black extension is already loaded. To reload it, use:
  %reload_ext lab_black
```

[10]:
```
###### Loading the necessary libraries #########

# data processing and wrangling:
import pandas as pd
import numpy as np
import re
import unicodedata
import inflection
import warnings

# statistical libraries
from scipy import stats

# data and statistical visualization:
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import HTML, Image
import plotly.express as px

# sklearn
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

# setting global parameters for visualizations:
warnings.filterwarnings("ignore")
pd.set_option("display.precision", 4)
pd.set_option("display.float_format", lambda x: "%.2f" % x)
```

2 0. Utility Functions

[98]:
```
def set_plot_settings():
    """Helper function to set standard plot settings for the jupyter notebook
    Note: these are hard-coded for my specific tastes
    """
    %matplotlib inline
    %pylab inline

    plt.rcParams["figure.figsize"] = [6, 4]
    plt.rcParams["figure.dpi"] = 120
    display(HTML("<style>.container { width:100% !important; }</style>"))
    pd.set_option("display.expand_frame_repr", False)
    sns.set_style("white")


def rotate_xticks(ax, rotation):
    """Helper function to rotate x-axis labels of a matplotlib-based plot

    Args:
        ax (axes.AxesSubplot): matplotlib subplot axis object that handles the␣
↪figure
        rotation (int): degrees for which to rotate the axis x-ticks

    Returns:
        None

    """
    for item in ax.get_xticklabels():
        item.set_rotation(rotation)


def get_avg_coordinates(df, x, y):
    """Function to get average coordinates of two given variables in a dataframe

    Args:
        df (pandas.DataFrame): dataset where the variables can be found
        x (str): variable in the x axis
```

```
        y (str): variable in the y axis

    Returns:
        coordinates (dict): a dictionary with name of the variable being key␣
↪and the average being the value

    """

    # unpacking mean values for indices and values
    x_coord, y_coord = df_customer[[x, y]].mean().values

    x_name, y_name = df_customer[[x, y]].mean().index

    # average coordinate output format
    coordinates = {x_name: x_coord, y_name: y_coord}

    return coordinates


def test_hypothesis(test_sample, control_sample, alternative, test_type,␣
↪ecdf=True):
    """Function to perform a curated set of hypothesis tests with support plots

    Args:
        test_sample (numpy.array): array containing the samples of the test␣
↪sample
        control_sample (numpy.array): array containing the samples of the␣
↪control sample
        alternative (str): type of alternative hypothesis
        test_type (str): the type of test that needs to be performed (either␣
↪Kolmogorov-Smirnov test or Mann-Whitney rank test)
        ecdf (bool): if True, the support graphic will be an ECDF plot

    Returns:
        None (prints to standard output)

    """
    if test_type == "ks":
        results = stats.ks_2samp(test_sample, control_sample,␣
↪alternative=alternative)

    else:
        results = stats.mannwhitneyu(
            test_sample, control_sample, alternative=alternative
        )
```

```
    p_value = results.pvalue
    statistic = results.statistic

    print(f"Test statistic = {statistic}")
    print(f"P-value = {p_value}")

    if ecdf == True:

        ax1 = sns.ecdfplot(test_sample, label="Test Sample")

        ax2 = sns.ecdfplot(control_sample, label="Control Sample")

        ax1.legend()
        ax2.legend()

    else:
        ax1 = sns.kdeplot(test_sample, label="Test Sample")

        ax2 = sns.kdeplot(control_sample, label="Control Sample")

        ax1.legend()
        ax2.legend()

    if p_value <= 0.05:
        print(f"H1 True: Test Sample {alternative.upper()} than Control")

    else:
        print(f"H1 False: Test Sample NOT {alternative} than Control")
```

[99]:
```
# setting the global variables for plotting:
set_plot_settings()
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
<IPython.core.display.HTML object>
```

3 1. Loading and Preparing the Data

[100]:
```
# instantiating the sqlite database:
df = pd.read_parquet("../data/processed/tb_customer.parquet")
```

[101]:
```
# verifying the data types:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5373 entries, 0 to 5372
Data columns (total 37 columns):
 #   Column                                  Non-Null Count  Dtype
```

```
---  ------                                     --------------  -----
 0   customer_id                                5373 non-null   int32
 1   customer_country                           5373 non-null   object
 2   first_purchase_date                        5373 non-null   object
 3   is_foreign                                 5373 non-null   bool
 4   account_age_days                           5373 non-null   int32
 5   recency                                    5373 non-null   int32
 6   n_orders                                   5373 non-null   int64
 7   gross_revenue                              5373 non-null   float64
 8   total_cancelled                            5373 non-null   float64
 9   frequency                                  5373 non-null   float64
 10  monetary_value                             5373 non-null   float64
 11  average_ticket                             5373 non-null   float64
 12  is_considered_reseller                     5373 non-null   bool
 13  average_time_between_purchases             3059 non-null   float64
 14  average_time_to_next_bank_holiday          5373 non-null   int64
 15  average_time_to_next_commercial_holiday    5373 non-null   int64
 16  month_most_active                          5373 non-null   int32
 17  week_most_active                           5373 non-null   int32
 18  day_of_week_most_active                    5373 non-null   int32
 19  day_of_month_most_active                   5373 non-null   int32
 20  average_basket_size                        5373 non-null   float64
 21  average_basket_diversity                   5373 non-null   float64
 22  total_items                                5373 non-null   int32
 23  total_cancelled_items                      5373 non-null   int64
 24  total_free_items                           5373 non-null   int64
 25  total_returned_items                       5373 non-null   int64
 26  total_sale_items                           5373 non-null   int64
 27  total_discounts_received                   5373 non-null   float64
 28  total_paid_fees                            5373 non-null   float64
 29  total_paid_manual                          5373 non-null   float64
 30  total_paid_postage                         5373 non-null   float64
 31  total_paid_returned                        5373 non-null   float64
 32  total_paid_sale                            5373 non-null   float64
 33  total_units_cancelled                      5373 non-null   float64
 34  total_units_free                           5373 non-null   float64
 35  total_units_returned                       5373 non-null   float64
 36  total_units_sale                           5373 non-null   float64
dtypes: bool(2), float64(18), int32(8), int64(7), object(2)
memory usage: 1.3+ MB
```

[102]:
```
# setting the customer_id as the index:
df = df.set_index("customer_id")

# fixing the data types:
df.loc[:, "first_purchase_date"] = pd.to_datetime(df["first_purchase_date"])
```

[103]:
```
# visualizing the dataset:
df.head()
```

[103]:
```
              customer_country first_purchase_date  is_foreign  account_age_days
recency  n_orders  gross_revenue  total_cancelled  frequency  monetary_value
…  total_discounts_received  total_paid_fees  total_paid_manual
total_paid_postage  total_paid_returned  total_paid_sale  total_units_cancelled
total_units_free  total_units_returned  total_units_sale
customer_id
…
12940          united kingdom          2017-09-11        False                 88
47          4          950.79          37.25       0.13          913.54  …
0.00            0.00              0.00                0.00
0.00            5.45                5.00              200.00
0.00             1.00
13285          united kingdom          2017-02-20        False                291
24          4         2709.12           0.00       0.13         2709.12  …
0.00            0.00              0.00                0.00
0.00           95.70                0.00              958.00
0.00            46.00
13623          united kingdom          2017-02-13        False                298
31          7          823.12          75.34       0.23          747.78  …
0.00            0.00             43.80                0.00
0.00          198.90                8.00               71.00
0.00            22.00
13832          united kingdom          2017-11-18        False                 20
18          2           63.45          11.25       0.07           52.20  …
0.00            0.00              0.00                0.00
0.00            0.00                3.00                0.00
0.00             0.00
14450          united kingdom          2017-01-21        False                321
181          3          483.25           0.00       0.10          483.25  …
0.00            0.00              0.00                0.00
0.00            0.00                0.00              104.00
0.00             0.00

[5 rows x 36 columns]
```

[104]:
```
# verifying the column data types:
for col, dtype in dict(zip(df.dtypes.index, df.dtypes.values)).items():
    print(f"{col} -> {dtype}")
```

```
customer_country -> object
first_purchase_date -> datetime64[ns]
is_foreign -> bool
account_age_days -> int32
recency -> int32
n_orders -> int64
```

```
gross_revenue -> float64
total_cancelled -> float64
frequency -> float64
monetary_value -> float64
average_ticket -> float64
is_considered_reseller -> bool
average_time_between_purchases -> float64
average_time_to_next_bank_holiday -> int64
average_time_to_next_commercial_holiday -> int64
month_most_active -> int32
week_most_active -> int32
day_of_week_most_active -> int32
day_of_month_most_active -> int32
average_basket_size -> float64
average_basket_diversity -> float64
total_items -> int32
total_cancelled_items -> int64
total_free_items -> int64
total_returned_items -> int64
total_sale_items -> int64
total_discounts_received -> float64
total_paid_fees -> float64
total_paid_manual -> float64
total_paid_postage -> float64
total_paid_returned -> float64
total_paid_sale -> float64
total_units_cancelled -> float64
total_units_free -> float64
total_units_returned -> float64
total_units_sale -> float64
```

4 2. Univariate Analysis

With the relevant columns identified, I will analyze the dataset's features in order to find a good way to segment the customers. The hypothesis map I developed below helps identifiying possible ways of tackling the segmentation problems.

In an E-commerce setting (especially a website that focuses on a specific category of retail, such as novelty and gifts), there are many kinds of clients. There are the seasonal clients and the regulars. Wholesalers and people who open an account to buy a single item. To identify customers how to make customers more rentable, these different nuances must be identified and properly handled.

I start by looking at the distributions of the variables in the dataset to identify general trends.

[105]:
```python
# extracting the numerical variables:
num_cols = sorted(
    list(set(df.select_dtypes(include=["int64", "float64"]).columns)),␣
 ↪reverse=True
```

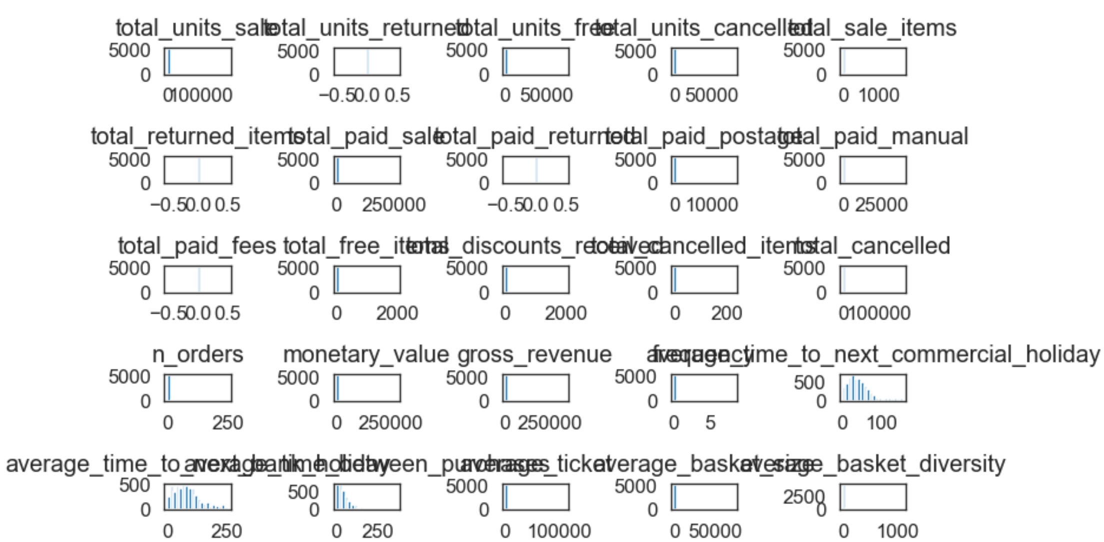
```
)


# descriptive statistics of the dataset:
df_stats = df[num_cols].describe()

# descriptive statistics of the dataset:
df_stats.loc["variance"] = df[num_cols].var().tolist()
df_stats.loc["skewness"] = df[num_cols].skew().tolist()
df_stats.loc["kurtosis"] = df[num_cols].kurtosis().tolist()

# calculating the descriptive statistics:
df_stats.T
```

[105]:

```
                                         count    mean     std  min    25%  \
50%     75%       max    variance  skewness  kurtosis
total_units_sale                       5373.00  402.72 4126.90 0.00   0.00
16.00   84.00 184907.00 17031324.80     29.33   1102.92
total_units_returned                   5373.00    0.00    0.00 0.00   0.00
0.00    0.00      0.00        0.00     0.00      0.00
total_units_free                       5373.00  397.08 1800.22 0.00  16.00
100.00  319.00  72085.00  3240793.13     21.06    625.34
total_units_cancelled                  5373.00   51.54 1518.91 0.00   0.00
0.00    2.00  80995.00  2307098.36    49.98   2559.81
total_sale_items                       5373.00   10.27   42.26 0.00   0.00
2.00    8.00   1687.00     1785.84    20.93    648.32
total_returned_items                   5373.00    0.00    0.00 0.00   0.00
0.00    0.00      0.00        0.00     0.00      0.00
total_paid_sale                        5373.00  742.51 6658.37 0.00   0.00
41.97  305.75 260634.30 44333860.42     27.13    921.56
total_paid_returned                    5373.00    0.00    0.00 0.00   0.00
0.00    0.00      0.00        0.00     0.00      0.00
total_paid_postage                     5373.00   53.50  325.44 0.00   0.00
0.00    0.00  16285.50   105913.46    33.40   1488.54
total_paid_manual                      5373.00   31.90  707.97 0.00   0.00
0.00    0.00  38970.00   501216.74    41.78   2025.07
total_paid_fees                        5373.00    0.00    0.00 0.00   0.00
0.00    0.00      0.00        0.00     0.00      0.00
total_free_items                       5373.00   20.68   52.85 0.00   2.00
8.00   23.00   1962.00     2792.68    17.64    507.24
total_discounts_received               5373.00    1.06   31.32 0.00   0.00
0.00    0.00   1987.86      981.16    51.38   3081.78
total_cancelled_items                  5373.00    1.68    6.71 0.00   0.00
0.00    1.00    226.00       44.97    14.11    327.49
total_cancelled                        5373.00  114.44 2663.89 0.00   0.00
0.00    9.10 168469.60  7096285.65    52.55   3113.43
n_orders                               5373.00    4.32    8.57 1.00   1.00
```

```
2.00    5.00    248.00       73.48      12.25    257.22
monetary_value                           5373.00 1965.62 8185.59 0.00 296.75
694.05 1740.00 280206.02 67003883.64     20.79    563.12
gross_revenue                            5373.00 2080.05 9565.67 0.00 303.16
709.71 1762.22 336942.10 91502015.56     21.63    593.33
frequency                                5373.00    0.14    0.29 0.03   0.03
0.07    0.17      8.27         0.08     12.25    257.10
average_time_to_next_commercial_holiday 5373.00   46.58   32.34 1.00  23.00
40.00   62.00    159.00     1045.58      1.25      1.65
average_time_to_next_bank_holiday        5373.00   89.10   57.81 1.00  45.00
81.00  119.00    253.00     3341.89      0.79      0.09
average_time_between_purchases           3059.00   60.37   61.84 0.00  21.00
41.00   76.00    366.00     3824.02      2.22      5.83
average_ticket                           5373.00  620.09 2365.28 0.00 163.19
273.42  480.00 112314.03  5594540.41     30.43   1214.41
average_basket_size                      5373.00  266.65 1331.30 1.00  78.00
144.50  270.50  74215.00  1772357.06     44.84   2271.61
average_basket_diversity                 5373.00   39.51   78.56 1.00   9.00
16.50   33.00   1106.00     6171.64      4.95     31.21
```

[106]:
```python
# plotting histograms to visualize the distributions:
df[num_cols].hist(bins=20, grid=False)
plt.tight_layout()
```



4.1 Key observations about the univariate analysis

1. There is a slight growing trend in number of orders towards the end of the year, peaking at around week 48, which is the most representative week in both 2016 and 2017. In both cases, the peaks occur in the week of Black Friday.

2. Most of the features related to sales are concentrated around zero. This represents a common behavior in E-commerce and other transactional businesses: there majority of clients are not very active;
3. The `recency` feature shows that many customers bought something at the website recently. This can represent an influx or increase of customers towards the end of the year;
4. The features related to time between features and days to the next commercial holiday or bank holiday are mostyly concentrated below 50 days. This behavior can be explored as a criteria for a fidelity program;

With these key remarks in mind, I consider to fix the entries affecting the results and properly identifying the outliers and other possible data issues.

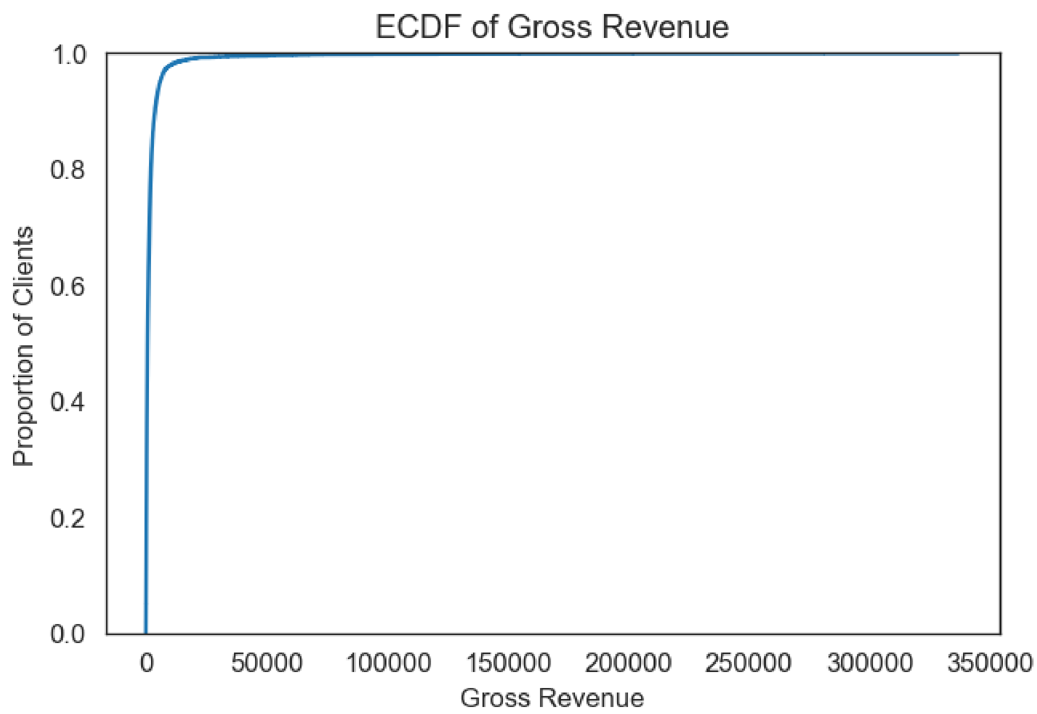
5 3. Identifying Resellers in the Customer Base

By analyzing some of the variables related to money spent on the website (`average_basket_size`, `n_orders`, `monetary_value`), we can spot several cases of outliers in the dataset. These are observed in very large purchases (I don't really believe a normal client without access to a warehouse would be able to have an average basket size of 74k items).

I will use these artifacts to separate regular customers from customers that are resellers, since it is a different category of client that should have special treatment.

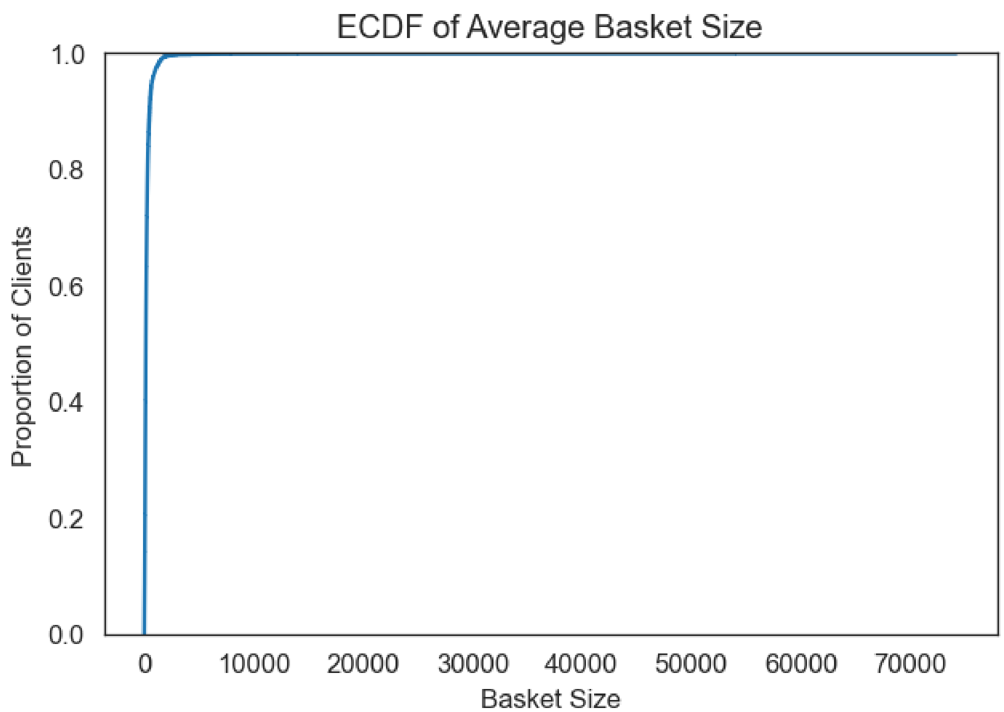
[107]:
```python
# visualizing the ECDF of the key variables:
fig, ax = plt.subplots(figsize=(6, 4))

ax = sns.ecdfplot(data=df, x="gross_revenue")

ax.set(
    title="ECDF of Gross Revenue",
    xlabel="Gross Revenue",
    ylabel="Proportion of Clients",
)
```

[107]:
```
[Text(0.5, 1.0, 'ECDF of Gross Revenue'),
 Text(0.5, 0, 'Gross Revenue'),
 Text(0, 0.5, 'Proportion of Clients')]
```



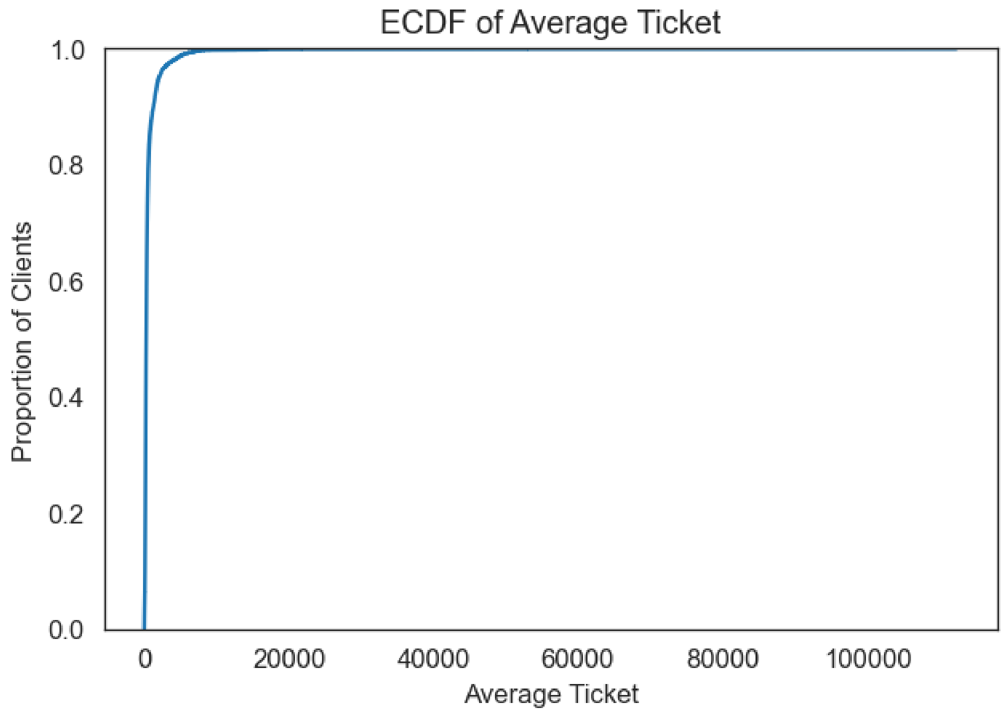
[108]:
```python
# basket size
fig, ax = plt.subplots(figsize=(6, 4))

ax = sns.ecdfplot(data=df, x="average_basket_size")

ax.set(
    title="ECDF of Average Basket Size",
    xlabel="Basket Size",
    ylabel="Proportion of Clients",
)
```

[108]:
```
[Text(0.5, 1.0, 'ECDF of Average Basket Size'),
 Text(0.5, 0, 'Basket Size'),
 Text(0, 0.5, 'Proportion of Clients')]
```



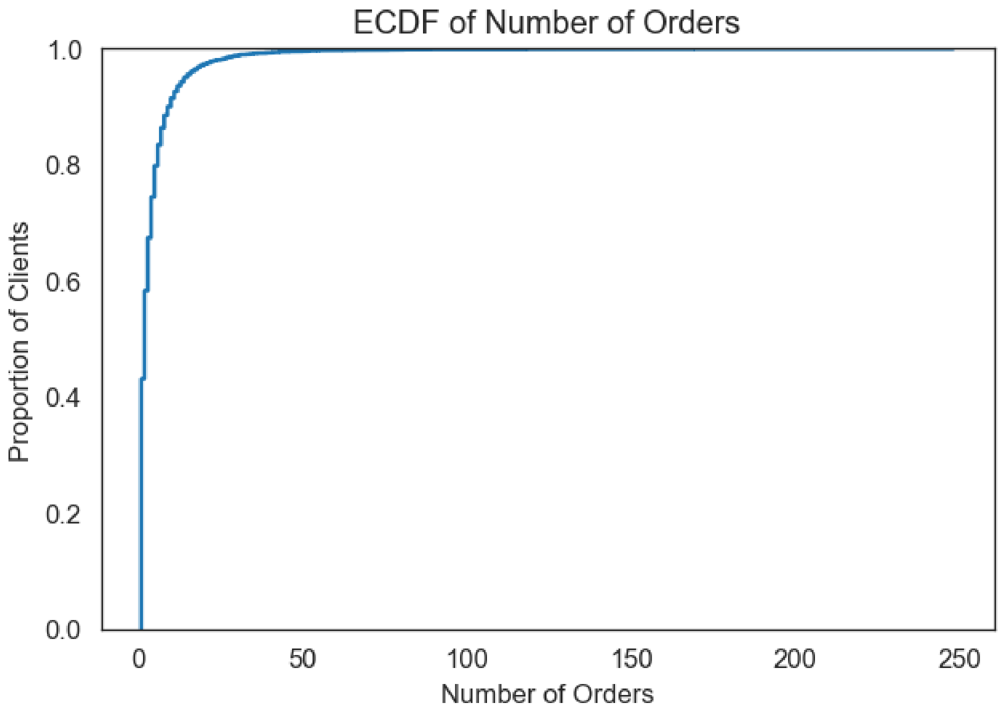
[109]:
```
# average ticket
fig, ax = plt.subplots(figsize=(6, 4))

ax = sns.ecdfplot(data=df, x="average_ticket")

ax.set(
    title="ECDF of Average Ticket",
    xlabel="Average Ticket",
    ylabel="Proportion of Clients",
)
```

[109]:
```
[Text(0.5, 1.0, 'ECDF of Average Ticket'),
 Text(0.5, 0, 'Average Ticket'),
 Text(0, 0.5, 'Proportion of Clients')]
```



[110]:
```
# number of orders:
fig, ax = plt.subplots(figsize=(6, 4))

ax = sns.ecdfplot(data=df, x="n_orders")

ax.set(
    title="ECDF of Number of Orders",
    xlabel="Number of Orders",
    ylabel="Proportion of Clients",
)
```

[110]:
```
[Text(0.5, 1.0, 'ECDF of Number of Orders'),
 Text(0.5, 0, 'Number of Orders'),
 Text(0, 0.5, 'Proportion of Clients')]
```



The ECDFs above suggest that we have a very high concentration of clients in the left-side of the distribution. That can indicate that we have very few (less than 1%) of clients in the higher values of the distribution. I will use this behavior to assume that these clients that place very highly in the ECDF quantiles to be Wholesalers. In the dynamics of an E-commerce, regular customers do not buy at such high-values or basket sizes.

[111]:

```
# finding key quantiles for the variables in the dataset:
cols_to_validate = [
    "gross_revenue",
    "average_basket_size",
    "average_ticket",
    "n_orders",
]

for col in cols_to_validate:
    quantiles = [0.8, 0.9, 0.95, 0.98, 0.99, 1.0]
    print(
        f"""
{col}:
{df[col].quantile(quantiles)}
"""
    )
```

```
gross_revenue:
```

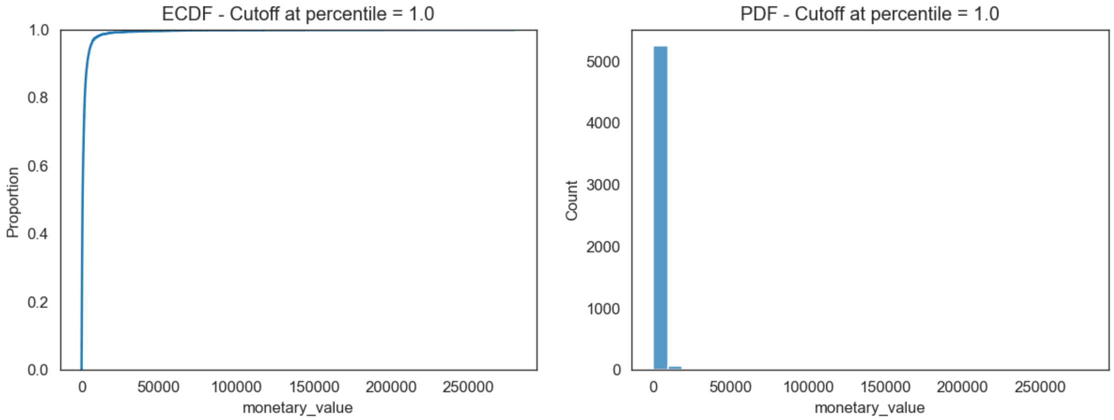
```
0.80      2127.76
0.90      3745.07
0.95      5800.89
0.98     10687.57
0.99     18718.85
1.00    336942.10
Name: gross_revenue, dtype: float64


average_basket_size:
0.80      313.65
0.90      487.48
0.95      706.45
0.98     1247.04
0.99     1608.56
1.00    74215.00
Name: average_basket_size, dtype: float64


average_ticket:
0.80       575.22
0.90      1273.76
0.95      2034.66
0.98      4108.45
0.99      5379.42
1.00    112314.03
Name: average_ticket, dtype: float64


n_orders:
0.80      6.00
0.90      9.00
0.95     14.00
0.98     24.00
0.99     33.00
1.00    248.00
Name: n_orders, dtype: float64
```

The variable that has the highest change per quantile is the `gross_revenue` column. I will explore this behavior to verify is we can get a more reasonable empirical distribution (that is more evenly distributed accross its range).

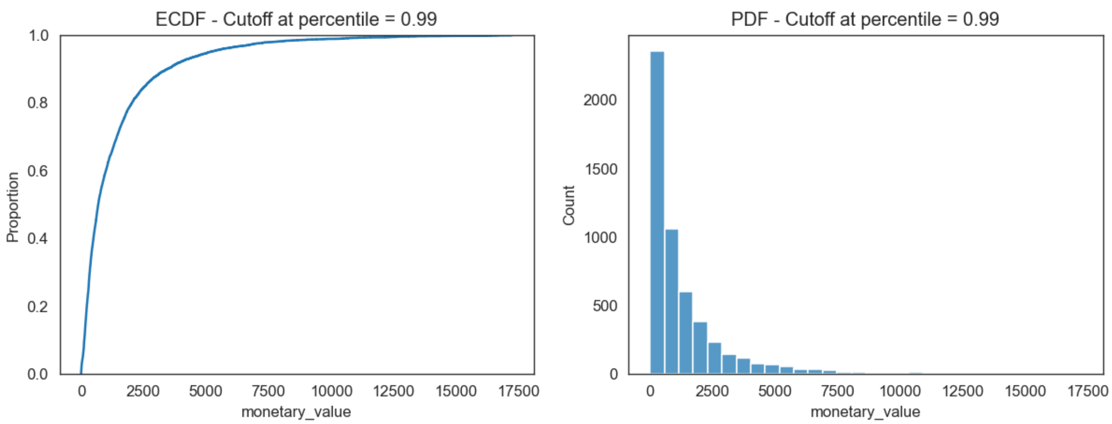
[112]:
```
# defining cutoffs
cutoffs = (df["monetary_value"].quantile(quantiles).values)[::-1]

# fig, axs = plt.subplots(len(cutoffs), 1, figsize=(6, 20))
```

```python
for idx, cutoff in enumerate(cutoffs):

    df_temp = df[df.monetary_value <= cutoff].copy()

    # plot the distribution:
    fig, axs = plt.subplots(1, 2, figsize=(12, 4))

    ax1 = sns.ecdfplot(data=df_temp, x="monetary_value", ax=axs[0])

    ax2 = sns.histplot(data=df_temp, x="monetary_value", ax=axs[1], bins=30)

    pop_size = df_temp.shape[0]

    print(
        f"""
Cutoff: {cutoff}
Population size: {pop_size}
Skewness: {df_temp.monetary_value.skew()}
Kurtosis: {df_temp.monetary_value.kurtosis()}

    """
    )

    ax1.set(title=f"ECDF - Cutoff at percentile = {quantiles[::-1][idx]}")

    ax2.set(title=f"PDF - Cutoff at percentile = {quantiles[::-1][idx]}")

    plt.show()
```

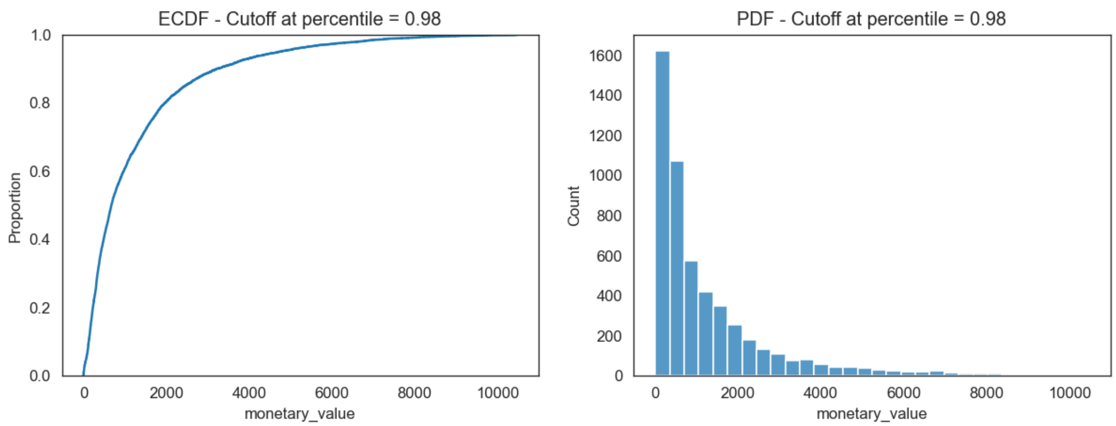
```
Cutoff: 280206.02
Population size: 5373
Skewness: 20.785373204810032
Kurtosis: 563.1249555241652
```



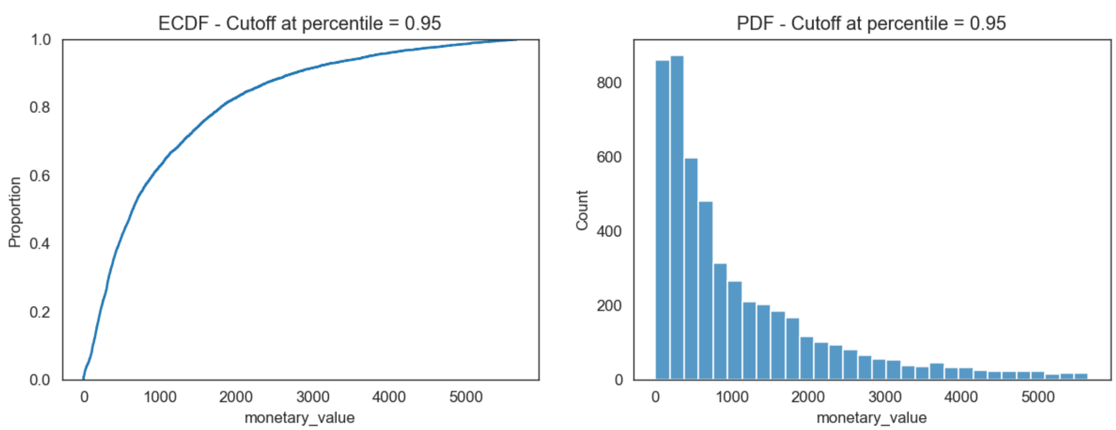

```
Cutoff: 17265.25279999999
Population size: 5319
Skewness: 3.4032421306923006
Kurtosis: 15.78662404657699
```



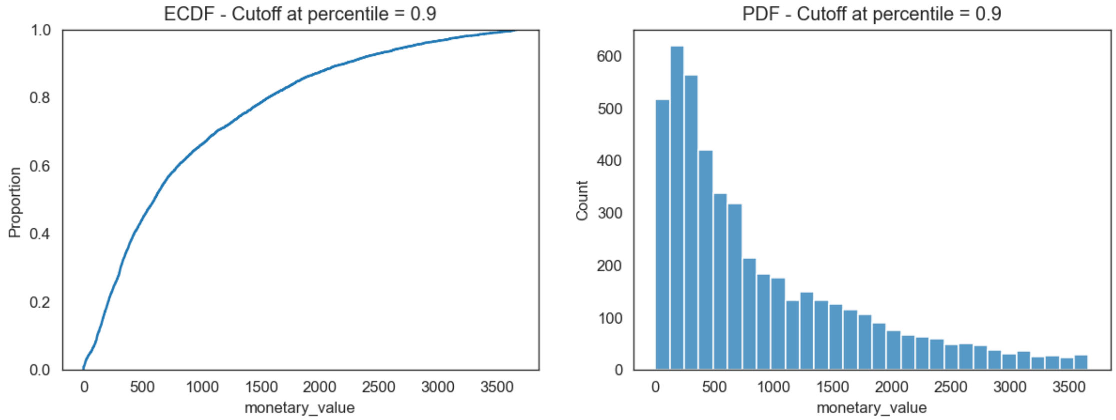

```
Cutoff: 10476.12839999999
Population size: 5265
Skewness: 2.3706088294121668
Kurtosis: 6.538826928478486
```



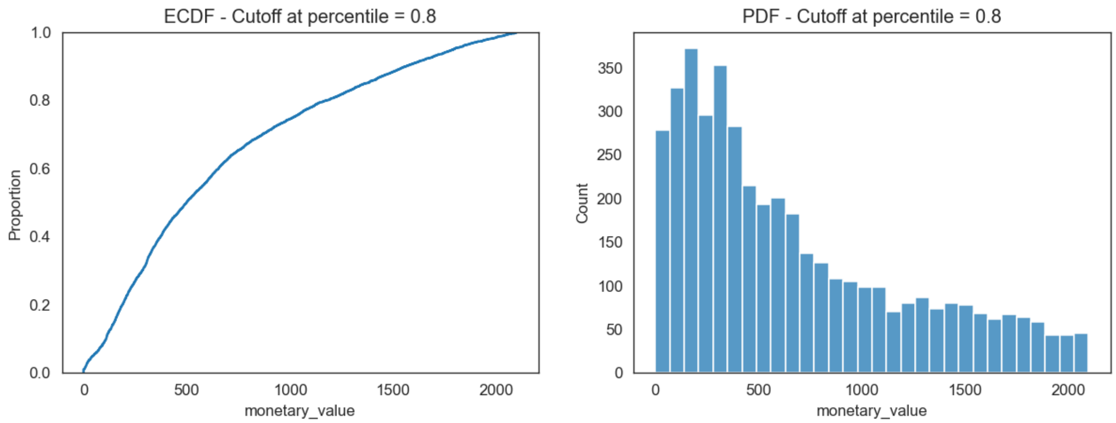

```
Cutoff: 5671.621999999994
Population size: 5104
Skewness: 1.6877515762515676
Kurtosis: 2.560144332466762
```




```
Cutoff: 3672.0160000000005
Population size: 4835
Skewness: 1.2527474456202847
Kurtosis: 0.8457816518280303
```




```
Cutoff: 2099.6
Population size: 4298
Skewness: 0.8788698626804539
Kurtosis: -0.28347484167860815
```



The most reasonable cutoff for `monetary_value` seems to be the `95th percentile`, as lower cutoffs remove much more people. I will this cutoff as the indicator for the customer being a `reseller`.

[113]:
```
# adding a boolean handle for the cutoff value:
reseller_cutoff = cutoffs[3]
print(
    reseller_cutoff
)  # this will be used retroactively in the data preparation notebooks
```

```
5671.621999999994
```

6 4. Bivariate Analysis

Given that the results for identifying the Outliers are meaningful, I will explore some key relationships in the dataset to observe relevant bivariate trends and rule out or confirm some hypothesis I have previously thought of while focusing on identifying potential rules and characteristics for establishing a fidelity program.

[114]:
```
# separating both groups for analysis:
regular_customers = df[~df.is_considered_reseller].copy()
resellers = df[df.is_considered_reseller].copy()
```

[115]:
```
# listing key featurs for bivariate analysis:
key_features = [
    "account_age_days",
    "n_orders",
    "recency",
    "gross_revenue",
    "total_cancelled",
    "average_ticket",
    "average_time_between_purchases",
    "average_time_to_next_bank_holiday",
    "average_time_to_next_commercial_holiday",
    "average_basket_size",
    "average_basket_diversity",
]

feature_descriptions = [
    "Customer Account Age",
    "Number of Orders",
    "Recency",
    "Gross Revenue",
    "Total Amount Cancelled",
    "Average Ticket",
    "Average Time Betweeen Purchases",
    "Average Time to Next Bank Holiday",
    "Average Time to Next Commercial Holiday",
    "Average Basket Size",
    "Average Basket Diversity",
]

display_cols = dict(zip(key_features, feature_descriptions))
```

6.1 4.1 Do customers who buy more frequently (less time between purchases) generate more revenue?

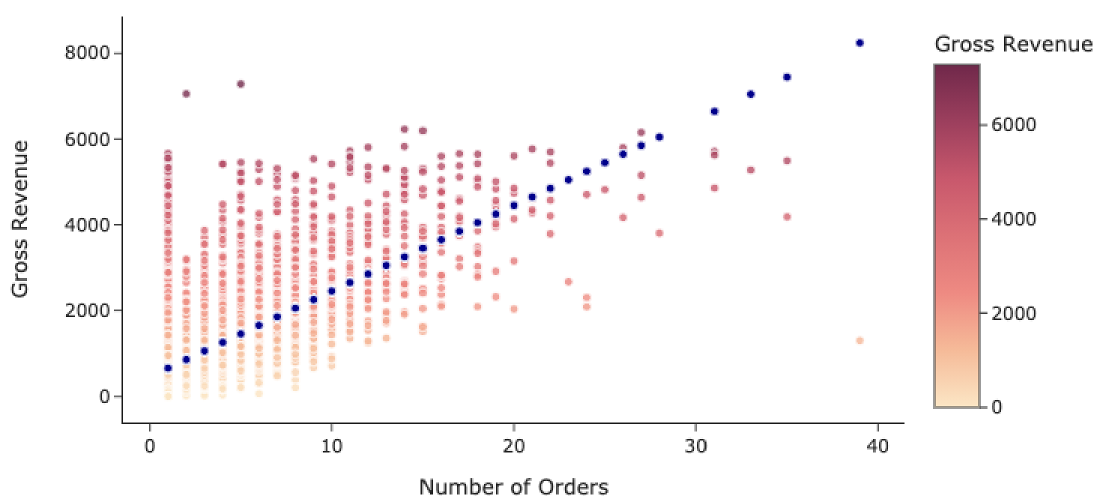
[116]:
```
fig = px.scatter(
    regular_customers,
    title="Gross Revenue x Average time between purchases",
    x="average_time_between_purchases",
    y="gross_revenue",
    color="gross_revenue",
    template="simple_white",
    color_continuous_scale="burgyl",
    labels=display_cols,
    render_mode="svg",
    opacity=0.8,
)

fig.update_layout(
    legend=dict(orientation="v", yanchor="bottom", xanchor="right", y=1, x=1)
)

fig.update_traces(mode="markers", marker_line_width=0.6,␣
↪marker_line_color="white")


fig.show()
```

6.1.1 4.1 True

By analyzing the scatter plot, we can spot a tendency of clients with lower average time betweeen orders to have higher Gross Revenue ratings. We can further observe these by verifying the trend in the `frequency` x `revenue` relationship.

[117]:
```
fig = px.scatter(
    regular_customers,
    title="Gross Revenue x Number of Orders",
    x="n_orders",
    y="gross_revenue",
    color="gross_revenue",
    template="simple_white",
    color_continuous_scale="burgyl",
    labels=display_cols,
    render_mode="svg",
    opacity=0.8,
    trendline="ols",
    trendline_color_override="darkblue",
)

regression_results = px.get_trendline_results(fig).px_fit_results.iloc[0].
 ↪summary()

fig.update_layout(
    legend=dict(orientation="v", yanchor="bottom", xanchor="right", y=1, x=1)
)

fig.update_traces(mode="markers", marker_line_width=1,␣
 ↪marker_line_color="white")

fig.show()
```

Gross Revenue x Number of Orders



[118]:
```
# plotting the regression lines:
regression_results
```

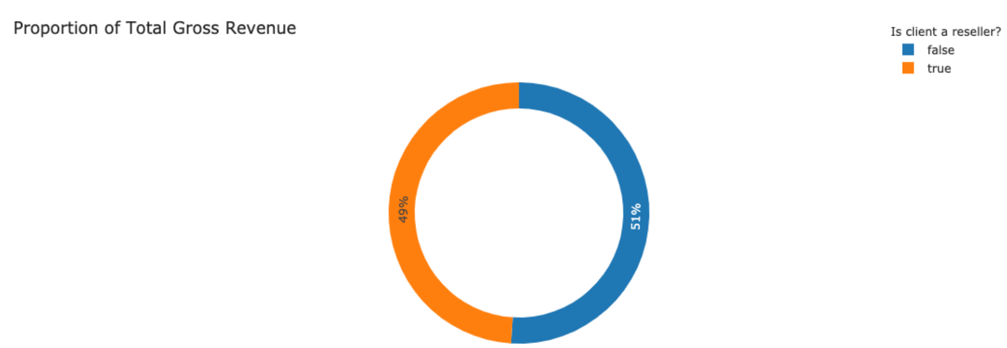
[118]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.390
Model:                            OLS   Adj. R-squared:                  0.390
Method:                 Least Squares   F-statistic:                     3262.
Date:                Tue, 10 Aug 2021   Prob (F-statistic):               0.00
Time:                        18:07:56   Log-Likelihood:                -42127.
No. Observations:                5104   AIC:                         8.426e+04
Df Residuals:                    5102   BIC:                         8.427e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        453.5076     17.435     26.011      0.000     419.327     487.688
x1           199.7861      3.498     57.117      0.000     192.929     206.643
==============================================================================
Omnibus:                     2235.017   Durbin-Watson:                   1.845
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            12768.591
Skew:                           2.037   Prob(JB):                         0.00
```

```
Kurtosis:                       9.592   Cond. No.                         6.80
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

As the plot illustrates, the estimated regression line suggests that the more orders a customer makes, the higher the gross revenue (positive slope).

6.2 4.2 Do resellers generate more revenue ?

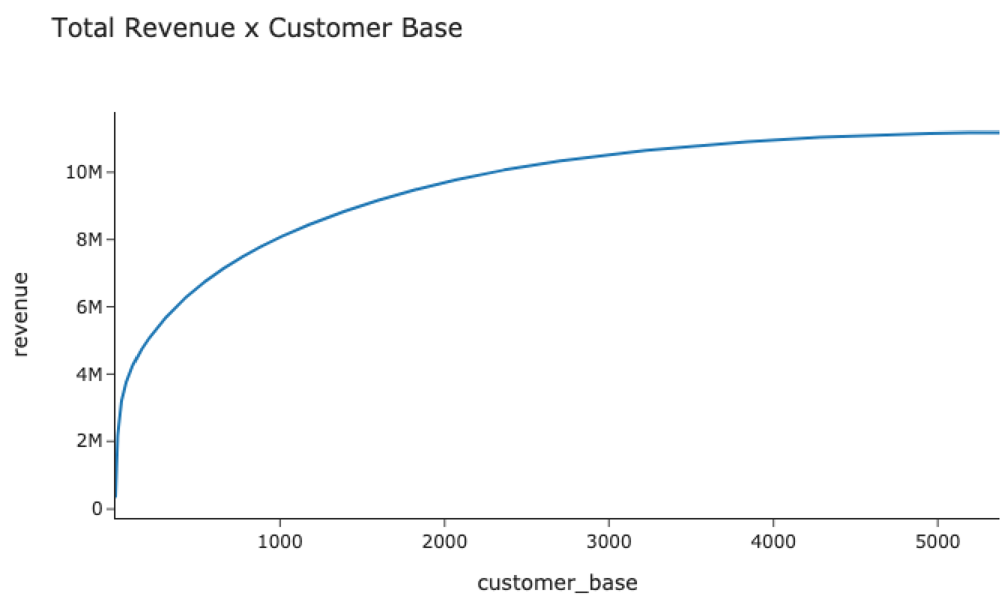
[119]:
```python
fig = px.pie(
    df,
    title="Proportion of Total Gross Revenue",
    values="gross_revenue",
    names="is_considered_reseller",
    template="simple_white",
    hole=0.8,
    labels=display_cols,
)

fig.update_layout(
    legend=dict(
        title="Is client a reseller?",
        orientation="v",
        yanchor="bottom",
        xanchor="right",
        y=1,
        x=1,
    )
)

fig.show()
```



6.2.1 4.2 False

From the chat above, we can observe that we have about 49% of the gross revenue coming from resellers. This means that the revenue made from the website is roughly split in half between these types of customers. The density of revenue concentraded in certain clients, however, is much different, as I will illustrate in the following plot.

[120]:
```python
# adding a boolean handle for better visualization:
df["customer_type"] = df["is_considered_reseller"].map(
    {True: "Reseller", False: "Regular"}
)

revenue_growth = df.gross_revenue.sort_values(ascending=False).cumsum()

size_growth = np.arange(1, df.shape[0] + 1, 1)

data = {"revenue": revenue_growth.values, "customer_base": size_growth}

# concatenating the series for plotting:
df_growth = pd.DataFrame(data=data)

fig = px.scatter(
    df_growth,
    title="Total Revenue x Customer Base",
    x="customer_base",
    y="revenue",
    template="simple_white",
    opacity=0.8,
    labels=display_cols,
    render_mode="svg",
)


fig.update_traces(mode="markers", marker_line_width=0.1,␣
 ↪marker_line_color="black")

fig.update_traces(mode="lines", marker_line_width=2, marker_line_color="blue")


fig.show()
```



The plot above shows that a very large portion of the total revenue from the website during the time period we are observing (Nov. 2016 to Dec. 2017) is concentrated in very few clients. In fact the very first 286 clients if we sort by total revenue contributed in the period represents 50% o the total revenue during the period.

6.3 4.3 Do Resellers buy more in advance towards holidays ?

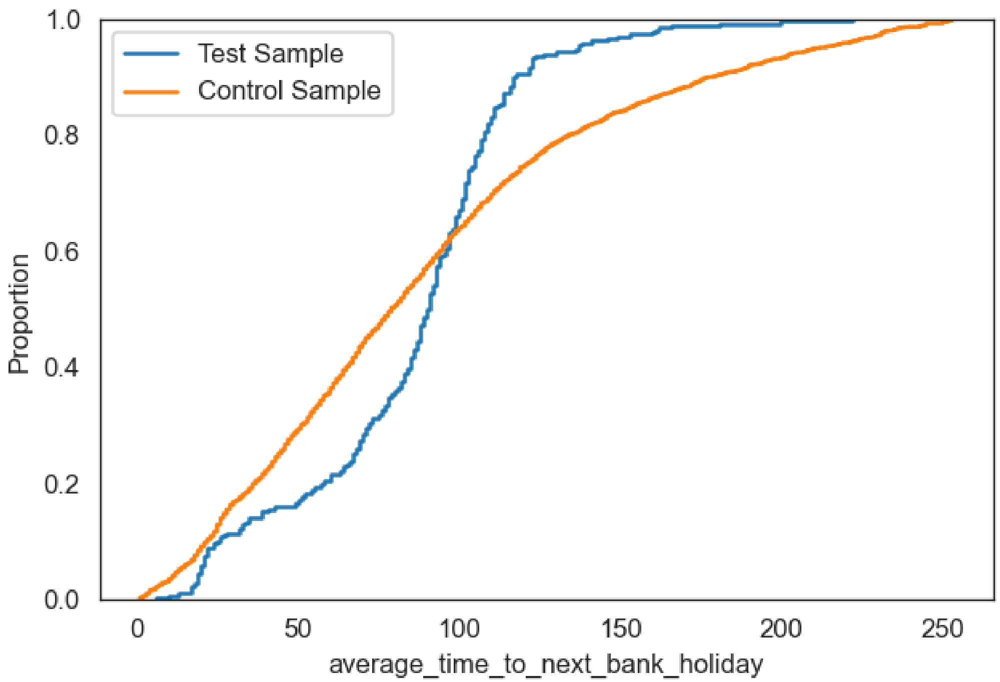
[121]:
```
fig = px.histogram(
    df,
    x="average_time_to_next_bank_holiday",
    template="simple_white",
    color="customer_type",
    marginal="box",
    labels=display_cols,
    title="Distribution of Average Time To Next Holiday - Resellers x Regular␣
 ↪Customers",
    hover_data=df.columns,
)

fig.update_layout(yaxis_title="Number of Clients")

fig.show()
```

From the Box plot above, it is hard to verify visually whether there is a significant or noticeable difference between the types of customers regarding the time to next holiday where their orders take place.

To verify if the little difference observed is in, in fact, relevant, we will run a non-parametric statistical test. More specifically, I will use the `KS` (Kolmogorov-Smirnov) test, which compares the distances between the ECDF of the test sample to the ECDF of the control sample.

[122]:
```
# running the test using the helper function previously defined:
test_hypothesis(
    test_sample=resellers.average_time_to_next_bank_holiday,
    control_sample=regular_customers.average_time_to_next_bank_holiday,
    alternative="greater",  # will test whether the test sample is larger than␣
↪the control sample
    test_type="ks",
    ecdf=True,
)
```

```
Test statistic = 0.17368111314400253
P-value = 1.7863439055825726e-07
H1 True: Test Sample GREATER than Control
```



6.3.1 4.3 True

The test verified that there is indeed a significant statistical relationship when comparing the reseller and regular customer groups in respect to the avergage time before a bank holiday in which purchases are made, with the resellers, on average, buying from website earlier.

6.4 4.4 Resellers who buy less often buy in larger quantities ?

[123]:
```
fig = px.scatter(
    resellers,
    title="Total Items Bought x Number of Orders",
    x="n_orders",
    y="total_items",
    color="monetary_value",
    template="simple_white",
    color_continuous_scale="burgyl",
    labels=display_cols,
)


fig.update_layout(
    legend=dict(orientation="v", yanchor="bottom", xanchor="right", y=1, x=1)
)
```

```
fig.update_traces(mode="markers", marker_line_width=1,␣
 ↪marker_line_color="white")

fig.show()
```

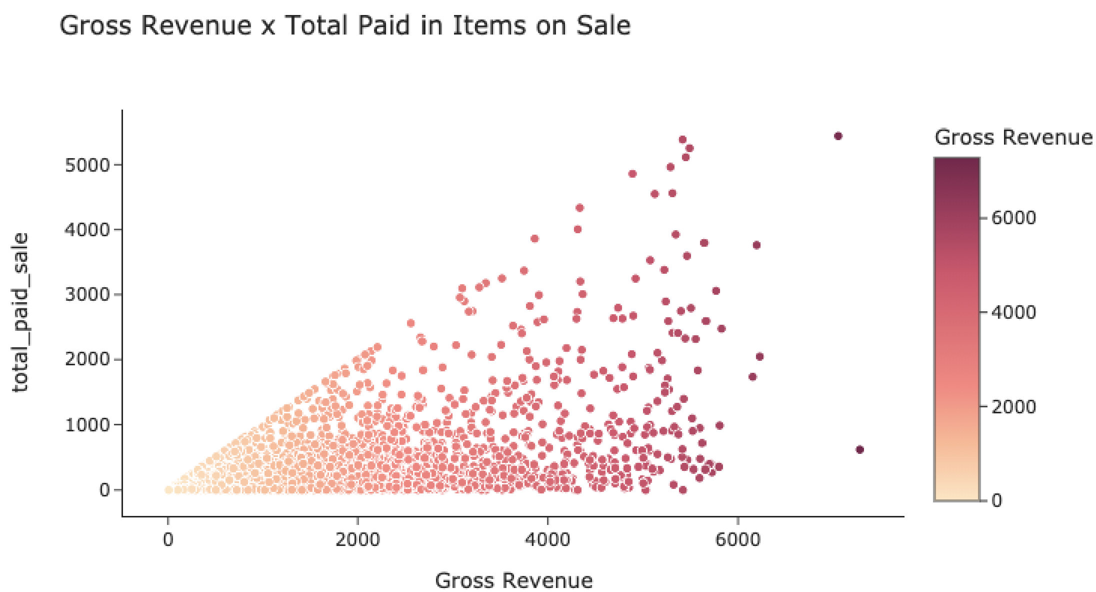
6.4.1 4.4 False

The results from the plot above suggest that there is somewhat of a tendency for clients that have a lower number of orders to buy more items at once. However, this tendency does not seem to be relevant, as the majority of the resellers are concentrated on the lower end of number of orders (up to 50 orders).

6.5 4.5 Customers who buy more items on sale generate more revenue?

[124]:
```
fig = px.scatter(
    regular_customers,
    title="Gross Revenue x Total Paid in Items on Sale",
    x="gross_revenue",
    y="total_paid_sale",
    color="gross_revenue",
    template="simple_white",
    color_continuous_scale="burgyl",
    labels=display_cols,
)
```

```
fig.update_layout(
    legend=dict(orientation="v", yanchor="bottom", xanchor="right", y=1, x=1)
)

fig.update_traces(mode="markers", marker_line_width=1,␣
 ↪marker_line_color="white")

fig.show()
```



6.5.1 4.5 True

The triangular shape of the scatterplot above suggest that there are quite a lot of customers that buy all of their items (and thus have the same gross revenue) as items on sale. This illustrates a very important information: items on sale are very relevant for customers and they seem to drive revenue.

7 5. Conclusions

With our analysis, we can make a few key conclusions about our main question (how to create a data-driven customer fidelity program):

There are many aspects of how a customer can be valuable. Some customers are one-time buyers and buy in bulk. Others buy very frequently, which indicates loyalty to the brand and/or website. Any model we develop to segment such customers needs to take such nuances into consideration.

We also need to consider customers of different kinds separately. It does not make much sense not to include resellers, for example, in a loyalty program, but if we consider them into a model, these will always be outliers, as they represent a different business segment altogether.