iFood - Data Analyst Test

Analysis Report

Alvaro Manzanas

23 julio, 2024

Contents

Objectives

Build an analysis to address the greatest benefit for the next direct marketing campaign (sixth) that aims to sell a new direct marketing campaign (sixth) that aims to sell a new gadget gadget. Pilot campaign:

- 2240 customers
- Sample campaign cost: 6,720MU (million euros)
- Revenue generated: 3,674MU
- Overall profit: -3,046MU
- Success rate 15%.

Develop a model that predicts customer behaviour and apply it to the rest of the customer base to the rest of the customer base.

- Select customers most likely to purchase the offer.
- Exclude non-responders.
- Characteristic traits of customers willing to buy the gadget.

1 Data Cleaning

First of all, the cleaning of data that may interfere with the analysis is started by checking the columns of the dataset and selecting those whose type is 'object' in order to see their unique values.

```python
# Exploring unique values in some columns

print("\n'Education' Column Values:\n", "\t",
      data["Education"].unique(),
      sep=''
      )
```

```
'Education' Column Values:
    ['Graduation' 'PhD' 'Master' 'Basic' '2n Cycle']
```

```python
print("\n'Marital_Status' Column Values:\n", "\t",
      data["Marital_Status"].unique(),
      sep=''
      )
```

```
'Marital_Status' Column Values:
    ['Single' 'Together' 'Married' 'Divorced' 'Widow' 'Alone' 'Absurd' 'YOLO']
```

```python
print("\n'Dt_Customer' Column sample:\n",
      data["Dt_Customer"].sample(3), "\n", "\t",
      sep=''
      )
```

```
'Dt_Customer' Column sample:
1938    2014-03-23
2173    2013-07-29
1248    2014-03-24
Name: Dt_Customer, dtype: object
```

Once we know what these columns are made up of, the type of column is changed to the appropriate format, this being 'category' or 'datetime'. In addition, new columns are added to find out the age of the customers, how many days they have been customers, whether they have made purchases in the last month, the total amount they have spent, the total number of purchases, the total number of campaigns accepted, and the total number of children they have at home.

Once these new columns have been found to be of interest for the following analyses, they are reordered to make them easier to read.

```python
data["Income"] = data["Income"].astype("Int64")
data["Education"] = data["Education"].astype("category")
data["Marital_Status"] = data["Marital_Status"].astype("category")
data["Dt_Customer"] = pd.to_datetime(data["Dt_Customer"], format="%Y-%m-%d")
```

```
today = pd.to_datetime(datetime.today().strftime('%Y-%m-%d'))
data["Year_Old"] = (today.year - data["Year_Birth"])
data["CustomerFor"] = (today - data["Dt_Customer"])

# Reordering columns 1

pop_column = data.pop("Dt_Customer")
data.insert(2, "Dt_Customer", pop_column)

last_columns = data.columns[-2:]
first_columns = data.columns[:2]
middle_columns = data.columns[2:-2]
new_order = list(first_columns) + list(last_columns) + list(middle_columns)

data = data[new_order]

# Checking if the customer bought in the last month
data["PurchaseLastMonth"] = (data["Recency"] < 30)
data["PurchaseLastMonth"] = data["PurchaseLastMonth"].replace({True:1,
                                                               False:0})
# Calculating total amount spent per customer
MntSpentTotal_sum = ["MntFishProducts", "MntFruits", "MntGoldProds",
                     "MntMeatProducts", "MntSweetProducts", "MntWines"]
data["MntSpentTotal"] = data[MntSpentTotal_sum].sum(axis=1)

# How many campaigns the customer accepted
AcceptedCmpTotal_sum = ["AcceptedCmp1", "AcceptedCmp2", "AcceptedCmp3",
                        "AcceptedCmp4", "AcceptedCmp5"]
data["AcceptedCmpTotal"] = data[AcceptedCmpTotal_sum].sum(axis=1)

# How many children (Kids and teenagers) the customer has at home

ChildrenHome_sum = ["Kidhome", "Teenhome"]
data["ChildrenHome"] = data[ChildrenHome_sum].sum(axis="columns")

NumPurchasesTotal_sum = ["NumWebPurchases",
                         "NumCatalogPurchases",
                         "NumStorePurchases"]

data["NumPurchasesTotal"] = data[NumPurchasesTotal_sum].sum(axis="columns")

# Reordering columns

pop_column = data.pop("AcceptedCmpTotal")
data.insert(27, "AcceptedCmpTotal", pop_column)

pop_column = data.pop("PurchaseLastMonth")
data.insert(17, "PurchaseLastMonth", pop_column)

pop_column = data.pop("MntSpentTotal")
data.insert(11, "MntSpentTotal", pop_column)

pop_column = data.pop("ChildrenHome")
```

```python
data.insert(10, "ChildrenHome", pop_column)

pop_column = data.pop("AcceptedCmp2")
data.insert(25, "AcceptedCmp2", pop_column)

pop_column = data.pop("AcceptedCmp1")
data.insert(25, "AcceptedCmp1", pop_column)

pop_column = data.pop("Response")
data.insert(30, "Response", pop_column)

pop_column = data.pop("NumPurchasesTotal")
data.insert(20, "NumPurchasesTotal", pop_column)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 36 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Year_Birth           2240 non-null   int64
 2   Year_Old             2240 non-null   int64
 3   CustomerFor          2240 non-null   timedelta64[ns]
 4   Dt_Customer          2240 non-null   datetime64[ns]
 5   Education            2240 non-null   category
 6   Marital_Status       2240 non-null   category
 7   Income               2216 non-null   Int64
 8   Kidhome              2240 non-null   int64
 9   Teenhome             2240 non-null   int64
 10  ChildrenHome         2240 non-null   int64
 11  Recency              2240 non-null   int64
 12  MntSpentTotal        2240 non-null   int64
 13  MntWines             2240 non-null   int64
 14  MntFruits            2240 non-null   int64
 15  MntMeatProducts      2240 non-null   int64
 16  MntFishProducts      2240 non-null   int64
 17  MntSweetProducts     2240 non-null   int64
 18  MntGoldProds         2240 non-null   int64
 19  PurchaseLastMonth    2240 non-null   int64
 20  NumPurchasesTotal    2240 non-null   int64
 21  NumDealsPurchases    2240 non-null   int64
 22  NumWebPurchases      2240 non-null   int64
 23  NumCatalogPurchases  2240 non-null   int64
 24  NumStorePurchases    2240 non-null   int64
 25  NumWebVisitsMonth    2240 non-null   int64
 26  AcceptedCmp1         2240 non-null   int64
 27  AcceptedCmp2         2240 non-null   int64
 28  AcceptedCmp3         2240 non-null   int64
 29  AcceptedCmp4         2240 non-null   int64
 30  AcceptedCmp5         2240 non-null   int64
 31  Response             2240 non-null   int64
```

```
 32  AcceptedCmpTotal     2240 non-null   int64
 33  Complain             2240 non-null   int64
 34  Z_CostContact        2240 non-null   int64
 35  Z_Revenue            2240 non-null   int64
dtypes: Int64(1), category(2), datetime64[ns](1), int64(31), timedelta64[ns](1)
memory usage: 602.3 KB
```

1.1 Discarding client entries

Customers who are deemed not to meet the inclusion criteria for further analysis will then be analysed and discarded. The exclusion criteria will be: not having all fields completed and those clients who are considered to be logically inconsistent in their answers.

First, missing values will be detected and the corresponding rows will be discarded. Then those clients who have answered in Marital Status (which we checked earlier) with 'Alone, Absurd, or Yolo' will be considered logically inconsistent.

```
# Missing Values
row_nan = data[data.isna().any(axis=1)]
data.drop(row_nan.index, inplace=True)

# Identifying logically incoherent customers and dropping from the dataframe
marital_filt = data[data["Marital_Status"].isin(['Alone', 'Absurd', 'YOLO'])]
data.drop(marital_filt.index, inplace=True)
```

Next, outliers in the previously created variable of age and in 'Income' will be identified. To do this, before starting the normality of both variables will be checked with the Kolmogorov-Smirnov test, whose Null Hypothesis is that the sample follows the normal distribution. It can be observed that in both cases with our data H0 does not hold and it will be considered that they do not follow the normal curve.

```
# KS-Test on 'Year_Old' column
ks_result = kstest(data["Year_Old"], stats.norm.cdf,
                   args=(data["Year_Old"].mean(), data["Year_Old"].std()))

print(f"Test statistic: {ks_result.statistic:.4f}"
      f"\np Value: {ks_result.pvalue:.4f}")
```

```
Test statistic: 0.0590
p Value: 0.0000
```

```
# KS-Test on 'Year_Old' column
ks_result = kstest(data["Income"], stats.norm.cdf,
                   args=(data["Income"].mean(), data["Income"].std()))

print(f"\nTest statistic: {ks_result.statistic:.4f}"
      f"\np Value: {ks_result.pvalue:.4f}")
```

```

Test statistic: 0.0542
p Value: 0.0000
```

Continue to obtain the outliers, which will be those extreme values according to the interquartile range. For the variable 'Year_Old' three outliers will be obtained, ages exceeding 120 years. For 'Income' 8 outliers are found, but only one of them is considered logically incoherent. Three clients will be discarded for age and one for 'Income'.

```
# Quartiles and IQR
quartiles = data["Year_Old"].quantile([0.25, 0.75])
iqr = quartiles[0.75] - quartiles[0.25]

# Identify Outliers
lower_bound = quartiles[0.25] - 1.5 * iqr
upper_bound = quartiles[0.75] + 1.5 * iqr

# Filtering
Year_Old_outliers = data[(data["Year_Old"] < lower_bound) |
                (data["Year_Old"] > upper_bound)]

temp = Year_Old_outliers[["ID", "Year_Old", "Income", "CustomerFor",
                "Marital_Status", "MntSpentTotal"]]
headers = temp.columns
temp = tabulate(temp, headers, tablefmt="grid")
print("IQR: ", iqr,
      "\nYear_Old outliers: \n",
      temp, sep='')
```

```
IQR: 18.0
Year_Old outliers:
+-----+-------+------------+----------+--------------------+------------------+------------------+
|     |    ID |   Year_Old |   Income | CustomerFor        | Marital_Status   |   MntSpentTotal |
+=====+=======+============+==========+====================+==================+==================+
| 192 |  7829 |        124 |    36640 | 3953 days 00:00:00 | Divorced         |               65 |
+-----+-------+------------+----------+--------------------+------------------+------------------+
| 239 | 11004 |        131 |    60182 | 3720 days 00:00:00 | Single           |               22 |
+-----+-------+------------+----------+--------------------+------------------+------------------+
| 339 |  1150 |        125 |    83532 | 3953 days 00:00:00 | Together         |             1853 |
+-----+-------+------------+----------+--------------------+------------------+------------------+
```

```
# Extracting Outliers from the Dataset
data.drop(Year_Old_outliers.index, inplace=True)

# === #
# Quartiles and IQR
quartiles = data["Income"].quantile([0.25, 0.75])
iqr = quartiles[0.75] - quartiles[0.25]

# Identify Outliers
lower_bound = quartiles[0.25] - 1.5 * iqr
upper_bound = quartiles[0.75] + 1.5 * iqr

# Filtering
income_outliers = data[(data["Income"] < lower_bound) |
                (data["Income"] > upper_bound)]
```

```python
temp = income_outliers[["ID", "Year_Old", "Income", "CustomerFor",
                "Marital_Status", "MntSpentTotal"]]\
                    .sort_values("Income", ascending=False)
headers = temp.columns
temp = tabulate(temp, headers, tablefmt="grid")
print("IQR: ", iqr,
      "\nYear_Old outliers: \n",
      temp, sep='')
```

```
IQR: 33383.5
Year_Old outliers:
+------+-------+-----------+----------+--------------------+------------------+-----------------+
|      |    ID |   Year_Old |   Income | CustomerFor        | Marital_Status   |   MntSpentTotal |
+======+=======+===========+==========+====================+==================+=================+
| 2233 |  9432 |         47 |   666666 | 4069 days 00:00:00 | Together         |              62 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
|  617 |  1503 |         48 |   162397 | 4068 days 00:00:00 | Together         |             107 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
|  687 |  1501 |         42 |   160803 | 4371 days 00:00:00 | Married          |            1717 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
| 1300 |  5336 |         53 |   157733 | 4067 days 00:00:00 | Together         |              59 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
|  164 |  8475 |         51 |   157243 | 3797 days 00:00:00 | Married          |            1608 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
| 1653 |  4931 |         47 |   157146 | 4103 days 00:00:00 | Together         |            1730 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
| 2132 | 11181 |         75 |   156924 | 3981 days 00:00:00 | Married          |               8 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
|  655 |  5555 |         49 |   153924 | 3819 days 00:00:00 | Divorced         |               6 |
+------+-------+-----------+----------+--------------------+------------------+-----------------+
```

```python
# Extracting logically incoherent Outlier from the Dataset
income_excluded = data.drop(2233, inplace=True)
```

Finally, all excluded customer entries are stored in a single variable for later export to a CSV file, so that no data will be lost if they are ever needed.

```python
# Storing excluded entries

data_excluded = pd.concat([income_excluded,
                          Year_Old_outliers,
                          marital_filt,
                          row_nan])

temp = data_excluded[["ID", "Year_Old", "Income",
                      "Marital_Status", "MntSpentTotal"]]\
                      .sort_values("ID")
```

```r
library(kableExtra)
```

```r
temp <- py$temp
temp <- kable(temp)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

	ID	Year_Old	Income	Marital_Status	MntSpentTotal
153	92	36	34176	Alone	89
131	433	66	61331	Alone	632
2177	492	51	48432	YOLO	424
339	1150	125	83532	Together	1853
133	1295	61	NA	Married	725
2061	1612	43	NA	Single	47
10	1994	41	NA	Married	19
312	2437	35	NA	Married	1611
319	2863	54	NA	Single	1052
1382	2902	66	NA	Together	45
2081	3117	69	NA	Single	450
1386	3769	52	NA	Together	42
1383	4345	60	NA	Single	21
2134	4369	67	65487	Absurd	1169
2078	5079	53	NA	Married	97
2084	5250	81	NA	Widow	1564
27	5255	38	NA	Single	637
92	5798	51	NA	Together	985
2059	7187	55	NA	Together	721
48	7244	73	NA	Single	124
43	7281	65	NA	Single	186
138	7660	51	35860	Alone	49
2093	7734	31	79244	Absurd	1216
192	7829	124	36640	Divorced	65
128	8268	63	NA	Married	404
58	8557	42	NA	Single	46
2228	8720	46	NA	Together	1679
90	8996	67	NA	Married	603
91	9235	67	NA	Single	18
2079	10339	70	NA	Together	207
1379	10475	54	NA	Together	317
71	10629	51	NA	Married	109
239	11004	131	60182	Single	22
2202	11133	51	48432	YOLO	424

1.2 Creating categorical columns

Without outliers it is possible to continue creating categorical columns as now an extreme value will not interfere with the intervals. Categories will be made for 'Year_Old', 'Income', 'MntSpentTotal' and 'Recency'.

```python
# Age binning categories
bins = [25, 35, 45, 55, 65, 75, 2000]
labels = ["25_34", "35_44", "45_54", "55_64", "65_74", "75_above"]

data["Age_cat"] = pd.cut(data["Year_Old"], bins, labels=labels, right=False)

# Income binning categories
labels = [f"D{i+1}" for i in np.arange(0,10)]
data["Income_cat"] = pd.cut(data["Income"], 10, precision=0, labels=labels)

# Total amount spent binning categories
data["MntTotal_cat"], intervals = pd.cut(data["MntSpentTotal"], 6,
                                         precision=0, retbins=True)

# Creating new, more descriptive bins
temp, first_int = pd.cut(np.arange(2, 426), 5, retbins=True)
bins = list(first_int) + list(intervals[2:])

data["MntTotal_cat"], intervals = pd.cut(data["MntSpentTotal"], bins,
                                         precision=0, right=False,
                                         retbins=True)

# Recency binning categories
labels = ["0_24", "25_49", "50_74", "75_99"]

data["Recency_cat"] = pd.cut(data["Recency"], 4, precision=0, labels=labels)

temp = data[["MntTotal_cat", "Age_cat",
             "Income_cat", "Recency_cat"]].sample(5)

headers = temp.columns
temp = tabulate(temp, headers, tablefmt="grid")
print(temp)
```

```
+------+----------------+----------+-------------+---------------+
|      | MntTotal_cat   |  Age_cat | Income_cat  |   Recency_cat |
+======+================+==========+=============+===============+
| 1098 | [2.0, 87.0)    |    35_44 | D2          |         25_49 |
+------+----------------+----------+-------------+---------------+
| 1195 | [845.0, 1265.0) |   55_64 | D4          |         50_74 |
+------+----------------+----------+-------------+---------------+
|  471 | [2.0, 87.0)    |    35_44 | D3          |          0_24 |
+------+----------------+----------+-------------+---------------+
|  350 | [425.0, 845.0) |    65_74 | D5          |         75_99 |
+------+----------------+----------+-------------+---------------+
| 1303 | [256.0, 340.0) |    45_54 | D3          |         25_49 |
+------+----------------+----------+-------------+---------------+
```

Finally, duplicate clients will be checked for their 'ID' and data types will be standardised to 'Int64' pandas.

```python
# Check duplicated customers
duplicated = data["ID"].duplicated().any()
print(f"There are duplicated customers based on 'ID' column?: {duplicated}")
```

```
There are duplicated customers based on 'ID' column?: False
```

```python
# standardizing int dtype
for col in data.columns:
    if data[col].dtype == "int64":
        data[col] = data[col].astype("Int64")
```

When saving the dataset as a CSV, the dataset will be saved with the cleaned data as 'ifood_cleaned.csv' and the discarded data as 'ifood_excluded.csv'. It will also be useful to store the column data types in a JSON to speed up the process when the dataset needs to be imported again.

```python
# Saving DataFrame as csv
data.to_csv("../../data/ifood_cleaned.csv", index=False)

# Saving dtypes of each column
data_dtypes = data.dtypes.to_frame('dtypes').reset_index()
dict = data_dtypes.set_index('index')['dtypes'].astype(str).to_dict()

with open('../../data/cleaned_dtypes.json', 'w') as f:
    json.dump(dict, f)

# Storing excluded rows
data.to_csv("../../data/ifood_excluded.csv", index=False)
```

2 Descriptive Analysis

This section will summarise the descriptive analysis with the data considered most interesting, the full analysis can be found in the notebook '02_descriptive.ipynb' in the repository.

Dataset loading and formatting of categorical variables as the 'dtype' import does not include the order in the categories:

2.1 Categorical variables

The main interest would be in how many people accepted which campaigns in order to take this into account in the next comparisons. It can be seen how, taking into account the total sample, in each of the campaigns the level of acceptance has been less than 10%, at most 7.44% in the fourth campaign.

```python
# Acceptance campaign groups
accp_cmp1 = data["AcceptedCmp1"].value_counts()
accp_cmp2 = data["AcceptedCmp2"].value_counts()
accp_cmp3 = data["AcceptedCmp3"].value_counts()
accp_cmp4 = data["AcceptedCmp4"].value_counts()
accp_cmp5 = data["AcceptedCmp5"].value_counts()

frec_abs = pd.concat([accp_cmp1, accp_cmp2, accp_cmp3, accp_cmp4, accp_cmp5],
```

```
                     keys=["AcceptedCmp1", "AcceptedCmp2", "AcceptedCmp3",
                           "AcceptedCmp4", "AcceptedCmp5"])

frec_abs.index.names = ["Campaign", "Acceptance"]
frec_abs = pd.DataFrame(frec_abs)
frec_abs.columns = ["Frequencies"]

frec_rel = (frec_abs["Frequencies"] / len(data["ID"])).round(4)
frec_per = 100 * frec_rel

frec_tab = pd.concat([frec_abs, frec_rel, frec_per], axis=1,
                     keys=["Absolute", "Relative", "Percentage"])

frec_tab.columns.names = ["Frequencies", "drop"]
frec_tab = frec_tab.droplevel(level="drop", axis=1)

temp = frec_tab.reset_index()
temp_index = list(temp.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$Age_cat <- temp_index
temp <- temp %>%
  select(Age_cat, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

Age_cat

Campaign

Acceptance

Absolute

Relative

Percentage

0

AcceptedCmp1

0

2064

0.9361

93.61

1

AcceptedCmp1

1

141

0.0639

6.39

2

AcceptedCmp2

0

2175

0.9864

98.64

3

AcceptedCmp2

1

30

0.0136

1.36

4

AcceptedCmp3

0

2043

0.9265

92.65

5

AcceptedCmp3

1

162

0.0735

7.35

6

AcceptedCmp4

0

2041

0.9256

92.56

7

AcceptedCmp4

1

164

0.0744

7.44

8

AcceptedCmp5

0

2045

0.9274

92.74

9

AcceptedCmp5

1

160

0.0726

7.26

The following table shows the frequencies of the age category 'Age_cat', showing that the majority of customers are aged 45 and over, with the category 45 to 55 having the highest number of customers, 33%.

```
frec_abs = data["Age_cat"].value_counts().sort_index()
frec_abs_acc = data["Age_cat"].value_counts().sort_index().cumsum()

frec_rel = (frec_abs / len(data["Age_cat"])).round(4)
frec_rel_acc = (frec_abs / len(data["Age_cat"])).cumsum().round(4)

frec_per = 100 * frec_rel
frec_per_acc = (100 * frec_rel).cumsum()

frec_tab = pd.concat([frec_abs, frec_rel, frec_per], axis=1,
                     keys=["frec_abs", "frec_rel", "frec_per"])
frec_acc = pd.concat([frec_abs_acc, frec_rel_acc, frec_per_acc], axis=1,
                     keys=["frec_abs_acc", "frec_rel_acc", "frec_per_acc"])
frec_tab = pd.concat([frec_tab, frec_acc], axis=1)

temp = frec_tab
temp_index = list(temp.index.values)
```

```r
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$Age_cat <- temp_index
temp <- temp %>%
  select(Age_cat, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

Age_cat

frec_abs

frec_rel

frec_per

frec_abs_acc

frec_rel_acc

frec_per_acc

25_34

60

0.0272

2.72

60

0.0272

2.72

35_44

357

0.1619

16.19

417

0.1891

18.91

45_54

729

0.3306

33.06

1146

0.5197

51.97

55_64

502

0.2277

22.77

1648

0.7474

74.74

65_74

451

0.2045

20.45

2099

0.9519

95.19

75_above

106

0.0481

4.81

2205

1.0000

100.00

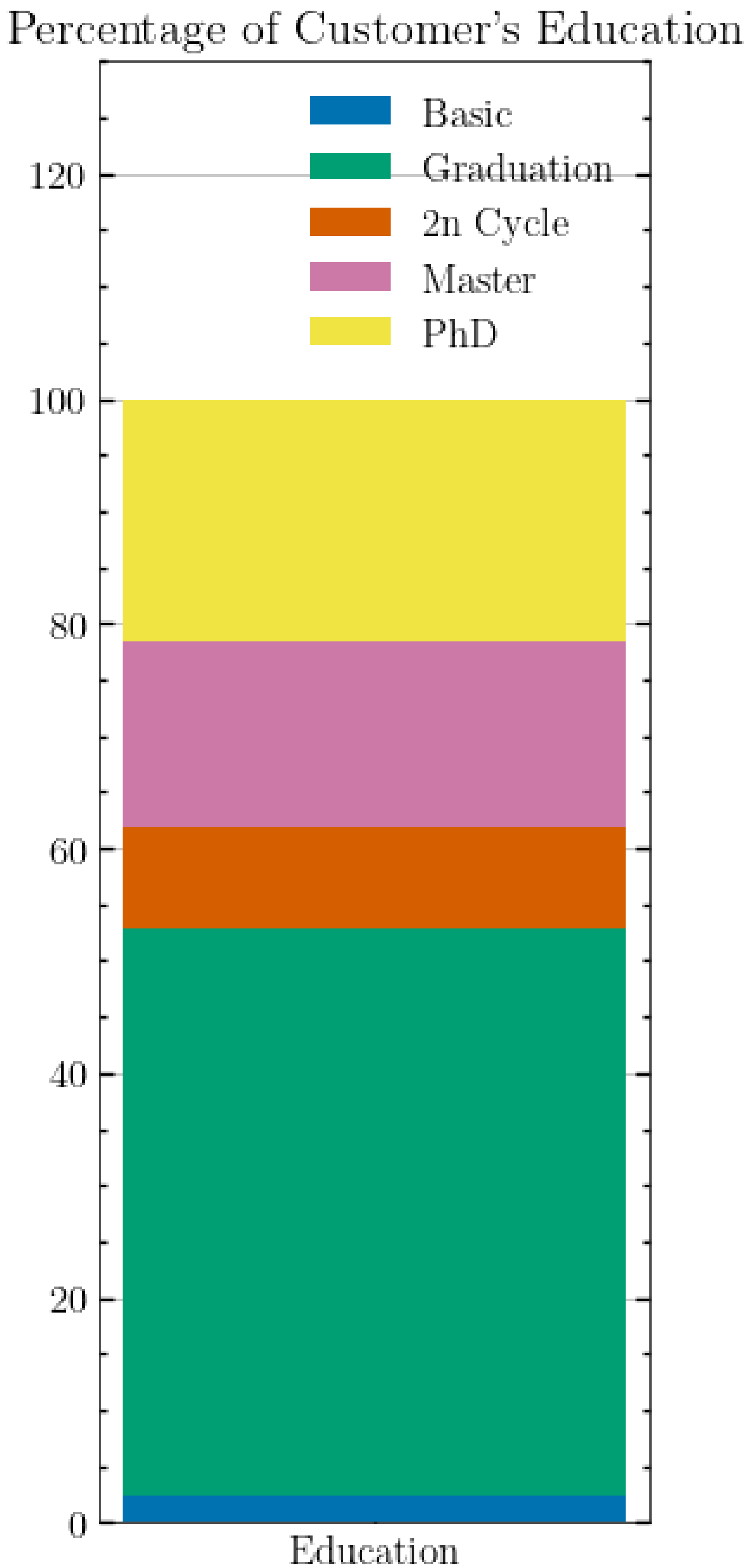
To finish with the categorical variables, another one that might be of interest is the level of study, shown below as a stacked bar chart showing the percentage of clients with different studies. The bulk are graduates but we should not underestimate PhD and Masters which between them also have a significant number of clients.

```
width = 0.2
label = "Education"
fig, ax = plt.subplots(figsize=(3,8))
bottom = np.zeros(1)

for boolean, values in frec_tab["frec_per"].items():
    p = ax.bar(label, values, width, label=boolean, bottom=bottom)
```

```
    bottom += values

ax.set_title("Percentage of Customer's Education")
ax.legend(loc="upper right")
ax.set_ylim(0,130);
plt.show();
```


Percentage of Customer's Education
Basic
Graduation
2n Cycle
Master
PhD
120
100
80
60
40
20
0
Education

2.2 Quantitative variables

First, the distribution of the variables will be checked for normal distribution by listing the quantitative columns and performing the Kolgomorov-Smirnov test on all of them.

```
# KS-Test integer columns

int_cols = ["Year_Old", "Income", "Recency", "MntSpentTotal", "MntWines",
            "MntFruits", "MntMeatProducts", "MntFishProducts",
            "MntSweetProducts", "MntGoldProds", "NumPurchasesTotal",
            "NumDealsPurchases", "NumWebPurchases", "NumCatalogPurchases",
            "NumStorePurchases", "ChildrenHome"]

def ksfunc(col):
    return stats.kstest(col, stats.norm.cdf,
                        args=(col.mean(), col.std()))

results = data[int_cols].apply(ksfunc)

results = results.T
results = results.applymap(lambda x: f"{x:.4f}")
results.columns = ["Statistic", "p-Value"]
results.columns.names = ["KS-Test"]
results.index.names = ["int_cols"]

temp = results
temp_index = list(results.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$KS_test <- temp_index
temp <- temp %>%
  select(KS_test, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

KS_test

Statistic

p-Value

Year_Old

0.0572

0.0000

Income

0.0455

0.0002

Recency

0.0685

0.0000

MntSpentTotal

0.1589

0.0000

MntWines

0.1829

0.0000

MntFruits

0.2539

0.0000

MntMeatProducts

0.2290

0.0000

MntFishProducts

0.2457

0.0000

MntSweetProducts

0.2551

0.0000

MntGoldProds

0.1977

0.0000

NumPurchasesTotal

0.1549

0.0000

NumDealsPurchases

0.2428

0.0000

NumWebPurchases

0.1491

0.0000

NumCatalogPurchases

0.1969

0.0000

NumStorePurchases

0.1810

0.0000

ChildrenHome

0.2616

0.0000

To continue, it would be interesting to observe in the same table the descriptive statistics for each variable, for which several functions will be defined to obtain the range between the minimum and the maximum, to obtain the coefficient of variation centred on the mean, and the trimmed mean as another robust measure of central tendency. In addition, a class will be defined to obtain the Winsorised Mean which is less robust than the trimmed mean and can give us an idea of the central tendency of the variables. Finally, two new columns will be generated in the results to help to see the dispersion of the data, its symmetry, thanks to the values of skewness and kurtosis.

```python
class WinsorizedMeanCalculator:
    def __init__(self, lower_percentile=0.05, upper_percentile=0.95):
        """
            Winsorized Mean (5%)
            ---
            Args:
                lower_percentile=0.05
                upper_percentile=0.95
            ---
            The k% winsorized mean is obtained by calculating the
            arithmetic mean after replacing the k% of the smallest values
            by the smallest value of the remaining values and the k% of the
            largest values by the largest value of the remaining values.
        """
        self.lower_percentile = lower_percentile
        self.upper_percentile = upper_percentile

    def winsorize(self, data):
        lower_limit = np.percentile(data, self.lower_percentile * 100)
        upper_limit = np.percentile(data, self.upper_percentile * 100)

        # Replace extreme values with the adjacent limits
        data = np.where(data < lower_limit, lower_limit, data)
        data = np.where(data > upper_limit, upper_limit, data)

        return data

    def calcu_winsor_mean(self, column):
```

```python
        winsorized_data = self.winsorize(column)
        return np.mean(winsorized_data)

winsor = WinsorizedMeanCalculator()


def range(col):
    return col.max() - col.min()

def cdv(col):
    """
        Coefficient of Variation Centered on the Mean.
        ---
        (std / mean) * 100
        ---
        Reasonable dispersion is associated with coefficients of variation
        less than 50. Coefficients of variation greater than 50 indicate a
        lot of dispersion. Coefficients greater than 100 are generally
        indicative of strong anomalies in the data.
    """
    return (col.std() / abs(col.mean())) * 100

def trimean(series):
    """
        BESA (best easy systematic average)
        ---
        (Q1 + 2*Q2 + Q3) / 4
        ---
        Highly robust central tendency statistic
    """
    Q1 = series.quantile(0.25)
    median = series.median()
    Q3 = series.quantile(0.75)
    return (Q1 + 2 * median + Q3) / 4
```

```python
results = data[int_cols].agg(["mean", "median", trimean,
                              #stats.median_abs_deviation,
                              "std", cdv,
                              range, "min", "max"])

winsor_mean = data[int_cols].agg(winsor.calcu_winsor_mean)
skew = data[int_cols].agg(stats.skew, bias=False)
kurt = data[int_cols].agg(stats.kurtosis, bias=False)
skew_kurt = pd.concat([winsor_mean, skew, kurt], axis=1)
skew_kurt.columns = ["winsor_mean", "skew", "kurt"]

results = results.T

results = pd.concat([results, skew_kurt], axis=1)
results["skew/std"] = (results["skew"] / results["std"])
results["kurt/std"] = (results["kurt"] / results["std"])

#    If the result is between -2 and 2, the distribution can be assumed to
```

```python
#     be symmetric (or meso-kurtic); if it is greater than 2, the distribution
#     can be said to be positively skewed (or leptokurtic); and if it is
#     less than -2, the distribution can be said to be
#     negatively skewed (or platykurtic).


results = results.map(lambda x: f"{x:.2f}")

results.columns.names = ["Descriptive Stats"]
results.index.names = ["int_cols"]

temp = results
temp_index = list(results.index.values)
```

```r
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$Columns <- temp_index
temp <- temp %>%
  select(Columns, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

Columns

mean

median

trimean

std

cdv

range

min

max

winsor_mean

skew

kurt

skew/std

kurt/std

Year_Old

55.10

54.00

55.00

11.70

21.23

56.00

28.00

84.00

55.07

0.10

-0.80

0.01

-0.07

Income

51954.62

51373.00

51607.25

21544.43

41.47

160667.00

1730.00

162397.00

51717.86

0.35

0.72

0.00

0.00

Recency

49.08

49.00

49.00

28.94

58.97

99.00

0.00

99.00

49.07

-0.00

-1.20

-0.00

-0.04

MntSpentTotal

607.38

396.00

477.25

602.97

99.27

2520.00

5.00

2525.00

595.94

0.86

-0.35

0.00

-0.00

MntWines

305.39

174.00

219.25

337.68

110.57

1493.00

0.00

1493.00

296.71

1.17

0.58

0.00

0.00

MntFruits

26.33

8.00

12.75

39.75

150.98

199.00

0.00

199.00

24.74

2.11

4.08

0.05

0.10

MntMeatProducts

167.20

68.00

96.25

224.41

134.22

1725.00

0.00

1725.00

159.66

2.03

5.06

0.01

0.02

MntFishProducts

37.57

12.00

19.25

54.60

145.35

259.00

0.00

259.00

35.70

1.92

3.10

0.04

0.06

MntSweetProducts

27.09

8.00

12.50

41.13

151.82

262.00

0.00

262.00

25.49

2.10

4.09

0.05

0.10

MntGoldProds

43.81

24.00

28.25

51.54

117.67

321.00

0.00

321.00

41.98

1.84

3.16

0.04

0.06

NumPurchasesTotal

12.57

12.00

12.25

7.21

57.38

32.00

0.00

32.00

12.45

0.29

-1.12

0.04

-0.16

NumDealsPurchases

2.32

2.00

2.00

1.92

82.93

15.00

0.00

15.00

2.23

2.43

9.03

1.26

4.69

NumWebPurchases

4.08

4.00

4.00

2.74

67.08

27.00

0.00

27.00

4.03

1.20

4.09

0.44

1.49

NumCatalogPurchases

2.67

2.00

2.00

2.93

109.51

28.00

0.00

28.00

2.59

1.88

8.11

0.64

2.77

NumStorePurchases

5.81

5.00

5.25

3.25

56.04

13.00

0.00

13.00

5.79

0.70

-0.64

0.21

-0.20

ChildrenHome

0.95

1.00

0.75

0.75

79.11

3.00

0.00

3.00

0.92

0.41

-0.26

0.55

-0.35

If the above table with the descriptives is observed, it can be seen that the six variables corresponding to the money spent by type of product are the ones with the greatest dispersion. Also the asymmetry corresponding to the purchase platform, whether web, in-store or catalogue, are the three variables with the highest leptokurtic asymmetry.

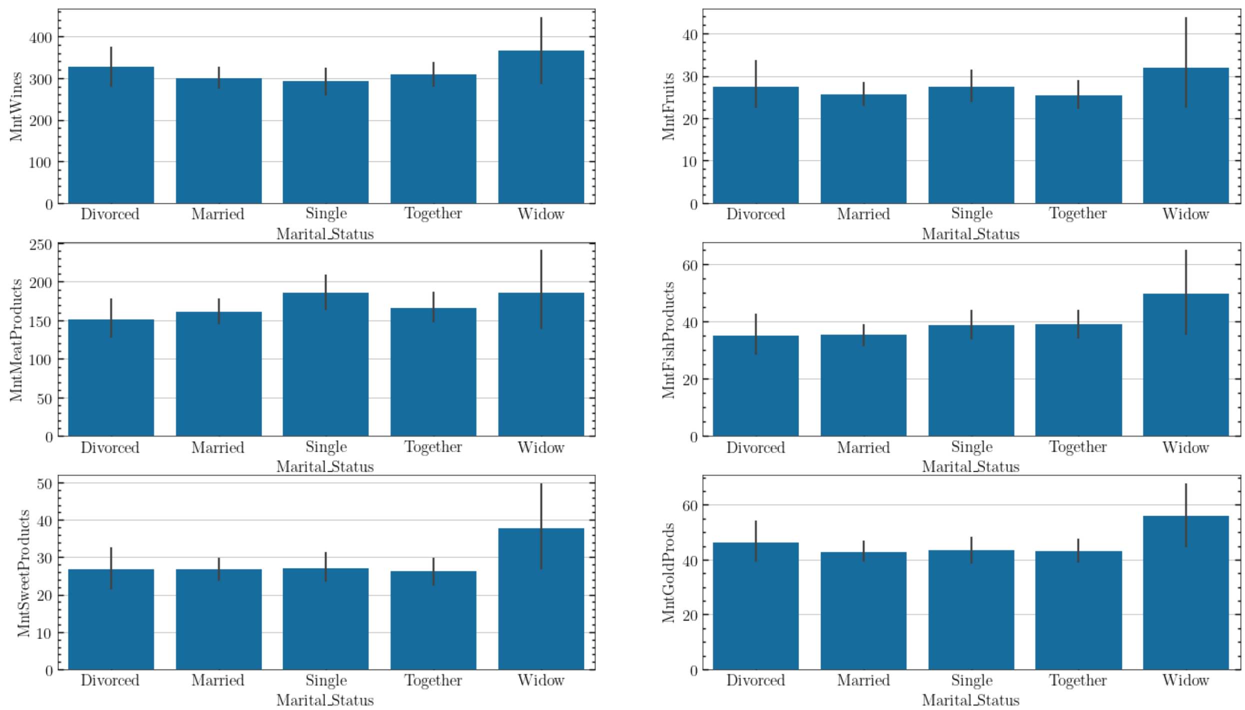
As mentioned before, in the notebook is in greater detail the analysis, showing a histogram with density line one of the variables mentioned.

Finally, taking into account the variable 'MntSpentTotal' the total amount of money they have spent, will be shown in comparison with age groups and marital status. Also, taking into account the marital status, bar charts will be displayed for each type of product and its corresponding expenditure.

```
fig, ax =plt.subplots(3,2)

sns.barplot(ax=ax[0,0], x="Marital_Status",
            y="MntWines", data=data);
sns.barplot(ax=ax[0,1], x="Marital_Status",
            y="MntFruits", data=data);
sns.barplot(ax=ax[1,0], x="Marital_Status",
            y="MntMeatProducts", data=data);
sns.barplot(ax=ax[1,1], x="Marital_Status",
            y="MntFishProducts", data=data);
sns.barplot(ax=ax[2,0], x="Marital_Status",
            y="MntSweetProducts", data=data);
sns.barplot(ax=ax[2,1], x="Marital_Status",
            y="MntGoldProds", data=data);

plt.show()
```



The following two tables that will conclude the descriptive analysis section show the descriptive statistics separating those who have accepted any campaign (represented by `_y`) from those who have not accepted any (`_n`).

```
spent_mnt = ["MntWines", "MntFruits", "MntMeatProducts",
             "MntFishProducts", "MntSweetProducts", "MntGoldProds"]

dataycmp = data[data["AcceptedCmpTotal"] > 0]
datancmp = data[data["AcceptedCmpTotal"] == 0]

results_y = dataycmp[spent_mnt].agg(["mean", "median", "std",
                                     cdv, "min", "max"])
results_n = datancmp[spent_mnt].agg(["mean", "median", "std",
                                     cdv, "min", "max"])

temp = pd.merge(results_y.T, results_n.T,
                on=results_y.columns, suffixes=["_y", "_n"])

temp.index = temp["key_0"]
temp = temp[['mean_y', 'mean_n', 'median_y', 'median_n', 'std_y', 'std_n',
            'cdv_y', 'cdv_n', 'min_y', 'min_n', 'max_y', 'max_n']]
temp.index.names = ["Amount Spent"]
temp.columns.names = ["Results"]
temp = temp.map(lambda x: f"{x:.2f}")
temp_index = list(temp.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
```

```r
temp_index = py$temp_index

# Creating Index
temp$Columns <- temp_index
temp <- temp %>%
  select(Columns, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

Columns

mean_y

mean_n

median_y

median_n

std_y

std_n

cdv_y

cdv_n

min_y

min_n

max_y

max_n

MntWines

613.20

225.14

603.00

104.00

401.09

265.97

65.41

118.14

0.00

0.00

1493.00

1396.00

MntFruits

36.02

23.80

20.50

7.00

45.02

37.87

124.99

159.09

0.00

0.00

190.00

199.00

MntMeatProducts

287.52

135.82

191.50

51.00

272.40

198.54

94.74

146.18

1.00

0.00

974.00

1725.00

MntFishProducts

54.60

33.13

29.00

11.00

64.52

50.80

118.15

153.35

0.00

0.00

253.00

259.00

MntSweetProducts

39.91

23.75

19.50

7.00

49.07

38.11

122.96

160.46

0.00

0.00

194.00

262.00

MntGoldProds

62.83

38.84

39.00

20.00

58.94

48.23

93.80

124.17

0.00

0.00

242.00

321.00

The table above shows how the averages for all types of products seem to be higher in the group of people who accepted a campaign, and also thanks to the value provided by the dispersion coefficient, there is less dispersion in their data.

```
numpurchases = ["NumPurchasesTotal", "NumWebPurchases", "NumStorePurchases",
                "NumCatalogPurchases", "NumDealsPurchases"]

results_y = dataycmp[numpurchases].agg(["mean", "median", "std",
                                        cdv, "min", "max"])
```

```
results_n = datancmp[numpurchases].agg(["mean", "median", "std",
                                        cdv, "min", "max"])

temp = pd.merge(results_y.T, results_n.T,
                on=results_y.columns, suffixes=["_y", "_n"])

temp.index = temp["key_0"]
temp = temp[['mean_y', 'mean_n', 'median_y', 'median_n', 'std_y', 'std_n',
            'cdv_y', 'cdv_n', 'min_y', 'min_n', 'max_y', 'max_n']]
temp.index.names = ["Purchases"]
temp.columns.names = ["Results"]
temp = temp.map(lambda x: f"{x:.2f}")
temp_index = list(temp.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$Columns <- temp_index
temp <- temp %>%
  select(Columns, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

Columns

mean_y

mean_n

median_y

median_n

std_y

std_n

cdv_y

cdv_n

min_y

min_n

max_y

max_n

NumPurchasesTotal

16.71

11.48

18.00

10.00

6.58

6.97

39.38

60.72

1.00

0.00

32.00

31.00

NumWebPurchases

5.24

3.78

5.00

3.00

2.60

2.70

49.62

71.26

0.00

0.00

11.00

27.00

NumStorePurchases

7.01

5.50

7.00

4.00

3.24

3.19

46.29

57.96

0.00

0.00

13.00

13.00

NumCatalogPurchases

4.46

2.20

4.00

1.00

2.85

2.76

63.90

125.21

0.00

0.00

11.00

28.00

NumDealsPurchases

2.00

2.40

1.00

2.00

1.76

1.96

88.08

81.37

0.00

0.00

11.00

15.00

Following the previous line, the average number of purchases is higher in those who accepted a campaign but not in purchases with an offer, which seem to be higher in the group that did not accept campaigns. Similarly, there is less dispersion of the data in those who accepted campaigns.

3 Inferential Analysis

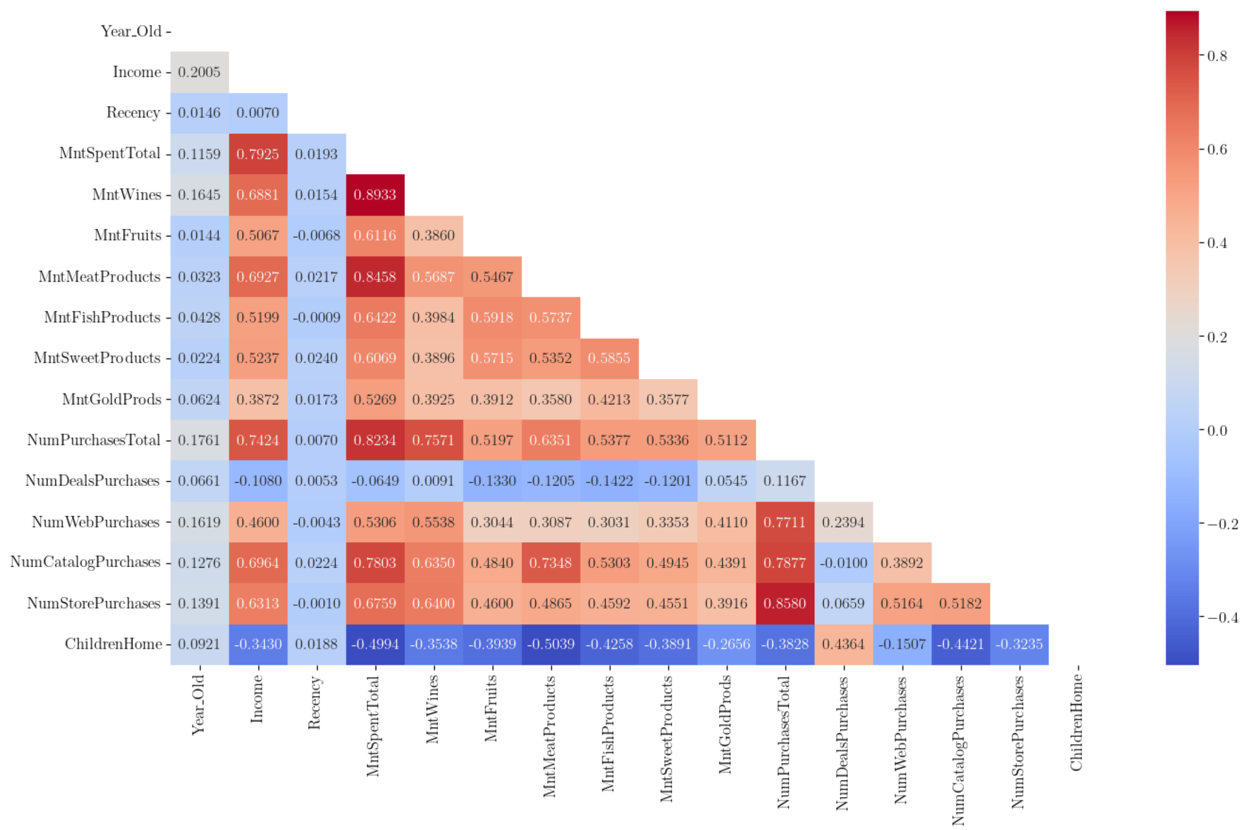
Thanks to the inferential analysis that follows in this section, it will be possible to observe relationships and comparisons of groups, mainly those who accept offers with those who do not, in order to obtain a customer profile that is interesting for the Marketing team and thus in the next campaign to obtain more beneficial numbers.

3.1 Correlations

Three correlation matrices of the quantitative variables will be shown below with heat maps for better visualisation, one for the whole dataset as a whole and then two differentiating between those who did and did not accept a campaign.

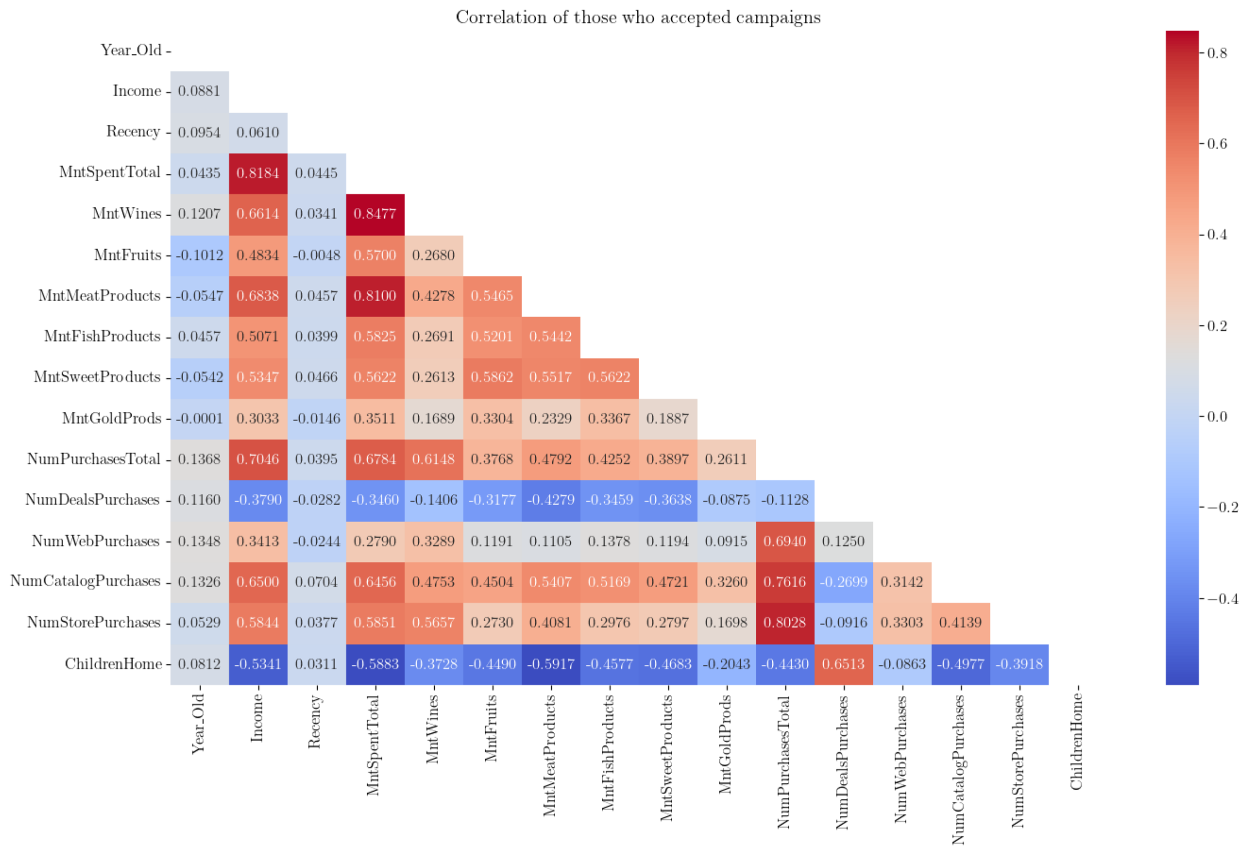
```
corr_matrix = data[int_cols].corr()
mask = np.triu(np.ones_like(corr_matrix, dtype = bool))

sns.heatmap(corr_matrix, cmap="coolwarm", annot=True, mask=mask, fmt=".4f")
plt.show()
```



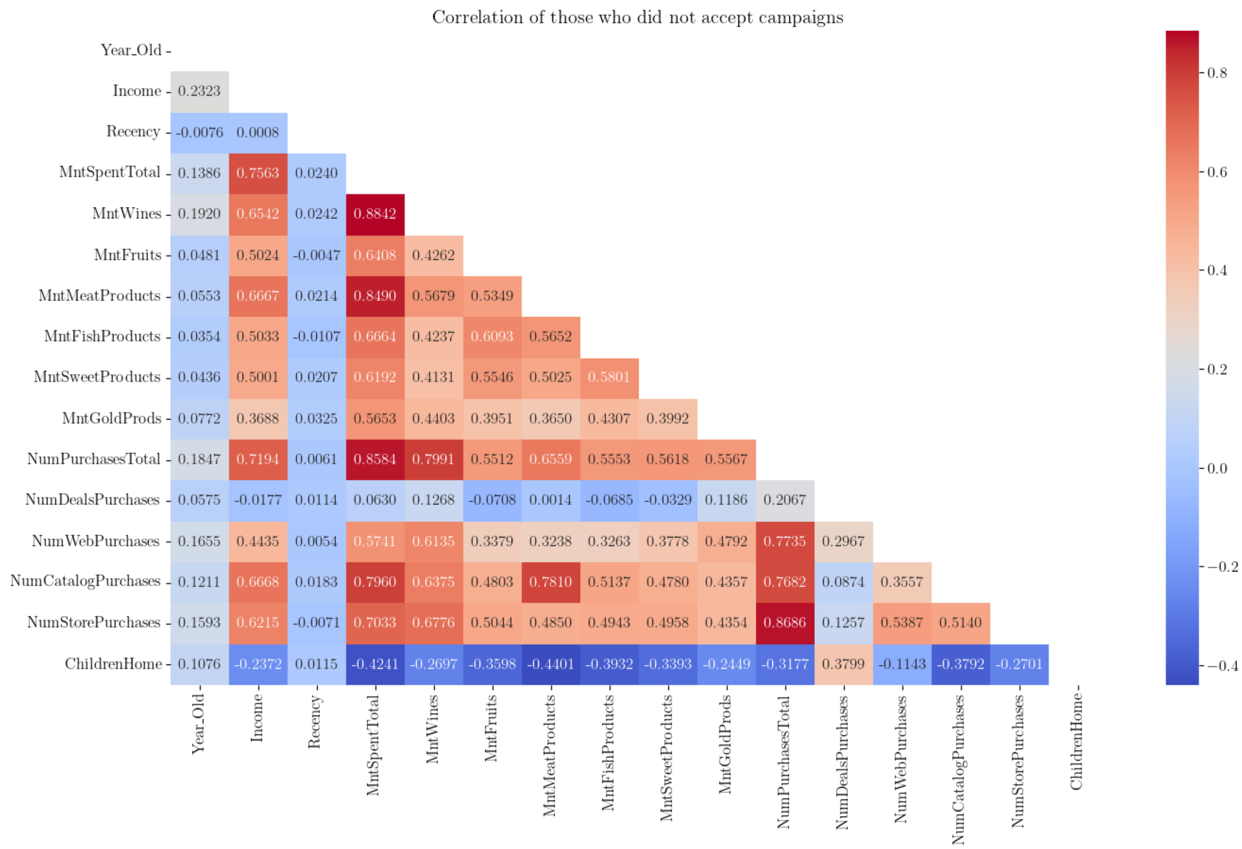

```
corr_matrix = data_ycmp[int_cols].corr()
mask = np.triu(np.ones_like(corr_matrix, dtype = bool))

sns.heatmap(corr_matrix, cmap="coolwarm", annot=True,
            mask=mask, fmt=".4f").set_title("Correlation of those "
                                            "who accepted campaigns")
plt.show()
```




```python
corr_matrix = data_ncmp[int_cols].corr()
mask = np.triu(np.ones_like(corr_matrix, dtype = bool))

sns.heatmap(corr_matrix, cmap="coolwarm", annot=True,
            mask=mask, fmt=".4f").set_title("Correlation of those who did "
                                             "not accept campaigns")
plt.show()
```

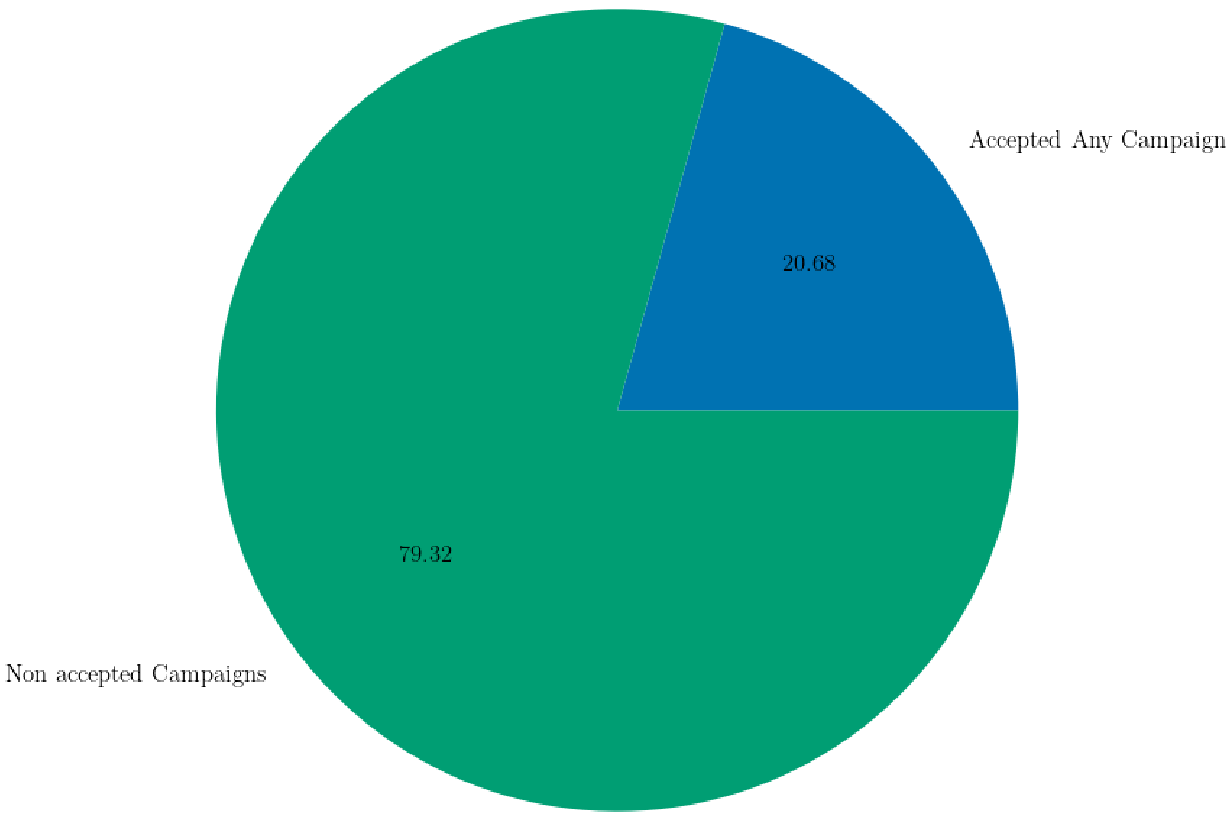


The strengths of the relationships are similar for all three correlation matrices. A stronger relationship is observed between the income customers have with the total spent as well as in the number of purchases made. It appears that the highest number of purchases is related to those made directly in the shop, with wine and meat being the most expensive, although in terms of money spent it appears to be related to purchases through the catalogue. Offers are then negatively related to those who have accepted a campaign, and offers are also positively related to the children they have at home.

3.2 Comparisons

First, the percentage of customers who accepted a campaign and those who did not accept a campaign will be illustrated in a clear and simple way.

```
fig, ax = plt.subplots()
ax.pie([data_ycmp["ID"].count(),
        data_ncmp["ID"].count()],
        labels=["Accepted Any Campaign", "Non accepted Campaigns"],
        autopct='%1.2f%%',
        )
plt.show();
```



Before starting with the comparisons, the assumptions of normality and homoscedasticity will be checked, thereby deciding on the most appropriate type of test to perform the between-group comparisons.

```
# KS-Test and Levene on integer columns by campaign acceptance

int_cols = ["Year_Old", "Income", "Recency", "MntSpentTotal", "MntWines",
            "MntFruits", "MntMeatProducts", "MntFishProducts",
            "MntSweetProducts", "MntGoldProds", "NumPurchasesTotal",
            "NumDealsPurchases", "NumWebPurchases", "NumCatalogPurchases",
            "NumStorePurchases", "ChildrenHome"]

# KS
def ksfunc(col):
    return stats.kstest(col, stats.norm.cdf,
                        args=(col.mean(), col.std()))

results_y = data_ycmp[int_cols].apply(ksfunc)
results_n = data_ncmp[int_cols].apply(ksfunc)

results_ks = pd.concat([results_n, results_y], )
```

```python
results_ks = results_ks.T
results_ks = results_ks.applymap(lambda x: f"{x:.4f}")
results_ks.columns = ["Statistic (y)", "p-Value (y)",
                      "Statistic (n)", "p-Value (n)"]
results_ks.columns.names = ["KS-Test"]
results_ks.index.names = ["int_cols"]

# Levene
results = stats.levene(data_ycmp[int_cols],
                       data_ncmp[int_cols],
                       center="mean")
results_stats = pd.Series(results.statistic)
results_stats.index = int_cols
results_pvalue = pd.Series(results.pvalue)
results_pvalue.index = int_cols

results_lev = pd.concat([results_stats, results_pvalue], axis=1)
results_lev.columns = ["Statistic", "p-Value"]
results_lev.columns.names = ["Levene"]
results_lev.index.names = ["int_cols"]

results_lev = results_lev.map(lambda x: f"{x:.4f}")

# Combining frames
results = pd.concat([results_ks, results_lev], axis=1)
column_names = [("Ks-test", "Statistic (y)"), ("Ks-test", "p-Value (y)"),
                ("Ks-test", "Statistic (n)"), ("Ks-test", "p-Value (n)"),
                ("Levene", "Statistic"), ("Levene", "p-Value")]
results.columns = pd.MultiIndex.from_tuples(column_names)

temp = results
temp_index = list(temp.index.values)
```

```r
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$Columns <- temp_index
temp <- temp %>%
  select(Columns, everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")
```

```
temp
```

Columns

('Ks-test', 'Statistic (y)')

('Ks-test', 'p-Value (y)')

('Ks-test', 'Statistic (n)')

('Ks-test', 'p-Value (n)')

('Levene', 'Statistic')

('Levene', 'p-Value')

Year_Old

0.0617

0.0000

0.0635

0.0483

8.2130

0.0042

Income

0.0466

0.0010

0.1010

0.0002

0.0055

0.9407

Recency

0.0698

0.0000

0.0887

0.0014

0.3926

0.5310

MntSpentTotal

0.1779

0.0000

0.0704

0.0208

86.3006

0.0000

MntWines

0.1986

0.0000

0.0710

0.0191

203.0794

0.0000

MntFruits

0.2648

0.0000

0.2118

0.0000

24.2888

0.0000

MntMeatProducts

0.2480

0.0000

0.1667

0.0000

151.9987

0.0000

MntFishProducts

0.2572

0.0000

0.1987

0.0000

64.9998

0.0000

MntSweetProducts

0.2666

0.0000

0.2080

0.0000

67.5303

0.0000

MntGoldProds

0.2103

0.0000

0.1937

0.0000

41.8228

0.0000

NumPurchasesTotal

0.1767

0.0000

0.1031

0.0001

23.0218

0.0000

NumDealsPurchases

0.2423

0.0000

0.3002

0.0000

2.4770

0.1157

NumWebPurchases

0.1690

0.0000

0.1024

0.0001

0.0167

0.8971

NumCatalogPurchases

0.2245

0.0000

0.1063

0.0001

14.1206

0.0002

NumStorePurchases

0.2073

0.0000

0.0933

0.0007

0.4095

0.5223

ChildrenHome

0.2806

0.0000

0.3075

0.0000

18.3733

0.0000

3.2.1 Parametric pairwise comparison

The T-test for independent samples will be performed on those columns where it has been observed that the homoscedasticity assumption is met, being a large sample (>50) normality may not be met, but both assumptions cannot be violated for this test. The Null Hypothesis of the T-test for independent samples is equality in the means of the groups compared, therefore a *p* value of less than 0.05 would indicate that the groups are not equal.

The comparison is whether there are differences between the means of the groups that did or did not accept a campaign on the variables 'Income', 'Recency', 'NumDealsPurchases', 'NumWebPurchases', and 'NumStorePurchases'. A *p* value of less than 0.05 indicates that the group means are not equal. The effect size was also tested with Cohen's *d* with the author's recommended cut-off points of 0.20: small, 0.50: medium, 0.80: large [Cohen, 1992].

```
cols_tt = ["Income", "Recency", "NumDealsPurchases",
           "NumWebPurchases", "NumStorePurchases"]

for col in cols_tt:

    results_ks1 = stats.kstest(data_ycmp[col], stats.norm.cdf,
                               args=(data_ycmp[col].mean(),
                                     data_ycmp[col].std()))
    results_ks2 = stats.kstest(data_ncmp[col], stats.norm.cdf,
                               args=(data_ncmp[col].mean(),
                                     data_ncmp[col].std()))
    results_lev = stats.levene(data_ycmp[col], data_ncmp[col])
    ttest = stats.ttest_ind(data_ycmp[col], data_ncmp[col])
    cohend = (data_ycmp[col].mean() - data_ncmp[col].mean()) / data[col].std()

    print(
        f"\t{col} comparison\n"
        "\nAccepted Campaign Group: "
        f"\n\tMean: {data_ycmp[col].mean():.3f}"
        f"\n\tStd: {data_ycmp[col].std():.3f}"
        f"\n\tKS (p): {results_ks1.pvalue:.3f}"
```

```
        "\nnon-Accepted Campaign Group: "
        f"\n\tMean: {data_ncmp[col].mean():.3f}"
        f"\n\tStd: {data_ncmp[col].std():.3f}"
        f"\n\tKS (p): {results_ks1.pvalue:.3f}"
        f"\n\nLevene: {results_lev.pvalue:.3f}"
        "\nT-Test: "
        f"\n\tStatistic: {ttest.statistic:.3f}"
        f"\n\tp-value: {ttest.pvalue:.3f}"
        f"\nCohen's D: {cohend:.3f}"
        "\n\n\t======\n"
    )
```

```
    Income comparison

Accepted Campaign Group:
    Mean: 65249.292
    Std: 20285.501
    KS (p): 0.000
non-Accepted Campaign Group:
    Mean: 48488.422
    Std: 20494.046
    KS (p): 0.000

Levene: 0.806
T-Test:
    Statistic: 15.587
    p-value: 0.000
Cohen's D: 0.778

    ======

    Recency comparison

Accepted Campaign Group:
    Mean: 48.096
    Std: 29.032
    KS (p): 0.001
non-Accepted Campaign Group:
    Mean: 49.340
    Std: 28.922
    KS (p): 0.001

Levene: 0.533
T-Test:
    Statistic: -0.817
    p-value: 0.414
Cohen's D: -0.043

    ======

    NumDealsPurchases comparison

Accepted Campaign Group:
```

```
    Mean: 1.998
    Std: 1.760
    KS (p): 0.000
non-Accepted Campaign Group:
    Mean: 2.405
    Std: 1.957
    KS (p): 0.000

Levene: 0.068
T-Test:
    Statistic: -4.036
    p-value: 0.000
Cohen's D: -0.211

    ======

    NumWebPurchases comparison

Accepted Campaign Group:
    Mean: 5.239
    Std: 2.600
    KS (p): 0.000
non-Accepted Campaign Group:
    Mean: 3.784
    Std: 2.696
    KS (p): 0.000

Levene: 0.661
T-Test:
    Statistic: 10.339
    p-value: 0.000
Cohen's D: 0.531

    ======

    NumStorePurchases comparison

Accepted Campaign Group:
    Mean: 7.007
    Std: 3.244
    KS (p): 0.001
non-Accepted Campaign Group:
    Mean: 5.496
    Std: 3.185
    KS (p): 0.001

Levene: 0.063
T-Test:
    Statistic: 8.987
    p-value: 0.000
Cohen's D: 0.464

    ======
```

3.2.2 Non-parametric pairwise comparison

For the remaining variables that do not meet the assumptions, the Mann-Whitney U test, a robust test equivalent to the independent samples t-test, will be used. Also the effect size will be measured this time with Wendt's formula for the Biserial Range [Wendt, 1972] specific to the U-statistic. In this case the strength of the effect will be 0.10: small, 0.30: medium, 0.50: large [Coolican and Coolican, 2013].

```
cols_umw = ["Year_Old", "MntSpentTotal", "MntWines",
            "MntFruits", "MntMeatProducts", "MntFishProducts",
            "MntSweetProducts", "MntGoldProds", "NumPurchasesTotal",
            "NumCatalogPurchases", "ChildrenHome"]

for col in cols_umw:

    results_ks1 = stats.kstest(data_ycmp[col], stats.norm.cdf,
                        args=(data_ycmp[col].mean(),
                                data_ycmp[col].std()))
    results_ks2 = stats.kstest(data_ncmp[col], stats.norm.cdf,
                        args=(data_ncmp[col].mean(),
                                data_ncmp[col].std()))
    results_lev = stats.levene(data_ycmp[col],
                        data_ncmp[col])
    ttest = stats.mannwhitneyu(data_ycmp[col],
                        data_ncmp[col],
                        use_continuity=True)

    # Rank-Biserial's Wendt formula
    n1, n2 = len(data_ycmp[col]), len(data_ncmp[col])
    rbis = 1 - (2 * ttest.statistic) / (n1 * n2)

    print(
        f"\t{col} comparison\n"
        "\nAccepted Campaign Group: "
        f"\n\tMedian: {data_ycmp[col].median():.3f}"
        f"\n\tStd: {data_ycmp[col].std():.3f}"
        f"\n\tKS (p): {results_ks1.pvalue:.3f}"
        "\nnon-Accepted Campaign Group: "
        f"\n\tMedian: {data_ncmp[col].median():.3f}"
        f"\n\tStd: {data_ncmp[col].std():.3f}"
        f"\n\tKS (p): {results_ks1.pvalue:.3f}"
        f"\n\nLevene: {results_lev.pvalue:.3f}"
        "\nMann-Whitney U test: "
        f"\n\tStatistic: {ttest.statistic:.3f}"
        f"\n\tp-value: {ttest.pvalue:.3f}"
        f"\nRank-Biserial r: {rbis:.3f}"
        "\n\n\t=====\n"
    )
```

```
    Year_Old comparison

Accepted Campaign Group:
    Median: 55.000
    Std: 12.514
    KS (p): 0.048
```

```
non-Accepted Campaign Group:
    Median: 54.000
    Std: 11.469
    KS (p): 0.048

Levene: 0.004
Mann-Whitney U test:
    Statistic: 416979.000
    p-value: 0.133
Rank-Biserial r: -0.046

    =====

    MntSpentTotal comparison

Accepted Campaign Group:
    Median: 1153.500
    Std: 672.186
    KS (p): 0.021
non-Accepted Campaign Group:
    Median: 257.000
    Std: 512.734
    KS (p): 0.021

Levene: 0.000
Mann-Whitney U test:
    Statistic: 607506.500
    p-value: 0.000
Rank-Biserial r: -0.523

    =====

    MntWines comparison

Accepted Campaign Group:
    Median: 603.000
    Std: 401.087
    KS (p): 0.019
non-Accepted Campaign Group:
    Median: 104.000
    Std: 265.975
    KS (p): 0.019

Levene: 0.000
Mann-Whitney U test:
    Statistic: 621951.500
    p-value: 0.000
Rank-Biserial r: -0.560

    =====

    MntFruits comparison

Accepted Campaign Group:
```

```
    Median: 20.500
    Std: 45.017
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 7.000
    Std: 37.867
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 477254.500
    p-value: 0.000
Rank-Biserial r: -0.197

    =====

    MntMeatProducts comparison

Accepted Campaign Group:
    Median: 191.500
    Std: 272.403
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 51.000
    Std: 198.543
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 551542.500
    p-value: 0.000
Rank-Biserial r: -0.383

    =====

    MntFishProducts comparison

Accepted Campaign Group:
    Median: 29.000
    Std: 64.516
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 11.000
    Std: 50.798
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 471952.500
    p-value: 0.000
Rank-Biserial r: -0.184

    =====
```

```
    MntSweetProducts comparison

Accepted Campaign Group:
    Median: 19.500
    Std: 49.070
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 7.000
    Std: 38.111
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 472530.500
    p-value: 0.000
Rank-Biserial r: -0.185

    =====

    MntGoldProds comparison

Accepted Campaign Group:
    Median: 39.000
    Std: 58.940
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 20.000
    Std: 48.234
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 528807.000
    p-value: 0.000
Rank-Biserial r: -0.326

    =====

    NumPurchasesTotal comparison

Accepted Campaign Group:
    Median: 18.000
    Std: 6.581
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 10.000
    Std: 6.973
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 559706.000
    p-value: 0.000
Rank-Biserial r: -0.404
```

```
    =====

    NumCatalogPurchases comparison

Accepted Campaign Group:
    Median: 4.000
    Std: 2.853
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 1.000
    Std: 2.761
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 594362.000
    p-value: 0.000
Rank-Biserial r: -0.490

    =====

    ChildrenHome comparison

Accepted Campaign Group:
    Median: 1.000
    Std: 0.709
    KS (p): 0.000
non-Accepted Campaign Group:
    Median: 1.000
    Std: 0.737
    KS (p): 0.000

Levene: 0.000
Mann-Whitney U test:
    Statistic: 279934.500
    p-value: 0.000
Rank-Biserial r: 0.298

    =====
```

```python
result_mwu = pg.pairwise_tests(data=data, dv="MntSpentTotal",
                               between="Response", parametric=False,
                               alpha=0.05,)

# Rank-Biserial's Wendt formula
n1, n2 = len(data[data["Response"] == 0]), len(data[data["Response"] == 1])
rbis = 1 - (2 * result_mwu["U-val"]) / (n1 * n2)

result_mwu = pd.concat([result_mwu, rbis], axis=1)
result_mwu.columns = result_mwu.columns[:-1].tolist() + ["rbis"]

print("MntSpentTotal mean in last campaign with no Acceptance:\n"
      f"\t{data[data["Response"] == 0]["MntSpentTotal"].mean():.4f}")
```

```
MntSpentTotal mean in last campaign with no Acceptance:
    540.1269
```

```
print("MntSpentTotal mean in last campaign with Acceptance:\n"
      f"\t{data[data["Response"] == 1]["MntSpentTotal"].mean():.4f}\n")
```

```
MntSpentTotal mean in last campaign with Acceptance:
    989.5030
```

```
print("MntSpentTotal std in last campaign with no Acceptance:\n"
      f"\t{data[data["Response"] == 0]["MntSpentTotal"].std():.4f}")
```

```
MntSpentTotal std in last campaign with no Acceptance:
    553.4761
```

```
print("MntSpentTotal std in last campaign with Acceptance:\n"
      f"\t{data[data["Response"] == 1]["MntSpentTotal"].std():.4f}\n")
```

```
MntSpentTotal std in last campaign with Acceptance:
    720.0314
```

```
temp = result_mwu
temp_index = list(temp.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$'.' <- temp_index
temp <- temp %>%
  select('.', everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

.

Contrast

A

B

Paired

Parametric

U-val

alternative

p-unc

hedges

rbis

0

Response

0

1

FALSE

FALSE

189583.5

two-sided

0

-0.7726728

0.3872048

3.2.3 Multiple comparison

In the case of comparisons of more than 2 groups, since the ANOVA test cannot be performed due to non-compliance with assumptions, the alternative will be the Kruskal Wallis H test. In the case of significant group comparisons, the Games Howell *post-hoc* test will perform all possible pairwise comparisons and we will obtain the strength of the effect of each comparison with Eta-square, with cut-off points at 0.04: minimum necessary, 0.25: moderate, 0.64: strong [Ferguson, 2009].

```
data["cmp_accp"] = np.where(data["AcceptedCmpTotal"] > 0, 1, 0)
```

Comenzando por los grupos en los que no se ha visto que haya diferencias entre sus categorías, están 'Recency', 'Recency_cat' y 'Marital_Status'.

```
result_kw1 = pg.kruskal(data=data,
                        dv="MntSpentTotal",
                        between="Recency_cat")


result_kw3 = pg.kruskal(data=data,
                        dv="MntSpentTotal",
                        between="Marital_Status")


result_kw2 = pg.kruskal(data=data,
```

```
                       dv="Recency",
                       between="AcceptedCmpTotal")

result_kw = pd.concat([result_kw1, result_kw3, result_kw2])

headers = result_kw.columns
result_kw = tabulate(result_kw, headers, tablefmt="grid")
print("MntSpentTotal on Recency_cat and Marital_Status; "
    "Also Recency on AcceptedCmpTotal:\n",
    result_kw, sep='')
```

```
MntSpentTotal on Recency_cat and Marital_Status; Also Recency on AcceptedCmpTotal:
+---------+------------------+---------+---------+----------+
|         | Source           |   ddof1 |       H |    p-unc |
+=========+==================+=========+=========+==========+
| Kruskal | Recency_cat      |       3 | 5.31594 | 0.150072 |
+---------+------------------+---------+---------+----------+
| Kruskal | Marital_Status   |       4 | 5.18688 | 0.268654 |
+---------+------------------+---------+---------+----------+
| Kruskal | AcceptedCmpTotal |       4 | 2.66907 | 0.614638 |
+---------+------------------+---------+---------+----------+
```

The following comparisons will be around how many campaigns customers have accepted, ranging from 0 to 4, on different variables.

```
result_kw = pg.kruskal(data=data,
                       dv="NumStorePurchases",
                       between="AcceptedCmpTotal")

result_pair = pg.pairwise_gameshowell(data=data,
                                      dv="NumStorePurchases",
                                      between="AcceptedCmpTotal",
                                      effsize="eta-square")

result_kw = round(result_kw, 4)
result_pair = round(result_pair, 4)

headers = result_kw.columns
result_kw = tabulate(result_kw, headers, tablefmt="grid")
print("NumStorePurchases on AcceptedCmpTotal: \n",result_kw, sep='')
```

```
NumStorePurchases on AcceptedCmpTotal:
+---------+------------------+---------+---------+---------+
|         | Source           |   ddof1 |       H |   p-unc |
+=========+==================+=========+=========+=========+
| Kruskal | AcceptedCmpTotal |       4 | 108.858 |       0 |
+---------+------------------+---------+---------+---------+
```

```
temp = result_pair
temp_index = list(temp.index.values)
```

```r
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$'.' <- temp_index
temp <- temp %>%
  select('.', everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

.

A

B

mean(A)

mean(B)

diff

se

T

df

pval

eta-square

0

0

1

5.4957

6.5919

-1.0962

0.2020

-5.4254

432.6136

0.0000

0.0283

1

0

2

5.4957

7.9750

-2.4793

0.3052

-8.1240

89.8256

0.0000

0.1331

2

0

3

5.4957

7.9545

-2.4588

0.4245

-5.7928

45.9074

0.0000

0.1303

3

0

4

5.4957

8.2727

-2.7770

1.0480

-2.6498

10.1065

0.1327

0.1595

4

1

2

6.5919

7.9750

-1.3831

0.3498

-3.9540

149.1447

0.0011

0.0440

5

1

3

6.5919

7.9545

-1.3626

0.4576

-2.9778

61.6737

0.0326

0.0411

6

1

4

6.5919

8.2727

-1.6808

1.0619

-1.5829

10.6510

0.5367

0.0590

7

2

3

7.9750

7.9545

0.0205

0.5116

0.0400

85.2235

1.0000

0.0000

8

2

4

7.9750

8.2727

-0.2977

1.0862

-0.2741

11.6532

0.9986

0.0029

9

3

4

7.9545

8.2727

-0.3182

1.1256

-0.2827

13.3675

0.9984

0.0030

```
result_kw = pg.kruskal(data=data,
                       dv="NumWebPurchases",
                       between="AcceptedCmpTotal")

result_pair = pg.pairwise_gameshowell(data=data,
                                      dv="NumWebPurchases",
                                      between="AcceptedCmpTotal",
                                      effsize="eta-square")

result_kw = round(result_kw, 4)
result_pair = round(result_pair, 4)
```

```
headers = result_kw.columns
result_kw = tabulate(result_kw, headers, tablefmt="grid")
print("NumWebPurchases on AcceptedCmpTotal: \n",result_kw, sep='')
```

```
NumWebPurchases on AcceptedCmpTotal:
+---------+------------------+---------+---------+---------+
|         | Source           |   ddof1 |       H |   p-unc |
+=========+==================+=========+=========+=========+
| Kruskal | AcceptedCmpTotal |       4 | 129.456 |       0 |
+---------+------------------+---------+---------+---------+
```

```
temp = result_pair
temp_index = list(temp.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$'.' <- temp_index
temp <- temp %>%
  select('.', everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

.

A

B

mean(A)

mean(B)

diff

se

T

df

pval

eta-square

0

0

1

3.7839

5.1028

-1.3189

0.1655

-7.9707

442.2072

0.0000

0.0562

1

0

2

3.7839

5.6000

-1.8161

0.2637

-6.8871

89.3485

0.0000

0.1030

2

0

3

3.7839

5.6136

-1.8298

0.3138

-5.8311

46.8719

0.0000

0.1042

3

0

4

3.7839

5.0909

-1.3070

0.8277

-1.5790

10.1225

0.5401

0.0555

4

1

2

5.1028

5.6000

-0.4972

0.2977

-1.6703

140.7103

0.4556

0.0087

5

1

3

5.1028

5.6136

-0.5108

0.3428

-1.4900

66.2461

0.5725

0.0092

6

1

4

5.1028

5.0909

0.0119

0.8392

0.0142

10.6933

1.0000

0.0000

7

2

3

5.6000

5.6136

-0.0136

0.3996

-0.0341

97.7222

1.0000

0.0000

8

2

4

5.6000

5.0909

0.5091

0.8639

0.5893

11.9983

0.9741

0.0117

9

3

4

5.6136

5.0909

0.5227

0.8805

0.5937

12.9040

0.9736

0.0141

```python
result_kw = pg.kruskal(data=data,
                       dv="NumCatalogPurchases",
                       between="AcceptedCmpTotal")

result_pair = pg.pairwise_gameshowell(data=data,
                                      dv="NumCatalogPurchases",
                                      between="AcceptedCmpTotal",
                                      effsize="eta-square")

result_kw = round(result_kw, 4)
result_pair = round(result_pair, 4)

headers = result_kw.columns
result_kw = tabulate(result_kw, headers, tablefmt="grid")
print("NumCatalogPurchases on AcceptedCmpTotal: \n",result_kw, sep='')
```

```
NumCatalogPurchases on AcceptedCmpTotal:
+---------+------------------+---------+---------+---------+
|         | Source           |   ddof1 |       H |   p-unc |
+=========+==================+=========+=========+=========+
| Kruskal | AcceptedCmpTotal |       4 | 303.195 |       0 |
+---------+------------------+---------+---------+---------+
```

```python
temp = result_pair
temp_index = list(temp.index.values)
```

```r
library(kableExtra)
library(dplyr)

# Importing data from python
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$'.' <- temp_index
temp <- temp %>%
  select('.', everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

.

A

B

mean(A)

mean(B)

diff

se

T

df

pval

eta-square

0

0

1

2.2047

3.8754

-1.6707

0.1684

-9.9183

443.9643

0.0000

0.0837

1

0

2

2.2047

5.4125

-3.2078

0.2882

-11.1306

87.9773

0.0000

0.2538

2

0

3

2.2047

6.4318

-4.2271

0.3671

-11.5134

45.9193

0.0000

0.3709

3

0

4

2.2047

6.9091

-4.7044

0.8168

-5.7595

10.1319

0.0013

0.4207

4

1

2

3.8754

5.4125

-1.5371

0.3205

-4.7960

131.5492

0.0000

0.0736

5

1

3

3.8754

6.4318

-2.5564

0.3930

-6.5047

60.0186

0.0000

0.1793

6

1

4

3.8754

6.9091

-3.0337

0.8288

-3.6606

10.7374

0.0257

0.2301

7

2

3

5.4125

6.4318

-1.0193

0.4573

-2.2289

92.2599

0.1784

0.0408

8

2

4

5.4125

6.9091

-1.4966

0.8611

-1.7380

12.4934

0.4474

0.0804

9

3

4

6.4318

6.9091

-0.4773

0.8906

-0.5359

14.1955

0.9820

0.0094

And finally, comparing the number of campaigns accepted with the type of product on which they have spent the most.

```
result_kw = pg.kruskal(data=data,
                       dv="MntSpentTotal",
                       between="AcceptedCmpTotal")

result_pair = pg.pairwise_gameshowell(data=data,
                                      dv="MntSpentTotal",
                                      between="AcceptedCmpTotal",
                                      effsize="eta-square")

result_kw = round(result_kw, 4)
result_pair = round(result_pair, 4)

headers = result_kw.columns
result_kw = tabulate(result_kw, headers, tablefmt="grid")
print("MntSpentTotal on AcceptedCmpTotal: \n",result_kw, sep='')
```

```
MntSpentTotal on AcceptedCmpTotal:
+---------+------------------+---------+---------+---------+
|         | Source           |   ddof1 |       H |   p-unc |
+=========+==================+=========+=========+=========+
| Kruskal | AcceptedCmpTotal |       4 | 353.006 |       0 |
+---------+------------------+---------+---------+---------+
```

```
temp = result_pair
temp_index = list(temp.index.values)
```

```
library(kableExtra)
library(dplyr)

# Importing data from python
```

```r
temp <- py$temp
temp_index = py$temp_index

# Creating Index
temp$'.' <- temp_index
temp <- temp %>%
  select('.', everything())

# Generating Table
temp <- kable(temp, format = "html", row.names = FALSE)
temp <- temp %>%
  kable_styling(full_width = FALSE) %>%
  row_spec(0, bold = TRUE, color = "#ffffff", background = "#232629") %>%
  kable_styling(latex_options = "striped") %>%
  kable_styling(position = "center")

temp
```

.

A

B

mean(A)

mean(B)

diff

se

T

df

pval

eta-square

0

0

1

480.4860

916.9751

-436.4891

38.4823

-11.3426

395.4593

0.0000

0.1418

1

0

2

480.4860

1412.3625

-931.8765

57.8149

-16.1183

86.6062

0.0000

0.4526

2

0

3

480.4860

1705.6364

-1225.1504

73.3475

-16.7034

45.5065

0.0000

0.5888

3

0

4

480.4860

1501.6364

-1021.1504

147.6739

-6.9149

10.1393

0.0003

0.4980

4

1

2

916.9751

1412.3625

-495.3874

67.2520

-7.3661

152.0597

0.0000

0.1350

5

1

3

916.9751

1705.6364

-788.6613

80.9946

-9.7372

67.0815

0.0000

0.2780

6

1

4

916.9751

1501.6364

-584.6613

151.6175

-3.8562

11.2652

0.0174

0.1686

7

2

3

1412.3625

1705.6364

-293.2739

91.7703

-3.1957

92.7150

0.0160

0.0802

8

2

4

1412.3625

1501.6364

-89.2739

157.6373

-0.5663

13.1291

0.9778

0.0078

9

3

4

1705.6364

1501.6364

204.0000

163.9720

1.2441

15.2062

0.7270

0.0430

```python
cols = ["MntWines", "MntFruits", "MntMeatProducts",
        "MntFishProducts", "MntSweetProducts", "MntGoldProds"]

for col in cols:
    result_kw = pg.kruskal(data=data,
                           dv=col,
                           between="AcceptedCmpTotal")

    result_pair = pg.pairwise_gameshowell(data=data,
                                          dv="NumCatalogPurchases",
                                          between="AcceptedCmpTotal",
                                          effsize="eta-square")

    result_kw = round(result_kw, 4)
    result_pair = round(result_pair, 4)
```

```python
    print(f"\nComparisons for 'AcceptedCmpTotal' on {col}:")
    print("\nH Kruskal Wallis:\n")

    headers = result_kw.columns
    result_kw = tabulate(result_kw, headers, tablefmt="grid")
    print(result_kw)

    print("\nGames Howell Pair Comparisons:\n")

    headers = result_pair.columns
    result_pair = tabulate(result_pair, headers, tablefmt="grid")
    print(result_pair)
```

```
Comparisons for 'AcceptedCmpTotal' on MntWines:

H Kruskal Wallis:

+---------+------------------+---------+---------+---------+
|         | Source           |   ddof1 |       H |   p-unc |
+=========+==================+=========+=========+=========+
| Kruskal | AcceptedCmpTotal |       4 | 389.468 |       0 |
+---------+------------------+---------+---------+---------+

Games Howell Pair Comparisons:

+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|    |   A |   B |   mean(A) |   mean(B) |    diff |     se |        T |       df |   pval |   eta-squa
+====+=====+=====+===========+===========+=========+========+==========+==========+========+============
|  0 |   0 |   1 |    2.2047 |    3.8754 | -1.6707 | 0.1684 |  -9.9183 | 443.964  | 0      |        0.083
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  1 |   0 |   2 |    2.2047 |    5.4125 | -3.2078 | 0.2882 | -11.1306 |  87.9773 | 0      |        0.253
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  2 |   0 |   3 |    2.2047 |    6.4318 | -4.2271 | 0.3671 | -11.5134 |  45.9193 | 0      |        0.370
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  3 |   0 |   4 |    2.2047 |    6.9091 | -4.7044 | 0.8168 |  -5.7595 |  10.1319 | 0.0013 |        0.420
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  4 |   1 |   2 |    3.8754 |    5.4125 | -1.5371 | 0.3205 |  -4.796  | 131.549  | 0      |        0.073
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  5 |   1 |   3 |    3.8754 |    6.4318 | -2.5564 | 0.393  |  -6.5047 |  60.0186 | 0      |        0.179
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  6 |   1 |   4 |    3.8754 |    6.9091 | -3.0337 | 0.8288 |  -3.6606 |  10.7374 | 0.0257 |        0.230
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  7 |   2 |   3 |    5.4125 |    6.4318 | -1.0193 | 0.4573 |  -2.2289 |  92.2599 | 0.1784 |        0.040
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  8 |   2 |   4 |    5.4125 |    6.9091 | -1.4966 | 0.8611 |  -1.738  |  12.4934 | 0.4474 |        0.080
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------
|  9 |   3 |   4 |    6.4318 |    6.9091 | -0.4773 | 0.8906 |  -0.5359 |  14.1955 | 0.982  |        0.009
+----+-----+-----+-----------+-----------+---------+--------+----------+----------+--------+------------

Comparisons for 'AcceptedCmpTotal' on MntFruits:

H Kruskal Wallis:
```

	Source	ddof1	H	p-unc
Kruskal	AcceptedCmpTotal	4	62.9684	0

Games Howell Pair Comparisons:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	eta-squa
0	0	1	2.2047	3.8754	-1.6707	0.1684	-9.9183	443.964	0	0.083
1	0	2	2.2047	5.4125	-3.2078	0.2882	-11.1306	87.9773	0	0.253
2	0	3	2.2047	6.4318	-4.2271	0.3671	-11.5134	45.9193	0	0.370
3	0	4	2.2047	6.9091	-4.7044	0.8168	-5.7595	10.1319	0.0013	0.420
4	1	2	3.8754	5.4125	-1.5371	0.3205	-4.796	131.549	0	0.073
5	1	3	3.8754	6.4318	-2.5564	0.393	-6.5047	60.0186	0	0.179
6	1	4	3.8754	6.9091	-3.0337	0.8288	-3.6606	10.7374	0.0257	0.230
7	2	3	5.4125	6.4318	-1.0193	0.4573	-2.2289	92.2599	0.1784	0.040
8	2	4	5.4125	6.9091	-1.4966	0.8611	-1.738	12.4934	0.4474	0.080
9	3	4	6.4318	6.9091	-0.4773	0.8906	-0.5359	14.1955	0.982	0.009

Comparisons for 'AcceptedCmpTotal' on MntMeatProducts:

H Kruskal Wallis:

	Source	ddof1	H	p-unc
Kruskal	AcceptedCmpTotal	4	200.496	0

Games Howell Pair Comparisons:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	eta-squa
0	0	1	2.2047	3.8754	-1.6707	0.1684	-9.9183	443.964	0	0.083
1	0	2	2.2047	5.4125	-3.2078	0.2882	-11.1306	87.9773	0	0.253
2	0	3	2.2047	6.4318	-4.2271	0.3671	-11.5134	45.9193	0	0.370

3	0	4	2.2047	6.9091	-4.7044	0.8168	-5.7595	10.1319	0.0013	0.42
4	1	2	3.8754	5.4125	-1.5371	0.3205	-4.796	131.549	0	0.07
5	1	3	3.8754	6.4318	-2.5564	0.393	-6.5047	60.0186	0	0.17
6	1	4	3.8754	6.9091	-3.0337	0.8288	-3.6606	10.7374	0.0257	0.23
7	2	3	5.4125	6.4318	-1.0193	0.4573	-2.2289	92.2599	0.1784	0.04
8	2	4	5.4125	6.9091	-1.4966	0.8611	-1.738	12.4934	0.4474	0.08
9	3	4	6.4318	6.9091	-0.4773	0.8906	-0.5359	14.1955	0.982	0.00

Comparisons for 'AcceptedCmpTotal' on MntFishProducts:

H Kruskal Wallis:

	Source	ddof1	H	p-unc
Kruskal	AcceptedCmpTotal	4	50.0565	0

Games Howell Pair Comparisons:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	eta-squa
0	0	1	2.2047	3.8754	-1.6707	0.1684	-9.9183	443.964	0	0.08
1	0	2	2.2047	5.4125	-3.2078	0.2882	-11.1306	87.9773	0	0.25
2	0	3	2.2047	6.4318	-4.2271	0.3671	-11.5134	45.9193	0	0.37
3	0	4	2.2047	6.9091	-4.7044	0.8168	-5.7595	10.1319	0.0013	0.42
4	1	2	3.8754	5.4125	-1.5371	0.3205	-4.796	131.549	0	0.07
5	1	3	3.8754	6.4318	-2.5564	0.393	-6.5047	60.0186	0	0.17
6	1	4	3.8754	6.9091	-3.0337	0.8288	-3.6606	10.7374	0.0257	0.23
7	2	3	5.4125	6.4318	-1.0193	0.4573	-2.2289	92.2599	0.1784	0.04
8	2	4	5.4125	6.9091	-1.4966	0.8611	-1.738	12.4934	0.4474	0.08
9	3	4	6.4318	6.9091	-0.4773	0.8906	-0.5359	14.1955	0.982	0.00

Comparisons for 'AcceptedCmpTotal' on MntSweetProducts:

H Kruskal Wallis:

	Source	ddof1	H	p-unc
Kruskal	AcceptedCmpTotal	4	68.83	0

Games Howell Pair Comparisons:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	eta-squa
0	0	1	2.2047	3.8754	-1.6707	0.1684	-9.9183	443.964	0	0.083
1	0	2	2.2047	5.4125	-3.2078	0.2882	-11.1306	87.9773	0	0.253
2	0	3	2.2047	6.4318	-4.2271	0.3671	-11.5134	45.9193	0	0.370
3	0	4	2.2047	6.9091	-4.7044	0.8168	-5.7595	10.1319	0.0013	0.420
4	1	2	3.8754	5.4125	-1.5371	0.3205	-4.796	131.549	0	0.073
5	1	3	3.8754	6.4318	-2.5564	0.393	-6.5047	60.0186	0	0.179
6	1	4	3.8754	6.9091	-3.0337	0.8288	-3.6606	10.7374	0.0257	0.230
7	2	3	5.4125	6.4318	-1.0193	0.4573	-2.2289	92.2599	0.1784	0.040
8	2	4	5.4125	6.9091	-1.4966	0.8611	-1.738	12.4934	0.4474	0.080
9	3	4	6.4318	6.9091	-0.4773	0.8906	-0.5359	14.1955	0.982	0.009

Comparisons for 'AcceptedCmpTotal' on MntGoldProds:

H Kruskal Wallis:

	Source	ddof1	H	p-unc
Kruskal	AcceptedCmpTotal	4	124.573	0

Games Howell Pair Comparisons:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	eta-squa
0	0	1	2.2047	3.8754	-1.6707	0.1684	-9.9183	443.964	0	0.083
1	0	2	2.2047	5.4125	-3.2078	0.2882	-11.1306	87.9773	0	0.253
2	0	3	2.2047	6.4318	-4.2271	0.3671	-11.5134	45.9193	0	0.370

3	0	4	2.2047	6.9091	-4.7044	0.8168	-5.7595	10.1319	0.0013	0.420
4	1	2	3.8754	5.4125	-1.5371	0.3205	-4.796	131.549	0	0.073
5	1	3	3.8754	6.4318	-2.5564	0.393	-6.5047	60.0186	0	0.179
6	1	4	3.8754	6.9091	-3.0337	0.8288	-3.6606	10.7374	0.0257	0.230
7	2	3	5.4125	6.4318	-1.0193	0.4573	-2.2289	92.2599	0.1784	0.040
8	2	4	5.4125	6.9091	-1.4966	0.8611	-1.738	12.4934	0.4474	0.080
9	3	4	6.4318	6.9091	-0.4773	0.8906	-0.5359	14.1955	0.982	0.009

4 Conclusions

Placeholder

References

Jacob Cohen. A power primer. *Psychological Bulletin*, 112(1):155–159, 1992. doi: 10.1037/0033-2909.112.1.155. URL http://dx.doi.org/10.1037/0033-2909.112.1.155.

Hugh Coolican and Hugh Coolican. *Research Methods and Statistics in Psychology*. Routledge, 03 2013. doi: 10.4324/9780203769669. URL http://dx.doi.org/10.4324/9780203769669.

Christopher J. Ferguson. An effect size primer: A guide for clinicians and researchers. *Professional Psychology: Research and Practice*, 40(5):532–538, 10 2009. doi: 10.1037/a0015808. URL http://dx.doi.org/10.1037/a0015808.

Hans W. Wendt. Dealing with a common problem in social science: A simplified rank-biserial coefficient of correlation based on the u statistic. *European Journal of Social Psychology*, 2(4):463–465, 10 1972. doi: 10.1002/ejsp.2420020412. URL http://dx.doi.org/10.1002/ejsp.2420020412.