Regression Project (Fabio Costantino)

My role in the Company

As a Content Analyst in the Localization Team, I am responsible for migrating data in and out of our Content Management System called STEP. These are mostly text data - a.k.a. content, which are then published daily in our website for all customers to consume.

As a Localization team, we mainly deal with translated text in 12 different languages, therefore also managing the flow of translation files in and out of the company. In order to translate our content, I have daily contact with external translation agencies and manage the use of a Translation Management System called XTM, which helps organize all files and workflows in a single tool.

Besides, we also provide our manager(s) and the rest of the team with reports of many kind: from tailored collection of data coming from different sources, to reports on products (or families of products) performance.

Some of the tools I use on a daily/weekly basis to perform my job are: **STEP Content Management System**, **XTM Translation Management System**, **Microsoft SQL Server Management Studio**, **Report Builder** from **Adobe Analytics**, **DOMO** as a BI dashboard tool, **Office package**, **Jupyter Notebook** (with **Python**).

Brief

Define additional key performance indicator(s) to monitor results in the Italian market

Business Problem

The Localization team supports the local markets in creating action plans to enrich the content of the website. In these action plans, the team tries to define a workable list of products that should be revised by the local **Content Enrichment Executive** (CEE for short) for the upcoming quarter or - if the list is long enough - for the upcoming fiscal year.

At the time of writing, the catalogue included over 1 million products, hence the need of intelligently reducing the scope of the task to a few hundreds by spotting those products that might be more relevant to the business. This allows also to optimze the effort of a limited number of CEEs, otherwise diluted to enrich product that will never bring much value to the business.

Current Process

Currently, the **Content plans** for specific markets look predominantly at Key Performance Indicator like **revenue**, **conversion rate** or **page views**. The products are then filtered, considering e.g. products with high revenue but low conversion rate - on the assumption that if the product converted even more, the revenue will be much higher.

Current issues

All the KPIs mentioned above don't necessarily focus on the content specifically, but rather try to capture a bigger picture. However, a customer might be very interested in the product because of how it is presented (i.e. for its content), she might decide to buy it but then encounter some problems at the check-out and leave the cart. The final decision of not buying a particular product will therefore not capture the customer's positive reception of the content but rather some other issue that is not in the hands of the Content or Localization teams.

Proposed solution

I am tasked with finding additional KPIs that can better reflect or help capture how customers respond to the content in a product page. I will proceed as follows:

- 1) I will explore options in Adobe Analytics to find KPIs that can be connected with the content directly.
- 2) I will import the necessary data in Jupyter Notebook for analysis.
- 3) I will perform a linear regression on the variables that seem more directly suggesting interesting results.

Pros and Cons

This experimental approach should give me a solid base on which to take more informed decisions.

On the other hand, the data at product level tend to be very sparse in our catalogue, with many products not getting many views or not converting very much. This might cause problems to my analysis.

Assumptions

1) With *additional KPI(s)*, I mean those indicators that need to be monitored along with the more general KPI(s) defined for the strategy of the department as a whole. For example, if the whole company uses **revenue** as a KPI, the Localization team might be interested in looking at e.g. the number of additions to cart or orders made, because these metrics are more directly related to the quality of the text.

2) The time period selected in Adobe Analytics covers **12 months** (from the 1st of January 2019 until the 31st of December 2019), assuming that this will give a good picture of the market in exam.

0. Importing libraries

First of all I need to import all relevant libraries to this analysis.

In [1]:
```python
# To work with dataframe and manipulate dataset
import pandas as pd

# To work in pandas with matrices and to perform calculations
import numpy as np

# To visualize data, create plots and evaluating visually the predictions
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.api import abline_plot

# To scale the dataset and prepare it for clustering
from sklearn.preprocessing import scale

# To perform cluster analysis
from sklearn.cluster import KMeans

# To perform silhouette analysis
from sklearn.metrics import silhouette_score

# To perform statistical tests
from scipy import stats

# To perform Levene test for homoscedasticity
from scipy.stats import levene

# To split the dataset between training and testing
from sklearn.model_selection import train_test_split

# To perform linear regressions
from sklearn import linear_model, preprocessing
import statsmodels.api as sm

# To evaluate models
from sklearn import metrics

# To suppress warnings
import warnings; warnings.simplefilter('ignore')
```

1. Sourcing data

Since importing data for all products in the catalogue would be too computational expensive, I decided to analyze first a section of about 150,000 products covering a category called **IT, Test & Safety Equipment**. This section includes products like *3D printing peripherals*, *accessories for the office and for telecommunications*, *software*, *wireless components*, *test and measurement tools*, etc.

As per point **1** in paragraph **Proposed solution**, I explored different options in Adobe Analytics and found the following 4 metrics to be added to my dataset and analyzed:

- **Cart additions** - the number of times a product has been added to the cart
- **Page views** - the number of times a page has been visited for the time period selected
- **Unique visitors** - the number of new visitors that have viewed a page in a specific period of time
- **Orders** - the number of unique orders in which a certain page features

As previously stated (paragraph ***Assumptions***), 12 months worth of data has been collected and merged with the list of products.

Among the variable chose, I decided to chose **orders** as my independent variable, since it is technically the closest to the revenue metrics used in the rest of the company.

In [2]:
```
# Loading the dataset
dataset = pd.read_excel('data_IT.xlsx', sheetname='Sheet3')
```

In [3]:
```
# Visualizing the last few rows of the dataset
dataset.head()
```

Out[3]:

	Product_ID	Cart_additions	Page_views	Unique_visitors	Orders
0	0489412	[redacted]	[redacted]	[redacted]	[redacted]
1	0489573	[redacted]	[redacted]	[redacted]	[redacted]
2	1213230	[redacted]	[redacted]	[redacted]	[redacted]
3	1213232	[redacted]	[redacted]	[redacted]	[redacted]
4	1213234	[redacted]	[redacted]	[redacted]	[redacted]

2. Exploring the dataset

I will now explore the dataset with the aim of cleaning it and prepare it for the next phase of the project.

2.1 Missing values and data type

In [4]:
```
# Gathering general information about observations and null values
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158283 entries, 0 to 158282
Data columns (total 5 columns):
Product_ID         158283 non-null object
Cart_additions     158283 non-null int64
Page_views         158283 non-null int64
Unique_visitors    158283 non-null int64
Orders             158283 non-null int64
dtypes: int64(4), object(1)
memory usage: 6.0+ MB
```

The dataset seem not to contain null values.

Another information I would like to visualize is the statistical description of the dataset. I will do so using the `describe()` function.

In [5]:
```
# For the visualization, I drop the first column since it only stores product IDs
dataset.drop(columns='Product_ID', axis=1).describe()
```

Out[5]:

	Cart_additions	Page_views	Unique_visitors	Orders
count	[redacted]	[redacted]	[redacted]	[redacted]
mean	[redacted]	[redacted]	[redacted]	[redacted]
std	[redacted]	[redacted]	[redacted]	[redacted]
min	[redacted]	[redacted]	[redacted]	[redacted]
25%	[redacted]	[redacted]	[redacted]	[redacted]
50%	[redacted]	[redacted]	[redacted]	[redacted]
75%	[redacted]	[redacted]	[redacted]	[redacted]
max	[redacted]	[redacted]	[redacted]	[redacted]

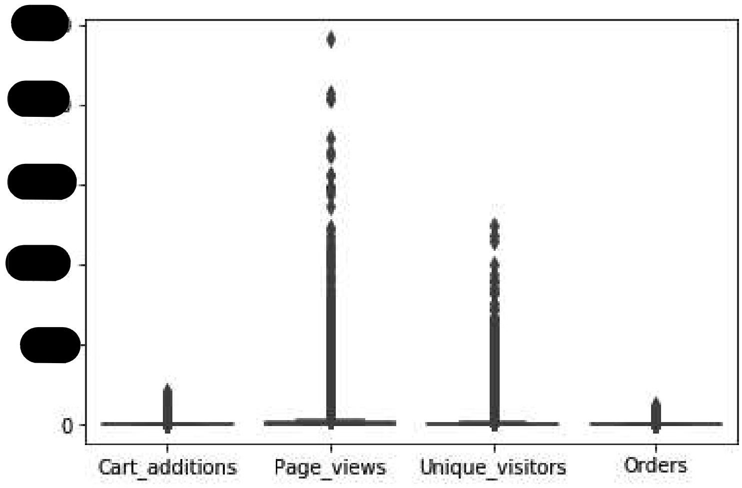
These results clearly show that I have to deal with outliers.

2.2 Visualizing distributions and relations

Before applying any other transformation to the dataset, I will explore visually the distributions of the observations and their relations to each other.

In [6]:
```
# Visualizing the data with boxplots
sns.boxplot(data=dataset.iloc[:,[1,2,3,4]])
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4c7bdef0>
```

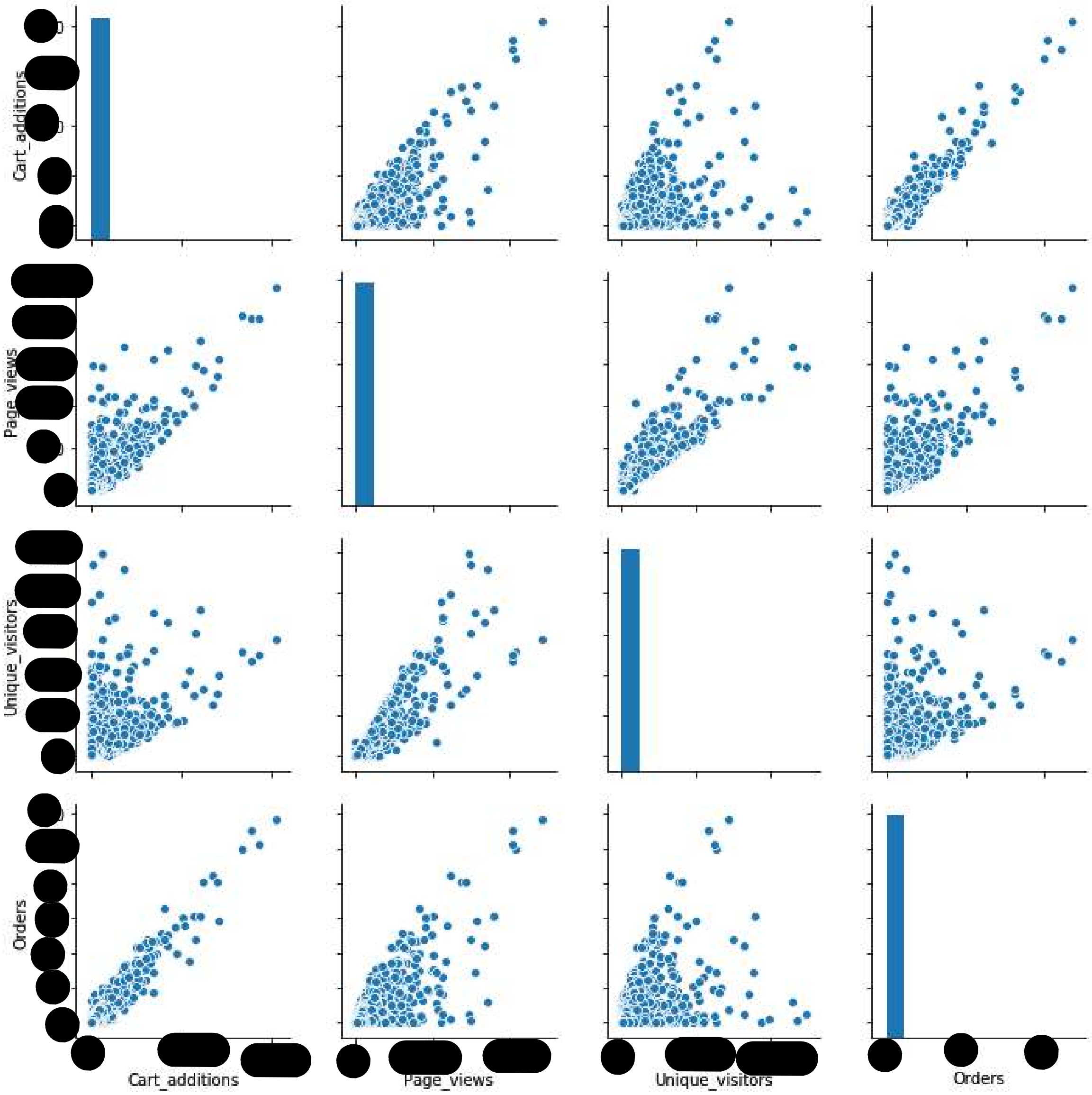


It is quite clear that I will have to treat outliers to avoid having my data skewed during analysis. However, before doing that, I want also to explore the relations between variables.

In [7]:

```
# Plotting all combinations of the columns in the dataset
sns.pairplot(data=dataset)
```

Out[7]: <seaborn.axisgrid.PairGrid at 0x24c5151bfd0>



In [8]:
```
# Setting the figure to a readable size and creating the heatmap
# The creation of the heatmap uses the function corr() that shows a matrix of
 correlation gradients
# for all possible combinations of columns

plt.figure(figsize=(20,10))
sns.heatmap(dataset.corr(), annot=True)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x24c4e950668>

Looking at these two visualization, I can see that the column **Unique_visitors** has the lowest correlation with **Orders** (my independent variable as previously stated). Furthermore, from the plotting above it is clear that the data are very spread out and heteroscedastic. The only homoscedastic relation seems to be with **Page_views**, which is to be expected.

Because of this - and after much experimentation, I decided to eliminate the column **Unique_visitors** from my dataset.

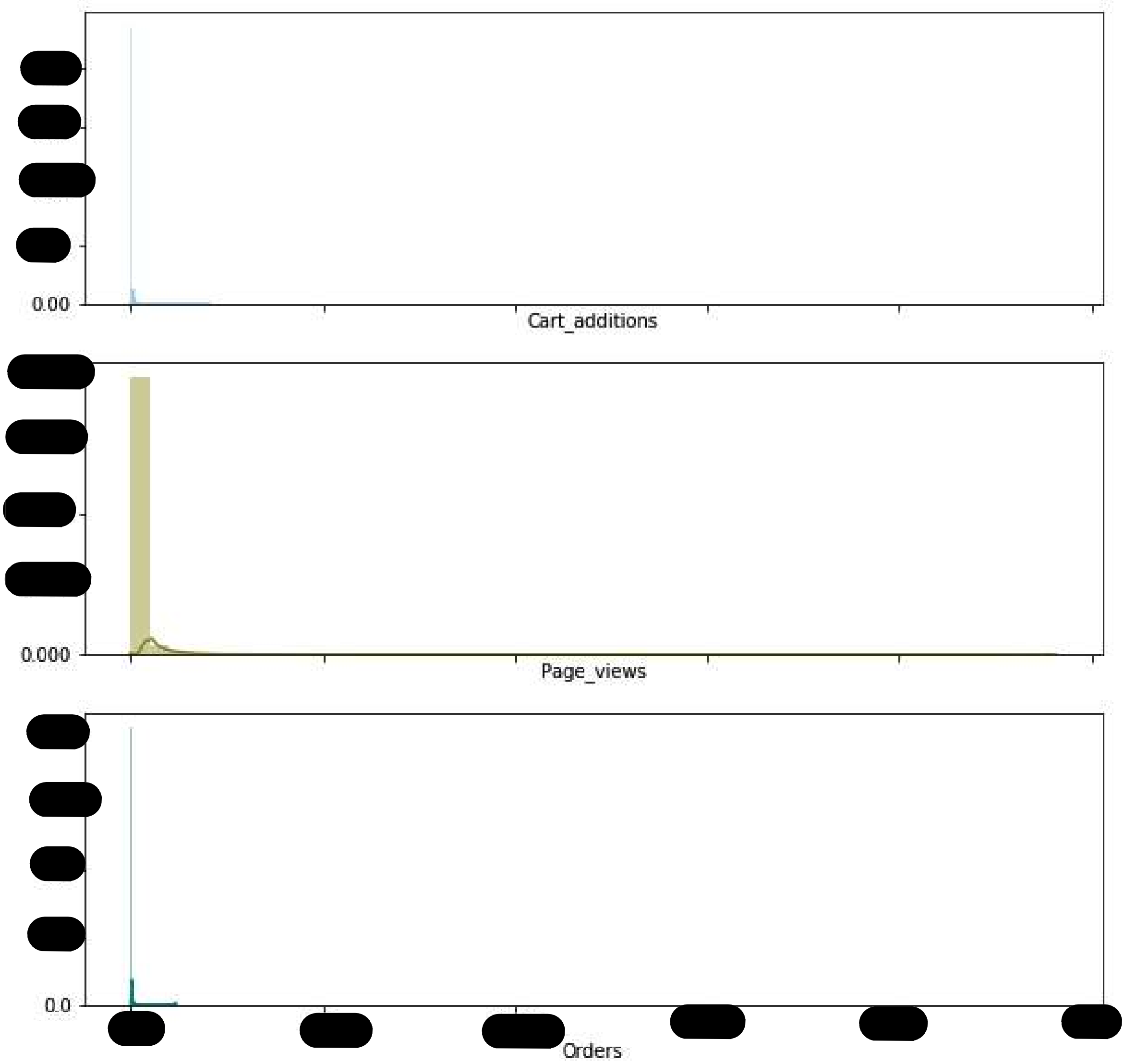
In [9]:
```
# Dropping the column
dataset = dataset.drop(columns='Unique_visitors')
```

In [10]:

```python
# Creating a quick visualization of the distributions for each column left
f, axes = plt.subplots(3, 1, figsize=(10, 10), sharex=True)
sns.distplot(dataset['Cart_additions'] , color="skyblue", ax=axes[0])
sns.distplot(dataset['Page_views'] , color="olive", ax=axes[1])
sns.distplot(dataset['Orders'] , color="teal", ax=axes[2])
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4dea07b8>
```



2.3 Treating outliers

From the visualization and from the `describe()` function in **2.1**, I can see that many products don't have useful information since they are just a series of `0` 's. This problem was already identified in the paragraph **Pros and Cons** and I start to see that it might negatively impact the rest of my analysis.

I decided anyway to clean the dataset from these observations and carry on with my project. I will drop all the lines that contain only `0` 's in all columns at the same time.

In [11]:

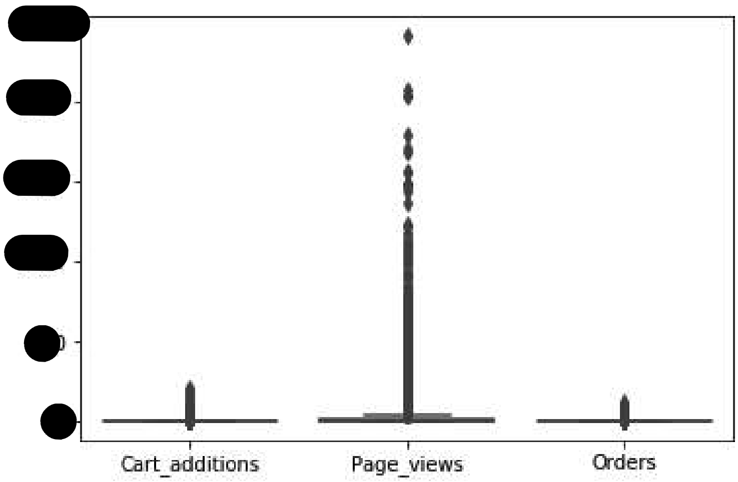
```
# First I create a subset of rows with '0' only
all_zero_columns = dataset[(dataset['Cart_additions'] == 0)
                           & (dataset['Page_views'] == 0)
                           & (dataset['Orders'] == 0)].index

# Then, I drop the previous subset from the original dataset
dataset.drop(all_zero_columns, inplace=True)
```

In [12]:

```
sns.boxplot(data=dataset.iloc[:,[1,2,3]])
```

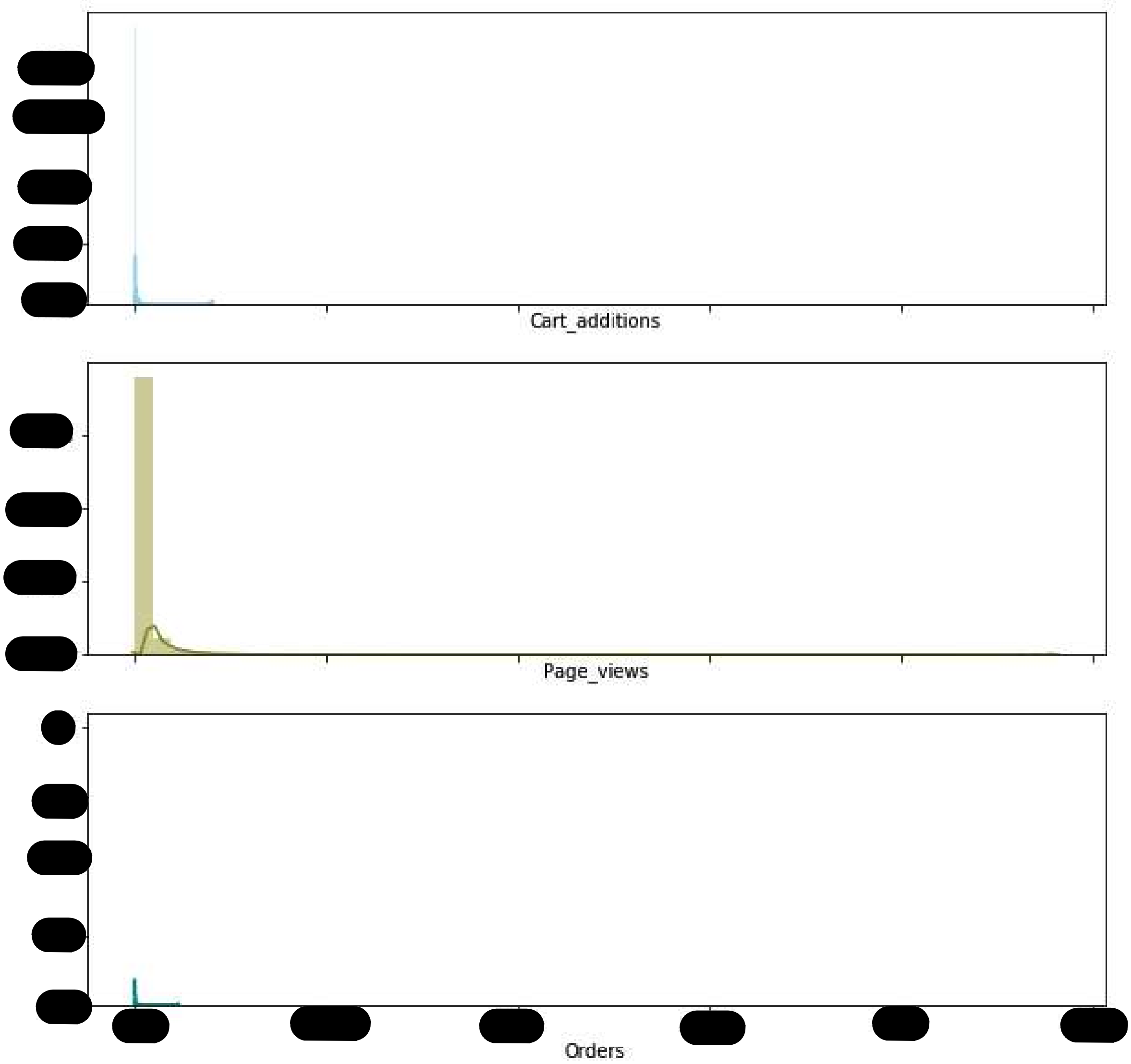
Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4d96d9b0>
```



In [13]:
```
# Creating a quick visualization of the distribution for each column
f, axes = plt.subplots(3, 1, figsize=(10, 10), sharex=True)
sns.distplot(dataset['Cart_additions'] , color="skyblue", ax=axes[0])
sns.distplot(dataset['Page_views'] , color="olive", ax=axes[1])
sns.distplot(dataset['Orders'] , color="teal", ax=axes[2])
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4c5c5e80>
```



As it is visually clear with these graphics, the data is heavily skewed because of outliers, which will result in an unsuccessful regression analysis. I decided to **cluster** the dataset and choose the best cluster to analyze further.

What I hope to find is a subsection of the dataset in which a linear regression can successfully be performed and a line of best fit created.

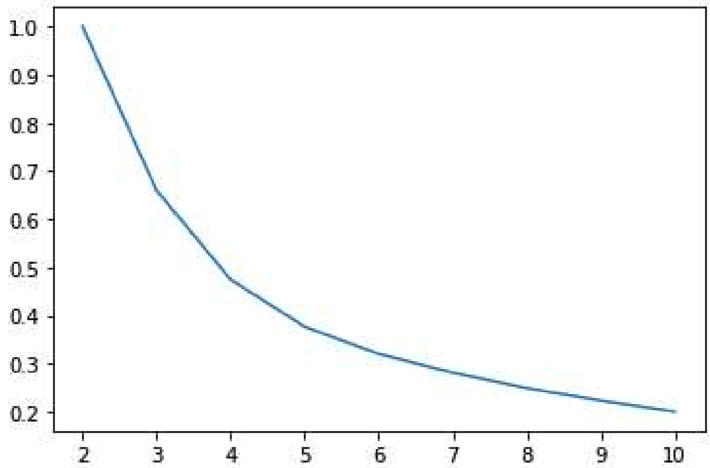
In [14]:
```
# First I create an additional dataset with only numeric columns
clustering_dataset = dataset.drop(columns=['Product_ID'], axis=1)

# Scaling the dataset just created, to prepare it for clustering
scaled_clustering_dataset = pd.DataFrame(scale(clustering_dataset))
```

In [15]:
```
# Creating iteration of the clustering model, to understand how many clusters
# might be appropriate to use.
cluster_nums = [2,3,4,5,6,7,8,9,10]

models = [KMeans(n_clusters=i, random_state=81) for i in cluster_nums]
```

In [16]:
```
# Exploring the scores to find the best cluster
scores = [models[j].fit(scaled_clustering_dataset).inertia_ for j in range(len
(models))]

score_normalised = scores/scores[0]
```

In [17]:
```
# Now I will plot the results of the scores to perform a simple elbow test
sns.lineplot(cluster_nums, score_normalised)
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4b948eb8>
```



From this graph, it looks like 5 or 6 might be suitable candidates for my clustering. I will now perform a **silhouette analysis** to make a more informed decision about it. Silhouette analysis gives me a score from `-1` to `+1` for each number of clusters that I have considered in `cluster_nums`. A score closer to `+1` indicates that the cluster fits the dataset better, whereas a score closer to `-1` indicates the contrary.

In [18]:
```python
# Calculating the silhouette score
for n_clusters in cluster_nums:
    clusterer = KMeans(n_clusters=n_clusters, random_state=81)
    preds = clusterer.fit_predict(scaled_clustering_dataset)
    centers = clusterer.cluster_centers_

    score = silhouette_score (scaled_clustering_dataset, preds, metric='euclidean')
    print ("For n_clusters = {}, silhouette score is {}".format(n_clusters, score))
```

```
For n_clusters = 2, silhouette score is 0.9082282718307136
For n_clusters = 3, silhouette score is 0.8321158084435679
For n_clusters = 4, silhouette score is 0.7972464886163589
For n_clusters = 5, silhouette score is 0.7301633263050697
For n_clusters = 6, silhouette score is 0.6987962320317206
For n_clusters = 7, silhouette score is 0.657400303102177
For n_clusters = 8, silhouette score is 0.6530930904681952
For n_clusters = 9, silhouette score is 0.6337356009731017
For n_clusters = 10, silhouette score is 0.6241471017853385
```

After a long series of trial-and-error's (based on the statistical tests for *normal distribution* that will follow), I chose to use `n_clusters = 5` to continue my analysis.

In [19]:
```python
# Clustering the dataset with the KMeans algorithm with 5 clusters
model = KMeans(n_clusters=5, random_state=81)
model.fit(scaled_clustering_dataset)
```

Out[19]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=81, tol=0.0001, verbose=0)
```

In [20]:
```python
# Creating a new column with the cluster numbers
model.labels_
dataset['Cluster_column'] = model.labels_
```

In [21]:
```python
# Checking that the dataset correctly contains the cluster number column
dataset.head()
```

Out[21]:

	Product_ID	Cart_additions	Page_views	Orders	Cluster_column
2	1213230	[redacted]	[redacted]	[redacted]	1
3	1213232	[redacted]	[redacted]	[redacted]	0
4	1213234	[redacted]	[redacted]	[redacted]	1
5	1213239	[redacted]	[redacted]	[redacted]	1
6	1213240	[redacted]	[redacted]	[redacted]	1

Among all clusters retrieved from this brief test, I found out that **cluster number 3** is normally distributed and homoskedastic - two pre-requisite to successfully perform a linear regression, as it is going to be proved by the following code.

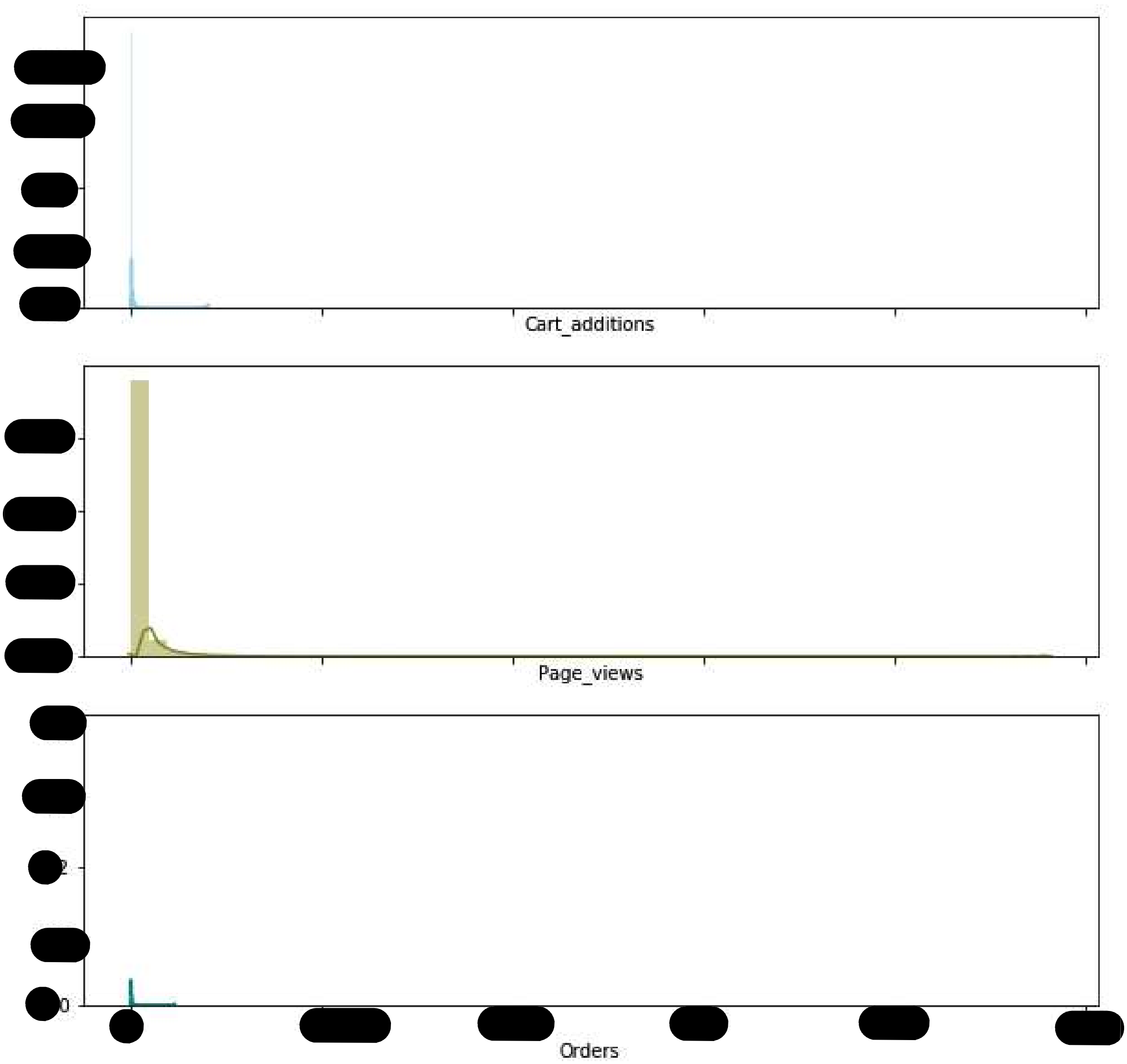
In [22]:
```
# Isolating Cluster 3
dataset_cluster = dataset[dataset['Cluster_column']==4]
dataset_cluster.head()
```

Out[22]:

	Product_ID	Cart_additions	Page_views	Orders	Cluster_column
2568	7004539	[redacted]	[redacted]	[redacted]	4
5666	6698319	[redacted]	[redacted]	[redacted]	4
7842	6151154	[redacted]	[redacted]	[redacted]	4
11520	6642887	[redacted]	[redacted]	[redacted]	4
12241	7004535	[redacted]	[redacted]	[redacted]	4

In [23]:
```
# Creating a quick visualization of the distribution for the cluster in exam
f, axes = plt.subplots(3, 1, figsize=(10, 10), sharex=True)
sns.distplot(dataset['Cart_additions'] , color="skyblue", ax=axes[0])
sns.distplot(dataset['Page_views'] , color="olive", ax=axes[1])
sns.distplot(dataset['Orders'] , color="teal", ax=axes[2])
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4cd98780>
```



One of the conditions to perform regression analysis is that the data be normally distributed. To check if my data meet this condition, I will use the `normaltest()` from `stats` in the `scipy` library.

The H_0 for this test is that the data are **normally distributed**.

In [24]:
```
# Performing a normality test on each column of the dataset
normality_cartadd = stats.normaltest(dataset_cluster['Cart_additions'])
normality_views = stats.normaltest(dataset_cluster['Page_views'])
normality_orders = stats.normaltest(dataset_cluster['Orders'])
```

In [25]:
```
# Storing the results of the normality tests in a new dataframe
pvalues = {'Columns':['Cart_additions','Page_views', 'Orders'],
          'pvalues': [normality_cartadd.pvalue, normality_views.pvalue,
                      normality_orders.pvalue]}

# Assigning a name to the new data frame, that I will use to store all other s
tatistical results
statistical_tests = pd.DataFrame(data=pvalues)
```

In [26]:
```
# Creating a function to check all the p-values obtained with the normality te
sts
def check_pvalue(i):
    for i in statistical_tests.iloc[:,1]:
        if i <= 0.05:
            val = 'Non-normally distributed' # null hypothesis is rejected
        else:
            val = 'Normally distributed' # null hypothesis is accepted
    return val
```

In [27]:
```
# Creating a new column in the dataframe 'statistical_tests' to store the resu
lts of the function check_pvalue()
statistical_tests['Distribution'] = statistical_tests.apply(check_pvalue, axis
=1)

# Showing the results
statistical_tests
```

Out[27]:

	Columns	pvalues	Distribution
0	Cart_additions	0.185426	Normally distributed
1	Page_views	0.403229	Normally distributed
2	Orders	0.300614	Normally distributed

Another condition for regression analysis is that the data plotted against the independent variable are homoscedastic. To check my data for this condition, I will use the `levene()` test from `stats` in the `scipy` library.

The H_0 for this test is that the data are **homoscedastic**.

In [50]:
```
# Performing the Levene test on each column of the dataset
# This test plots the independent variable 'Orders' against all dependent vari
ables
levene_cartadd = stats.levene(dataset_cluster['Orders'], dataset_cluster['Cart
_additions'])
levene_views = stats.levene(dataset_cluster['Orders'], dataset_cluster['Page_v
iews'])


# Storing the results of the Levene tests in a new dataframe
scedasticity = {levene_cartadd, levene_views}


# Creating a function that checks the p-value of the Levene test, to accept or
reject the null hypothesis
def check_scedasticity(i):
    for i in scedasticity:
        if i.pvalue < 0.05:
            val = 'Heteroscedastic data' # null hypothesis is rejected
        elif i.pvalue >= 0.05:
            val = 'Homoscedastic data' # null hypothesis is accepted
        return val


# Showing only the results without visualizing the p-values
statistical_tests['Scedasticity'] = statistical_tests.apply(check_scedasticity
, axis=1)
statistical_tests[['Columns', 'Distribution', 'Scedasticity']]
```

Out[50]:

	Columns	Distribution	Scedasticity
0	Cart_additions	Normally distributed	Homoscedastic data
1	Page_views	Normally distributed	Homoscedastic data
2	Orders	Normally distributed	Homoscedastic data

In [29]:
```
# Checking my final dataset
dataset_cluster.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 22 entries, 2568 to 156560
Data columns (total 5 columns):
Product_ID        22 non-null object
Cart_additions    22 non-null int64
Page_views        22 non-null int64
Orders            22 non-null int64
Cluster_column    22 non-null int32
dtypes: int32(1), int64(3), object(1)
memory usage: 968.0+ bytes
```

Although we are left with a small number of observations, I am confident that these can now be passed through the regression model and give us results that are not skewed.

3. Modelling

3.1 Preparing the dataset for the model

Before applying the model to the dataset, I need to split it in test and training subsets. This split will allow me to run the model on the training subset and to check the results on the test one.

I will use the `train_test_split()` function to achieve this.

In [30]:
```
# Create test and train datasets
X = dataset_cluster[['Cart_additions', 'Page_views']]
y = dataset_cluster['Orders']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, ra
ndom_state = 123)
```

In [31]:
```
# Printing the shape of the 4 subsets help us to see that the split has been
# correctly carried out
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(16, 2) (16,)
(6, 2) (6,)
```

In [32]:
```
# Storing the linear regression model in a function I can call on the train se
ts
model = linear_model.LinearRegression()
```

In [33]:
```
# Calling the model function on the train subsets
model.fit(X_train, y_train)
```

Out[33]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

3.2 Evaluating the model

Firstly, I want to evaluate how much variation does the model explain. This value is captured by the R_2 coefficient, which can be calculated with the `score()` function as follows.

In [34]:
```
# Printing the r-squared coefficient for the model
print('The model explains {}% of the variations.'.format(model.score(X_train,
y_train)*100))
```

```
The model explains 81.77737157549161% of the variations.
```

Another result I can check is the *y-intercept*, also called *coefficient*. This coefficient tells us how many more orders do we get for an increase of 1 in each of the 2 metrics we are examining.

In [35]:
```
# Printing the results
results = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])
results
```

Out[35]:

	Coefficient
Cart_additions	0.673145
Page_views	-0.012340

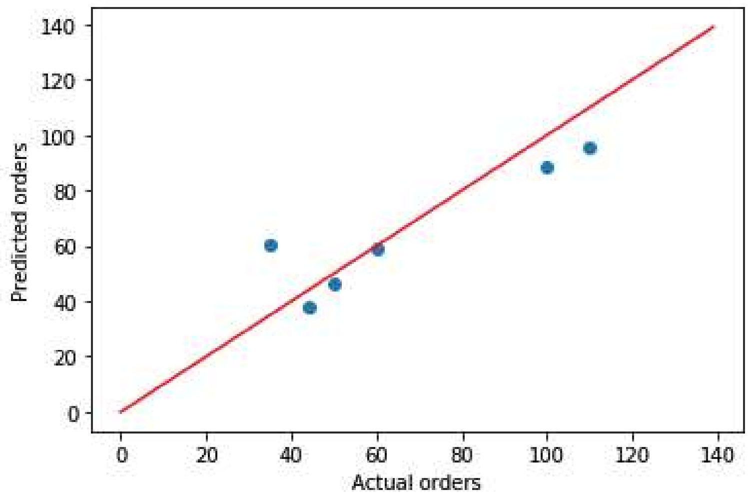
Interestingly enough, we can see that the coefficient for **page views** suggests a negative correlation. Although this is a very slight decrease, it still tells us that page views might not be the best predictor for orders.

So far, we have been evaluating the training subset. Now I will make predictions on the testing subset. Also, I will provide a visualization of the results with a line of best fit to make the results more clear.

In [36]:
```
# Making prediction based on the test set
y_pred = model.predict(X_test)
```

In [37]:
```
# Plotting the predictions against the results, adding a line of perfect corre
lation
plt.scatter(y_test, y_pred)

plt.plot([x for x in range(0,140)],[x for x in range(0,140)], color='red')

plt.xlabel('Actual orders')
plt.ylabel('Predicted orders')
```

Out[37]: `Text(0, 0.5, 'Predicted orders')`



Visually, it seems that the model has predicted fairly well based on the training data. Of course, I am aware of the limited scope of the cluster we analyzed, therefore I will check yet again some statistical errors to have more clues about the strength of the model.

In [38]:
```
# Evaluating the model now that we've used it on the test data. What do each o
f these tell us?
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y
_pred)))
```

```
Mean Absolute Error: 10.345325414825163
Mean Squared Error: 174.2530308552813
Root Mean Squared Error: 13.200493583774863
```

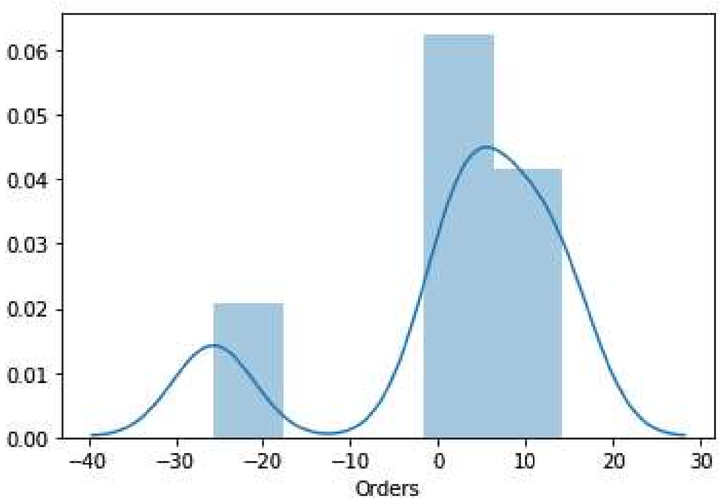
The first of these three, the **mean absolute error** calculates the range of orders that we can loose if the model is wrong - it seems like we can loose on average 10 orders.

As a last test, I will calculate the residuals and check whether they are normally distributed.

In [39]:
```
# Calculating residuals as the difference between actual and predicted values.
residuals = y_test - y_pred
```

In [40]:
```
# Plotting the residuals to check for normal distribution
sns.distplot(residuals)
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24c4c6eea58>
```



Residual clearly look not normally distributed. This, along with the quantity of observations used and some of the results obtained makes me conclude that the model was not successful.

4. Conclusions

General conclusions

Clearly, the model does not present us with solid evidence that cart additions and page views are strongly related with order.

Although this project does not give me as positive an answer as I would have hoped for, I believe testing our metrics with linear regression is the way forward to find good insight. The next step to be taken based on this project is, in my opinion, to perform similar experimental tests on the whole catalogue.

Suggesting a new KPI

An important finding during my analysis has been the rate at which products drop out of the cart. Based on this finding and on internet research on the subject, I communicated an additional KPI to the team - that we are currently experimenting in the French market.

The new KPI is called **Cart abandonment ratio**, which is explained as follows:

$$CartAbandonmentRatio = \frac{Orders}{CartAddition}$$

Clearly a larger number indicates a product that has a higher risk of dropping out of the cart. This information can inform a revision of the web page for these particular products and hopefully boost orders in the future