POLIST_01_notebookanalyse

September 8, 2021

[1]:
```
import numpy as np
import pandas as pd

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score,␣
 ↪adjusted_rand_score

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.cm as cm

%matplotlib inline
import seaborn as sns

import datetime
from dateutil.relativedelta import *
import scipy.stats as stats
import math as mt
import timeit

from sklearn import decomposition, preprocessing

import plotly.graph_objects as go
import plotly.express as px
```

Reading input data

[2]:
```
customers = pd.read_csv("./data/olist_customers_dataset.csv")
geoloc = pd.read_csv("./data/olist_geolocation_dataset.csv")
items = pd.read_csv("./data/olist_order_items_dataset.csv")
payments = pd.read_csv("./data/olist_order_payments_dataset.csv")
reviews = pd.read_csv("./data/olist_order_reviews_dataset.csv")
orders = pd.read_csv("./data/olist_orders_dataset.csv")
```

Removing the undelivered orders

[3]:
```
undev_orders = orders[orders['order_delivered_customer_date'].
 ↪isna()]['order_id']
```

[4]:
```
items = items.drop(undev_orders.index)
payments = payments.drop(undev_orders.index)
reviews = reviews.drop(undev_orders.index)
orders = orders.drop(undev_orders.index)
```

Customers

[5]:
```
customers.head()
```

[5]:

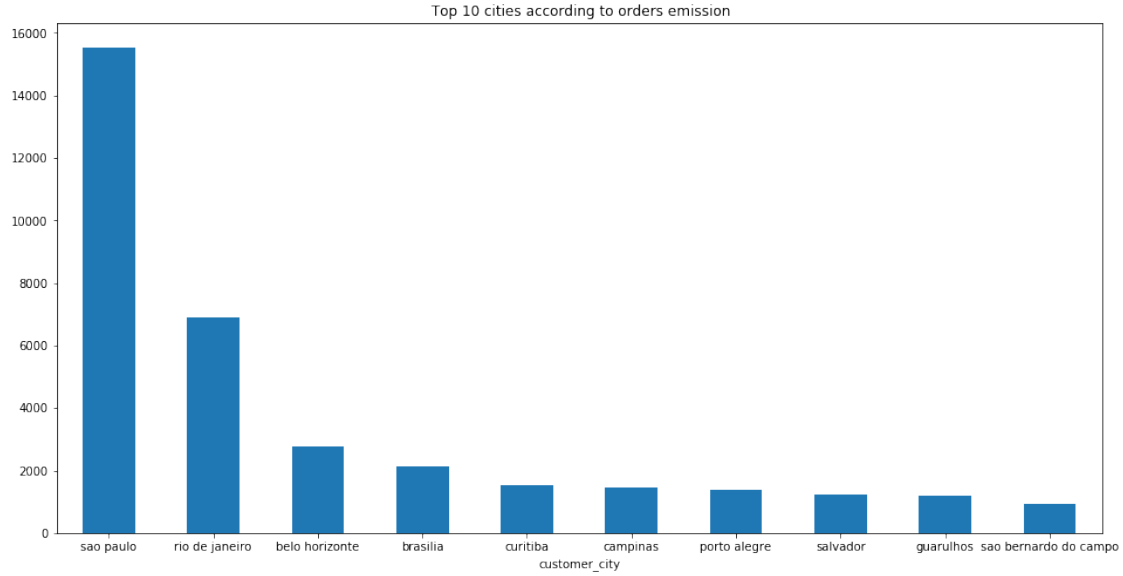
	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state
0	06b8999e2fba1a1fbc88172c00ba8bc7	861eff4711a542e4b93843c6dd7febb0	14409	franca	SP
1	18955e83d337fd6b2def6b18a428ac77	290c77bc529b7ac935b93aa66c333dc3	9790	sao bernardo do campo	SP
2	4e7b3e00288586ebd08712fdd0374a03	060e732b5b29e8181a18229c7b0b2b5e	1151	sao paulo	SP
3	b2b6027bc5c5109e529d4dc6358b12c3	259dac757896d24d7702b9acbbff3f3c	8775	mogi das cruzes	SP
4	4f2d8ab171c80ec8364f7c12e35b23ad	345ecd01c38d18a9036ed96c73b8d066	13056	campinas	SP

[6]:
```
print("Customers database contains", customers.shape[0], "rows and", customers.
 ↪shape[1], "columns expanding from ",
     customers["customer_unique_id"].nunique(), " unique customers.")
```

```
Customers database contains 99441 rows and 5 columns expanding from  96096
unique customers.
```

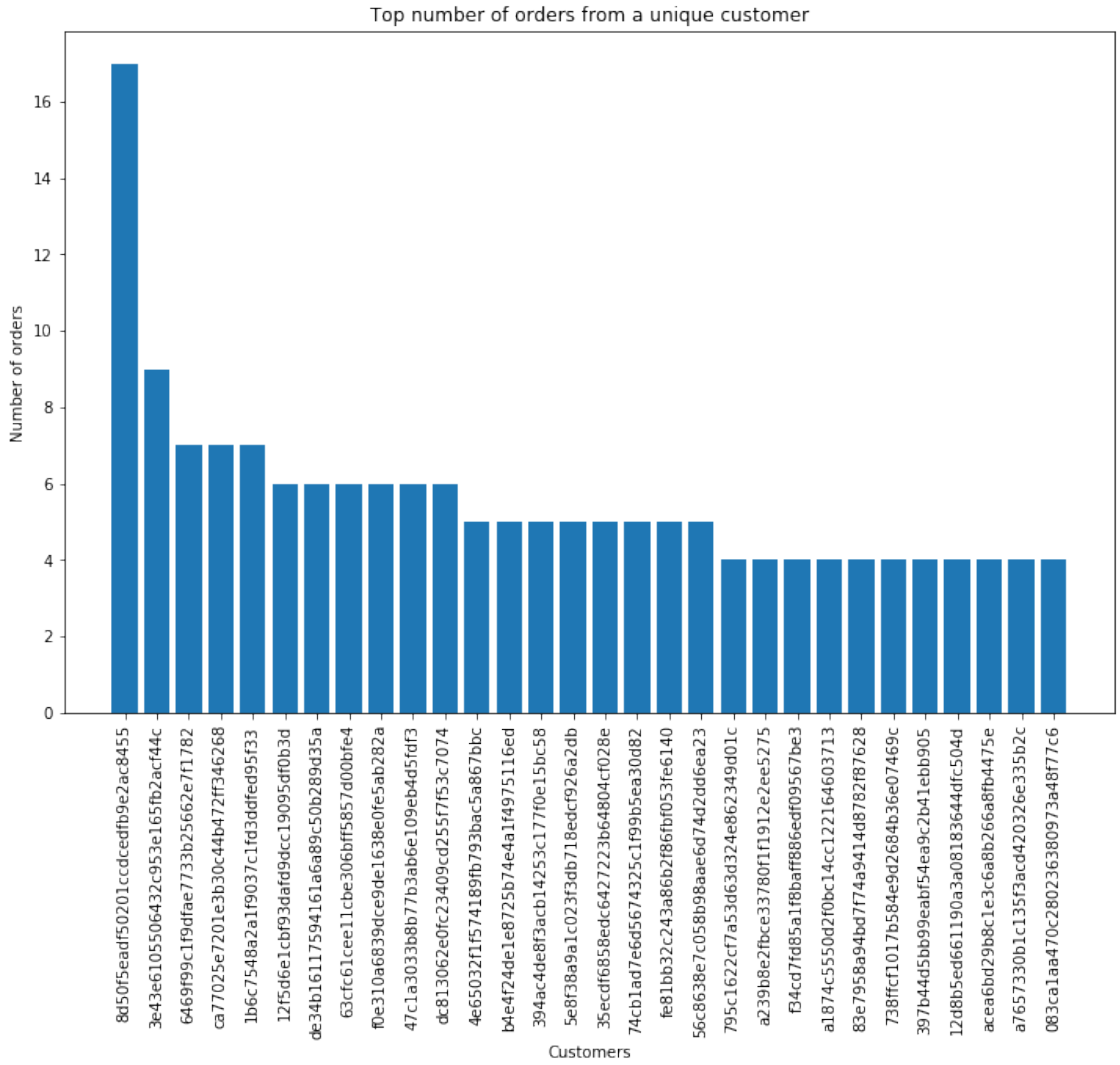
[7]:
```
cities = customers["customer_city"].nunique()
c1 = customers.groupby('customer_city')['customer_id'].nunique().
 ↪sort_values(ascending=False)
print("There are",cities,"unique cities in the database. The TOP 10 cities are:
 ↪\n")
c2 = c1.head(10)
print(c2)
print("\nTOP 10 cities covers", round(c2.sum()/customers.
 ↪shape[0]*100,1),"percent of all the orders.")
plt.figure(figsize=(16,8))
plt.title(" Top 10 cities according to orders emission")
c2.plot(kind="bar",rot=0);
plt.savefig('./Top10Cities', bbox_inches='tight')
```

```
There are 4119 unique cities in the database. The TOP 10 cities are:

customer_city
sao paulo                15540
rio de janeiro            6882
belo horizonte            2773
brasilia                  2131
curitiba                  1521
campinas                  1444
porto alegre              1379
salvador                  1245
guarulhos                 1189
sao bernardo do campo      938
Name: customer_id, dtype: int64

TOP 10 cities covers 35.2 percent of all the orders.
```



[8]:
```
order_freq = customers[['customer_id', 'customer_unique_id']].
 ↪groupby(['customer_unique_id']).count()
order_freq = order_freq.sort_values(by='customer_id', ascending=False)

ax = plt.figure(figsize=(12,8))
#ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
plt.title('Top number of orders from a unique customer')
plt.xlabel("Customers");
plt.ylabel("Number of orders");
plt.bar(height=order_freq.head(30)['customer_id'], x=order_freq.
 ↪head(30)['customer_id'].index);
```

```
plt.xticks(rotation=90);
plt.savefig('./TopNumOrders.png', bbox_inches='tight');
```



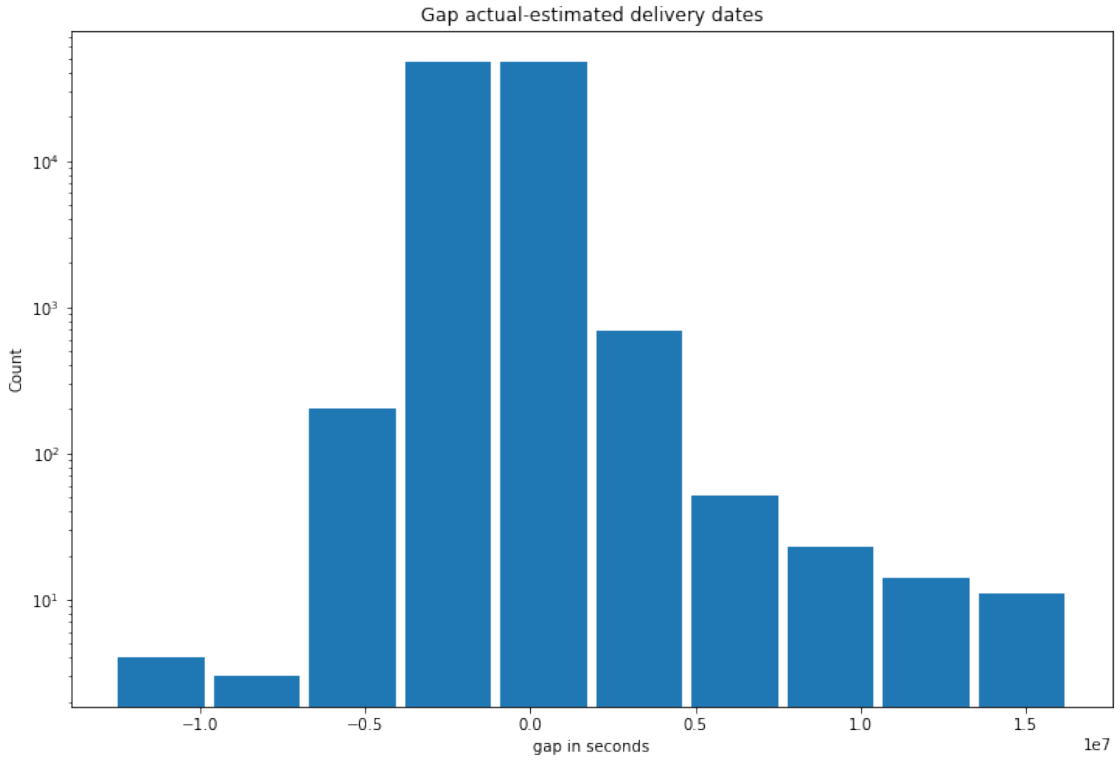
Orders

[9]:
```
orders.head();
```

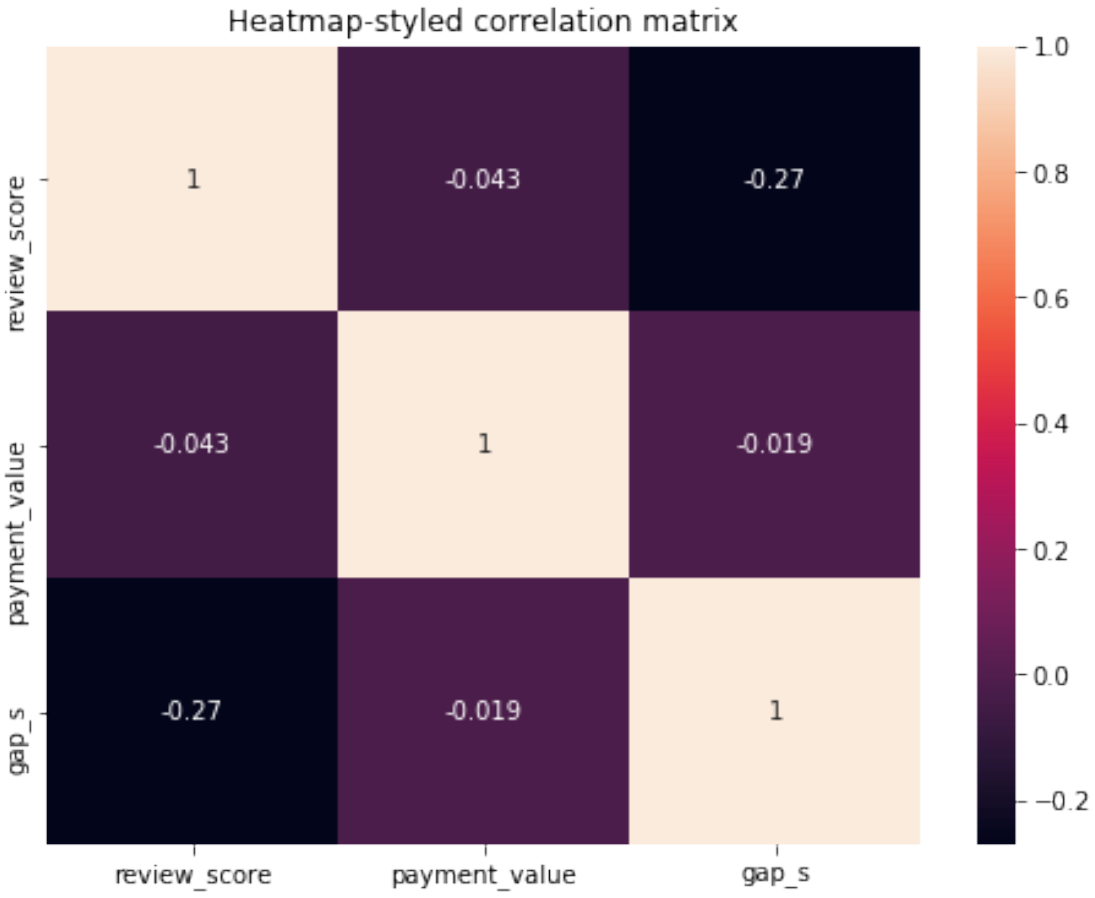
[10]:
```
#convert ['order_purchase_timestamp','order_approved_at',␣
↪'order_delivered_carrier_date','order_delivered_customer_date',␣
↪'order_estimated_delivery_date'] to datetime
date_columns = ['order_purchase_timestamp','order_approved_at',␣
↪'order_delivered_carrier_date','order_delivered_customer_date',␣
↪'order_estimated_delivery_date']
for col in date_columns:
```

```
    orders[col] = pd.to_datetime(orders[col])

# function transforming timedelta values to seconds
def to_seconds(x):
    return x.total_seconds()
```

[11]:
```
delivering_gap = pd.DataFrame()
delivering_gap['gap'] =␣
 ↪orders['order_delivered_customer_date']-orders['order_estimated_delivery_date']
delivering_gap['gap_s'] = delivering_gap['gap'].apply(lambda x: to_seconds(x))

ax = plt.figure(figsize=(12,8))
plt.hist(delivering_gap['gap_s'], rwidth=0.9, label='gap')
plt.semilogy()
plt.title('Gap actual-estimated delivery dates')
plt.xlabel("gap in seconds");
plt.ylabel("Count");

plt.savefig('./GapDelivery.png', bbox_inches='tight');
#plt.xticks()
#ax.set_xticklabels(delivering_gap['gap'])
#(labels='gap')
```



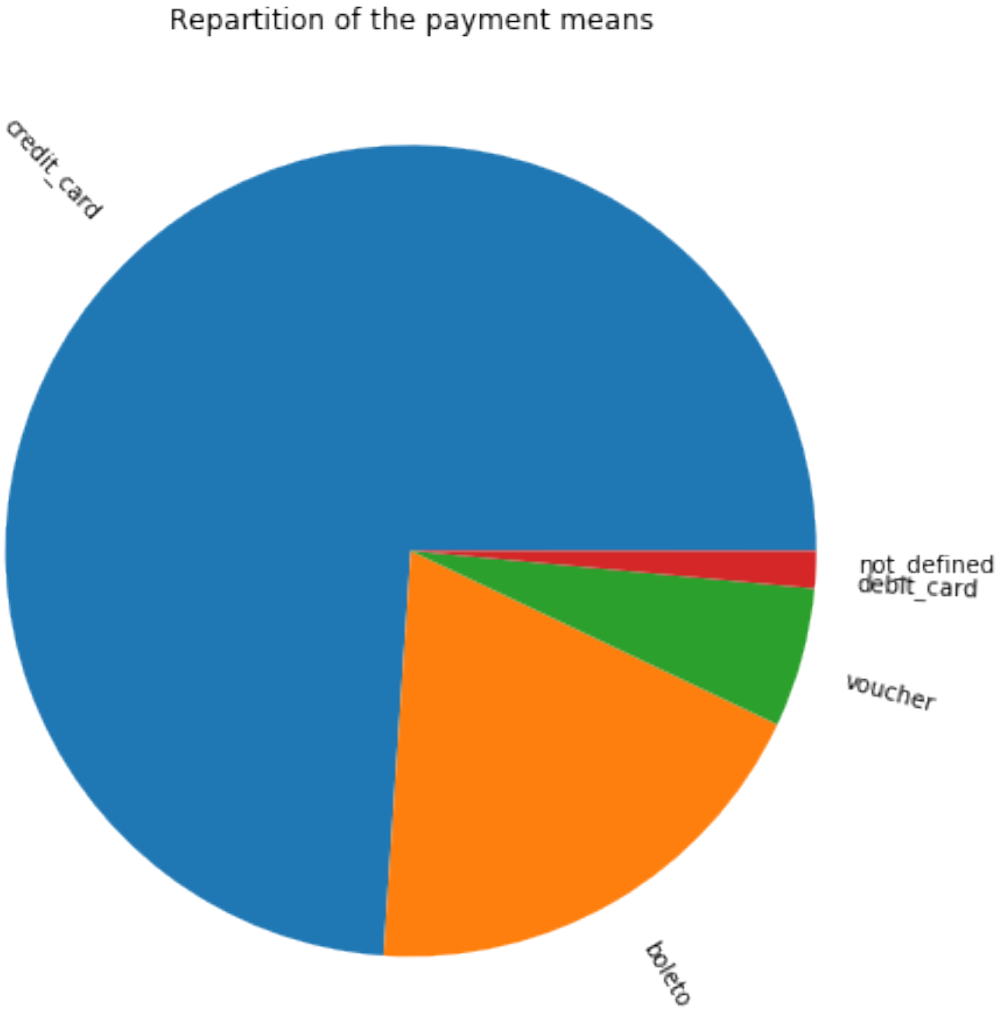
[12]:
```
mean_rev_score = reviews[['order_id','review_score']].groupby(['order_id']).
 ↪mean()
payment_per_order = payments[['order_id','payment_value']].
 ↪groupby(['order_id']).sum()
delivering_gap['order_id'] = orders['order_id']
#rev_del['gap'].corr(rev_del['review_score'])
pay_rev_del = (delivering_gap.merge(mean_rev_score,on='order_id')).
 ↪merge(payment_per_order, on='order_id')
corr_Matrix = pay_rev_del[['review_score', 'payment_value', 'gap_s']].corr()

#Création de la heatmap
plt.figure(figsize=(8,6));
sns.heatmap(corr_Matrix, annot=True) ;#
plt.title('Heatmap-styled correlation matrix');
plt.show();

plt.savefig('./ReviewCorr.png', bbox_inches='tight');
```




```
<Figure size 432x288 with 0 Axes>
```

Payments

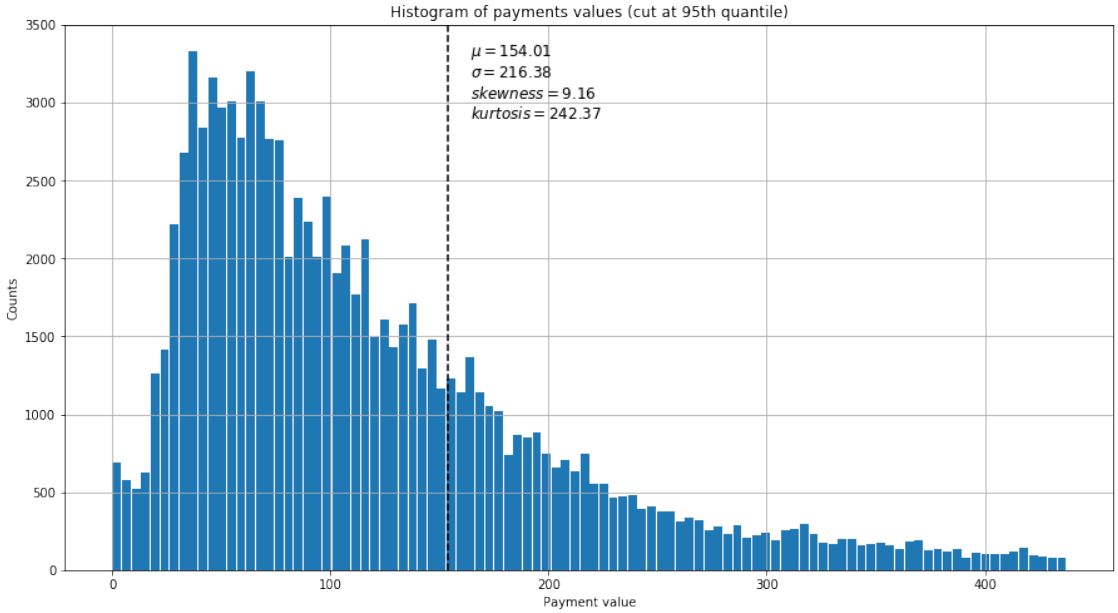
[13]:
```
#print(payments.describe())
#print(payments["payment_type"].value_counts())

payment_means = payments["payment_type"].value_counts()
ax = plt.figure(figsize=(8,8))
plt.pie(payment_means, labels=payment_means.index, rotatelabels=30)
plt.title('Repartition of the payment means')

plt.savefig('./PaymentMeans.png', bbox_inches='tight');

credit = payments[payments["payment_type"]=="credit_card"]
```



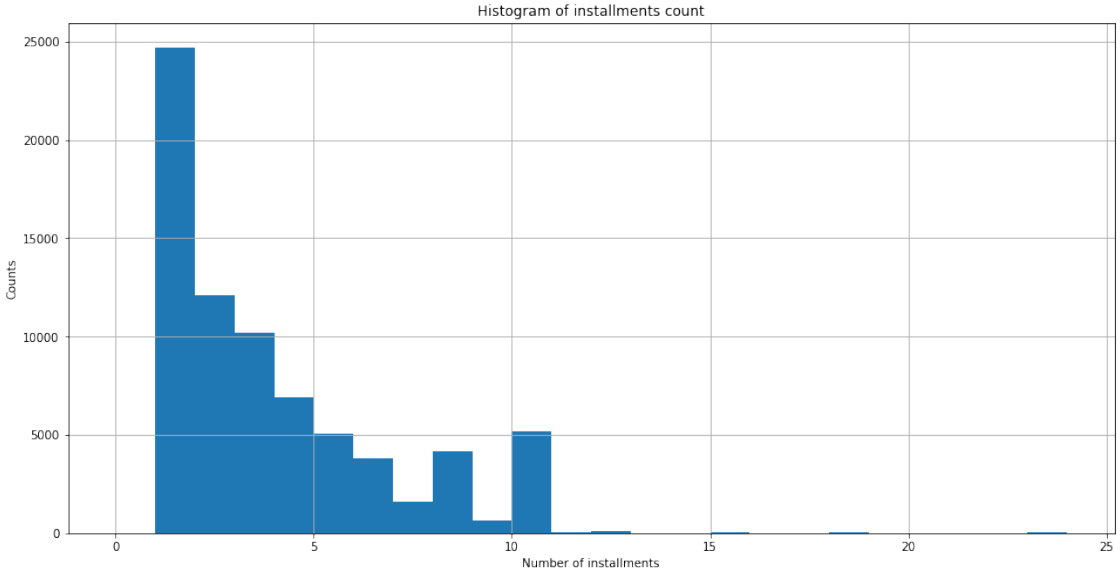
[14]:
```
payments["payment_value"].describe()
```

[14]:
```
count    100921.000000
mean        154.006196
std         216.380967
min           0.000000
25%          56.780000
50%         100.000000
75%         171.840000
max       13664.080000
Name: payment_value, dtype: float64
```

[15]:
```
mean = payments["payment_value"].mean()
std = payments["payment_value"].std()
skew = payments["payment_value"].skew()
kurt = payments["payment_value"].kurtosis()

text1 = '$\mu=$' + str(round(mean,2))
text2 = '$\sigma=$' +str(round(std,2))
text3 = '$skewness=$' + str(round(skew,2))
text4 = '$kurtosis=$' + str(round(kurt,2))
text = text1 + "\n" + text2 + "\n" + text3 + "\n" + text4

q95 = payments["payment_value"].quantile(.95)
payments_q95 = payments[payments["payment_value"]<q95]

payments_q95.hist(column = "payment_value", bins = 100, figsize=(15,8),␣
 ↪rwidth=0.9);
plt.axvline(mean, color='k', linestyle='--')
plt.text(mean+10, 2900, text, fontsize=12)
plt.xlabel("Payment value")
plt.ylabel("Counts")
plt.title("Histogram of payments values (cut at 95th quantile)")

plt.savefig('./HistoPayValues.png', bbox_inches='tight');
```



[16]:
```
fig, ax = plt.subplots(figsize=(16,8))
credit.hist(column = "payment_installments", bins =␣
 ↪credit["payment_installments"].max(), ax=ax);
plt.xlabel("Number of installments")
plt.ylabel("Counts")
plt.title("Histogram of installments count")

plt.savefig('./HistoInstallmentCounts.png', bbox_inches='tight');
```



Retrieving the useful datasets

[17]:
```
orders.to_csv('orders.zip')
customers.to_csv('customers.zip')
payments.to_csv('payments.zip')
items.to_csv('items.zip')
reviews.to_csv('reviews.zip')
```