Greg Lee-Podgorski - Head of Machine Learning & Technology Platforms @ RS

CLASS PREDICTION USING CLASSIFICATION - TOP SELLING PRODUCTS

I. Define & Scope

The Brief

Based on internal RS corporate data on best-selling products we carry, use classification method(s) to analyze product features available in the dataset and find most suitables ones for class prediction (conversion rate below/above mean). Then create a visualization of the resulting analysis. Draw conclusions, if findings allow it. ¶

i) Intro & my role in the organization

In this exercise I'm going to apply classification method(s) of Machine Learning to our best-selling products in RS Components product catalogue. While classification algorithms consist of a number of algorithms available to be used, from naive Bayes, through Decision Tree(s) to Random Forest, none of those is perfect. i'm going to utilize couple of those aforementioned to mine for most correlated features that affect class prediction among those top selling products, trying to determine which class each observation belongs to and hoping to find some hidden patterns or shared characteristics in the process.

I happen to hold a position of Head of Machine Learning, Automation & Technology Platform within the Electrocomponents PLC Group (RS Components is a trading name in the UK and EMEA). Thus I am an outspoken advocate for increased use of ML in our organization and also a passionate believer in value that deploying ML algorithms into our operations can unlock for us. I'm hoping that by showing use cases of certain algorithms, I will lead by example, showing how ML algos can be used to find hidden patterns in data and provide business colleagues with valuable insights.

ii) Background

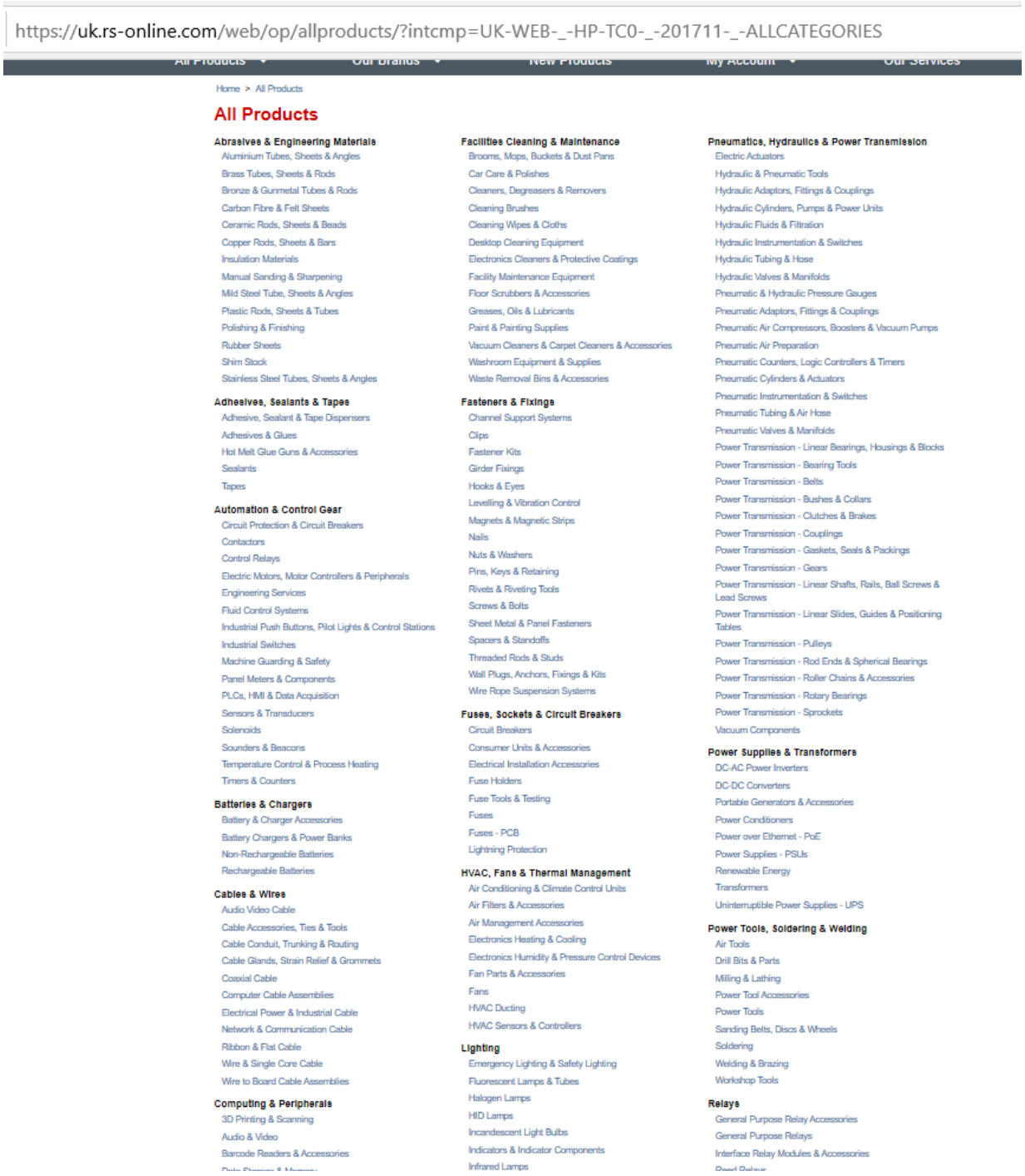
RS Components is a publicly listed corporation and trades on LSE (London Stock Exchange) under Electrocomponents PLC trade name. We have significant trading presence in each of the top economically most active regions - EMEA, APAC and Americas. Due to wildly varying standards across any number of electronic products specifications that have local requirements, whether voltage, number of prongs/holes in a electric socket, certification standards or just local engineer's preferences, standarization is a challenge for RS and thus we need to stock very large range of products. As it stands RS currently has over 500,000 SKU's (over 0.5 million product in stock) and over 2.5 million non-stock products which we can drop-ship for our clients from our global supplier base. That means that while most of online stores would be content with coming up with a Top 100 best selling products, out Best Sellers list features slightly over 10,000 products. Some of the products on that best-sellers' list vary only slightly from their "cousins", e.g. by different AC/DC output or electrical resistance (in Ohms).

To make matters even more complicated we carry a number of distinct categories of products, e.g. BLE (Board Level Electronics) - something a computer maker like Asus or Dell would need, or IA&C (Undustrial Automation & Control) - e.g. assembly line sensors monitoring health of a conveyor belt, or PPE (Personal Protection Equipment) - e.g. gloves or protective masks engineers or doctors can cover their faces with.

iii) Problem statement and potential business application

With such huge range or products covering a number of distinct categories and sourced from over 2500 unique suppliers across different geographies, it's little wonder that our investory management function is 3-digit strong when it comes to headcount and inventory planners are always playing a catch-up game. Whether it comes to updating ERP system with up-to-date stock counts from a number of out trade counters (brick and mortar store locations), keeping track what's about to run out in our warehouses located in key countries globally or even trying to assign priorities correctly across several different team within Inventory Management, while RS as a company noticeably improved our approach to managing stock, we are surely in need of further improvements. Those will be difficult to come by and even more so - to implement - unless we manage to identify signicant patterns in underlying data.

Being able to predict if a given product, product family or newly sourced parts' range will have above or below average conversion rate will enable our marketing team to customize our sales offers and highlight promotions to online shoppers when they are interacting with those products that based on their features we expect to fight uphill battle to reach average conversion rate and direct inventory teams to stock less of those where we expect conversion rates to struggle. In more clear-cut cases they might even be able to terminate orders from our suppliers or renegotiate better deals so that we in turn can offer better pricing and thus provide a lift to those struggling conversion rate products.



iv) Data

The dataset has been sourced internally, with a help of a product analyst, who is a part of inventory management function and looks after best-selling products, making sure they are in stock with maximum availability. The Excel files utilized here has over 11,000 rows with multiple columns providing information such as Category, Technology type, number of orders, etc.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Article	Description	FSTD Global	Category	Cell	Technology	Visit	Product Views	In Stock Views	Cart Additions	Orde	Conversion Rate	Revenue
26	9092061	PCM INTERFACE PAD FOR HS300, 128X72.5MM	12/31/9999	II&T	EMECH	Thermal Management	98	101	0.713	6	2	0.0204	135.68
27	291717	NYLON 6.6 HOSE CLIP,26.5-29.5MM	12/31/9999	TCFM	Consumables & Safety	Fasteners	281	256	0.835	43	17	0.0605	182.56
28	5217528	DELRIN SPUR GEAR - 1.0 MODULE 25 TEETH	12/31/9999	IA&C	Mech & Flu Pwr	Pow Tran & Struc	226	247	0.781	36	15	0.0664	205.26
29	9225141	SOLID STATE RELAY	12/31/9999	IA&C	Sensors & Switches	Ctrl & Ind Relay	274	312	0.254	16	1	0.0036	211.08
30	2664268	MEDIUM DISPOSABLE CO-POLYMER GLOVES	12/31/9999	TCFM	Consumables & Safety	PPE	274	264	0.989	41	6	0.0219	213.6

v) Data protection

The dataset used doesn't contain any customer-related info (other than number of orders and views in our online store) so there is no personally-identifiable information available. Thus there is no risk of any GDPR violation. As far as corporate data is concerned, there is no strictly confidential data being used. Almost all of the data used is publicaly available on our web store. The only potentially sensitive data in there is revenue generated by each of the top selling products. If accessed in its entirety, one could imagine a potential risk of a competitor being able to peek behind the curtain and try to replicate - say - our Top 1000 selling products and undercutting our pricing. To avoid that, I have sorted revenue in ascending order and never show the column content - only few rows at a time.

Loading libraries

In [1]:
```
# for loading and manipulating dataframes
import pandas as pd

# for additional mathematical functionality
import numpy as np

# Seaborn and matplotlib for visualisation
import matplotlib.pyplot as plt
import seaborn as sns

# for building and evaluating our classification models
from sklearn import tree, metrics # This is for us to simply evaluate our mo
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier # This provides the impl
from sklearn.metrics import confusion_matrix #This is for a more thorough ev
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

# for plotting our tree:
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

```
C:\Users\oszer\anaconda3\lib\site-packages\sklearn\externals\six.py:31: FutureW
arning: The module is deprecated in version 0.21 and will be removed in version
0.23 since we've dropped support for Python 2.7. Please rely on the official ve
rsion of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
```

Sourcing data

In [2]:
```
1 best_sell = pd.read_excel("..\Top Sellers Digital Data_KP.xlsx")
```

In [3]:
```
1 best_sell.tail()
```

Out[3]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	Sto Viev
11453	7609005	ALUMINIUM PAINT ENCLOSURE 160X160X90MM	9999-12-31 00:00:00	IA&C	Mach Guard & Enc	Enclosures	390	383	0.7
11454	4992470	PIPE CLEANING BRUSH FOR 15MM DIA PIPE	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	1194	1125	0.9
11455	1257961	INDUSTRIAL CABLE 10MM2 10M	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Elec & Data Cable	291	335	0.2
11456	3971315	MINERAL INSULATED PT 100 SENSOR,3X150MM	9999-12-31 00:00:00	IA&C	Sensors & Switches	Proc Auto Sen	392	408	0.9
11457	2414782	ADHESIVE BASE MICRO WIRE SADDLE,9X7.3MM	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	591	561	0.9

Exploring, Cleaning & Transforming data

Exploring

Let's first have a quick look at what type data and its shape we are dealing with..

In [4]:
```
1 best_sell.shape
```

Out[4]:
```
(11458, 13)
```

As we can see, our dataframe has almost 11,500 rows and 13 columns

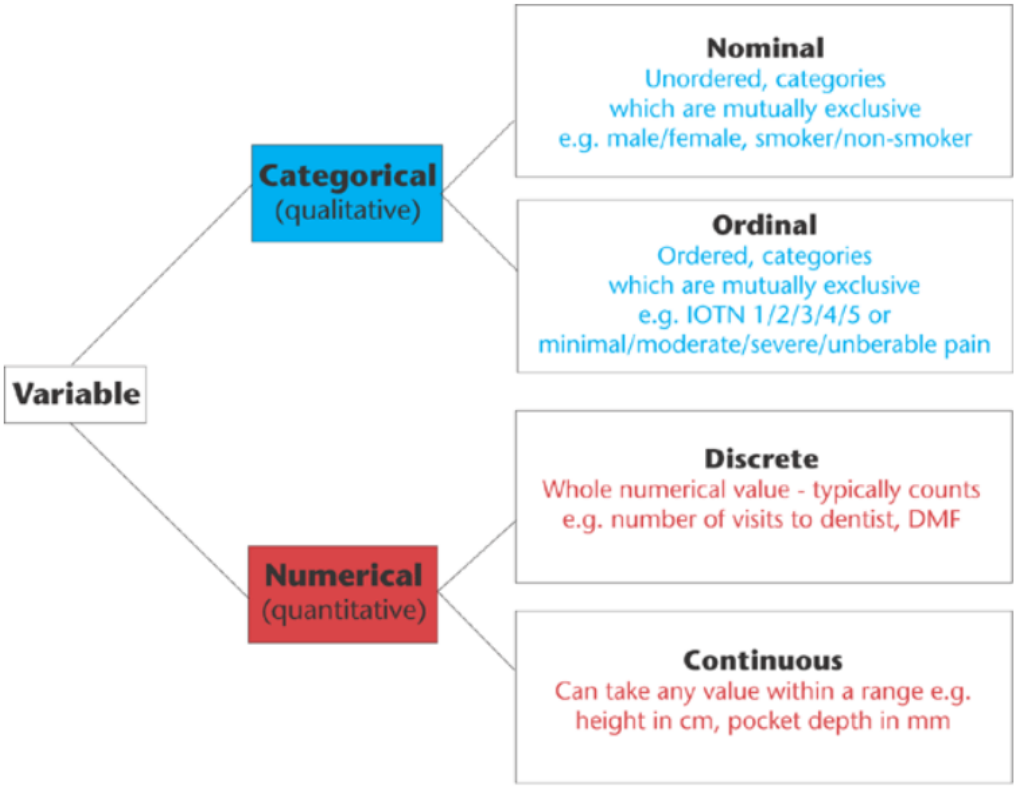
In [5]:
```
1 best_sell.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11458 entries, 0 to 11457
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Article          11458 non-null  int64  
 1   Description      11458 non-null  object 
 2   FSTD Global      11458 non-null  object 
 3   Category         11458 non-null  object 
 4   Cell             11458 non-null  object 
 5   Technology       11458 non-null  object 
 6   Visits           11458 non-null  int64  
 7   Product Views    11458 non-null  int64  
 8   In Stock Views   11458 non-null  float64
 9   Cart Additions   11458 non-null  int64  
 10  Orders           11458 non-null  int64  
 11  Conversion Rate  11458 non-null  float64
 12  Revenue          11458 non-null  float64
dtypes: float64(3), int64(5), object(5)
memory usage: 1.1+ MB
```

Good news - no null values in any of the rows - that doesn't happen often. I guess my colleague made sure the dataset has been of proper quality.

We can see that majority of the columns contain categorical values, while remaining 5 columns (1st column, 7th, 8th, 10th and 11th col) have numerical records.

Here is a quick refresher on how we classify types of variables:



As per above dichotomy, we can see that columns 6 through 11 contain discrete numerical values, the same for Article column and while Revenue columns appear to contain continuous numerical variables. Remaining columns possess nominal variables.

In [6]:
```
1 best_sell.dtypes
```

Out[6]:
```
Article              int64
Description         object
FSTD Global         object
Category            object
Cell                object
Technology          object
Visits               int64
Product Views        int64
In Stock Views     float64
Cart Additions       int64
Orders               int64
Conversion Rate    float64
Revenue            float64
dtype: object
```

We can further see that columns Article, Visits, Product Views, Cart Additions and Orders contain whole numbers (aka integral = integers) while columns In Stock Views, Conversion rate and Revenue contain fractionals (in this case 64-bit floating point data types).

Interesting trivia - Nonintegral Numeric Types:

Nonintegral data types are those that represent numbers with both integer and fractional parts. The nonintegral numeric data types are Decimal (128-bit fixed point), Single Data Type (32-bit floating point), and Double Data Type (64-bit floating point). They are all signed types. If a variable can contain a fraction, declare it as one of these types. Decimal is not a floating-point data type. Decimal numbers have a binary integer value and an integer scaling factor that specifies what portion of the value is a decimal fraction. You can use Decimal variables for money values. The advantage is the precision of the values. The Double data type is faster and requires less memory, but it is subject to rounding errors. The Decimal data type retains complete accuracy to 28 decimal places. Floating-point (Single and Double) numbers have larger ranges than Decimal numbers but can be subject to rounding errors. Floating-point types support fewer significant digits than Decimal but can represent values of greater magnitude.

Cleaning

Let's check for unique values now in columns with categorical record - to make sure there are redundancies in naming (which could negatively impact our model):

In [7]:
```
1 best_sell["Category"].unique()
```

Out[7]:
```
array(['II&T', 'TCFM', 'IA&C', 'BLE', 'OKdo'], dtype=object)
```

In [8]:
```
1 best_sell["Cell"].unique()
```

Out[8]:
```
array(['Cable, Anti-Vib & Ac', 'Consumables & Safety',
       'Tools Storage & FM', 'I-Conn', 'Sensors & Switches',
       'Process Control', 'Mach Guard & Enc', 'E-Conn',
       'Test & Measurement', 'Passives', 'EMECH', 'Mech & Flu Pwr',
       'BLE E-Mech', 'Opto', 'Analog Disc Power', 'SBC & IOT'],
      dtype=object)
```

In [9]:
```
1 best_sell["Technology"].unique()
```

Out[9]:
```
array(['Cable Accs', 'Battery', 'Hand Tools', 'Adhesives, Clean & M',
       'Fasteners', 'Measurement & Insp', 'Crimp Term & Power', 'HVAC',
       'Lighting', 'Proc Auto Sen', 'Power Management', 'Enclosures',
       'Pow Tools & Abras', 'Site Safety & Janit', 'IT', 'Fact Auto Sen',
       'Rect Connectors', 'Electronic T&M', 'Resistors',
       'Push But & Swit', 'Electrical T&M', 'Soldering & ESD',
       'Indicators', 'Acc, Stor & Mat Hand', 'Pow Tran & Struc',
       'Printing & Office', '3D Printing', 'Pwr & Ind Cable',
       'RF Datacomm Conns', 'Motors & Control', 'Ind & AV Conns',
       'Capacitors', 'PPE', 'Plumbing', 'Level & Flow Con',
       'Engineering  Mat', 'Cable Mgt & Elec Acc', 'Hydraulics',
       'Thermal Management', 'Hose & Pumps', 'Pneumatics',
       'Elec & Data Cable', 'Security', 'Ctrl & Ind Relay',
       "DIN Rail PSU's", 'Timers, Counters', 'BLE E-Mech Tech',
       'Temp Control', "PLC's & HMI's", 'Inductors', 'Switches-EMECH',
       'Fuses', 'DIN Rail & Term', 'PCB Connectors', 'Circuit Protection',
       'MIL Ind & Test Conns', 'Sounders/Beacons', 'Anti-Vibration',
       'Relays-EMECH', 'Contactors', 'Heavy Duty Conn', 'Machine Safety',
       'LED', 'Power PCB', 'Displays', 'SBC IOT Dev. Tools', 'Sensors'],
      dtype=object)
```

I am purposely skipping Article column as I expect there to be 1000's of unique product names.

We are lucky - there doesn't seem to be a need for further data cleaning at this point. If that need arises, I'll do it on the fly.

Investigating

Let's see now what are the most commonly ordered products:

In [10]:
```
1 best_sell.sort_values(by=['Orders'], ascending=False).head(1)
```

Out[10]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views	A
14	8660672	RS LITHIUM COIN CELL CR2032 5PK	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	45475	28009	0.977	

Oh, the revenue... I'm going to hide it, as not to reveal too much at this stage:

In [11]:
```
1 best_sellhid=best_sell.drop(['Revenue'], axis=1)
```

In [12]:
```
1 best_sellhid.head(7)
```

Out[12]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	I Stoc View
0	233487	BLACK CABLE TIE, 300X4.8MM, PACK 100	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	43371	28655	0.99
1	5375488	RS SEALED LEAD-ACID BATTERY,12V 7AH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	40171	41566	0.99
2	7348885	88PC 1/2IN. SOCKET & VDE TOOL KIT	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	79286	89157	0.97
3	1681644	DUST REMOVER CLEANER 400ML	9999-12-31 00:00:00	TCFM	Consumables & Safety	Adhesives, Clean & M	8704	8340	0.93
4	6191506	SEALING STRIP,PVC,WIRE INSERT,1-2,9X6.	9999-12-31 00:00:00	TCFM	Consumables & Safety	Fasteners	9605	9731	0.99
5	8412518	ELECTRONIC CALIPER 150MM/6"	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Measurement & Insp	35240	34067	0.99
6	100282	BRASS SLOT'D SCREW,NUT & WASHER KIT	9999-12-31 00:00:00	TCFM	Consumables & Safety	Fasteners	1609	1760	0.99

ok, now we can safely go back to seeing most frequently ordered products:

In [13]:
```
1 best_sellhid.sort_values(by=['Orders'], ascending=False).head(10)
```

Out[13]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views
14	8660672	RS LITHIUM COIN CELL CR2032 5PK	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	45475	28009
89	7757233	LR44 ALKALINE COIN CELL 1.5V 158MAH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	51456	37928
0	233487	BLACK CABLE TIE, 300X4.8MM, PACK 100	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	43371	28655
29	512238	PTFE THREAD SEAL TAPE,12M L X 12MM W	9999-12-31 00:00:00	TCFM	Consumables & Safety	Adhesives, Clean & M	36293	26000
157	233455	CABLE TIE,100 X 2.5,BLACK,PACK 100	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	27824	17143
72	458702	RED INSUL BOOTLACE FERRULE,8MMPIN 1MMSQ.	9999-12-31 00:00:00	II&T	I-Conn	Crimp Term & Power	29405	19737
33	7442199	NON-RECHARGEABLE AA ALKALINE BATTERY	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	22215	13742
36	458718	BLK INSUL BOOTLACE FERRULE,1.5MMSQ. WIRE	9999-12-31 00:00:00	II&T	I-Conn	Crimp Term & Power	29259	19750
26	593423	RS SR44 COIN CELL BATTERY, 1.55V	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	23043	17434
138	467406	STANDARD TOP HAT PUNCHED DIN RAIL,0.5M	9999-12-31 00:00:00	IA&C	Mach Guard & Enc	Enclosures	32740	25221

It looks like batteries and cables are most popular among our customers.

Now let's find out which products got most eyeballs (but not necessarily most orders):

In [14]:
```
1 best_sellhid.sort_values(by=['Product Views'], ascending=False).head()
```

Out[14]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views
2	7348885	88PC 1/2IN. SOCKET & VDE TOOL KIT	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	79286	89157	0.970
1293	8179236	55PC BIT AND SOCKET SET 1/4" DRIVE	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	55287	58233	0.730
1	5375488	RS SEALED LEAD-ACID BATTERY,12V 7AH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	40171	41566	0.992
89	7757233	LR44 ALKALINE COIN CELL 1.5V 158MAH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	51456	37928	0.997
397	7613319	PROFILE 8 40X40 LIGHT 1M	9999-12-31 00:00:00	IA&C	Mech & Flu Pwr	Pow Tran & Struc	28691	37330	0.995

Pandas is telling us that it wasn't batteries or cables but, actually, sockets. Interesting. It would appear that conversion rate of sockets is noticeably lower than that of the 2 former groups. Perhaps it's driven by price. Either way - interesting fact...

Finally, let's learn which product was **least** often in stock, when viewed:

In [15]:
```
1 best_sellhid.sort_values(by=['In Stock Views']).head()
```

Out[15]:

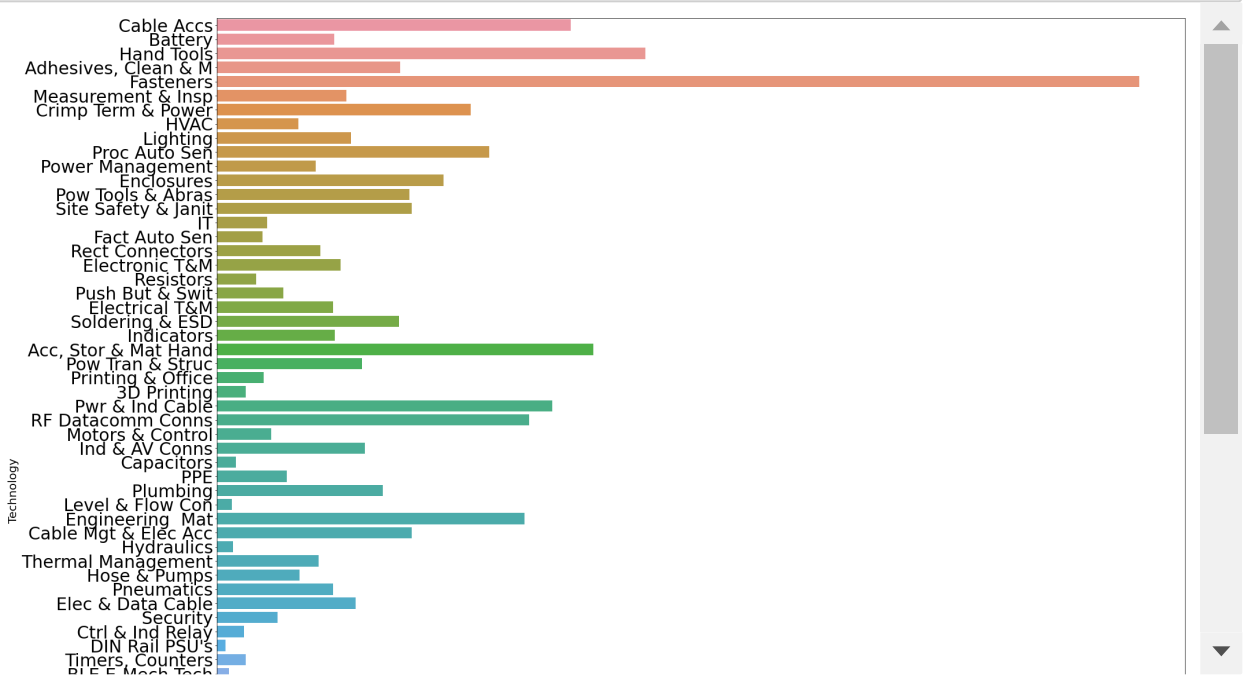
	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views	Add
1622	6693677	THREE STEP MOBILE PLATFORM	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Acc, Stor & Mat Hand	0	0	0.000	
8995	1777087	OIL FILLED RADIATOR 2KW 9FIN UK PLUG	9999-12-31 00:00:00	TCFM	Tools Storage & FM	HVAC	8	0	0.000	
6164	9076422	5/2 PILOT OPERATED SOLENOID VALVE, G1/2"	9999-12-31 00:00:00	IA&C	Mech & Flu Pwr	Pneumatics	190	222	0.009	
10254	8938534	FUSE KIT 6.3X32 FAST AND SLOW GLASS 250V	9999-12-31 00:00:00	II&T	EMECH	Fuses	1921	1922	0.051	
10273	7761989	ORANGE 3 CORE ARCTIC CABLE 2.5MM 100M	2024-12-19 00:00:00	II&T	Cable, Anti-Vib & Ac	Pwr & Ind Cable	520	542	0.174	

I won't pretend to be able to exlain what's so special about solenoid valve but it looks like it was viewed 222 times and was only in stock less than %1 of those times. And fuse kit was clicked on to view 1921 times and yet it was available for purchase only 5.1% of the time...

Visualizing the data

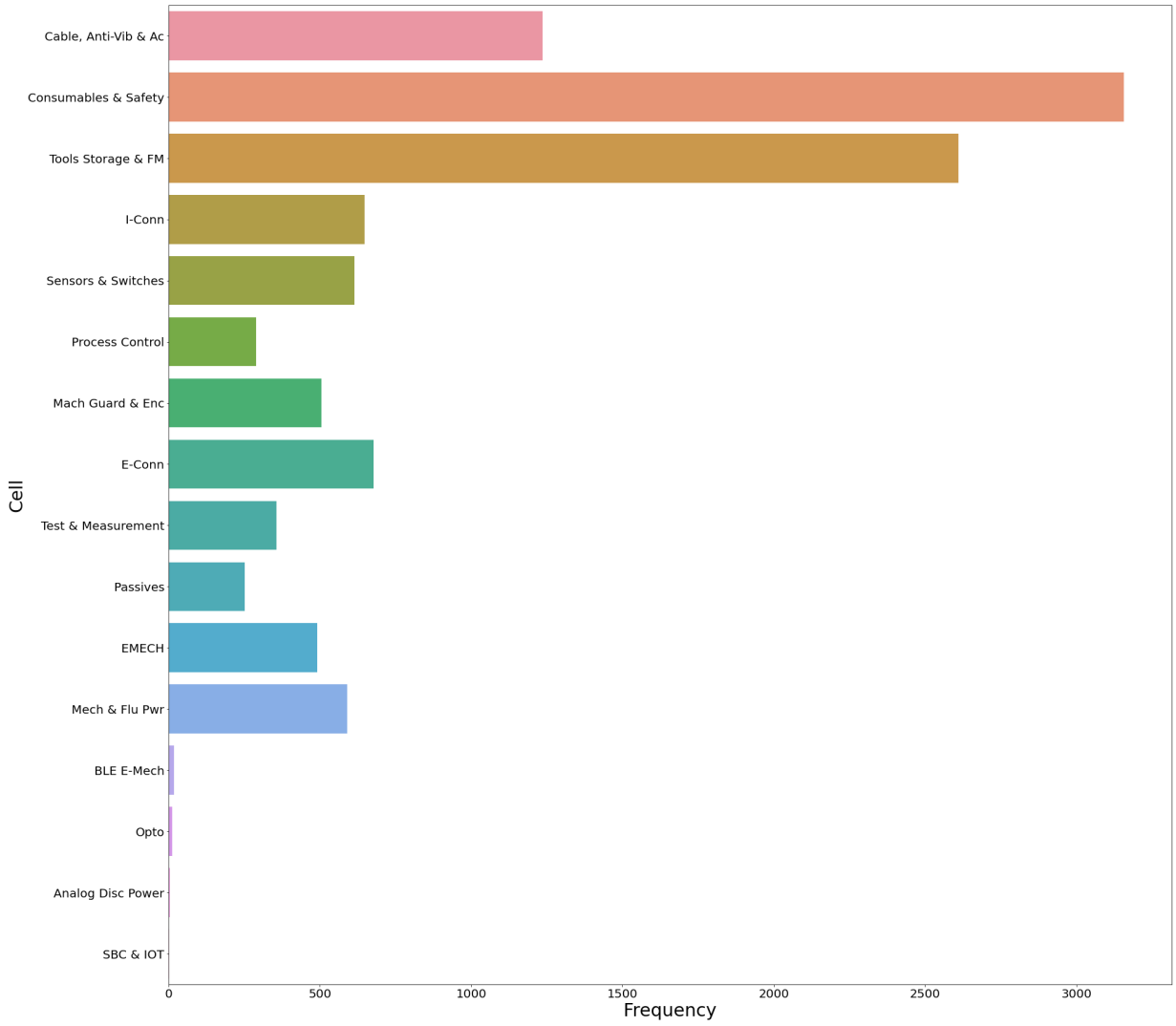
In [16]:

```
# bar-plotting distribbution of Technology
fig1 = sns.countplot(y = 'Technology', data = best_sellhid)

fig1.figure.set_size_inches(30,30)


fig1.set_ylabel("Technology", fontsize=20)
fig1.set_xlabel("Frequency", fontsize=30)
fig1.tick_params(labelsize = 30)

plt.show()
```



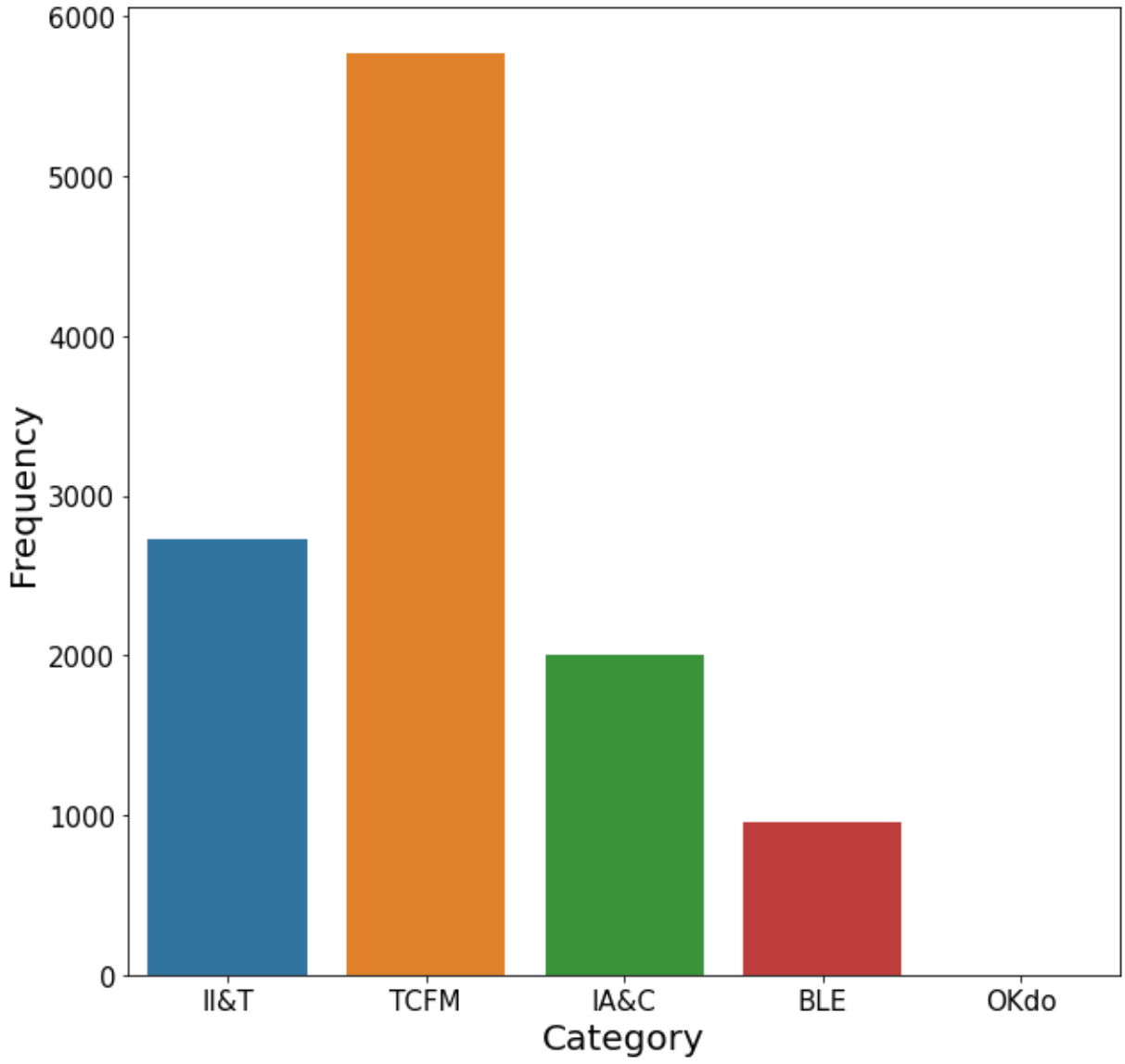
In [17]:

```
# bar-plotting distribbution of Technology
fig1 = sns.countplot(y = 'Cell', data = best_sellhid)

fig1.figure.set_size_inches(30,30)


fig1.set_ylabel("Cell", fontsize=30)
fig1.set_xlabel("Frequency", fontsize=30)
fig1.tick_params(labelsize = 20)

plt.show()
```

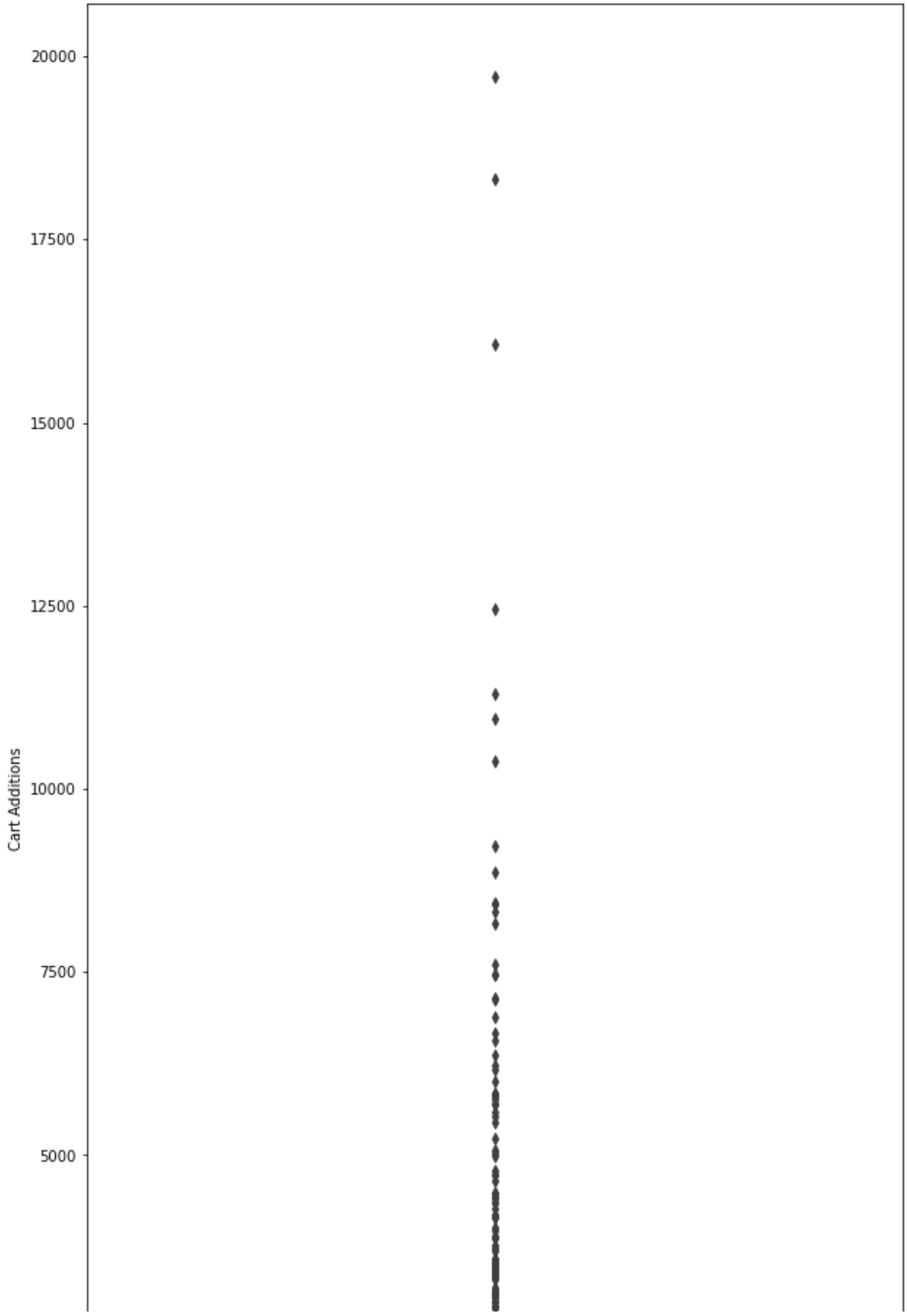


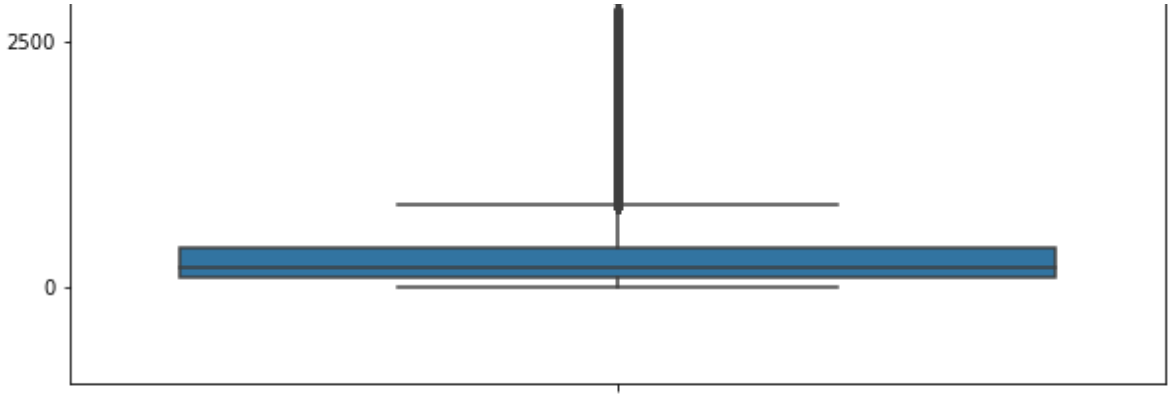
In [18]:

```
# bar-plotting distribbution of Technology
fig1 = sns.countplot(x = 'Category', data = best_sellhid)

fig1.figure.set_size_inches(10,10)


fig1.set_xlabel("Category", fontsize=20)
fig1.set_ylabel("Frequency", fontsize=20)
fig1.tick_params(labelsize = 15)

plt.show()
```

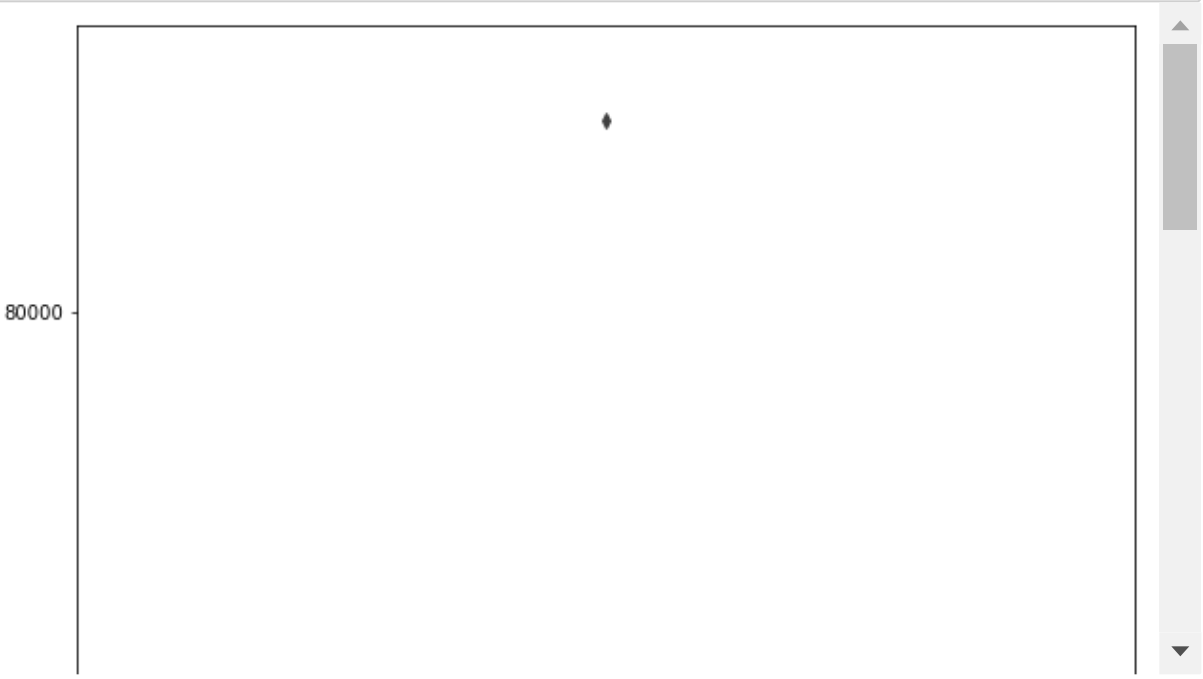


In [19]:
```
# boxplotting to check the variation of Cart Additions
fig2 = sns.boxplot(y = 'Cart Additions', data = best_sellhid)
fig2.figure.set_size_inches(10,20)
plt.show()
```





In [20]:
```
# boxplotting to check the variation of Product Views
fig3 = sns.boxplot(y = 'Product Views', data = best_sellhid)
fig3.figure.set_size_inches(10,20)
plt.show()
```



Let's use seaborn's pairplot to see if anything pops up, looking at Category feature:

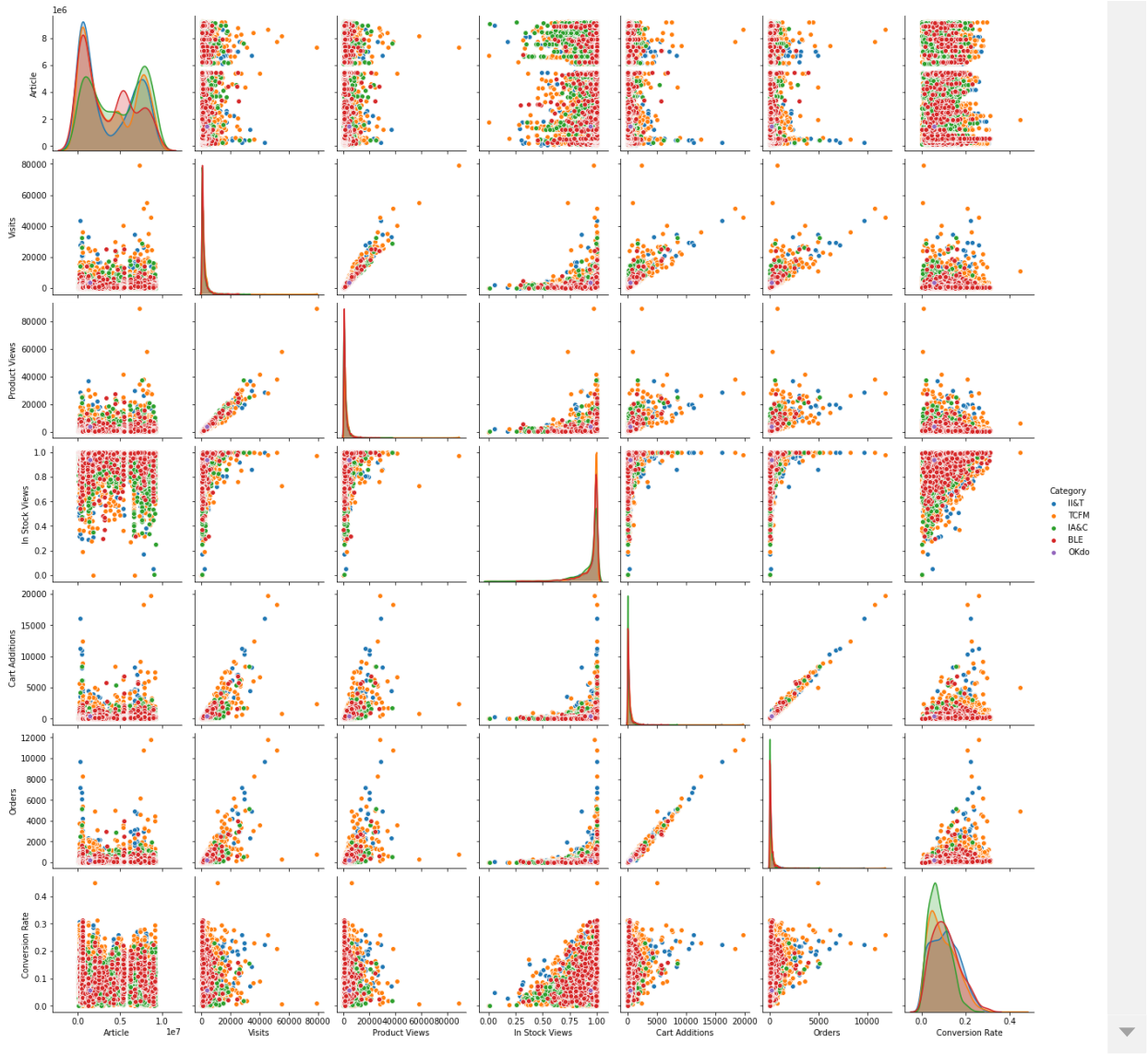
In [21]:

```
pair = sns.pairplot(best_sellhid, hue="Category")
pair
# Run this cell twice if you have a warning message to see the plot
```

```
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\oszer\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWa
rning: Data must have variance to compute a kernel density estimate.
  warnings.warn(msg, UserWarning)
```

Out[21]:

```
<seaborn.axisgrid.PairGrid at 0x1d6d924a088>
```



That's too noisy to discern much. But looking at OkDo category there seemed to have been hardly any. let's confirm it:

In [22]:
```
okdo = best_sell["Category"].value_counts()
okdo
```

Out[22]:
```
TCFM    5768
II&T    2730
IA&C    1998
BLE      961
OKdo       1
Name: Category, dtype: int64
```

Indeed there is only 1 product in OkDo category. It's safe to drop it and see if that helps...

In [23]:
```
#dropping OKdo from Category column:
best_sellhidwithoutOk = best_sellhid[best_sellhid['Category'] != 'OKdo']
```

In [24]:
```
1 best_sellhidwithoutOk["Category"].value_counts()
```

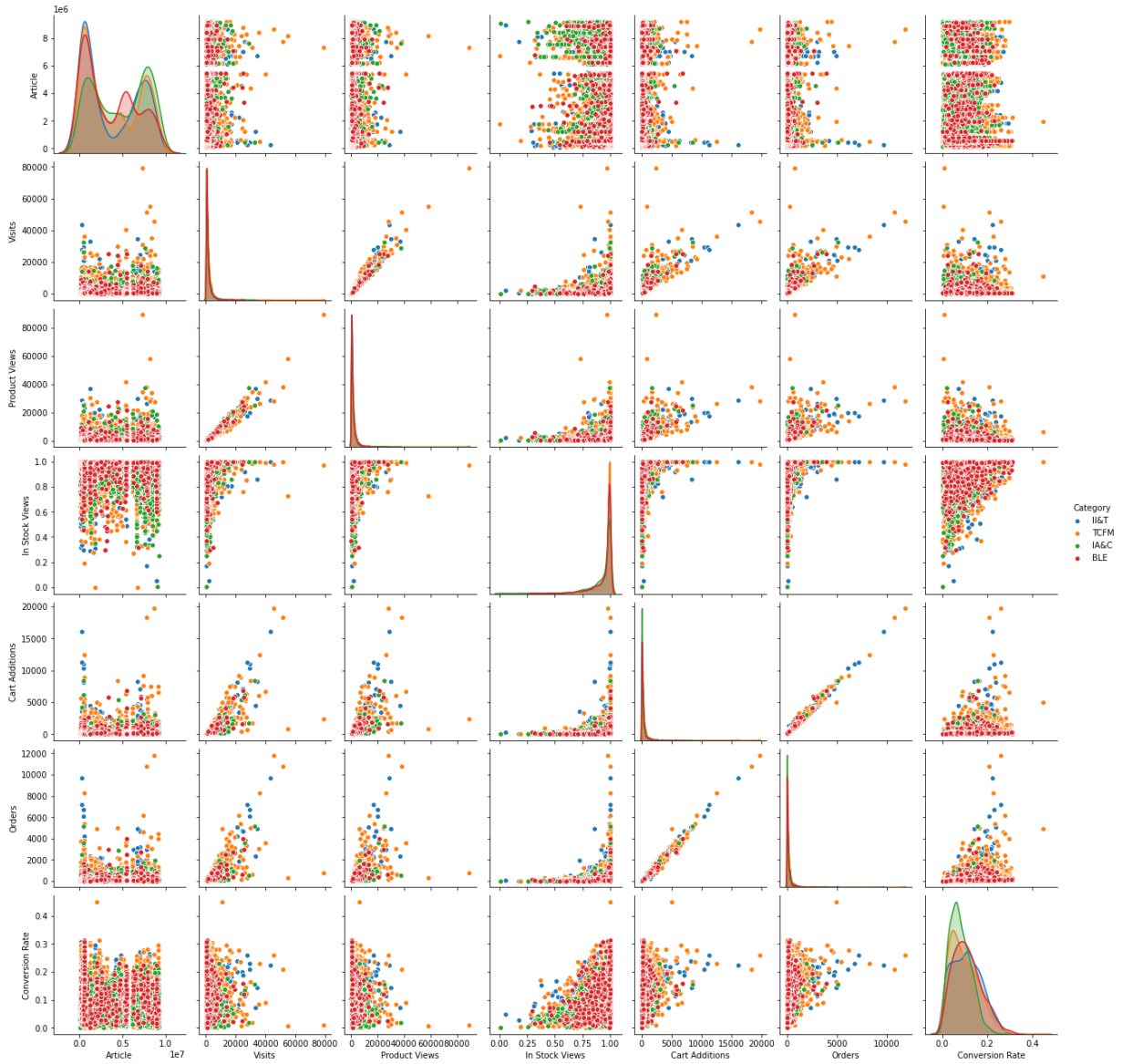
Out[24]:
```
TCFM    5768
II&T    2730
IA&C    1998
BLE      961
Name: Category, dtype: int64
```

In [25]:
```
1 best_sellhidwithoutOk.shape
```

Out[25]:
```
(11457, 12)
```

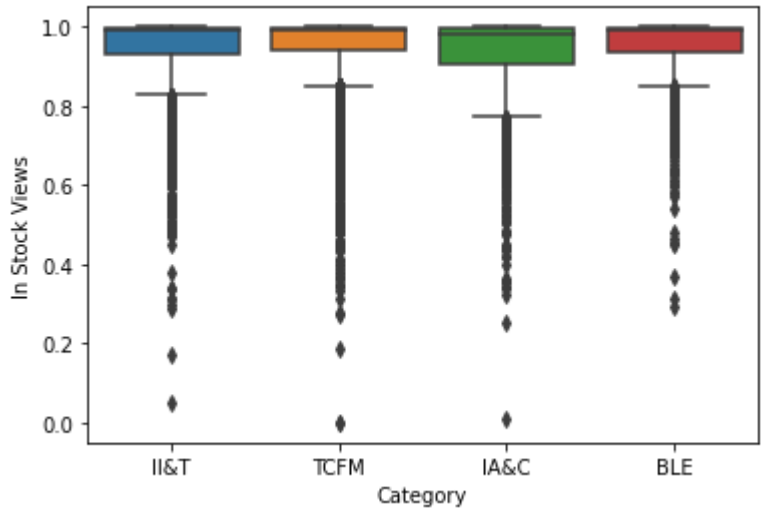
In [26]:
```
1 #Trying this again - this time with less categories (without Okdo):
2 pair2 = sns.pairplot(best_sellhidwithoutOk, hue="Category")
3 pair2
4 # Run this cell twice if you have a warning message to see the plot
```

Out[26]:
```
<seaborn.axisgrid.PairGrid at 0x1d6db6a9ec8>
```



It's still too much overlap but we can't drop any more categories - they have too many records.

In [27]:
```
# box plot of In Stock Views based on Category:
sns.boxplot(y = 'In Stock Views', x = 'Category', data = best_sellhidwithout
plt.show()
```



It appears we stock BLE best as observations don't reach the bottom of the graph.

In [28]:
```
# Assigning dataframe new name - shorter:
data = best_sellhidwithoutOk
```

In [29]:
```
data.sample(3)
```

Out[29]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views
5420	8117689	PE SPIRAL WRAP 15MM BLACK 10M	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	3286	3022	0.813
8407	7887144	TYPE K IMMERSION PROBE	9999-12-31 00:00:00	II&T	Test & Measurement	Electrical T&M	1502	1733	0.989
4526	2830746	5 WAY DOUBLE SCREW EARTH TERMINAL BLOCK	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Cable Mgt & Elec Acc	1062	1042	0.968

Preparing dataframe for modelling:

1) Since we don't have any binary type data, we will need to create new column(s) with dychotomy split. It could sense to look at key statistical measures and use one of them as a cut-off value to cut dataset in 2...

In [30]:
```
1 data['Product Views'].describe()
```

Out[30]:
```
count    11457.000000
mean      1946.953915
std       2752.089032
min          0.000000
25%        632.000000
50%       1153.000000
75%       2245.000000
max      89157.000000
Name: Product Views, dtype: float64
```

We can see that mean value of the distribution is 1946 (that's how many times a product among Top Sellers was viewed) so it could be an option to consider...

In [31]:
```
1 data['Cart Additions'].describe()
```

Out[31]:
```
count    11457.000000
mean       361.568561
std        657.753388
min          0.000000
25%         95.000000
50%        195.000000
75%        390.000000
max      19733.000000
Name: Cart Additions, dtype: float64
```

For Cart Additions the average value is 361.

Important: We can see (from "min" value) in both columns examined above that value of 0 (zero) makes an appearance. It's only few instances but it will cause problems down the line (in fact it did - producing infinity error in my first attempts) so they have to go.

In [32]:
```
1 #Filtering out zero value sfrom columns Product Views and Cart Additions:
2 nonzero = data[~((data['Product Views'] == 0) | (data['Cart Additions'] == 0
```

In [33]:
```
1 nonzero.head()
```

Out[33]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views
0	233487	BLACK CABLE TIE, 300X4.8MM, PACK 100	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	43371	28655	0.996
1	5375488	RS SEALED LEAD-ACID BATTERY,12V 7AH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	40171	41566	0.992
2	7348885	88PC 1/2IN. SOCKET & VDE TOOL KIT	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	79286	89157	0.970
3	1681644	DUST REMOVER CLEANER 400ML	9999-12-31 00:00:00	TCFM	Consumables & Safety	Adhesives, Clean & M	8704	8340	0.933
4	6191506	SEALING STRIP,PVC,WIRE INSERT,1-2,9X6.	9999-12-31 00:00:00	TCFM	Consumables & Safety	Fasteners	9605	9731	0.997

In [34]:
```
1 #checking the shape of thus filtered out dataframe:
2 nonzero.shape
```

Out[34]: (11454, 12)

We can see - comparing to result of .shape function used earlier (11457,12) that we've only dropped 3 additional rows - minuscule amount of data.

In [35]:
```
#Double-checking if zeros are truly gone, by sorting Product Views in ascend
nonzero.sort_values(by=['Product Views']).head()
```

Out[35]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	I Stoc View
7440	521358	INDUSTRIAL LOCKER 1 DOOR RED 450MM	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Acc, Stor & Mat Hand	40	40	0.97
4353	212547	U/BRIGHT RED LED BLK CHR RECESSED,12VDC	9999-12-31 00:00:00	II&T	EMECH	Indicators	45	47	1.00
7386	519469	CLAD EXTENSION BAY,500MM 24KG	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Acc, Stor & Mat Hand	48	48	0.78
4278	212171	U/BRIGHT GRN LED BLK CHR RECESSED,24VDC	9999-12-31 00:00:00	II&T	EMECH	Indicators	77	58	0.93
10605	7033123	110V 40W/M CONSTANT WATT HEATER TAPE-50M	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Pwr & Ind Cable	62	61	0.95

2)...I came up with a different approach, though. Why not compare 2 variables as a ratio and then split data in 2 according to the mean value of such ratio:

In [36]:
```
nonzero['CartAdd_to_ProductView'] = nonzero['Cart Additions']/nonzero['Produ
```

```
C:\Users\oszer\anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  """Entry point for launching an IPython kernel.
```

This is SettingWithCopyWarning - basically pandas advising me I'm working on a copy of a dataframe and not original one. In this case it's fine - that's what I want to do.

In [37]:
```
1 nonzero.head()
```

Out[37]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views
0	233487	BLACK CABLE TIE, 300X4.8MM, PACK 100	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	43371	28655	0.996
1	5375488	RS SEALED LEAD-ACID BATTERY,12V 7AH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	40171	41566	0.992
2	7348885	88PC 1/2IN. SOCKET & VDE TOOL KIT	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	79286	89157	0.970
3	1681644	DUST REMOVER CLEANER 400ML	9999-12-31 00:00:00	TCFM	Consumables & Safety	Adhesives, Clean & M	8704	8340	0.933
4	6191506	SEALING STRIP,PVC,WIRE INSERT,1-2,9X6.	9999-12-31 00:00:00	TCFM	Consumables & Safety	Fasteners	9605	9731	0.997

Now we can calculate the average (mean) of the newly created column:

In [38]:
```
1 # mean calculation of CartAdd_to_ProductView:
2 nonzero['CartAdd_to_ProductView'].mean()
```

Out[38]:
```
0.20035606227139058
```

In [39]:
```
1 # Checking stats of newly created ratio:
2 nonzero['CartAdd_to_ProductView'].describe()
```

Out[39]:
```
count    11454.000000
mean         0.200356
std          0.130551
min          0.011609
25%          0.102109
50%          0.171446
75%          0.267959
max          0.986318
Name: CartAdd_to_ProductView, dtype: float64
```

We can see that the average ratio of Cart Addition(s) to Product View(s) is around **20% - or 1:5**, which intuitively feels right.

I'm curious, though to find out what that one product with super high ratio of over 98% is... Let's find out:

In [40]:
```
max = nonzero[nonzero['CartAdd_to_ProductView'] > 0.9863]
```

In [41]:
```
max.head()
```

Out[41]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views
272	9176588	RS ALKALINE AA 20 PK	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	13527	6651	0.995

it's the batteries. Looks like clients really like them and once they view them, they virtually always add them to their carts.

I can now create a new column, where there will be indication if our CartAdd_to_ProductView ratio is above or below the average.

In [42]:
```
#Creating and assigning new variable for our average:
a_mean = nonzero['CartAdd_to_ProductView'].mean()
a_mean
```

Out[42]:
```
0.20035606227139058
```

In [43]:
```
#Using lambda function for comparison to mean and creating new column with b
nonzero['aboveMean'] = nonzero['CartAdd_to_ProductView'].apply(lambda x: x >
```

```
C:\Users\oszer\anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

In [44]:
```
1 nonzero.head()
```

Out[44]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views
0	233487	BLACK CABLE TIE, 300X4.8MM, PACK 100	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	43371	28655	0.996
1	5375488	RS SEALED LEAD-ACID BATTERY,12V 7AH	9999-12-31 00:00:00	TCFM	Consumables & Safety	Battery	40171	41566	0.992
2	7348885	88PC 1/2IN. SOCKET & VDE TOOL KIT	9999-12-31 00:00:00	TCFM	Tools Storage & FM	Hand Tools	79286	89157	0.970
3	1681644	DUST REMOVER CLEANER 400ML	9999-12-31 00:00:00	TCFM	Consumables & Safety	Adhesives, Clean & M	8704	8340	0.933
4	6191506	SEALING STRIP,PVC,WIRE INSERT,1-2,9X6.	9999-12-31 00:00:00	TCFM	Consumables & Safety	Fasteners	9605	9731	0.997

We now have a new column with binary values and we can treat it as a class for predictions.

Correlations:

In [45]:
```
1 #Assigning new name to the dataframe:
2 dataset = nonzero
```

In [46]:
```
1 dataset.sample(3)
```

Out[46]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views	Ad
6042	7840085	G 1/2 FRL PORTABLE AIR CRADLE	9999-12-31 00:00:00	IA&C	Mech & Flu Pwr	Pneumatics	716	807	0.791	
8666	7004523	RED HEATSHRINK TUBE 24/8MM I/D	9999-12-31 00:00:00	II&T	Cable, Anti-Vib & Ac	Cable Accs	934	744	0.999	
9757	2512224	YELLOW SEALED LOW PROFILE FOOT SWITCH,3A	9999-12-31 00:00:00	IA&C	Sensors & Switches	Push But & Swit	302	321	0.994	

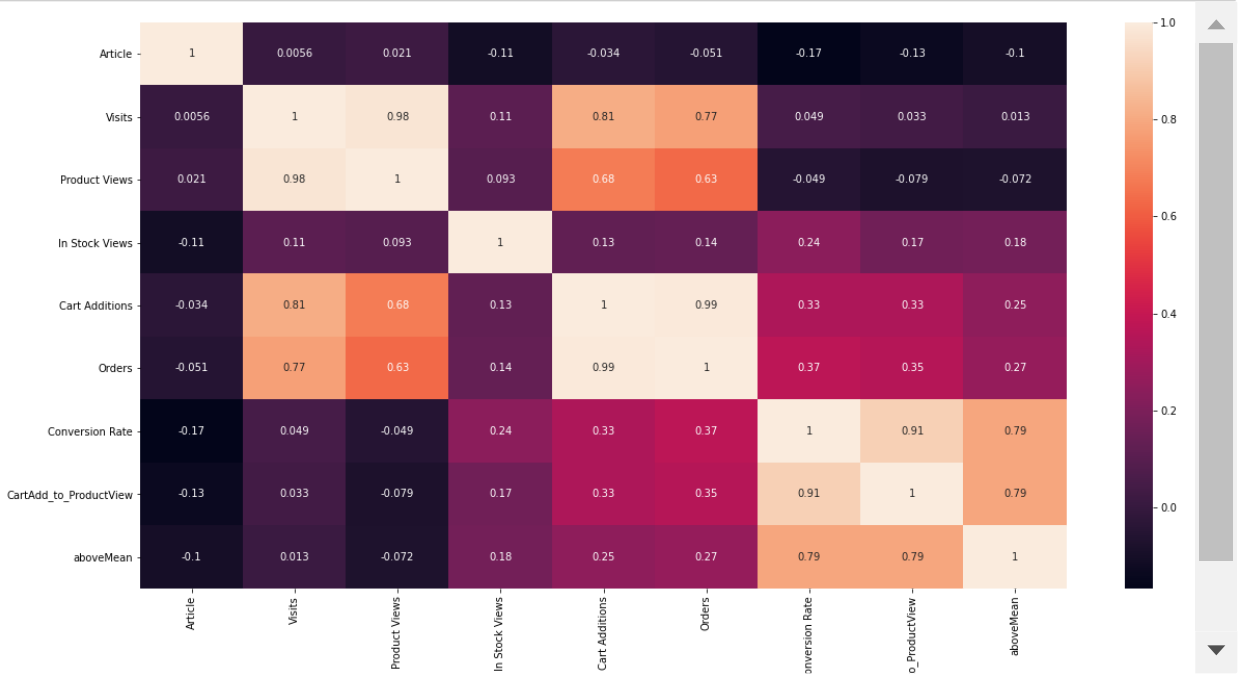
We can now check correlations:

In [47]:
```
1 dataset.corr()
```

Out[47]:

	Article	Visits	Product Views	In Stock Views	Cart Additions	Orders	Conversio Rat
Article	1.000000	0.005626	0.021034	-0.110331	-0.033665	-0.050572	-0.16737
Visits	0.005626	1.000000	0.977554	0.106543	0.808881	0.772378	0.04915
Product Views	0.021034	0.977554	1.000000	0.092866	0.677227	0.633791	-0.04917
In Stock Views	-0.110331	0.106543	0.092866	1.000000	0.126527	0.135813	0.24137
Cart Additions	-0.033665	0.808881	0.677227	0.126527	1.000000	0.991103	0.33049
Orders	-0.050572	0.772378	0.633791	0.135813	0.991103	1.000000	0.37159
Conversion Rate	-0.167378	0.049156	-0.049179	0.241371	0.330492	0.371596	1.00000
CartAdd_to_ProductView	-0.132046	0.033293	-0.079390	0.169494	0.332880	0.354950	0.91464
aboveMean	-0.101098	0.013253	-0.071804	0.176931	0.248793	0.266624	0.79104

In [48]:
```
#Plotting correlations above as a seaborn heatmap:
plt.figure(figsize = (20, 10))
fig1 = sns.heatmap(dataset.corr(), annot = True)
fig1.set_xlabel("Correlations", fontsize=20)
plt.show()
```



In [49]:
```
#let's remind ourselves of the structure of dataframe:
dataset.sample()
```

Out[49]:

	Article	Description	FSTD Global	Category	Cell	Technology	Visits	Product Views	In Stock Views	A
2916	7755362	MILD STEEL IP66 WALL BOX, 800X600X300MM	9999-12-31 00:00:00	IA&C	Mach Guard & Enc	Enclosures	992	1049	0.827	

Since we will need to convert categorical columns' values into numerical, we need to get rid of mostly useless values in 2 columns: Description and FSTD Global. They don't help us much but make next step very time-consuming.

In [50]:
```
dataset.drop(['Description', 'FSTD Global'], axis=1, inplace=True)
```

```
C:\Users\oszer\anaconda3\lib\site-packages\pandas\core\frame.py:3997: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  errors=errors,
```

In [51]:
```
1 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11454 entries, 0 to 11457
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Article                11454 non-null  int64
 1   Category               11454 non-null  object
 2   Cell                   11454 non-null  object
 3   Technology             11454 non-null  object
 4   Visits                 11454 non-null  int64
 5   Product Views          11454 non-null  int64
 6   In Stock Views         11454 non-null  float64
 7   Cart Additions         11454 non-null  int64
 8   Orders                 11454 non-null  int64
 9   Conversion Rate        11454 non-null  float64
 10  CartAdd_to_ProductView  11454 non-null  float64
 11  aboveMean              11454 non-null  bool
dtypes: bool(1), float64(3), int64(5), object(3)
memory usage: 1.1+ MB
```

Dummy data (variables):

To evaluate the correlation between categorical variables it is necessary to first perform one-hot encoding to create numerical dummy variables - one binary variable for each level of the original categorical variable.

In [52]:
```
# Generating new data set with dummy variables:
data_dummy = pd.get_dummies(dataset)

data_dummy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11454 entries, 0 to 11457
Data columns (total 94 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Article                          11454 non-null  int64
 1   Visits                           11454 non-null  int64
 2   Product Views                    11454 non-null  int64
 3   In Stock Views                   11454 non-null  float64
 4   Cart Additions                   11454 non-null  int64
 5   Orders                           11454 non-null  int64
 6   Conversion Rate                  11454 non-null  float64
 7   CartAdd_to_ProductView           11454 non-null  float64
 8   aboveMean                        11454 non-null  bool
 9   Category_BLE                     11454 non-null  uint8
 10  Category_IA&C                    11454 non-null  uint8
 11  Category_II&T                    11454 non-null  uint8
 12  Category_TCFM                    11454 non-null  uint8
 13  Cell_Analog Disc Power           11454 non-null  uint8
 14  Cell_BLE E-Mech                  11454 non-null  uint8
 15  Cell_Cable, Anti-Vib & Ac        11454 non-null  uint8
 16  Cell_Consumables & Safety        11454 non-null  uint8
 17  Cell_E-Conn                      11454 non-null  uint8
 18  Cell_EMECH                       11454 non-null  uint8
 19  Cell_I-Conn                      11454 non-null  uint8
 20  Cell_Mach Guard & Enc            11454 non-null  uint8
 21  Cell_Mech & Flu Pwr              11454 non-null  uint8
 22  Cell_Opto                        11454 non-null  uint8
 23  Cell_Passives                    11454 non-null  uint8
 24  Cell_Process Control             11454 non-null  uint8
 25  Cell_Sensors & Switches          11454 non-null  uint8
 26  Cell_Test & Measurement          11454 non-null  uint8
 27  Cell_Tools Storage & FM          11454 non-null  uint8
 28  Technology_3D Printing           11454 non-null  uint8
 29  Technology_Acc, Stor & Mat Hand  11454 non-null  uint8
 30  Technology_Adhesives, Clean & M  11454 non-null  uint8
 31  Technology_Anti-Vibration        11454 non-null  uint8
 32  Technology_BLE E-Mech Tech       11454 non-null  uint8
 33  Technology_Battery               11454 non-null  uint8
 34  Technology_Cable Accs            11454 non-null  uint8
 35  Technology_Cable Mgt & Elec Acc  11454 non-null  uint8
 36  Technology_Capacitors            11454 non-null  uint8
 37  Technology_Circuit Protection    11454 non-null  uint8
 38  Technology_Contactors            11454 non-null  uint8
 39  Technology_Crimp Term & Power    11454 non-null  uint8
 40  Technology_Ctrl & Ind Relay      11454 non-null  uint8
 41  Technology_DIN Rail & Term       11454 non-null  uint8
 42  Technology_DIN Rail PSU's        11454 non-null  uint8
 43  Technology_Displays              11454 non-null  uint8
 44  Technology_Elec & Data Cable     11454 non-null  uint8
 45  Technology_Electrical T&M        11454 non-null  uint8
 46  Technology_Electronic T&M        11454 non-null  uint8
```

```
 47  Technology_Enclosures            11454 non-null  uint8
 48  Technology_Engineering  Mat      11454 non-null  uint8
 49  Technology_Fact Auto Sen         11454 non-null  uint8
 50  Technology_Fasteners             11454 non-null  uint8
 51  Technology_Fuses                 11454 non-null  uint8
 52  Technology_HVAC                  11454 non-null  uint8
 53  Technology_Hand Tools            11454 non-null  uint8
 54  Technology_Heavy Duty Conn       11454 non-null  uint8
 55  Technology_Hose & Pumps          11454 non-null  uint8
 56  Technology_Hydraulics            11454 non-null  uint8
 57  Technology_IT                    11454 non-null  uint8
 58  Technology_Ind & AV Conns        11454 non-null  uint8
 59  Technology_Indicators            11454 non-null  uint8
 60  Technology_Inductors             11454 non-null  uint8
 61  Technology_LED                   11454 non-null  uint8
 62  Technology_Level & Flow Con      11454 non-null  uint8
 63  Technology_Lighting              11454 non-null  uint8
 64  Technology_MIL Ind & Test Conns  11454 non-null  uint8
 65  Technology_Machine Safety        11454 non-null  uint8
 66  Technology_Measurement & Insp    11454 non-null  uint8
 67  Technology_Motors & Control      11454 non-null  uint8
 68  Technology_PCB Connectors        11454 non-null  uint8
 69  Technology_PLC's & HMI's         11454 non-null  uint8
 70  Technology_PPE                   11454 non-null  uint8
 71  Technology_Plumbing              11454 non-null  uint8
 72  Technology_Pneumatics            11454 non-null  uint8
 73  Technology_Pow Tools & Abras     11454 non-null  uint8
 74  Technology_Pow Tran & Struc      11454 non-null  uint8
 75  Technology_Power Management      11454 non-null  uint8
 76  Technology_Power PCB             11454 non-null  uint8
 77  Technology_Printing & Office     11454 non-null  uint8
 78  Technology_Proc Auto Sen         11454 non-null  uint8
 79  Technology_Push But & Swit       11454 non-null  uint8
 80  Technology_Pwr & Ind Cable       11454 non-null  uint8
 81  Technology_RF Datacomm Conns     11454 non-null  uint8
 82  Technology_Rect Connectors       11454 non-null  uint8
 83  Technology_Relays-EMECH          11454 non-null  uint8
 84  Technology_Resistors             11454 non-null  uint8
 85  Technology_Security              11454 non-null  uint8
 86  Technology_Sensors               11454 non-null  uint8
 87  Technology_Site Safety & Janit   11454 non-null  uint8
 88  Technology_Soldering & ESD       11454 non-null  uint8
 89  Technology_Sounders/Beacons      11454 non-null  uint8
 90  Technology_Switches-EMECH        11454 non-null  uint8
 91  Technology_Temp Control          11454 non-null  uint8
 92  Technology_Thermal Management    11454 non-null  uint8
 93  Technology_Timers, Counters      11454 non-null  uint8
dtypes: bool(1), float64(3), int64(5), uint8(85)
memory usage: 1.7 MB
```

We can see now that one-hot encoding resulted in creation of additional columns.

In [53]:
```
1 data_dummy.head()
```

Out[53]:

	Article	Visits	Product Views	In Stock Views	Cart Additions	Orders	Conversion Rate	CartAdd_to_ProductView	abov
0	233487	43371	28655	0.996	16065	9658	0.2227	0.560635	
1	5375488	40171	41566	0.992	6666	3611	0.0899	0.160371	
2	7348885	79286	89157	0.970	2373	789	0.0100	0.026616	
3	1681644	8704	8340	0.933	2245	1517	0.1743	0.269185	
4	6191506	9605	9731	0.997	2152	1556	0.1620	0.221149	

5 rows × 94 columns

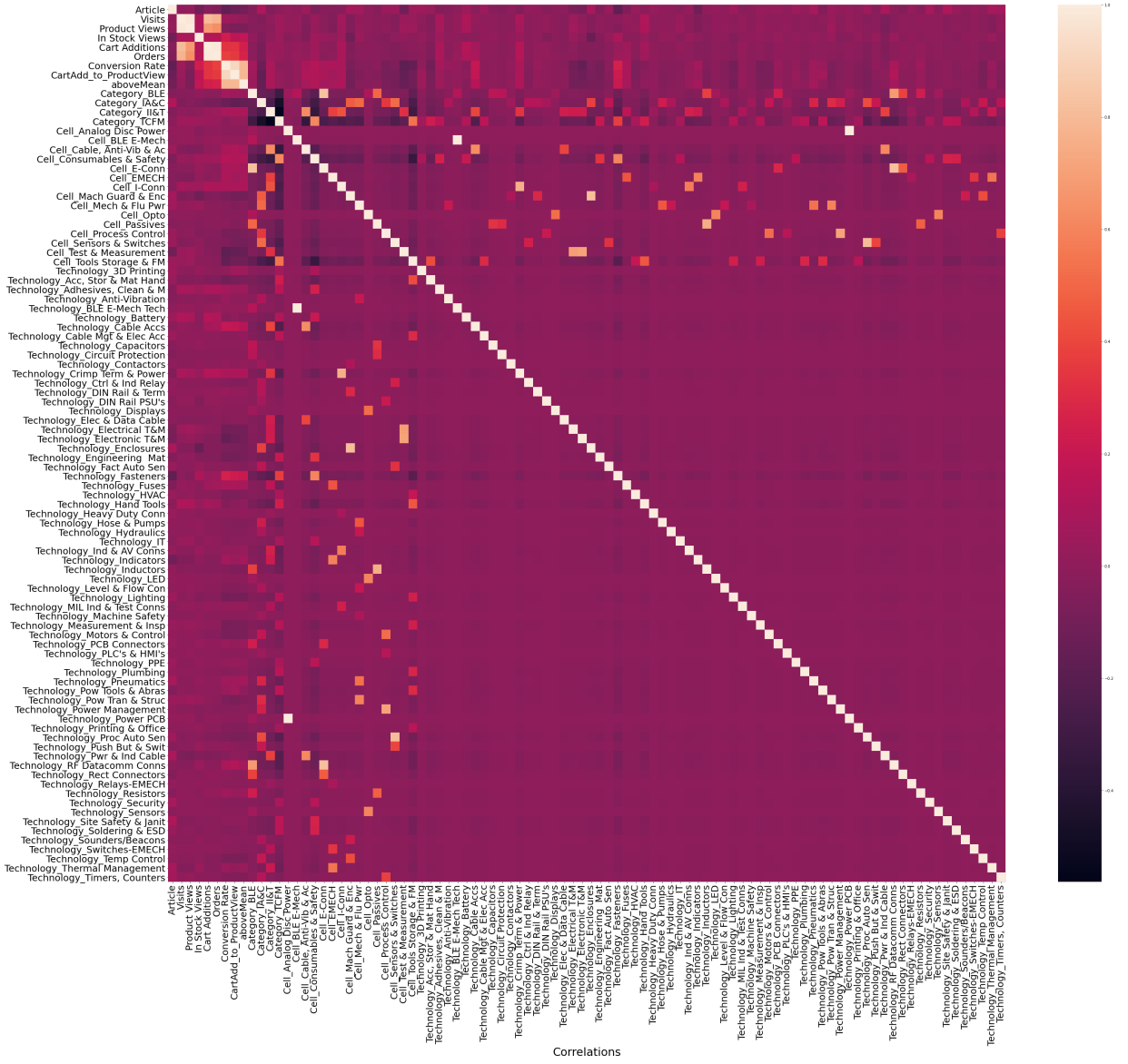
Correlation matrix

We can now calculate the correlation coefficients again using the function `.corr` as before:

In [54]:
```
1 cor_matrix = data_dummy.corr()
```

Since it's huge, printing this matrix wouldn't make much sense. Let's try to see the whole picture by displaying it as a heatmap:

In [55]:
```
plt.figure(figsize=(50,43))
fig1=sns.heatmap(cor_matrix)
fig1.set_xlabel("Correlations", fontsize=30)
fig1.tick_params(labelsize = 25)
```



Since the scale on the right hand side is difficult to read, it's worth pointing out that the scale extremes are from -1 (light color) to +1 (the darkest coloring).

Along the diagonal there are a couple of small squares of fully white and black cells - this is just due to the dummy variables we produced, where the generated features representing different values (e.g. "above average") will always have perfect anti-correlation. In the rest of the heatmap, there are some non-zero correlations too however. Let's explore these more thoroughly.

Calculating *r*-coefficients and *p*-values

The important thing to determine for us is, not just what the correlation coefficient is, but whether a correlation is statistically significant or not i.e. not just due to chance.

We can assess this using the Pearson correlation test. This calculates the p-value, which recall is the probability of observing the current result (in this case the r coefficient) given that there is in fact no significant correlation i.e. the result is due to chance. If that probability is tiny, then the correlation is indeed significant and it is not due to chance. Note that the p-value is not related to the strength of the correlation as measured by the r-coefficient - it only tells us whether a correlation is due to chance or not. Sometimes you can observe r-coefficients very close to zero but with p-values that suggest that the correlation is not due to chance, whereas in other cases r-coefficients can be very large in magnitude (close to -1 or 1), but with non-significant p-values.

Time to import the library `researchpy` which will help a lot in our correlation analysis - in particular, the functions `corr_case` and `corr_pair` :

In [56]:
```
import researchpy as rp

corr_type, corr_matrix, corr_ps = rp.corr_case(data_dummy)
```

The function `.corr_case` generates 3 matrices - one about the type of correlation, the 2nd matrix showing the r-coefficient, and the last showing the p-value of the correlation.

In [57]:
```
# Running the last subfunction (i.e. matrix) of .corr_case - corr_ps - showi
corr_ps.head(13)
```

Out[57]:

	Article	Visits	Product Views	In Stock Views	Cart Additions	Orders	Conversion Rate	CartA
Article	0.0000	0.5471	0.0244	0.0000	0.0003	0.0000	0.0000	
Visits	0.5471	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
Product Views	0.0244	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
In Stock Views	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
Cart Additions	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
Orders	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
Conversion Rate	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
CartAdd_to_ProductView	0.0000	0.0004	0.0000	0.0000	0.0000	0.0000	0.0000	
aboveMean	0.0000	0.1561	0.0000	0.0000	0.0000	0.0000	0.0000	
Category_BLE	0.0000	0.0009	0.0014	0.3254	0.1292	0.0820	0.0000	

In [58]:
```
# 2nd matrix of .corr_case - type of correlation:
corr_type.head()
```

Out[58]:

	Pearson correlation test using list-wise deletion
0	Total observations used = 11454

In [59]:
```
# Checking the 2nd matrix that shows the r-coefficient:
corr_matrix.head(15)
```

Out[59]:

	Article	Visits	Product Views	In Stock Views	Cart Additions	Orders	Conversion Rate	CartAd
Article	1	0.0056	0.021	-0.1103	-0.0337	-0.0506	-0.1674	
Visits	0.0056	1	0.9776	0.1065	0.8089	0.7724	0.0492	
Product Views	0.021	0.9776	1	0.0929	0.6772	0.6338	-0.0492	
In Stock Views	-0.1103	0.1065	0.0929	1	0.1265	0.1358	0.2414	
Cart Additions	-0.0337	0.8089	0.6772	0.1265	1	0.9911	0.3305	
Orders	-0.0506	0.7724	0.6338	0.1358	0.9911	1	0.3716	
Conversion Rate	-0.1674	0.0492	-0.0492	0.2414	0.3305	0.3716	1	
CartAdd_to_ProductView	-0.132	0.0333	-0.0794	0.1695	0.3329	0.3549	0.9146	
aboveMean	-0.1011	0.0133	-0.0718	0.1769	0.2488	0.2666	0.791	
Category_BLE	-0.0403	-0.031	-0.0298	0.0092	-0.0142	-0.0162	0.0659	
Category_IA&C	0.1137	-0.0527	-0.0332	-0.0571	-0.0826	-0.0819	-0.132	
Category_II&T	-0.0455	-0.0006	-0.0154	-0.007	0.0409	0.0451	0.09	
Category_TCFM	-0.0252	0.0576	0.0548	0.0442	0.0357	0.0328	-0.013	
Cell_Analog Disc Power	0.0145	0.0162	0.0216	-0.0314	0.0014	-0	-0.0157	
Cell_BLE E-Mech	-0.0052	0.0181	0.0179	-0.0049	0.0228	0.0236	0.0223	

15 rows × 94 columns

This is what researchpy documentation says about .corr_pair function:

"Conducts Pearson (default method), Spearman rank, or Kendall's Tau-b correlation analysis using pair wise deletion. Returns the relevant information and results in 1 DataFrame for easy exporting".

In [60]:
```
corr_table = rp.corr_pair(data_dummy)
```

In [61]:
```
# Checking what that correlations' table looks like:
corr_table.head(20)
```

Out[61]:

	r value	p-value	N
Article & Visits	0.0056	0.5471	11454
Article & Product Views	0.0210	0.0244	11454
Article & In Stock Views	-0.1103	0.0000	11454
Article & Cart Additions	-0.0337	0.0003	11454
Article & Orders	-0.0506	0.0000	11454
Article & Conversion Rate	-0.1674	0.0000	11454
Article & CartAdd_to_ProductView	-0.1320	0.0000	11454
Article & aboveMean	-0.1011	0.0000	11454
Article & Category_BLE	-0.0403	0.0000	11454
Article & Category_IA&C	0.1137	0.0000	11454
Article & Category_II&T	-0.0455	0.0000	11454
Article & Category_TCFM	-0.0252	0.0069	11454
Article & Cell_Analog Disc Power	0.0145	0.1212	11454
Article & Cell_BLE E-Mech	-0.0052	0.5745	11454
Article & Cell_Cable, Anti-Vib & Ac	0.0435	0.0000	11454
Article & Cell_Consumables & Safety	-0.0678	0.0000	11454
Article & Cell_E-Conn	-0.0292	0.0018	11454
Article & Cell_EMECH	-0.0343	0.0002	11454
Article & Cell_I-Conn	-0.0292	0.0018	11454
Article & Cell_Mach Guard & Enc	0.0711	0.0000	11454

In [62]:
```
corr_table.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4371 entries, Article & Visits to Technology_Thermal Management & Techno
logy_Timers, Counters
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   r value  4371 non-null   object
 1   p-value  4371 non-null   object
 2   N        4371 non-null   int64
dtypes: int64(1), object(2)
memory usage: 296.6+ KB
```

In [63]:
```
# Let's sort now by r value (correlation coefficient):
corr_table.sort_values(by=['r value'], ascending=False).head(21)
```

Out[63]:

	r value	p-value	N
Cell_Analog Disc Power & Technology_Power PCB	1.0000	0.0000	11454
Cell_BLE E-Mech & Technology_BLE E-Mech Tech	1.0000	0.0000	11454
Cart Additions & Orders	0.9911	0.0000	11454
Visits & Product Views	0.9776	0.0000	11454
Conversion Rate & CartAdd_to_ProductView	0.9146	0.0000	11454
Category_BLE & Cell_E-Conn	0.8282	0.0000	11454
Cell_E-Conn & Technology_RF Datacomm Conns	0.8217	0.0000	11454
Cell_Mach Guard & Enc & Technology_Enclosures	0.8119	0.0000	11454
Visits & Cart Additions	0.8089	0.0000	11454
Cell_Sensors & Switches & Technology_Proc Auto Sen	0.8062	0.0000	11454
CartAdd_to_ProductView & aboveMean	0.7916	0.0000	11454
Conversion Rate & aboveMean	0.7910	0.0000	11454
Visits & Orders	0.7724	0.0000	11454
Cell_I-Conn & Technology_Crimp Term & Power	0.7560	0.0000	11454
Cell_Passives & Technology_Inductors	0.7497	0.0000	11454
Cell_Test & Measurement & Technology_Electronic T&M	0.7124	0.0000	11454
Cell_Process Control & Technology_Power Management	0.7075	0.0000	11454
Cell_Test & Measurement & Technology_Electrical T&M	0.6904	0.0000	11454
Category_BLE & Technology_RF Datacomm Conns	0.6805	0.0000	11454
Product Views & Cart Additions	0.6772	0.0000	11454
Product Views & Orders	0.6338	0.0000	11454

We can see that Cart Additions & Orders have **r value (correlation coefficient)** close to 1 (0.9911), denoting very strong positive linear relationship, which means in practical terms that act of adding a product to the shopping cart almost always means ordering (buying) that product.

Product Views & Cart Additions correlation coefficient value of 0.6772 also shows us reatively strong positive linear relationship - it tells us that on typically with a product viewed there is 67.72% chance that it will be added to the cart.

It bears observing that the r-coefficient and p-value are not numeric values by default. We can use pd.to_numeric to rectify this however:

In [64]:
```
corr_table['p-value'] = pd.to_numeric(corr_table['p-value'])
corr_table['r value'] = pd.to_numeric(corr_table['r value'])
```

In [65]:
```
# Checking data types again:
corr_table.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4371 entries, Article & Visits to Technology_Thermal Management & Techno
logy_Timers, Counters
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   r value  4371 non-null   float64
 1   p-value  4371 non-null   float64
 2   N        4371 non-null   int64
dtypes: float64(2), int64(1)
memory usage: 296.6+ KB
```

We can see now that r and p - values have been converted to numbers.

In [66]:
```
corr_table.tail(4)
```

Out[66]:

	r value	p-value	N
Technology_Switches-EMECH & Technology_Timers, Counters	-0.0044	0.6393	11454
Technology_Temp Control & Technology_Thermal Management	-0.0104	0.2667	11454
Technology_Temp Control & Technology_Timers, Counters	-0.0055	0.5566	11454
Technology_Thermal Management & Technology_Timers, Counters	-0.0071	0.4462	11454

Now let's look at only those correlations that are high (positive or negative) and significant:

In [67]:
```
#Searching for correlations that are high positive and significant:
corr_table[(corr_table['r value'] > 0.5) & (corr_table['p-value'] < 0.05)]
```

Out[67]:

	r value	p-value	N
Visits & Product Views	0.9776	0.0	11454
Visits & Cart Additions	0.8089	0.0	11454
Visits & Orders	0.7724	0.0	11454
Product Views & Cart Additions	0.6772	0.0	11454
Product Views & Orders	0.6338	0.0	11454
Cart Additions & Orders	0.9911	0.0	11454
Conversion Rate & CartAdd_to_ProductView	0.9146	0.0	11454
Conversion Rate & aboveMean	0.7910	0.0	11454
CartAdd_to_ProductView & aboveMean	0.7916	0.0	11454
Category_BLE & Cell_E-Conn	0.8282	0.0	11454
Category_BLE & Technology_RF Datacomm Conns	0.6805	0.0	11454
Category_IA&C & Cell_Mech & Flu Pwr	0.5070	0.0	11454
Category_IA&C & Cell_Sensors & Switches	0.5173	0.0	11454
Category_II&T & Cell_Cable, Anti-Vib & Ac	0.6214	0.0	11454
Category_TCFM & Cell_Consumables & Safety	0.6129	0.0	11454
Category_TCFM & Cell_Tools Storage & FM	0.5393	0.0	11454
Cell_Analog Disc Power & Technology_Power PCB	1.0000	0.0	11454
Cell_BLE E-Mech & Technology_BLE E-Mech Tech	1.0000	0.0	11454
Cell_Cable, Anti-Vib & Ac & Technology_Cable Accs	0.6323	0.0	11454
Cell_Cable, Anti-Vib & Ac & Technology_Pwr & Ind Cable	0.6146	0.0	11454
Cell_Consumables & Safety & Technology_Fasteners	0.5989	0.0	11454
Cell_E-Conn & Technology_RF Datacomm Conns	0.8217	0.0	11454
Cell_EMECH & Technology_Indicators	0.5903	0.0	11454
Cell_EMECH & Technology_Thermal Management	0.5480	0.0	11454
Cell_I-Conn & Technology_Crimp Term & Power	0.7560	0.0	11454
Cell_I-Conn & Technology_Ind & AV Conns	0.5733	0.0	11454
Cell_Mach Guard & Enc & Technology_Enclosures	0.8119	0.0	11454
Cell_Mech & Flu Pwr & Technology_Pneumatics	0.5314	0.0	11454
Cell_Mech & Flu Pwr & Technology_Pow Tran & Struc	0.5949	0.0	11454
Cell_Opto & Technology_Displays	0.5221	0.0	11454
Cell_Opto & Technology_LED	0.6028	0.0	11454
Cell_Opto & Technology_Sensors	0.6028	0.0	11454
Cell_Passives & Technology_Inductors	0.7497	0.0	11454
Cell_Process Control & Technology_Motors & Control	0.5236	0.0	11454

	r value	p-value	N
Cell_Process Control & Technology_Power Management	0.7075	0.0	11454
Cell_Sensors & Switches & Technology_Proc Auto Sen	0.8062	0.0	11454
Cell_Test & Measurement & Technology_Electrical T&M	0.6904	0.0	11454
Cell_Test & Measurement & Technology_Electronic T&M	0.7124	0.0	11454

In [68]:
```
#Searching for correlations that are high negative and significant:
corr_table[(corr_table['r value']  < -0.45)  & (corr_table['p-value'] < 0.05
```

Out[68]:

	r value	p-value	N
Category_IA&C & Category_TCFM	-0.4627	0.0	11454
Category_II&T & Category_TCFM	-0.5631	0.0	11454

Highly correlated features can cause lower performance in certain classification algorithms (though non-parametric models such as decision trees/random forests are not so badly affected) because the feature redunancy, relative to how much data there is, might increase the chance of overfitting. As such, it can sometimes be useful to remove some of the correlated features, or use dimensionality reduction techniques like **Principal Component Analysis (PCA) - which I have used in my Clustering projects**. We will leave the dataset as it is for now, and see how our model performs.

Model & Evaluate

During the Classification workshop we've evaluated a classification model (i.e. a Decision Tree) - below we are going to follow similar steps, but in this case we will use an "ensemble method" known as Random Forests. As the name suggests, these are closely related to Decision Trees.

Random Forests are an example of an "ensemble" method. In essence, a Random Forest is simply a "forest" (collection or ensemble) of multiple decision trees that each use a different random subsample of features - within this, each decision tree "votes" for the best solution, and the average is taken as the final prediction.

First - as is an accepted custom - let's separate the features from the class. SInce typing them up one by one would've been way too time-consumming, I'm gonna "cheat" a bit:

In [69]:
```
1 list(data_dummy.columns)
```

Out[69]:
```
['Article',
 'Visits',
 'Product Views',
 'In Stock Views',
 'Cart Additions',
 'Orders',
 'Conversion Rate',
 'CartAdd_to_ProductView',
 'aboveMean',
 'Category_BLE',
 'Category_IA&C',
 'Category_II&T',
 'Category_TCFM',
 'Cell_Analog Disc Power',
 'Cell_BLE E-Mech',
 'Cell_Cable, Anti-Vib & Ac',
 'Cell_Consumables & Safety',
 'Cell_E-Conn',
 'Cell_EMECH',
 'Cell_I-Conn',
 'Cell_Mach Guard & Enc',
 'Cell_Mech & Flu Pwr',
 'Cell_Opto',
 'Cell_Passives',
 'Cell_Process Control',
 'Cell_Sensors & Switches',
 'Cell_Test & Measurement',
 'Cell_Tools Storage & FM',
 'Technology_3D Printing',
 'Technology_Acc, Stor & Mat Hand',
 'Technology_Adhesives, Clean & M',
 'Technology_Anti-Vibration',
 'Technology_BLE E-Mech Tech',
 'Technology_Battery',
 'Technology_Cable Accs',
 'Technology_Cable Mgt & Elec Acc',
 'Technology_Capacitors',
 'Technology_Circuit Protection',
 'Technology_Contactors',
 'Technology_Crimp Term & Power',
 'Technology_Ctrl & Ind Relay',
 'Technology_DIN Rail & Term',
 "Technology_DIN Rail PSU's",
 'Technology_Displays',
 'Technology_Elec & Data Cable',
 'Technology_Electrical T&M',
 'Technology_Electronic T&M',
 'Technology_Enclosures',
 'Technology_Engineering  Mat',
 'Technology_Fact Auto Sen',
 'Technology_Fasteners',
 'Technology_Fuses',
 'Technology_HVAC',
 'Technology_Hand Tools',
 'Technology_Heavy Duty Conn',
```

```
 'Technology_Hose & Pumps',
 'Technology_Hydraulics',
 'Technology_IT',
 'Technology_Ind & AV Conns',
 'Technology_Indicators',
 'Technology_Inductors',
 'Technology_LED',
 'Technology_Level & Flow Con',
 'Technology_Lighting',
 'Technology_MIL Ind & Test Conns',
 'Technology_Machine Safety',
 'Technology_Measurement & Insp',
 'Technology_Motors & Control',
 'Technology_PCB Connectors',
"Technology_PLC's & HMI's",
 'Technology_PPE',
 'Technology_Plumbing',
 'Technology_Pneumatics',
 'Technology_Pow Tools & Abras',
 'Technology_Pow Tran & Struc',
 'Technology_Power Management',
 'Technology_Power PCB',
 'Technology_Printing & Office',
 'Technology_Proc Auto Sen',
 'Technology_Push But & Swit',
 'Technology_Pwr & Ind Cable',
 'Technology_RF Datacomm Conns',
 'Technology_Rect Connectors',
 'Technology_Relays-EMECH',
 'Technology_Resistors',
 'Technology_Security',
 'Technology_Sensors',
 'Technology_Site Safety & Janit',
 'Technology_Soldering & ESD',
 'Technology_Sounders/Beacons',
 'Technology_Switches-EMECH',
 'Technology_Temp Control',
 'Technology_Thermal Management',
 'Technology_Timers, Counters']
```

In [70]:

```
feature_cols = ['Article',
 'Visits',
 'Product Views',
 'In Stock Views',
 'Cart Additions',
 'Orders',
 'Category_BLE',
 'Category_IA&C',
 'Category_II&T',
 'Category_TCFM',
 'Cell_Analog Disc Power',
 'Cell_BLE E-Mech',
 'Cell_Cable, Anti-Vib & Ac',
 'Cell_Consumables & Safety',
 'Cell_E-Conn',
 'Cell_EMECH',
 'Cell_I-Conn',
 'Cell_Mach Guard & Enc',
 'Cell_Mech & Flu Pwr',
 'Cell_Opto',
 'Cell_Passives',
 'Cell_Process Control',
 'Cell_Sensors & Switches',
 'Cell_Test & Measurement',
 'Cell_Tools Storage & FM',
 'Technology_3D Printing',
 'Technology_Acc, Stor & Mat Hand',
 'Technology_Adhesives, Clean & M',
 'Technology_Anti-Vibration',
 'Technology_BLE E-Mech Tech',
 'Technology_Battery',
 'Technology_Cable Accs',
 'Technology_Cable Mgt & Elec Acc',
 'Technology_Capacitors',
 'Technology_Circuit Protection',
 'Technology_Contactors',
 'Technology_Crimp Term & Power',
 'Technology_Ctrl & Ind Relay',
 'Technology_DIN Rail & Term',
 "Technology_DIN Rail PSU's",
 'Technology_Displays',
 'Technology_Elec & Data Cable',
 'Technology_Electrical T&M',
 'Technology_Electronic T&M',
 'Technology_Enclosures',
 'Technology_Engineering  Mat',
 'Technology_Fact Auto Sen',
 'Technology_Fasteners',
 'Technology_Fuses',
 'Technology_HVAC',
 'Technology_Hand Tools',
 'Technology_Heavy Duty Conn',
 'Technology_Hose & Pumps',
 'Technology_Hydraulics',
 'Technology_IT',
 'Technology_Ind & AV Conns',
```

```
 'Technology_Indicators',
 'Technology_Inductors',
 'Technology_LED',
 'Technology_Level & Flow Con',
 'Technology_Lighting',
 'Technology_MIL Ind & Test Conns',
 'Technology_Machine Safety',
 'Technology_Measurement & Insp',
 'Technology_Motors & Control',
 'Technology_PCB Connectors',
 "Technology_PLC's & HMI's",
 'Technology_PPE',
 'Technology_Plumbing',
 'Technology_Pneumatics',
 'Technology_Pow Tools & Abras',
 'Technology_Pow Tran & Struc',
 'Technology_Power Management',
 'Technology_Power PCB',
 'Technology_Printing & Office',
 'Technology_Proc Auto Sen',
 'Technology_Push But & Swit',
 'Technology_Pwr & Ind Cable',
 'Technology_RF Datacomm Conns',
 'Technology_Rect Connectors',
 'Technology_Relays-EMECH',
 'Technology_Resistors',
 'Technology_Security',
 'Technology_Sensors',
 'Technology_Site Safety & Janit',
 'Technology_Soldering & ESD',
 'Technology_Sounders/Beacons',
 'Technology_Switches-EMECH',
 'Technology_Temp Control',
 'Technology_Thermal Management',
 'Technology_Timers, Counters']

X = data_dummy[feature_cols]
y = dataset.aboveMean
```

Now let's perform a train/test split (recall that it is important that we did the one-hot encoding to dummy variables *before* the train/test split):

In [71]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
```

We can easily train and fit a Random Forest classifier with certain `hyperparameters` as follows:

In [72]:
```
clf2 = RandomForestClassifier(n_estimators = 500, criterion = 'entropy', max

clf2.fit(X_train, y_train)
```

Out[72]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=123,
                       verbose=0, warm_start=False)
```

But how do I pick the best values of the hyperparameters?

The best way to do this is with a **grid search** and **cross-validation**:

In [73]:
```
from sklearn.model_selection import GridSearchCV

rfc = RandomForestClassifier(random_state = 123)   # we set the random_state

param_grid = [{'max_depth': [1,5,10], 'n_estimators': [1, 10, 100], 'criteri

grid_search = GridSearchCV(rfc, param_grid, cv = 5, scoring = 'accuracy', re

grid_search.fit(X_train, y_train)

grid_search.best_params_
```

Out[73]:
```
{'criterion': 'gini', 'max_depth': 10, 'n_estimators': 100}
```

Let's pick the best estimator from the grid search as our final model:

In [74]:
```
clf2 = grid_search.best_estimator_
```

Evaluating model:

Now that we have built a model (and validated it using cross-validation), let's see how well it performs on the (so far unused) test set:

In [75]:
```
# Making predictions on the test set
y_pred = clf2.predict(X_test)
```

In [76]:
```
# Evaluating performance on the test by computing several metrics
print("Accuracy (test):", metrics.accuracy_score(y_test, y_pred))
print("Precision (test)", metrics.precision_score(y_test, y_pred))
print("Recall (test)", metrics.recall_score(y_test, y_pred))
print("F1-score (test)", metrics.f1_score(y_test, y_pred))
```

```
Accuracy (test): 0.8997905027932961
Precision (test) 0.8955996548748921
Recall (test) 0.8621262458471761
F1-score (test) 0.8785442234447737
```

In [77]:
```
#I'm getting elementwise comparison warning and - as eper advice from StackO
import warnings
with warnings.catch_warnings():
    warnings.filterwarnings('ignore', r'elementwise comparison failed; retur
```

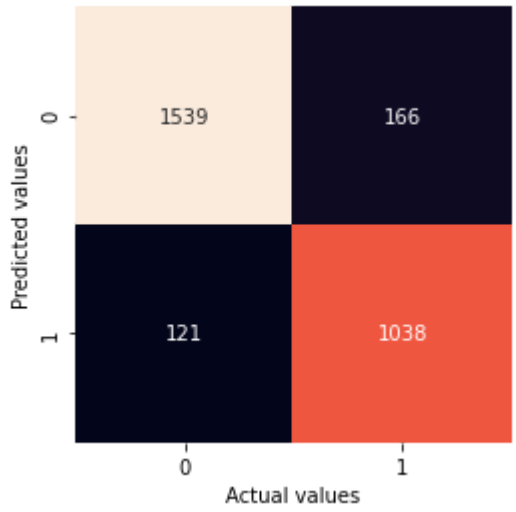
In [78]:
```
# for comparison, see performance on the train set too
y_pred_train = clf2.predict(X_train)

print("Accuracy (train):", metrics.accuracy_score(y_train, y_pred_train))
print("Precision (train)", metrics.precision_score(y_train, y_pred_train))
print("Recall (train)", metrics.recall_score(y_train, y_pred_train))
print("F1-score (train)", metrics.f1_score(y_train, y_pred_train))
```

```
Accuracy (train): 0.9514551804423749
Precision (train) 0.9558350394852296
Recall (train) 0.9247311827956989
F1-score (train) 0.9400258881058536
```

We can see better scores for training set but that is to be expected.

The model is performing pretty well - an accuracy of upper 80-90% is quite good, especially considering our dataset was fairly balanced (since it has been split along the mean) to begin with so our benchmark guess would only be approximately 50% accurate. The precision and recall are also similarly high, so the model is performing well for both "below the mean" and "above the mean" predictions. The model also has similar performance on the train and test set, which means it is not drastically overfitting.

In [79]:
```
# look at the confusion matrix to analyse exactly what the errors are
conf_mat = metrics.confusion_matrix(y_test, y_pred, labels = clf2.classes_)

sns.heatmap(conf_mat.T, square = True, annot = True, fmt = 'd', cbar = False
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.show()
```

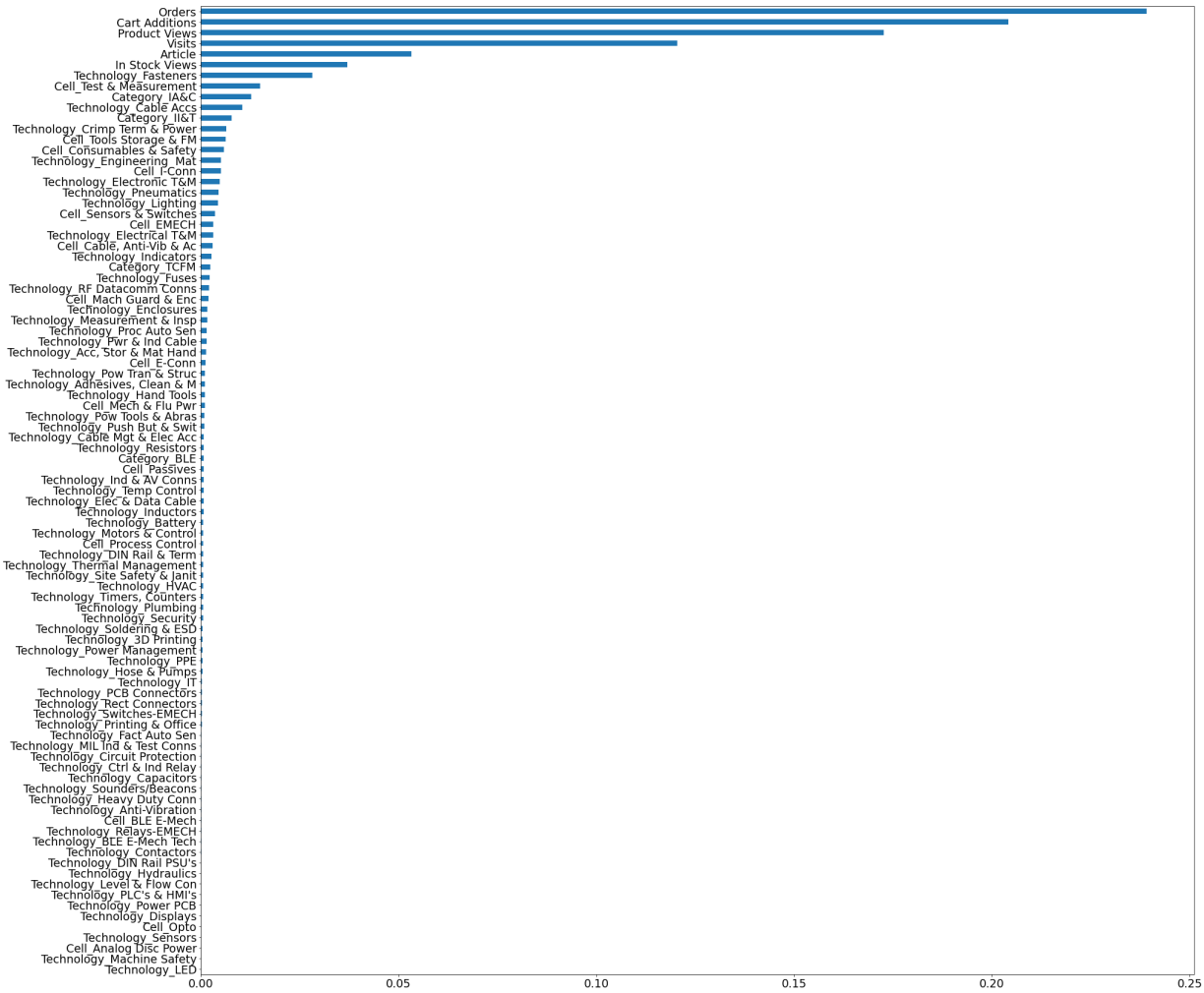


Reading the above confusion matrix, we can see that there were 1539 True Negatives and 1038 True Positives and 166 False Negatives, with 121 False Positives - in the training set (let's remember test set was 25% of all the dataset).

Let's check which features are most **important** in classifying the data - this is determined by which features appear higher up in the trees (on average in the ensemble):

In [80]:
```
feature_importances = pd.Series(clf2.feature_importances_, index = X.columns
feature_importances = feature_importances.sort_values()
feature_importances.plot(kind='barh', figsize = (30,30), fontsize=19)
```

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x1d6e1455f08>



Not surprisingly, Orders have been indicated as Nr 1 and Cart Additions as Nr 2 when it comes to features' importance in predicting the class.

Then we have Product Views and Visits. They matter - just not as much as Orders and Cart Additions.

Alternative approach:

All top 5 features appear to have much higher importance than other. The challenge to the above approach is that given we are trying to predict a binary class of Above-Below mean of CartAdd_to_ProductView ratio, one could argue that ratio can be calculated - directly or indirectly - from the values of those top features (columns).

Let's try different approach and ignore them - dropping 'Visits','Product Views','Cart Additions' and 'Orders' from our training set features. We can certainly expect noticeably lower model scores but perhaps this approach will highlight patterns easier to miss at first glance.

In [81]:

```
feature_cols = ['Article',
 'In Stock Views',
 'Category_BLE',
 'Category_IA&C',
 'Category_II&T',
 'Category_TCFM',
 'Cell_Analog Disc Power',
 'Cell_BLE E-Mech',
 'Cell_Cable, Anti-Vib & Ac',
 'Cell_Consumables & Safety',
 'Cell_E-Conn',
 'Cell_EMECH',
 'Cell_I-Conn',
 'Cell_Mach Guard & Enc',
 'Cell_Mech & Flu Pwr',
 'Cell_Opto',
 'Cell_Passives',
 'Cell_Process Control',
 'Cell_Sensors & Switches',
 'Cell_Test & Measurement',
 'Cell_Tools Storage & FM',
 'Technology_3D Printing',
 'Technology_Acc, Stor & Mat Hand',
 'Technology_Adhesives, Clean & M',
 'Technology_Anti-Vibration',
 'Technology_BLE E-Mech Tech',
 'Technology_Battery',
 'Technology_Cable Accs',
 'Technology_Cable Mgt & Elec Acc',
 'Technology_Capacitors',
 'Technology_Circuit Protection',
 'Technology_Contactors',
 'Technology_Crimp Term & Power',
 'Technology_Ctrl & Ind Relay',
 'Technology_DIN Rail & Term',
 "Technology_DIN Rail PSU's",
 'Technology_Displays',
 'Technology_Elec & Data Cable',
 'Technology_Electrical T&M',
 'Technology_Electronic T&M',
 'Technology_Enclosures',
 'Technology_Engineering  Mat',
 'Technology_Fact Auto Sen',
 'Technology_Fasteners',
 'Technology_Fuses',
 'Technology_HVAC',
 'Technology_Hand Tools',
 'Technology_Heavy Duty Conn',
 'Technology_Hose & Pumps',
 'Technology_Hydraulics',
 'Technology_IT',
 'Technology_Ind & AV Conns',
 'Technology_Indicators',
 'Technology_Inductors',
 'Technology_LED',
 'Technology_Level & Flow Con',
```

```
 'Technology_Lighting',
 'Technology_MIL Ind & Test Conns',
 'Technology_Machine Safety',
 'Technology_Measurement & Insp',
 'Technology_Motors & Control',
 'Technology_PCB Connectors',
 "Technology_PLC's & HMI's",
 'Technology_PPE',
 'Technology_Plumbing',
 'Technology_Pneumatics',
 'Technology_Pow Tools & Abras',
 'Technology_Pow Tran & Struc',
 'Technology_Power Management',
 'Technology_Power PCB',
 'Technology_Printing & Office',
 'Technology_Proc Auto Sen',
 'Technology_Push But & Swit',
 'Technology_Pwr & Ind Cable',
 'Technology_RF Datacomm Conns',
 'Technology_Rect Connectors',
 'Technology_Relays-EMECH',
 'Technology_Resistors',
 'Technology_Security',
 'Technology_Sensors',
 'Technology_Site Safety & Janit',
 'Technology_Soldering & ESD',
 'Technology_Sounders/Beacons',
 'Technology_Switches-EMECH',
 'Technology_Temp Control',
 'Technology_Thermal Management',
 'Technology_Timers, Counters']

X = data_dummy[feature_cols]
y = dataset.aboveMean


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,

rfc = RandomForestClassifier(random_state = 123)

param_grid = [{'max_depth': [1,5,10], 'n_estimators': [1, 10, 100], 'criteri

grid_search = GridSearchCV(rfc, param_grid, cv = 5, scoring = 'accuracy', re
grid_search.fit(X_train, y_train)

clf2 = grid_search.best_estimator_

y_pred = clf2.predict(X_test)
```

In [82]:
```
# Evaluating performance on the test by computing several metrics
print("Accuracy (test):", metrics.accuracy_score(y_test, y_pred))
print("Precision (test)", metrics.precision_score(y_test, y_pred))
print("Recall (test)", metrics.recall_score(y_test, y_pred))
print("F1-score (test)", metrics.f1_score(y_test, y_pred))
```

```
Accuracy (test): 0.7081005586592178
Precision (test) 0.7233009708737864
Recall (test) 0.4950166112956811
F1-score (test) 0.5877712031558185
```

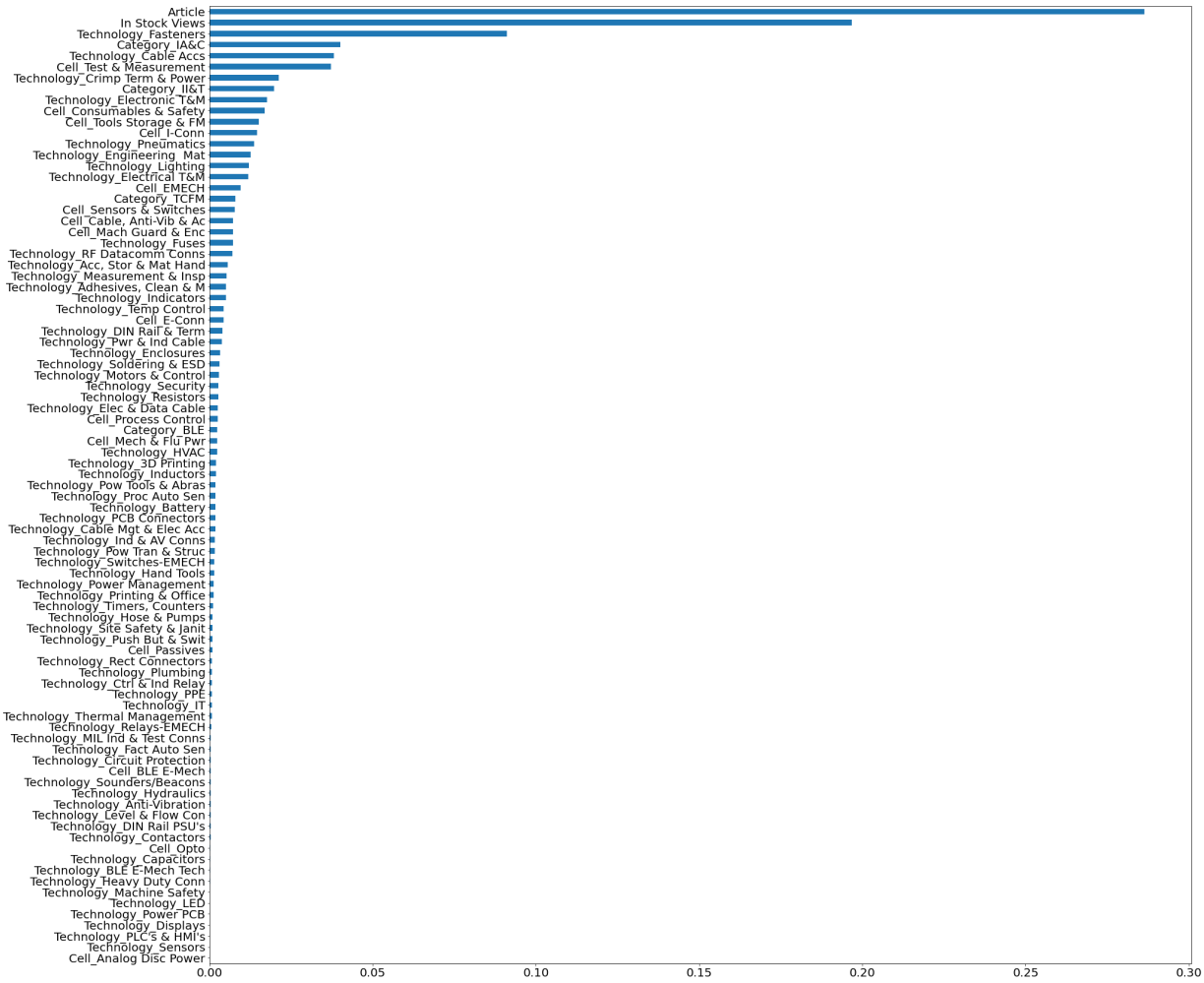
In [83]:
```
# for comparison, let's see performance on the train set too
y_pred_train = clf2.predict(X_train)

print("Accuracy (train):", metrics.accuracy_score(y_train, y_pred_train))
print("Precision (train)", metrics.precision_score(y_train, y_pred_train))
print("Recall (train)", metrics.recall_score(y_train, y_pred_train))
print("F1-score (train)", metrics.f1_score(y_train, y_pred_train))
```

```
Accuracy (train): 0.7628637951105938
Precision (train) 0.8094253823894171
Recall (train) 0.5540464063384267
F1-score (train) 0.6578195867629766
```

We can see that testing set scores are a bit worse than those of training set, e.g. testing Accuracy seems to be circa 6% points lower than in the training set while Precision - 7.6% points worse. It's noticeable but not drastically different.

In [84]:
```
#Checking features importance in the alternative approach:
feature_importances = pd.Series(clf2.feature_importances_, index = X.columns
feature_importances = feature_importances.sort_values()
feature_importances.plot(kind='barh', figsize = (30,30), fontsize=20)
```

Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1d6d9093c88>



Since Articles and In-Stock Views were Nr 5 and 6th accordingly in our first model, not surprisingly they rose here to top 2 spots when it comes to features' importance. But we can now see Fasteners in Technology and IA&C in category rising to prominent positions when it comes to how important each feature is.

Visualizing decision trees in our classifier models:

We can also visualise the individual trees (as we did in the workshop).

Drawing a decision tree in a single decision tree model (simpler and easier to compute):

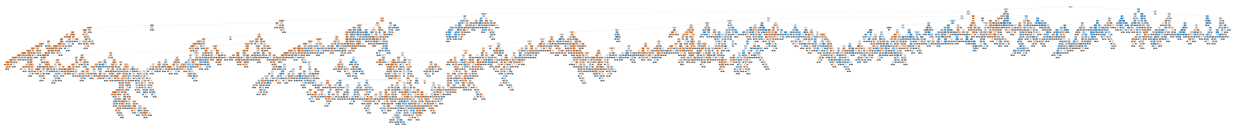
In [85]:
```
# Telling system to use a single decision tree from scikit
model=tree.DecisionTreeClassifier()
#Training model:
model.fit(X_train, y_train)
```

Out[85]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [86]:
```
#Drawing decision tree from a single tree classifier using graphviz:
dot_data=StringIO()
tree.export_graphviz(model, out_file=dot_data, precision = 3, filled=True,
                     rounded=True, special_characters=False, feature_names=X_
                     class_names=True)
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.384305 to fit
```

Out[86]:



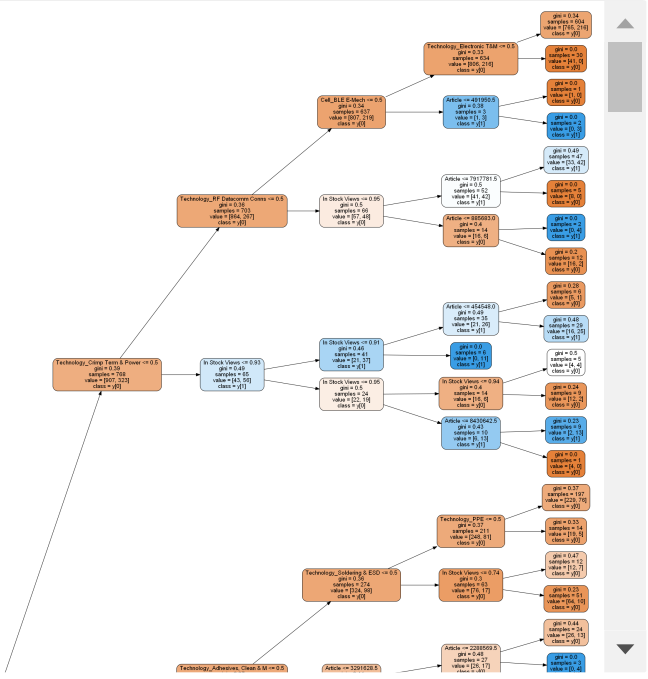
Drawing a selected decision tree from Random Forest classifier model (more complex and compute-intensive but more robust model):

Each tree represents a subset of the random forest ensemble model. For example, we might want to visualise tree number 39, but we could also explore any of the trees from tree 0 to tree 50 in our 100 tree ensemble.

Let's draw a tree number 39 from our random forest ensemble model:

In [87]:
```
# Extracting single tree from our rfc:
estimator = clf2.estimators_[39]

# Exporting tree as a dot file:
dot_data = StringIO()
export_graphviz(estimator, out_file = dot_data,
                special_characters = False, feature_names = X_train.columns,
                rounded = True, proportion = False,
                precision = 2, filled = True, rotate = True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[87]:



Now I'm going to visualize a decision tree number 50 from our random forest model:

This is alternative approach that I have discovered while dealing with an unforeseen error (having to do with Python running into problems when trying to concatenate np.bool_() with strings (I have since managed to resolve it but in the process discovered that multi-step approach - detailed below):

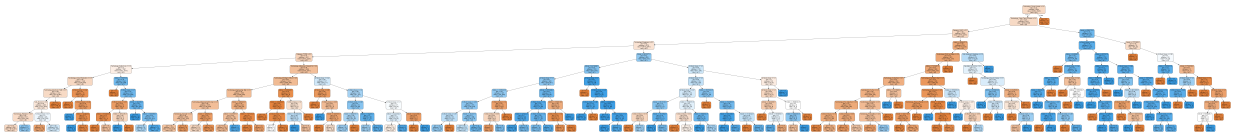
In [88]:
```
# Extracting single tree from our rfc:
estimator = clf2.estimators_[50]

# Exporting tree as a dot file:
export_graphviz(estimator,
                out_file='tree.dot',
                feature_names = X_train.columns,
                class_names = True,
                rounded = True, proportion = False,
                precision = 2, filled = True, impurity=True)
```

In [89]:
```
# Convert to png
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])
```

Out[89]: 0

In [90]:
```
# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

Out[90]:



BONUS MATERIAL: Catboost

CatBoost is a machine learning algorithm that uses gradient boosting on decision trees. It is available as an open source library. It has been developed by Yandex and more information is available here:

https://catboost.ai/docs/concepts/about.html (https://catboost.ai/docs/concepts/about.html)

In [91]:
```
# Importing necessary components of catboost:
from catboost import CatBoostClassifier, Pool, cv
# for visualizing metrics
from catboost import MetricVisualizer
```

Now let's create the model itself:

I'm going to go here with default parameters (as they provide a really good baseline almost all the time).The only thing I'd like to specify here is custom_loss parameter, as this would give us an ability to see what's going on in terms of key metric - accuracy, as well as to be able to watch for logloss, as it would be more smooth on dataset of this size.

In [92]:
```
#Creating catboost model:
model3 = CatBoostClassifier(
    custom_loss=['Accuracy'],
    random_seed=49,
    logging_level='Silent'
)
```

Below I'll train our model in catboost gradient boosting, using default parameters (which are quite useful As Is), plotting the logloss of Accuracy.

After Wikipedia: **"Logarithmic loss (related to cross-entropy) measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of our machine learning models is to minimize this value. A perfect model would have a log loss of 0. Log loss increases as the predicted probability diverges from the actual label. So predicting a probability of .015 when the actual observation label is 1 would be bad and result in a high log loss."**

It's worth pointing out that Catboost allows for customizing loss function so instead of Accuracy, one could e.g. plot Recall. But let's stick to Accuracy.

In the oputput we can interact with the graph, by (un)checking the boxes - e.g by clicking on Smooth and adjust the value of smoothing between 0 to 1 or applying logarithmic scale:

In [93]:
```
#Training our model:
model3.fit(
    X_train, y_train,
    cat_features=None,
    eval_set=(X_test, y_test),
#     logging_level='Verbose',  # you can uncomment this for text output but
    plot=True
)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

Out[93]:
```
<catboost.core.CatBoostClassifier at 0x1d6855e5708>
```

Learn line is basically plotting of logarithmic loss of Accuracy of our training set while Eval line - for Evaluation - denotes log-loss of Accuracy in our training set. The lower the log-loss the better.

***We can see that the best result has been achieved by catboost in 986th boosting step in evaluation (testing) dataset - 13 steps sooner than in training set.**

In [94]:
```
print('Simple model validation accuracy: {:.4}'.format(
    accuracy_score(y_test, model.predict(X_test))
))
```

```
Simple model validation accuracy: 0.7336
```

Catboost enables one to choose from many supported metrics for overfitting detection and best model selection. Let's do the same, but this time highlighting Precision as key metric to watch for as a criterium to avoid overfitting:

In [95]:
```
#Creating catboost model:
model4 = CatBoostClassifier(
    custom_loss=['Precision'],
    random_seed=36,
    logging_level='Silent'
)
```

In [96]:
```
#Training our model:
model4.fit(
    X_train, y_train,
    cat_features=None,
    eval_set=(X_test, y_test),
    #Logging_level='Verbose',
    plot=True,
)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

Out[96]:
```
<catboost.core.CatBoostClassifier at 0x1d6858a0ec8>
```

This time best Precision log-loss was achieved by 998th boosting step.

In [97]:
```
print('Simple model validation precision: {:.4}'.format(precision_score(y_te
```

```
Simple model validation precision: 0.6864
```

Model cross-validation in Catboost:

It's valuable to validate our model, but to cross-validate it - even better:

In [98]:
```
cv_params = model3.get_params()
cv_params.update({
    'loss_function': 'Logloss'
})
cv_data = cv(
    Pool(X, y),
    cv_params,
    plot=True
)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

Now we have values of our loss functions at each boosting step averaged by 3 folds, which should provide us with a more accurate estimation of our model performance.

In [99]:
```
print('Best validation accuracy score: {:.2f}±{:.2f} on step {}'.format(
    np.max(cv_data['test-Accuracy-mean']),
    cv_data['test-Accuracy-std'][np.argmax(cv_data['test-Accuracy-mean'])],
    np.argmax(cv_data['test-Accuracy-mean'])
))
```

```
Best validation accuracy score: 0.75±0.00 on step 999
```

In [100]:
```
# Printing validation Accuracy score:
print('Precise validation accuracy score: {}'.format(np.max(cv_data['test-Ac
```

```
Precise validation accuracy score: 0.7489091469740589
```

In [101]:
```
# Cross-validating 2nd catboost model (with Precision as a key metric to mon
cv_params = model4.get_params()
cv_params.update({
    'loss_function': 'Logloss'
})
cv_data = cv(
    Pool(X, y),
    cv_params,
    plot=True
)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

In [102]:
```
# Printing validation Precision score:
print('Precise validation Precision score: {}'.format(np.max(cv_data['test-P
```

```
Precise validation Precision score: 0.7089567224477711
```

Applying the model (predictions):

In [103]:
```
predictions = model3.predict(X_test)
predictions_probs = model3.predict_proba(X_test)
# Printing first 10 predictions:
print(predictions[:10])
print(predictions_probs[:10])
```

```
['False' 'True' 'False' 'False' 'False' 'True' 'True' 'False' 'False'
 'True']
[[0.99402407 0.00597593]
 [0.08929608 0.91070392]
 [0.77264062 0.22735938]
 [0.61959351 0.38040649]
 [0.99608816 0.00391184]
 [0.08776633 0.91223367]
 [0.13130309 0.86869691]
 [0.92669961 0.07330039]
 [0.76693001 0.23306999]
 [0.31261148 0.68738852]]
```

We can get better ones by using some catboost's built in features:

Let's define some params and create Pool for more convenience. It stores all information about dataset (features, labeles, categorical features indices, weights and and much more):

In [104]:
```
params = {
    'iterations': 1500,
    'learning_rate': 0.1,
    'eval_metric': 'Accuracy',
    'random_seed': 55,
    'logging_level': 'Silent',
    'use_best_model': False
}
train_pool = Pool(X_train, y_train)
validate_pool = Pool(X_test, y_test)
```

Using best model:

If we essentially have a validation set, it's always better to use the use_best_model parameter during training. By default, this parameter is enabled. If it is enabled, the resulting trees ensemble is shrinking to the best iteration.

In [105]:
```
model = CatBoostClassifier(**params)
model.fit(train_pool, eval_set=validate_pool)

best_model_params = params.copy()
best_model_params.update({
    'use_best_model': True
})
best_model = CatBoostClassifier(**best_model_params)
best_model.fit(train_pool, eval_set=validate_pool)
```

Out[105]:
```
<catboost.core.CatBoostClassifier at 0x1d684ebe908>
```

Another brilliant feature of catboost is MetricVisualizer which functionality allow one to plot multiple evaluation metrics on 1 chart for quick comparison, saving space and lines of code. One can simply click on different metrics on the same graph (on top of the graph scale):

In [106]:
```
eval_metrics = best_model.eval_metrics(validate_pool, ['Accuracy', 'Precisio
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

Feature importance:

Finally using catboost get_feature_importance function we can easily get a sorted ranking of importance of features from our training dataset in the model:

In [107]:
```
feature_importances = model.get_feature_importance(train_pool)
feature_names = X_train.columns
for score, name in sorted(zip(feature_importances, feature_names), reverse=T
    print('{}: {}'.format(name, score))
```

```
Article: 38.58037746569475
In Stock Views: 19.945221434607994
Cell_Test & Measurement: 4.453229995694893
Technology_Fasteners: 3.075684543833669
Category_IA&C: 1.6238240967871782
Technology_Cable Accs: 1.4707558703879104
Technology_Engineering  Mat: 1.1884922718997997
Technology_Lighting: 1.147151798085583
Technology_Pwr & Ind Cable: 0.9592125696495065
Technology_Measurement & Insp: 0.9299113084553916
Technology_Acc, Stor & Mat Hand: 0.9189732693412925
Category_II&T: 0.8977574611686182
Category_TCFM: 0.8925073446495475
Technology_Hand Tools: 0.8246671364422077
Technology_Elec & Data Cable: 0.8176029254335158
Cell_Tools Storage & FM: 0.7664729863487418
Technology_Pneumatics: 0.7436176399754864
Technology_Temp Control: 0.7099025402698625
Technology_Pow Tools & Abras: 0.7016134327066847
Technology_RF Datacomm Conns: 0.6966923380765181
Technology_Electronic T&M: 0.6857253391616259
Cell_Consumables & Safety: 0.6822667058639351
Technology_Crimp Term & Power: 0.6780522582571002
Technology_Pow Tran & Struc: 0.6603148434657569
Cell_Cable, Anti-Vib & Ac: 0.631736602587004
Technology_Adhesives, Clean & M: 0.6256506845186643
Technology_Security: 0.5827222591984218
Technology_Inductors: 0.5677006330463804
Technology_Motors & Control: 0.5228057572782586
Technology_Cable Mgt & Elec Acc: 0.5176601404728891
Cell_I-Conn: 0.5076422052768433
Cell_Sensors & Switches: 0.4910705998732789
Technology_Fuses: 0.4792971519701038
Technology_Enclosures: 0.4773347663579847
Technology_Power Management: 0.47541735577385286
Technology_Soldering & ESD: 0.47486220140071844
Technology_Rect Connectors: 0.46903053700430714
Cell_E-Conn: 0.4569292387099711
Technology_Site Safety & Janit: 0.45292488442642465
Technology_HVAC: 0.43287257318519134
Technology_Electrical T&M: 0.4272260616941882
Technology_Proc Auto Sen: 0.4001105898200931
Technology_Battery: 0.3910137579678977
Technology_PPE: 0.3809122384251792
Technology_Indicators: 0.357662521528442
Technology_Printing & Office: 0.3431593597501616
Technology_Ind & AV Conns: 0.34178870467270256
Cell_Process Control: 0.33726930743886985
Technology_Plumbing: 0.3178627058294164
Technology_DIN Rail & Term: 0.30867118446173974
Technology_Resistors: 0.2539185766786817
Cell_Passives: 0.25371366606773055
```

```
Technology_3D Printing: 0.2535336477392696
Technology_Push But & Swit: 0.24886900189267627
Technology_Thermal Management: 0.2400101878899823
Category_BLE: 0.23513256991463735
Technology_MIL Ind & Test Conns: 0.22962205686294399
Technology_IT: 0.22689345679450904
Technology_Ctrl & Ind Relay: 0.22290589300734792
Technology_Timers, Counters: 0.20274748617712016
Technology_Hose & Pumps: 0.19149955195612337
Technology_Switches-EMECH: 0.1681194915506174
Technology_PCB Connectors: 0.16125821398442616
Cell_Mech & Flu Pwr: 0.14268532546053342
Technology_Fact Auto Sen: 0.11478949659434402
Technology_Level & Flow Con: 0.11358785023799228
Cell_EMECH: 0.11358387193298426
Technology_Hydraulics: 0.10441735285386344
Cell_Mach Guard & Enc: 0.10177369985271964
Technology_Anti-Vibration: 0.08657259395555832
Technology_Sounders/Beacons: 0.08163294097334729
Technology_Capacitors: 0.07695762536498478
Cell_BLE E-Mech: 0.07654191146334507
Cell_Opto: 0.06016903645760899
Technology_DIN Rail PSU's: 0.050758276842717304
Technology_Circuit Protection: 0.047242602696098174
Technology_Relays-EMECH: 0.032711571615089426
Technology_Machine Safety: 0.02280257891368375
Technology_Contactors: 0.02203406502316675
Technology_BLE E-Mech Tech: 0.020719409859415387
Technology_Heavy Duty Conn: 0.011327756138073224
Cell_Analog Disc Power: 0.005022242293213834
Technology_Power PCB: 0.002460753797098796
Technology_PLC's & HMI's: 0.0018232699393543673
Technology_Displays: 0.0006637554983654682
Technology_LED: 0.00013661279379845789
Technology_Sensors: 0.0
```

Finally, catboost allows us to draw statistics of selected features on a graph:

In [108]:

```
stats = model.calc_feature_statistics(X_train,
                        y_train,
                        feature = [1,4,6],
                        plot=True,
                        max_cat_features_on_plot=10,
                        thread_count=-1,
                        plot_file=None)
```

Summary:

So far there isn't anything obvious popping up and begging for a deep dive. That could be due to multi-dimensional analysis performed above. Let's refresh our memory by having another look at correlation table of our features, this time filtering all features correlated in some way with our prediction class, namely aboveMean. As a reminder, that class covers a ratio of Cart Additions to Product Views that is above mean value for all values of that column.

In [109]:
```
#Filtering our previous correlation table by rows that have a correlation pa
corr_aboveavg = corr_table.filter(like='aboveMean', axis=0)
```

In [110]:
```
corr_aboveavg.sample()
```

Out[110]:

	r value	p-value	N
aboveMean & Technology_Electronic T&M	-0.1017	0.0	11454

In [111]:
```
#Sorting our filtered correlation table in descending order (to see highest
corr_aboveavg.sort_values(by=['r value'], ascending=False).head(15)
```

Out[111]:

	r value	p-value	N
CartAdd_to_ProductView & aboveMean	0.7916	0.0	11454
Conversion Rate & aboveMean	0.7910	0.0	11454
Orders & aboveMean	0.2666	0.0	11454
Cart Additions & aboveMean	0.2488	0.0	11454
aboveMean & Technology_Fasteners	0.2104	0.0	11454
In Stock Views & aboveMean	0.1769	0.0	11454
aboveMean & Technology_Cable Accs	0.1366	0.0	11454
aboveMean & Cell_Consumables & Safety	0.1232	0.0	11454
aboveMean & Technology_Crimp Term & Power	0.1073	0.0	11454
aboveMean & Cell_I-Conn	0.1032	0.0	11454
aboveMean & Category_II&T	0.0960	0.0	11454
aboveMean & Cell EMECH	0.0831	0.0	11454

In [112]:
```
#Sorting correlation table by r value in ascending order:
corr_aboveavg.sort_values(by=['r value'], ascending=True).head(10)
```

Out[112]:

	r value	p-value	N
aboveMean & Category_IA&C	-0.1595	0.0	11454
aboveMean & Cell_Test & Measurement	-0.1435	0.0	11454
aboveMean & Cell_Tools Storage & FM	-0.1118	0.0	11454
aboveMean & Technology_Electronic T&M	-0.1017	0.0	11454
Article & aboveMean	-0.1011	0.0	11454
aboveMean & Cell_Mach Guard & Enc	-0.1001	0.0	11454
aboveMean & Technology_Electrical T&M	-0.0997	0.0	11454
aboveMean & Cell_Sensors & Switches	-0.0958	0.0	11454
aboveMean & Technology_Lighting	-0.0957	0.0	11454
aboveMean & Technology_Enclosures	-0.0857	0.0	11454

CONCLUSIONS:

As we can see above, that wasn't a straight line journey. I have explored multiple approaches to classification problem, from a single decision tree through a random forest ensemble all the way to relatively little know Catboost gradient boosting model approach. None of those managed to reveal anything ground-breaking, not because of any major inherent flaws but more so because our data set hasn't been as rich as one would have liked. Out of necessity, I have elected to try different features to predict aboveMean class to find out which of those model features would be the best predictors of a ratio of cart additions to product views being above average.

Not surprisingly, given it's a ratio, the top 2 were Cart Additions and Product Views and just behind them - Conversion Rate. The latter because our ratio can be treated as a proxy for Conversion Rate (which is calculated by dividing Orders to Product Views). Since typically a customer will proceed from adding product to a shopping cart to ordering and purchasing items, it was to be expected.

There are some interesting observations to be made, though. Among our main categories, Industrial Automation & Control (IA & C) has the highest negative r value in a correlation pair with aboveMean which tells us there exists an inverse relationship between products in that category being viewed and added to the cart and above average proportion of those 2 activities. It might sound a bit confusing but what it basically means is if a product being viewed belongs to Industrial Automation & Control category, chances are client is less likely to add it to the cart and thus our conversion rate will be below average (when it comes to all 10,000+ product we sell).

The same can be said about Cell_Test & Measurement (which our random forest ensemble model has picked as 5th most important features when it comes to predicting if our ratio will be above or below mean) or Cell_Tools Storage & FM product ranges. While r values of -0.1435 or -0.1118 are not very high, they are among the top negative relationships with aboveMean so for some reason our clients are less likely to add them to the shopping cart than virtually all other best selling product ranges. It could be due to pricing, insufficient product information being available, poor picture quality of myriad other reasons, but the pattern exists. It would be interesting to learn what profit margin those product provide, because if it's significant and sold in large enough volumes, it might make sense to try to improve their conversion ratio to bring it closer to our aggregate average.

On the positive side, some of our technologies, like Fasteners and Cable Accessories have slightly positive correlation with above average ratio (and thus conversion rate). For whatever reason our clients are slightly more likely to add those items to the cart and - presumably - pull the trigger when it comes to shopping for them. The same can be said about Cell_Consumables & Safety. Those products seem to be more tempting to our clients so perhaps there are some learning around the way we do promoting and pricing those products so that they get higher than average conversion rates.

While I can't see a full path yet to implementing these findings into Inventory Management function's operations or to directly boost our Marketing department efforts, it further reinforces my view that we, as a global organization, would be well served to undertake to collect more data on our product features and customer journey across our online sales channel, so that my team has

more features to work with. Those we seem to be collecting are too often either relatively highly correlated (thus risking overfitting) or don't provide much insight. Overall, we would definitely benefit from an ability to notice otherwise unseen patterns and to come up with a model to generalize those patterns onto observations and it's precisely why tools like major classification algorithms are essential in any data scientist's toolbox and of value to us as a corporation stocking hundreds of thousands of products and hoping to predict which ones are going to sell better or foresee which ones will sell out faster . That said, even the best algorithm on its own won't help us much unless the quality and breadth of data we collect gets expanded so that we draw insights from it.

In [ ]:
```
1
```