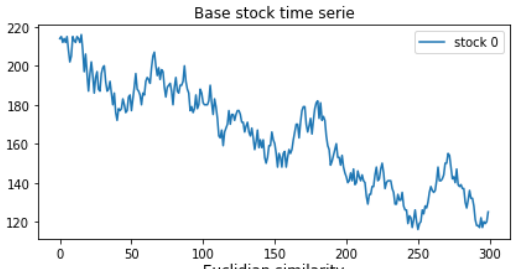
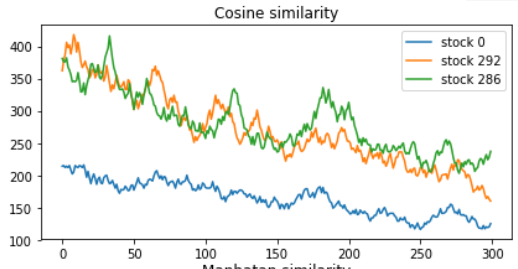
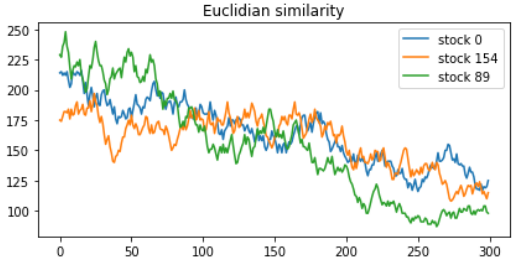
# Google Colab

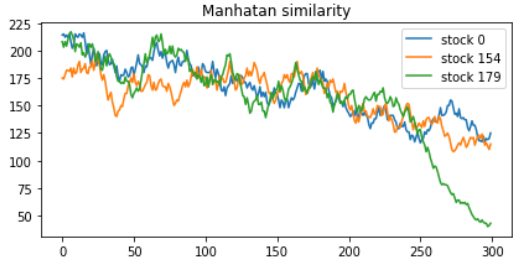
```
class Stock:
    def __init__(self, name, vector):
        self.name = name
        self.vector = vector

def vector_len(v):
    return math.sqrt(sum([x*x for x in v]))

def dot_product(v1, v2):
    assert len(v1) == len(v2)
    return sum([x*y for (x,y) in zip(v1, v2)])

def euclidean_distance_similarity(v1, v2):
    return math.sqrt(sum(pow(a-b,2) for a, b in zip(v1, v2)))


def manhattan_distance(v1,v2):
    return sum(abs(a-b) for a,b in zip(v1,v2))


def cosine_similarity(v1, v2):
    """
    Returns the cosine of the angle between the two vectors.
    Results range from -1 (very different) to 1 (very similar).
    """
    return dot_product(v1, v2) / (vector_len(v1) * vector_len(v2))
```

# Google Colab



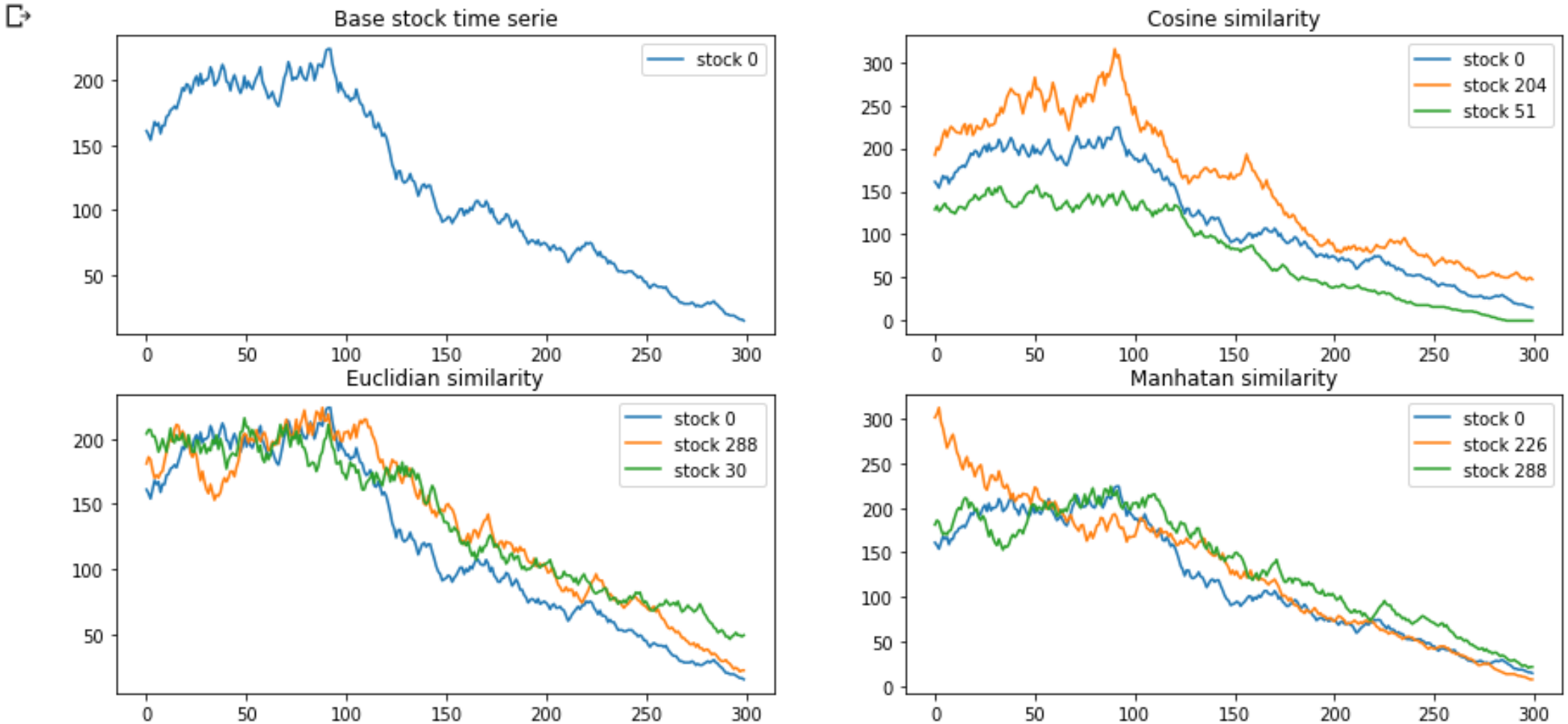

```
def plot_k_most_similar_to_stock(dataSet, stock_name, k):
    plt.figure(figsize=(15,7))
    v = [ stock.vector for stock in dataSet if stock.name == stock_name][0]

    plt.subplot(221)
    plt.title('Base stock time serie')
    plt.plot(range(len(v)), v[0:], label = str(stock_name))
    plt.legend()
    dict ={}
    for i in dataSet:
        dict[i.name] = cosine_similarity(i.vector, v)
    d = Counter(dict)
    plt.subplot(222)
    plt.title('Cosine similarity')
    for name, vector in d.most_common(k):
        vector = [ stock.vector for stock in dataSet if stock.name == name][0]
        plt.plot(range(len(vector[0:])), vector[0:], label = str(name))
        plt.legend()


    dict ={}
    for i in dataSet:
        dict[i.name] = euclidean_distance_similarity(i.vector, v)
    d = Counter(dict)
    plt.subplot(223)
    plt.title('Euclidian similarity')
    for name, vector in (d.most_common()[:-k-1:-1]):
        vector = [ stock.vector for stock in dataSet if stock.name == name][0]
        plt.plot(range(len(vector[0:])), vector[0:], label = str(name))
        plt.legend()

    dict ={}
    for i in dataSet:
        dict[i.name] = manhattan_distance(i.vector, v)
    d = Counter(dict)
    plt.subplot(224)
    plt.title('Manhatan similarity')
    for name, vector in d.most_common()[:-k-1:-1]:
        vector = [ stock.vector for stock in dataSet if stock.name == name][0]
        plt.plot(range(len(vector[0:])), vector[0:], label = str(name))
        plt.legend()
    plt.show()
```

# Google Colab

```
def generateRandomVector():
    import random
    lst = [random.randint(100, 1000)]
    for i in range(299):
        lst.append(random.randint(int((lst[-1]  - lst[-1]*0.05)),int((lst[-1]  + lst[-1]*0.05))))
    return lst

def creatRandomStockTimeSerie():
    timeSerieBank = []
    for i in range(300):
        stock = str('stock ' + str(i))
        stockVector = generateRandomVector()
        timeSerieBank.append(Stock(stock, stockVector))
    return timeSerieBank
```

# Google Colab

```
[5]  dataSet = creatRandomStockTimeSerie()
     plot_k_most_similar_to_stock(dataSet, 'stock 0', 3)
```



# client2vec: Towards Systematic Baselines for Banking Applications

Leonardo Baldassini
BBVA Data & Analytics
leonardo.baldassini@bbvadata.com

Jose Antonio Rodríguez Serrano
BBVA Data & Analytics
joseantonio.rodriguez.serrano@bbvadata.com



## ABSTRACT

The workflow of data scientists normally involves potentially inefficient processes such as data mining, feature engineering and model selection. Recent research has focused on automating this workflow, partly or in its entirety, to improve productivity. We choose the former approach and in this paper share our experience in designing the client2vec: an internal library to rapidly build baselines for banking applications. Client2vec uses marginalized stacked denoising autoencoders on current account transactions data to create vector embeddings which represent the behaviors of our clients. These representations can then be used in, and optimized against, a variety of tasks such as client segmentation, profiling and targeting. Here we detail how we selected the algorithmic machinery of client2vec and the data it works on and present experimental results on several business cases.

form (client embedding). These embeddings make it possible to quantify how similar two customers are and, when input into clustering or regression algorithms, outperform the sociodemographic customer attributes traditionally used for customer segmentation or marketing campaigns. We pursued a solution with minimal computational and preprocessing requirements that could run even on simple infrastructures. This is not damaging: baselines need to provide an informative starting point rather than an off-the-shelf solution, and client2vec helps generating them in few minutes, with few lines of code. Additionally, client2vec offers our data scientists the possibility to optimize the embeddings against the business problem at hand. For instance, the embedding may be tuned to optimize the average precision for the task of retrieving suitable targets for a campaign.

This paper describes our experience and what we learned while building client2vec; it is organized as follows:
in Section 2 we go through the principles we built client2vec