CUSTOMER SEGMENTATION IN PYTHON

Customer Segmentation in Python

Karolis Urbonas
Head of Data Science, Amazon

About me

- Head of Data Science at Amazon
- 10+ years experience with analytics and ML
- Worked in eCommerce, banking, consulting, finance and other industries

Prerequisites

- `pandas` library
- `datetime` objects
- basic plotting with `matplotlib` or `seaborn`
- basic knowledge of k-means clustering

What is Cohort Analysis?

- Mutually exclusive segments - cohorts
- Compare metrics across **product** lifecycle
- Compare metrics across **customer** lifecycle

Types of cohorts

- Time cohorts
- Behavior cohorts
- Size cohorts

Elements of cohort analysis

- Pivot table

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Elements of cohort analysis

- Pivot table
- Assigned cohort in **rows**

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Elements of cohort analysis

- Pivot table
- Assigned cohort in **rows**
- Cohort Index in **columns**

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Elements of cohort analysis

- Pivot table
- Assigned cohort in **rows**
- Cohort Index in **columns**
- Metrics in the **table**

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Elements of cohort analysis

- First cohort was acquired in

  December 2010

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Elements of cohort analysis

- First cohort was acquired in December 2010
- Last cohort was acquired in December 2011

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

CUSTOMER SEGMENTATION IN PYTHON

Explore the cohort table

CUSTOMER SEGMENTATION IN PYTHON

Time cohorts

Karolis Urbonas
Head of Data Science, Amazon

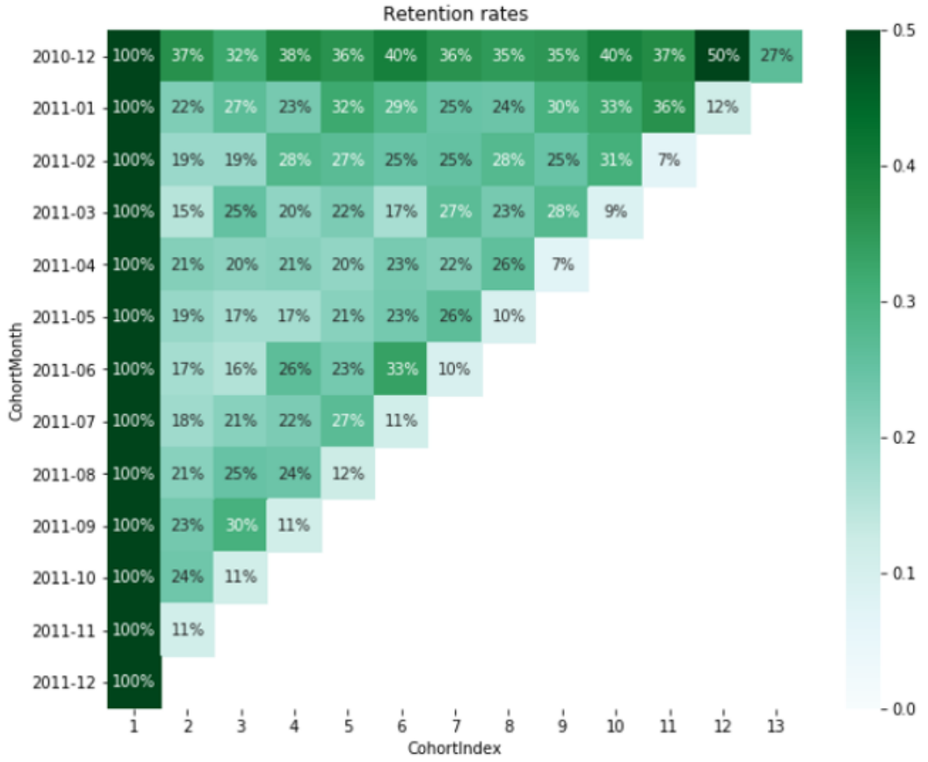
Cohort analysis heatmap

Rows:

- First activity
- Here - month of acquisition

Columns:

- Time since first activity
- Here - months since acquisition



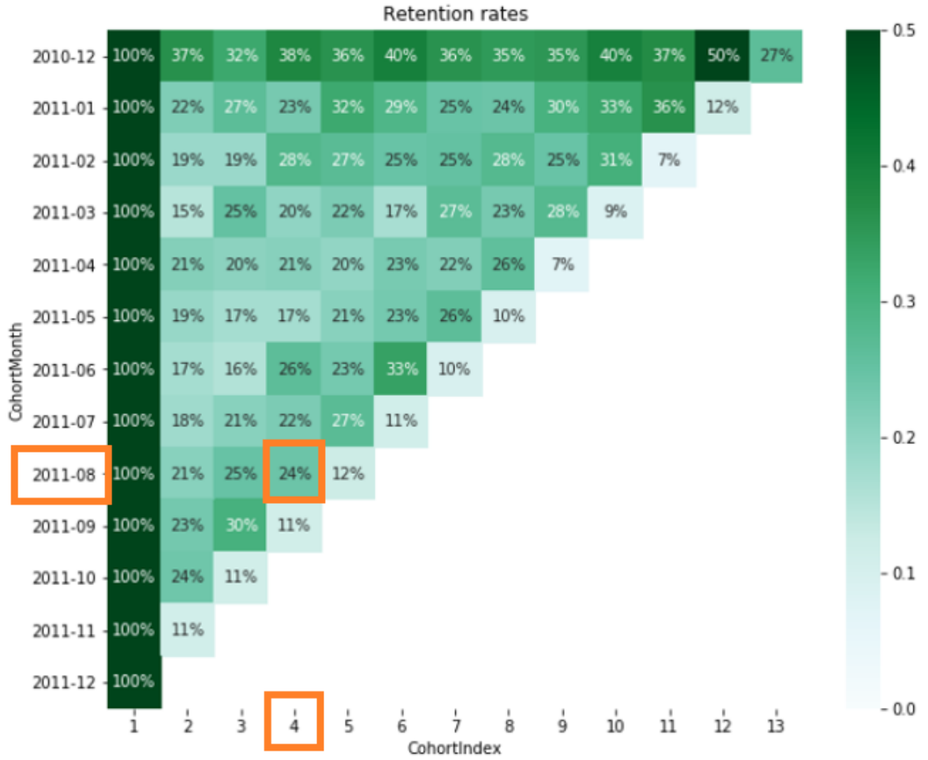
Cohort analysis heatmap

Rows:

- First activity
- Here - month of acquisition

Columns:

- Time since first activity
- Here - months since acquisition



Online Retail data

Over 0.5 million transactions from a UK-based online retail store.

We will use a randomly sampled 20% subset of this dataset throughout the course.

Online Retail Data Set

Download: Data Folder, Data Set Description

Top 5 rows of data

```
online.head()
```

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
572558	22745	POPPY'S PLAYHOUSE BEDROOM	6	2011-10-25 08:26:00	2.10	14286	United Kingdom
577485	23196	VINTAGE LEAF MAGNETIC NOTEPAD	1	2011-11-20 11:56:00	1.45	16360	United Kingdom
560034	23299	FOOD COVER WITH BEADS SET 2	6	2011-07-14 13:35:00	3.75	13933	United Kingdom
578307	72349B	SET/6 PURPLE BUTTERFLY T-LIGHTS	1	2011-11-23 15:53:00	2.10	17290	United Kingdom
554656	21756	BATH BUILDING BLOCK WORD	3	2011-05-25 13:36:00	5.95	17663	United Kingdom

Assign acquisition month cohort

```
def get_month(x): return dt.datetime(x.year, x.month, 1)
```

```
online['InvoiceMonth'] = online['InvoiceDate'].apply(get_month)
```

```
grouping = online.groupby('CustomerID')['InvoiceMonth']
```

```
online['CohortMonth'] = grouping.transform('min')
```

```
online.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth
416792	572558	22745	POPPY'S PLAYHOUSE BEDROOM	6	2011-10-25 08:26:00	2.10	14286.0	United Kingdom	2011-10-01	2011-04-01
482904	577485	23196	VINTAGE LEAF MAGNETIC NOTEPAD	1	2011-11-20 11:56:00	1.45	16360.0	United Kingdom	2011-11-01	2011-09-01
263743	560034	23299	FOOD COVER WITH BEADS SET 2	6	2011-07-14 13:35:00	3.75	13933.0	United Kingdom	2011-07-01	2011-07-01
495549	578307	72349B	SET/6 PURPLE BUTTERFLY T-LIGHTS	1	2011-11-23 15:53:00	2.10	17290.0	United Kingdom	2011-11-01	2011-11-01
204384	554656	21756	BATH BUILDING BLOCK WORD	3	2011-05-25 13:36:00	5.95	17663.0	United Kingdom	2011-05-01	2011-02-01

Extract integer values from data

Define function to extract `year`, `month` and `day` integer values.

We will use it throughout the course.

```
def get_date_int(df, column):
    year = df[column].dt.year
    month = df[column].dt.month
    day = df[column].dt.day
    return year, month, day
```

Assign time offset value

```
invoice_year, invoice_month, _ = get_date_int(online, 'InvoiceMonth')
cohort_year, cohort_month, _ = get_date_int(online, 'CohortMonth')
```

```
years_diff = invoice_year - cohort_year
months_diff = invoice_month - cohort_month
```

```
online['CohortIndex'] = years_diff * 12 + months_diff + 1
online.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth	CohortIndex
416792	572558	22745	POPPY'S PLAYHOUSE BEDROOM	6	2011-10-25 08:26:00	2.10	14286.0	United Kingdom	2011-10-01	2011-04-01	7
482904	577485	23196	VINTAGE LEAF MAGNETIC NOTEPAD	1	2011-11-20 11:56:00	1.45	16360.0	United Kingdom	2011-11-01	2011-09-01	3
263743	560034	23299	FOOD COVER WITH BEADS SET 2	6	2011-07-14 13:35:00	3.75	13933.0	United Kingdom	2011-07-01	2011-07-01	1
495549	578307	72349B	SET/6 PURPLE BUTTERFLY T-LIGHTS	1	2011-11-23 15:53:00	2.10	17290.0	United Kingdom	2011-11-01	2011-11-01	1
204384	554656	21756	BATH BUILDING BLOCK WORD	3	2011-05-25 13:36:00	5.95	17663.0	United Kingdom	2011-05-01	2011-02-01	4

Count monthly active customers from each cohort

```
grouping = online.groupby(['CohortMonth', 'CohortIndex'])
```

```
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
```

```
cohort_data = cohort_data.reset_index()
```

```
cohort_counts = cohort_data.pivot(index='CohortMonth',
                                  columns='CohortIndex',
                                  values='CustomerID')
```

```
print(cohort_counts)
```

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

CUSTOMER SEGMENTATION IN PYTHON

Your turn to build some cohorts!

CUSTOMER SEGMENTATION IN PYTHON

Calculate cohort metrics

Karolis Urbonas
Head of Data Science, Amazon

Customer retention: cohort_counts table

- How many customers originally in each cohort in the `cohort_counts` table?

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Customer retention: cohort_counts table

- How many customers originally in each cohort?
- How many of them were active in following months?

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	716.0	246.0	221.0	251.0	245.0	285.0	249.0	236.0	240.0	265.0	254.0	348.0	172.0
2011-01-01	332.0	69.0	82.0	81.0	110.0	90.0	82.0	86.0	104.0	102.0	124.0	45.0	NaN
2011-02-01	316.0	58.0	57.0	83.0	85.0	74.0	80.0	83.0	86.0	95.0	28.0	NaN	NaN
2011-03-01	388.0	63.0	100.0	76.0	83.0	67.0	98.0	85.0	107.0	38.0	NaN	NaN	NaN
2011-04-01	255.0	49.0	52.0	49.0	47.0	52.0	56.0	59.0	17.0	NaN	NaN	NaN	NaN
2011-05-01	249.0	40.0	43.0	36.0	52.0	58.0	61.0	22.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	207.0	33.0	26.0	41.0	49.0	62.0	19.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	173.0	28.0	31.0	38.0	44.0	17.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	139.0	30.0	28.0	35.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	279.0	56.0	78.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	318.0	67.0	30.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	291.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	38.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Calculate Retention rate

1. Store the first column as `cohort_sizes`

```
cohort_sizes = cohort_counts.iloc[:,0]
```

2. Divide all values in the `cohort_counts` table by `cohort_sizes`

```
retention = cohort_counts.divide(cohort_sizes, axis=0)
```

3. Review the retention table

```
retention.round(3) * 100
```

Retention table

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12	100.0	34.4	30.9	35.1	34.2	39.8	34.8	33.0	33.5	37.0	35.5	48.6	24.0
2011-01	100.0	20.8	24.7	24.4	33.1	27.1	24.7	25.9	31.3	30.7	37.3	13.6	NaN
2011-02	100.0	18.4	18.0	26.3	26.9	23.4	25.3	26.3	27.2	30.1	8.9	NaN	NaN
2011-03	100.0	16.2	25.8	19.6	21.4	17.3	25.3	21.9	27.6	9.8	NaN	NaN	NaN
2011-04	100.0	19.2	20.4	19.2	18.4	20.4	22.0	23.1	6.7	NaN	NaN	NaN	NaN
2011-05	100.0	16.1	17.3	14.5	20.9	23.3	24.5	8.8	NaN	NaN	NaN	NaN	NaN
2011-06	100.0	15.9	12.6	19.8	23.7	30.0	9.2	NaN	NaN	NaN	NaN	NaN	NaN
2011-07	100.0	16.2	17.9	22.0	25.4	9.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08	100.0	21.6	20.1	25.2	10.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	100.0	20.1	28.0	12.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	100.0	21.1	9.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	100.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Other metrics

```
grouping = online.groupby(['CohortMonth', 'CohortIndex'])

cohort_data = grouping['Quantity'].mean()

cohort_data = cohort_data.reset_index()

average_quantity = cohort_data.pivot(index='CohortMonth',
                                     columns='CohortIndex',
                                     values='Quantity')


average_quantity.round(1)
```

Average quantity for each cohort

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12	11.1	12.3	12.2	13.2	13.7	11.3	10.6	12.8	13.2	13.0	14.5	10.9	13.7
2011-01	10.9	10.8	10.0	10.1	14.3	13.2	17.4	16.4	18.7	10.2	10.7	13.2	NaN
2011-02	12.1	14.3	10.6	11.5	17.5	12.2	17.3	13.2	13.4	15.9	14.3	NaN	NaN
2011-03	9.6	14.2	13.0	10.2	16.1	12.7	11.6	11.5	9.0	9.6	NaN	NaN	NaN
2011-04	9.9	11.1	12.4	11.5	11.4	7.7	10.4	9.4	6.6	NaN	NaN	NaN	NaN
2011-05	14.1	9.6	15.3	11.6	11.9	8.5	9.8	7.3	NaN	NaN	NaN	NaN	NaN
2011-06	10.6	16.1	18.1	11.2	12.4	7.2	9.7	NaN	NaN	NaN	NaN	NaN	NaN
2011-07	10.9	16.4	5.6	10.1	6.2	7.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08	10.5	7.4	5.5	5.7	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	11.5	6.3	8.4	9.9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	9.3	7.5	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	7.8	7.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	21.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

CUSTOMER SEGMENTATION IN PYTHON

Let's practice on other cohort metrics!

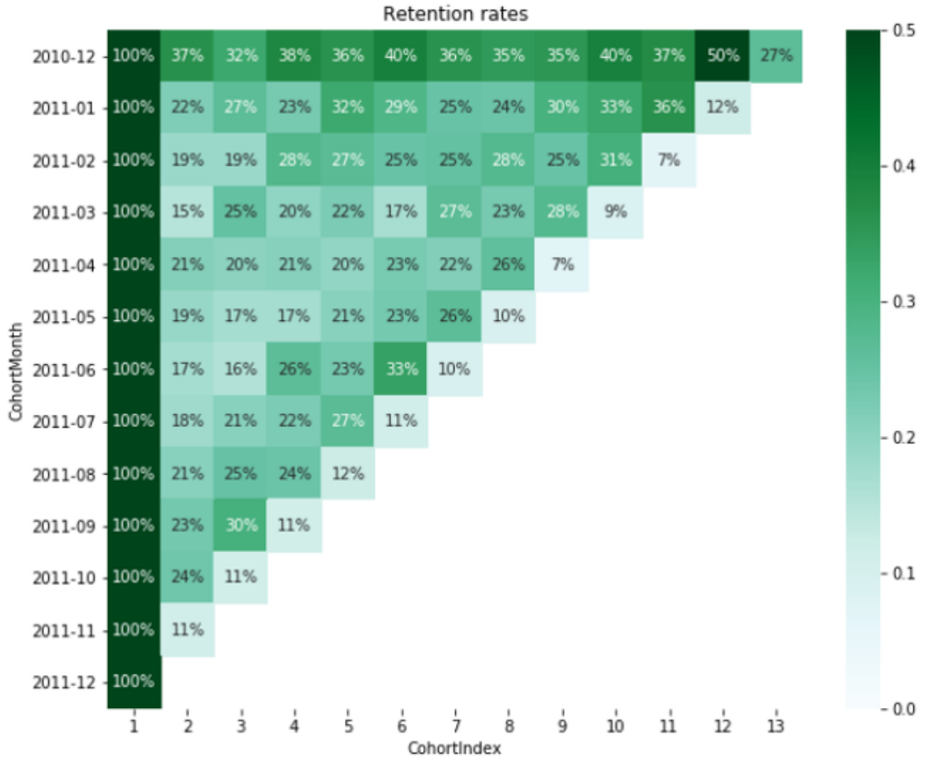
CUSTOMER SEGMENTATION IN PYTHON

Cohort analysis visualization

Karolis Urbonas
Head of Data Science, Amazon

Heatmap

- Easiest way to visualize cohort analysis
- Includes both data and visuals
- Only few lines of code with `seaborn`



Load the retention table

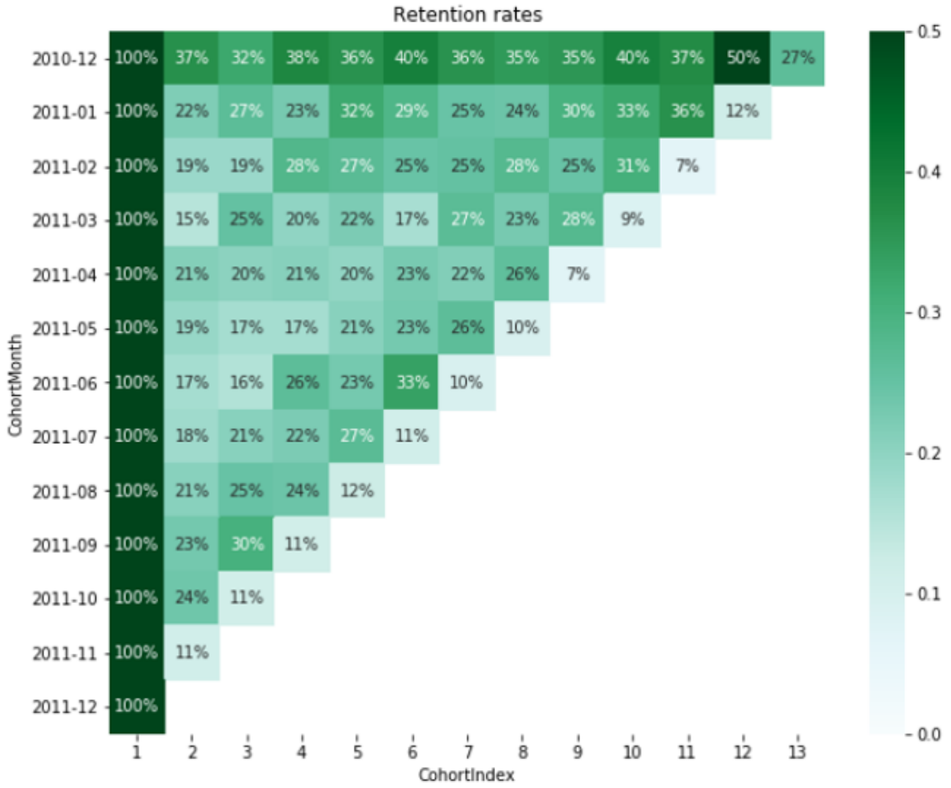
```
retention.round(3)*100
```

CohortMonth	1	2	3	4	5	6	7	8	9	10	11	12	13
2010-12	100.0	34.4	30.9	35.1	34.2	39.8	34.8	33.0	33.5	37.0	35.5	48.6	24.0
2011-01	100.0	20.8	24.7	24.4	33.1	27.1	24.7	25.9	31.3	30.7	37.3	13.6	NaN
2011-02	100.0	18.4	18.0	26.3	26.9	23.4	25.3	26.3	27.2	30.1	8.9	NaN	NaN
2011-03	100.0	16.2	25.8	19.6	21.4	17.3	25.3	21.9	27.6	9.8	NaN	NaN	NaN
2011-04	100.0	19.2	20.4	19.2	18.4	20.4	22.0	23.1	6.7	NaN	NaN	NaN	NaN
2011-05	100.0	16.1	17.3	14.5	20.9	23.3	24.5	8.8	NaN	NaN	NaN	NaN	NaN
2011-06	100.0	15.9	12.6	19.8	23.7	30.0	9.2	NaN	NaN	NaN	NaN	NaN	NaN
2011-07	100.0	16.2	17.9	22.0	25.4	9.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08	100.0	21.6	20.1	25.2	10.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	100.0	20.1	28.0	12.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	100.0	21.1	9.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	100.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Build the heatmap

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))

plt.title('Retention rates')

sns.heatmap(data = retention,
            annot = True,
            fmt = '.0%',
            vmin = 0.0,
            vmax = 0.5,
            cmap = 'BuGn')

plt.show()
```

Retention heatmap



CUSTOMER SEGMENTATION IN PYTHON

Practice visualizing cohorts

CUSTOMER SEGMENTATION IN PYTHON

Introduction to RFM segmentation

Karolis Urbonas
Head of Data Science, Amazon

What is RFM segmentation?

Behavioral customer segmentation based on three metrics:

- Recency (R)
- Frequency (F)
- Monetary Value (M)

Grouping RFM values

The RFM values can be grouped in several ways:

- Percentiles e.g. quantiles
- Pareto 80/20 cut
- Custom - based on business knowledge

We are going to implement percentile-based grouping.

Short review of percentiles

Process of calculating percentiles:

1. Sort customers based on that metric
2. Break customers into a pre-defined number of groups of equal size
3. Assign a label to each group

Calculate percentiles with Python

Data with eight `CustomerID` and a randomly calculated `Spend` values.

	CustomerID	Spend
0	0	137
1	1	335
2	2	172
3	3	355
4	4	303
5	5	233
6	6	244
7	7	229

Calculate percentiles with Python

```
spend_quartiles = pd.qcut(data['Spend'], q=4, labels=range(1,5))

data['Spend_Quartile'] = spend_quartiles

data.sort_values('Spend')
```

	CustomerID	Spend	Spend_Quartile
0	0	137	1
2	2	172	1
7	7	229	2
5	5	233	2
6	6	244	3
4	4	303	3
1	1	335	4
3	3	355	4

Assigning labels

- Highest score to the best metric - best is not always highest e.g. recency
- In this case, the label is inverse - the more recent the customer, the better

	CustomerID	Recency_Days
0	0	37
1	1	235
2	2	396
3	3	72
4	4	255
5	5	393
6	6	203
7	7	133

Assigning labels

```
# Create numbered labels
r_labels = list(range(4, 0, -1))

# Divide into groups based on quartiles
recency_quartiles = pd.qcut(data['Recency_Days'], q=4, labels=r_labels)

# Create new column
data['Recency_Quartile'] = recency_quartiles

# Sort recency values from lowest to highest
data.sort_values('Recency_Days')
```

Assigning labels

As you can see, the quartile labels are reversed, since the more recent customers are more valuable.

	CustomerID	Recency_Days	Recency_Quartile
0	0	37	4
3	3	72	4
7	7	133	3
6	6	203	3
1	1	235	2
4	4	255	2
5	5	393	1
2	2	396	1

Custom labels

We can define a list with string or any other values, depending on the use case.

```
# Create string labels
r_labels = ['Active', 'Lapsed', 'Inactive', 'Churned']

# Divide into groups based on quartiles
recency_quartiles = pd.qcut(data['Recency_Days'], q=4, labels=r_labels)

# Create new column
data['Recency_Quartile'] = recency_quartiles

# Sort values from lowest to highest
data.sort_values('Recency_Days')
```

Custom labels

Custom labels assigned to each quartile

	CustomerID	Recency_Days	Recency_Quartile
0	0	37	Active
3	3	72	Active
7	7	133	Lapsed
6	6	203	Lapsed
1	1	235	Inactive
4	4	255	Inactive
5	5	393	Churned
2	2	396	Churned

CUSTOMER SEGMENTATION IN PYTHON

Let's practice with percentiles!

CUSTOMER SEGMENTATION IN PYTHON

Recency, Frequency, Monetary Value calculation

Karolis Urbonas
Head of Data Science, Amazon

Definitions

- Recency - days since last customer transaction
- Frequency - number of transactions in the last 12 months
- Monetary Value - total spend in the last 12 months

Dataset and preparations

- Same `online` dataset like in the previous lessons
- Need to do some data preparation
- New `TotalSum` column = `Quantity` x `UnitPrice`.

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	TotalSum
416792	572558	22745	POPPY'S PLAYHOUSE BEDROOM	6	2011-10-25	2.10	14286	United Kingdom	12.60
482904	577485	23196	VINTAGE LEAF MAGNETIC NOTEPAD	1	2011-11-20	1.45	16360	United Kingdom	1.45
263743	560034	23299	FOOD COVER WITH BEADS SET 2	6	2011-07-14	3.75	13933	United Kingdom	22.50
495549	578307	72349B	SET/6 PURPLE BUTTERFLY T-LIGHTS	1	2011-11-23	2.10	17290	United Kingdom	2.10
204384	554656	21756	BATH BUILDING BLOCK WORD	3	2011-05-25	5.95	17663	United Kingdom	17.85

Data preparation steps

We're starting with a pre-processed `online` DataFrame with only the latest 12 months of data:

```
print('Min:{}; Max:{}'.format(min(online.InvoiceDate),
                              max(online.InvoiceDate)))

Min:2010-12-10; Max:2011-12-09
```

Let's create a hypothetical **snapshot_day** data as if we're doing analysis recently.

```
snapshot_date = max(online.InvoiceDate) + datetime.timedelta(days=1)
```

Calculate RFM metrics

```
# Aggregate data on a customer level
datamart = online.groupby(['CustomerID']).agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'count',
    'TotalSum': 'sum'})

# Rename columns for easier interpretation
datamart.rename(columns = {'InvoiceDate': 'Recency',
                           'InvoiceNo': 'Frequency',
                           'TotalSum': 'MonetaryValue'}, inplace=True)

# Check the first rows
datamart.head()
```

Final RFM values

Our table for RFM segmentation is completed!

CustomerID	Recency	Frequency	MonetaryValue
12747	3	25	948.70
12748	1	888	7046.16
12749	4	37	813.45
12820	4	17	268.02
12822	71	9	146.15

CUSTOMER SEGMENTATION IN PYTHON

Let's practice calculating RFM values!

CUSTOMER SEGMENTATION IN PYTHON

Building RFM segments

Karolis Urbonas
Head of Data Science, Amazon

Data

- Dataset we created previously
- Will calculate quartile value for each column and name then `R`, `F`, `M`

CustomerID	Recency	Frequency	MonetaryValue
12747	3	25	948.70
12748	1	888	7046.16
12749	4	37	813.45
12820	4	17	268.02
12822	71	9	146.15

Recency quartile

```
r_labels = range(4, 1, -1)

r_quartiles = pd.qcut(datamart['Recency'], 4, labels = r_labels)

datamart = datamart.assign(R = r_quartiles.values)
```

CustomerID	Recency	Frequency	MonetaryValue	R
12747	3	25	948.70	4
12748	1	888	7046.16	4
12749	4	37	813.45	4
12820	4	17	268.02	4
12822	71	9	146.15	2

Frequency and Monetary quartiles

```
f_labels = range(1,5)
m_labels = range(1,5)

f_quartiles = pd.qcut(datamart['Frequency'], 4, labels = f_labels)
m_quartiles = pd.qcut(datamart['MonetaryValue'], 4, labels = m_labels)

datamart = datamart.assign(F = f_quartiles.values)
datamart = datamart.assign(M = m_quartiles.values)
```

CustomerID	Recency	Frequency	MonetaryValue	R	F	M
12747	3	25	948.70	4	4	4
12748	1	888	7046.16	4	4	4
12749	4	37	813.45	4	4	4
12820	4	17	268.02	4	3	3
12822	71	9	146.15	2	2	3

Build RFM Segment and RFM Score

- Concatenate RFM quartile values to `RFM_Segment`
- Sum RFM quartiles values to `RFM_Score`

```
def join_rfm(x): return str(x['R']) + str(x['F']) + str(x['M'])

datamart['RFM_Segment'] = datamart.apply(join_rfm, axis=1)

datamart['RFM_Score'] = datamart[['R','F','M']].sum(axis=1)
```

Final result

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_Segment	RFM_Score
18108	255	4	58.60	1	1	1	111	3.0
13803	256	3	57.70	1	1	1	111	3.0
12922	162	4	57.24	1	1	1	111	3.0
13304	352	4	57.21	1	1	1	111	3.0
17496	359	2	57.15	1	1	1	111	3.0
16063	261	4	57.10	1	1	1	111	3.0
17531	191	2	57.00	1	1	1	111	3.0
14206	240	3	57.00	1	1	1	111	3.0
13784	219	2	55.80	1	1	1	111	3.0
14476	258	4	55.75	1	1	1	111	3.0

CUSTOMER SEGMENTATION IN PYTHON

Let's practice building RFM segments

CUSTOMER SEGMENTATION IN PYTHON

Analyzing RFM segments

Karolis Urbonas
Head of Data Science, Amazon

Largest RFM segments

```
datamart.groupby('RFM_Segment').size().sort_values(ascending=False)[:10]
```

```
RFM_Segment
444    372
111    345
211    169
344    156
233    129
222    128
333    120
122    117
311    114
433    113
dtype: int64
```

Filtering on RFM segments

- Select bottom RFM segment "111" and view top 5 rows

```
datamart[datamart['RFM_Segment']=='111'][:5]
```

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_Segment	RFM_Score
12837	174	2	10.55	1	1	1	111	3.0
12852	295	2	32.55	1	1	1	111	3.0
12902	265	4	42.03	1	1	1	111	3.0
12915	149	2	35.90	1	1	1	111	3.0
12922	162	4	57.24	1	1	1	111	3.0

Summary metrics per RFM Score

```
datamart.groupby('RFM_Score').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': ['mean', 'count'] }).round(1)
```

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
RFM_Score				
3.0	246.9	2.1	28.4	345
4.0	162.2	3.1	47.8	337
5.0	138.9	4.3	78.2	393
6.0	101.0	6.3	146.3	444
7.0	78.0	8.5	160.2	382
8.0	62.6	12.8	196.3	376
9.0	46.8	16.7	330.3	345
10.0	31.9	24.0	443.1	355
11.0	21.8	38.9	705.3	294
12.0	8.0	75.6	1653.9	372

Grouping into named segments

Use RFM score to group customers into **Gold**, **Silver** and **Bronze** segments.

```
def segment_me(df):
    if df['RFM_Score'] >= 9:
        return 'Gold'
    elif (df['RFM_Score'] >= 5) and (df['RFM_Score'] < 9):
        return 'Silver'
    else:
        return 'Bronze'
```

```
datamart['General_Segment'] = datamart.apply(segment_me, axis=1)
```

```
datamart.groupby('General_Segment').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': ['mean', 'count']
}).round(1)
```

New segments and their values

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
General_Segment				
1. Gold	27.0	39.4	800.8	1366
2. Silver	95.8	7.9	144.6	1595
3. Bronze	205.0	2.6	38.0	682

CUSTOMER SEGMENTATION IN PYTHON

Practice building custom segments

CUSTOMER SEGMENTATION IN PYTHON

Data pre-processing for k-means clustering

Karolis Urbonas
Head of Data Science, Amazon

Advantages of k-means clustering

- One of the most popular unsupervised learning method
- Simple and fast
- Works well*

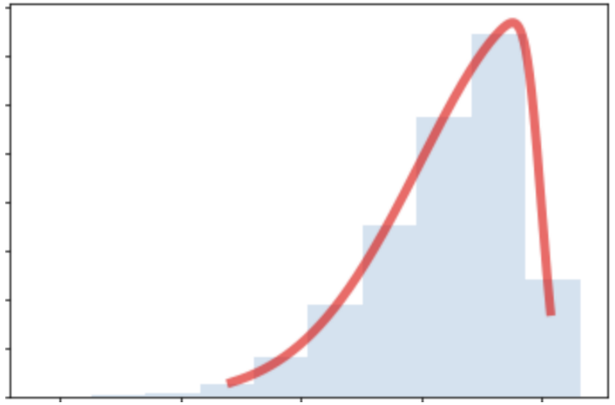
** with certain assumptions about the data*

Key k-means assumptions

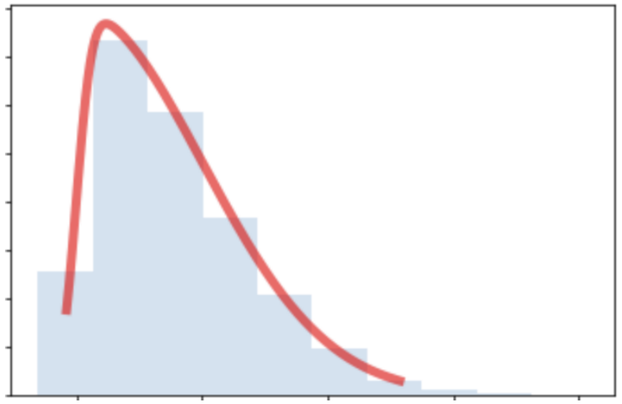
- Symmetric distribution of variables (not skewed)
- Variables with same average values
- Variables with same variance

Skewed variables

- Left-skewed

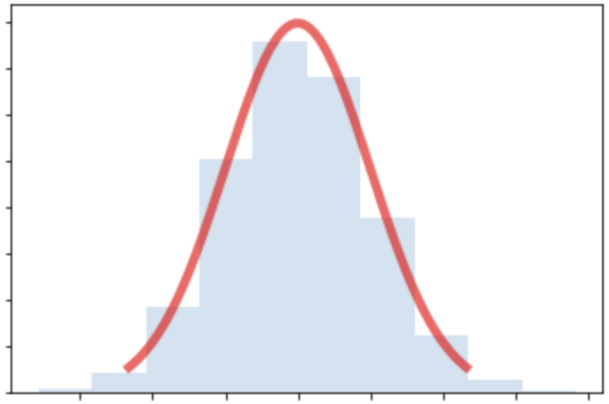


- Right-skewed



Skewed variables

- Skew removed with logarithmic transformation



Variables on the same scale

- K-means assumes equal mean
- And equal variance
- It's not the case with RFM data

```
datamart_rfm.describe()
```

	Recency	Frequency	MonetaryValue
count	3643.00000	3643.000000	3643.000000
mean	90.43563	18.714247	370.694387
std	94.44651	43.754468	1347.443451
min	1.00000	1.000000	0.650000
25%	19.00000	4.000000	58.705000
50%	51.00000	9.000000	136.370000
75%	139.00000	21.000000	334.350000
max	365.00000	1497.000000	48060.350000

CUSTOMER SEGMENTATION IN PYTHON

Let's review the concepts

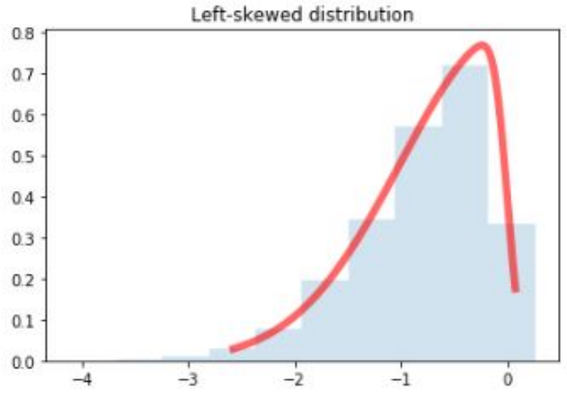
CUSTOMER SEGMENTATION IN PYTHON

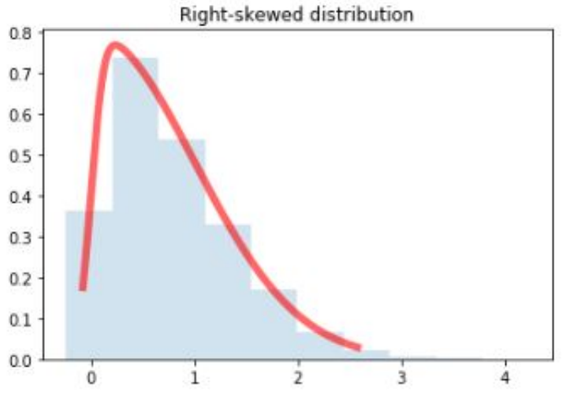
Managing skewed variables

Karolis Urbonas
Head of Data Science, Amazon

Identifying skewness

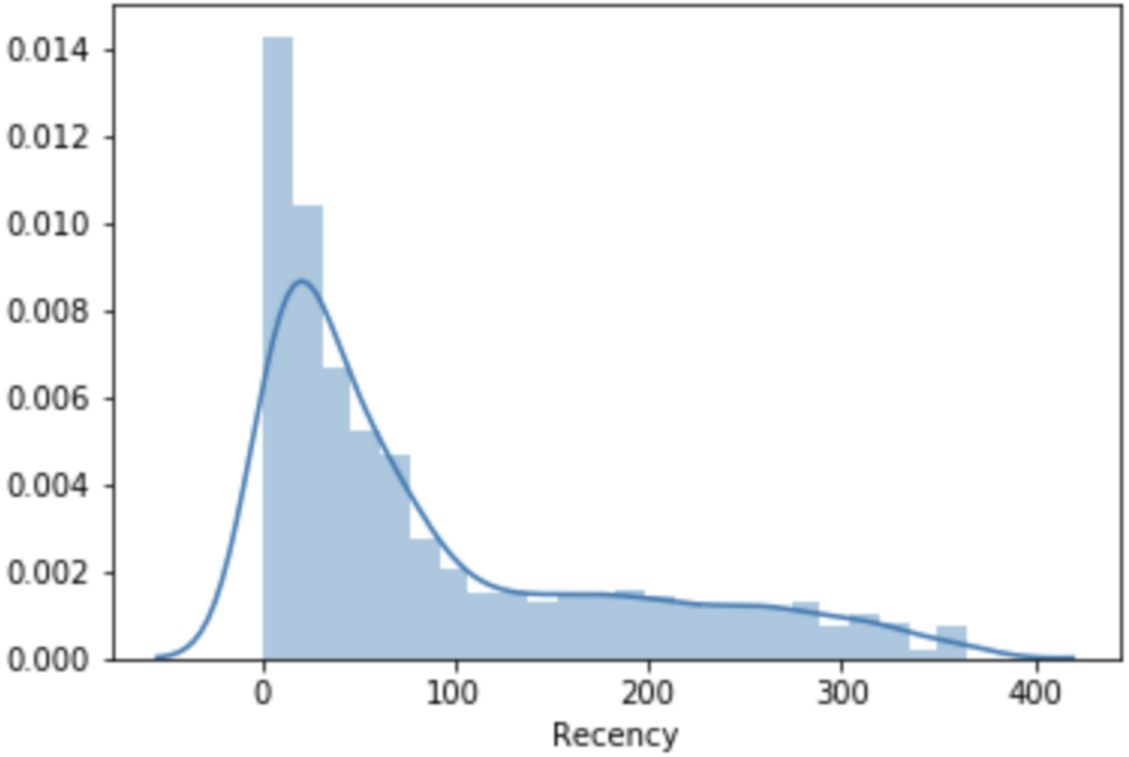
- Visual analysis of the distribution
- If it has a tail - it's skewed





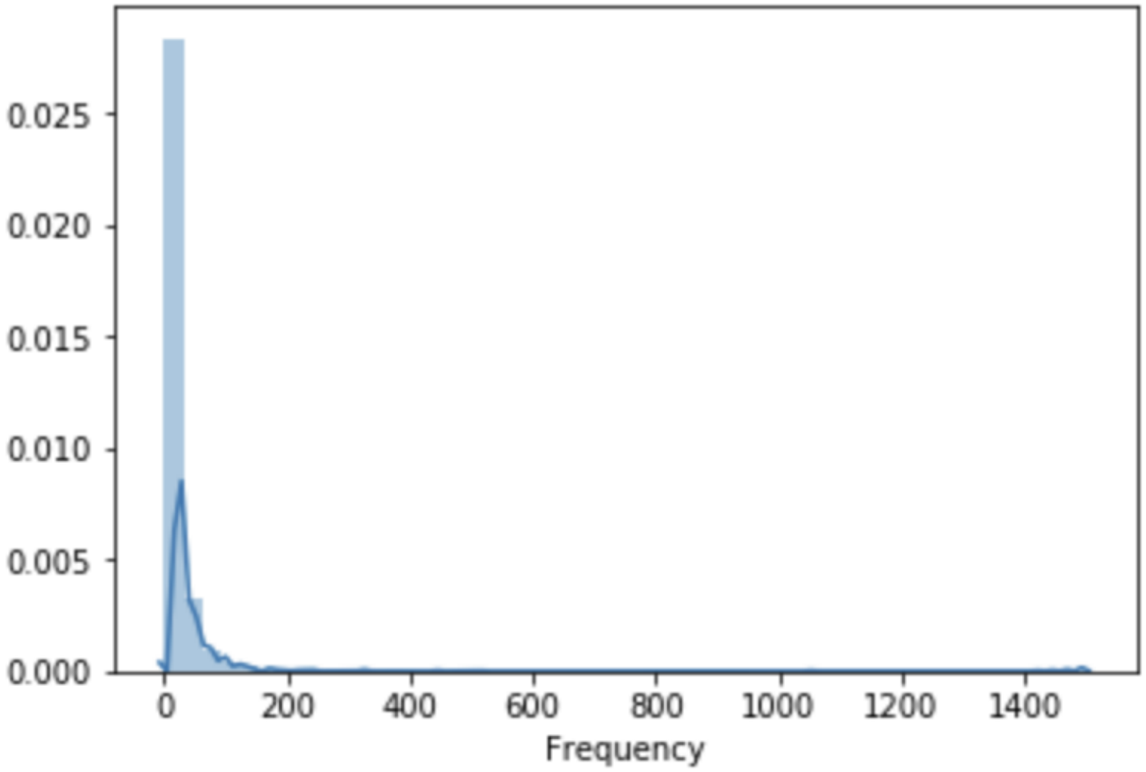
Exploring distribution of Recency

```
import seaborn as sns
from matplotlib import pyplot as plt

sns.distplot(datamart['Recency'])
plt.show()
```



Exploring distribution of Frequency

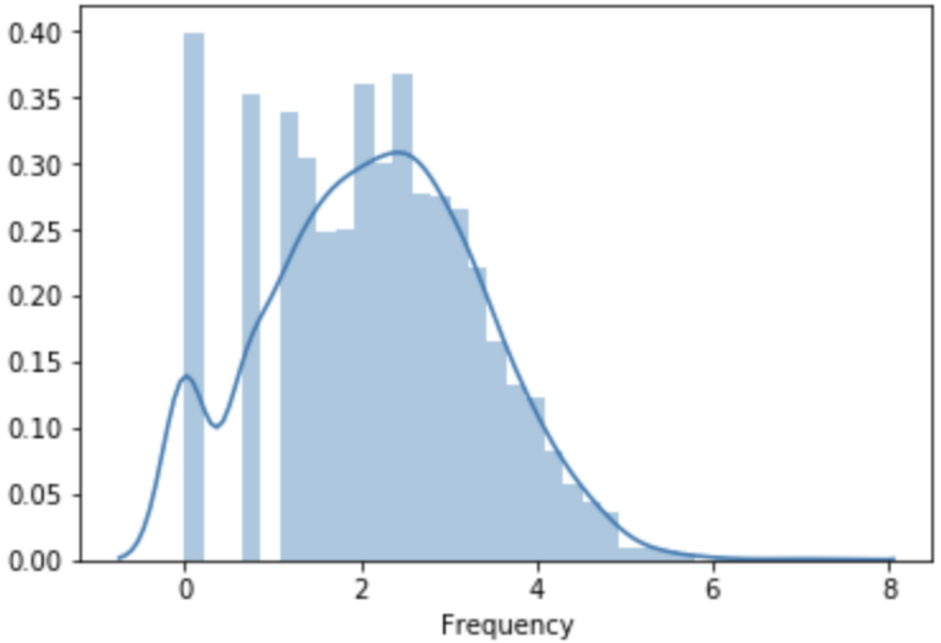
```
sns.distplot(datamart['Frequency'])
plt.show()
```



Data transformations to manage skewness

- Logarithmic transformation (positive values only)

```
import numpy as np
frequency_log= np.log(datamart['Frequency'])

sns.distplot(frequency_log)
plt.show()
```



Dealing with negative values

- Adding a constant before log transformation
- Cube root transformation

CUSTOMER SEGMENTATION IN PYTHON

Let's practice how to identify and manage skewed variables!

CUSTOMER SEGMENTATION IN PYTHON

Centering and scaling variables

Karolis Urbonas
Head of Data Science, Amazon

Identifying an issue

- Analyze key statistics of the dataset
- Compare mean and standard deviation

```
datamart_rfm.describe()
```

	Recency	Frequency	MonetaryValue
count	3643.00000	3643.000000	3643.000000
mean	90.43563	18.714247	370.694387
std	94.44651	43.754468	1347.443451
min	1.00000	1.000000	0.650000
25%	19.00000	4.000000	58.705000
50%	51.00000	9.000000	136.370000
75%	139.00000	21.000000	334.350000
max	365.00000	1497.000000	48060.350000

Centering variables with different means

- K-means works well on variables with the same mean
- Centering variables is done by subtracting average value from each observation

```
datamart_centered = datamart_rfm - datamart_rfm.mean()
datamart_centered.describe().round(2)
```

	Recency	Frequency	MonetaryValue
count	3643.00	3643.00	3643.00
mean	0.00	-0.00	0.00
std	94.45	43.75	1347.44
min	-89.44	-17.71	-370.04
25%	-71.44	-14.71	-311.99
50%	-39.44	-9.71	-234.32
75%	48.56	2.29	-36.34
max	274.56	1478.29	47689.66

Scaling variables with different variance

- K-means works better on variables with the same variance / standard deviation
- Scaling variables is done by dividing them by standard deviation of each

```
datamart_scaled = datamart_rfm / datamart_rfm.std()
datamart_scaled.describe().round(2)
```

	Recency	Frequency	MonetaryValue
count	3643.00	3643.00	3643.00
mean	0.96	0.43	0.28
std	1.00	1.00	1.00
min	0.01	0.02	0.00
25%	0.20	0.09	0.04
50%	0.54	0.21	0.10
75%	1.47	0.48	0.25
max	3.86	34.21	35.67

Combining centering and scaling

- Subtract mean and divide by standard deviation manually
- Or use a scaler from `scikit-learn` library (returns `numpy.ndarray` object)

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(datamart_rfm)
datamart_normalized = scaler.transform(datamart_rfm)
```

```
print('mean: ', datamart_normalized.mean(axis=0).round(2))
print('std: ', datamart_normalized.std(axis=0).round(2))

mean:  [-0. -0.  0.]
std:  [1. 1. 1.]
```

CUSTOMER SEGMENTATION IN PYTHON

Test different approaches by yourself!

CUSTOMER SEGMENTATION IN PYTHON

Sequence of structuring pre-processing steps

Karolis Urbonas
Head of Data Science, Amazon

Why the sequence matters?

- Log transformation only works with positive data
- Normalization forces data to have negative values and `log` will not work

Sequence

1. Unskew the data - log transformation
2. Standardize to the same average values
3. Scale to the same standard deviation
4. Store as a separate array to be used for clustering

Coding the sequence

Unskew the data with log transformation

```
import numpy as np
datamart_log = np.log(datamart_rfm)
```

Normalize the variables with `StandardScaler`

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(datamart_log)
```

Store it separately for clustering

```
datamart_normalized = scaler.transform(datamart_log)
```

CUSTOMER SEGMENTATION IN PYTHON

Practice on RFM data!

CUSTOMER SEGMENTATION IN PYTHON

Practical implementation of k-means clustering

Karolis Urbonas
Head of Data Science, Amazon

Key steps

- Data pre-processing
- Choosing a number of clusters
- Running k-means clustering on pre-processed data
- Analyzing average RFM values of each cluster

Data pre-processing

We've completed the pre-processing steps and have these two objects:

- `datamart_rfm`
- `datamart_normalized`

Code from previous lesson:

```
import numpy as np
datamart_log = np.log(datamart_rfm)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(datamart_log)

datamart_normalized = scaler.transform(datamart_log)
```

Methods to define the number of clusters

- Visual methods - elbow criterion
- Mathematical methods - silhouette coefficient
- Experimentation and interpretation

Running k-means

Import `KMeans` from `sklearn` library and initialize it as `kmeans`

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2, random_state=1)
```

Compute k-means clustering on pre-processed data

```
kmeans.fit(datamart_normalized)
```

Extract cluster labels from `labels_` attribute

```
cluster_labels = kmeans.labels_
```

Analyzing average RFM values of each cluster

Create a cluster label column in the **original** DataFrame:

```
datamart_rfm_k2 = datamart_rfm.assign(Cluster = cluster_labels)
```

Calculate average RFM values and size for each cluster:

```
datamart_rfm_k2.groupby(['Cluster']).agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': ['mean', 'count'],
}).round(0)
```

Analyzing average RFM values of each cluster

The result of a simple 2-cluster solution:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
cluster				
0	137.0	5.0	92.0	2023
1	32.0	35.0	719.0	1620

CUSTOMER SEGMENTATION IN PYTHON

Let's practice running k-means clustering!

CUSTOMER SEGMENTATION IN PYTHON

Choosing number of clusters

Karolis Urbonas
Head of Data Science, Amazon

Methods

- Visual methods - elbow criterion
- Mathematical methods - silhouette coefficient
- Experimentation and interpretation

Elbow criterion method

- Plot the number of clusters against within-cluster sum-of-squared-errors (SSE) - *sum of squared distances from every data point to their cluster center*
- Identify an "elbow" in the plot
- Elbow - a point representing an "optimal" number of clusters

Elbow criterion method

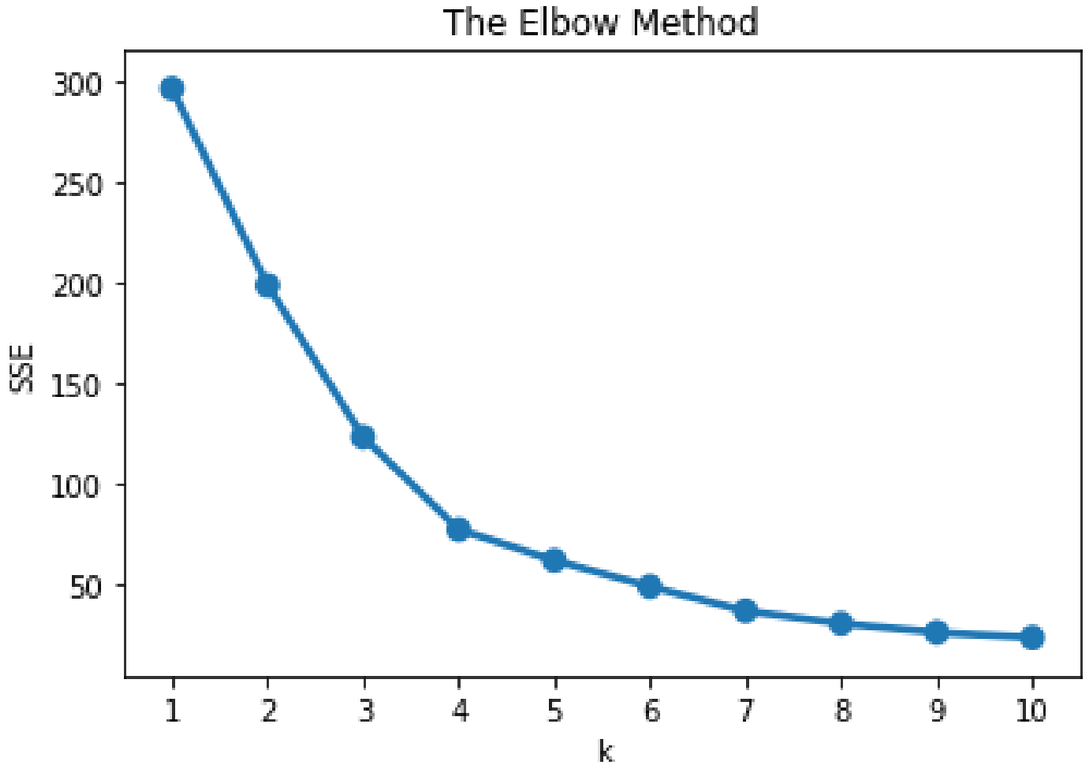
```
# Import key libraries
from sklearn.cluster import KMeans
import seaborn as sns
from matplotlib import pyplot as plt
```

```
# Fit KMeans and calculate SSE for each *k*
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1)
    kmeans.fit(data_normalized)
    sse[k] = kmeans.inertia_ # sum of squared distances to closest cluster cente
```

```
# Plot SSE for each *k*
plt.title('The Elbow Method')
plt.xlabel('k'); plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```

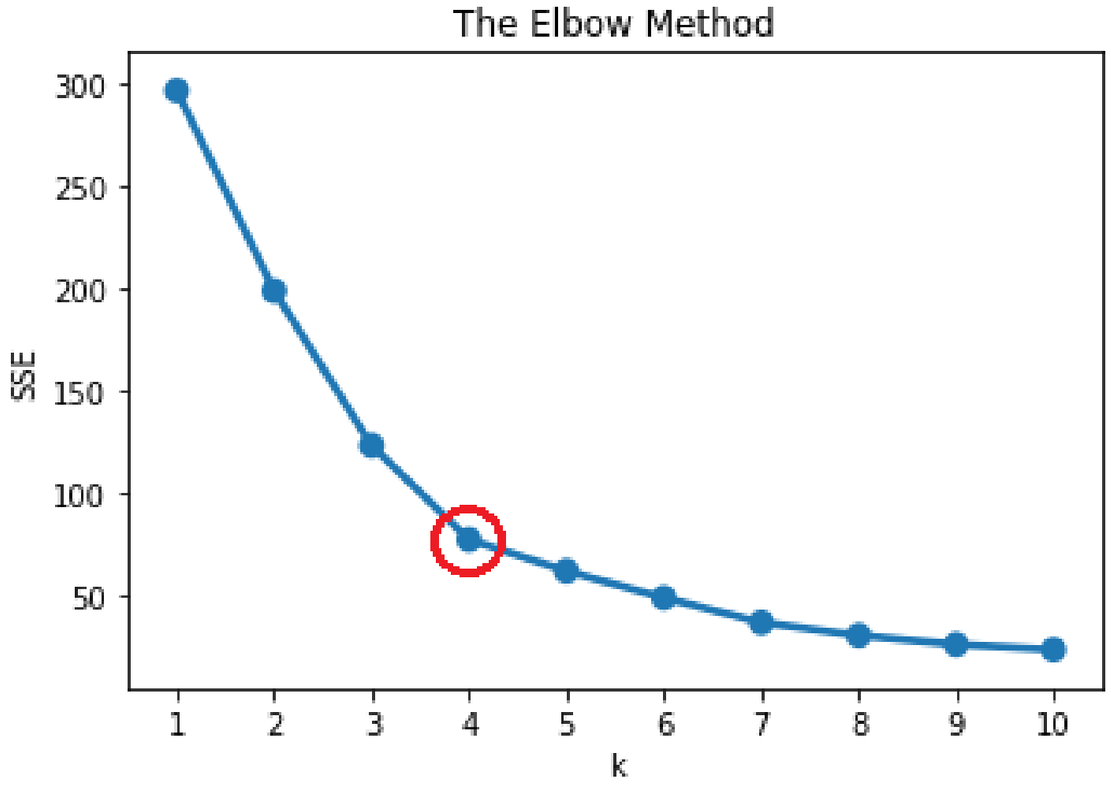
Elbow criterion method

The elbow criterion chart:



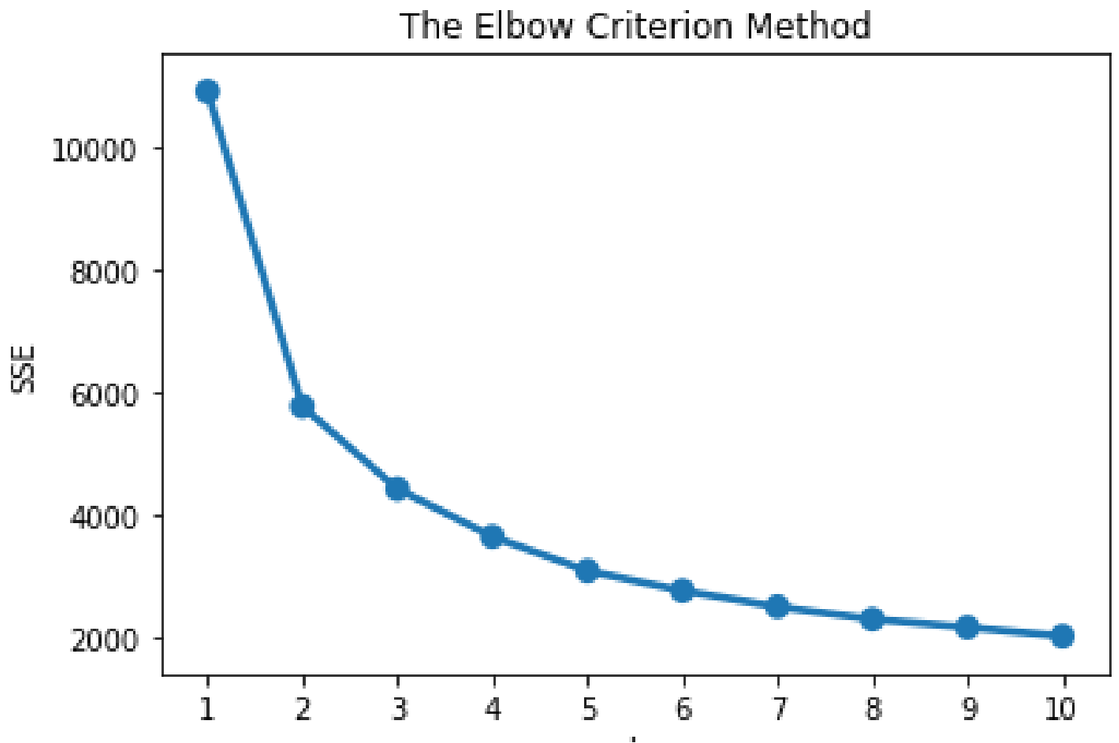
Elbow criterion method

The elbow criterion chart:



Using elbow criterion method

- Best to choose the point on elbow, or the next point
- Use as a guide but test multiple solutions
- Elbow plot built on `datamart_rfm`



Experimental approach - analyze segments

- Build clustering at and around elbow solution
- Analyze their properties - average RFM values
- Compare against each other and choose one which makes most business sense

Experimental approach - analyze segments

- Previous 2-cluster solution

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
cluster				
0	137.0	5.0	92.0	2023
1	32.0	35.0	719.0	1620

- 3-cluster solution on the same normalized RFM dataset

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
cluster				
0	16.0	50.0	1051.0	901
1	167.0	3.0	53.0	1156
2	77.0	12.0	216.0	1586

CUSTOMER SEGMENTATION IN PYTHON

Let's practice finding the optimal number of clusters!

CUSTOMER SEGMENTATION IN PYTHON

Profile and interpret segments

Karolis Urbonas
Head of Data Science, Amazon

Approaches to build customer personas

- Summary statistics for each cluster e.g. average RFM values
- Snake plots (from market research
- Relative importance of cluster attributes compared to population

Summary statistics of each cluster

- Run k-means segmentation for several **k** values around the recommended value.
- Create a cluster label column in the **original** DataFrame:

```
datamart_rfm_k2 = datamart_rfm.assign(Cluster = cluster_labels)
```

Calculate average RFM values and sizes for each cluster:

```
datamart_rfm_k2.groupby(['Cluster']).agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': ['mean', 'count'],
}).round(0)
```

- Repeat the same for **k=3**

Summary statistics of each cluster

- Compare average RFM values of each clustering solution

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
cluster				
0	137.0	5.0	92.0	2023
1	32.0	35.0	719.0	1620

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
cluster				
0	16.0	50.0	1051.0	901
1	167.0	3.0	53.0	1156
2	77.0	12.0	216.0	1586

Snake plots to understand and compare segments

- Market research technique to compare different segments
- Visual representation of each segment's attributes
- Need to first normalize data (center & scale)
- Plot each cluster's average normalized values of each attribute

Prepare data for a snake plot

Transform `datamart_normalized` as DataFrame and add a `Cluster` column

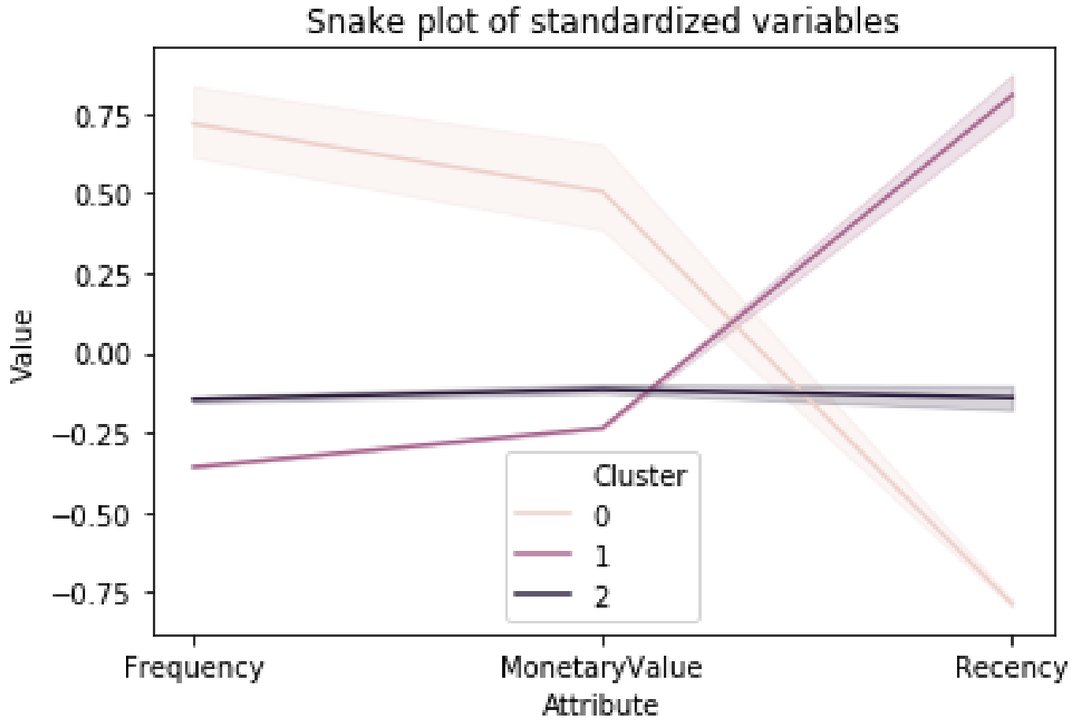
```
datamart_normalized = pd.DataFrame(datamart_normalized,
                                   index=datamart_rfm.index,
                                   columns=datamart_rfm.columns)
datamart_normalized['Cluster'] = datamart_rfm_k3['Cluster']
```

Melt the data into a long format so RFM values and metric names are stored in 1 column each

```
datamart_melt = pd.melt(datamart_normalized.reset_index(),
                    id_vars=['CustomerID', 'Cluster'],
                    value_vars=['Recency', 'Frequency', 'MonetaryValue'],
                    var_name='Attribute',
                    value_name='Value')
```

Visualize a snake plot

```
plt.title('Snake plot of standardized variables')
sns.lineplot(x="Attribute", y="Value", hue='Cluster', data=datamart_melt)
```



Relative importance of segment attributes

- Useful technique to identify relative importance of each segment's attribute
- Calculate average values of each cluster
- Calculate average values of population
- Calculate importance score by dividing them and subtracting 1 (*ensures 0 is returned when cluster average equals population average*)

```
cluster_avg = datamart_rfm_k3.groupby(['Cluster']).mean()
```

```
population_avg = datamart_rfm.mean()
```

```
relative_imp = cluster_avg / population_avg - 1
```

Analyze and plot relative importance

- The further a ratio is from 0, the more important that attribute is for a segment relative to the total population.

```
relative_imp.round(2)
```

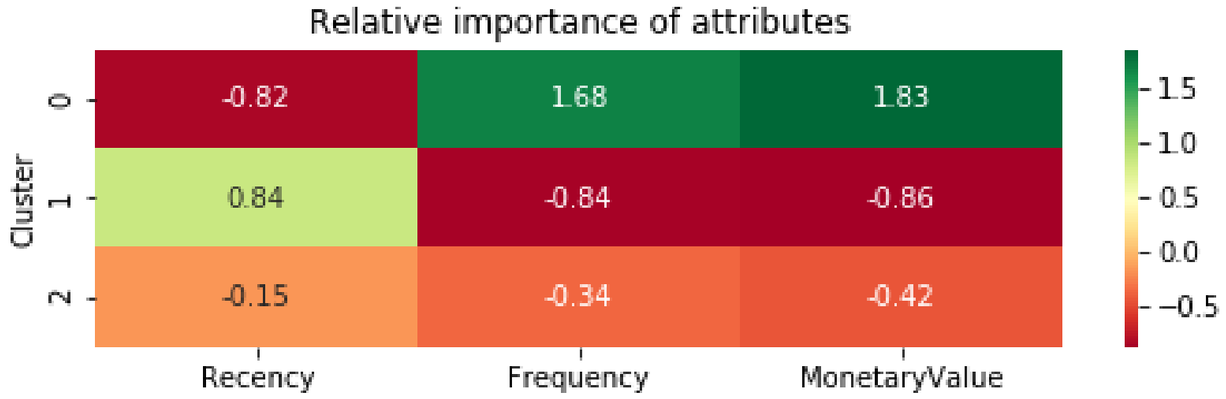
```
         Recency  Frequency  MonetaryValue
Cluster
0          -0.82       1.68           1.83
1           0.84      -0.84          -0.86
2          -0.15      -0.34          -0.42
```

Plot a heatmap for easier interpretation:

```
plt.figure(figsize=(8, 2))
plt.title('Relative importance of attributes')
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='RdYlGn')
plt.show()
```

Relative importance heatmap

Heatmap plot:



vs. printed output:

```
           Recency  Frequency  MonetaryValue
Cluster
0            -0.82       1.68           1.83
1             0.84      -0.84          -0.86
2            -0.15      -0.34          -0.42
```

CUSTOMER SEGMENTATION IN PYTHON

Your time to experiment with different customer profiling techniques!

CUSTOMER SEGMENTATION IN PYTHON

Implement end-to-end segmentation solution

Karolis Urbonas
Head of Data Science, Amazon

Key steps of the segmentation project

- Gather data - updated data with an additional variable
- Pre-process the data
- Explore the data and decide on the number of clusters
- Run k-means clustering
- Analyze and visualize results

Updated RFM data

- Same RFM values plus additional `Tenure` variable
- Tenure - time since the first transaction
- Defines how long the customer has been with the company

CustomerID	Recency	Frequency	MonetaryValue	Tenure
12747	3	25	948.70	362
12748	1	888	7046.16	365
12749	4	37	813.45	214
12820	4	17	268.02	327
12822	71	9	146.15	88

Goals for this project

- Remember key pre-processing rules
- Apply data exploration techniques
- Practice running several k-means iterations
- Analyze results quantitatively and visually

CUSTOMER SEGMENTATION IN PYTHON

Let's dig in!

CUSTOMER SEGMENTATION IN PYTHON

Final thoughts

Karolis Urbonas
Head of Data Science, Amazon

What you have learned

- Cohort analysis and visualization
- RFM segmentation
- Data pre-processing for k-means
- Customer segmentation with k-means
  - Evaluating number of clusters
  - Reviewing and visualizing segmentation solutions

CUSTOMER SEGMENTATION IN PYTHON

Congratulations!