Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

# BIG DATA IN MACHINE LEARNING

## Bài 5: Data Pre-processing

Phòng LT & Mạng

https://csc.edu.vn/lap-trinh-va-csdl/Big-Data-in-Machine-Learning_198

2020


## Nội dung


1. Data Cleaning
2. Feature Engineering
3. Một số kỹ thuật khác

## Data Cleaning


### ❑Dữ liệu có thể có các vấn đề sau:

- Thiếu dữ liệu
- Nhập liệu sai
- Trùng lắp
- Định dạng sai
- Không liên quan (không cần thiết cho quá trình phân tích)
- ...



## Data Cleaning


### ❑Xóa các thuộc tính không liên quan

- Dùng <New_DataFrame> = <DataFrame_name>.drop(*[list_column_names])
  - Ví dụ: Bỏ 2 thuộc tính 'STREETNUMBERNUMERIC' (postal address number on the home) và 'LOTSIZEDIMENSIONS' (free text describing the lot shape) trong dữ liệu.

## Data Cleaning


```
df_sub.show(5)
```

```
+------------------+---------+----------------+---------------+-----+
|STREETNUMBERNUMERIC|FIREPLACES|LOTSIZEDIMENSIONS|       LISTTYPE|ACRES|
+------------------+---------+----------------+---------------+-----+
|             11511|        0|         279X200|Exclusive Right| 1.28|
|             11200|        0|         100x140|Exclusive Right| 0.32|
|              8583|        0|         120x296|Exclusive Right|0.822|
|              9350|        1|         208X208|Exclusive Right| 0.94|
|              2915|        1|         116x200|Exclusive Right|  0.0|
+------------------+---------+----------------+---------------+-----+
only showing top 5 rows
```

```
# Drop columns in list
df_sub = df_sub.drop(*['STREETNUMBERNUMERIC', 'LOTSIZEDIMENSIONS'])
df_sub.show(5)
```

```
+----------+---------------+-----+
|FIREPLACES|       LISTTYPE|ACRES|
+----------+---------------+-----+
|         0|Exclusive Right| 1.28|
|         0|Exclusive Right| 0.32|
|         0|Exclusive Right|0.822|
|         1|Exclusive Right| 0.94|
|         1|Exclusive Right|  0.0|
+----------+---------------+-----+
only showing top 5 rows
```



## Data Cleaning


- **Lọc dữ liệu theo text**
  - where(condition)
  - filter(condition)
  - like()
  - ~

# Data Cleaning

- Ví dụ: Lọc dữ liệu trong cột 'ASSUMABLEMORTGAGE'

```
# Inspect unique values in the column 'ASSUMABLEMORTGAGE'
df_sub.select(['ASSUMABLEMORTGAGE']).distinct().show()
```

```
+--------------------+
|  ASSUMABLEMORTGAGE|
+--------------------+
|  Yes w/ Qualifying|
| Information Coming|
|                null|
|Yes w/No Qualifying|
|      Not Assumable|
+--------------------+
```

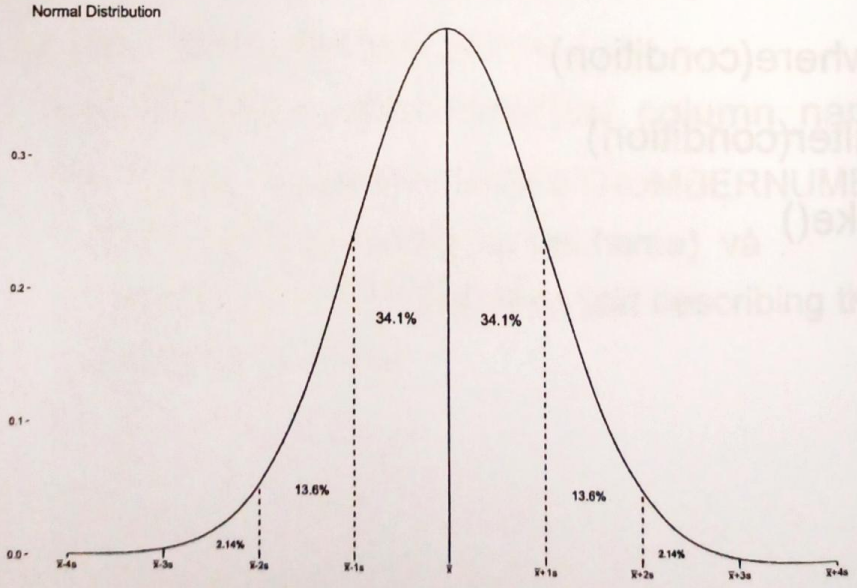
```
print("Before:", df_sub.count())
# List of possible values containing 'yes'
yes_values = ['Yes w/ Qualifying', 'Yes w/No Qualifying']

# Filter the text values out of df_sub but keep null values
text_filter = ~df_sub['ASSUMABLEMORTGAGE'].isin(yes_values) | df_sub['ASSUMABLEMORTGAGE'].isNull()
df_sub = df.where(text_filter)

# Print count of remaining records
print("After:", df_sub.count())
```

```
Before: 5000
After: 4976
```



# Data Cleaning

## ❑ Xóa dữ liệu outlier theo phân phối chuẩn



## Data Cleaning


- Ví dụ:

```
from pyspark.sql.functions import mean, stddev, col, log

df_sub = df_sub.withColumn("log_SalesClosePrice", log(col("SalesClosePrice")))

df_sub.count()

4976


# Calculate values used for outlier filtering
mean_val = df_sub.agg({'log_SalesClosePrice': 'mean'}).collect()[0][0]
stddev_val = df_sub.agg({'log_SalesClosePrice': 'stddev'}).collect()[0][0]

# Create three standard deviation (μ ± 3σ) lower and upper bounds for data
low_bound = mean_val - (3 * stddev_val)
hi_bound = mean_val + (3 * stddev_val)

# Filter the data to fit between the lower and upper bounds
df_sub = df_sub.where((df_sub['log_SalesClosePrice'] < hi_bound) & (df_sub['log_SalesClosePrice'] > low_bound))

df_sub.count()

4946
```



## Data Cleaning


### ❑Xóa dữ liệu NA, NULL

- Dùng <DataFrame_name>.dropna()
  - how = “all” : xóa record khi tất cả các value đều null
  - how = “any” : xóa record khi chỉ cần có một value null
  - thresh = x : xóa record khi tối thiểu x colums có null value
  - subset[‘column_name_1’, ‘column_name_2’, ...] : xóa record theo các cột trong subset

## Data Cleaning


- Ví dụ:

```
# drop records if both ListPrice and SalesClosePrice are NULL
df_sub = df_sub.dropna(how='all', subset=['ListPrice', 'SalesClosePrice'])

# Drop any records with NULL values
df_sub = df_sub.dropna()
# Drop records where at least two columns have NULL values
df_sub = df_sub.dropna(thresh=2)
```



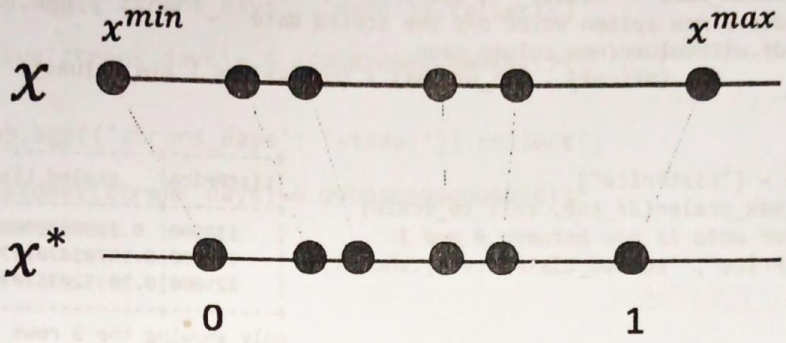
## Data Cleaning


### ❑ Xóa dữ liệu trùng lắp

- Khi hai hay nhiều record có cùng thông tin
- Sau khi xóa cột hoặc kết hợp dataset, cần kiểm tra xem dữ liệu có bị trùng lắp hay không, nếu có thì xóa.
- Dùng <DataFrame_name>. drop_duplicates()

## ❑MinMax Scaling

$$x_{i,j}^{*} = \frac{x_{i,j} - x_j^{min}}{x_j^{max} - x_j^{min}}$$








- Ví dụ:

```
# Calc max and min for new column
print(df_sub.agg({'SalesClosePrice': 'max'}).collect())
print(df.agg({'SalesClosePrice': 'min'}).collect())

# Define max and min values and collect them
max_price = df_sub.agg({'SalesClosePrice': 'max'}).collect()[0][0]
min_price = df_sub.agg({'SalesClosePrice': 'min'}).collect()[0][0]

# Create a new column based off the scaled data
df_sub = df_sub.withColumn('scaled_price',
                (df['SalesClosePrice'] - min_price) / (max_price - min_price))

# Calc max and min for new column
print(df_sub.agg({'scaled_price': 'max'}).collect())
print(df_sub.agg({'scaled_price': 'min'}).collect())
```

```
[Row(max(SalesClosePrice)=920000)]
[Row(min(SalesClosePrice)=48000)]
[Row(max(scaled_price)=1.0)]
[Row(min(scaled_price)=0.0)]
```

```
df_sub.select('scaled_price').show(5)
+--------------------+
|        scaled_price|
+--------------------+
|0.09545983701979045|
|0.15017462165308498|
| 0.1909196740395809|
|0.23748544819557627|
|  0.219906868451688|
+--------------------+
only showing top 5 rows
```

## Data Cleaning

- Tổng quát:

```
def min_max_scaler(df, cols_to_scale):
    # Takes a dataframe and list of columns to minmax scale. Returns a dataframe.
    for col in cols_to_scale:
        # Define min and max values and collect them
        max_values = df.agg({col: 'max'}).collect()[0][0]
        min_values = df.agg({col: 'min'}).collect()[0][0]
        new_column_name = 'scaled_' + col
        # Create a new column based off the scaled data
        df = df.withColumn(new_column_name,
                           (df[col] - min_values) / (max_values - min_values))
    return df
```

```
cols_to_scale = ["ListPrice"]
df_sub = min_max_scaler(df_sub, cols_to_scale)
# Show that our data is now between 0 and 1
df_sub[['ListPrice', 'scaled_ListPrice']].show(3)
```

```
+---------+-------------------+
|ListPrice|   scaled_ListPrice|
+---------+-------------------+
|   139900| 0.1099009900990099|
|   210000|0.18701870187018702|
|   225000|0.20352035203520352|
+---------+-------------------+
only showing top 3 rows
```



## Data Cleaning

### ❑ Standard Scaling – Standardization (Z-score Normalization)

- Chuyển dữ liệu sang standard normal distribution
  - $\bar{x}$ : Mean
  - $\sigma$ : Standard Deviation

$$x' = \frac{x - \bar{x}}{\sigma}$$

## Data Cleaning

- Ví dụ:

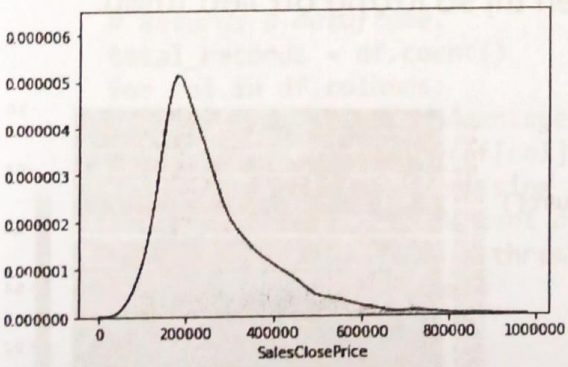
```
mean_days = df_sub.agg({'DAYSONMARKET': 'mean'}).collect()[0][0]
stddev_days = df_sub.agg({'DAYSONMARKET': 'stddev'}).collect()[0][0]
# Create a new column with the scaled data
df_sub = df_sub.withColumn("ztrans_days",
                           (df_sub['DAYSONMARKET'] - mean_days) / stddev_days)
df_sub.agg({'ztrans_days': 'mean'}).collect()

[Row(avg(ztrans_days)=-5.459082886955885e-17)]

df_sub.agg({'ztrans_days': 'stddev'}).collect()

[Row(stddev(ztrans_days)=0.9999999999999994)]
```



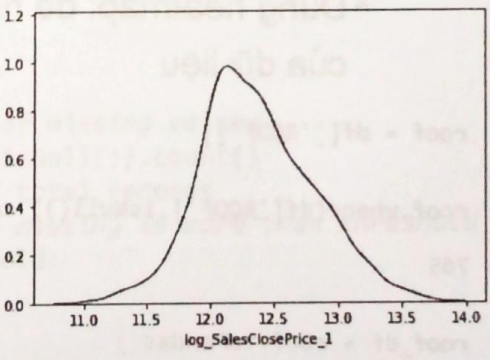
## Data Cleaning

### ❑Log Scaling

- Áp dụng khi có right skew data




Chưa chuẩn với log




Đã chuẩn với log

```
df_sub = df_sub.withColumn('log_SalesClosePrice_1', log(df['SALESCLOSEPRICE']))
```

## ❑Dữ liệu bị thiếu (Missing Data)

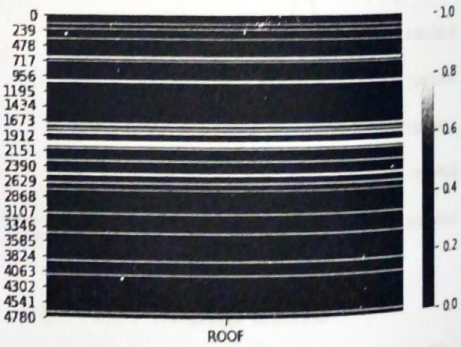
- Vì sao dữ liệu bị thiếu?
  - Quá trình thu thập dữ liệu (Data Collection) có sự cố. Ví dụ: Sensor bị hỏng
  - Vi phạm quy tắc lưu trữ dữ liệu (Data Storage Rule). Ví dụ: 2020-01-10 vs. January 10st, 2020
  - Khi kết từ nhiều nguồn dữ liệu khác nhau
  - Thiếu có chủ ý (Intentionally Missing). Ví dụ: Dữ liệu cá nhân





- Hiển thị dữ liệu thiếu
  - Dùng isNull(): để kiểm tra dữ liệu thiếu
  - Dùng heatmap: để hiển thị số lượng dữ liệu thiếu của dữ liệu

```
roof = df['ROOF']

roof.where(df['ROOF'].isNull()).count()

765

roof_df = roof.toPandas()

sns.heatmap(data=roof_df.isnull())
```



## Data Cleaning

- Xóa dữ liệu bị thiếu khi:
  - Giá trị thiếu rất ít
  - Thiếu hoàn toàn ngẫu nhiên (Missing Completely at Random)



## Data Cleaning

- Xóa dữ liệu theo tỷ lệ dữ liệu thiếu

```
def column_dropper(df, threshold):
    # Takes a dataframe and threshold for missing values.
    # Returns a dataframe.
    total_records = df.count()
    for col in df.columns:
        # Calculate the percentage of missing values
        missing = df.where(df[col].isNull()).count()
        missing_percent = missing / total_records
        # Drop column if percent of missing is more than threshold
        if missing_percent > threshold:
            df = df.drop(col)
    return df

# Drop columns that are more than 70% missing
df = column_dropper(df, 0.7)
```

## Data Cleaning


- Điền dữ liệu khi bị thiếu
  - Rule Based: giá trị điền dựa trên business logic
  - Statistics Based: giá trị điền có thể là mean, median...
  - Model Based: giá trị điền là giá trị dự đoán từ model (sử dụng model để dự đoán dữ liệu)



## Data Cleaning


- Điền dữ liệu khi bị thiếu
  - Sử dụng: fillna(value, subset=None)
    - value là giá trị thay thế cho giá trị thiếu
    - Subset: là danh sách tên các cột mà giá trị thiếu sẽ được thay thế
  - Ví dụ:

```
# Count missing rows
missing = df.where(df['PDOM'].isNull()).count()
# Calculate the mean value
col_mean = df.agg({'PDOM': 'mean'}).collect()[0][0]

# Replacing with the mean value for that column
df.fillna(col_mean, subset=['PDOM'])
```

## Nội dung

1. Data Cleaning
2. Feature Engineering
3. Một số kỹ thuật khác



## Feature Engineering

- **Có thể tạo ra tính năng mới bằng cách:**
  - Tổng các thuộc tính (Sum)
  - Hiệu các thuộc tính (Difference)
  - Tích các thuộc tính (Multiplication)
  - Thương các thuộc tính (Divide)
  - Tính toán tỷ lệ (Ratio)
  - Dùng regular expression extract (pyspark.sql .functions.regexp_extract)

# Feature Engineering


- Ví dụ:

```
from pyspark.sql.functions import datediff

# Sum two columns
df = df.withColumn('TSQFT', (df['SQFTBELOWGROUND'] + df['SQFTABOVEGROUND']))
# Divide two columns
df = df.withColumn('PRICEPERTSQFT', (df['LISTPRICE'] / df['TSQFT']))
# Difference two columns
df = df.withColumn('DAYSONMARKET', datediff('OFFMARKETDATE', 'LISTDATE'))
```



# Feature Engineering


- Ví dụ:

```
# Lot size in square feet
acres_to_sqfeet = 43560
df = df.withColumn('LOT_SIZE_SQFT', df['ACRES'] * acres_to_sqfeet)

# Create new column YARD_SIZE
df = df.withColumn('YARD_SIZE', df['LOT_SIZE_SQFT'] - df['FOUNDATIONSIZE'])

# Corr of ACRES vs SalesClosePrice
print("Corr of ACRES vs SalesClosePrice: " + str(df.corr('ACRES', 'SalesClosePrice')))
# Corr of FOUNDATIONSIZE vs SALESCLOSEPRICE
print("Corr of FOUNDATIONSIZE vs SalesClosePrice: " + str(df.corr('FOUNDATIONSIZE', 'SalesClosePrice')))
# Corr of YARD_SIZE vs SALESCLOSEPRICE
print("Corr of YARD_SIZE vs SalesClosePrice: " + str(df.corr('YARD_SIZE', 'SalesClosePrice')))
```

```
Corr of ACRES vs SalesClosePrice: 0.22060612588935327
Corr of FOUNDATIONSIZE vs SalesClosePrice: 0.6152231695664401
Corr of YARD_SIZE vs SalesClosePrice: 0.20714585430854263
```

## Feature Engineering

- Ví dụ:

```
# Ratio: BED_TO_BATHS
df = df.withColumn('BED_TO_BATHS', df['BEDROOMS'] / df['BATHSTOTAL'])
df[['BED_TO_BATHS', 'BEDROOMS', 'BATHSTOTAL']].show(5)
```

```
+------------------+--------+----------+
|      BED_TO_BATHS|BEDROOMS|BATHSTOTAL|
+------------------+--------+----------+
|               1.5|       3|         2|
|1.3333333333333333|       4|         3|
|               2.0|       2|         1|
|               1.0|       2|         2|
|               1.5|       3|         2|
+------------------+--------+----------+
only showing top 5 rows
```



## Feature Engineering

- regexp_extract(column_name, regex, group_number)
  - Ví dụ:

```
df = df.withColumn('Employee', regexp_extract(col('Notes'), '(.)(by)(\s+)(\w+)', 4))
df.show()
```

```
+----+--------------------------------------+
|ID  |Notes                                 |
+----+--------------------------------------+
|2345|Checked by John                       |
|2398|Verified by Stacy                     |
|2328|Verified by Srinivas than some random text|
|3983|Double Checked on 2/23/17 by Marsha   |
+----+--------------------------------------+
```

→

```
+----+--------------------+--------+
|  ID|               Notes|Employee|
+----+--------------------+--------+
|2345|    Checked by John|    John|
|2398|   Verified by Stacy|   Stacy|
|2328|Verified by Srini...|Srinivas|
|3983|Double Checked on...|  Marsha|
+----+--------------------+--------+
```

## ❑Time Feature

- Các thuộc tính thời gian có thể được sử dụng để tạo ra các tính năng qua trong cũng như giúp chúng ta khám phá, tìm hiểu về dữ liệu.
  - Ví dụ: Tìm hiểu xem có mẫu nào được liệt kê vào ngày nào đó trong tuần không.
  - Chú ý: Tuần của PySpark bắt đầu vào Chủ nhật, với giá trị là 1 và kết thúc vào Thứ Bảy, giá trị là 7.





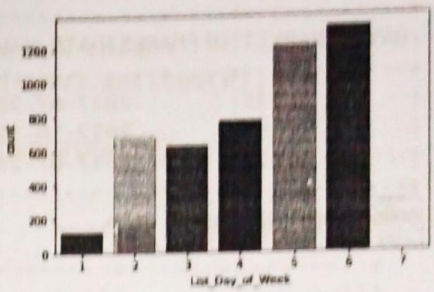
- Tạo ra các thành phần từ Time Feature
  - Ví dụ

```
from pyspark.sql.functions import to_date, dayofweek, to_timestamp
from pyspark.sql import types
from pyspark.sql.functions import col, udf
from datetime import datetime
from pyspark.sql.types import DateType

# Convert to date type
func =  udf (lambda x: datetime.strptime(x, '%m/%d/%Y %H:%M'), DateType())
df = df.withColumn('LISTDATE_new', func(col('LISTDATE')))
# df = df.withColumn('LISTDATE_new', to_date('LISTDATE'))

# Get the day of the week
df = df.withColumn('List_Day_of_Week', dayofweek('LISTDATE_new'))

# Sample and convert to pandas dataframe
sample_df = df.sample(False, 0.5, 42).toPandas()
```

# Feature Engineering

```
sample_df[['LISTDATE', 'LISTDATE_new', 'List_Day_of_Week']].head(5)
```

| | LISTDATE | LISTDATE_new | List_Day_of_Week |
|---|---|---|---|
| 0 | 7/15/2017 0:00 | 2017-07-15 | 7 |
| 1 | 10/9/2017 0:00 | 2017-10-09 | 2 |
| 2 | 6/26/2017 0:00 | 2017-06-26 | 2 |
| 3 | 8/25/2017 0:00 | 2017-08-25 | 6 |
| 4 | 9/12/2017 0:00 | 2017-09-12 | 3 |




```
# Plot count plot of of day of week
ax = sns.countplot(x="List_Day_of_Week", data=sample_df)
plt.show()
```



# Feature Engineering

```
from pyspark.sql.functions import year, month
from pyspark.sql.functions import dayofmonth, weekofyear
# Create a new column of year number
df = df.withColumn('LIST_YEAR', year('LISTDATE_new'))

# Create a new column of month number
df = df.withColumn('LIST_MONTH', month('LISTDATE_new'))

# Create new columns of the day number within the month
df = df.withColumn('LIST_DAYOFMONTH', dayofmonth('LISTDATE_new'))

# Create new columns of the week number within the year
df = df.withColumn('LIST_WEEKOFYEAR', weekofyear('LISTDATE_new'))

df[['LISTDATE_new', 'LIST_YEAR', 'LIST_MONTH', 'LIST_DAYOFMONTH', 'LIST_WEEKOFYEAR']].show(3)
```

```
+------------+---------+----------+---------------+---------------+
|LISTDATE_new|LIST_YEAR|LIST_MONTH|LIST_DAYOFMONTH|LIST_WEEKOFYEAR|
+------------+---------+----------+---------------+---------------+
|  2017-07-15|     2017|         7|             15|             28|
|  2017-10-09|     2017|        10|              9|             41|
|  2017-06-26|     2017|         6|             26|             26|
+------------+---------+----------+---------------+---------------+
only showing top 3 rows
```

## Feature Engineering

```
df = df.withColumn('DAYSONMARKET', datediff(to_date(df['OFFMARKETDATE_new']),
                                            to_date(df['LISTDATE_new'])))
df['DAYSONMARKET','OFFMARKETDATE_new', 'LISTDATE_new' ].show(3)
```

```
+------------+-----------------+------------+
|DAYSONMARKET|OFFMARKETDATE_new|LISTDATE_new|
+------------+-----------------+------------+
|          15|       2017-07-30|  2017-07-15|
|           4|       2017-10-13|  2017-10-09|
|          28|       2017-07-24|  2017-06-26|
+------------+-----------------+------------+
only showing top 3 rows
```



## Feature Engineering

### ❑ Trích xuất tính năng (Extracting Feature)

- Trích xuất text thành feature mới

```
# Import needed functions
from pyspark.sql.functions import when
# Create boolean conditions for string matches
has_attached_garage = df['GARAGEDESCRIPTION'].like('%Attached Garage%')
has_detached_garage = df['GARAGEDESCRIPTION'].like('%Detached Garage%')
# Conditional value assignment
df = df.withColumn('has_attached_garage', (when(has_attached_garage, 1)
                                           .when(has_detached_garage, 0)
                                           .otherwise(None)))

# Inspect results
df[['GARAGEDESCRIPTION', 'has_attached_garage']].show(truncate=50, n=3)
```

```
+--------------------------------------------------+------------------+
|                                GARAGEDESCRIPTION|has_attached_garage|
+--------------------------------------------------+------------------+
|                                  Attached Garage|                 1|
|Attached Garage, Driveway - Asphalt, Garage Doo...|                 1|
|                                  Attached Garage|                 1|
+--------------------------------------------------+------------------+
only showing top 3 rows
```

# Feature Engineering

- Cắt chuỗi (splitting) thành feature mới

```
from pyspark.sql.functions import split
# Split the column on commas into a list
split_col = split(df['ROOF'], ',')
# Put the first value of the list into a new column
df = df.withColumn('Roof_Material', split_col.getItem(0))
# Inspect results
df[['ROOF', 'Roof_Material']].show(5, truncate=100)
```

```
+----------------------------------------------+----------------+
|                                          ROOF|   Roof_Material|
+----------------------------------------------+----------------+
|                                          null|            null|
|Asphalt Shingles, Pitched, Age 8 Years or Less|Asphalt Shingles|
|                                          null|            null|
|Asphalt Shingles, Pitched, Age 8 Years or Less|Asphalt Shingles|
|             Asphalt Shingles, Age Over 8 Years|Asphalt Shingles|
+----------------------------------------------+----------------+
only showing top 5 rows
```



# Feature Engineering

- Tạo nhiều thuộc tính mới (exploding) cho record

```
# Import needed functions
from pyspark.sql.functions import split, explode
# Convert string to list-like array
df = df.withColumn('roof_list', split(df['ROOF'], ', '))
df[["No", "roof_list"]].show(4, truncate=50)

# Explode the values into new records
roof_df = df.withColumn('ex_roof_list', explode(df['roof_list']))

# Inspect the values
roof_df[["No", "ex_roof_list"]].show(n=5, truncate=50)
```

```
+---+--------------------------------------------------+
| No|                                         roof_list|
+---+--------------------------------------------------+
|  1|                                            null|
|  2|[Asphalt Shingles, Pitched, Age 8 Years or Less]|
|  3|                                            null|
|  4|[Asphalt Shingles, Pitched, Age 8 Years or Less]|
+---+--------------------------------------------------+
only showing top 4 rows

+---+--------------------+
| No|       ex_roof_list|
+---+--------------------+
|  2|   Asphalt Shingles|
|  2|            Pitched|
|  2|Age 8 Years or Less|
|  4|   Asphalt Shingles|
|  4|            Pitched|
+---+--------------------+
only showing top 5 rows
```

# Feature Engineering

- Pivot

```
from pyspark.sql.functions import split, explode, lit, coalesce, first

# Create a dummy column of constant value
roof_df = roof_df.withColumn('constant_val', lit(1))

# Pivot the values into boolean columns
roof_piv_df = roof_df.groupBy('NO').pivot('ex_roof_list')\
                     .agg(coalesce(first('constant_val')))
```

Exploded Record

| NO | ex_roof_list |
|---|---|
| 2 | Asphalt Shingles |
| 2 | Pitched |
| 2 | Age 8 Years or Less |

Pivoted Record

| NO | Age 8 Years or Less | Age Over 8 Years | Asphalt Shingles | Flat | Metal | Other | Pitched | ... |
|---|---|---|---|---|---|---|---|---|
| 2 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | ... |



# Feature Engineering

- Join

```
# Join the dataframes together and fill null
joined_data = df.join(roof_piv_df, on='No', how='left')

# Columns to zero fill
zfill_cols = roof_piv_df.columns

# Zero fill the pivoted values
zfilled_df = joined_data.fillna(0, subset=zfill_cols)
```

## ❑ Binarizing, Bucketing & Encoding

- Binarizing: Là kỹ thuật tạo ra một feature mới từ một feature đang có nhưng dưới dạng nhị phân (0 hoặc 1) theo một ngưỡng cho trước (mặc định: threshold = 0)





- Ví dụ:

```
from pyspark.ml.feature import Binarizer
# Cast the data type to double
df = df.withColumn('FIREPLACES', df['FIREPLACES'].cast('double'))
# Create binarizing transformer
bin = Binarizer(threshold=0.0, inputCol='FIREPLACES', outputCol='FireplaceB')
# Apply the transformer
df = bin.transform(df)

# Inspect the results
df[['FIREPLACES','FireplaceB']].show(6)
```

```
+----------+----------+
|FIREPLACES|FireplaceB|
+----------+----------+
|       0.0|       0.0|
|       0.0|       0.0|
|       0.0|       0.0|
|       1.0|       1.0|
|       1.0|       1.0|
|       1.0|       1.0|
+----------+----------+
only showing top 6 rows
```

## Feature Engineering

- Bucketing: Là kỹ thuật tối ưu hóa phân tách dữ liệu thành các phần để dễ quản lý hơn (buckets), để phân vùng dữ liệu (data partitioning). Động lực là để tối ưu hóa hiệu suất của truy vấn join query bằng cách tránh xáo trộn (avoiding shuffles) các bảng trong join. Các kết quả của Bucketing ít trao đổi hơn, vì việc xáo trộn có thể không cần thiết - cả hai DataFrame có thể đã được đặt trong cùng một phân vùng (partition).


https://luminousmen.com/post/the-5-minute-guide-to-using-bucketing-in-pyspark



## Feature Engineering

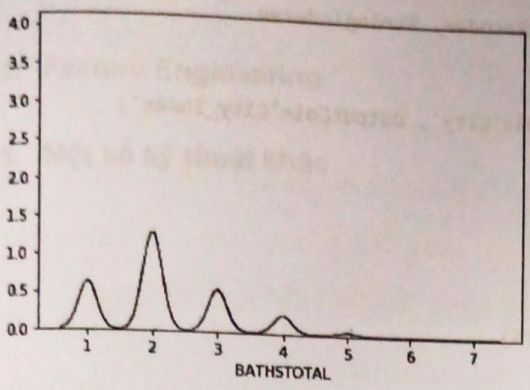
- Ví dụ:

```
from pyspark.ml.feature import Bucketizer
# Define how to split data
splits = [0, 1, 2, 3, 4, float('Inf')]
# Create bucketing transformer
buck = Bucketizer(splits=splits, inputCol='BATHSTOTAL', outputCol='baths')
# Apply transformer
df = buck.transform(df)

# Inspect results
df[['BATHSTOTAL', 'baths']].show(5)
```

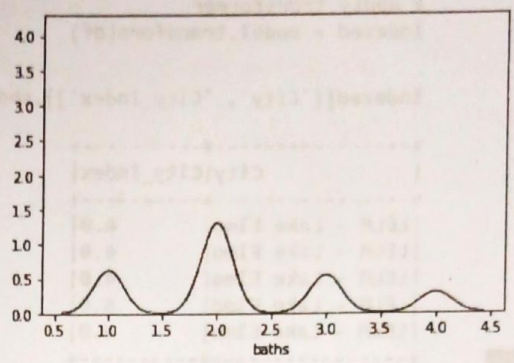
```
+----------+-----+
|BATHSTOTAL|baths|
+----------+-----+
|         2|  2.0|
|         3|  3.0|
|         1|  1.0|
|         2|  2.0|
|         2|  2.0|
+----------+-----+
only showing top 5 rows
```

# Feature Engineering

```
sns.distplot(sample_df['BATHSTOTAL'])
plt.show()
```




```
sns.distplot(sample_df['baths'])
plt.show()
```






# Feature Engineering

- One hot encoding: là kỹ thuật mã hóa các thuộc tính phân loại dưới dạng một one-hot numeric array.

| Original data: | | One hot encoding format: | | | | | |
|---|---|---|---|---|---|---|---|
| id | Color | id | White | Red | Black | Purple | Gold |
| 1 | White | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | Red | 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | Black | 3 | 0 | 0 | 1 | 0 | 0 |
| 4 | Purple | 4 | 0 | 0 | 0 | 1 | 0 |
| 5 | Gold | 5 | 0 | 0 | 0 | 0 | 1 |

# Feature Engineering

- Ví dụ:

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer

# Create indexer transformer
stringIndexer = StringIndexer(inputCol='City', outputCol='City_Index')

# Fit transformer
model = stringIndexer.fit(df)
# Apply transformer
indexed = model.transform(df)

indexed[['City', 'City_Index']].show(5)
```

```
+----------------+----------+
|            City|City_Index|
+----------------+----------+
|LELM - Lake Elmo|       4.0|
|LELM - Lake Elmo|       4.0|
|LELM - Lake Elmo|       4.0|
|LELM - Lake Elmo|       4.0|
|LELM - Lake Elmo|       4.0|
+----------------+----------+
only showing top 5 rows
```



# Feature Engineering

- Ví dụ:

```
# Create encoder transformer
encoder = OneHotEncoder(inputCol='City_Index', outputCol='City_Vec')

# Apply the encoder transformer
encoded_df = encoder.transform(indexed)

# Inspect results
encoded_df[['City_Index', 'City_Vec']].show(5)
```

```
+----------+---------+
|City_Index| City_Vec|
+----------+---------+
|       4.0|(4,[],[])|
|       4.0|(4,[],[])|
|       4.0|(4,[],[])|
|       4.0|(4,[],[])|
|       4.0|(4,[],[])|
+----------+---------+
only showing top 5 rows
```

## Nội dung

1. Data Cleaning
2. Feature Engineering
3. Một số kỹ thuật khác



## Một số kỹ thuật khác

### ❑Caching

- Lưu trữ các DataFrame trong memory hoặc trên disk
- Cải thiện tốc độ trên các transformation/ action
- Giảm tài nguyên sử dụng

## Một số kỹ thuật khác


- Khi làm việc với các task trong PySpark
  - Chỉ cache khi cần
  - Hãy thử caching các DataFrame tại nhiều điểm khác nhau và xác định xem hiệu suất có cải thiện không
  - Cache trong memory và fast SSD/ NVMe storage
  - Cache để làm chậm local disk nếu cần
  - Sử dụng trung gian



## Một số kỹ thuật khác


- Tuy nhiên, caching sẽ không phù hợp khi:
  - Dataset quá lớn sẽ không vừa với memory
  - Local disk dựa trên caching có thể không phải là cách cải tiến hiệu suất
  - Các đối tượng lưu trữ có thể không có sẵn

## Một số kỹ thuật khác

- Làm việc với cache
  - Sử dụng <DataFrame_name>.cache() trên DataFrame trước Action
    - Ví dụ:

```
df = spark.read.csv(["AA_data"], header=True, inferSchema=True)

df.cache().count()

583718
```



## Một số kỹ thuật khác

- Sử dụng <DataFrame_name>.is_cached để xác định trạng thái của DataFrame
- Sử dụng <DataFrame_name>.unpersist() khi bỏ DataFrame ra khỏi cache

```
# Determine if df_new is in the cache
print("Is df_new cached?: %s" % df_new.is_cached)
print("Removing df_new from cache")

# Remove df_new from the cache
df_new.unpersist()

# Check the cache status again
print("Is df_new cached?: %s" % df_new.is_cached)

Is df_new cached?: True
Removing df_new from cache
Is df_new cached?: False
```

## Một số kỹ thuật khác

### ❑Parquet

- Là một loại định dạng dữ liệu
- Dữ liệu được cung cấp dưới dạng parquet
  - Column-wise được lưu trữ: Truy vấn một bộ các column nhanh
  - Schema được khai báo có cấu trúc
    - Field và DataType được khai báo
    - Phù hợp với messy text data

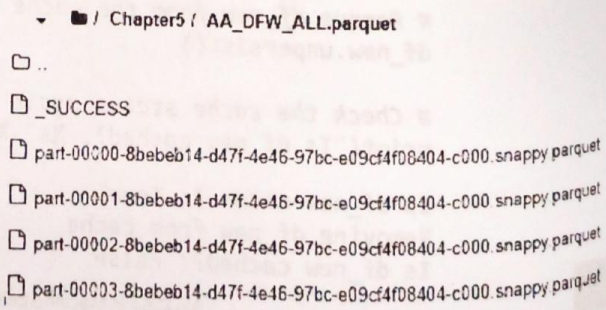


## Một số kỹ thuật khác

- Làm việc với parquet

```
# Save the df3 DataFrame in Parquet format
df.write.parquet('AA_DFW_ALL.parquet', mode='overwrite')

# Read the Parquet file into a new DataFrame
df_new = spark.read.parquet('AA_DFW_ALL.parquet')

print(df_new.count())
```

```
583718
```




# Chapter 5: Data Pre-processing

## Ex1: DallasCouncilVoters

### Cho dữ liệu DallasCouncilVoters.csv

### Yêu cầu:

1. Đọc dữ liệu => df
2. Cho biết dữ liệu có bao nhiêu dòng, in scheme. Hiển thị 5 dòng dữ liệu đầu tiên.
3. Kiểm tra dữ liệu NaN, null. Nếu dòng nào 'VOTER_NAME' có dữ liệu null thì xóa hết các dòng đó.
4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.
5. Tìm các VOTER_NAME duy nhất và hiển thị 10 thông tin đầu tiên.
6. Lọc dữ liệu theo điều kiện 'VOTER_NAME' có chiều dài từ 1-20 ký tự.
7. Loại bỏ các dữ liệu mà trong 'VOTER_NAME' có chứa dấu '_' (underscore)
8. Tạo cột 'splits' chứa thông tin được cắt theo khoảng trắng từ 'VOTER_NAME'
9. Tạo cột 'first_name' lấy dữ liệu từ phần tử đầu tiên trong cột 'splits'
10. Tạo cột 'last_name' lấy dữ liệu từ phần tử cuối cùng trong cột 'splits'
11. Tạo cột 'random_val' theo điều kiện: nếu cột 'TITLE' có nội dung là 'Councilmember' thì 'random_val' sẽ có giá trị rand(), nếu có nội dung là 'Mayor' thì 'random_val' sẽ có giá trị là 2, ngược lại sẽ có giá trị là 0.
12. Lọc các dòng dữ liệu có 'random_val' = 0. Hiển thị.
13. Xây dựng function: getFirstAndMiddle(names) trả về kết quả gồm First và Middle (names). Khai báo function vừa viết dưới dạng udf đặt tên là udfFirstAndMiddle.
14. Tạo cột first_and_middle_name bằng cách gọi udf trên với tham số truyền vào là cột 'splits'. In kết quả.
15. Xóa bỏ các cột 'first_name', 'splits'. In kết quả.
16. Thêm cột 'ROW_ID' bằng phương thức: monotonically_increasing_id() (trong pyspark.sql.functions).
17. Hiển thị 10 dòng đầu của dữ liệu với ROW_ID tăng dần.

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark import SparkContext
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
```

In [3]:
```
sc =SparkContext()
```

In [4]:
```
spark = SparkSession(sc)
```

In [5]:
```
#1.
df = spark.read.csv('voters_data/DallasCouncilVoters.csv', header=True,
                    inferSchema=True)
```

In [6]:
```
#2.
df.count()
```

Out[6]: 44625

In [7]:
```
df.printSchema()
```

```
root
 |-- DATE: string (nullable = true)
 |-- TITLE: string (nullable = true)
 |-- VOTER_NAME: string (nullable = true)
```

In [8]:
```
df.show(5)
```

```
+----------+-------------+-------------------+
|      DATE|        TITLE|         VOTER_NAME|
+----------+-------------+-------------------+
|02/08/2017|Councilmember|  Jennifer S. Gates|
|02/08/2017|Councilmember| Philip T. Kingston|
|02/08/2017|        Mayor|Michael S. Rawlings|
|02/08/2017|Councilmember|       Adam Medrano|
|02/08/2017|Councilmember|       Casey Thomas|
+----------+-------------+-------------------+
only showing top 5 rows
```

In [9]:
```
from pyspark.sql.functions import col, udf
from pyspark.sql.functions import isnan, when, count, col
```

In [10]:
```
#3. Kiểm tra dữ liệu NaN, null
df.select([count(when(isnan(c), c)).alias(c) for c in df.columns]).toPandas().T
```

Out[10]:

| | 0 |
|---|---|
| DATE | 0 |
| TITLE | 0 |
| VOTER_NAME | 0 |

In [11]:
```
# => Không có dữ liệu NaN
```

In [12]:
```
df.select([count(when(col(c).isNull(), c)).alias(c) for c in
           df.columns]).toPandas().T
```

Out[12]:

| | 0 |
|---|---|
| DATE | 0 |
| TITLE | 195 |
| VOTER_NAME | 503 |

In [13]:
```
# => Có dữ liệu null. Xóa dữ liệu có VOTER_NAME null
```

In [14]:
```
df = df.dropna(subset='VOTER_NAME')
```

In [15]:
```
df.select([count(when(col(c).isNull(), c)).alias(c) for c in
           df.columns]).toPandas().T
```

Out[15]:

| | 0 |
|---|---|
| DATE | 0 |
| TITLE | 0 |
| VOTER_NAME | 0 |

In [16]:
```
# => hết dữ liệu null
```

In [17]:
```
#4.
num_rows = df.count()
num_dist_rows = df.distinct().count()
dup_rows = num_rows - num_dist_rows
```

In [18]:
```
display(num_rows, num_dist_rows, dup_rows)
```

```
44122

1273

42849
```

In [19]:
```
# Check duplicate
df.filter(df['VOTER_NAME'] == 'Philip T. Kingston').show(5)
```

```
+----------+-------------+-----------------+
|      DATE|        TITLE|       VOTER_NAME|
+----------+-------------+-----------------+
|02/08/2017|Councilmember|Philip T. Kingston|
|02/08/2017|Councilmember|Philip T. Kingston|
|01/11/2017|Councilmember|Philip T. Kingston|
|09/14/2016|Councilmember|Philip T. Kingston|
|01/04/2017|Councilmember|Philip T. Kingston|
+----------+-------------+-----------------+
only showing top 5 rows
```

In [20]:
```
df = df.drop_duplicates()
```

In [21]:
```
df.count()
```

Out[21]: 1273

In [22]:
```
#5. Show the distinct VOTER_NAME entries
df.select(df['VOTER_NAME']).distinct().show(10)
```

```
+--------------------+
|          VOTER_NAME|
+--------------------+
|      Tennell Atkins|
|  the  final   20...|
|       Scott Griggs|
|      Scott  Griggs|
|      Sandy Greyson|
| Michael S. Rawlings|
| the final 2018 A...|
|       Kevin Felder|
|       Adam Medrano|
|      Casey  Thomas|
+--------------------+
only showing top 10 rows
```

In [23]:
```
from pyspark.sql.functions import *
```

In [24]:
```
#6. Filter df where the VOTER_NAME is 1-20 characters in length
df = df.filter('length(VOTER_NAME) > 0 and length(VOTER_NAME) < 20')
```

In [25]:
```
df.show(5)
```

```
+----------+--------------------+--------------------+
|      DATE|               TITLE|          VOTER_NAME|
+----------+--------------------+--------------------+
|04/11/2018|Deputy Mayor Pro Tem|                null|
|02/14/2018|       Councilmember|       Adam Medrano|
|04/25/2018|       Councilmember|   Lee M. Kleinman|
|08/29/2018|       Councilmember|     Tennell Atkins|
|10/18/2017|       Councilmember|       Kevin Felder|
+----------+--------------------+--------------------+
only showing top 5 rows
```

In [26]:
```
#7. Filter out df where the VOTER_NAME contains an underscore
df = df.filter(~ col('VOTER_NAME').contains('_'))
```

In [27]:
```
# Show the distinct VOTER_NAME entries again
df.select('VOTER_NAME').distinct().show(10, truncate=False)
```

```
+------------------+
|VOTER_NAME        |
+------------------+
|Tennell Atkins    |
|Scott Griggs      |
|Scott  Griggs     |
|Sandy Greyson     |
|Michael S. Rawlings|
|Kevin Felder      |
|Adam Medrano      |
|Casey  Thomas     |
|Mark  Clayton     |
|Casey Thomas      |
+------------------+
only showing top 10 rows
```

## Modifying DataFrame

In [28]:
```
#8. Add a new column called splits separated on whitespace
df = df.withColumn('splits', split(df.VOTER_NAME, '\s+'))
```

In [29]:
```
#9. Create a new column called first_name based on the first item in splits
df = df.withColumn('first_name', df.splits.getItem(0))
```

In [30]:
```
#10. Get the last entry of the splits list and create a column called last_name
df = df.withColumn('last_name', df.splits.getItem(size('splits') - 1))
```

In [31]:
```
# Drop the splits column
# df = df.drop('splits')
```

In [32]:
```
# Show the voter_df DataFrame
df.show(3)
```

```
+----------+--------------------+--------------+--------------------+----------
+---------+
|      DATE|               TITLE|    VOTER_NAME|              splits|first_name
|last_name|
+----------+--------------------+--------------+--------------------+----------
+---------+
|04/11/2018|Deputy Mayor Pro Tem|  Adam Medrano|    [Adam, Medrano]|      Adam
|  Medrano|
|02/14/2018|       Councilmember|Lee M. Kleinman|[Lee, M., Kleinman]|       Lee
| Kleinman|
|04/25/2018|       Councilmember| Tennell Atkins|  [Tennell, Atkins]|   Tennell
|   Atkins|
+----------+--------------------+--------------+--------------------+----------
+---------+
only showing top 3 rows
```

In [33]:
```
#11. Add a column to df for any voter with the title 'Councilmember'
df = df.withColumn('random_val', when(df.TITLE == 'Councilmember', rand()))
```

In [34]:
```
# Show some of the DataFrame rows, noting whether the when clause worked
#df.show(5)
```

In [35]:
```
#. Add a column to df for a voter based on their position
df = df.withColumn('random_val',
                                  when(df.TITLE == 'Councilmember', rand())
                                  .when(df.TITLE == 'Mayor', 2)
                                  .otherwise(0))
```

In [36]:
```
# Show some of the DataFrame rows
df.show(5)
```

```
+----------+--------------------+----------------+--------------------+------
----+---------+--------------------+
|      DATE|               TITLE|      VOTER_NAME|              splits|first_
name|last_name|          random_val|
+----------+--------------------+----------------+--------------------+------
----+---------+--------------------+
|04/11/2018|Deputy Mayor Pro Tem|    Adam Medrano|    [Adam, Medrano]|
Adam|  Medrano|                 0.0|
|02/14/2018|        Councilmember|  Lee M. Kleinman|[Lee, M., Kleinman]|
Lee| Kleinman|  0.7266891908590055|
|04/25/2018|        Councilmember|   Tennell Atkins|  [Tennell, Atkins]|   Ten
nell|   Atkins|1.716281340619074...|
|08/29/2018|        Councilmember|     Kevin Felder|    [Kevin, Felder]|     K
evin|   Felder|0.047122114981064556|
|10/18/2017|        Councilmember|Jennifer S.  Gates|[Jennifer, S., Ga...|  Jenn
ifer|    Gates| 0.47584042942379867|
+----------+--------------------+----------------+--------------------+------
----+---------+--------------------+
only showing top 5 rows
```

In [37]:
```
#12. Use the .filter() clause with random_val
df.filter(df.random_val == 0).show(5)
```

```
+----------+--------------------+----------------+--------------------+-------
---+---------+----------+
|      DATE|               TITLE|      VOTER_NAME|              splits|first_n
ame|last_name|random_val|
+----------+--------------------+----------------+--------------------+-------
---+---------+----------+
|04/11/2018|Deputy Mayor Pro Tem|    Adam Medrano|    [Adam, Medrano]|      A
dam|  Medrano|       0.0|
|04/12/2017|       Mayor Pro Tem| Monica R. Alonzo|[Monica, R., Alonzo]|    Mon
ica|   Alonzo|       0.0|
|06/28/2017|Deputy Mayor Pro Tem|    Adam Medrano|    [Adam, Medrano]|      A
dam|  Medrano|       0.0|
|01/03/2018|Deputy Mayor Pro Tem|    Adam Medrano|    [Adam, Medrano]|      A
dam|  Medrano|       0.0|
|01/17/2018|       Mayor Pro Tem|Dwaine R. Caraway|[Dwaine, R., Cara...|    Dwa
ine|  Caraway|       0.0|
+----------+--------------------+----------------+--------------------+-------
---+---------+----------+
only showing top 5 rows
```

## UDF

In [38]:
```
from pyspark.sql.types import *
```

In [39]:
```
def getFirstAndMiddle(names):
    # Return a space separated string of names
    return ' '.join(names[:-1])
```

In [40]:
```
#13. Define the method as a UDF
udfFirstAndMiddle = udf(getFirstAndMiddle, StringType())
```

In [41]:
```
#14. Create a new column using your UDF
df = df.withColumn('first_and_middle_name', udfFirstAndMiddle(df.splits))
```

In [42]:
```
#15. Drop the unnecessary columns then show the DataFrame
df = df.drop('first_name')
df = df.drop('splits')
```

In [43]:
```
df.show(5)
```

```
+----------+--------------------+--------------------+---------+--------------------
---+--------------------+
|      DATE|               TITLE|         VOTER_NAME|last_name|                random_
val|first_and_middle_name|
+----------+--------------------+--------------------+---------+--------------------
---+--------------------+
|04/11/2018|Deputy Mayor Pro Tem|      Adam Medrano|  Medrano|
0.0|                Adam|
|02/14/2018|       Councilmember|   Lee M. Kleinman| Kleinman|  0.726689190859
055|               Lee M.|
|04/25/2018|       Councilmember|    Tennell Atkins|   Atkins|1.71628134061967
4...|             Tennell|
|08/29/2018|       Councilmember|      Kevin Felder|   Felder|0.047122114981864
556|               Kevin|
|10/18/2017|       Councilmember|Jennifer S.  Gates|    Gates| 0.4758404294237
867|          Jennifer S.|
+----------+--------------------+--------------------+---------+--------------------
---+--------------------+
only showing top 5 rows
```

## Adding an ID Field

In [44]:
```
# Select all the unique council voters
df = df.select(df["VOTER_NAME"]).distinct()

# Count the rows in voter_df
print("\nThere are %d rows in the df DataFrame.\n" % df.count())
```

```
There are 27 rows in the df DataFrame.
```

In [45]:
```
#16. Add a ROW_ID
df = df.withColumn('ROW_ID', monotonically_increasing_id())
```

In [46]:
```
#17. Show the rows with 10 highest IDs in the set
df.orderBy(df.ROW_ID.desc()).show(10)
```

```
+--------------------+-------------+
|          VOTER_NAME|       ROW_ID|
+--------------------+-------------+
|       Lee Kleinman|1709396983808|
|         Erik Wilson|1700807049216|
|Carolyn King Arnold|1632087572480|
|Rickey D.  Callahan|1597727834112|
|   Monica R. Alonzo|1382979469312|
|    Lee M. Kleinman|1228360646656|
|  Jennifer S. Gates|1194000908288|
|Philip T.  Kingston|1185410973696|
|  Dwaine R. Caraway|1142461300736|
| Rickey D. Callahan|1125281431553|
+--------------------+-------------+
only showing top 10 rows
```

## IDs with different partitions

In [47]:
```
# Mở rộng
```

In [48]:
```
# Print the number of partitions in each DataFrame
print("\nThere are %d partitions in the df DataFrame.\n" % df.rdd.getNumPartitio
```

```
There are 200 partitions in the df DataFrame.
```

- Make sure to store the result of .rdd.max()[0] in the variable.
- monotonically_increasing_id() returns an integer. You can modify that value in-line.
- Make sure to show both Data Frames.

In [49]:
```
# Determine the highest ROW_ID and save it in previous_max_ID
previous_max_ID = df.select('ROW_ID').rdd.max()[0]

# Add a ROW_ID column to df_april starting at the desired value
voter_df_april = df.withColumn('ROW_ID',
                               monotonically_increasing_id() + previous_max_ID)
```

In [50]:
```
# Show the ROW_ID from both DataFrames and compare
df.select('ROW_ID').show(5)
voter_df_april.select('ROW_ID').show(5)
```

```
+------------+
|      ROW_ID|
+------------+
|  8589934592|
| 34359738368|
| 42949672960|
| 51539607552|
|103079215104|
+------------+
only showing top 5 rows

+-------------+
|       ROW_ID|
+-------------+
|1717986918400|
|1743756722176|
|1752346656768|
|1760936591360|
|1812476198912|
+-------------+
only showing top 5 rows
```

# Chapter 5: Data Pre-processing

## Ex2: AA data

### Cho dữ liệu về thông tin các chuyến bay trong thư mục "AA_data"

### Yêu cầu:

1. Đọc dữ liệu => df
2. Cho biết dữ liệu có bao nhiêu dòng, in scheme. Hiển thị 5 dòng dữ liệu đầu tiên.
3. Kiểm tra dữ liệu NaN, null
4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.
5. Trong df, thêm cột 'airport' lấy dữ liệu từ cột 'Destination Airport', định dạng chữ thường cho nội dung
6. Trong df, thêm cột 'date' lấy dữ liệu từ cột 'Date (MM/DD/YYYY)', sau đó xóa bỏ cột 'Date (MM/DD/YYYY)'
7. Trong df, đổi tên cột "Flight Number" thành "flight_num", cột "Actual elapsed time (Minutes)" thành "actual_time"
8. Lưu df dưới dạng Parquet format với tên là "AA_DFW_ALL.parquet"
9. Đọc parquet "AA_DFW_ALL.parquet" => df_new
10. Tạo một bảng tạm 'flights'. Cho biết trung bình của 'actual_time' trong 'flight'
11. Caching các dòng dữ liệu duy nhất của df_new. Đếm số dòng. Cho biết thời gian thực hiện các công việc này.
12. Đếm lại số dòng. Cho biết thời gian thực hiện các công việc này.
13. Kiểm tra xem df_new có trong cache hay không? Nếu có thì bỏ df_new ra khỏi cache.

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark import SparkContext
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
```

In [3]:
```
sc =SparkContext()
```

In [4]:
```
spark = SparkSession(sc)
```

In [5]:
```
#1.
df = spark.read.csv(["AA_data"], header=True, inferSchema=True)
```

In [6]:
```
#2.
df.count()
```

Out[6]:
```
583718
```

In [7]:
```
df.show(5)
```

```
+----------------+------------+-----------------+----------------------
---+
|Date (MM/DD/YYYY)|Flight Number|Destination Airport|Actual elapsed time (Minut
es)|
+----------------+------------+-----------------+----------------------
---+
|      01/01/2014|           5|              HNL|
519|
|      01/01/2014|           7|              OGG|
505|
|      01/01/2014|          35|              SLC|
174|
|      01/01/2014|          43|              DTW|
153|
|      01/01/2014|          52|              PIT|
137|
+----------------+------------+-----------------+----------------------
---+
only showing top 5 rows
```

In [8]:
```
df.printSchema()
```

```
root
 |-- Date (MM/DD/YYYY): string (nullable = true)
 |-- Flight Number: integer (nullable = true)
 |-- Destination Airport: string (nullable = true)
 |-- Actual elapsed time (Minutes): integer (nullable = true)
```

In [9]:
```
from pyspark.sql.functions import col, udf
from pyspark.sql.functions import isnan, when, count, col
```

In [10]:
```
#3. Kiểm tra dữ liệu NaN, null
df.select([count(when(isnan(c), c)).alias(c) for c in df.columns]).toPandas().T
```

Out[10]:

| | 0 |
|---|---|
| Date (MM/DD/YYYY) | 0 |
| Flight Number | 0 |
| Destination Airport | 0 |
| Actual elapsed time (Minutes) | 0 |

In [11]:
```
# => Không có dữ liệu NaN
```

In [12]:
```
df.select([count(when(col(c).isNull(), c)).alias(c) for c in
           df.columns]).toPandas().T
```

Out[12]:

| | 0 |
|---|---|
| Date (MM/DD/YYYY) | 0 |
| Flight Number | 0 |
| Destination Airport | 0 |
| Actual elapsed time (Minutes) | 0 |

In [13]:
```
# Không có dữ liệu null
```

In [14]:
```
#4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.
```

In [15]:
```
num_rows = df.count()
```

In [16]:
```
num_dist_rows = df.distinct().count()
```

In [17]:
```
dup_rows = num_rows - num_dist_rows
```

In [18]:
```
dup_rows
```

Out[18]:
```
0
```

In [19]:
```
# Không có dữ liệu trùng
```

## Lazy processing operations

In [20]:
```
from pyspark.sql.functions import *
```

In [21]:
```
#5. Add the airport column using the F.lower() method
df = df.withColumn('airport', lower(df['Destination Airport']))
```

In [22]:
```
df = df.drop('Destination Airport')
```

In [23]:
```
df.show(5)
```

```
+----------------+-------------+-----------------------------+-------+
|Date (MM/DD/YYYY)|Flight Number|Actual elapsed time (Minutes)|airport|
+----------------+-------------+-----------------------------+-------+
|      01/01/2014|            5|                          519|    hnl|
|      01/01/2014|            7|                          505|    ogg|
|      01/01/2014|           35|                          174|    slc|
|      01/01/2014|           43|                          153|    dtw|
|      01/01/2014|           52|                          137|    pit|
+----------------+-------------+-----------------------------+-------+
only showing top 5 rows
```

In [24]:
```
#6. Add column date, using column Date (MM/DD/YYYY), drop  Date (MM/DD/YYYY)
```

In [25]:
```
df = df.withColumn('date', df['Date (MM/DD/YYYY)'])
```

In [26]:
```
df = df.drop('Date (MM/DD/YYYY)')
```

In [27]:
```
df.show(5)
```

```
+-------------+-----------------------------+-------+----------+
|Flight Number|Actual elapsed time (Minutes)|airport|      date|
+-------------+-----------------------------+-------+----------+
|            5|                          519|    hnl|01/01/2014|
|            7|                          505|    ogg|01/01/2014|
|           35|                          174|    slc|01/01/2014|
|           43|                          153|    dtw|01/01/2014|
|           52|                          137|    pit|01/01/2014|
+-------------+-----------------------------+-------+----------+
only showing top 5 rows
```

In [28]:
```
#7.
df = df.withColumnRenamed("Flight Number", "flight_num")
df = df.withColumnRenamed("Actual elapsed time (Minutes)", "actual_time")
```

In [29]:
```
df.show(5)
```

```
+----------+-----------+-------+----------+
|flight_num|actual_time|airport|      date|
+----------+-----------+-------+----------+
|         5|        519|    hnl|01/01/2014|
|         7|        505|    ogg|01/01/2014|
|        35|        174|    slc|01/01/2014|
|        43|        153|    dtw|01/01/2014|
|        52|        137|    pit|01/01/2014|
+----------+-----------+-------+----------+
only showing top 5 rows
```

## Parquet format

In [30]:
```
#8. Save the df DataFrame in Parquet format
df.write.parquet('AA_DFW_ALL.parquet.1', mode='overwrite')
```

In [31]:
```
#9 Read the Parquet file into a new DataFrame
df_new = spark.read.parquet('AA_DFW_ALL.parquet.1')
```

In [32]:
```
print(df_new.count())
```

```
583718
```

### SQL and Parquet

In [33]:
```
#10. Register the temp table
df_new.createOrReplaceTempView('flights')
```

In [34]:
```
# Run a SQL query of the average Actual elapsed time
avg_duration = spark.sql('SELECT avg(actual_time) from flights').collect()[0]
print('The average flight time is: %d' % avg_duration)
```

```
The average flight time is: 147
```

# Improving Performance

### Caching a DataFrame

- Caching can improve performance when reusing DataFrames

In [35]:
```
import time
```

In [36]:
```
11.
start_time = time.time()

# Add caching to the unique rows in df_new
df_new = df_new.distinct().cache()

# Count the unique rows in df_new, noting how long the operation takes
print("Counting %d rows took %f seconds" %
      (df_new.count(), time.time() - start_time))
```

```
Counting 583718 rows took 2.336898 seconds
```

In [37]:
```
# Count the rows again, noting the variance in time of a cached DataFrame
start_time = time.time()
print("Counting %d rows again took %f seconds" %
      (df_new.count(), time.time() - start_time))
```

```
Counting 583718 rows again took 0.877655 seconds
```

## Removing a DataFrame from cache

In [38]:
```
# Determine if df_new is in the cache
print("Is df_new cached?: %s" % df_new.is_cached)
print("Removing df_new from cache")

# Remove df_new from the cache
df_new.unpersist()

# Check the cache status again
print("Is df_new cached?: %s" % df_new.is_cached)
```

```
Is df_new cached?: True
Removing df_new from cache
Is df_new cached?: False
```

- Note: Converting to a larger number of files with approximately equal quantity of rows lets Spark decide how best to read the data.

## Cluster configurations

In [39]:
```
# Name of the Spark application instance
app_name = spark.conf.get('spark.app.name')

# Driver TCP port
driver_tcp_port = spark.conf.get('spark.driver.port')

# Number of join partitions
num_partitions = spark.conf.get('spark.sql.shuffle.partitions')

# Show the results
print("Name: %s" % app_name)
print("Driver TCP port: %s" % driver_tcp_port)
print("Number of partitions: %s" % num_partitions)
```

```
Name: pyspark-shell
Driver TCP port: 62414
Number of partitions: 200
```

In [40]:
```
# Store the number of partitions in variable
before = df_new.rdd.getNumPartitions()

# Configure Spark to use 500 partitions
spark.conf.set('spark.sql.shuffle.partitions', 500)

# Recreate the DataFrame using the departures data file
df_new = spark.read.parquet('AA_DFW_ALL.parquet').distinct()

# Print the number of partitions for each instance
print("Partition count before change: %d" % before)
print("Partition count after change: %d" % df_new.rdd.getNumPartitions())
```

```
Partition count before change: 200
Partition count after change: 500
```

In [41]:
```
### save data to json file
```

In [42]:
```
#df_new.write.json('AA_DFW_ALL.json')
```

# Chapter 5: Pre-processing Data

## Ex3 : Consumer Complaint

**Cho dữ liệu complaints.csv (Tham khảo chi tiết và download dữ liệu tại: https://www.consumerfinance.gov/data-research/consumer-complaints/ (https://www.consumerfinance.gov/data-research/consumer-complaints/))**

### Yêu cầu:

1. Đọc dữ liệu => data.
2. Cho biết dữ liệu có bao nhiêu dòng, in scheme. Hiển thị 3 dòng dữ liệu đầu tiên.
3. Kiểm tra dữ liệu NaN, null
4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.
5. Kiểm tra lại dữ liệu null. Tính tỉ lệ %.
6. Tạo dữ liệu mới, trong đó không có các cột có dữ liệu thiếu trên 30%.
7. Xoá các dòng có "Date received" là null và/hoặc "Product" là null
8. Tạo cột "date_from_text" chứa dữ liệu yyyy-dd-mm từ "Date received" nếu có, nếu không sẽ là ''. Lọc dữ liệu data_sub với yêu cầu "date_from_text" khác ''
9. Tạo cột "CCP" mới lấy dữ liệu từ cột "Consumer consent provided?". Với cột "CCP" hãy điền "Consent not provided" thay cho null
10. Tạo cột "SV" mới lấy dữ liệu từ cột "Submitted via". Với cột "SV" hãy điền "Other" thay cho null
11. Tạo cột "Sub-pr" mới lấy dữ liệu từ cột "Sub-product". Với cột "Sub-pr" hãy điền "I do not know" thay cho null
12. Tạo cột State_new lấy giá trị từ cột State theo điều kiện sau: nếu có thông tin State thì lấy thông tin, nếu không có thông tin thì điền "unknown"
13. Tạo cột Date_received với dữ liệu lấy từ cột "date_from_text", định dạng thời gian yyyy-mm-dd.
14. Tạo cột mới chứa ngày trong tuần day_of_week (1: Monday, 7: Sunday) với dữ liệu lấy từ cột 'Date_received'
15. Tạo cột year, month chứa năm, tháng với dữ liệu lấy từ cột 'Date_received'
16. Vẽ biểu đồ thể hiện tần suất nhận complaint theo ngày trong tuần
17. Hãy cho biết 20 sản phẩm nhận complaint nhiều nhất. Đó là những sản phẩm nào? Biểu diễn bằng đồ thị.

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark import SparkContext
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
```

In [3]:
```
sc =SparkContext()
```

In [4]:
```
spark = SparkSession(sc)
```

In [5]:
```
#1. Đọc dữ liệu => data.
# file from hdfs
# file_name = "hdfs://172.24.40.251:19000/complaints.csv"
file_name = "complaints.csv"
data = spark.read.csv(file_name, header=True, inferSchema=True)
```

In [6]:
```
#2. Cho biết dữ liệu có bao nhiêu dòng, in scheme.
# Hiển thị 3 dòng dữ liệu đầu tiên.
data.count()
```

Out[6]:
```
2083368
```

In [7]:
```
data.printSchema()
```

```
root
 |-- Date received: string (nullable = true)
 |-- Product: string (nullable = true)
 |-- Sub-product: string (nullable = true)
 |-- Issue: string (nullable = true)
 |-- Sub-issue: string (nullable = true)
 |-- Consumer complaint narrative: string (nullable = true)
 |-- Company public response: string (nullable = true)
 |-- Company: string (nullable = true)
 |-- State: string (nullable = true)
 |-- ZIP code: string (nullable = true)
 |-- Tags: string (nullable = true)
 |-- Consumer consent provided?: string (nullable = true)
 |-- Submitted via: string (nullable = true)
 |-- Date sent to company: string (nullable = true)
 |-- Company response to consumer: string (nullable = true)
 |-- Timely response?: string (nullable = true)
 |-- Consumer disputed?: string (nullable = true)
 |-- Complaint ID: string (nullable = true)
```

In [8]:
```
for row in data.head(3):
    print(row)
    print("\n")
```

```
Row(Date received='2019-09-24', Product='Debt collection', Sub-product='I do no
t know', Issue='Attempts to collect debt not owed', Sub-issue='Debt is not your
s', Consumer complaint narrative='transworld systems inc. ', Company public res
ponse=None, Company=None, State=None, ZIP code=None, Tags=None, Consumer consen
t provided?=None, Submitted via=None, Date sent to company=None, Company respon
se to consumer=None, Timely response?=None, Consumer disputed?=None, Complaint
ID=None)


Row(Date received='is trying to collect a debt that is not mine', Product=' not
owed and is inaccurate."', Sub-product=None, Issue='TRANSWORLD SYSTEMS INC', Su
b-issue='FL', Consumer complaint narrative='335XX', Company public response=Non
e, Company='Consent provided', State='Web', ZIP code='2019-09-24', Tags='Closed
with explanation', Consumer consent provided?='Yes', Submitted via='N/A', Date
sent to company='3384392', Company response to consumer=None, Timely response?=
None, Consumer disputed?=None, Complaint ID=None)


Row(Date received='2019-09-19', Product='Credit reporting, credit repair servic
es, or other personal consumer reports', Sub-product='Credit reporting', Issue
='Incorrect information on your report', Sub-issue='Information belongs to some
one else', Consumer complaint narrative=None, Company public response='Company
has responded to the consumer and the CFPB and chooses not to provide a public
response', Company='Experian Information Solutions Inc.', State='PA', ZIP code
='15206', Tags=None, Consumer consent provided?='Consent not provided', Submitt
ed via='Web', Date sent to company='2019-09-20', Company response to consumer
='Closed with non-monetary relief', Timely response?='Yes', Consumer disputed?
='N/A', Complaint ID='3379500')
```

In [9]:
```
from pyspark.sql.functions import col, udf
```

In [10]:
```
from pyspark.sql.functions import isnan, when, count, col
```

In [11]:
```
#3. Kiểm tra dữ liệu NaN, null
data.select([count(when(isnan(c), c)).alias(c) for c in
             data.columns]).toPandas().T
```

Out[11]:

| | 0 |
|---|---|
| Date received | 0 |
| Product | 0 |
| Sub-product | 0 |
| Issue | 0 |
| Sub-issue | 0 |
| Consumer complaint narrative | 0 |
| Company public response | 0 |
| Company | 0 |
| State | 0 |
| ZIP code | 0 |
| Tags | 0 |
| Consumer consent provided? | 0 |
| Submitted via | 0 |
| Date sent to company | 0 |
| Company response to consumer | 0 |
| Timely response? | 0 |
| Consumer disputed? | 0 |
| Complaint ID | 0 |

In [12]:
```
# => Không có dữ liệu NaN
```

```
In [13]: data.select([count(when(col(c).isNull(), c)).alias(c) for c in
                      data.columns]).toPandas().T
```

Out[13]:

| | 0 |
|---|---|
| Date received | 40 |
| Product | 209258 |
| Sub-product | 511176 |
| Issue | 325183 |
| Sub-issue | 931248 |
| Consumer complaint narrative | 1454171 |
| Company public response | 1412034 |
| Company | 573431 |
| State | 597994 |
| ZIP code | 673059 |
| Tags | 1720420 |
| Consumer consent provided? | 600884 |
| Submitted via | 583444 |
| Date sent to company | 659690 |
| Company response to consumer | 694041 |
| Timely response? | 711651 |
| Consumer disputed? | 721662 |
| Complaint ID | 728036 |

```
In [14]: # Có rất nhiều dữ liệu null theo từng cột

In [15]: #4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.

In [16]: num_rows = data.count()

In [17]: num_dist_rows = data.distinct().count()

In [18]: num_dist_rows
Out[18]: 2020609

In [19]: # Có dữ liệu trùng
         dup_rows = num_rows - num_dist_rows
```

In [20]:
```
dup_rows
```

Out[20]:
```
62759
```

In [21]:
```
# Xóa dữ liệu trùng
data = data.drop_duplicates()
```

In [22]:
```
data.distinct().count()
```

Out[22]:
```
2020609
```

In [23]:
```
# Hết dữ liệu trùng
```

In [24]:
```
#5. Kiểm tra lại dữ liệu null. Tính tỉ lệ %. Xem xét cách xử lý dữ liệu null
null_data = data.select([count(when(col(c).isNull(), c)).alias(c) for c in
                         data.columns]).toPandas().T
```

In [25]:
```
type(null_data)
```

Out[25]:
```
pandas.core.frame.DataFrame
```

In [26]:
```
null_data
```

Out[26]:

| | 0 |
|---|---|
| Date received | 39 |
| Product | 178373 |
| Sub-product | 472168 |
| Issue | 281416 |
| Sub-issue | 884470 |
| Consumer complaint narrative | 1405173 |
| Company public response | 1353196 |
| Company | 513337 |
| State | 537111 |
| ZIP code | 611785 |
| Tags | 1659004 |
| Consumer consent provided? | 539324 |
| Submitted via | 521790 |
| Date sent to company | 597981 |
| Company response to consumer | 632256 |
| Timely response? | 649891 |
| Consumer disputed? | 659883 |
| Complaint ID | 666220 |

```
In [27]: null_data["percentage"] = (null_data[0]/num_dist_rows)*100
```

```
In [28]: null_data
```

Out[28]:

| | 0 | percentage |
|---|---|---|
| Date received | 39 | 0.001930 |
| Product | 178373 | 8.827685 |
| Sub-product | 472168 | 23.367608 |
| Issue | 281416 | 13.927286 |
| Sub-issue | 884470 | 43.772447 |
| Consumer complaint narrative | 1405173 | 69.542054 |
| Company public response | 1353196 | 66.969711 |
| Company | 513337 | 25.405064 |
| State | 537111 | 26.581639 |
| ZIP code | 611785 | 30.277258 |
| Tags | 1659004 | 82.104158 |
| Consumer consent provided? | 539324 | 26.691161 |
| Submitted via | 521790 | 25.823403 |
| Date sent to company | 597981 | 29.594098 |
| Company response to consumer | 632256 | 31.290368 |
| Timely response? | 649891 | 32.163125 |
| Consumer disputed? | 659883 | 32.657629 |
| Complaint ID | 666220 | 32.971248 |

```
In [29]: #6. Tạo dữ liệu mới, trong đó không có các cột có dữ liệu thiếu trên 30%
         data_sub = data.select(["Date received", "Product",
                                 "Sub-product", "Issue",
                                 "Company", "State",
                                 "Consumer consent provided?",
                                 "Submitted via"])
```

In [30]:
```
data_sub.show(2)
```

```
+--------------------+-----+--------------------+--------------+--------------------
|      Date received|                 Product|     Sub-product|                Issu
e|             Company|State|Consumer consent provided?|Submitted via|
+--------------------+-----+--------------------+--------------+--------------------
|o While we do not...| we have been inf...|                null|                nul
l|                null| null|                null|          null|
|          2019-09-19|Credit reporting,...|Credit reporting|Improper use of
y...|TRANSUNION INTERM...|   WI|   Consent provided|           Web|
+--------------------+-----+--------------------+--------------+--------------------
only showing top 2 rows
```

In [31]:
```
#7. Xoá các dòng có "Date received" là null và/hoặc "Product" là null
```

In [32]:
```
data_sub = data_sub.dropna(how="any",
                           subset=["Date received", "Product"])
```

In [33]:
```
data_sub.select([count(when(col(c).isNull(), c)).alias(c) for c in
                 data_sub.columns]).toPandas().T
```

Out[33]:

| | 0 |
|---|---|
| Date received | 0 |
| Product | 0 |
| Sub-product | 332225 |
| Issue | 141159 |
| Company | 373348 |
| State | 397120 |
| Consumer consent provided? | 399331 |
| Submitted via | 381799 |

In [34]:
```
#8. Tạo cột "date_from_text" chứa dữ liệu yyyy-dd-mm từ "Date received" nếu có
# nếu không sẽ là ''.
# Lọc dữ liệu data_sub với yêu cầu "date_from_text" khác ''
```

In [35]:
```
from pyspark.sql import functions as F
```

In [36]:
```
data_sub = data_sub.withColumn("date_from_text",
                               F.regexp_extract(data_sub["Date received"],
                                                r"(\d{4}-\d{1,2}-\d{1,2})", 0))
```

In [37]:
```
data_sub.select("date_from_text", "Date received").head(5)
```

Out[37]:
```
[Row(date_from_text='', Date received='o While we do not know the details of th
e XXXX/CMHL compensation structure'),
 Row(date_from_text='2019-09-19', Date received='2019-09-19'),
 Row(date_from_text='', Date received='-XX/XX/XXXX letter where Certegy Payment
Solutions '),
 Row(date_from_text='2019-04-04', Date received='2019-04-04'),
 Row(date_from_text='2019-02-18', Date received='2019-02-18')]
```

In [38]:
```
data_sub = data_sub.filter(data_sub["date_from_text"]!='')
```

In [39]:
```
data_sub.select("date_from_text", "Date received").head(10)
```

Out[39]:
```
[Row(date_from_text='2019-09-19', Date received='2019-09-19'),
 Row(date_from_text='2019-04-04', Date received='2019-04-04'),
 Row(date_from_text='2019-02-18', Date received='2019-02-18'),
 Row(date_from_text='2019-05-25', Date received='2019-05-25'),
 Row(date_from_text='2019-03-28', Date received='2019-03-28'),
 Row(date_from_text='2018-12-28', Date received='2018-12-28'),
 Row(date_from_text='2019-08-08', Date received='2019-08-08'),
 Row(date_from_text='2019-04-04', Date received='2019-04-04'),
 Row(date_from_text='2019-04-22', Date received='2019-04-22'),
 Row(date_from_text='2019-03-27', Date received='2019-03-27')]
```

In [40]:
```
#9. Tạo cột "CCP" mới lấy dữ liệu từ cột "Consumer consent provided?"
# Với cột "CCP" hãy điền "Consent not provided" thay cho null
data_sub.select(["Consumer consent provided?"]).distinct().count()
```

Out[40]:
```
12830
```

In [41]:
```
data_sub = data_sub.withColumn("CCP", data_sub["Consumer consent provided?"])
```

In [42]:
```
data_sub = data_sub.fillna("Consent not provided", subset="CCP")
```

In [43]:
```
#10. Tạo cột "SV" mới lấy dữ liệu từ cột "Submitted via"
# Với cột "SV" hãy điền "Other" thay cho null
```

In [44]:
```
data_sub = data_sub.withColumn("SV", data_sub["Submitted via"])
```

In [45]:
```
data_sub = data_sub.fillna("Other", subset="SV")
```

In [46]:
```
#11. Tạo cột "Sub-pr" mới lấy dữ liệu từ cột "Sub-product"
# Với cột "Sub-pr" hãy điền "I do not know" thay cho null
```

In [47]:
```
data_sub = data_sub.withColumn("Sub-pr", data_sub["Sub-product"])
```

In [48]:
```
data_sub = data_sub.fillna("I do not know", subset="Sub-pr")
```

In [49]:
```
data_sub.select([count(when(col(c).isNull(), c)).alias(c) for c in
                 data_sub.columns]).toPandas().T
```

Out[49]:

| | 0 |
|---|---|
| Date received | 0 |
| Product | 0 |
| Sub-product | 294541 |
| Issue | 0 |
| Company | 147497 |
| State | 173009 |
| Consumer consent provided? | 174052 |
| Submitted via | 156052 |
| date_from_text | 0 |
| CCP | 0 |
| SV | 0 |
| Sub-pr | 0 |

In [50]:
```
#12. Tạo cột State_new lấy giá trị từ cột State theo điều kiện sau:
# nếu có thông tin State thì lấy thông tin
# nếu không có thông tin thì điền "unknown"
```

In [51]:
```
data_sub = data_sub.withColumn('State_new',
                               when(col('State').isNull(),'unknown')
                               .otherwise(col('State')))
```

In [52]:
```
from pyspark.sql.functions import to_date, dayofweek, to_timestamp
from pyspark.sql import types
from pyspark.sql.functions import col, udf
from datetime import datetime
from pyspark.sql.types import DateType
```

In [53]:
```
#13. Tạo cột Date_received với dữ liệu lấy từ cột "date_from_text",
# định dạng thời gian yyyy-mm-dd .
func =  udf (lambda x: datetime.strptime(x, '%Y-%m-%d'), DateType())
data_sub = data_sub.withColumn('Date_received', func(col('date_from_text')))
```

In [54]:
```
data_sub.printSchema()
```

```
root
 |-- Date received: string (nullable = true)
 |-- Product: string (nullable = true)
 |-- Sub-product: string (nullable = true)
 |-- Issue: string (nullable = true)
 |-- Company: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Consumer consent provided?: string (nullable = true)
 |-- Submitted via: string (nullable = true)
 |-- date_from_text: string (nullable = true)
 |-- CCP: string (nullable = false)
 |-- SV: string (nullable = false)
 |-- Sub-pr: string (nullable = false)
 |-- State_new: string (nullable = true)
 |-- Date_received: date (nullable = true)
```

In [55]:
```
data_sub.select('Date_received').show(5)
```

```
+-------------+
|Date_received|
+-------------+
|   2019-09-19|
|   2019-04-04|
|   2019-02-18|
|   2019-05-25|
|   2019-03-28|
+-------------+
only showing top 5 rows
```

In [56]:
```
#14. Tạo cột mới chứa ngày trong tuần day_of_week (1: Monday, 7: Sunday)
# với dữ liệu lấy từ cột 'Date_received'
data_sub = data_sub.withColumn('Day_of_Week', dayofweek('Date_received'))
```

In [57]:
```
from pyspark.sql.functions import year, month
```

In [58]:
```
#15. Tạo cột year, month chứa năm, tháng với dữ liệu lấy từ cột 'Date_received'
data_sub = data_sub.withColumn('year', year('Date_received')) \
                   .withColumn('month', month('Date_received'))
```

In [59]:
```
data_sub.select('Date_received', 'Day_of_Week', 'month', 'year').show(5)
```

```
+-------------+-----------+-----+----+
|Date_received|Day_of_Week|month|year|
+-------------+-----------+-----+----+
|   2019-09-19|          5|    9|2019|
|   2019-04-04|          5|    4|2019|
|   2019-02-18|          2|    2|2019|
|   2019-05-25|          7|    5|2019|
|   2019-03-28|          5|    3|2019|
+-------------+-----------+-----+----+
only showing top 5 rows
```

In [60]:
```
data_sub.select([count(when(col(c).isNull(), c)).alias(c) for c in
                 data_sub.columns]).toPandas().T
```

Out[60]:

| | 0 |
|---|---|
| Date received | 0 |
| Product | 0 |
| Sub-product | 234541 |
| Issue | 0 |
| Company | 147497 |
| State | 173009 |
| Consumer consent provided? | 174052 |
| Submitted via | 156052 |
| date_from_text | 0 |
| CCP | 0 |
| SV | 0 |
| Sub-pr | 0 |
| State_new | 0 |
| Date_received | 0 |
| Day_of_Week | 0 |
| year | 0 |
| month | 0 |

In [61]:
```
#16. Vẽ biểu đồ thể hiện tần suất nhận complaint theo ngày trong tuần
df = data_sub.select('Day_of_Week').toPandas()
```

In [62]:
```
df.hist()
```

Out[62]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001C1C917A5F8>]],
      dtype=object)
```

In [63]:
```
# Khách hàng complaint nhiều nhất vào thứ 4 và thứ 5
```

In [66]:
```
#17. Hãy cho biết 20 sản phẩm nhận complaint nhiều nhất.
# Đó là những sản phẩm nào? Biểu diễn bằng đồ thị.
df_products = data_sub.groupby('product').count().orderBy('count',
                                                ascending=False).toPandas()
```

In [67]:
```
df_products.head(20)
```

Out[67]:

| | product | count |
|---|---|---|
| 0 | Credit reporting, credit repair services, or o... | 331635 |
| 1 | Mortgage | 295097 |
| 2 | Debt collection | 278961 |
| 3 | Credit reporting | 139825 |
| 4 | Credit card | 89175 |
| 5 | Bank account or service | 86193 |
| 6 | Credit card or prepaid card | 67849 |
| 7 | Checking or savings account | 57461 |
| 8 | Student loan | 56365 |
| 9 | Consumer Loan | 31592 |
| 10 | Vehicle loan or lease | 15540 |
| 11 | Money transfer, virtual currency, or money ser... | 14165 |
| 12 | Payday loan, title loan, or personal loan | 11962 |
| 13 | Payday loan | 5541 |
| 14 | Money transfers | 5354 |
| 15 | Prepaid card | 3819 |
| 16 | Other financial service | 1059 |
| 17 | Virtual currency | 18 |

In [68]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [69]:
```
plt.figure(figsize=(10,8))
sns.barplot(data = df_products, x="count", y="product", )
plt.show()
```