Feature Engineering with Pyspark (Divyanshu)

(1) Outlier Removal - Quantitative Filtering

- from pyspark.sql.functions import mean, stddev
- μ = df.agg({'x': 'mean'}).collect()[0][0]
- σ = df.agg({'x': 'stddev'}).collect()[0][0]
- low = $\mu - 3*\sigma$
- High = $\mu + 3*\sigma$
- df = df.filter((df['x'] ≤ high) & (df['x'] ≥ low)):

df.where also works.

(2) Text based filtering : where + like.

- df = df.where(~df['gender'].like('MALE'))

~ → NOT

(3) Dropping Columns : -df.drop(*['x1', 'x2'])

* → unpacker.

(4) Dropping NULL Records.

- df.dropna()
- df.dropna(how='all', subset=['x1', 'x2'])
- df.dropna(thresh=2)

(5) Drop Duplicates w.r.t a col - df.dropDuplicates(['zipcode'])

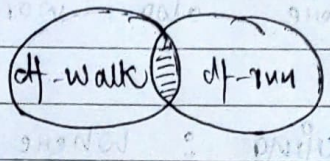
(6) Function for Minmax scaling (UDF) →

```
-def min_max_scaler(df, cols_to_scale):
    for col in cols_to_scale:
        max = df.agg({col: 'max'}).collect()[0][0]
        min = df.agg({col: 'min'}).collect()[0][0]
        new_col_name = 'scaled_' + col
        df = df.withColumn(new_col_name, ((df[col] - min)/(max - min)))
    return df
```

(7) Create a UDF that drops columns with NULLs > Threshold :

```
def column_dropper(df, threshhold):
    total_rows = df.count()
    for col in df.columns:
        missing_rows = df.filter(df[col].isNull()).count()
        missing% = missing_rows / total_rows
        if missing% > threshold:
            df = df.drop(col)
    return df
```

(8) LEFT Join Example →




```
- df_walk = df_walk.withColumn('LAT', df_walk['LAT'].cast('double'))
- df_walk = df_walk.withColumn('LON', df_walk['LON'].cast('double'))

- df_run = df_run.withColumn('LAT', round('LAT', 5))
-   ''                      ('LON', round(...))

- # join condn
- condn = [df_walk['LAT'] == df_run['LAT'],
           df_walk['LON'] == df_run['LON']]

- Join
- df = df_walk.join(df_run, on=condn, how='left')
```

(5) Creating New features off already existing.

```
- df = df.withColumn('Area', df['L'] * df['B'])
```

⑥ Working with DateTime Features.

- from pyspark.sql.functions import to_date
- df = df.withColumn('DATE', to_date('mydate'))

→ if you wish to keep the time-component, use → to_timestamp

- other options : from pyspark.sql.functions import year
  : " " " " import month.
  dayofmonth
  weekofyear

to take delta btw 2 dates. → datediff.

(#) Creating LAGGED Features →

- window()
- lag(col, count=1)

- from pyspark.sql.functions import lag
- from pyspark.sql.window import Window

- w = Window().orderBy(df['Date'])
- df = df.withColumn('Xlagby1wk', lag('x', count=1).over(w))

⇓

DATE	X	Xlagby1wk
2013-01-01	2.3 ↓	null
2013-01-08	3.4 ↓	2.3
2013-01-15	4.2	3.4

Note: PySpark week starts → Sunday (=1) → Saturday (=7)

(#) Joining on DateTime Features

A → has 'Year' column
B → has 'Date' column.

→ B = B.withColumn('Year', year('Date'))
→ condn = [A['id'] == B['id'], A['Year'] == B['Year']]
→ A = A.join(B, on = condn, how = 'left')