Basic Column Expectations

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Davina Moossazadeh
Data Scientist

datacamp

The Shein Footwear Dataset

	name	link	price_usd	mark_price_usd	star_rating	colour	seller_name	review_count	sku_id	hero_image
0	Women's Casual Sports...	https://us.shein.com/...	19.40	24.40	4.83	Khaki	Womens Shoes	77	sx2305270713671620	//img.ltwebstatic.com...
1	2024 New Spring/Autum...	https://us.shein.com/...	14.90	18.80	0.00	NaN	Limeiya Factory	0	sx2404088145930048	//img.ltwebstatic.com...
2	Women's Casual Slip-o...	https://us.shein.com/...	7.50	18.80	4.92	NaN	Dawanfu	66	sx2309025401152825	//img.ltwebstatic.com...
3	Women Mesh Breathable...	https://us.shein.com/...	12.06	21.30	0.00	Purple	NaN	41	sx2308191847675463	//img.ltwebstatic.com...
4	Women's Fashionable C...	https://us.shein.com/...	13.53	17.05	0.00	Grey	NaN	100+	sx2307234044444164	//img.ltwebstatic.com...

[1] https://www.kaggle.com/datasets/atharvataras/shein-footwear-dataset

Row-level Expectations

- Row-Level Expectations are applied to each row independently
  - succeed only if the condition holds for every row

Row-level Expectations

Missingness Expectation

```
gx.expectations.ExpectColumnValuesToNotBeNull(
    column="colour"
)
```

Type Expectation

```
gx.expectations.ExpectColumnValuesToBeOfType(
    column="review_count", type_="str"
)
```

Aggregate-level Expectations: distinct values

Distinct values Expectation

```
gx.expectations.ExpectColumnDistinctValuesToEqualSet(
    column="seller_name", value_set={"Womens Shoes"}
)
```

Aggregate-level Expectations: unique value count

Unique value count Expectation

```
gx.expectations.ExpectColumnUniqueValueCountToBeBetween(
    column="review_count", min_value=5, max_value=101
)
```

Aggregate-level Expectations: uniqueness

Uniqueness Expectation

```
gx.expectations.ExpectColumnValuesToBeUnique(
    column="sku_id"
)
```

Aggregate-level Expectations: mode

Mode Expectation

```
gx.expectations.ExpectColumnMostCommonValueToBeInSet(
    column="colour", value_set={"Khaki", "Purple", "Grey"}
)
```

Cheat sheet

Row-level Expectations:

```
ExpectColumnValuesToNotBeNull(
    column: str
)
ExpectColumnValuesToBeOfType(
    column: str, type_: str
)
```

Aggregate-level Expectations:

```
ExpectColumnDistinctValuesToEqualSet(
    column: str, value_set: set
)
ExpectColumnUniqueValueCountToBeBetween(
    column: str,
    min_value: int, max_value: int
)
ExpectColumnValuesToBeUnique(column: str)
ExpectColumnMostCommonValueToBeInSet(
    column: str, value_set: set
)
```

Let's practice!

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Type-Specific Expectations

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Davina Moossazadeh
Data Scientist

datacamp

Numeric Expectations: aggregate-level

```
gx.expectations.ExpectColumnMeanToBeBetween(
    column="mark_price_usd", min_value=20, max_value=25
)
gx.expectations.ExpectColumnMedianToBeBetween(
    column="mark_price_usd", min_value=20, max_value=25
)
gx.expectations.ExpectColumnStdevToBeBetween(
    column="mark_price_usd", min_value=15, max_value=20
)
gx.expectations.ExpectColumnSumToBeBetween(
    column="mark_price_usd", min_value=20000, max_value=21000
)
```

Numeric Expectations: row-level

Set row-level Expectations for value ranges:

```
gx.expectations.ExpectColumnValuesToBeBetween(
    column="star_rating", min_value=0, max_value=5
)
```

or order:

```
gx.expectations.ExpectColumnValuesToBeIncreasing(
    column="price_usd"
)
gx.expectations.ExpectColumnValuesToBeDecreasing(
    column="price_usd"
)
```

String Expectations

Set string-level Expectations (these are all on the row-level):

```
gx.expectations.ExpectColumnValueLengthsToEqual(
    column="sku_id", value=18
)
gx.expectations.ExpectColumnValuesToMatchRegex(
    column="link", regex="^https://us.shein.com/[\w-]+"
)
regex_list = [
    "//img.ltwebstatic.com/images3_(spmp)|(pi)/202[0-4]/[0-1][0-9]/.*",
    "//sheinsz.ltwebstatic.com/she_dist/images/bg-g.*"
]
gx.expectations.ExpectColumnValuesToMatchRegexList(
    column="hero_image", regex_list=regex_list
)
```

String parseability Expectations

Set Expectations for string columns to ensure data is parseable

`ExpectColumnValuesToBeDateutilParseable()` for dates:

```
gx.expectations.ExpectColumnValuesToBeDateutilParseable(
    column="colour"
)
```

`ExpectColumnValuesToBeJsonParseable()` for JSON:

```
gx.expectations.ExpectColumnValuesToBeJsonParseable(
    column="colour"
)
```

Cheat sheet

Numeric Expectations:

```
ExpectColumn<METRIC>ToBeBetween(
    column, min_value, max_value
)
# <METRIC> in {"Mean", "Median", "Stdev", "Sum"}

ExpectColumnValuesToBeBetween(
    column, min_value, max_value
)

ExpectColumnValuesToBeIncreasing(column)

ExpectColumnValuesToBeDecreasing(column)
```

String Expectations:

```
ExpectColumnValueLengthsToEqual(column, value)

ExpectColumnValuesToMatchRegex(column, regex)

ExpectColumnValuesToMatchRegexList(
    column, regex_list
)

ExpectColumnValuesToBeDateutilParseable(column)

ExpectColumnValuesToBeJsonParseable(column)
```

Let's practice!

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Conditional Expectations

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Davina Moossazadeh
Data Scientist

datacamp

What are Conditional Expectations?

Conditional Expectations - Expectations for a subset of the data

Why? Because some variables are dependent on the values of other variables

For example:

- An Expectation that the value of a column `star_rating` must be `0` for all rows with a value of `0` for `review_count`

[1] https://docs.greatexpectations.io/docs/core/customize_expectations/expectation_conditions/

Syntax for Conditional Expectations

Dataset Expectations can be converted into Conditional Expectations with two additional arguments:

1. `row_condition`
   - a boolean expression string that defines the subset of data to which to apply the Conditional Expectation
2. `condition_parser`
   - a string that defines the syntax of the `row_condition`

The condition parser

When implementing Conditional Expectations with pandas, this argument must be set to `"pandas"`

```
expectation = gx.Expect...(
    **kwargs,
    condition_parser="pandas",
    row_condition=...
)
```

The row condition

pandas Syntax

```
df["foo"] == 'Two Two'
```

```
df["foo"].notNull()
```

```
df["foo"] <= datetime.date(2023, 3, 13)
```

```
(df["foo"] < 5) & (df["foo"] >= 3.14)
```

```
df["foo"].str.startswith("bar")
```

Great Expectations `row_condition`

```
'foo == "Two Two"'
```

```
'foo.notNull()'
```

```
'foo <= datetime.date(2023, 3, 13)'
```

```
'(foo > 5) & (foo <= 3.14)'
'foo > 5 and foo <= 3.14'
```

```
'foo.str.startswith("bar")'
```

The row condition

Rules

1. Don't use single quotes inside
   - `row_condition="foo=='Two Two'"`
   - `row_condition='foo=="Two Two"'` 
2. Don't use line breaks inside
   - ```
     row_condition="""
     foo=="Two Two"
     """
     ```
   - `row_condition='foo=="Two Two"'` 
     ```

Example Expectation: star rating

No condition

```
expectation = gx.expectations.\
ExpectColumnValuesToBeBetween(
    column="price_usd",
    max_value=10,
)
validation_results = batch.validate(
    expect=expectation
)
print(validation_results.success)
```

```
False
```

Conditional

```
expectation = gx.expectations.\
ExpectColumnValuesToBeBetween(
    column="price_usd",
    max_value=10,
    condition_parser='pandas',
    row_condition='mark_price_usd < 10',
)
validation_results = batch.validate(
    expect=expectation
)
print(validation_results.success)
```

```
True
```

Let's practice!

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Wrap-Up

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS

Davina Moossazadeh
Data Scientist

Chapter 1: Connecting to Data

Create a Data Context

```
context = gx.get_context()
```

Chapter 1: Connecting to Data

Connect to Data

Create Data Source from Data Context:

```
data_source = context.data_sources.add_pandas(
    name: str
)
```

Create Data Asset from Data Source:

```
data_asset = data_source.add_dataframe_asset(
    name: str
)
```

Chapter 1: Connecting to Data

Read Data in Batches

Create Batch Definition from Data Asset:

```
batch_definition = data_asset. \
add_batch_definition_whole_dataframe(
  name: str
)
```

Create Batch from Batch Definition:

```
batch = batch_definition.get_batch(
  batch_parameters={"dataframe": dataframe}
)
```

Get Batch DataFrame rows:

```
batch.head(fetch_all: bool)
```

Get Batch DataFrame column list:

```
batch.columns()
```

Chapter 2: Establishing Expectations

Create Expectations

Create an Expectation:

`gx.expectations.Expect...(...)`

Create a row count Expectation:

```
expectation = gx.expectations. \
ExpectTableRowCountToEqual(
    value: int
)
```

Validate Expectation:

```
validation_results = batch.validate(
    expect=expectation
)
```

Check Validation Results:

```
validation_results.describe()
validation_results.success
validation_results.result
```

Chapter 2: Establishing Expectations

Schema Expectations

Shape Expectations:

```
ExpectTableRowCountToEqual(value: int)
ExpectTableRowCountToBeBetween(
    min_value: int, max_value: int
)
ExpectTableColumnCountToEqual(
    value: int
)
ExpectTableColumnCountToBeBetween(
    min_value: int, max_value: int
)
```

Column name Expectations:

```
ExpectTableColumnsToMatchSet(
    column_set: set
)
ExpectColumnToExist(column: str)
```

Chapter 2: Establishing Expectations

Create a Suite of Expectations

Create Expectation Suite:

```
suite = gx.ExpectationSuite(name: str)
```

Add Expectation to Suite:

```
suite.add_expectation(expectation)
```

Access Suite's Expectations:

```
suite.expectations
```

Validate Expectation Suite:

```
validation_results = batch.validate(
    expect=suite
)
```

Check Validation Results:

```
validation_results.success
validation_results.describe()
```

Chapter 2: Establishing Expectations

Validate Expectation Suites

Add Expectation Suite to Data Context:

```
context.suites.add(suite)
```

Create Validation Definition:

```
validation_definition = \
gx.ValidationDefinition(
  name: str,
  data=batch_definition,
  suite=suite
)
```

Run Validation:

```
validation_results = \
validation_definition.run(
  batch_parameters={"dataframe": dataframe}
)
```

Check Validation Results:

```
validation_results.success
validation_results.describe()
```

Chapter 3: GX in Practice

Deploy Validation Definitions

Add Validation Definition to Data Context:

```
context.validation_definitions.add(
    validation_definition
)
```

Create Checkpoint:

```
checkpoint = gx.Checkpoint(
    name: str,
    validation_definitions: list,
)
```

Run Checkpoint:

```
checkpoint_results = checkpoint.run(
    batch_parameters={
        "dataframe": dataframe
    }
)
```

Check Checkpoint Results:

```
checkpoint_results.success
```

Chapter 3: GX in Practice

Update Expectation Suites

Copy Expectation:

```
expectation_copy = expectation.copy()
expectation_copy.id = None
```

Check if Expectation is in Suite:

```
expectation in suite.expectations
```

Delete Expectation:

```
suite.delete_expectation(expectation)
```

Update Expectation value:

```
expectation.value = new_value
```

Save changes to Expectation:

```
expectation.save()
```

Save changes to Expectation Suite:

```
suite.save()
```

Chapter 3: GX in Practice

Manage Components

Add a component to Data Context:

```
context.data_sources.add(data_source)

context.suites.add(suite)

context.validation_definitions.add(
    validation_definition
)

context.checkpoints.add(checkpoint)
```

Retrieve a component:

```
.get(name: str)
```

List components:

```
.all()
```

Delete a component:

```
.delete(name: str)
```

Chapter 4: All About Expectations

Basic Column Expectations

Row-level Expectations:

```
ExpectColumnValuesToNotBeNull(
    column: str
)
ExpectColumnValuesToBeOfType(
    column: str, type_: str
)
```

Aggregate-level Expectations:

```
ExpectColumnDistinctValuesToEqualSet(
    column: str, value_set: set
)
ExpectColumnUniqueValueCountToBeBetween(
    column: str,
    min_value: int, max_value: int
)
ExpectColumnValuesToBeUnique(column: str)
ExpectColumnMostCommonValueToBeInSet(
    column: str, value_set: set
)
```

Chapter 4: All About Expectations

Type-Specific Expectations

Numeric Expectations:

```
ExpectColumn<METRIC>ToBeBetween(
  column: str, min_value: int, max_value: int
)
# <METRIC> in
# {"Mean", "Median", "Stdev", "Sum"}
```

```
ExpectColumnValuesToBeBetween(
  column: str, min_value: int, max_value: int
)
ExpectColumnValuesToBeIncreasing(column: str)
ExpectColumnValuesToBeDecreasing(column: str)
```

String Expectations:

```
ExpectColumnValueLengthsToEqual(
  column: str, value: int
)
ExpectColumnValuesToMatchRegex(
  column: str, regex: str
)
ExpectColumnValuesToMatchRegexList(
  column: str, regex_list: list
)
ExpectColumnValuesToBe{Dateutil,Json}Parseable(
  column: str
)
```

Chapter 4: All About Expectations

Conditional Expectations

```
expectation = gx.expectations.Expect...(
    expetation_parameters,
    ...,
    condition_parser='pandas',
    row_condition: str,
)
```

Final Words

Resources:

https://docs.greatexpectations.io/docs/core/introduction

https://docs.greatexpectations.io/docs/reference

Expectation Gallery:

www.greatexpectations.io/expectations

Congratulations!

INTRODUCTION TO DATA QUALITY WITH GREAT EXPECTATIONS