1CDC DATA PLATFORM

Foundry Pyspark Cheat Sheet

Developers and Data Analysts are encouraged to use this Foundry Pyspark Cheat Sheet as an everyday reference guide for coding python transformations in 1CDP.

IMPORT

PYSPARK IMPORTS

from pyspark.sql import	
functions as F	# Imports PySpark SQL functions
types as T	# Imports PySpark SQL types

COMMON FOUNDRY IMPORTS

from transforms.api import	
transform	# A decorator used to define a transform function in Python
transform_df	# A decorator to define a transform function that outputs a dataframe
FileSystem	# A call used to access files in an unstructured dataset
Input	# A class used to specify input datasets for a transform
Output	# A class used to specify output datasets for a transform
Incremental	# Used for incremental builds (i.e., append builds)
configure	# Infrequently used to edit the spark profile in each instance
Pipeline	# A class used in Java to define a pipeline and register transforms
Markings	# Used to add/remove security markings

FOUNDRY MEDIA SET IMPORTS

from transforms.mediasets import	
MediaSetInput	# A class used to specify an input media set
MediaSetOutput	# A class used to specify an output media set

FOUNDRY MODEL IMPORTS

from palantir_models.transforms import	
ModelInput	# A class used for common model input
ModelOutput	# A class used for a common model output

INPUT/OUTPUT A DATAFRAME

WITH TRANSFORM_DF

```
from transforms.api import transform_df, Input, Output

@transform(
    data_in=Input(“input_path”),
    Output("output_path"),
)
def compute(data_in):
      return data_in
```

WITH TRANSFORM

```
from transforms.api import transform, Output
from pyspark.sql import types as T
from pyspark.sql import functions as F
@transform(
    data_in=Input(“input_path”),
    data_out=Output("output_path"),
)
def compute(data_out, data_in):
      df = data_in.dataframe()

      data_out.write_dataframe(df)
```

DE-DUPLICATE DATA

```
.distinct()                          # Drops Duplicates for whole row
.drop_duplicates(subset=['age'])     # Drop Duplicates on specific column(s)
```

QUERY/SELECT DATA

```
.select('age', 'id')                                 # Select Statement
.select(F.col('age').alias('Patient_Age'), 'id')    # Select that Changes Col Name
.select(F.col('age') > 24, 'id')                     # Selects rows where age > 24
```

USE BASIC FUNCTIONS

```
F.lit('Any Data')         # Insert literal Value
F.col('col_a')            # Column Name
F.min()                   # Return Min Value
F.max()                   # Return Max Value
F.avg()                   # Return Average Value
F.concat(data)            # Concatenate Data
F.concat_ws(sep, data)    # Concatenate Data with a spacer
F.sha2(col, 256)          # Returns the hex string for sha-256 hash function
df.collect()        # Return list of DataFrame
df.dtypes           # Return df column names and data types
df.columns          # Return the columns of df
df.schema           # Return the schema
df.head()           # Return first n rows
df.first()          # Return first row
df.describe()       # Compute summary statistics
df.count()          # Count the number of rows in df
```

FILTER DATA

```
.filter(df["age"] > 24)             # Filter on Number
.filter(df.city != 'Atlanta')       # Filter on String
```

MODIFY COLUMNS

ADD A COLUMN

```
.withColumn('new_age', F.col('age') + 1)    # Creates New Column with Age + 1
.withColumn('age_cat', F.when(F.col('age') < 30,  # Create new Column using When
condition
                           F.lit('Group 1')
                           ).otherwise(F.lit('Group 2')))
```

CHANGE A COLUMN

```
.withColumnRenamed('age', 'Patient_Age')   # Rename Column
F.col(‘age’).cast(T.StringType())          # Changes Datatype of Column
F.col(‘age’).asType(T.StringType())        # Alternate way to change Dtype
```

DROP VALUES

```
.dropna()                # Drops all Null Values
.drop('age')             # Drops Age Column
```

AGGREGATE DATA

GROUP BY

```
.groupBy('age').count()              # Group By and Return Count
.groupBy('age').avg()                # Group By and Return Average
```

AGG

```
df.agg({"age": "max"})            # Aggregate Data short for groupBy.agg()
df.agg(F.min(df.age))             # Alternate Agg syntax for Min value
```

USE SPARK CONTEXT

```
dictionary_to_df = {'col_a': [1, 2, 3], 'col_b': ['Data', 'Row', 'Example']}
schema = T.StructType[
                T.StructField("col_a", T.IntegerType()),
                T.StructField("col_b", T.StringType())]
df = ctx.spark_session.createDataFrame(dictionary_to_df, schema)
```

PARTITION DATA

```
# Returns a Window partitioned by state/county

Window().partitionBy("state", "county").orderBy(“state”))
```

SORT DATA

```
.sort(df.age.desc())                                  # Sort
.sort('age', ascending=False)                         # Sort (Different Format)
.orderBy(['age' , 'location'], ascending=[0, 1])  # Sort on Multiple Columns
```

JOIN DATA*

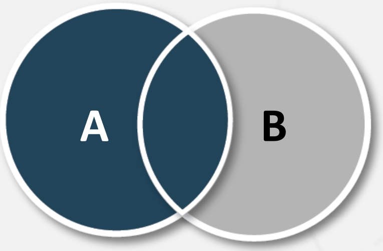
```
.join(df2, "state", 'left')                     # Join
.join(df2, df.state == df2.name, 'fullouter')   # Join with mismatched key names
```

See next page for more information about Pyspark Joins

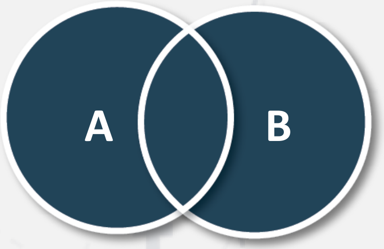
1CDC DATA PLATFORM

Foundry Pyspark Cheat Sheet

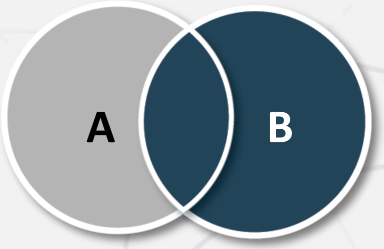
Pyspark Join Types



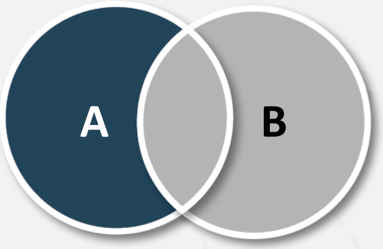
LEFT



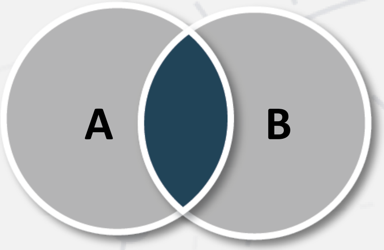
FULL



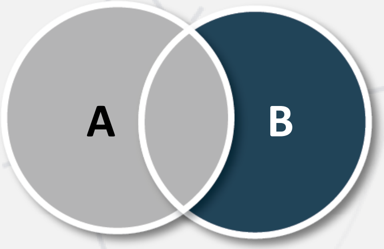
RIGHT



LEFTANTI



INNER (DEFAULT)



RIGHTANTI

? Questions? Please contact your project lead