DATA 1050 Final

PySpark

- Speed - run workloads 100x faster
- Ease of Use - write applications quickly in Java, Scala, Python, R, and SQL
- Generality - combine SQL, streaming, and complex analytics
- Runs Everywhere - runs on Hadoop, Apache Mesos, Kubemetes, standalone, or in the cloud. Access divers data sources

Install and import PySpark library

```
!pip install pyspark
import pyspark
```

Create Spark session

```
from pyspark.sql import SaprkSession
spark=SparkSession.builder.appName('optional').getOrCreate()
# appName() is optional; builder.getOrCreate() also works
```

Read data

```
df = spark.read.csv('path/xxx.csv', header=True, inferSchema=
# csv/table/text/json/...

# check schema
df.printSchema()
df.dtypes
# example output:
# root
# |-- c1_: string (nullable = true)
# |-- c2_: string (nullable = true)
```

```
df.show(truncate=False, n=df.count())
df.head()
```

```
# get column names
df.columns
# show only one column
df.select('column1').show()
# show multiple columns
df.select(['columns1','column2']).show()

# show summary of table
df.describe().show()
```

Changing columns

```
# add columns in dataframe
df_new = df.withColumn('newcol',df['col1']*2) #newcol=col1*2

# drop column
df_new = df_new.drop('newcol')

# rename columns
df.withColumnRenamed('oldName','newName')

# drop duplicate columns
df.dropDuplicates()
```

Drop null values

```
df.na.drop(how='any', thresh=None, subset=None).show()
# drop/fill/replace
# how = 'any'(default) drop a row if contains any nulls / 'al
# thresh = n at least n non-null values in a row
# subset = ['colname'] drop row if there's null value in 'col
```

Fill null values

```
# replace all missing values
df.na.fill('replace_val')
# replace missing values in certain columns
df.na.fill('replace_val',['col1','col2'])
```

Filter operation

```
# one condition
df.filter('col1 < 1050').show()
df.filter('col1 < 1050').select(['col1','col2']).show()

# >1 condition
df.filter((df['col1']<1050) & (df['col2']>1030)).show()
# & |
# inverse filtering
df.filter(~('col1 < 1050')).show() #showing everything >=1050
```

Groupby and aggregate functions

```
# groupby
df.groupBy('col1')

# groupby and aggregate functions
df.groupBy('col1').sum() #sum of every unique values in col1
# sum/mean/count/max/min/avg/...
```

Join

```
df1.join(df2, df1.key == df2.key, join_type)
# join_type = inner,outer, left, right, leftsemi, leftanti
```