PSC NCSA NSF Award 2417789
Carnegie Mellon University
University of Pittsburgh

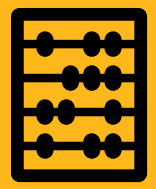
# CI Pathways

*Leading the Way to Effective Cyberinfrastructure Use*

**Your Pathway**: Apache Spark II

**Bryon Gill** – Pittsburgh Supercomputing Center

# Roadmap



**SQL**

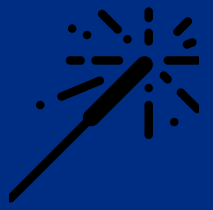
Traditional tools for data analysis

**Pandas**

A modern and flexible tool for data analysis

**Big Data**

The need for a new paradigm



**Spark**

A distributed, scalable tool for data analysis

**Spark II**

A deeper dive

# Real Real Estate Data

*Applying what we've learned*

# Real Data Three Ways

- SQL
- Pandas
- Apache Spark

# Public Datasets

Data.gov

Mission: The United States Government's open data site is designed to unleash the power of government open data to inform decisions by the public and policymakers, drive innovation and economic activity, achieve agency missions, and strengthen the foundation of an open and transparent government.

# Public Datasets, Cont'd

Western Pennsylvania Regional Data Center

(WPRDC.org)

A long-running partnership between the University of Pittsburgh, Allegheny County, and the City of Pittsburgh, the Western Pennsylvania Regional Data Center has been sharing open data from our partners and publishers for nearly 10 years. Our mission is to make local data publicly accessible, machine-readable, and free to use, modify, and share.

# The Data

Real Estate data for Allegheny County, PA

**sales.csv**

Sales Data for real estate transactions in Allegheny County, PA

https://data.wprdc.org/datastore/dump/5bbe6c55-bce6-4edb-9d04-68edeb6bf7b1

**locs.csv**

Location Data for parcels in Allegheny County, PA

https://data.wprdc.org/dataset/6bb2a968-761d-48cf-ac5b-c1fc80b4fe6a/resource/42231cab-8341-48d6-b695-47612dd6514a/download/parcelcoords.csv

# Setup

Today I encourage you to follow along.

#copy the data folder to your home dir and cd to the dir

cp -nr /projects/becs/bgill/homesales .

cd homesales

# commands listed on these slides will assume you're in this directory

# Setup (cont'd)

```
# make a new database

sqlite3 sales.db

# now we'll have an sqlite prompt:

sqlite> .mode csv

sqlite> .import sales.csv sales

sqlite> .import locs.csv locs
```

# Working in SQL

Let’s first get the data into SQL.

```
# sqlite3 sales.db

SQLite version 3.45.3 2024-04-15 13:34:05

Enter ".help" for usage hints.

sqlite> .mode csv

sqlite> .import sales.csv sales

sqlite> select * from sales limit 1;

57610312,1010G00075000000,"0 HILL ST, SOUTH PARK, PA 15129",0,"

","",HILL,ST,"","","SOUTH PARK",PA,15129,37,"South Park",945,"South Park

",2022-07-06,2022-06-21,1,18965,534,H,"MULTI-PARCEL SALE",DE,DEED
```

# Working in SQL

```
sqlite> .import locs.csv locs

sqlite> SELECT * FROM locs LIMIT 1;

0764B00296000000,-79.89129773,40.30581767
```

# Working in SQL

```
#how many records?

    SELECT COUNT(*) FROM sales;

#how many parcels do we have geodata for?

    SELECT COUNT(*) FROM locs;

#how many zip codes?

    SELECT COUNT(DISTINCT PROPERTYZIP) FROM sales;
```

# Working in SQL

#parcel with most sales?

```
SELECT PARID, COUNT(*) AS numsales FROM sales GROUP BY PARID ORDER BY numsales DESC LIMIT 1;
```

```
+----------------+---------+
|      PARID      | numsales |
+----------------+---------+
| 0431B00017000000 | 21      |
+----------------+---------+
```

# Working in SQL – Cleaning the Data

- Badly formatted values (typos, out of place headers)
- Missing Values  (addresses with no zip codes)
- Unwanted Values (property sales that are actually corrections or were for $1)
- Note: In Pandas dropna() is your friend! SQL takes a little more work.

# Pandas and Pyspark Setup

```
# make sure we're set up for pandas and spark

ssh login.delta.ncsa.illinois.edu
# connect to an interactive job
srun -A becs-delta-cpu --time=04:00:00 --partition=cpu --ntasks-per-node=8 --mem=8G --pty /bin/bash

module load anaconda3_cpu
module load spark

# assuming we've already copied the homesales dir, if not you'll get an error
cd ~/homesales
pyspark
>>> import pandas as pd
```

# Pandas Data Ingestion

```
#ingest the data again, pandas style

sales = pd.read_csv("sales.csv")

locs = pd.read_csv("locs.csv")
```

# Pandas Data Ingestion Cont'd

We can also import directly from the sqlite3 database

```
>>> import sqlite3

>>> conn = sqlite3.connect('sales.db')

>>> query = "SELECT * FROM sales"

>>> sales_from_db=pd.read_sql_query(query,conn)

>>> conn.close()
```

# Pandas Data Ingestion Cont'd

The sqlite3 table data is now in your dataframe

```
>>> sales_from_db.head()
       _id             PARID                          FULL_ADDRESS PROPERTYHOUSENUM  ... SALECODE               SALEDESC INSTRTYP    INSTRTYPDESC
0  57610312  1010G00075000000        0 HILL ST, SOUTH PARK, PA 15129               0  ...        H       MULTI-PARCEL SALE       DE            DEED
1  57610313  1075F00108000000  4720 HIGHPOINT DR, GIBSONIA, PA 15044            4720  ...        3  LOVE AND AFFECTION SALE       DE            DEED
2  57610314  0011A00237000000     0 LOMBARD ST, PITTSBURGH, PA 15219               0  ...        2      CITY TREASURER SALE       TS  TREASURER DEED
3  57610315  0011J00047000000  1903 FORBES AVE, PITTSBURGH, PA 15219            1903  ...        2      CITY TREASURER SALE       TS  TREASURER DEED
4  57610316  0011J00191000000   1806 TUSTIN ST, PITTSBURGH, PA 15219            1806  ...       GV          GOVERNMENT SALE       TS  TREASURER DEED

[5 rows x 26 columns]
```

# Pandas Review

```
>>> sales.columns
Index(['_id', 'PARID', 'FULL_ADDRESS', 'PROPERTYHOUSENUM', 'PROPERTYFRACTION',
       'PROPERTYADDRESSDIR', 'PROPERTYADDRESSSTREET', 'PROPERTYADDRESSSUF',
       'PROPERTYADDRESSUNITDESC', 'PROPERTYUNITNO', 'PROPERTYCITY',
       'PROPERTYSTATE', 'PROPERTYZIP', 'SCHOOLCODE', 'SCHOOLDESC', 'MUNICODE',
       'MUNIDESC', 'RECORDDATE', 'SALEDATE', 'PRICE', 'DEEDBOOK', 'DEEDPAGE',
       'SALECODE', 'SALEDESC', 'INSTRTYP', 'INSTRTYPDESC'],
      dtype='object')
>>> locs.columns
Index(['PIN', 'x', 'y'], dtype='object')
```

A url for the cheatsheet from the previous slides if you want it:

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

# Pandas Examples

```
# filter some sales

sales[sales['PROPERTYZIP']==15213]

# let’s filter by date to get sales before 2020

# convert the column to the pandas datetime type so we can compare it

sales['SALEDATE']=pd.to_datetime(sales['SALEDATE’])

# let's make a name for the first day of 2020

twenty = pd.to_datetime('2020-01-01')

# get a list of sales before that date

sales[sales['SALEDATE'] < twenty ]
```

# Merging with Pandas

```
#merge (note the left_on/right_on see docs; could edit csv headers instead if we wanted)
# https://pandas.pydata.org/docs/reference/api/pandas.merge.html
>>> both = pd.merge(sales,locs, how='left', left_on='PARID', right_on='PIN')
>>> both.columns
Index(['_id', 'PARID', 'FULL_ADDRESS', 'PROPERTYHOUSENUM', 'PROPERTYFRACTION',
       'PROPERTYADDRESSDIR', 'PROPERTYADDRESSSTREET', 'PROPERTYADDRESSSUF',
       'PROPERTYADDRESSUNITDESC', 'PROPERTYUNITNO', 'PROPERTYCITY',
       'PROPERTYSTATE', 'PROPERTYZIP', 'SCHOOLCODE', 'SCHOOLDESC', 'MUNICODE',
       'MUNIDESC', 'RECORDDATE', 'SALEDATE', 'PRICE', 'DEEDBOOK', 'DEEDPAGE',
       'SALECODE', 'SALEDESC', 'INSTRTYP', 'INSTRTYPDESC', 'PIN', 'x', 'y'],
      dtype='object')
```

# Output

#No need to follow along, but if you want to make a csv of the data to use in say Excel you can do this:

both.to_csv('both.csv', index=False)

# Convert to RDD

We will just take two columns, and use a workaround for a dataframes compatibility issue with Spark:

```
>>> longlats = both[['x','y']].dropna()

>>> longlats.iteritems=longlats.items

>>> lldf = spark.createDataFrame(longlats)

>>> llrdd = lldf.rdd

>>> llrdd.take(4)

2025-04-23 12:13:34,074 WARN scheduler.TaskSetManager: Stage 2245 contains a task of very large size (8686 KiB). The maximum recommended task size is 1000 KiB.

[Row(x=-80.02269987, y=40.28454008), Row(x=-79.97604172, y=40.59502985), Row(x=-79.97974311, y=40.44135999), Row(x=-79.97930308, y=40.43758897)]
```

# Creating a pairRDD

```
>>> pairll = llrdd.map(lambda x: [float(x[0]),float(x[1])])

>>> pairll.take(4)

2025-04-23 12:21:00,555 WARN scheduler.TaskSetManager: Stage 2247 contains a task of very large size (8686 KiB). The maximum
    recommended task size is1000 KiB.

[[-80.02269987, 40.28454008], [-79.97604172, 40.59502985], [-79.97974311, 40.44135999], [-79.97930308, 40.43758897]]

>>> pairll = llrdd.map(lambda x: [float(x[0]),float(x[1])])
```

# Finding Some Clusters

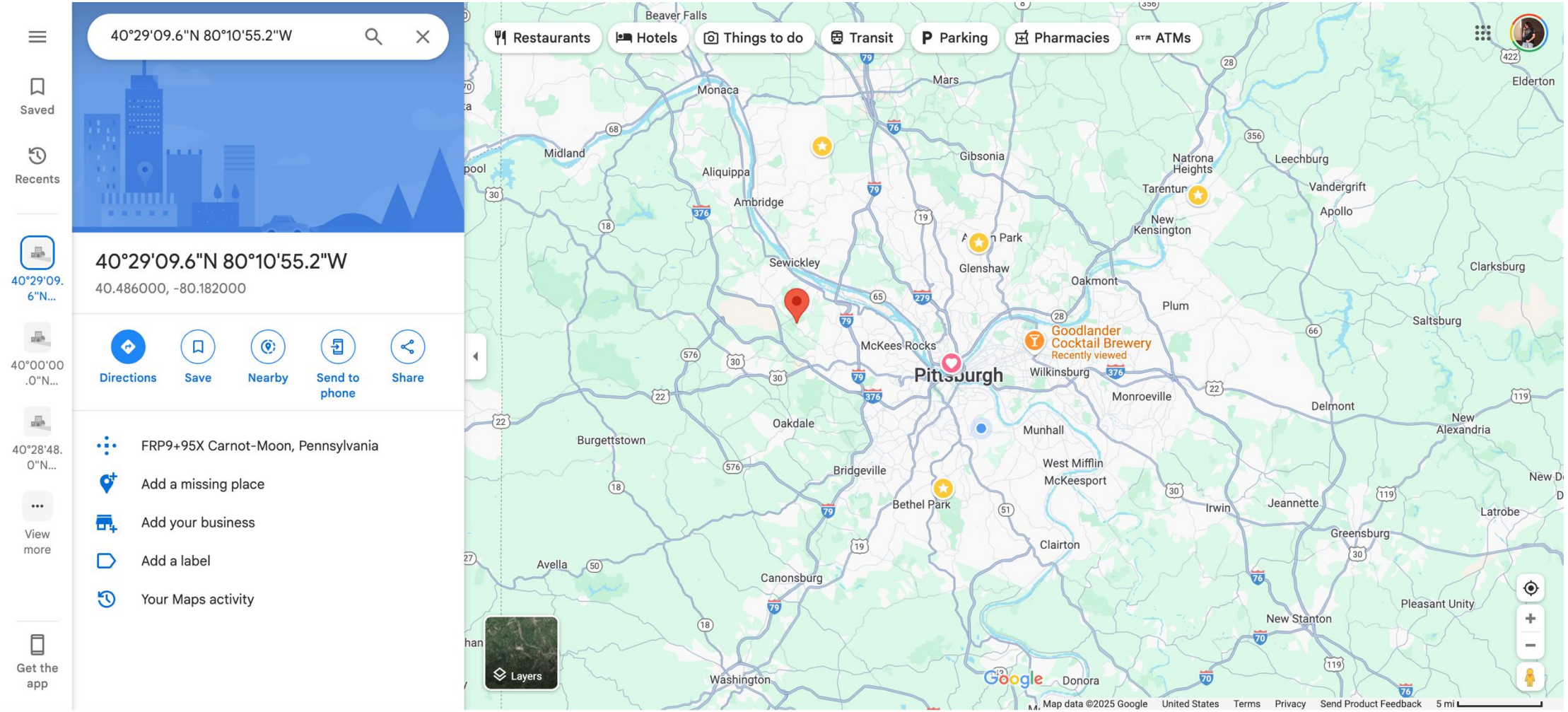
```
# We'll just find 5 clusters.

model = KMeans.train(pairll, 5)

cost = model.computeCost(pairll)

centers = model.clusterCenters            #Let's grab cluster centers

print (clusters, cost)

for coords in centers:

 print (float(coords[0]), float(coords[1])) #note indentation!
```

# Output

Note: put the y coordinate (latitude) first on google maps

```
>>> for coords in centers:
...  print (float(coords[0]), float(coords[1]))
...
-80.18213834252963 40.486391153664904
-80.0290661467611 40.38859189902571
-79.79222349552916 40.52050743009944
-79.87817359722749 40.39551867102404
-80.00539659644558 40.53580475954974
>>>
```

# A coordinate on the map



# Week 3 Assignment

1. Clean the data: Remove all sales that are for less than $2 and any sales that are deemed deed corrections. Also remove any rows that are missing price, longitude, latitude, or zip code data. Your other answers should use this cleaned dataset.

2. Convert the latitude and longitude coordinates of all sold parcels in our cleaned dataset into a Spark pairRDD. Find 5 clusters of these locations (don't worry about calculating error for now, this is just practice using the k-means algorithm).

3. EXTRA CREDIT: This problem will require some additional legwork. Figure out what zip code the centroids of those clusters is in and compare the average sale prices in those zip codes to the average sale price in our whole cleaned data set.