CS 365 FINAL PROJECT: TRANSIT REGINA DATA WRANGLING

JANICE COTCHER

DECEMBER 5, 2025

Data Source: City of Regina Open Data Portal

Dataset: Transit Stops and Routes (November 20, 2025)

DATASET SOURCE & LICENSE

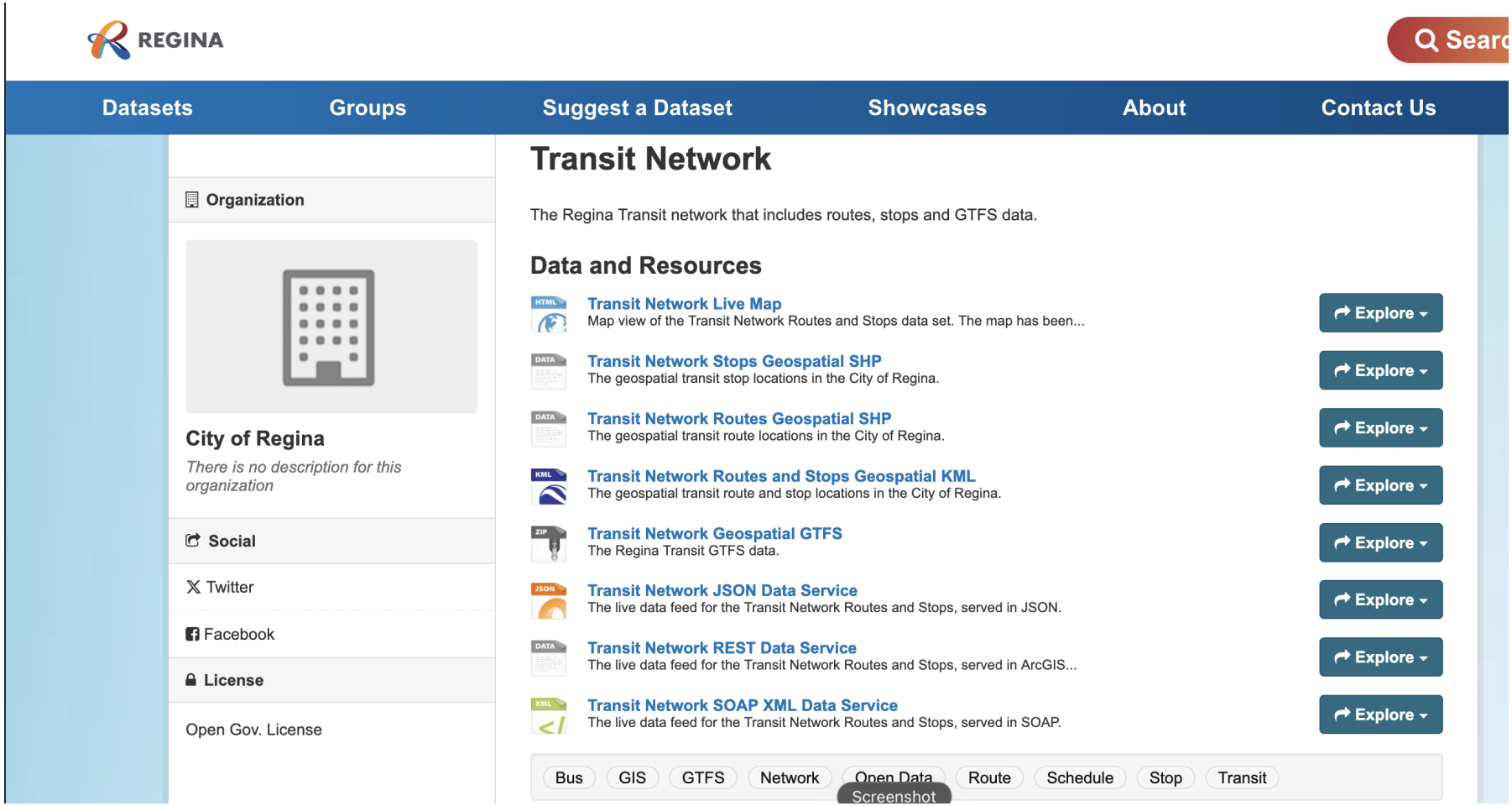
Source: City of Regina Open Data Portal

- Bus Stop Locations (yqrStops.json)
- Transit Routes (yqrRoutes.json)
- General Transit Feed Specification(routes.txt, stops.txt, trips.txt, stop_times.txt)

License: Open Government License - Regina

- Allows educational and commercial use
- No Personally Identifiable Information - only public infrastructure data

Why this matters: Understanding transit accessibility and route coverage for urban planning


REGINA
Search
Datasets
Groups
Suggest a Dataset
Showcases
About
Contact Us
Organization
City of Regina
There is no description for this organization
Social
Twitter
Facebook
License
Open Gov. License
Transit Network
The Regina Transit network that includes routes, stops and GTFS data.
Data and Resources
Transit Network Live Map
Map view of the Transit Network Routes and Stops data set. The map has been...
Explore
Transit Network Stops Geospatial SHP
The geospatial transit stop locations in the City of Regina.
Explore
Transit Network Routes Geospatial SHP
The geospatial transit route locations in the City of Regina.
Explore
Transit Network Routes and Stops Geospatial KML
The geospatial transit route and stop locations in the City of Regina.
Explore
Transit Network Geospatial GTFS
The Regina Transit GTFS data.
Explore
Transit Network JSON Data Service
The live data feed for the Transit Network Routes and Stops, served in JSON.
Explore
Transit Network REST Data Service
The live data feed for the Transit Network Routes and Stops, served in ArcGIS...
Explore
Transit Network SOAP XML Data Service
The live data feed for the Transit Network Routes and Stops, served in SOAP.
Explore
Bus
GIS
GTFS
Network
Open Data
Route
Schedule
Stop
Transit
Screenshot

TOOLS

VS CODE EXTENSIONS:

- Python
- Python Environments
- Data Wrangler
- Jupyter

RATIONALE

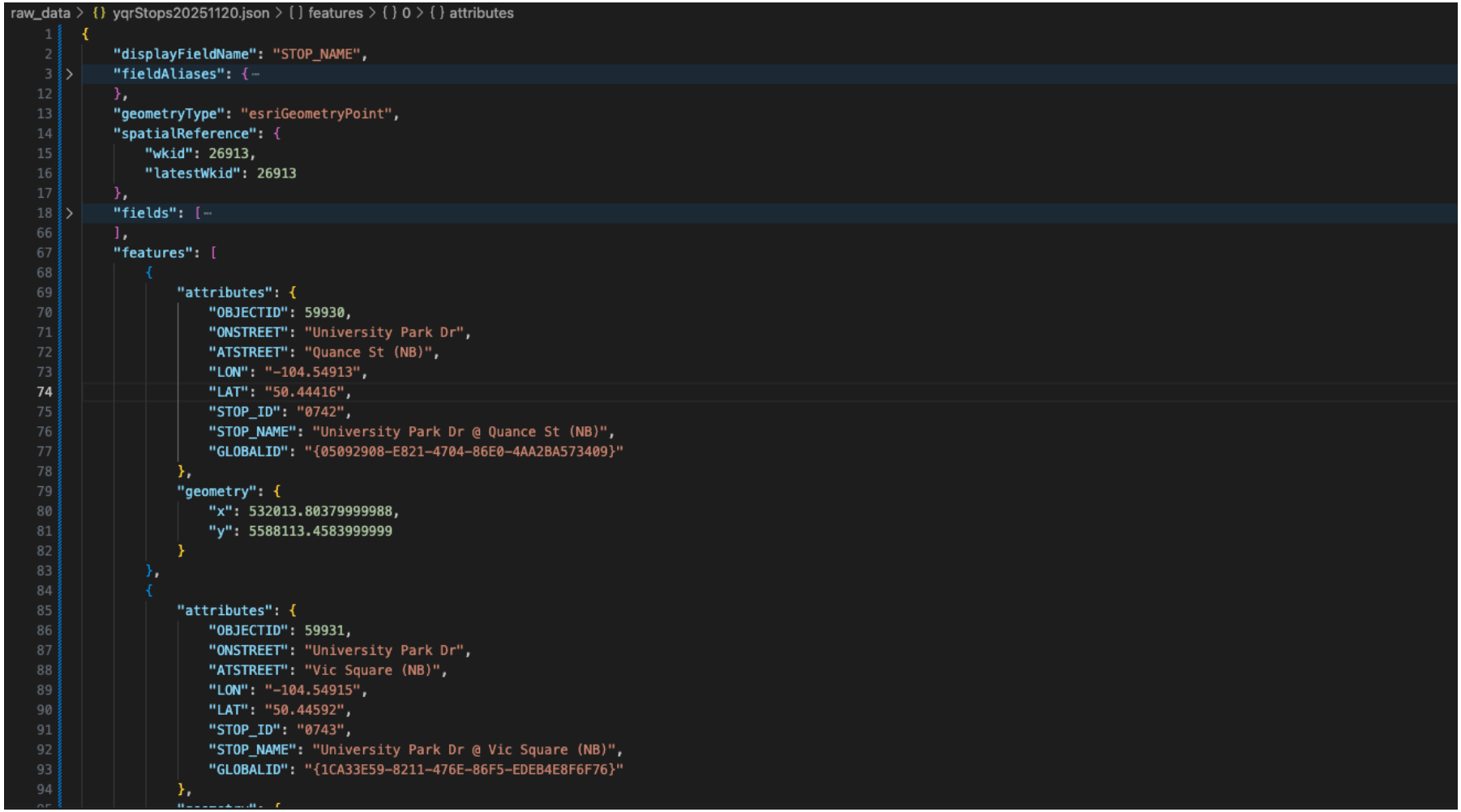
- large dataset[1]
- flexibility, memory management
- macOS
- personal expertise

[1] CS 365 - Data Cleaning: Concepts & Algorithms, Lecture 6: September 15, 2025

IMPORTED PYTHON LIBRARIES

```
import numpy as np # scientific computing for large, multi-dimensional arrays
from matplotlib import pyplot as plt # creates static, animated, and interactive
visualizations
import pandas as pd #creates 2D, size-mutable, heterogeneous tables called data
frames
import json # read and write json files
import plotly.graph_objects as go # interactive graphics like maps
from pyproj import Transformer # cartography and coordinate transformations
```

RAW DATA SNAPSHOT: BUS STOPS



```
# Load JSON data
try:
    with open('raw_data/yqrStops.json', 'r') as f:
        stop_data = json.load(f)
except json.decoder.JSONDecodeError as e:
    print("Invalid JSON", e)

# Normalize nested JSON structure
df_stops = pd.json_normalize(stop_data['features'])
```

DATA PROFILING: QUALITY ASSESSMENT - STOPS

- **Data types** - Are coordinates stored correctly?
- **Missing values** - Which columns have gaps?
- **Duplicates** - Any duplicate stop IDs?
- **Outliers** - Any stops in unexpected locations?
- **Cardinalities** - How many unique stops?

STOP DATA TYPES

```
attributes.OBJECTID       int64
attributes.ONSTREET      object
attributes.ATSTREET      object
attributes.LON           object
attributes.LAT           object
attributes.STOP_ID       object
attributes.STOP_NAME     object
attributes.GLOBALID      object
geometry.x              float64
geometry.y              float64
dtype: object
```

MISSING VALUES

```
Number of Missing Values: 1
Missing Values
attributes.OBJECTID     0
attributes.ONSTREET     0
attributes.ATSTREET     1
attributes.LON          0
attributes.LAT          0
attributes.STOP_ID      0
attributes.STOP_NAME    0
attributes.GLOBALID     0
geometry.x              0
geometry.y              0
dtype: int64
```

DUPLICATES

Number of Duplicate Stops - checked by stop ID: 0

CLEANING: BUS STOP TEXT STANDARDIZATION & TYPE CONVERSIONS

PROBLEMS IDENTIFIED

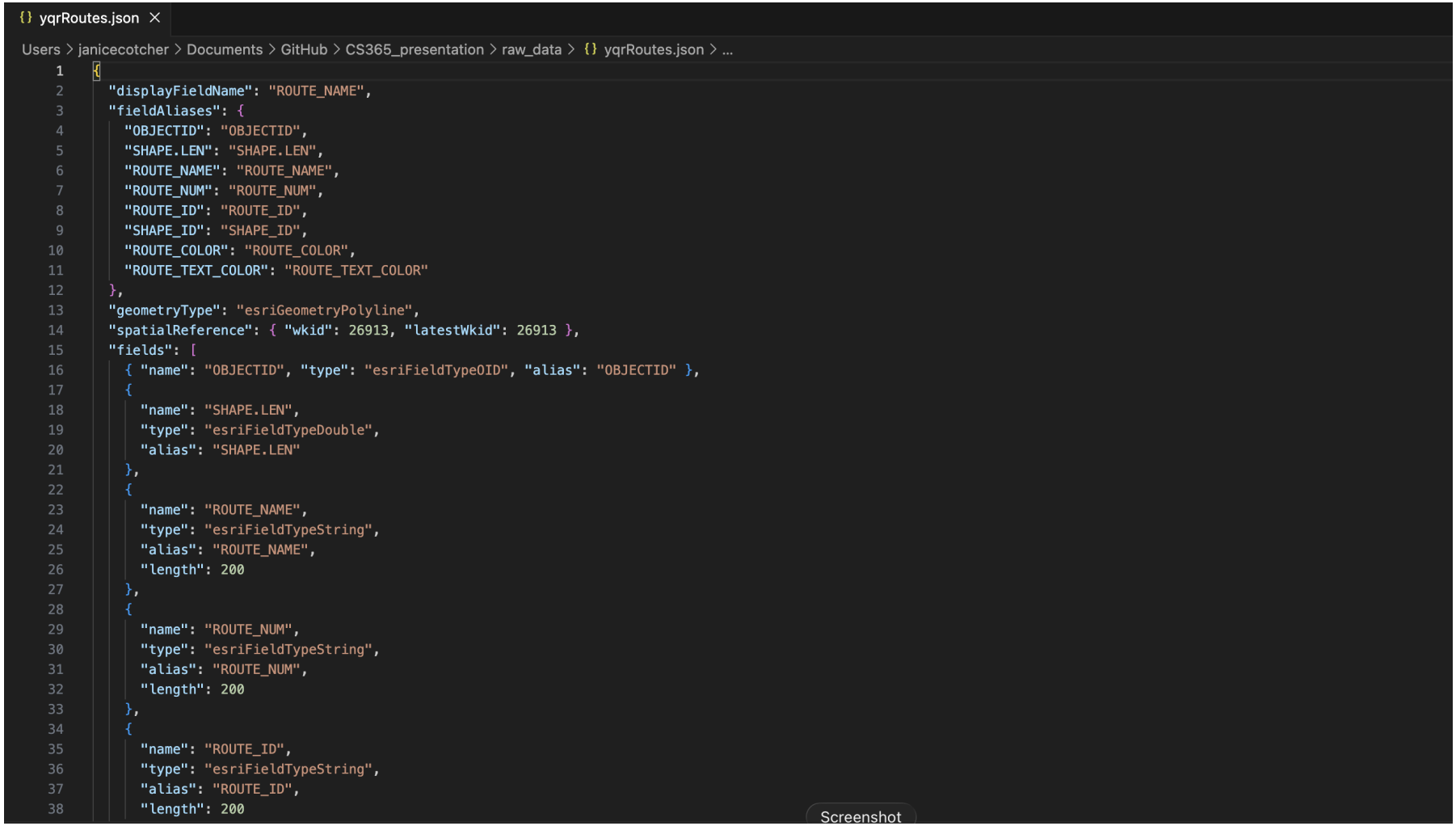
- Inconsistent text formatting (mixed case, whitespace)
- Coordinates stored as strings instead of numeric latitude and longitude
- Missing values in street names

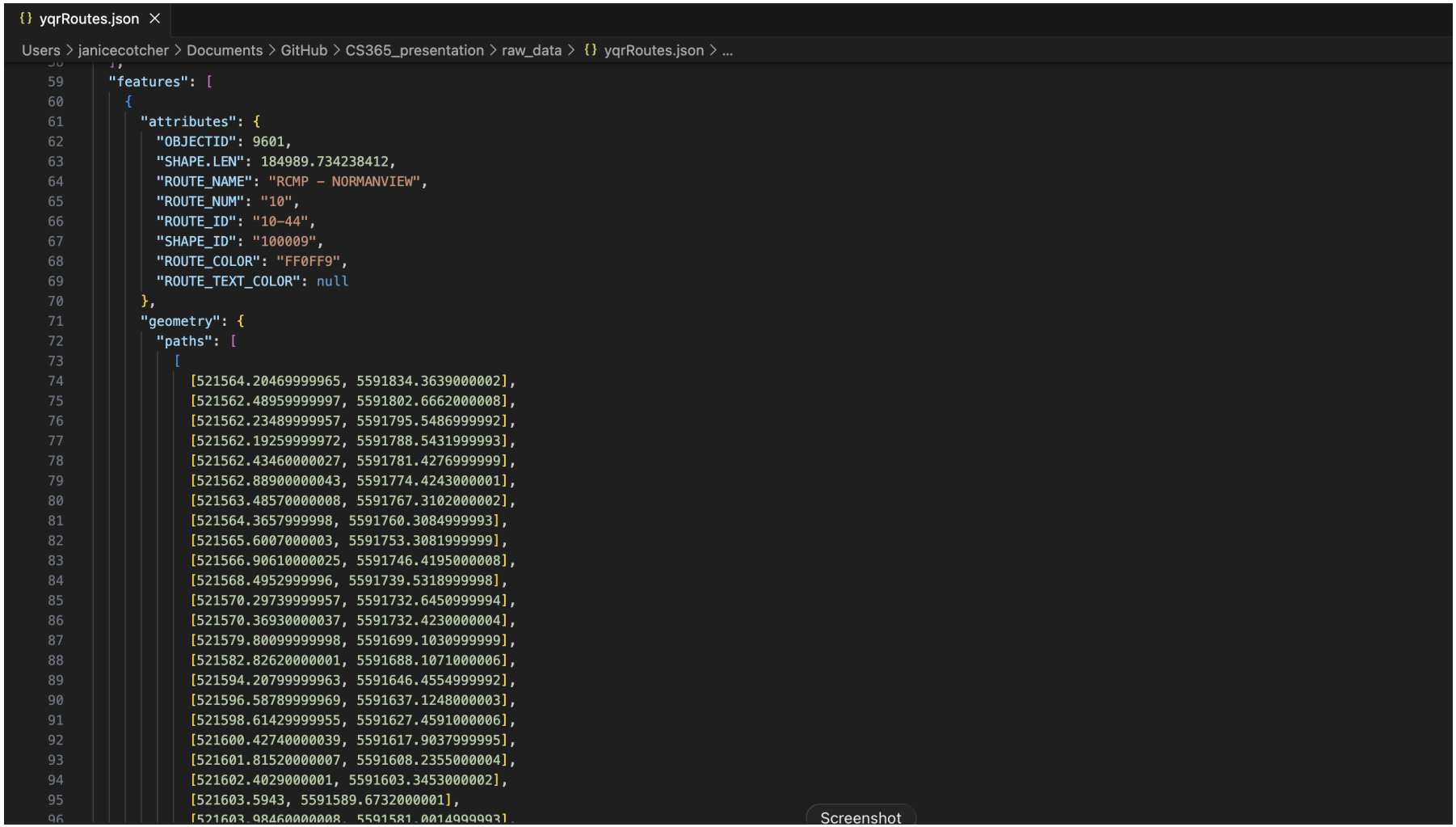
Sample Solutions:

```
# Text standardization
df_stops['attributes.ONSTREET'] = df_stops['attributes.ONSTREET'].str.strip().str.
upper()
df_stops['attributes.ATSTREET'] = df_stops['attributes.ATSTREET'].str.strip().str.
upper()
...
# Missing value imputation
df_stops = df_stops.fillna({'attributes.ATSTREET': "DOROTHY ST (SB)"})

# Data correction
df_stops['attributes.ONSTREET'] = df_stops['attributes.ONSTREET'].str.replace(
    "1060 DOROTHY ST (SB)", "DOROTHY ST", regex=False
)
```

RAW DATA SNAPSHOT: BUS ROUTES




yqrRoutes.json
Users > janicecotcher > Documents > GitHub > CS365_presentation > raw_data > yqrRoutes.json > ...
"features": [
{
"attributes": {
"OBJECTID": 9601,
"SHAPE.LEN": 184989.734238412,
"ROUTE_NAME": "RCMP – NORMANVIEW",
"ROUTE_NUM": "10",
"ROUTE_ID": "10-44",
"SHAPE_ID": "100009",
"ROUTE_COLOR": "FF0FF9",
"ROUTE_TEXT_COLOR": null
},
"geometry": {
"paths": [
[
[521564.20469999965, 5591834.3639000002],
[521562.48959999997, 5591802.6662000008],
[521562.23489999957, 5591795.5486999992],
[521562.19259999972, 5591788.5431999993],
[521562.43460000027, 5591781.4276999999],
[521562.88900000043, 5591774.4243000001],
[521563.48570000008, 5591767.3102000002],
[521564.3657999998, 5591760.3084999993],
[521565.6007000003, 5591753.3081999999],
[521566.90610000025, 5591746.4195000008],
[521568.4952999996, 5591739.5318999998],
[521570.29739999957, 5591732.6450999994],
[521570.36930000037, 5591732.4230000004],
[521579.80099999998, 5591699.1030999999],
[521582.82620000001, 5591688.1071000006],
[521594.20799999963, 5591646.4554999992],
[521596.58789999969, 5591637.1248000003],
[521598.61429999955, 5591627.4591000006],
[521600.42740000039, 5591617.9037999995],
[521601.81520000007, 5591608.2355000004],
[521602.4029000001, 5591603.3453000002],
[521603.5943, 5591589.6732000001],
Screenshot

OVER 90K LINES, ~4.9MB

DATA PROFILE: QUALITY ASSESSMENT - ROUTES

- **Data types** - Are coordinates stored correctly?
- **Missing values** - Which columns have gaps?
- **Duplicates** - Any duplicate route IDs?
- **Outliers** - Any routes in unexpected locations?
- **Cardinalities** - How many unique routes?

CLEANING: BUS ROUTE TEXT STANDARDIZATION & TYPE CONVERSIONS

DATA WRANGLER DEMO

BUS ROUTE CLEANING SUMMARY

- inconsistent text formatting
- missing colours
- hex values missing #
- standardize column names

Cleaned routes: 22

	route_num	route_name	route_color
0	10	RCMP - NORMANVIEW	#FF0FF9
1	18	HARBOUR LANDING - UNIVERSITY	#80FF00
2	3	UNIVERSITY - SHERWOOD ESTATES	#A8A800
3	40	ALBERT S EXPRESS - ALBERT N EXPRESS	#00CECE
4	60	ARCOLA E EXP - ARCOLA DWTN EXP	#808000

PREPARE ROUTES FOR VISUALIZATION

```
# Create transformer to convert from UTM to lat/lon
transformer = Transformer.from_crs("EPSG:26913", "EPSG:4326", always_xy=True)
```

VISUAL CHECK OF A BUS ROUTE

TRANSFORMATION STEP 1: LOADING GTFS SCHEDULE DATA

GTFS (General Transit Feed Specification) provides detailed schedule information:

```
# Load GTFS files
stops_gtfs = pd.read_csv('raw_data/gtfs_data/stops.txt')
routes_gtfs = pd.read_csv('raw_data/gtfs_data/routes.txt')
trips_gtfs = pd.read_csv('raw_data/gtfs_data/trips.txt')
times_gtfs = pd.read_csv('raw_data/gtfs_data/stop_times.txt')

# Clean and standardize - sample
stops_gtfs['stop_name'] = stops_gtfs['stop_name'].str.upper().str.strip()
routes_gtfs['route_long_name'] = routes_gtfs['route_long_name'].str.upper().str.st
rip()
```

GTFS Data Loaded:

- 1400 stops
- 44 routes
- 5078 trips
- 285722 stop times

TRANSFORMATION STEP 2: PARSING DATE/TIME DATA

Problem: Time data stored as strings (HH:MM:SS)

Solution: Parse to datetime and derive time-based features

```
# Parse time columns
times_gtfs_clean['departure_datetime'] = pd.to_datetime(
    times_gtfs_clean['departure_time'],
    format='%H:%M:%S',
    errors='coerce'
)

times_gtfs_clean['departure_datetime'] = pd.to_datetime(
    times_gtfs_clean['departure_time'],
    format='%H:%M:%S',
    errors='coerce'
)

# Derive hour and minute features
times_gtfs_clean['arrival_hour'] = times_gtfs_clean['arrival_datetime'].dt.hour
times_gtfs_clean['arrival_minute'] = times_gtfs_clean['arrival_datetime'].dt.minut
e
times_gtfs_clean['departure_hour'] = times_gtfs_clean['departure_datetime'].dt.hou
r
times_gtfs_clean['departure_minute'] = times_gtfs_clean['departure_datetime'].dt.m
inute
```

Parsed time data and derived hour/minute features

	arrival_time	arrival_datetime	arrival_hour	arrival_minute	departure_datetime	departure
0	06:10:00	1900-01-01 06:10:00	6	10	1900-01-01 06:10:00	06
1	06:11:00	1900-01-01 06:11:00	6	11	1900-01-01 06:11:00	06
2	06:12:00	1900-01-01 06:12:00	6	12	1900-01-01 06:12:00	06
3	06:13:00	1900-01-01 06:13:00	6	13	1900-01-01 06:13:00	06
4	06:14:00	1900-01-01 06:14:00	6	14	1900-01-01 06:14:00	06
5	06:15:00	1900-01-01 06:15:00	6	15	1900-01-01 06:15:00	06
6	06:15:00	1900-01-01 06:15:00	6	15	1900-01-01 06:15:00	06
7	06:16:00	1900-01-01 06:16:00	6	16	1900-01-01 06:16:00	06
8	06:17:00	1900-01-01 06:17:00	6	17	1900-01-01 06:17:00	06
9	06:18:00	1900-01-01 06:18:00	6	18	1900-01-01 06:18:00	06

TRANSFORMATION STEP 3: MERGE/JOIN OPERATIONS

- 1400 stops
- Open Regina ASP.NET limit of 1000
- GTFS data contains some fields
- Imputation for the remaining

```
# Find stops in GTFS but not in geographic JSON data
missing_stops = stops_gtfs_clean[
    ~stops_gtfs_clean['stop_id'].isin(clean_stops['stop_id'])
]

# Merge datasets
merged_stops = pd.concat([clean_stops, missing_stops], ignore_index=True, sort=Fal
se)

# Impute missing street names from stop_name
for index, stop in merged_stops.iterrows():
    if pd.isna(stop['on_street']):
        merged_stops.at[index, 'on_street'] = stop['stop_name'].split(' @')[0]
    if pd.isna(stop['at_street']):
        merged_stops.at[index, 'at_street'] = stop['stop_name'].split('@ ')[-1]

1224 new stops added
```

Found stops in GTFS data:*1224

Total stops after merge: 2224 (1224 added)

	object_id	on_street	at_street	lon	lat	stop_id	stop_name	g
0	59930.0	UNIVERSITY PARK DR	QUANCE ST (NB)	-104.54913	50.44416	0742	UNIVERSITY PARK DR @ QUANCE ST (NB)	{05 E8 4AA2BA
1	59931.0	UNIVERSITY PARK DR	VIC SQUARE (NB)	-104.54915	50.44592	0743	UNIVERSITY PARK DR @ VIC SQUARE (NB)	{1C 82 EDEB4I
2	59932.0	FLEET ST	NORTH SERVICE RD (NB)	-104.549126	50.448609	0744	FLEET ST @ NORTH SERVICE RD (NB)	{AF 583 3DD54
3	59933.0	FLEET ST	FINES DR (NB)	-104.549111	50.449614	0745	FLEET ST @ FINES DR (NB)	{63 70 6FC74F
4	59934.0	CAMBRIDGE AVE	MILFORD CRES (WB)	-104.5513	50.45041	0746	CAMBRIDGE AVE @ MILFORD CRES (WB)	{E 6F 4A928E

TRANSFORMATION: FEATURE DERIVATION - GEOGRAPHIC REGIONS

Divide the city into four quadrants based on approximate city centre coordinates (Albert St & Victoria Ave Intersection)

TRANSFORMATION: FEATURE DERIVATION - GEOGRAPHIC REGIONS

Divide the city into four quadrants based on approximate city centre coordinates (Albert St & Victoria Ave Intersection)

```
city_centre_lon = -104.618
city_centre_lat = 50.447
merged_stops_clean['region'] = ''

# Assign quadrants (NE, NW, SE, SW)
for stop in range(len(merged_stops_clean)):
    if float(merged_stops_clean['lat'].iloc[stop]) > city_centre_lat:
        if float(merged_stops_clean['lon'].iloc[stop]) > city_centre_lon:
            merged_stops_clean.at[stop, 'region'] = "NE"
        else:
            merged_stops_clean.at[stop, 'region'] = "NW"
    else:
        if float(merged_stops_clean['lon'].iloc[stop]) > city_centre_lon:
            merged_stops_clean.at[stop, 'region'] = "SE"
        else:
            merged_stops_clean.at[stop, 'region'] = "SW"
```

Derived regional classifications for all stops

Stop distribution by region:

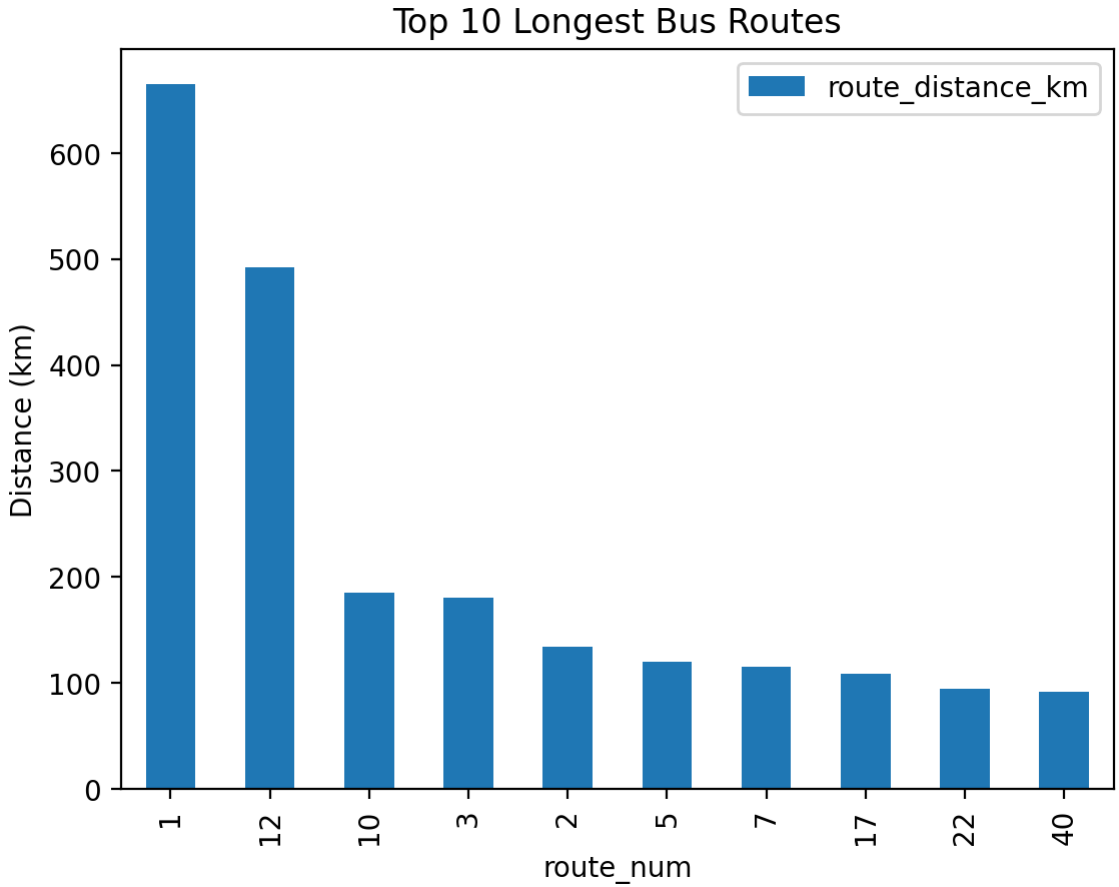
- NW: 333 stops
- NE: 224 stops
- SW: 1403 stops
- SE: 264 stops

TRANSFORMATION 5: FEATURE DERIVATION - DISTANCE CALCULATIONS

- In ArcGIS/GIS systems, shape.len (shape length) represents the total length of the geometry in metres

```
# convert shape_length into km
clean_routes['route_distance_km'] = clean_routes['shape_length'] / 1000
```

	route_num	route_distance_km
0	10	184.989734
1	18	47.401586
2	3	180.172680
3	40	92.096531
4	60	57.978962
5	7	115.065916
6	1	664.900170
7	12	492.246799
8	24	24.622480
9	22	94.968615
10	2	134.424603
11	30	63.697887
12	4	85.020879
13	16	47.980064
14	17	108.643699
15	15	60.200280
16	9	90.660507
17	6	64.226528
18	8	89.851981
19	5	119.587762
20	21	68.741318

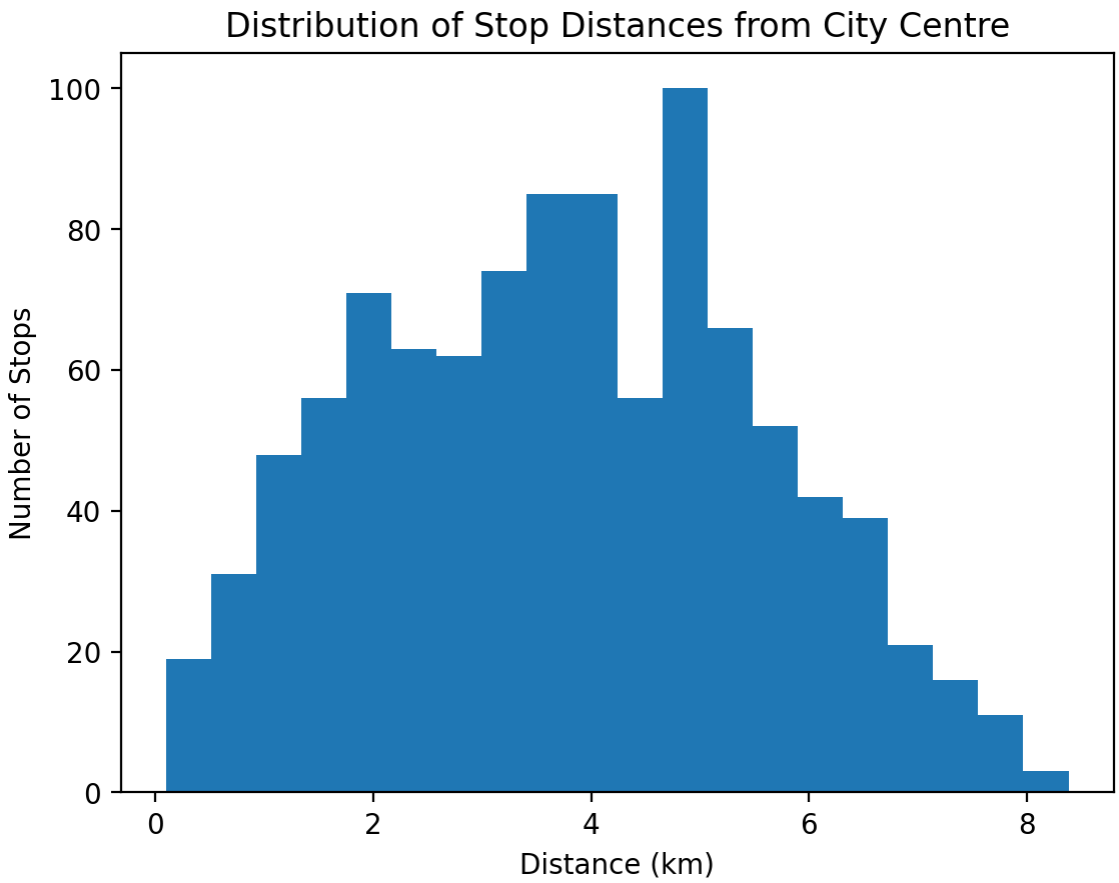

Top 10 Longest Bus Routes
route_distance_km
Distance (km)
0
100
200
300
400
500
600
1
12
10
3
2
5
7
17
22
40
route_num

Calculate distance from city centre using coordinate geometry:

```
# Approximate conversion: ~111 km per degree latitude, ~85 km per degree longitude
 at this latitude
# Euclidean distance
merged_stops_clean['distance_from_centre_km'] = np.sqrt(
    ((merged_stops_clean['lat'].astype(float) - city_centre_lat) * 111)**2 +
    ((merged_stops_clean['lon'].astype(float) - city_centre_lon) * 85)**2
)
```

```
Distance statistics (km):
count    1000.000000
mean        3.775980
std         1.791905
min         0.098964
25%         2.319224
50%         3.785285
75%         5.068146
max         8.381130
Name: distance_from_centre_km, dtype: float64
```

Calculated distance from city centre for all stops


Distribution of Stop Distances from City Centre
Number of Stops
Distance (km)
0
20
40
60
80
100
0
2
4
6
8

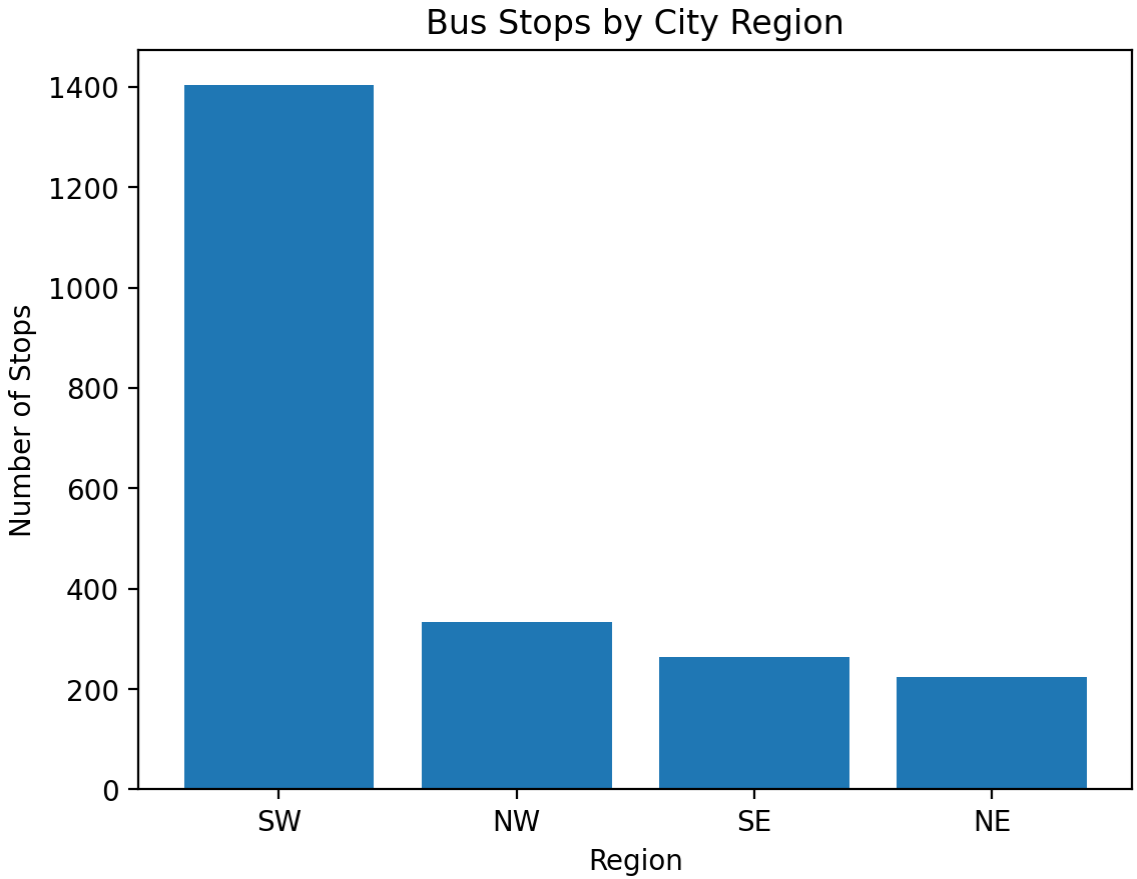
TRANSFORMATION 6: AGGREGATION

Aggregate stops by region to understand service distribution:

```
region_summary = clean_stops.groupby('region').agg({
    'stop_id': 'count',
    'distance_from_centre_km': ['mean', 'max']
}).reset_index()
```

	stop_id	distance_from_centre_km		
	count	mean	min	max
region				
NE	224	4.113674	0.132092	7.650399
NW	333	3.994900	0.306790	8.381130
SE	264	3.926796	0.286489	7.536191
SW	1403	2.723696	0.098964	5.336768

Regional Summary Statistics:


Bus Stops by City Region
Number of Stops
0
200
400
600
800
1000
1200
1400
SW
NW
SE
NE
Region

TRANSFORMATION 7: RESHAPE (PIVOT)

Pivot analysis: How many stops does each route serve in each region?

```
# Join stop times → trips → stops to get route-region relationships
route_stops = (
    times_gtfs_clean
    .merge(trips_gtfs_clean[['trip_id', 'route_id']], on='trip_id')
    .merge(merged_stops_clean[['stop_id', 'region']], left_on='stop_id', right_on=
'stop_id')
    .groupby(['route_id', 'region'])
    .size()
    .reset_index(name='stop_count')
)

# Pivot to wide format
route_region_pivot = route_stops.pivot(
    index='route_id',
    columns='region',
    values='stop_count'
).fillna(0)
```

Routes by Region (Pivoted):

region	NE	NW	SE	SW
route_id				
1-44	0.0	0.0	0.0	11105.0
1-45	0.0	0.0	0.0	10997.0
10-44	222.0	1966.0	0.0	6468.0
10-45	210.0	1862.0	0.0	6124.0
12-44	0.0	1400.0	1536.0	8978.0
12-45	0.0	1400.0	1536.0	8978.0
15-44	28.0	0.0	316.0	766.0
15-45	28.0	0.0	316.0	766.0
16-44	0.0	56.0	0.0	388.0
16-45	0.0	56.0	0.0	388.0

PROFILING: OUTLIERS & CARDINALITIES

```
Found 0 outlier stops
```

Outlier Detection: Identify stops unusually far from city centre

Stops beyond 15km from city centre:

```
Unique stops: 2224
Unique routes: 44
Unique regions: 4
Unique trips: 5078
```

Cardinality Analysis: Count unique values in key dimensions

BEFORE/AFTER EVIDENCE

QUANTITATIVE COMPARISON OF DATA QUALITY IMPROVEMENTS:

```
Stop rows: 1000
Route rows: 22
Missing ATSTREET: 1
Coordinate type: object (string)
Stop ID type: object (string)
Features: 8 columns
Stop rows: 2224 (+1224 from GTFS)
Route rows: 22
Missing ATSTREET: 0
Coordinate type: object (numeric-ready)
Stop ID type: object (int32)
Features: 20 columns
New derived features: region, route_distance_km, distance_from_centre_km
Parsed time features: arrival_hour, arrival_minute, departure_hour, depart
ure_minute
```

BEFORE (RAW DATA)

AFTER (CLEANED & TRANSFORMED)

SUMMARY OF TRANSFORMATIONS

COMPLETED TRANSFORMATIONS (8 OPERATIONS ACROSS 5 CATEGORIES):

1. Type Fixes & Parsing

- Converted stop_id from string to int32
- Parsed arrival/departure times to datetime
- Derived hour and minute features

2. Text Cleanup

- Stripped whitespace from all text columns
- Converted to uppercase for consistency
- Fixed malformed addresses

3. Missing Data Handling

- Imputed missing ATSTREET values
- Generated street names from stop names for GTFS stops

4. Join/Merge

- Merged geographic stops with GTFS schedule data
- Joined stop times → trips → routes → stops

5. Feature Derivation

REPRODUCIBILITY

HOW TO REPRODUCE THIS ANALYSIS:

1. Install dependencies:

```
pip install pandas plotly pyproj numpy jupyter
```

2. Directory structure:

```
project/
├── presentation.ipynb
├── transit_data.ipynb
├── raw_data/
│   ├── yqrStops.json
│   ├── yqrRoutes.json
│   └── gtfs_data/
│       ├── stops.txt
│       ├── routes.txt
│       ├── trips.txt
│       └── stop_times.txt
└── README.md
```

3. Run notebook:

- Command line: jupyter notebook transit_data.ipynb
- In VS Code: Execute all cells sequentially
- Or run with Mercury: `mercury run presentation.ipynb`