Final Project Grading Rubric

The project is out of 100 points, with an additional 15 points possible for extra credit.

No late submissions will be accepted.

Git (20%)

There are NO EXCEPTIONS here!
You must use git to collaborate across each group member's computer. If your group uses one computer or there's only one person committing code, *your group will lose up to 14 points.*

Max Points	***What***
5	More than one commit per person in the group.
9	Roughly equal bytes/lines of code changes per person in the group.
1	GitHub, GitLab, or BitBucket URL submitted
5	The necessary files exist in the repo: ● Project files: ○ A single .ipynb file containing the final project (named anything) ○ A single schema.sql file ○ Other .sql files for queries ● README.md ● requirements.txt ● .gitignore Points will be subtracted if: ● Unnecessary files in repo (JSON & GeoJSON files, db files, pngs, jpgs, etc)

Total max points: 20

Code Style (10%)

This section tends to be an area where groups lose a lot of points.

Max Points	***What***
3	Code in notebook follows PEP8
2	Variable, function, method, class names are meaningful e.g. no single-character variable names
3	Every function, method, and class has a docstring, and follows PEP257 for docstring format.
2	Docstrings contain docs on arguments and returns (types for arguments & returns are optional).

Total max points: 10

Meets Project Criteria (70%)

Max Points	***What***
	General
3	**Communication & Understanding** Prose written in every part that tells the reader what's going on, why they made the choices they did, why they chose certain graph types for their visualizations (where relevant), etc. Basically, can a non-technical person sort of understand what's going on without needing to read the code. This can be in Markdown cells, code comments, or docstrings (or any combination of such).
4	**Notebook Execution & Cleanliness** Each cell can execute and does not error out. All cells and lines of code are used & relevant (as in, there are no cells present that are obviously draft code). The notebook is broken down into functions and/or classes for individual tasks. All 3rd-party dependencies/packages are declared in requirements.txt. Points will be subtracted if: ● code is not broken down into functions or classes ● missing dependencies (do graders have to pip install anything other than the requirements file?) ● any code is commented out

	● if there is code that is not used at all (e.g. draft cells, uncalled functions, etc) ● any cells error out when executing the cells one after another ● any commands that start with `!` (e.g. !psql …) *Note: groups may execute the commands they need (e.g. !psql …) within their notebooks, but the commands must be removed before submitting the project.*
2	**Descript README file with required information** The repo has a README file that contains the project's description, the group number, and the UNIs for each member of the group. Should be in one of the following formats/extensions: .md, .rst, or .txt
Max points: 9	
	Part 1: Data Preprocessing
2	**Programmatically download needed data** Uses requests or another library to download 311 complaint & tree data. An application token is used for downloading. Data is loaded into a pandas or geopandas dataframe.
2	**Cleaning: Remove unnecessary columns** Removes unnecessary columns for all datasets. Groups are able to choose what they consider unneeded columns, but they must remove columns they don't end up needing for part 3 & 4. Filtering columns may be done either during the initial request for data (i.e. via query parameters), or after downloading. Points will be subtracted if: ● columns exist that aren't used/needed
2	**Cleaning: Remove invalid data** Drops invalid data points, e.g. removing duplicate zip codes loaded from the zip code shape file; removing data points outside of NYC; etc. Groups are able to choose what they consider invalid data (beyond the aforementioned examples), but some sort of "cleaning" of the data must happen.
2	**Cleaning: Column normalization**

	Column normalization. For instance, are the differences in column names for the different datasets accounted for? Is one SRID consistently used for all datasets?
2	**Appropriate column data types** Column types in (geo)pandas dataframes are appropriate and consistent. For example, transform date columns from strings to datetime Python objects.
Max points: 10	
	Part 2: Storing Data
2	**Schema: Construction & Execution** A "schema.sql" file is included. It does not fail when running the following command in Bash within the project directory (where N is your group number): `$ createdb groupNproject` `$ psql --dbname groupNproject -c 'CREATE EXTENSION postgis'` `$ psql --dbname groupNproject -f schema.sql`
2	**Schema: At least 4 tables created** At least 4 tables are created, one for: NYC zip codes, 311 complaints, trees census, and zillow rent history. Groups may choose to use pure SQL, SQLAlchemy, or another approach to create the tables. Groups do not have to stick to one approach to create all tables. The 4 tables should show up when running the following query: `SELECT table_name` `FROM information_schema.tables` `WHERE table_schema=public` `AND table_type='BASE TABLE';` *Another table may show up that you don't recognize, called spatial_ref_sys. This is fine - this was created when you turned on the postgis extension for the database.*
2	**Schema: Primary Keys** Each of the 4 main tables have a primary key designated.

	Primary keys will be detected when running the following query on each table: `SELECT column_name, is_nullable` `FROM information_schema.columns` `WHERE table_name = '$TABLE_NAME';` This will list information about the columns in the table. Example: `column_name | is_nullable` `-------------+-------------` `id | NO` `zipcode | YES` `geom | YES` If one of the column names is not nullable, then that's the primary key (the above example shows that 'id' is the primary key). Otherwise, no primary keys were found for the table.
8	**Schema: Appropriate data types used** Appropriate data types are used for each column. Graders will be looking at the "schema.sql" file and confirming using the query below for the column types. `SELECT column_name, udt_name` `FROM information_schema.columns` `WHERE table_name = '$TABLE_NAME';` Be sure to check this yourself – depending on how you write data to the database, column types may change unless you're careful. *Note that the udt_name is the column type, and the result may look slightly different than what you defined; this is okay. For example, if you used sqlalchemy's String type, it would say 'varchar'.*
3	**Schema: Consistent Geometry** Every geometry-type column across all relevant tables uses the same SRID. SRIDs will be detected when running the following query on each table that should have a geometry column: `SELECT Find_SRID(`

	`'public', '$TABLE_NAME', '$COLUMN_NAME');`
2	**Data is added to a PostgreSQL Database** Data is added to the tables. How this is done does not matter (pandas, geopandas, SQLAlchemy, pure SQL, etc). It's okay to choose different approaches depending on the table. Be sure to preserve the geometry-types.
Max points: 19	
	Part 3: Understanding Data
6	**Query: Construction & Execution** There are at least 6 files with .sql extension (other than the schema.sql file). Each SQL file has a "human-friendly" name for the query that it contains. Each file contains a distinct query that does not fail when executing.
2	**Query: Correctness (Queries 1 & 2)** 1 point each. General/relative correctness for each query question/prompt. We're ***not*** expecting numbers to be exact anywhere. Just that it seems sane, and it appropriately addresses the question/prompt. Must be valid SQL syntax to get any points. Code that fails to execute will result in a 0 for that query. Appropriate use of relevant PostgreSQL-supported functions (e.g. count, avg, etc).
6	**Query: Correctness (Queries 3, 4, & 5)** 2 points each. Queries 3-5 are worth more than 1 & 2 since they are more complex. General/relative correctness for each query question/prompt. We're ***not*** expecting numbers to be exact anywhere. Just that it seems sane, and it appropriately addresses the question/prompt. Must be valid SQL syntax to get any points. Code that fails to execute will result in a 0 for that query. Appropriate use of relevant PostgreSQL-supported functions (e.g. count,

	avg, etc). Queries more than one table in a single query statement using appropriate joins.
3	**Query: Correctness (Query 6)** Query 6 is worth more than 1-5 since it is a pretty complex query. General/relative correctness for each query question/prompt. We're ***not*** expecting numbers to be exact anywhere. Just that it seems sane, and it appropriately addresses the question/prompt. Must be valid SQL syntax to get any points. Code that fails to execute will result in a 0 for that query. Appropriate use of relevant PostgreSQL-supported functions (e.g. count, avg, etc). A portion of the query can first be constructed using plain Python logic, if desired. Meaning, you may calculate the radius outside SQL.
Max points: 17	
	Part 4: Visualizing Data
6	**Visualizations: at least 6 visualizations are created** There are at least 6 visualizations created - one per question/prompt. If possible, the visualizations are already executed and therefore rendered (so they can be seen when looking at the GitHub URL directly).
3	**Data for visualizations comes from SQL tables** Data that is used for visualizations comes from querying the relevant SQL tables before populating into Pandas/GeoPandas dataframes or any other Python object.
3	**Visualizations: appropriate plot types** The appropriate graph/plot types have been chosen for the given question/prompt.
3	**Visualizations: readability** All graphs and visualizations should have titles, axes labeled, legends (if there is more than one series being plotted), and should be readable (i.e. axis tick values do not overlap).

Max points: 15	

Total max points: 70

Extra Credit (+15%)

Max Points	What
2.5	**1 or more animations and/or widgets for visualizations** At least 1 of the visualizations are enhanced with either an animation (like these animated matplotlib graphs), or an interactive widget (via Jupyter interactive widgets or matplotlib widgets or something similar).
2.5	**100% test coverage** Every implemented function and class method has at least one unit test. Students may use any test framework they'd like. Partial points awarded for partial coverage.
2.5	**Type hints for all functions & classes** Every implemented function, class, and class method signature has type annotation/type hints. Global variables/constants don't need to be type hinted to get full points. Partial points awarded for partial coverage.
2.5	**Additional table + query or visualization** A third dataset from NYC's open data site has been downloaded and appropriately cleaned & normalized. A 5th table has been created. With that table, and joining with one or more existing tables, another SQL query or visualization is made. Groups should make it clear (e.g. in a markdown cell) what question they're trying to answer with the SQL query or visualization. A schema file is nice but not required. Following the same rubric for everything else: ● The table: ○ Has a primary key ○ Uses appropriate data types ○ Contains data ● The query (if chosen): ○ Can be successfully executed ○ Is appropriately implemented (uses appropriate SQL constructs that answers their question) ● The visualization (if chosen): ○ Uses data from the SQL table

	○ Uses an appropriate viz/graph type for the question posed ○ Is readable with a title, labeled axes, and a legend (if needed) Partial points can be awarded for only meeting some of the rubric above. No extra points will be given if both a SQL query and a visualization is done (but you're free to do so).
5.0	**Early submission** Submit the project 1 week early, by Sunday, December 3rd, 11:59PM ET.

Total max points: 15