Anna Kerhoulas , Willow Traylor , Claire Fuller
Si 201
Barbara Ericson
12/11/25

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (5 points)

Our goal for this project was to collect data from 3 different APIs: Spotipy, TMDB, and TVMaze. From each of these APIs, we planned to gather data about songs/movies/shows, their genres, popularity, and release date. For song data, we also planned to collect data about the song's artist. With this data, we planned to calculate the average popularity of a form of media per month, compare the popularity of different media types, and search for a correlation between song popularity and movie/show popularity.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (5 points)

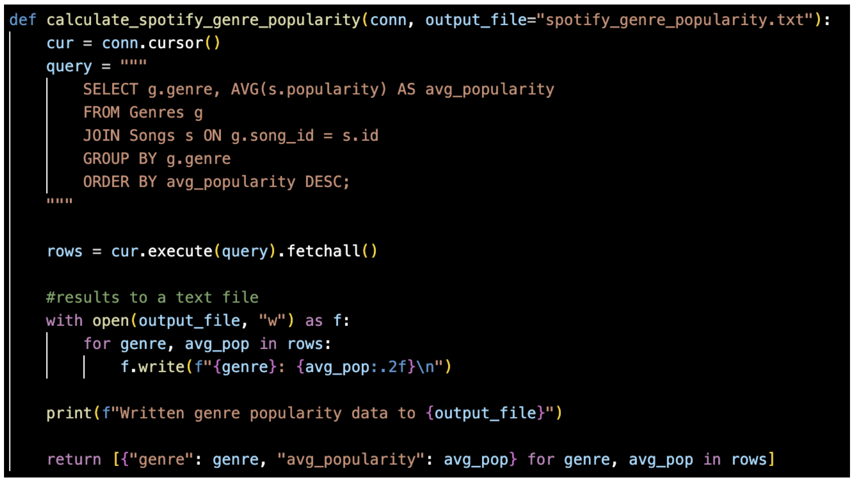
We used the Spotipy, TMDB, and TVMaze APIs to gather the information about each different media type. With that data we decided to compare the most popular genre in regards to each media type.

3. The problems that you faced (5 points)

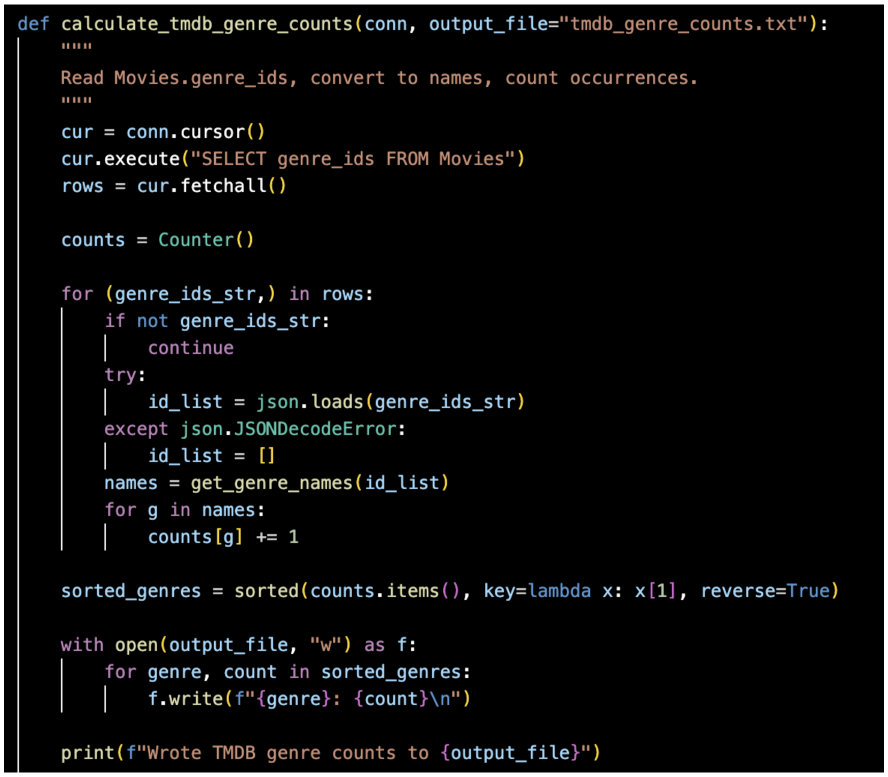
Git as a collaborative platform is quite hard to get used to, we had a hard time not editing at the same time. The Spotipy API was also difficult to work with and (Claire) had a hard time working with the correct query while still having enough data to work with.

Another issue we faced was collecting shows with recent premiere dates from the TVMaze API. We tried looking through multiple pages of the API to find more recently premiered shows, but this was slow and still resulted in older shows being collected. Because of this, we decided not to calculate popularity based on dates. Instead, we calculated and compared the most popular genres across the three media types.

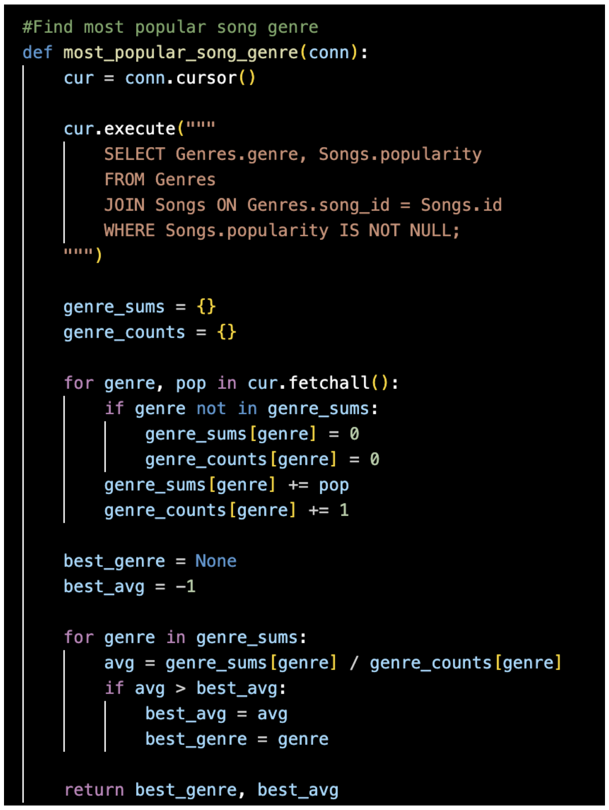
4. The calculations from the data in the database (i.e. a screen shot) (5 points)



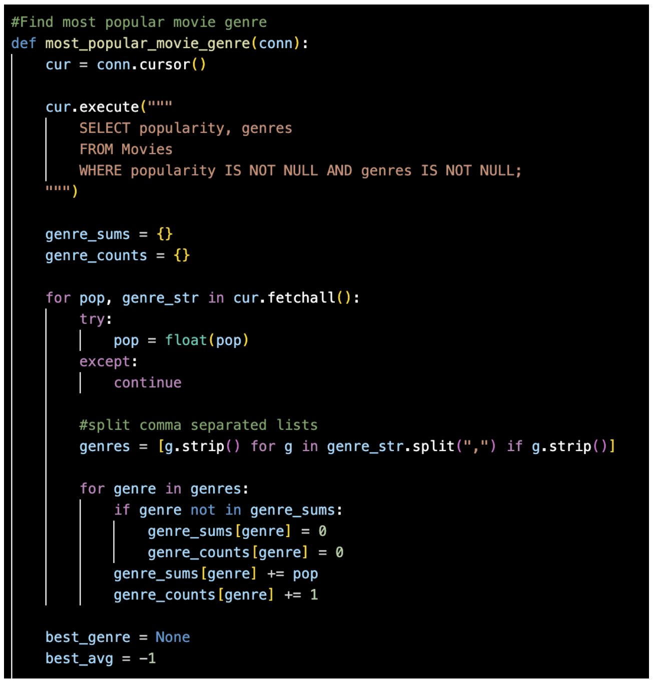
```python
def calculate_spotify_genre_popularity(conn, output_file="spotify_genre_popularity.txt"):
    cur = conn.cursor()
    query = """
        SELECT g.genre, AVG(s.popularity) AS avg_popularity
        FROM Genres g
        JOIN Songs s ON g.song_id = s.id
        GROUP BY g.genre
        ORDER BY avg_popularity DESC;
    """

    rows = cur.execute(query).fetchall()

    #results to a text file
    with open(output_file, "w") as f:
        for genre, avg_pop in rows:
            f.write(f"{genre}: {avg_pop:.2f}\n")

    print(f"Written genre popularity data to {output_file}")

    return [{"genre": genre, "avg_popularity": avg_pop} for genre, avg_pop in rows]
```



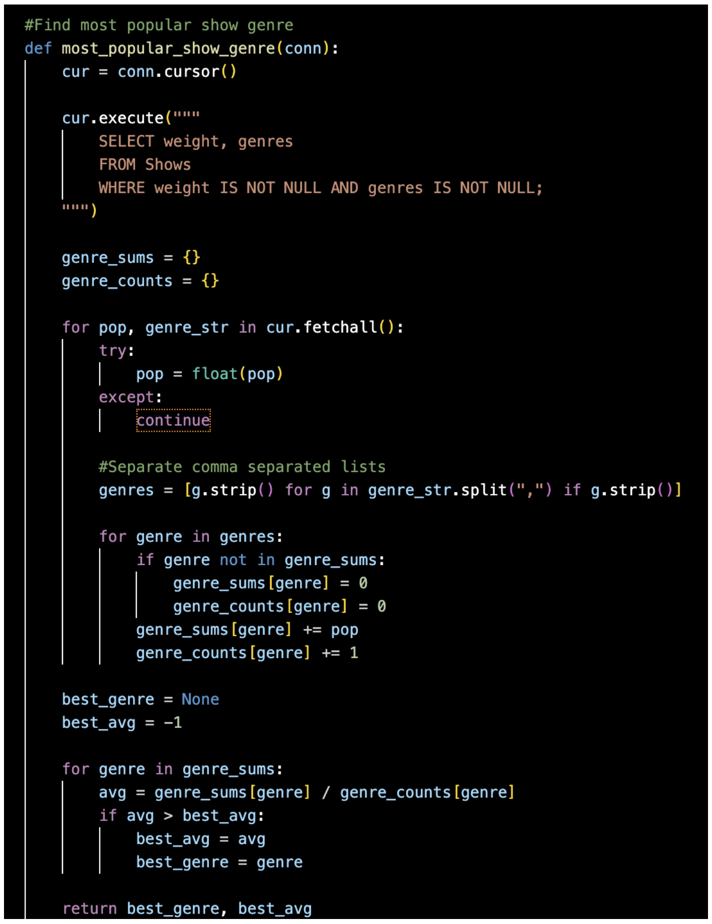
```python
def calculate_tmdb_genre_counts(conn, output_file="tmdb_genre_counts.txt"):
    """
    Read Movies.genre_ids, convert to names, count occurrences.
    """
    cur = conn.cursor()
    cur.execute("SELECT genre_ids FROM Movies")
    rows = cur.fetchall()

    counts = Counter()

    for (genre_ids_str,) in rows:
        if not genre_ids_str:
            continue
        try:
            id_list = json.loads(genre_ids_str)
        except json.JSONDecodeError:
            id_list = []
        names = get_genre_names(id_list)
        for g in names:
            counts[g] += 1

    sorted_genres = sorted(counts.items(), key=lambda x: x[1], reverse=True)

    with open(output_file, "w") as f:
        for genre, count in sorted_genres:
            f.write(f"{genre}: {count}\n")

    print(f"Wrote TMDB genre counts to {output_file}")
```



```python
#Find most popular song genre
def most_popular_song_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT Genres.genre, Songs.popularity
        FROM Genres
        JOIN Songs ON Genres.song_id = Songs.id
        WHERE Songs.popularity IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for genre, pop in cur.fetchall():
        if genre not in genre_sums:
            genre_sums[genre] = 0
            genre_counts[genre] = 0
        genre_sums[genre] += pop
        genre_counts[genre] += 1

    best_genre = None
    best_avg = -1

    for genre in genre_sums:
        avg = genre_sums[genre] / genre_counts[genre]
        if avg > best_avg:
            best_avg = avg
            best_genre = genre

    return best_genre, best_avg
```

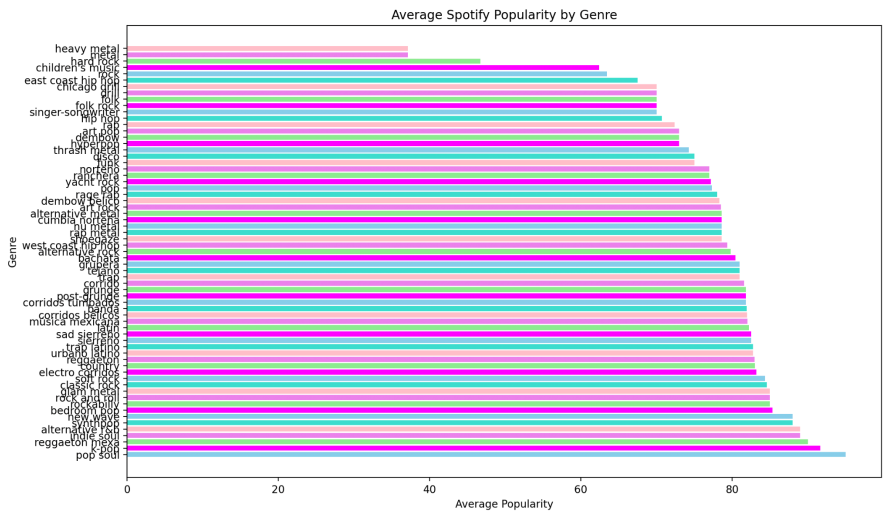


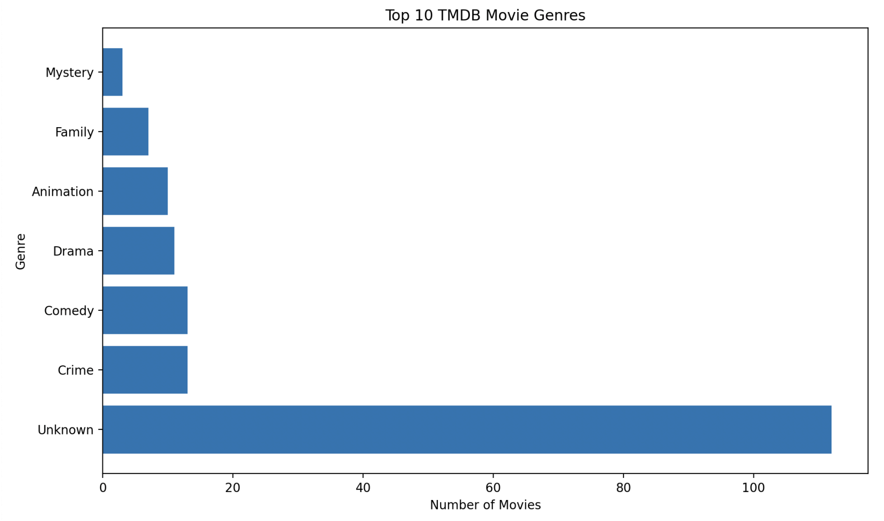
```python
#Find most popular movie genre
def most_popular_movie_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT popularity, genres
        FROM Movies
        WHERE popularity IS NOT NULL AND genres IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for pop, genre_str in cur.fetchall():
        try:
            pop = float(pop)
        except:
            continue

        #split comma separated lists
        genres = [g.strip() for g in genre_str.split(",") if g.strip()]

        for genre in genres:
            if genre not in genre_sums:
                genre_sums[genre] = 0
                genre_counts[genre] = 0
            genre_sums[genre] += pop
            genre_counts[genre] += 1

    best_genre = None
    best_avg = -1
```

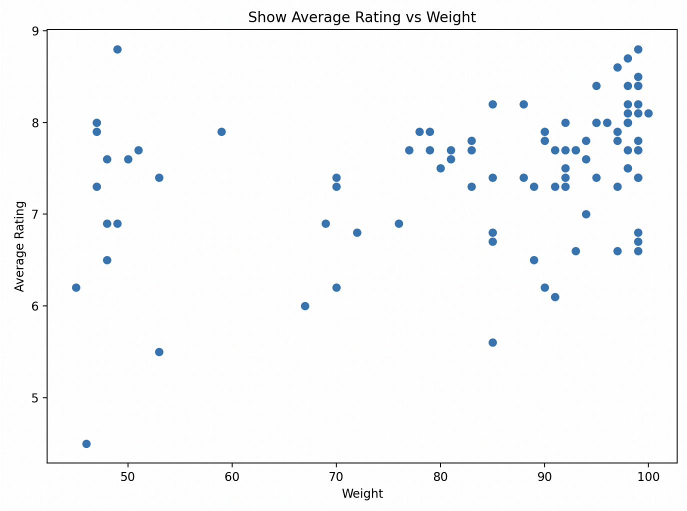


```python
#Find most popular show genre
def most_popular_show_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT weight, genres
        FROM Shows
        WHERE weight IS NOT NULL AND genres IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for pop, genre_str in cur.fetchall():
        try:
            pop = float(pop)
        except:
            continue

        #Separate comma separated lists
        genres = [g.strip() for g in genre_str.split(",") if g.strip()]

        for genre in genres:
            if genre not in genre_sums:
                genre_sums[genre] = 0
                genre_counts[genre] = 0
            genre_sums[genre] += pop
            genre_counts[genre] += 1

    best_genre = None
    best_avg = -1

    for genre in genre_sums:
        avg = genre_sums[genre] / genre_counts[genre]
        if avg > best_avg:
            best_avg = avg
            best_genre = genre

    return best_genre, best_avg
```

5. The visualization that you created (i.e. screen shot or image file) (5 points)

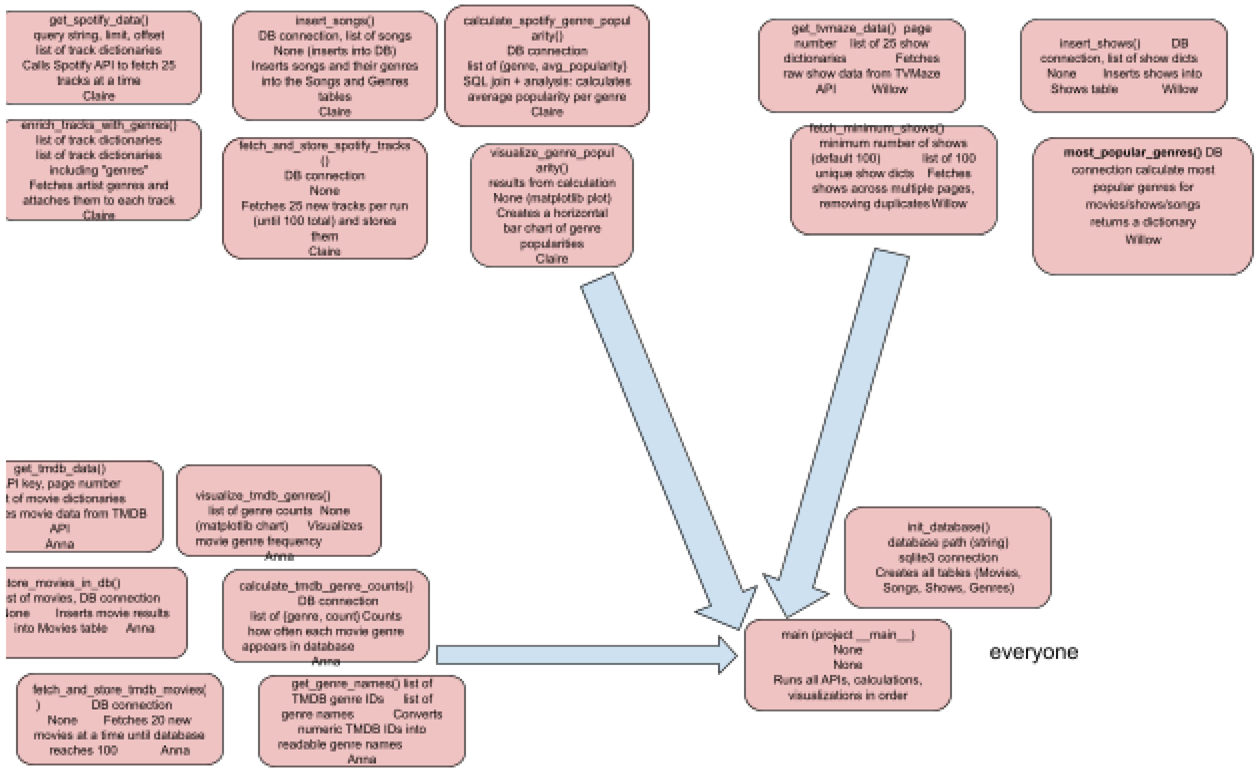






6. Instructions for running your code (5 points)
- Simply run the code.
- To get more data in the DataBase run more than once and more data will be added.

7. An updated function diagram with the names of each function, the input, and output and who was responsible for that function (10 points)


get_spotify_data()
query string, limit, offset
list of track dictionaries
Calls Spotify API to fetch 25 tracks at a time
Claire
insert_songs()
DB connection, list of songs
None (inserts into DB)
Inserts songs and their genres into the Songs and Genres tables
Claire
calculate_spotify_genre_popularity()
DB connection
list of (genre, avg_popularity)
SQL join + analysis: calculates average popularity per genre
Claire
enrich_tracks_with_genres()
list of track dictionaries
list of track dictionaries including "genres"
Fetches artist genres and attaches them to each track
Claire
fetch_and_store_spotify_tracks()
DB connection
None
Fetches 25 new tracks per run (until 100 total) and stores them
Claire
visualize_genre_popularity()
results from calculation
None (matplotlib plot)
Creates a horizontal bar chart of genre popularities
Claire
get_tvmaze_data() page number list of 25 show dictionaries Fetches raw show data from TVMaze API Willow
insert_shows() DB connection, list of show dicts None Inserts shows into Shows table Willow
fetch_minimum_shows() minimum number of shows (default 100) list of 100 unique show dicts Fetches shows across multiple pages, removing duplicates Willow
most_popular_genres() DB connection calculate most popular genres for movies/shows/songs returns a dictionary Willow
get_tmdb_data()
PI key, page number
t of movie dictionaries
es movie data from TMDB API
Anna
visualize_tmdb_genres() list of genre counts None (matplotlib chart) Visualizes movie genre frequency Anna
tore_movies_in_db()
st of movies, DB connection
lone Inserts movie results into Movies table Anna
calculate_tmdb_genre_counts() DB connection list of (genre, count) Counts how often each movie genre appears in database Anna
fetch_and_store_tmdb_movies() DB connection None Fetches 20 new movies at a time until database reaches 100 Anna
get_genre_names() list of TMDB genre IDs list of genre names Converts numeric TMDB IDs into readable genre names Anna
init_database()
database path (string)
sqlite3 connection
Creates all tables (Movies, Songs, Shows, Genres)
main (project __main__)
None
None
Runs all APIs, calculations, visualizations in order
everyone

8. You must also clearly document all resources you used. The documentation should be of the following form (10 points)

| Date | Issue Description | Location of Resource | Result |
|---|---|---|---|
| 12/09/25 | Understanding stashing changes with Github | https://git-scm.com/docs/git-stash | Successfully work on the file at the same time as other group members rather than one at a time |
| 12/09/25 | Understanding TVMaze API | https://www.tvmaze.com/api | Successfully used documentation to call API and use data |
| 12/10/25 | Getting started with spotipy API | https://spotipy.readthedocs.io/en/2.22.1/ | Successfully use information in spotipy API. |
| 12/10/25 | Inserting data into databases | https://www.sqlitetutorial.net/sqlite-python/i | Was able to successfully insert |

| Date | Issue Description | Location of Resource | Result |
|---|---|---|---|
| 12/09/25 | Understanding stashing changes with Github | https://git-scm.com/docs/git-stash | Successfully work on the file at the same time as other group members rather than one at a time |
|  |  | nsert/ | data into database |
| 12/10/25 | Using Sqlite is confusing | https://www.freecodecamp.org/news/work-with-sqlite-in-python-handbook/ | Gathered more info on how to use SQlite |
| 12/10/25 | Difficulty analyzing spotipy API. | https://developer.spotify.com/documentation/web-api/reference/search | Helped understand query, limit, etc. |