Project Final Submission Template

Step 1a: Planning

Identify the information in the file your program will read

Double click this cell to edit.

About the song:

- Title of the song
- Artist who created the song
- Year song was released
- Popularity: **It is unclear what the popularity exactly represents, as they are represented as floats. However, since they are to 9 decimal places, it will be assumed that popularity is represented as views in billions. Also, some place "NaN" as the popularity. Some popularities are 0.**
- Duration of the song in seconds
- Tempo of the song is beats per minute

As "NaN" is a non-float type, it is unreliable as a popularity, thus rows with "NaN" for popularity may be skipped. Popularities of 0 most likely occur due to a very low value in popularity (eg: 0.000000100), but will not be marked as unreliable, as they still portray that the popularity was extremely low, which is still useful information.

Step 1b: Planning

Brainstorm ideas for what your program will produce

Select the idea you will build on for subsequent steps

The objective of the program is to produce a graph that illustrates how the tempo of songs have evolved over time. This can be done in three different ways:

1. Display the relationship between tempo (bpm) and popularity throughout time, represented by a scatterplot. Each point will represent a decade and will be plotted in a plane with tempo and popularity on the axis, derived by computing the average popularity and tempos of all songs in a given decade, and repeating for each decade (that of which represents each point). If possible, each point (which represents a particular decade) will be coloured differently to help see the change over time.

2. Display the distribution of tempos and their popularity through a histogram by creating intervals of tempos (eg: 100-110bpm, 110-120bpm) and calculating the average popularities of songs given the interval of tempos.

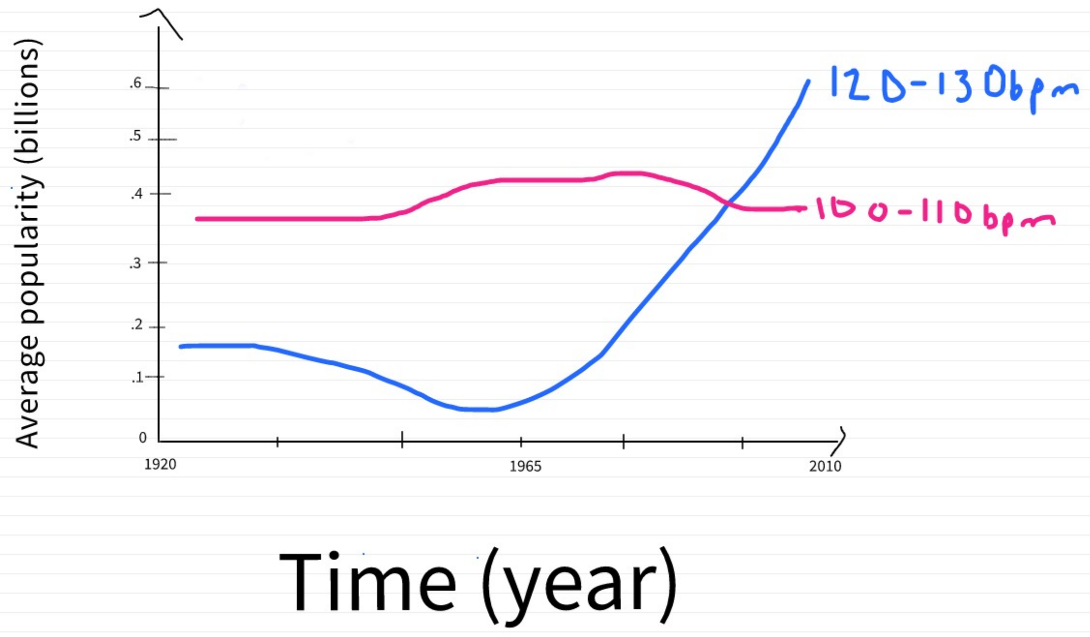
3. Similiar to the output of 1, graph how popularities of certain tempos have changed over time through a line chart. The process in doing so will be similar to 2 in that first, the program would filter all the rows with certain bpms (eg: 100-110bpm, 110-120bpm), representing the line itself, and place it on a plane with time on its x-axis and popularity on the y-axis. Then, for each year, plot the average popularities of all songs for the given year. The process will be repeated for other tempos, creating a line chart with multiple lines.

The chosen program will be the one described in 3.

Step 1c: Planning

Write or draw examples of what your program will produce

The graph will have one line for each range of tempo: 80-90bpm, 90-100bpm, 100-110bpm, 110-120bpm, 120-130bpm and 130-140bpm, meaning the graph will display 6 lines in total. For the list of songs specified under the tempo group of each line, the average popularities of all songs released within a certain calendar year will be plotted for all years, showing how the popularity for that tempo has changed over time. **The graph should look like this, but with more lines:**



Step 2a: Building

Document which information you will represent in your data definitions

Design data definitions

The programs objective is to display how popularity of certain tempos have changed over time, ultimately represented by a line graph. The graph requires three variables: time (given in year), popularity (given in units of billion views) and tempo (given in bpm). The data definition for compound will consist of those three fields: year, popularity and tempo.

The csv file contains over 50,000 rows, thus song names that start with letters N-Z will be removed to allow the program to run within a sensible amount of time. Although songs are filtered by their title, the title does not need to be a field, as rows, which are interpreted as list of strings will be removed under the condition that the first letter of the first element in the row (i.e. first letter in the list of strings) is within the range A-M.

In [1]:

```
from cs103 import *
from typing import NamedTuple, List
import csv

##################
# Data Definitions

SongData = NamedTuple("SongData", [("year", int), # in range (0, ...)
                                   ("popularity", float), # in range [0, ...)
                                   ("tempo", float)]) # in range [0, ...)
# interp. the attributes of a song: the title of the song (title), the year it was released (year),
# its popularity in units of billion views, and its tempo in bpm.

SD1 = SongData(2009, 0.355285536, 139.94)
SD2 = SongData(1973, 0.487112195, 80.29)
SD3 = SongData(2001, 0.572983739, 100.23)
SD4 = SongData(2010, 0.284309293, 109.99)
SD5 = SongData(2009, 0.793812398, 83.23)
SD6 = SongData(1973, 0.982437823, 85.67)
SD7 = SongData(2009, 0.827393833, 85.67)
SD8 = SongData(2009, 0.578439847, 89.99)
SD9 = SongData(2001, 0.578439847, 89.99)
SD10 = SongData(2009, 0.487112195, 132.29)
SD11 = SongData(1999, 0.538929309, 135.55)
SD12 = SongData(1999, 0.538929309, 105.55)
SD13 = SongData(2002, 0.538929309, 145.55)
SD14 = SongData(2009, 0.638929309, 140.00)

# template based on Compound
@typecheck
def fn_for_song_data(sd: SongData) -> ...:
    return ...(sd.year,
               sd.popularity,
               sd.tempo)

##############################################################################################################

# List[SongData]
# interp. a list of SongData

LOSD0 = []
LOSD1 = [SD1]
LOSD2 = [SD1, SD2]
LOSD3 = [SD1, SD2, SD3]
LOSD4 = [SD1, SD2, SD3, SD4]
LOSD5 = [SD8, SD7, SD5] # all from 2009 and within bpm 80-90
LOSD6 = [SD1, SD2, SD3, SD4, SD5, SD6, SD7, SD8]
LOSD7 = [SD1, SD2, SD3, SD4, SD5, SD6, SD7, SD8,
         SD9, SD10, SD11, SD12, SD13, SD14]

# template based on arbitrary-sized data and reference rule
@typecheck
def fn_for_losd(losd: List[SongData]) -> ...:
    # description of accumulator
    acc = ...   # type: ...

    for sd in losd:
        acc = ...(fn_for_song_data(sd), acc)

    return ...(acc)

# or choose another template

# List[float]
# interp. a list of floats
LOS0 = []
LOS1 = [34.28, -23.2]

# template based on arbitrary-sized data
@typecheck
def fn_for_lof(lof: List[float]) -> ...:
    # description of accumulator
    acc = ... # type: ...

    for f in lof:
        acc = ...(f, acc)

    return ...(acc)

##############################################################################################################

# List[int]
# interp. a list of integers
LOI0 = []
LOI1 = [5, -4]

# template based on arbitrary-sized data
@typecheck
def fn_for_loi(loi: List[int]) -> ...:
    # description of accumulator
    acc = ... # type: ...

    for i in loi:
        acc = ...(i, acc)

    return ...(acc)
```

```
##############################################################################################################

# List[str]
# interp. a list of strings
LOS0 = []
LOS1 = ['cat', 'dog']

# template based on arbitrary-sized data
@typecheck
def fn_for_los(los: List[str]) -> ...:
    # description of accumulator
    acc = ... # type: ...

    for s in los:
        acc = ...(s, acc)

    return ...(acc)
```

Step 2b and 2c: Building

Design a function to read the information and store it as data in your program

Design functions to analyze the data

Complete these steps in the code cell below. You will likely want to rename the analyze function so that the function name describes what your analysis function does.

In [2]:

```
# # Functions

@typecheck
def read(filename: str) -> List[SongData]:
    """
    reads information from the specified file and returns a list of song information,
    whos rows fit the following criteria:
    (a) The first letter of the song title beings with the letters A-M,
    (b) The data is reliable, meaning it does not include "nan" as its popularity.
    returns all columns EXCEPT those that fail to reach the previous citeria.
    """
    # return []  #stub
    # Template from HtDAP and reference rule

    # losd contains the result (rows in csv file) seen so far
    losd = [] # type: List[SongData]

    with open(filename) as csvfile:

        reader = csv.reader(csvfile)
        next(reader) # skip header line

        for row in reader:
            if passes_filter(row): # removes song titles starting with N-Z and whose popularity is "NaN"
                sd = SongData(parse_int(row[2]), parse_float(row[3]), parse_float(row[5]))
                losd.append(sd)

    return losd

@typecheck
def passes_filter(row: List[str]) -> bool:
    """
    return True if the first lettr of the song Title starts with letters "A" through "M"
    AND the popularity is a float type, not "nan"
    """

    # return True # stub

    # template treats row as atomic and uses reference rule
    # return ...(row) # template

    if within_a_to_m(row) and is_reliable(row):
        return True
    else:
        return False


@typecheck
def within_a_to_m(row: List[str]) -> bool:
    """
    return True if the first letter of the song Title starts with letters "A" through "M"
    False otherwise
    List CANNOT be empty
    """
    # return True # stub
    # template treats row as atomic and uses indexing instead
    # return ...(row)  #template

    if (row[0])[0] == "A":
        return True
    elif (row[0])[0] == "B":
        return True
    elif (row[0])[0] == "C":
        return True
    elif (row[0])[0] == "D":
        return True
    elif (row[0])[0] == "E":
        return True
    elif (row[0])[0] == "F":
        return True
    elif (row[0])[0] == "G":
        return True
    elif (row[0])[0] == "H":
        return True
    elif (row[0])[0] == "I":
        return True
    elif (row[0])[0] == "J":
        return True
    elif (row[0])[0] == "K":
        return True
    elif (row[0])[0] == "L":
        return True
    elif (row[0])[0] == "M":
        return True
    else:                           # if song title starts with N-Z
        return False

@typecheck
def is_reliable(row: List[str]) -> bool:
    """
    return True if the popularity is a float type, not "nan"

    Assumes that popularity from every column falls on the 4th row,
    and list is never empty
    """
    # return True # stub
    # template treats row as atomic and uses indexing instead
    # return ...(row)  #template
    return row[3] != "nan"


# Begin testing:
###########################################################################################################

start_testing() # Examples and tests for read, has to filter out A-M  and nan

expect(read("empty.csv"), [])
expect(read("6_songs_small.csv"), [SongData(2007, 0.439525605, 177.61),
                                   SongData(1989, 0.54005084, 119.849)])
expect(read("9_songs_small.csv"), [SongData(2002, 0.442094857, 80.525),
                                   SongData(2009, 0.355285536, 140.703),
                                   SongData(2004, 0.672121944, 84.682),
                                   SongData(1992, 0.427446571, 71.254)])

summary()

###########################################################################################################

start_testing() # Examples and tests for passes_filter

expect(passes_filter(["A Man Without a Country",
                      "Stinking Lizaveta","2009",
                      "0.355285536","152.97261","140.703"]), True)
expect(passes_filter(["On the Surface","Stemm",
                      "2005","0.473346541","250.98404","135.598"]), False)
expect(passes_filter(["Don't Blame Me","Timi Yuro",
                      "1961","0","131.5522","119.549"]), True)
expect(passes_filter(["Voices", "Blue Foundation",
                      "2006","0.544514002","325.09342","105.952"]), False)

summary()

###########################################################################################################

start_testing() # Examples and tests for within_a_to_m

expect(within_a_to_m(["A Man Without a Country",
                      "Stinking Lizaveta","2009",
                      "0.355285536","152.97261","140.703"]), True)
expect(within_a_to_m(["On the Surface","Stemm",
                      "2005","0.473346541","250.98404","135.598"]), False)
expect(within_a_to_m(["Don't Blame Me","Timi Yuro",
                      "1961","0","131.5522","119.549"]), True)
expect(within_a_to_m(["Voices", "Blue Foundation",
                      "2006","0.544514002","325.09342","105.952"]), False)

summary()

###########################################################################################################

start_testing() # Examples and tests for is_reliable

expect(is_reliable(["A Man With a Country",
                    "Pleasantly Smelling Lizaveta","2009",
                    "nan","152.97261","140.703"]), False)
expect(is_reliable(["On the Surface","Stemm",
                    "2005","0.473346541","250.98404","135.598"]), True)
expect(is_reliable(["Blame Me","Timmy Yuro",
                    "1961","nan","131.5522","119.549"]), False)
expect(is_reliable(["Voices", "Blue Foundation",
                      "2006","0.544514002","325.09342","105.952"]), True)

summary()
```

```
3 of 3 tests passed
4 of 4 tests passed
4 of 4 tests passed
4 of 4 tests passed
```

In [3]:

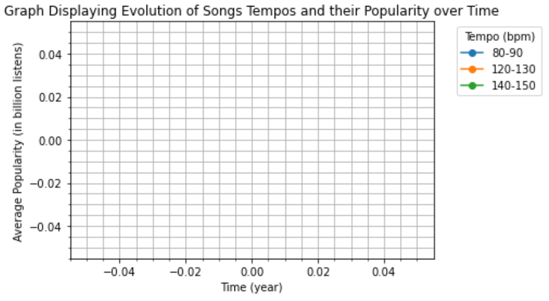
```
from matplotlib import pyplot

##################
# Functions

@typecheck
def main(filename: str, lotempo: List[int]) -> None:
    """
    Reads the file from given filename, analyzes the data, and
    returns a line graph with Time on the x-axis and Popularity on the y-axis,
    and multiple lines representing a range of bpm (eg: 80-90 bpm, 120-130 bpm),
    which are plotted by averaging the popularity of all songs in a year.

    The user is given the choice to choose which range of bpms they would like
    to display, and they can choose up to four entries for bpm.
    """
    # return None # stub

    # Template from HtDAP, based on composition
    return popularity_bpm_graph(read(filename), lotempo)


@typecheck
def popularity_bpm_graph(losd: List[SongData], lotempo: List[int]) -> None:
    """
    returns a line chart with Time (in year) on x-axis and Average
    Popularity (in billions) on the y-axis, which can be a value between 0.0 and 1.0.

    A line representing a range of bpm (eg: 80-90 bpm, 120-130 bpm) will be displayed,
    each point of the line plotted by averaging the popularity of all songs in every
    year.

    The user is given the choice to choose which range of bpms they would like
    to display, and they can choose up to 4 entries for bpm.
    The first number of the bpm range must be divisible by 10 which will create
    an interval of bpms equal to [bpm, bpm + 9.999].
    """
    # return None # stub
    # Template from viz and indexing

    first_bpm_x = sort_years(losd, lotempo[0])                        # sort time in ascending order
    first_bpm_y = average_popularity_per_bpm(losd, lotempo[0])

    if len(lotempo) >= 2:                                             # if user enters >= 2 bpm entries
        second_bpm_x = sort_years(losd, lotempo[1])
        second_bpm_y = average_popularity_per_bpm(losd, lotempo[1])  # create y-data set for 2nd bpm

    if len(lotempo) >= 3:                                             # if user enters >= 3 bpm entries
        third_bpm_x = sort_years(losd, lotempo[2])
        third_bpm_y = average_popularity_per_bpm(losd, lotempo[2])   # create y-data set for 3rd bpm

    if len(lotempo) == 4:                                             # if user enters 4 bpm entries
        fourth_bpm_x = sort_years(losd, lotempo[3])
        fourth_bpm_y = average_popularity_per_bpm(losd, lotempo[3])  # create y-data set for 4th bpm


    pyplot.title("Graph Displaying Evolution of Songs Tempos and their Popularity over Time")
    pyplot.xlabel("Time (year)")
    pyplot.ylabel("Average Popularity (in billion listens)")

    line1 = pyplot.plot(first_bpm_x, first_bpm_y,
                        marker = "o",
                        label = say_bpm_range(lotempo[0]))
    pyplot.legend(bbox_to_anchor=(1.05, 1), loc=2, title="Tempo (bpm)")       # puts legend outside graph

    if len(lotempo) >= 2:                                                      # if user enters >= 2 bpm entries
        line2 = pyplot.plot(second_bpm_x, second_bpm_y,                        # create line for 2nd bpm
                            marker = "o",
                            label = say_bpm_range(lotempo[1]))                 # state bpm range in legend
        pyplot.legend(bbox_to_anchor=(1.05, 1), loc=2, title="Tempo (bpm)")

    if len(lotempo) >= 3:                                                      # if user enters >= 3 bpm entries
        line3 = pyplot.plot(third_bpm_x, third_bpm_y,                          # create line for 3rd bpm
                            marker = "o",
                            label = say_bpm_range(lotempo[2]))
        pyplot.legend(bbox_to_anchor=(1.05, 1), loc=2, title="Tempo (bpm)")

    if len(lotempo) == 4:                                                      # if user enters 4 bpm entries
        line4 = pyplot.plot(fourth_bpm_x, fourth_bpm_y,                        # create line for 4th bpm
                            marker = "o",
                            label = say_bpm_range(lotempo[3]))
        pyplot.legend(bbox_to_anchor=(1, 1), loc=2, title="Tempo (bpm)")


    pyplot.minorticks_on()                           # displays ticks for all years in x-axis
    pyplot.grid(True, which="both", axis="both")     # applies gridlines to both time and popularity
    pyplot.show()


@typecheck
def say_bpm_range(tempo: int) -> str:
    """
    return the bpm range given the inputted tempo
    """

    # return "" # stub

    # template based on simple-atomic data

    return str(tempo) + "-" + str(tempo+10)



################################################################################################################
#                                FUNCTIONS RELATING TO X-VALUE YEARS                                          #
################################################################################################################

@typecheck
def sort_years(losd: List[SongData], tempo: int) -> List[int]:
    """
    return list of years in losd in ascending chronological order
    """

    # return [] # stub

    # template from composition

    return sorted(get_years(get_years_bpm(losd, tempo)))


@typecheck
def get_years_bpm(losd: List[SongData], tempo: int) -> List[SongData]:
    """
    returns list of song data that is the same tempo as given tempo
    """

    # return [] # stub

    # template from List[SongData] and reference rule

    # record song data for given tempo seen so far
    tempo_data = []   # type: List[SongData]

    for sd in losd:
        if is_in_bpm(sd, tempo):
            tempo_data.append(sd)
    return tempo_data


@typecheck
def get_years(losd: List[SongData]) -> List[int]:
    """
    returns all the years in the list of Song Data
    """

    # return [] # stub

    # template from List[SongData] and reference rule

    # record years in songs seen so far
    years = []   # type: List[int]

    for sd in get_years_without_repeat(losd):
        years.append(sd.year)                                 # returns list of years without duplicates

    return years

@typecheck
def get_years_without_repeat(losd: List[SongData]) -> List[SongData]:
    """
    return list of song data without repeating values for year
    """

    # return [] # stub

    # template from List[SongData] and reference rule

    # records SongData that has not repeated years so far
    polished_list = [] # type: List[SongData]

    for sd in losd:
        if check_year_repeat(polished_list, sd.year) == 0:        # if year is not already in the list
            polished_list.append(sd)

    return polished_list

@typecheck
def check_year_repeat(losd: List[SongData], year: int) -> int:
    """
    count number of times variable "year" appears on list of song data
    """

    # return True # stub

    # template from List[SongData]

    # records number of times "year" appears as years for list of song data seen so far
    num_of_times = 0 # type: int

    for sd in losd:
        if sd.year == year:
            num_of_times = num_of_times + 1

    return num_of_times



################################################################################################################
#                                FUNCTIONS RELATING TO Y-VALUE POPULARITY                                      #
################################################################################################################


@typecheck
def average_popularity_per_bpm(losd: List[SongData], tempo: int) -> List[float]:
    """
    return the average popularity of songs per bpm for each year

    for a year, if there are no losd, add [] to list
    """

    # return [] # stub

    # template from List[int] and reference rule

    # records average popularity of each year seen so far
    lo_average = [] # type: List[float]

    for year in sort_years(losd, tempo):                          # loop goes from smallest year to largest
        losd_for_year = filter_bpm_and_year(losd, tempo, year)

        average = average_single_popularity(losd, tempo, year)
        if count_list(losd_for_year) > 0:                         # if there are no sd for a particular year
            lo_average.append(average)

    return lo_average

@typecheck
def average_single_popularity(losd: List[SongData], tempo: int, year: int) -> float:
    """
    return the average popularity of songs per bpm for one year

    if list is empty, return average of 0.0
    """

    # return 0.0 # stub

    # template from composition and treating list as simple atomic through reference rule

    valid_list = filter_bpm_and_year(losd, tempo, year)       # filter for one bpm and single year

    if count_list(valid_list) == 0:                           # if list is empty, return average of 0.0
        return 0.0

    average = sum_popularity(valid_list) / count_list(valid_list)
    return average
```

```python
@typecheck
def sum_popularity(losd: List[SongData]) -> float:
    """
    return the sum of popularities in the list
    """

    # return 0.0 # stub

    # template from List[SongData]

    # sums song popularities seen so far
    acc = 0.0   # type: float

    for sd in losd:
        acc = acc + sd.popularity

    return acc

@typecheck
def count_list(losd: List[SongData]) -> int:
    """
    return the number of items in the list
    """

    # return 0 # stub

    # template from List[SongData]

    # counts number of SongData seen in list so far
    num = 0   # type: int

    for sd in losd:
        num = num + 1

    return num

@typecheck
def filter_bpm(losd: List[SongData], tempo: int) -> List[SongData]:
    """
    return a list of song data for songs within the specified bpm
    """
    # return [] # stub

    # template based on List[SongData] with one additional parameter

    # records song data within tempo requirements seen so far
    valid_bpm = []   # type: List[SongData]

    for sd in losd:
        if is_in_bpm(sd, tempo):
            valid_bpm.append(sd)

    return valid_bpm

@typecheck
def is_in_bpm(sd: SongData, tempo: int) -> bool:
    """
    return True if the tempo is in the bpm range
    """
    # return True # stub

    # template from SongData with one additional parameter

    if tempo <= sd.tempo and sd.tempo < (tempo + 10):
        return True
    else:
        return False

@typecheck
def filter_bpm_and_year(losd: List[SongData], tempo: int, year: int) -> List[SongData]:
    """
    return a list of SongData that are in the specified year and tempo range
    """

    # return [] # stub

    # template from List[SongData] and reference rule

    # records SongData within tempo and year requirements seen so far
    final_valid = []   # type: List[SongData]

    for sd in filter_bpm(losd, tempo):
        if is_in_year(sd, year):
            final_valid.append(sd)

    return final_valid

@typecheck
def is_in_year(sd: SongData, year: int) -> bool:
    """
    return True if song is in year
    """

    # return True # stub

    # template from SongData with one additional parameter

    return sd.year == year


# Begin testing:

##############################################################################################################

start_testing() # Examples and tests for main

# Empty graph
expect(main("empty.csv", [80, 120, 140]), None)

# 2 lines with 2 time values: [2005, 2006]
# but will appear without lines because only one
# point per tempo
expect(main("songs_subset1.csv", [110, 130]), None)

# 3 lines with 4 time values: [1985, 1992, 2008, 2009]
# but will appear only one line because tempos 110 and 120
# only have one point
# Line for tempo 80 coords: (1985, ~0.0.266), (1992, ~0.212)
# Tempo 110 coord: (2009, ~0.355)
# Tempo 120 coord: (2008, ~0.631)
expect(main("songs_subset2.csv", [80, 110, 120]), None)

summary()

##########################################################################################################

start_testing() # Examples and tests for popularity_bpm_graph

# Empty graph
expect(popularity_bpm_graph([], [80, 120, 140]), None)

# Will appear without lines because only one point
# Coordinate: (2009, 0.355285536)
expect(popularity_bpm_graph(LOSD1, [130]), None)

# 2 lines with 3 time values: [1973, 2001, 2009]
# but will appear without lines because only one point
# per tempo
expect(popularity_bpm_graph(LOSD3, [80, 100]), None)

# 3 lines with 4 time values: [1973, 2009, 2001, 2010]
# but will appear with two lines because only one point
# for tempo 130.
# Line for tempo 80 coords: (1973, ~0.735), (2009, ~0.733)
# Line for tempo 100 coords: (2001, 0.572983739), (2010, 0.284309293)
# Coordinate for tempo 130: (2009, 0.355285536)
expect(popularity_bpm_graph(LOSD6, [80, 100, 130]), None)

# 4 lines with 5 time values: [1973, 1999, 2001, 2009, 2010]
# Line for tempo 80 coords: (1973, ~0.735), (2001, ~0.578), (2009, ~0.733)
# Line for tempo 100 coords: (1999, ~0.539), (2001, ~0.572), (2010, 0.284)
# Line for tempo 130 coords: (1999, ~0.539), (2009, ~0.467)
# Line for tempo 140 coords: (2002, ~0.539), (2009, ~0.639)
expect(popularity_bpm_graph(LOSD7, [80, 100, 130, 140]), None)

summary()

#########################################################################################################

start_testing() # Examples and tests for say_bpm_range

expect(say_bpm_range(80), "80-90")
expect(say_bpm_range(90), "90-100")
expect(say_bpm_range(100), "100-110")
expect(say_bpm_range(110), "110-120")

summary()

#########################################################################################################

start_testing() # Examples and tests for sort_years

expect(sort_years([SD1, SD2, SD5, SD7, SD8], 80), [1973, 2009])
expect(sort_years(LOSD6, 100), [2001, 2010])
expect(sort_years(LOSD7, 130), [1999, 2009])

summary()

##########################################################################################################

start_testing() # Examples and tests for get_years_bpm

expect(get_years_bpm([], 100), [])
expect(get_years_bpm(LOSD1, 130), LOSD1)
expect(get_years_bpm(LOSD1, 100), [])
expect(get_years_bpm(LOSD3, 100), [SD3])
expect(get_years_bpm([SD2, SD3, SD4], 100), [SD3, SD4])
expect(get_years_bpm([SD1, SD5, SD7, SD8], 80), [SD5, SD7, SD8])
expect(get_years_bpm(LOSD6, 100), [SD3, SD4])
expect(get_years_bpm(LOSD5, 80), [SD8, SD7, SD5])

summary()

##########################################################################################################

start_testing() # Examples and tests for get_years

expect(get_years([]), [])
expect(get_years(LOSD1), [2009])
expect(get_years(LOSD2), [2009, 1973])
expect(get_years(LOSD3), [2009, 1973, 2001])
expect(get_years(LOSD6), [2009, 1973, 2001, 2010])

summary()

##########################################################################################################

start_testing() # Examples and tests for get_years_without_repeat

expect(get_years_without_repeat([]), [])
expect(get_years_without_repeat(LOSD2), LOSD2)
expect(get_years_without_repeat([SD2, SD6, SD3]), [SD2, SD3])
expect(get_years_without_repeat([SD1, SD2, SD5]), [SD1, SD2])
expect(get_years_without_repeat([SD8, SD7, SD6, SD5, SD4]), [SD8, SD6, SD4])
expect(get_years_without_repeat(LOSD6), [SD1, SD2, SD3, SD4])

summary()

##########################################################################################################

start_testing() # Examples and tests for check_year_repeat
```

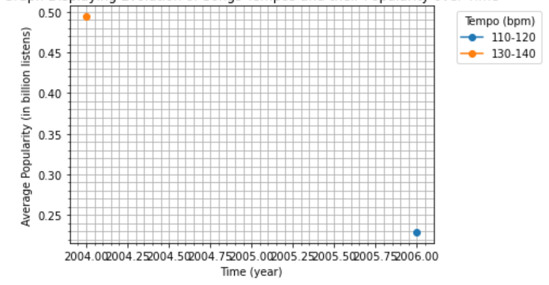
```
expect(check_year_repeat([], 1980), 0)
expect(check_year_repeat(LOSD2, 2010), 0)
expect(check_year_repeat(LOSD2, 1973), 1)
expect(check_year_repeat(LOSD5, 1973), 0)
expect(check_year_repeat(LOSD5, 2009), 3)
expect(check_year_repeat(LOSD6, 2009), 4)
expect(check_year_repeat(LOSD6, 1973), 2)

summary()

##############################################################################################################

start_testing() # Examples and tests for average_popularity_per_bpm

expect(average_popularity_per_bpm([], 80), [])
expect(average_popularity_per_bpm(LOSD1, 130), [0.355285536])
expect(average_popularity_per_bpm([SongData(2009, 0.487112195, 132.29), # SD2 and SD3 not valid bpm
                                   SD1, SD2, SD3], 130), [((0.355285536
                                                                   +0.487112195)/2)])
expect(average_popularity_per_bpm([SD3, SD4], 100), [0.572983739,
                                                            0.284309293])
expect(average_popularity_per_bpm([SD2, SD5, SD6, SD7, SD8], 80), [((0.487112195
                                                                            +0.982437823)/2),
                                                                          (0.793812398
                                                                           +0.827393833
                                                                           +0.578439847)/3])
summary()

##############################################################################################################

start_testing() # Examples and tests for average_single_popularity

expect(average_single_popularity([], 50, 1960), 0.0)
expect(average_single_popularity(LOSD2, 130, 2009), 0.355285536)
expect(average_single_popularity([SD2, SD6], 80, 1973), (0.487112195 + 0.982437823)/2)
expect(average_single_popularity([SD2, SD6], 100, 1973), 0) # no sd passes filter, division by 0
expect(average_single_popularity(LOSD5, 80, 2009), (0.578439847 + 0.827393833 + 0.793812398)/3)

summary()

##############################################################################################################

start_testing() # Examples and tests for sum_popularity

expect(sum_popularity([]), 0)
expect(sum_popularity(LOSD1), 0.355285536)
expect(sum_popularity(LOSD2), (0.355285536 + 0.487112195))
expect(sum_popularity(LOSD3), (0.355285536 + 0.487112195 + 0.572983739))

summary()

##############################################################################################################

start_testing() # Examples and tests for count_list

expect(count_list([]), 0)
expect(count_list(LOSD1), 1)
expect(count_list(LOSD2), 2)
expect(count_list(LOSD3), 3)
expect(count_list(LOSD4), 4)

summary()

##############################################################################################################

start_testing() # Examples and tests for filter_bpm

expect(filter_bpm([], 130), [])
expect(filter_bpm(LOSD1, 130), [SD1])
expect(filter_bpm(LOSD2, 80), [SD2])
expect(filter_bpm(LOSD2, 120), [])
expect(filter_bpm(LOSD3, 100), [SD3])
expect(filter_bpm(LOSD3, 200), [])
expect(filter_bpm(LOSD4, 100), [SD3, SD4])
expect(filter_bpm(LOSD5, 80), LOSD5)

summary()

##############################################################################################################

start_testing() # Examples and tests for is_in_bpm

expect(is_in_bpm(SD1, 130), True)
expect(is_in_bpm(SD1, 80), False)
expect(is_in_bpm(SD2, 80), True)
expect(is_in_bpm(SD2, 100), False)
expect(is_in_bpm(SD3, 100), True)
expect(is_in_bpm(SD3, 90), False)

summary()

##############################################################################################################

start_testing() # Examples and tests for filter_bpm_and_year

expect(filter_bpm_and_year([], 130, 2009), [])
expect(filter_bpm_and_year(LOSD1, 140, 2009), []) # correct year but incorrect bpm
expect(filter_bpm_and_year(LOSD2, 80, 1973), [SD2])
expect(filter_bpm_and_year([SD5, SD1], 130, 2009), [SD1]) # only only one passes
expect(filter_bpm_and_year([SD2, SD6], 80, 1973), [SD2, SD6]) # both pass

summary()

##############################################################################################################

start_testing() # Examples and tests for is_in_year

expect(is_in_year(SD1, 2009), True)
expect(is_in_year(SD1, 2003), False)
expect(is_in_year(SD2, 1973), True)
expect(is_in_year(SD2, 2009), False)
```

```
summary()
```

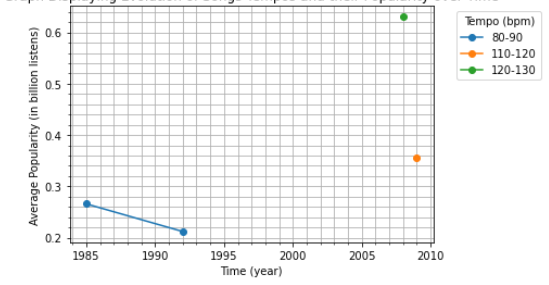
Graph Displaying Evolution of Songs Tempos and their Popularity over Time



Graph Displaying Evolution of Songs Tempos and their Popularity over Time

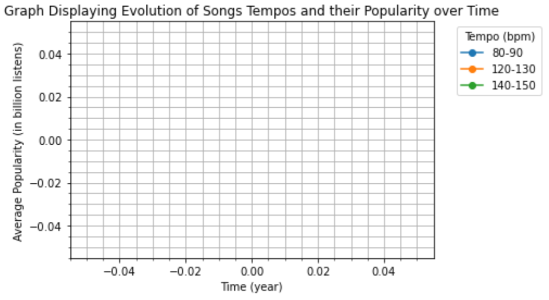


Graph Displaying Evolution of Songs Tempos and their Popularity over Time

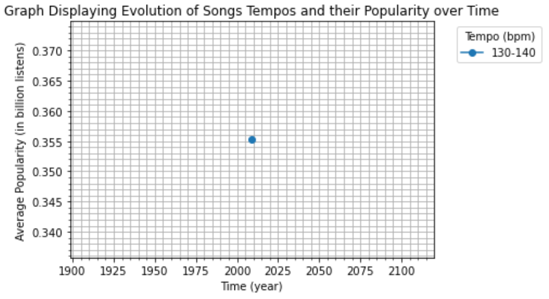


```
3 of 3 tests passed
```

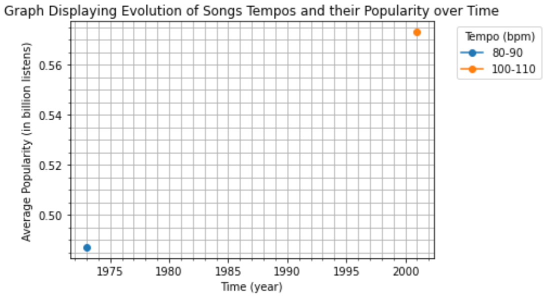
Graph Displaying Evolution of Songs Tempos and their Popularity over Time

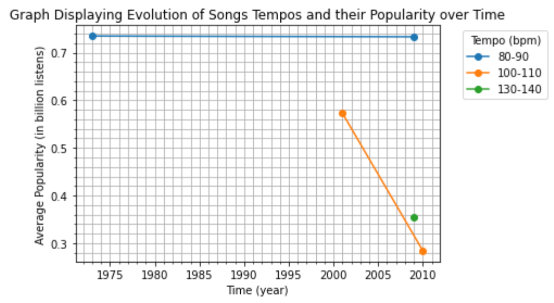


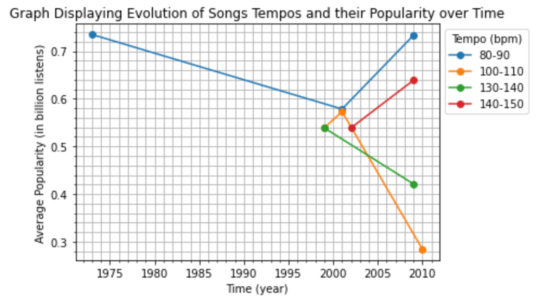
Graph Displaying Evolution of Songs Tempos and their Popularity over Time



Graph Displaying Evolution of Songs Tempos and their Popularity over Time







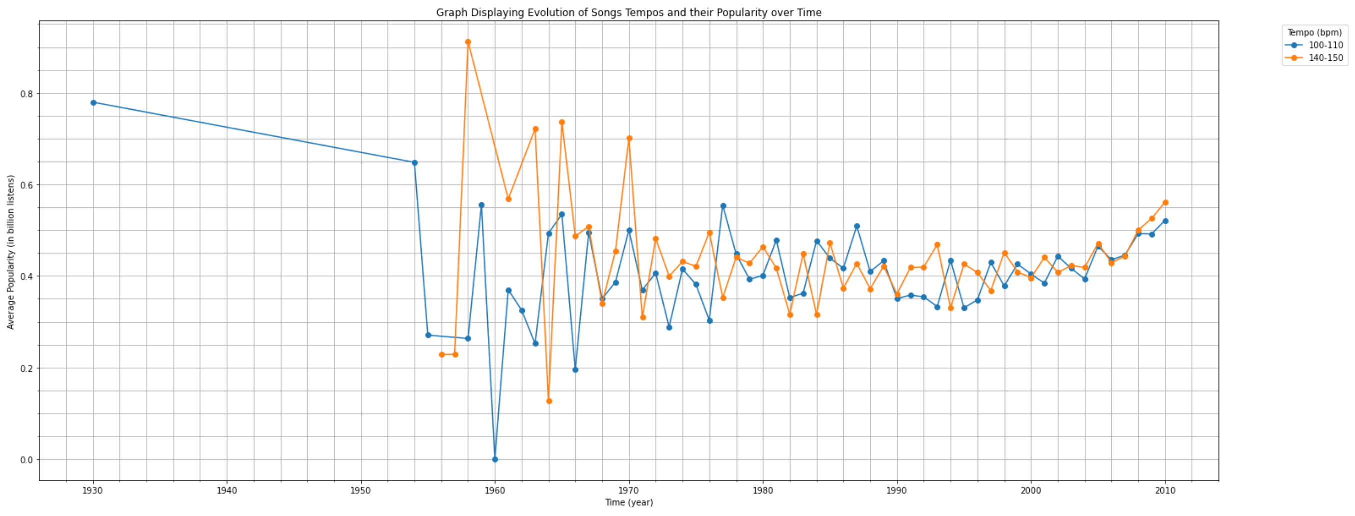
```
5 of 5 tests passed
4 of 4 tests passed
3 of 3 tests passed
8 of 8 tests passed
5 of 5 tests passed
6 of 6 tests passed
7 of 7 tests passed
5 of 5 tests passed
5 of 5 tests passed
4 of 4 tests passed
5 of 5 tests passed
8 of 8 tests passed
6 of 6 tests passed
5 of 5 tests passed
4 of 4 tests passed
```

In [ ]:

Final Graph/Chart

Now that everything is working, you **must** call `main` on the intended information source in order to display the final graph/chart:

In [4]:
```
pyplot.figure(figsize=(25,10))    # increase size of graph

main("songs.csv", [100, 140])
```



In [5]:
```
# Be sure to select ALL THE FILES YOU NEED (including csv's)
# when you submit. As usual, you cannot edit this cell.
# Instead, run this cell to start the submission process.
from cs103 import submit

COURSE = 78012
ASSIGNMENT = 943542 # Final submission

submit(COURSE, ASSIGNMENT)

# If your submission fails, SUBMIT by downloading your files and uploading them
# to Canvas. You can learn how on the page "How to submit your Jupyter notebook"
# on our Canvas site.
```

Token

Files

project_final_submission_template.ipynb
empty.csv
songs.csv
9_songs_small.csv
songs_subset1.csv
6_songs_small.csv
songs_subset2.csv

submit

Please double check your submission on Canvas to ensure that the right files (Jupyter file + CSVs) have been submitted