Christmas Data Science Competition

Author: Hans-Jörg Stark
Date: December 2021

For easier access links to main sections in this document:

Questions to answer:

1. What can you tell me about week position?
2. What can you tell me about weeks on the chart?

Just in case you need to install additional packages - uncomment the leading hash and rund the code!

In [3679]:

```
import sys
#!{sys.executable} -m pip install pandasql
#!{sys.executable} -m pip install statsmodels
#!{sys.executable} -m pip install Pingouin #necessary for Welch's ANOVA
#!{sys.executable} -m pip install seaborn_qqplot
```

In [3680]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3681]:

```
#Load some modules first
import os
import pandas as pd
from pandasql import sqldf
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Part 0: Load the data and do some data-wrangling and quality checks

In [3682]:

```
#in MS Excel I added a new column 'performerGroup' that indicates whether the pe
#or and individual (0)

#load the data
path='/Users/hansjoerg.stark/Library/Mobile Documents/com~apple~CloudDocs/Privat
rawDataFl = os.path.join(path,'christmas_billboard_data.xlsx')
dfXmasSongsRaw = pd.read_excel(rawDataFl)
dfXmasSongsRaw.head(3)

```

Out[3682]:

	url	weekid	week_position	song	performer	performerGroup
0	http://www.billboard.com/charts/hot-100/2000-0...	2000-01-01 00:00:00	53	THIS GIFT	98 Degrees	1
1	http://www.billboard.com/charts/hot-100/2000-0...	2000-08-01 00:00:00	49	THIS GIFT	98 Degrees	1
2	http://www.billboard.com/charts/hot-100/2000-0...	1/15/2000	79	THIS GIFT	98 Degrees	1

*Compute the **season** for each song - some songs with instance > 1 will have more than one season*
The season is the year in which Christmas was - if songs are in January of the following year still on the chart set the season for this record to "year-1" so it matches with the proper season to which it belongs to.

In [3683]:

```
#create date from year,month,day
dfXmasSongsRaw['date'] = pd.to_datetime(dfXmasSongsRaw[['year', 'month', 'day']]
dfXmasSongsRaw['date']
```

Out[3683]:

```
0      2000-01-01
1      2000-01-08
2      2000-01-15
3      2006-12-09
4      2017-01-07
          ...
382    2017-01-07
383    2017-01-14
384    1980-12-27
385    1981-01-03
386    1981-01-10
Name: date, Length: 387, dtype: datetime64[ns]
```

In [3684]:

```
dfXmasSongsRaw['season'] = dfXmasSongsRaw['year']
yearMinusOne4Season = (dfXmasSongsRaw.month==1)
dfXmasSongsRaw.loc[yearMinusOne4Season, 'season']=dfXmasSongsRaw['year']-1
dfXmasSongsRaw[['season','date']][2:5]
```

Out[3684]:

	season	date
2	1999	2000-01-15
3	2006	2006-12-09
4	2016	2017-01-07

Do Some Data Wrangling

In [3685]:

```
#there is a nasty $ in the song column of the 5th dataset - replace $ with S
dfXmasSongsRaw['songid'] = dfXmasSongsRaw['songid'].apply(lambda x: str(x.replac
dfXmasSongsRaw['song'] = dfXmasSongsRaw['song'].apply(lambda x: str(x.replace('$
dfXmasSongsRaw[4:5]
```

Out[3685]:

	url	weekid	week_position	song	performer	performe
4	http://www.billboard.com/charts/hot-100/2017-0...	2017-07-01 00:00:00	48	IT'S THE MOST WONDERFUL TIME OF THE YEAR	Andy Williams	

Get the Progress - backwards or forwards in the charts - and save it as a new variable 'positionChangeWithinWeek'.

In [3686]:

```
dfXmasSongsRaw['positionChangeWithinWeek'] = dfXmasSongsRaw['week_position'] - df
dfXmasSongsWithPositionChange = dfXmasSongsRaw[dfXmasSongsRaw['positionChangeWith
dfXmasSongsWithPositionChange.info()

```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 279 entries, 1 to 386
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   url                       279 non-null    object
 1   weekid                    279 non-null    object
 2   week_position             279 non-null    int64
 3   song                      279 non-null    object
 4   performer                 279 non-null    object
 5   performerGroup            279 non-null    int64
 6   songid                    279 non-null    object
 7   instance                  279 non-null    int64
 8   previous_week_position    279 non-null    float64
 9   peak_position             279 non-null    int64
 10  weeks_on_chart            279 non-null    int64
 11  year                      279 non-null    int64
 12  month                     279 non-null    int64
 13  day                       279 non-null    int64
```

There are 279 records of 387 that have a change in position from a previous week (72.1%).

Compute the number of days between the date of the record and Christmas.
First: define closest Xmas date: for dates in Nov and Dec it is the same year, for dates in Jan it is the previous year.

In [3687]:

```
dfXmasSongsRaw['xmasDate'] = np.where(dfXmasSongsRaw['month'] >10,dfXmasSongsRaw
dfXmasSongsRaw['xmasDate'] = pd.to_datetime(dfXmasSongsRaw['xmasDate'])
```

In [3688]:

```
dfXmasSongsRaw['numberOfDaysToXmas'] = (dfXmasSongsRaw['xmasDate'] - dfXmasSongs
dfXmasSongsRaw[['numberOfDaysToXmas','date']][:6]
```

Out[3688]:

	numberOfDaysToXmas	date
0	-7	2000-01-01
1	-14	2000-01-08
2	-21	2000-01-15
3	16	2006-12-09
4	-13	2017-01-07
5	18	2013-12-07

In [3689]:

```
#get some basic information on the dataset, especially data-types
dfXmasSongsRaw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 387 entries, 0 to 386
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype         
---  ------                     --------------  -----         
 0   url                        387 non-null    object        
 1   weekid                     387 non-null    object        
 2   week_position              387 non-null    int64         
 3   song                       387 non-null    object        
 4   performer                  387 non-null    object        
 5   performerGroup             387 non-null    int64         
 6   songid                     387 non-null    object        
 7   instance                   387 non-null    int64         
 8   previous_week_position     279 non-null    float64       
 9   peak_position              387 non-null    int64         
 10  weeks_on_chart             387 non-null    int64         
 11  year                       387 non-null    int64         
 12  month                      387 non-null    int64         
 13  day                        387 non-null    int64         
 14  date                       387 non-null    datetime64[ns]
 15  season                     387 non-null    int64         
 16  positionChangeWithinWeek   279 non-null    float64       
 17  xmasDate                   387 non-null    datetime64[ns]
 18  numberOfDaysToXmas         387 non-null    int64         
dtypes: datetime64[ns](2), float64(2), int64(10), object(5)
memory usage: 57.6+ KB
```

In [3690]:

```
#how many unique songs are there? --> 70
print(len(dfXmasSongsRaw['song'].unique()))
uniqueSongs=dfXmasSongsRaw['song'].unique()
uniqueSongs
```

```
70
```

Out[3690]:

```
array(['THIS GIFT', 'GREATEST TIME OF YEAR',
       "IT'S THE MOST WONDERFUL TIME OF THE YEAR", 'LAST CHRISTMAS',
       'SANTA TELL ME', "DO THEY KNOW IT'S CHRISTMAS?", 'WHITE CHRISTM
AS',
       'SILENT NIGHT', "MONSTERS' HOLIDAY", 'CHRISTMAS AULD LANG SYN
E',
       'CHILD OF GOD', 'JINGLE BELL ROCK', 'LET IT SNOW',
       "ROCKIN' AROUND THE CHRISTMAS TREE", 'THIS TIME OF THE YEAR',
       'BELIEVE', 'A HOLLY JOLLY CHRISTMAS',
       'PLEASE COME HOME FOR CHRISTMAS', 'THIS CHRISTMAS',
       'RUN RUDOLPH RUN', 'MISTLETOE', 'CHRISTMAS LIGHTS',
       "BABY'S FIRST CHRISTMAS", 'SAME OLD LANG SYNE',
       'THE HAPPY REINDEER',
       "THE CHIPMUNK SONG (CHRISTMAS DON'T BE LATE)",
       'GRANDMA GOT RUN OVER BY A REINDEER', 'WINTER WORLD OF LOVE',
       'WHERE ARE YOU CHRISTMAS?', 'WELCOME CHRISTMAS',
       "BABY, IT'S COLD OUTSIDE", 'BETTER DAYS',
       'I BELIEVE IN FATHER CHRISTMAS', 'MY FAVORITE THINGS',
       "SANTA'S A FAT BITCH", "IT DOESN'T HAVE TO BE THAT WAY",
       'CHRISTMAS FOR COWBOYS', 'FELIZ NAVIDAD',
       "I'LL BE HOME FOR CHRISTMAS", 'UNDERNEATH THE TREE',
       'AULD LANG SYNE', 'THE GREATEST GIFT OF ALL', 'MACARENA CHRISTM
AS',
       'ALL I WANT FOR CHRISTMAS IS YOU', 'OH SANTA!',
       'IF WE MAKE IT THROUGH DECEMBER',
       "IT'S BEGINNING TO LOOK A LOT LIKE CHRISTMAS",
       'WHEN A CHILD IS BORN',
       'THE CHRISTMAS SONG (MERRY CHRISTMAS TO YOU)',
       "THIS ONE'S FOR THE CHILDREN", 'THE CHRISTMAS SHOES',
       'CHRISTMAS DREAM', 'SANTA CLAUS IS WATCHING YOU', 'PRETTY PAPE
R',
       'HAVE YOURSELF A MERRY LITTLE CHRISTMAS', "TWISTIN' BELLS",
       'RIVER', 'DECK THE HALLS', 'GREEN CHRISTMAS', 'BLUE CHRISTMAS',
       'THE MARVELOUS TOY', 'HAPPY XMAS (WAR IS OVER)', 'AMEN',
       'A GREAT BIG SLED', "I'LL BE HOME",
       'WHAT CAN YOU GET A WOOKIEE FOR CHRISTMAS (WHEN HE ALREADY OWNS
A COMB?)',
       'THE SANTA CLAUS BOOGIE', 'SHAKE UP CHRISTMAS', 'LITTLE ALTAR B
OY',
       'MERRY CHRISTMAS IN THE NFL'], dtype=object)
```

In [3691]:

```
#how many unique artists are there? --> 69
print(len(dfXmasSongsRaw['performer'].unique()))
uniquePerformer=dfXmasSongsRaw['performer'].unique()
uniquePerformer
```

```
69
```

Out[3691]:

```
array(['98 Degrees', 'Aly & AJ', 'Andy Williams', 'Ariana Grande',
       'Band-Aid', 'Bing Crosby',
       'Bobby "Boris" Pickett And The Crypt-Kickers', 'Bobby Darin',
       'Bobby Helms', 'Bobby Rydell/Chubby Checker', 'Boyz II Men',
       'Brenda Lee', 'Brook Benton', 'Brooks & Dunn', 'Burl Ives',
       'Charles Brown', 'Chris Brown', 'Chuck Berry', 'Colbie Cailla
t',
       'Coldplay', 'Connie Francis', 'Dan Fogelberg',
       'Dancer, Prancer And Nervous', 'David Seville And The Chipmunk
s',
       'Eagles', 'Elmo & Patsy', 'Engelbert Humperdinck', 'Faith Hil
l',
       'Glee Cast', 'Goo Goo Dolls', 'Greg Lake',
       'Herb Alpert & The Tijuana Brass', 'Insane Clown Posse',
       'Jim Croce', 'John Denver', 'Jose Feliciano', 'Justin Bieber',
       'Kelly Clarkson', 'Kenny G', 'Kenny Rogers', 'Los Del Rio',
       'Mariah Carey', 'Merle Haggard', 'Michael Buble', 'Michael Hol
m',
       'Nat King Cole', 'New Kids On The Block', 'NewSong', 'Perry Com
o',
       'Ray Stevens', 'Roy Orbison', 'Sam Smith', 'Santo & Johnny',
       'Sarah McLachlan', 'SHeDAISY', 'Stan Freberg',
       'The Browns Featuring Jim Edward Brown', 'The Chad Mitchell Tri
o',
       'The Drifters Featuring Clyde McPhatter And Bill Pinkney',
       'The Fray', 'The Impressions',
       'The Killers Featuring Toni Halliday', 'The Platters',
       'The Star Wars Intergalactic Droid Choir & Chorale',
       'The Tractors', 'Train', 'Vic Dana', 'Wham!',
       'Willis "The Guard" & Vigorish'], dtype=object)
```

In [3692]:

```
#how many unique combinations of songs and artists are there? --> 78
print(len(dfXmasSongsRaw['songid'].unique()))

```

```
78
```

In [3693]:

```
sqlUniqueSongs = sqldf("select distinct performer, song from dfXmasSongsRaw orde
sqlUniqueSongs

```

	performer	song
0	98 Degrees	THIS GIFT
1	Aly & AJ	GREATEST TIME OF YEAR
2	Andy Williams	IT'S THE MOST WONDERFUL TIME OF THE YEAR
3	Ariana Grande	LAST CHRISTMAS
4	Ariana Grande	SANTA TELL ME
...	...	...
73	The Tractors	THE SANTA CLAUS BOOGIE
74	Train	SHAKE UP CHRISTMAS
75	Vic Dana	LITTLE ALTAR BOY
76	Wham!	LAST CHRISTMAS
77	Willis "The Guard" & Vigorish	MERRY CHRISTMAS IN THE NFL

In [3694]:

```
#check if there are entries without a perfomer
dfXmasSongsRaw[dfXmasSongsRaw['song'].isna()]
```

Out[3694]:

	url	weekid	week_position	song	performer	performerGroup	songid	instance	previous_weel

In [3695]:

```
#quality check
sqlCheck = sqldf("""select songid,song,performer,
                    peak_position as PeakPosition,
                    min(week_position) as MinWeekPosition,
                    peak_position - min(week_position) as delta
                    from dfXmasSongsRaw
                    group by songid
                    having peak_position - min(week_position) != 0
                    order by delta desc;""")
sqlCheck
```

Out[3695]:

	songid	song	performer	PeakPosition	MinWeekPosition	delta
0	Same Old Lang SyneDan Fogelberg	SAME OLD LANG SYNE	Dan Fogelberg	9	14	-5
1	Better DaysGoo Goo Dolls	BETTER DAYS	Goo Goo Dolls	36	48	-12
2	BelieveBrooks & Dunn	BELIEVE	Brooks & Dunn	60	86	-26

--> data anomalies: for these 3 songs the min(week-Position) and the peak-position are not consistent!!

Part 1: Investigation on Week Position

What were the songs with the most successful advance in the charts?

Assumption: success = largest interval from worst to best week-position per season

In [3696]:

```
sqlBest1 = sqldf("""select song,performer,season,year||"-"||month||"-"||day as t
                    min(week_position) as MinWeekPosition,
                    max(week_position) as MaxWeekPosition,
                    max(week_position)-min(week_position) as weekPositionRange,
                    avg(week_position) as AverageWeekPosition
                    from dfXmasSongsRaw
                group by songid,season order by weekPositionRange desc limit 10;"
sqlBest1

```

Out[3696]:

	song	performer	season	timestamp	MinWeekPosition	MaxWeekPosition	weekPosi
0	AMEN	The Impressions	1964	1964-11-21	7	96	
1	MISTLETOE	Justin Bieber	2011	2011-12-3	11	96	
2	AULD LANG SYNE	Kenny G	1999	1999-12-25	7	89	
3	THIS ONE'S FOR THE CHILDREN	New Kids On The Block	1989	1989-11-11	7	82	
4	IF WE MAKE IT THROUGH DECEMBER	Merle Haggard	1973	1973-11-24	28	97	
5	WINTER WORLD OF LOVE	Engelbert Humperdinck	1969	1969-12-6	16	84	
6	SAME OLD LANG SYNE	Dan Fogelberg	1980	1980-12-13	14	75	
7	PLEASE COME HOME FOR CHRISTMAS	Eagles	1978	1978-12-9	18	78	
8	THE HAPPY REINDEER	Dancer, Prancer And Nervous	1959	1960-1-9	34	93	
9	CHRISTMAS LIGHTS	Coldplay	2010	2011-1-1	25	83	

Assumption: success = best week-position ever during their presence in the charts - regardless of season.

In [3697]:

```
sqlBest2 = sqldf("""select song,performer,year||"-"||month||"-"||day as timestam
                    peak_position as PeakPosition,
                    weeks_on_chart as WeeksOnChart,
                    min(week_position) as MinWeekPosition,
                    max(week_position) as MaxWeekPosition,
                    max(week_position)-min(week_position) as weekPositionRange,
                    avg(week_position) as AverageWeekPosition
                    from dfXmasSongsRaw
                group by songid order by peak_position asc limit 10;""")
sqlBest2
```

1	AULD LANG SYNE	Kenny G	1999-12-25	7	5	7	89
2	THIS ONE'S FOR THE CHILDREN	New Kids On The Block	1989-11-11	7	16	7	82
3	SAME OLD LANG SYNE	Dan Fogelberg	1980-12-13	9	18	14	75
4	ALL I WANT FOR CHRISTMAS IS YOU	Mariah Carey	2000-1-8	11	19	11	83
5	MISTLETOE	Justin Bieber	2011-12-3	11	10	11	96
6	WHITE CHRISTMAS	Bing Crosby	1958-12-20	12	13	12	86
7	DO THEY KNOW IT'S CHRISTMAS?	Band-Aid	1984-12-22	13	9	13	65

Analysis on whether the average week-position is better before or after Christmas

Analysis on the songs position before or after Christmas

In [3698]:

```
#Get the months songs are in the charts
sqlMonthsOfPresenceInCharts = sqldf("select DISTINCT month from dfXmasSongsRaw c
sqlMonthsOfPresenceInCharts
```

Out[3698]:

	month
0	1
1	11
2	12

In [3699]:

```
#Check if a song has better position before Christmas
sqlBetterBeforeChristmas = sqldf("select count(*) as CountOfBetterBeforeXmas fro
sqlBetterBeforeChristmas
```

Out[3699]:

	CountOfBetterBeforeXmas
0	81

In [3700]:

```

untOfBetterAfterXmas from dfXmasSongsRaw where positionChangeWithinWeek<0 and (monti


```

Out[3700]:

	CountOfBetterAfterXmas
0	119

In [3701]:

```
#get all songs with their initial rank
dfXmasSongsRaw['initialStartPosition'] = dfXmasSongsRaw['previous_week_position']
dfXmasSongsRaw['initialStartPosition'][:5]
```

Out[3701]:

```
0    53
1     0
2     0
3    96
4    48
Name: initialStartPosition, dtype: int64
```

In [3702]:

```
sqlPerformerSongRanking = sqldf("""select distinct performer, song, min(initials
                as initialPosition from dfXmasSongsRaw where initialStartPositior
                group by songid order by performer;""")
sqlPerformerSongRanking
```

Out[3702]:

	performer	song	initialPosition
0	98 Degrees	THIS GIFT	53
1	Aly & AJ	GREATEST TIME OF YEAR	96
2	Andy Williams	IT'S THE MOST WONDERFUL TIME OF THE YEAR	48
3	Ariana Grande	LAST CHRISTMAS	96
4	Ariana Grande	SANTA TELL ME	65
...	...	...	...
72	The Tractors	THE SANTA CLAUS BOOGIE	91
73	Train	SHAKE UP CHRISTMAS	99
74	Vic Dana	LITTLE ALTAR BOY	99
75	Wham!	LAST CHRISTMAS	50
76	Willis "The Guard" & Vigorish	MERRY CHRISTMAS IN THE NFL	82

77 rows × 3 columns

In [3703]:

```
meanInitialPosition = sqldf("select avg(initialPosition) as averageInitialPositi
meanInitialPosition['averageInitialPosition']


```

Out[3703]:

```
0    75.25974
Name: averageInitialPosition, dtype: float64
```

In [3704]:

```
meanOverallPosition = sqldf("select avg(week_position) as averageOverallPositior
meanOverallPosition['averageOverallPosition']

```

Out[3704]:

```
0    57.204134
Name: averageOverallPosition, dtype: float64
```

This means that on average the inital position of a song is worse or higher than the overall average position of all songs during their presence in the charts.

Checking the median instead of average

In [3705]:

```
dfInitialPostBySongId = sqldf("select Distinct songid,initialPosition from (sele
dfInitialPostBySongId['initialPosition'].median()
```

Out[3705]:

```
82.0
```

In [3706]:

```
dfPostBySongId = sqldf("select week_position from dfXmasSongsRaw where week_posi
dfPostBySongId['week_position'].median()
```

Out[3706]:

```
58.0
```

In [3707]:

```
dfXmasSongsRaw['week_position'].describe()
```

Out[3707]:

```
count    387.000000
mean      57.204134
std       25.398527
min        7.000000
25%       38.500000
50%       58.000000
75%       78.000000
max      100.000000
Name: week_position, dtype: float64
```

In [3708]:

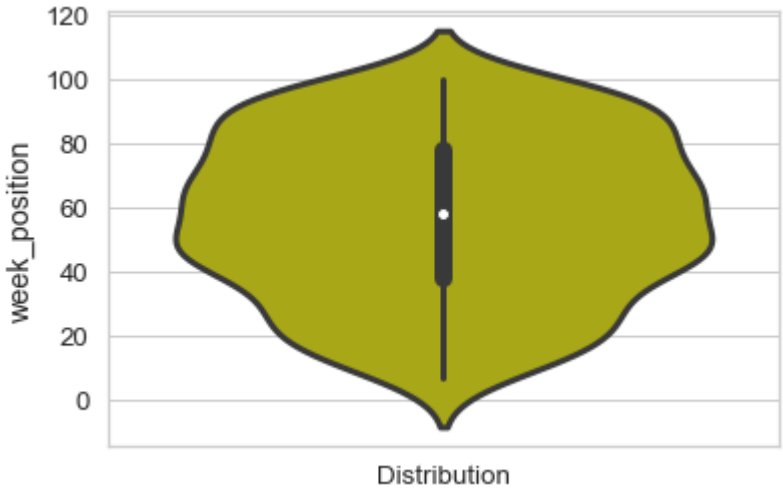
```
ax = dfXmasSongsRaw['week_position'].plot(kind='hist', bins=50, alpha=0.2, color
ax.set_xlabel("Week Position")
ax.set_ylabel("Frequency")
Text(0.5, 0, 'Week Position')
```

Out[3708]:

```
Text(0, 0.5, 'Frequency')
```

In [3709]:

```
#Distribution and their shape of the values of week-position of the songs
ax = sns.violinplot(y=dfXmasSongsRaw['week_position'], color='y', )
ax = ax.set_xticklabels(['Distribution'])

```



Influence of week-position before or after Christmas

Null Hypthesis H0 = ***The average week-position is independent of whether the week-position is before or after Christmas. i.e. the two group means of the groups – before and after Christmas – are the same.***

Run a t-test

In [3710]:

```
#Datawrangling: variable that indicateds week-position before (1) or after(0) Ch
dfXmasSongsRaw['BeforeXmas'] = np.where(dfXmasSongsRaw.numberOfDaysToXmas>=0,1,0
dfXmasSongsRaw[['BeforeXmas','date']][:6]
```

Out[3710]:

	BeforeXmas	date
0	0	2000-01-01
1	0	2000-01-08
2	0	2000-01-15
3	1	2006-12-09
4	0	2017-01-07
5	1	2013-12-07

In [3711]:

```
#Split the dataset
dfBeforeXmas = dfXmasSongsRaw.loc[dfXmasSongsRaw['BeforeXmas'] == 1, 'week_posit
dfAfterXmas = dfXmasSongsRaw.loc[dfXmasSongsRaw['BeforeXmas'] == 0, 'week_positi
from scipy import stats as st
st.ttest_ind(a=dfBeforeXmas, b=dfAfterXmas, equal_var=True)
```

Out[3711]:

```
Ttest_indResult(statistic=6.039361586928741, pvalue=3.6477473502018717
e-09)
```

The result of the t-test is

```
Ttest_indResult(statistic=6.039361586928741, pvalue=3.6477473502018717e-09)
```

p-value = 3.6477e-09 which is <<0.05 --> the test is significant!

H0 is being rejected. The average week-position before Christmas differs significantly from the average week-position after Christmas!

In [3712]:

```
#get the mean values:
print("Mean Week-Position before Xmas: {}, \nMean Week-Position after Xmas: {}".
```

```
Mean Week-Position before Xmas: 65.5906432748538,
Mean Week-Position after Xmas: 50.56481481481482
```

Analysis on whether the Song contains "Christmas" in its title

Does the song contain "Christmas" in its title?

Null Hypthesis H0 = ***The average position of a song is not dependant on whether the songtitle contains "Christmas".***

In [3713]:

```
#add a column that indicates whether the song contains the word "Christmas"
#dfXmasSongsRaw.loc[dfXmasSongsRaw['stream'] == 2, ['feat','another_feat']] = 'a
dfXmasSongsRaw['xmasInSong'] = dfXmasSongsRaw['song'].str.contains('CHRISTMAS')
dfXmasSongsRaw[4:7]
```

Out[3713]:

	url	weekid	week_position	song	performer	perfor
4	http://www.billboard.com/charts/hot-100/2017-0...	2017-07-01 00:00:00	48	IT'S THE MOST WONDERFUL TIME OF THE YEAR	Andy Williams	
5	http://www.billboard.com/charts/hot-100/2013-1...	2013-07-12 00:00:00	96	LAST CHRISTMAS	Ariana Grande	
6	http://www.billboard.com/charts/hot-100/2014-1...	12/13/2014	65	SANTA TELL ME	Ariana Grande	

3 rows × 22 columns

In [3714]:

```
#Count the number of songs that have or have not 'Christmas' in their title
dfXmasSongsRaw.xmasInSong.value_counts()
```

Out[3714]:

```
False    234
True     153
Name: xmasInSong, dtype: int64
```

In [3715]:

```
sqlUniqueSongsWithChristmasInTitle = sqldf("select distinct songid from dfXmasSo
sqlUniqueSongsWithChristmasInTitle.count()
```

Out[3715]:

```
songid    34
dtype: int64
```

In [3716]:

```
sqlUniqueSongsWithoutChristmasInTitle = sqldf("select distinct songid from dfXma
sqlUniqueSongsWithoutChristmasInTitle.count()
```

Out[3716]:

```
songid    44
dtype: int64
```

In [3717]:

```
#split dataframe into 2 parts: one that contains songs that contain "Christmas"
#the other part that contains songst that do not have "Christmas" in their title
containsXmasInSongname = dfXmasSongsRaw[dfXmasSongsRaw['xmasInSong']]
containsNotXmasInSongname = dfXmasSongsRaw[dfXmasSongsRaw['xmasInSong']==False]
print(containsXmasInSongname.shape)
print(containsNotXmasInSongname.shape)
```

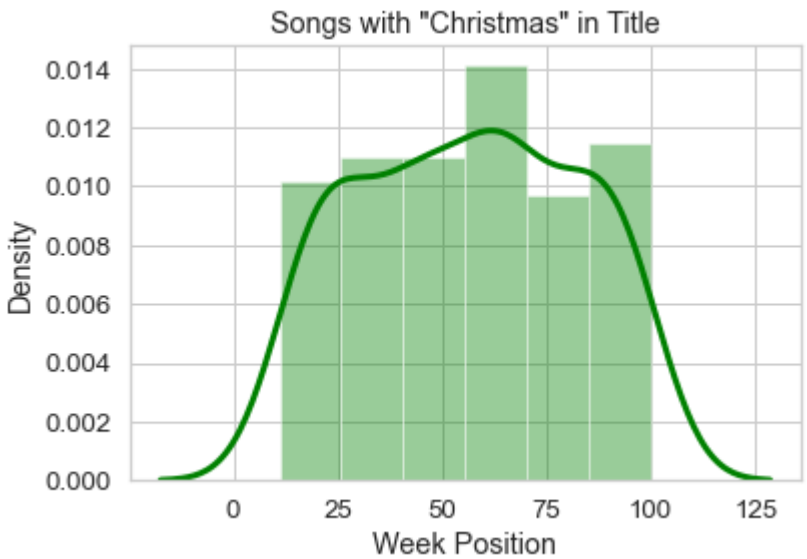
```
(153, 22)
(234, 22)
```

In [3718]:

```
#Histogram of week-position from Songs that contain 'Christmas' in their title
ax = sns.distplot(containsXmasInSongname['week_position'], color = 'green')
ax.set(xlabel='Week Position', ylabel='Density', title='Songs with "Christmas" i

```

Out[3718]:

```
[Text(0.5, 0, 'Week Position'),
 Text(0, 0.5, 'Density'),
 Text(0.5, 1.0, 'Songs with "Christmas" in Title')]
```

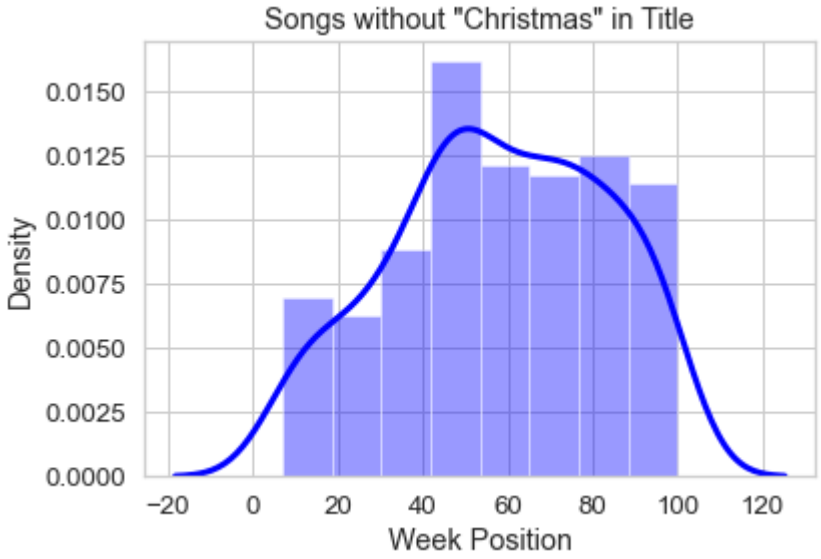


In [3719]:

```
#Histogram of week-position from Songs that do not contain 'Christmas' in their
ax = sns.distplot(containsNotXmasInSongname['week_position'], color='blue')
ax.set(xlabel='Week Position', ylabel='Density', title='Songs without "Christmas
```

Out[3719]:

```
[Text(0.5, 0, 'Week Position'),
 Text(0, 0.5, 'Density'),
 Text(0.5, 1.0, 'Songs without "Christmas" in Title')]
```



In [3720]:

```
# t-test for independent samples
from math import sqrt
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from scipy.stats import sem
from scipy.stats import t

# function for calculating the t-test for two independent samples
def independent_ttest(data1, data2, alpha):
    # calculate means
    mean1, mean2 = mean(data1), mean(data2)
    # calculate standard errors
    se1, se2 = sem(data1), sem(data2)
    # standard error on the difference between the samples
    sed = sqrt(se1**2.0 + se2**2.0)
    # calculate the t statistic
    t_stat = (mean1 - mean2) / sed
    # degrees of freedom
    df = len(data1) + len(data2) - 2
    # calculate the critical value
    cv = t.ppf(1.0 - alpha, df)
    # calculate the p-value
    p = (1.0 - t.cdf(abs(t_stat), df)) * 2.0
    # return everything
    return t_stat, df, cv, p

# calculate the t test
data1 = containsXmasInSongname['week_position']
data2 = containsNotXmasInSongname['week_position']
alpha = 0.05
t_stat, df, cv, p = independent_ttest(data1, data2, alpha)
print('t=%.3f, df=%d, cv=%.3f, p=%.3f' % (t_stat, df, cv, p))
# interpret via critical value
if abs(t_stat) <= cv:
    print('Accept null hypothesis H0 that the means are equal.')
else:
    print('Reject the null hypothesis H0 that the means are equal.')
# interpret via p-value
if p > alpha:
    print('Accept null hypothesis H0 that the means are equal.')
else:
    print('Reject the null hypothesis H0 that the means are equal.')
```

```
t=-0.569, df=385, cv=1.649, p=0.570
Accept null hypothesis H0 that the means are equal.
Accept null hypothesis H0 that the means are equal.
```

Perform an independent t-test to check whether the average position of a song is significantly different if the song contains or does not contain "Christmas" in the title.

H0 = the average position of a song is not dependant on whether the songtitle contains "Christmas" or not.

(Source: https://machinelearningmastery.com/how-to-code-the-students-t-test-from-scratch-in-python/)

Does the Performer - individual or group - have a significant influence on week-position?

Null Hypthesis H0 = ***Whether a song is performed by an individual or a group does not have an influence on the week-position in the chart.***

In [3721]:

```
sqlGroupPerformer = sqldf("select distinct songid from dfXmasSongsRaw where perf
sqlGroupPerformer.count()
```

Out[3721]:

```
songid    33
dtype: int64
```

In [3722]:

```
sqlIndividualPerformer = sqldf("select distinct songid from dfXmasSongsRaw where
sqlIndividualPerformer.count()
```

Out[3722]:

```
songid    45
dtype: int64
```

In [3723]:

```
#split dataframe into 2 parts: one that contains songs performed by a group,
#the other part performed by individuals
groupPerformer = dfXmasSongsRaw[dfXmasSongsRaw['performerGroup']==1]
individualPerformer = dfXmasSongsRaw[dfXmasSongsRaw['performerGroup']==0]
print(groupPerformer.shape)
print(individualPerformer.shape)
```

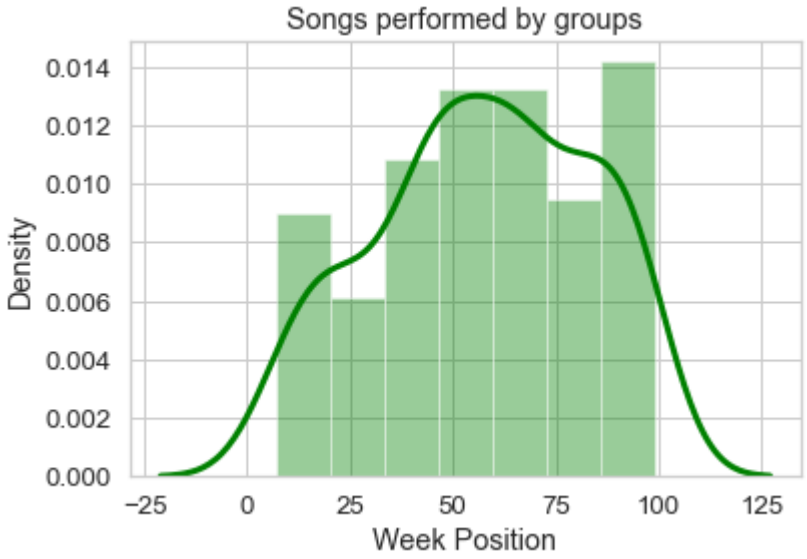
```
(161, 22)
(226, 22)
```

In [3724]:

```
#Histogram of week-position from Songs performed by groups
ax = sns.distplot(groupPerformer['week_position'], color = 'green')
ax.set(xlabel='Week Position', ylabel='Density', title='Songs performed by group
```

Out[3724]:

```
[Text(0.5, 0, 'Week Position'),
 Text(0, 0.5, 'Density'),
 Text(0.5, 1.0, 'Songs performed by groups')]
```

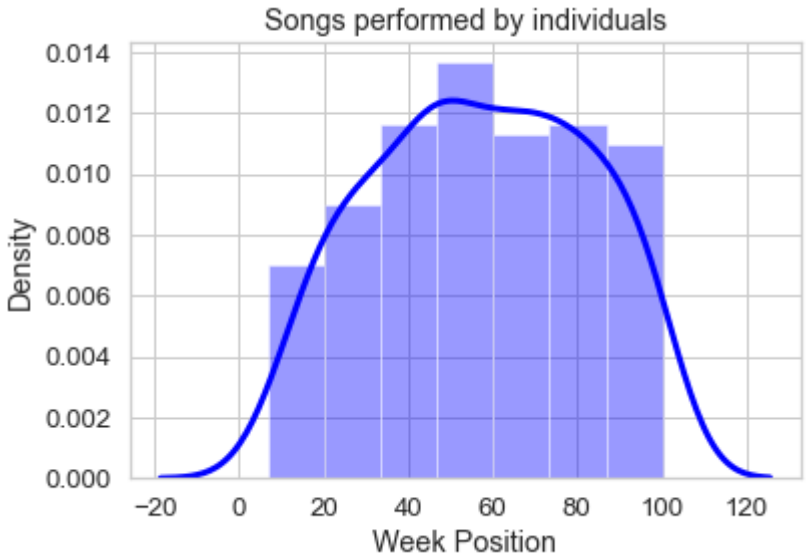


In [3725]:

```
#Histogram of week-position from Songs performed by individuals
ax = sns.distplot(individualPerformer['week_position'], color = 'blue')
ax.set(xlabel='Week Position', ylabel='Density', title='Songs performed by indiv
```

Out[3725]:

```
[Text(0.5, 0, 'Week Position'),
 Text(0, 0.5, 'Density'),
 Text(0.5, 1.0, 'Songs performed by individuals')]
```



In [3726]:

```
# t-test for independent samples
# calculate the t test
data1 = groupPerformer['week_position']
data2 = individualPerformer['week_position']
alpha = 0.05
t_stat, df, cv, p = independent_ttest(data1, data2, alpha)
print('t=%.3f, df=%d, cv=%.3f, p=%.3f' % (t_stat, df, cv, p))
# interpret via critical value
if abs(t_stat) <= cv:
    print('Accept null hypothesis H0 that the means are equal.')
else:
    print('Reject the null hypothesis H0 that the means are equal.')
# interpret via p-value
if p > alpha:
    print('Accept null hypothesis H0 that the means are equal.')
else:
    print('Reject the null hypothesis H0 that the means are equal.')
```

```
t=0.049, df=385, cv=1.649, p=0.961
Accept null hypothesis H0 that the means are equal.
Accept null hypothesis H0 that the means are equal.
```

p-value = 0.961 --> H0 is accepted: **There is no indication that the performer has a signifcant influence on the week-position of a song in the chart.**

Influence of Time to Christmas on Week Position

Check if the closeness (timewise) to christmas has an influence of week-position

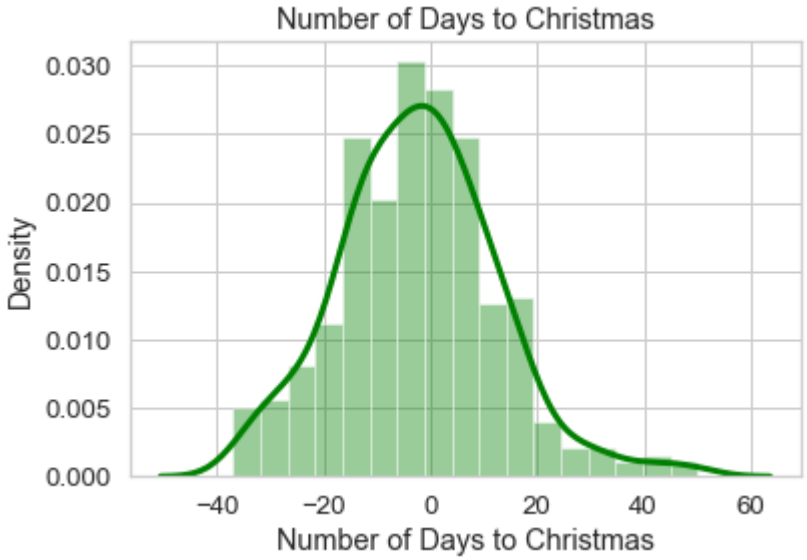
Null Hypthesis H0 = ***The time-distance to Christmas has no significant influence on the week-position of a song in the charts.***

In [3727]:

```
#Histogram of the number of days to Christmas
ax = sns.distplot(dfXmasSongsRaw['numberOfDaysToXmas'], color='green')
ax.set(xlabel='Number of Days to Christmas', ylabel='Density', title='Number of
```

Out[3727]:

```
[Text(0.5, 0, 'Number of Days to Christmas'),
 Text(0, 0.5, 'Density'),
 Text(0.5, 1.0, 'Number of Days to Christmas')]
```



The distribution looks fairly normally distributed!

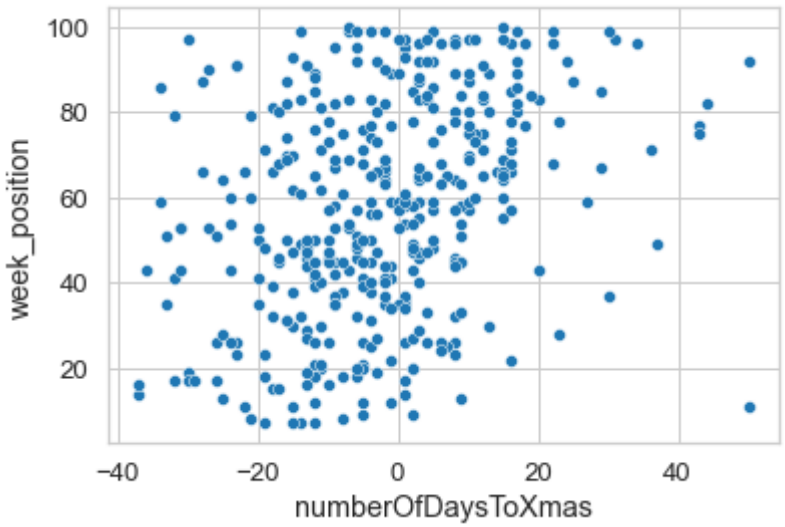
Create a Scatterplot on number of days to Christmas vs Week-Position:

In [3728]:

```
sns.scatterplot(data=dfXmasSongsRaw, x='numberOfDaysToXmas', y='week_position')
```

Out[3728]:

```
<AxesSubplot:xlabel='numberOfDaysToXmas', ylabel='week_position'>
```



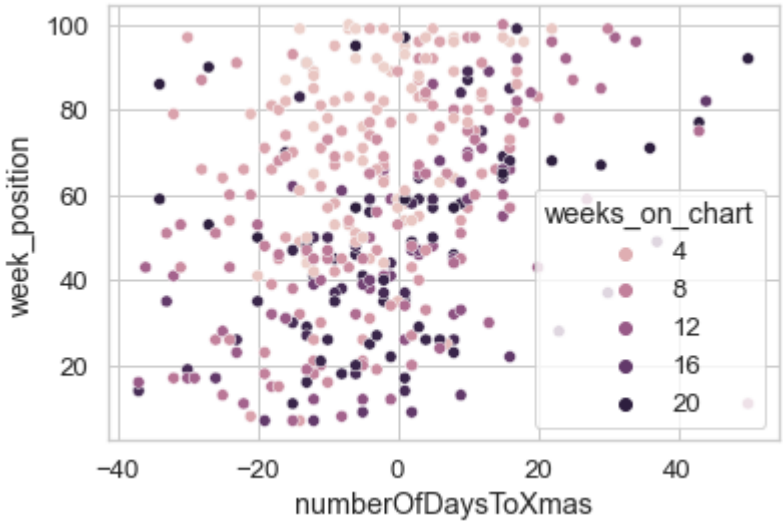
Another Scatter-Plot with emphasis on how long the song was on the charts:

In [3729]:

```
1 sns.scatterplot(data=dfXmasSongsRaw, x='numberOfDaysToXmas', y='week_position',h
```

Out[3729]:

```
<AxesSubplot:xlabel='numberOfDaysToXmas', ylabel='week_position'>
```



Run a simple linear regression on number of days to Christmas (=IV, continuous) versus week-position (=DV, continuous).

Null Hypothesis H0: **The time-distance to Christmas has no significant influence on the week-position of a song in the charts.**

In [3730]:

```
import statsmodels.api as sm
import statsmodels.stats.api as sms
from scipy.stats import boxcox

#extract only necessary data into a separate dataframe
dfNoD2XvsWP = dfXmasSongsRaw[['numberOfDaysToXmas','week_position']]
dfNoD2XvsWP
```

Out[3730]:

	numberOfDaysToXmas	week_position
0	-7	53
1	-14	49
2	-21	79
3	16	96
4	-13	48
...	...	...
382	-13	50
383	-20	41
384	-2	82
385	-9	82
386	-16	82

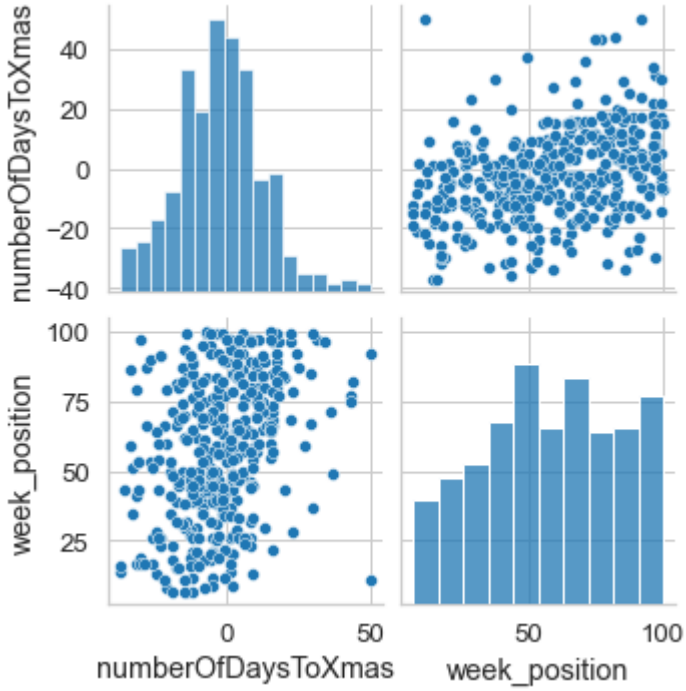
387 rows × 2 columns

In [3731]:

```
1 sns.pairplot(dfNoD2XvsWP)
```

Out[3731]:

```
<seaborn.axisgrid.PairGrid at 0x7fb9ff871220>
```

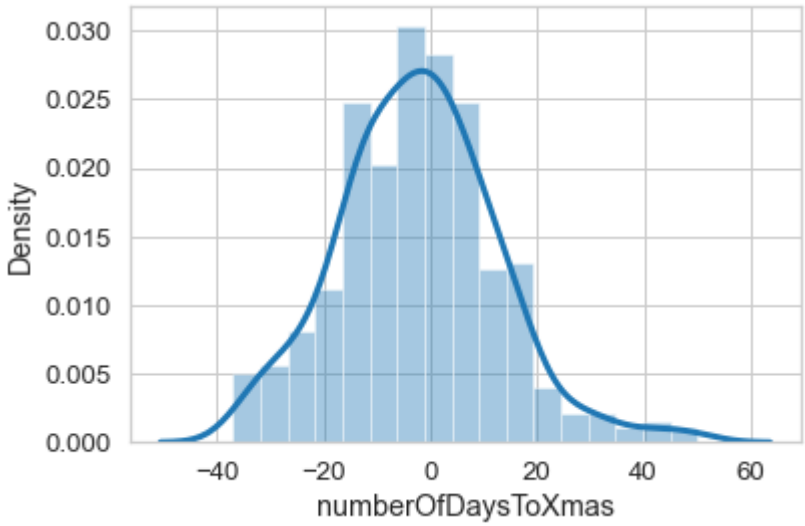


In [3732]:

```
1 sns.distplot(dfNoD2XvsWP['numberOfDaysToXmas'])
```

Out[3732]:

```
<AxesSubplot:xlabel='numberOfDaysToXmas', ylabel='Density'>
```

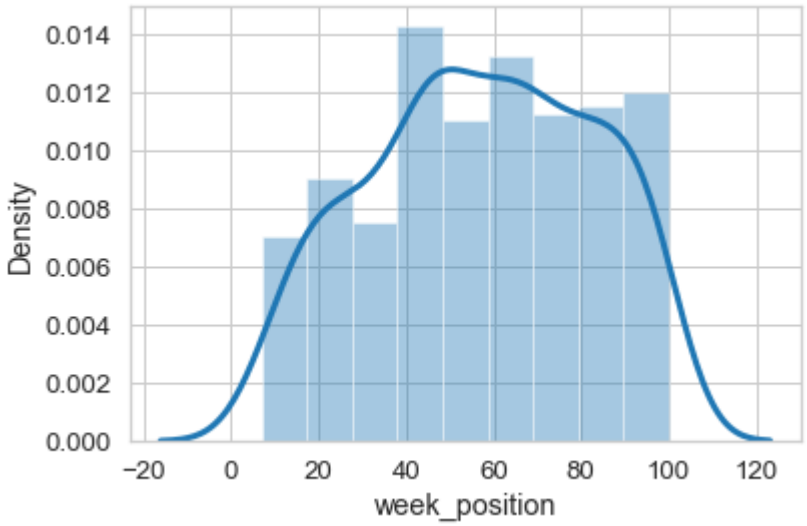


In [3733]:

```
sns.distplot(dfNoD2XvsWP['week_position'])
```

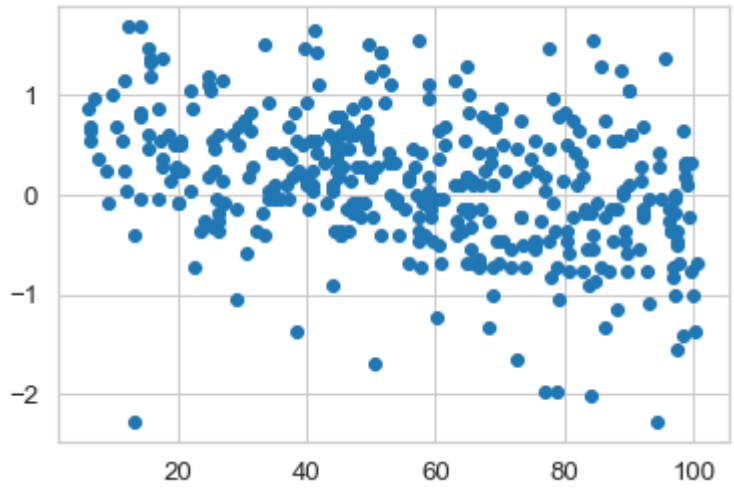
Out[3733]:

```
<AxesSubplot:xlabel='week_position', ylabel='Density'>
```



In [3734]:

```
#Testing for Homoscedasticity
x = dfNoD2XvsWP['numberOfDaysToXmas']
y = dfNoD2XvsWP['week_position']
model = sm.OLS(y,x).fit()
pred_val = model.fittedvalues.copy()
true_val = dfNoD2XvsWP['week_position'].values.copy()
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6, 4))
_ = ax.scatter(residual, pred_val)
```



In [3735]:

```
sms.diagnostic.het_breuschpagan(residual, dfNoD2XvsWP[['numberOfDaysToXmas']])
# lagrange multiplier statistic: 2.485
# p-value for the lagrange multiplier statistic: nan
# F value to test for homoscedasticity: 2.494 - looks good
# p-value for test for homoscedasticity: 0.115 --> not significant, no violation
```

Out[3735]:

```
(2.4848718028584504, nan, 2.4944675659460818, 0.11506609039502463)
```

In [3736]:

```
# Testing for Multicollinearity:
dfNoD2XvsWP.corr()
```

Out[3736]:

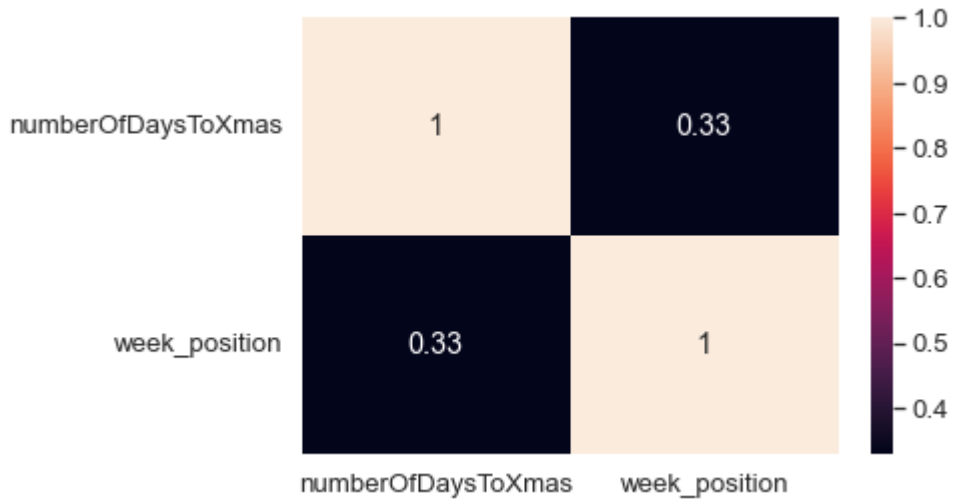
	numberOfDaysToXmas	week_position
numberOfDaysToXmas	1.000000	0.329601
week_position	0.329601	1.000000

In [3737]:

```
sns.heatmap(dfNoD2XvsWP.corr(), annot=True)
```

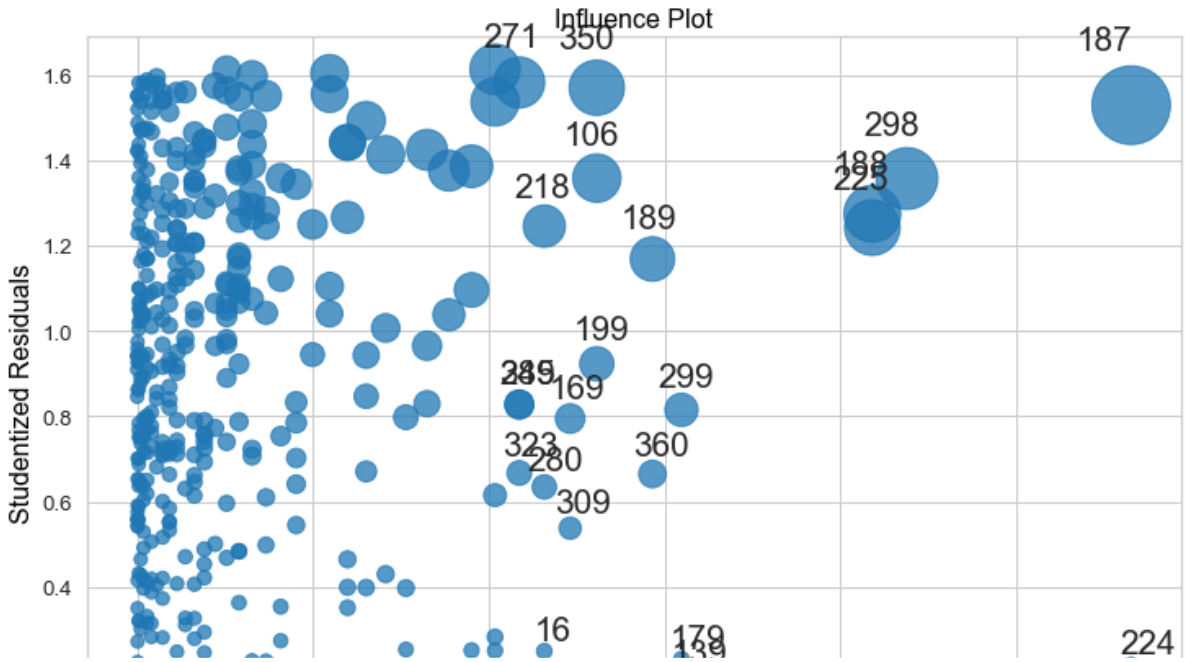
Out[3737]:

```
<AxesSubplot:>
```



In [3738]:

```
#Screening for Outliers
fig, ax = plt.subplots(figsize=(12,8))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



In [3739]:

```
# --> there are outliers!!
# exclude them! :
exclList=[370,212,241,2,76,358,101,191,227,363,301,137,177,345,270,374,272,337,2
exclListSorted = sorted(exclList)
print(exclListSorted)
print(len(exclListSorted))
```

```
[2, 15, 16, 27, 54, 55, 76, 77, 96, 101, 105, 106, 126, 137, 138, 139,
141, 142, 143, 150, 151, 152, 153, 160, 167, 168, 169, 170, 171, 177,
178, 179, 181, 182, 183, 184, 187, 188, 189, 190, 191, 192, 198, 199,
207, 208, 212, 213, 217, 218, 223, 224, 225, 226, 227, 234, 235, 236,
237, 238, 239, 241, 242, 246, 247, 254, 265, 270, 271, 272, 277, 279,
280, 281, 282, 283, 284, 285, 288, 289, 295, 296, 297, 298, 299, 300,
301, 303, 308, 309, 322, 323, 326, 327, 328, 329, 337, 344, 345, 350,
351, 358, 359, 360, 363, 370, 373, 374, 381]
109
```

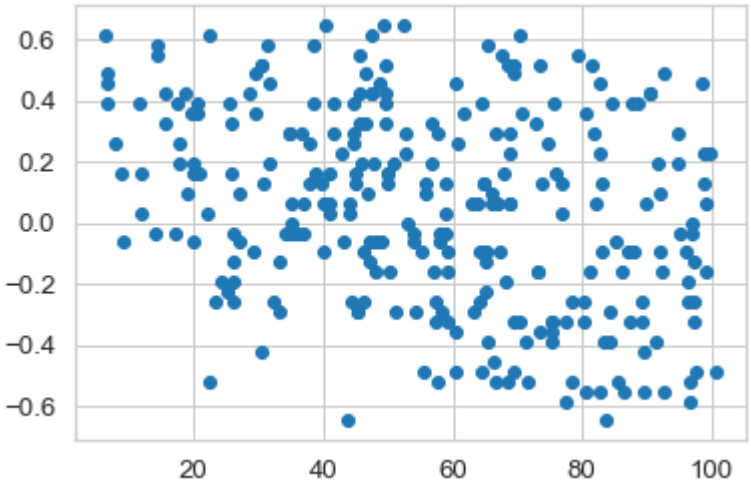
In [3740]:

```
dfNoD2XvsWPCleanedFromOutliers = dfNoD2XvsWP.drop(exclList)
dfNoD2XvsWPCleanedFromOutliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 278 entries, 0 to 386
Data columns (total 2 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   numberOfDaysToXmas  278 non-null    int64
 1   week_position       278 non-null    int64
dtypes: int64(2)
memory usage: 6.5 KB
```

In [3741]:

```
#re-create model:
#Testing for Homoscedasticity
x = dfNoD2XvsWPCleanedFromOutliers['numberOfDaysToXmas']
y = dfNoD2XvsWPCleanedFromOutliers['week_position']
model = sm.OLS(y,x).fit()
pred_val = model.fittedvalues.copy()
true_val = dfNoD2XvsWPCleanedFromOutliers['week_position'].values.copy()
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6, 4))
_ = ax.scatter(residual, pred_val)
```



In [3742]:

```
sms.diagnostic.het_breuschpagan(residual, dfNoD2XvsWPCleanedFromOutliers[['numbe
# --> p-Value = 0.07, still > 0.05, Homoscedasticity met
```

Out[3742]:

```
(1.3314109936150724, nan, 1.333005841233617, 0.2492662479941024)
```

In [3743]:

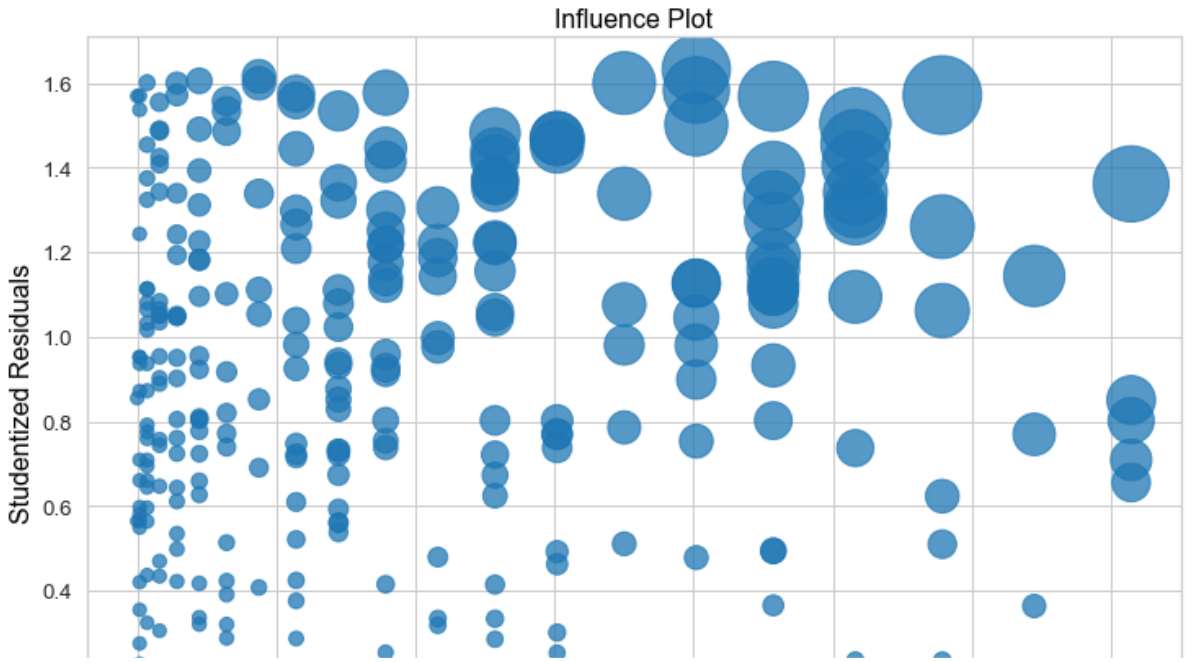
```
# Testing for Multicollinearity:
dfNoD2XvsWPCleanedFromOutliers.corr()
```

Out[3743]:

	numberOfDaysToXmas	week_position
numberOfDaysToXmas	1.000000	0.312153
week_position	0.312153	1.000000

In [3744]:

```
#Screening for Outliers
fig, ax = plt.subplots(figsize=(12,8))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



In [3745]:

```
infl = model.get_influence()
infl.summary_frame()[:5]
```

Out[3745]:

	dfb_numberOfDaysToXmas	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid	
0	-0.035683	0.001275	0.852473	0.001751	0.035701	0.852051	0
1	-0.065985	0.004360	0.786270	0.007003	0.066031	0.785727	0
3	0.150754	0.022607	1.564907	0.009147	0.150358	1.569031	0
4	-0.059990	0.003604	0.770224	0.006039	0.060034	0.769657	0
5	0.170132	0.028792	1.567881	0.011577	0.169682	1.572039	0

In [3746]:

```
infl.summary_frame()['dfb_numberOfDaysToXmas'].sort_values(ascending=False)[:5]
# --> no problem here

```

Out[3746]:

```
5      0.170132
140    0.163987
158    0.153720
3      0.150754
111    0.148698
Name: dfb_numberOfDaysToXmas, dtype: float64
```

In [3747]:

```
infl.summary_frame()['dffits'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3747]:

```
5      0.170132
140    0.163987
158    0.153720
3      0.150754
111    0.148698
Name: dffits, dtype: float64
```

In [3748]:

```
infl.summary_frame()['hat_diag'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3748]:

```
140    0.014292
197    0.014292
233    0.014292
352    0.014292
383    0.014292
Name: hat_diag, dtype: float64
```

In [3749]:

```
infl.summary_frame()['student_resid'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3749]:

```
290    1.633221
252    1.616370
200    1.605674
201    1.601364
122    1.600653
Name: student_resid, dtype: float64
```

In [3750]:

```
model.summary()
```

Out[3750]:

OLS Regression Results

Dep. Variable:	week_position	**R-squared (uncentered):**	0.000
Model:	OLS	**Adj. R-squared (uncentered):**	-0.004
Method:	Least Squares	**F-statistic:**	0.007640
Date:	Fri, 31 Dec 2021	**Prob (F-statistic):**	0.930
Time:	18:53:22	**Log-Likelihood:**	-1541.1
No. Observations:	278	**AIC:**	3084.
Df Residuals:	277	**BIC:**	3088.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
numberOfDaysToXmas	-0.0324	0.370	-0.087	0.930	-0.761	0.697

Omnibus:	37.637	**Durbin-Watson:**	0.162
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	10.448
Skew:	-0.086	**Prob(JB):**	0.00539
Kurtosis:	2.066	**Cond. No.**	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Interpretation: **Accept H0 that time-distance to Christmas has no significant influence on the week-position of the song!**

In [ ]:

```

```

In [ ]:

```

```

UNTERBRUCH 20211230

Influence of Decade on Week Position

Test if the week-position differs significantly during the decades.

Null Hypthesis H0 = ***The decades a song was on the chart has no significant influence on the quality of the week-position of a song.***

First a re-coding of week-position into four categories is planned. But this turned out to bee to filigrane - it lead to a violation of the prerequisite of at least 5 members per cell. So the recoding was more constrained into two categories as 'good' and 'bad' positions, being divided according to the song's position compared to the median.

Run an Independent Chi-Square Test

In [3751]:

```
dfXmasSongsRaw['decade'] = ((dfXmasSongsRaw['year'])/10 ).astype(int)*10
dfXmasSongsRaw['decade'][:5]

```

Out[3751]:

```
0    2000
1    2000
2    2000
3    2000
4    2010
Name: decade, dtype: int64
```

In [3752]:

```
dfXmasSongsRaw[['decade','date']][10:18]

```

Out[3752]:

	decade	date
10	2010	2015-01-10
11	1980	1984-12-22
12	1980	1984-12-29
13	1980	1985-01-05
14	1980	1985-01-12
15	1980	1985-01-19
16	1980	1985-01-26
17	1950	1958-12-20

In [3753]:

```
#reclassify the week position to a more categorical value based on simple statis
# 25%: 38:  0 < x <= 38 --> Class 1
# 50%: 58: 38 < x <= 58 --> Class 2
# 75%: 78: 58 < x <= 78 --> Class 3
#               x >  78 --> Class 4

dfXmasSongsRaw['weekPosCat'] = 4
dfXmasSongsRaw['weekPosCat'] = np.where(dfXmasSongsRaw['week_position']<=78,3,df
dfXmasSongsRaw['weekPosCat'] = np.where(dfXmasSongsRaw['week_position']<=58,2,df
dfXmasSongsRaw['weekPosCat'] = np.where(dfXmasSongsRaw['week_position']<=38,1,df
dfXmasSongsRaw['weekPosCat'].value_counts()
```

Out[3753]:

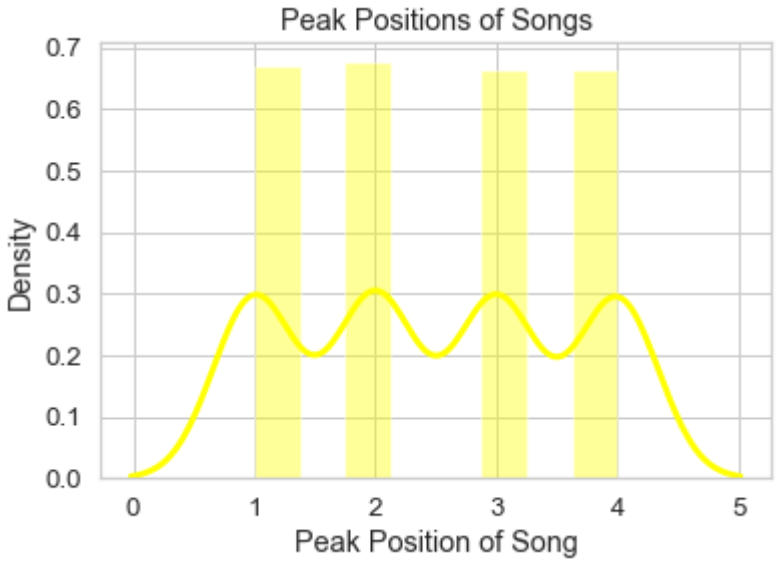
```
2    98
1    97
4    96
3    96
Name: weekPosCat, dtype: int64
```

In [3754]:

```
#Histogram of the peak position
ax = sns.distplot(dfXmasSongsRaw['weekPosCat'], color='yellow')
ax.set(xlabel='Peak Position of Song', ylabel='Density', title='Peak Positions o

```

Out[3754]:

```
[Text(0.5, 0, 'Peak Position of Song'),
 Text(0, 0.5, 'Density'),
 Text(0.5, 1.0, 'Peak Positions of Songs')]
```



Do a ***Chi square test of independence***. Assumption: both the DV and IV are categorical, because they are ordinal (decade and re-categorisation of rank).
Null Hypothesis H0: **there is no significant dependence between means of 'Decade' and 'Week Position'**.

Create a Contingency Table:

In [3755]:

```
xmasWPCByDecCrosstab = pd.crosstab(dfXmasSongsRaw['decade'] , dfXmasSongsRaw['we
xmasWPCByDecCrosstab
```

Out[3755]:

weekPosCat	1	2	3	4	All
decade					
1950	2	5	8	6	21
1960	31	52	37	24	144
1970	12	7	13	12	44
1980	16	1	7	11	35
1990	5	3	7	9	24
2000	2	11	18	19	50
2010	29	19	6	15	69
All	97	98	96	96	387

Problem: We have cells with fewer than 5 members!

Run the test anyway in a first approach...

In [3756]:

```
from scipy import stats
stats.chi2_contingency(xmasWPCByDecCrosstab)

```

Out[3756]:

```
(65.0065554156709,
 8.993000292227184e-05,
 28,
 array([[  5.26356589,   5.31782946,   5.20930233,   5.20930233,
          21.        ],
        [ 36.09302326,  36.46511628,  35.72093023,  35.72093023,
         144.        ],
        [ 11.02842377,  11.14211886,  10.91472868,  10.91472868,
          44.        ],
        [  8.77260982,   8.8630491 ,   8.68217054,   8.68217054,
          35.        ],
        [  6.01550388,   6.07751938,   5.95348837,   5.95348837,
          24.        ],
        [ 12.53229974,  12.66149871,  12.40310078,  12.40310078,
          50.        ],
        [ 17.29457364,  17.47286822,  17.11627907,  17.11627907,
          69.        ],
        [ 97.        ,  98.        ,  96.        ,  96.        ,
         387.        ]]))
```

--> **p-value = 8.993e-05** --> strong indication for significance! ***Reject H0!!***

Re-categorise / Recode the values with just two classes: 0 for rankings lower/better than 58 and 1 for rankings

above / worse than 58.

In [3757]:

```
#reclassify the week position to a more categorical value based on simple statis
# 50%: 58: x <= 58 --> Class 0
# 50%: 58: x > 58 --> Class 1

dfXmasSongsRaw['weekPosCat50'] = 1
dfXmasSongsRaw['weekPosCat50'] = np.where(dfXmasSongsRaw['week_position']<=58,0,
dfXmasSongsRaw['weekPosCat50'].value_counts()
```

Out[3757]:

```
0    195
1    192
Name: weekPosCat50, dtype: int64
```

In [3758]:

```
xmasWPCByDec50Crosstab = pd.crosstab(dfXmasSongsRaw['decade'] , dfXmasSongsRaw['
xmasWPCByDec50Crosstab
```

Out[3758]:

weekPosCat50	0	1	All
decade			
1950	7	14	21
1960	83	61	144
1970	19	25	44
1980	17	18	35
1990	8	16	24
2000	13	37	50
2010	48	21	69
All	195	192	387

Now all cells have at least 5 members! Condition fulfilled!

In [3759]:

```
from scipy import stats
stats.chi2_contingency(xmasWPCByDec50Crosstab)

```

Out[3759]:

```
(31.271705131485355,
 0.005077737416163921,
 14,
 array([[ 10.58139535,  10.41860465,  21.        ],
        [ 72.55813953,  71.44186047, 144.        ],
        [ 22.17054264,  21.82945736,  44.        ],
        [ 17.63565891,  17.36434109,  35.        ],
        [ 12.09302326,  11.90697674,  24.        ],
        [ 25.19379845,  24.80620155,  50.        ],
        [ 34.76744186,  34.23255814,  69.        ],
        [195.        , 192.        , 387.        ]]))
```

--> **p-value = 0.00507** --> There is indication of significance! ***Reject H0!!***

A second test on the median week position of a song per decade

Null Hypthesis H0 = ***The decade a song was on the chart has no significant influence on the median week-position of a song.***

Run an Independent Chi-Square Test

Prepare the data for analysis on decades vs week-position

In [3760]:

```
dfXmasDecadeWeekPosDec = dfXmasSongsRaw.groupby(dfXmasSongsRaw['decade'])[['deca
dfXmasDecadeWeekPosDec['rownr'] = np.arange(dfXmasDecadeWeekPosDec.shape[0])
dfXmasDecadeWeekPosDec

```

Out[3760]:

	decade	rownr
decade		
1950	1950.0	0
1960	1960.0	1
1970	1970.0	2
1980	1980.0	3
1990	1990.0	4
2000	2000.0	5
2010	2010.0	6

In [3761]:

```
#create a pandas' series
serDec = dfXmasDecadeWeekPosDec.iloc[:,0]
serDec

```

Out[3761]:

```
decade
1950    1950.0
1960    1960.0
1970    1970.0
1980    1980.0
1990    1990.0
2000    2000.0
2010    2010.0
Name: decade, dtype: float64
```

In [3762]:

```
dfXmasDecadeWeekPosAvg = dfXmasSongsRaw.groupby(dfXmasSongsRaw['decade'])[['week
dfXmasDecadeWeekPosAvg['rownr'] = np.arange(dfXmasDecadeWeekPosAvg.shape[0])
dfXmasDecadeWeekPosAvg.rename(columns={'week_position': 'WeekPositionAvg'},inpla
dfXmasDecadeWeekPosAvg
```

Out[3762]:

decade	WeekPositionAvg	rownr
1950	65.904762	0
1960	54.736111	1
1970	59.522727	2
1980	51.028571	3
1990	63.250000	4
2000	69.940000	5
2010	50.028986	6

In [3763]:

```
#create a pandas' series
serAvgWP = dfXmasDecadeWeekPosAvg.iloc[:,0]
serAvgWP

```

Out[3763]:

```
decade
1950    65.904762
1960    54.736111
1970    59.522727
1980    51.028571
1990    63.250000
2000    69.940000
2010    50.028986
Name: WeekPositionAvg, dtype: float64
```

In [3764]:

```
dfXmasDecadeWeekPosMedian = dfXmasSongsRaw.groupby(dfXmasSongsRaw['decade'])[['w
dfXmasDecadeWeekPosMedian['rownr'] = np.arange(dfXmasDecadeWeekPosMedian.shape[0
dfXmasDecadeWeekPosMedian.rename(columns={'week_position': 'WeekPositionMedian'}
dfXmasDecadeWeekPosMedian
```

Out[3764]:

decade	WeekPositionMedian	rownr
1950	69.0	0
1960	54.0	1
1970	64.5	2
1980	59.0	3
1990	66.5	4
2000	70.0	5
2010	44.0	6

In [3765]:

```
#create a pandas' series
serMedianWP = dfXmasDecadeWeekPosMedian.iloc[:,0]
serMedianWP

```

Out[3765]:

```
decade
1950    69.0
1960    54.0
1970    64.5
1980    59.0
1990    66.5
2000    70.0
2010    44.0
Name: WeekPositionMedian, dtype: float64
```

In [3766]:

```
#merge the series
dfAvgMedWPByDecade = pd.concat([serAvgWP,serMedianWP,serDec],axis=1)
dfAvgMedWPByDecade

```

Out[3766]:

decade	WeekPositionAvg	WeekPositionMedian	decade
1950	65.904762	69.0	1950.0
1960	54.736111	54.0	1960.0
1970	59.522727	64.5	1970.0
1980	51.028571	59.0	1980.0
1990	63.250000	66.5	1990.0
2000	69.940000	70.0	2000.0
2010	50.028986	44.0	2010.0

Do a ***Chi square test of independence***.

Assumption: both the DV and IV are categorical, because they are ordinal (decade and rank).

Create a Contingency Table:

In [3767]:

```
xmasMedByDecCrosstab = pd.crosstab(dfAvgMedWPByDecade.decade, dfAvgMedWPByDecade
xmasMedByDecCrosstab
```

Out[3767]:

WeekPositionMedian	44.0	54.0	59.0	64.5	66.5	69.0	70.0	All
decade								
1950.0	0	0	0	0	0	1	0	1
1960.0	0	1	0	0	0	0	0	1
1970.0	0	0	0	1	0	0	0	1
1980.0	0	0	1	0	0	0	0	1
1990.0	0	0	0	0	1	0	0	1
2000.0	0	0	0	0	0	0	1	1
2010.0	1	0	0	0	0	0	0	1
All	1	1	1	1	1	1	1	7

In [3768]:

```
from scipy import stats
stats.chi2_contingency(xmasMedByDecCrosstab)

```

Out[3768]:

```
(42.000000000000014,
 0.7503696058068774,
 49,
 array([[0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714,
         0.14285714, 0.14285714, 1.        ],
        [1.        , 1.        , 1.        , 1.        , 1.        ,
         1.        , 1.        , 7.        ]]))
```

p-value = 0.75 --> no significant differences of rankings between the decades.
BUT... problem again: we do not have at least 5 cases per cell...

In [3769]:

```
#Try correlation between Median week position and decade:
dfAvgMedWPByDecade['WeekPositionMedian'].corr(dfAvgMedWPByDecade['decade'])
```

Out[3769]:

```
-0.33672228418685873
```

Interpretation: there is a weak to medium negative correlation between median week position and decade.
In other words: in last century the median week positions were in tendence better than in the recent years.

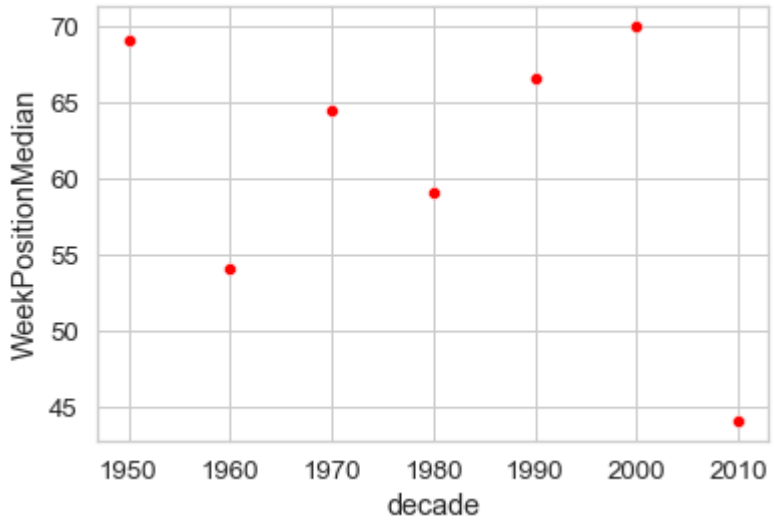
Do a more visual analysis with a scatter plot

In [3770]:

```
#Scatterplot of date vs weeks on chart
sns.scatterplot(data=dfAvgMedWPByDecade, x='decade', y='WeekPositionMedian',colo
```

Out[3770]:

```
<AxesSubplot:xlabel='decade', ylabel='WeekPositionMedian'>
```

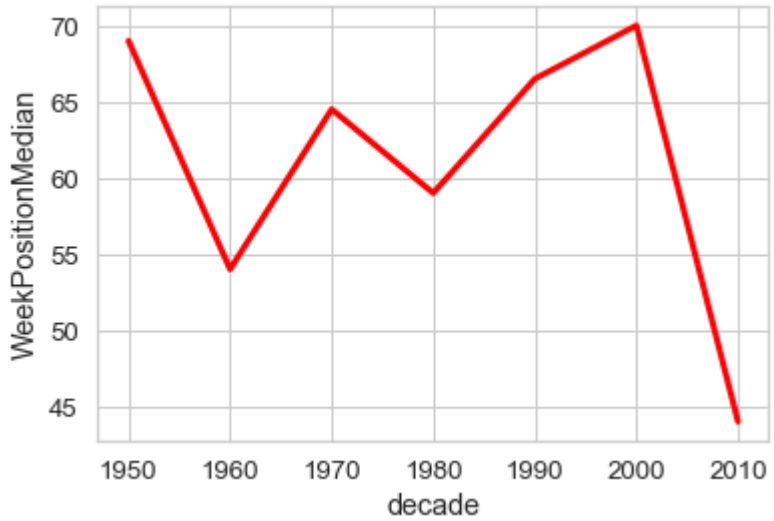


In [3771]:

```
1 sns.lineplot(data=dfAvgMedWPByDecade, x='decade', y='WeekPositionMedian',color='
```

Out[3771]:

```
<AxesSubplot:xlabel='decade', ylabel='WeekPositionMedian'>
```

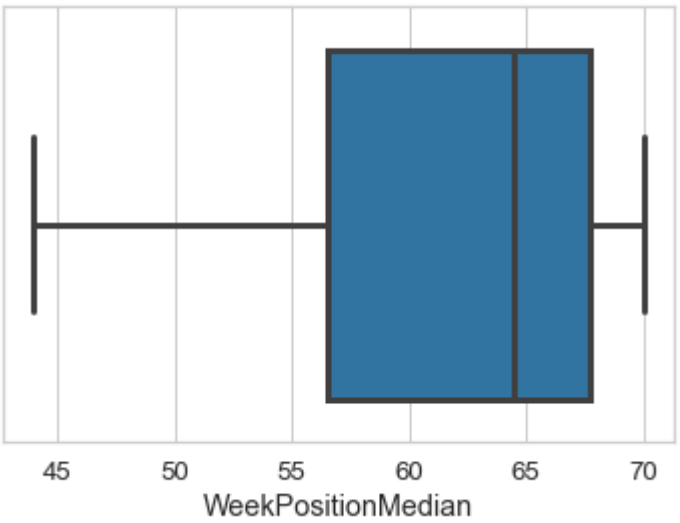


In [3772]:

```
1 sns.boxplot(data=dfAvgMedWPByDecade, x='WeekPositionMedian')
```

Out[3772]:

```
<AxesSubplot:xlabel='WeekPositionMedian'>
```



Test if the month (Nov, Dec or Jan) is significantly influencing the week-position

Null Hypthesis H0 = ***The month of a song's position has no significant influence on the week-position of a song.***

IV: month as categorical variable (with 3 levels, one for each month) DV: week-position as continuous variable

Run ANOVA test

In [3773]:

```
import scipy
from scipy import stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from statsmodels.stats.multicomp import MultiComparison
```

In [3774]:

```
#Only Keep the 2 Varaibles that are necessary
xmasSongsMonthVsWP = dfXmasSongsRaw[['week_position', 'month']]
print(xmasSongsMonthVsWP['week_position'].value_counts())
print(xmasSongsMonthVsWP['month'].value_counts())
print(xmasSongsMonthVsWP.info())
```

```
45    10
26    10
59     9
97     9
65     9
      ..
15     2
42     2
79     2
36     1
24     1
Name: week_position, Length: 89, dtype: int64
12    224
1     148
11     15
Name: month, dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 387 entries, 0 to 386
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   week_position  387 non-null    int64
 1   month          387 non-null    int64
dtypes: int64(2)
memory usage: 6.2 KB
None
```

In [3775]:

```
#Recode month information
def recode (mth):
    if mth == 11:
        return "Nov"
    if mth == 12:
        return "Dec"
    if mth == 1:
        return "Jan"

xmasSongsMonthVsWP['monthR'] = xmasSongsMonthVsWP['month'].apply(recode)
print(xmasSongsMonthVsWP.info())
print(xmasSongsMonthVsWP['monthR'].value_counts())
```

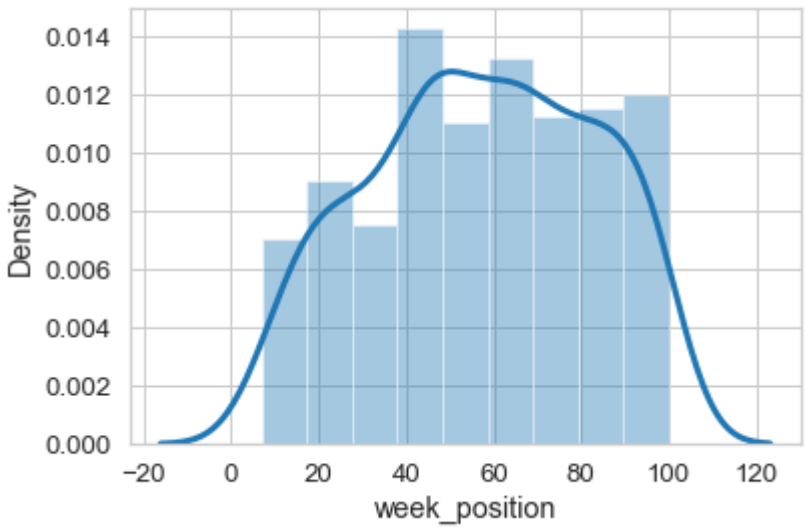
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 387 entries, 0 to 386
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   week_position  387 non-null    int64 
 1   month          387 non-null    int64 
 2   monthR         387 non-null    object
dtypes: int64(2), object(1)
memory usage: 9.2+ KB
None
Dec    224
Jan    148
Nov     15
Name: monthR, dtype: int64
```

In [3776]:

```
#Test for normality
sns.distplot(xmasSongsMonthVsWP['week_position'])
```

Out[3776]:

```
<AxesSubplot:xlabel='week_position', ylabel='Density'>
```

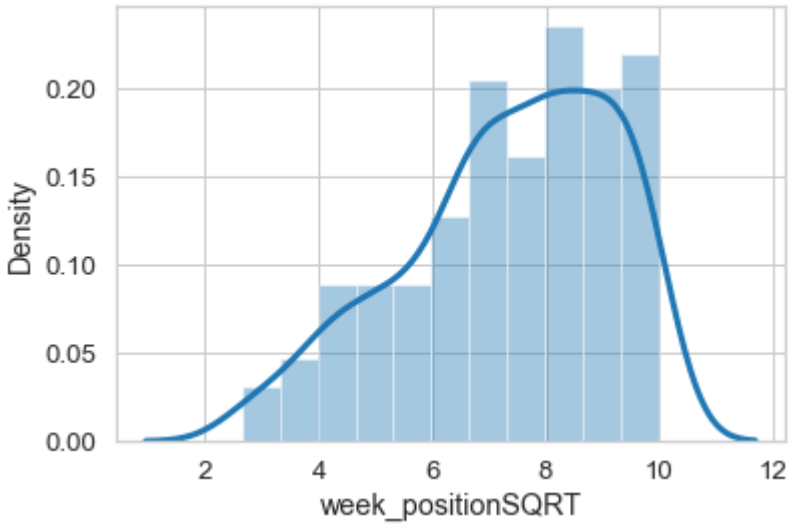


In [3777]:

```
xmasSongsMonthVsWP['week_positionSQRT'] = np.sqrt(xmasSongsMonthVsWP['week_posit
sns.distplot(xmasSongsMonthVsWP['week_positionSQRT'])
```

Out[3777]:

```
<AxesSubplot:xlabel='week_positionSQRT', ylabel='Density'>
```

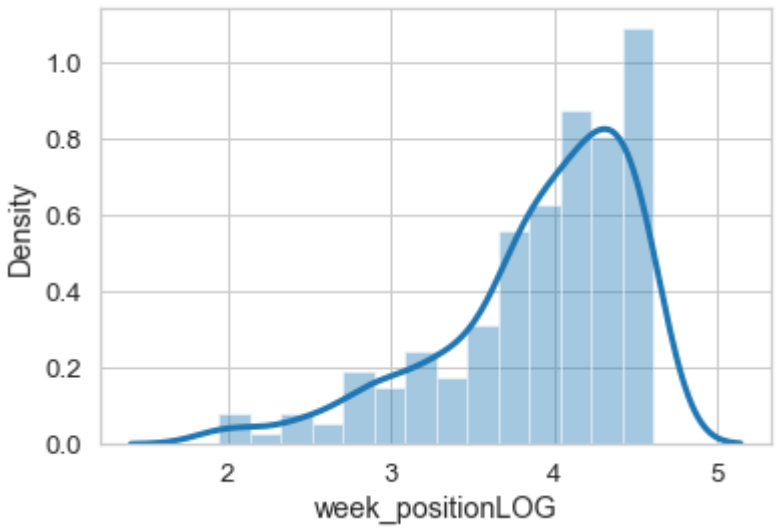


In [3778]:

```
xmasSongsMonthVsWP['week_positionLOG'] = np.log(xmasSongsMonthVsWP['week_positio
sns.distplot(xmasSongsMonthVsWP['week_positionLOG'])
```

Out[3778]:

```
<AxesSubplot:xlabel='week_positionLOG', ylabel='Density'>
```



In [3779]:

```
xmasSongsMonthVsWP.drop(['week_positionSQRT', 'week_positionLOG'], axis=1,inplac
```

--> the original data looks best - no log or sqrt performed!

In [3780]:

```
#Test for Homogeneity of Variance
scipy.stats.bartlett(xmasSongsMonthVsWP['week_position'], xmasSongsMonthVsWP['mo
```

Out[3780]:

```
BartlettResult(statistic=703.1903972002012, pvalue=6.0524826721612594e
-155)
```

Unfortunately does not meet the assumption of homogeneity of variance: p-value = 6.052e-155 Variance is unequal, violation of assumption of homogeneity!
Keep going anyway... run the analysis:

In [3781]:

```
stats.f_oneway(xmasSongsMonthVsWP['week_position'][xmasSongsMonthVsWP['monthR']=
                    xmasSongsMonthVsWP['week_position'][xmasSongsMonthVsWP['monthR']=
                xmasSongsMonthVsWP['week_position'][xmasSongsMonthVsWP['monthR']=
```

Out[3781]:

```
F_onewayResult(statistic=14.180633815759432, pvalue=1.1435227378678987
e-06)
```

p-value is < 0.05 *--> reject HO*. In other words: **The months significantly influences the week position of a song!**

In [3782]:

```
#PostHoc
postHoc = MultiComparison(xmasSongsMonthVsWP['week_position'], xmasSongsMonthVsW
postHocResults = postHoc.tukeyhsd()
print(postHocResults)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower    upper  reject
----------------------------------------------------
     1     11  23.1045 0.0017   7.4373 38.7717   True
     1     12  12.3468  0.001    6.222 18.4715   True
    11     12 -10.7577 0.2298 -26.1784  4.6629  False
----------------------------------------------------
```

In [3783]:

```
#Examine means first
xmasSongsMonthVsWP.groupby('monthR').mean()
```

Out[3783]:

monthR	week_position	month
Dec	61.508929	12.0
Jan	49.162162	1.0
Nov	72.266667	11.0

Looks like in *November* the week-positions are best, followed by *December* and finally *January*.

In [3784]:

```
# Because of violation of the assumption of homegeineity run Welch's ANOVA
# Source: https://www.statology.org/welchs-anova-in-python/
import pingouin as pg
pg.welch_anova(dv='week_position', between='monthR', data=xmasSongsMonthVsWP)

```

Out[3784]:

	Source	ddof1	ddof2	F	p-unc	np2
0	monthR	2	38.344556	13.612837	0.000034	0.068778

The overall p-value (0.000034) from the ANOVA table is less than α = .05, which means we can reject the null hypothesis that the week positions are equal between the three months.

In [3785]:

```
#Perform a Games-Howell post-hoc test to determine exactly which group means are
pg.pairwise_gameshowell(dv='week_position', between='monthR', data=xmasSongsMont

```

Out[3785]:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	he
0	Dec	Jan	61.508929	49.162162	12.346766	2.627381	4.699267	304.571722	0.001000	0.49
1	Dec	Nov	61.508929	72.266667	-10.757738	6.572620	-1.636750	15.842499	0.259827	-0.43
2	Jan	Nov	49.162162	72.266667	-23.104505	6.701693	-3.447562	17.110140	0.008113	-0.92

The p-values in the table above show:

- **the mean difference between December and January are significantly different**
- **the mean difference between November and January are significantly different**

Finally some general information on week's position of songs:

Finally some general information on week's position of songs:

In [3786]:

```
avg(week_position) as MinWeekPosition,max(week_position)-min(week_position) as weekP
```

Out[3786]:

	MinWeekPosition	MaxWeekPosition	MinWeekPosition	weekPosRange	songid
0	7	96	31.818182	89	AmenThe Impressions
1	11	96	50.900000	85	MistletoeJustin Bieber
2	7	89	44.800000	82	Auld Lang SyneKenny G
3	21	97	49.625000	76	Jingle Bell RockBobby Rydell/Chubby Checker
4	7	82	26.250000	75	This One's For The ChildrenNew Kids On The Block
...	...	...	...	...	...
73	92	92	92.000000	0	Do They Know It's Christmas?Glee Cast
74	95	95	95.000000	0	Child Of GodBobby Darin
75	97	97	97.000000	0	Blue ChristmasThe Browns Featuring Jim Edward ...
76	99	99	99.000000	0	All I Want For Christmas Is YouMichael Buble
77	46	46	46.000000	0	A Holly Jolly ChristmasBurl Ives

78 rows × 5 columns

Part 2: Investigation Weeks on Chart

Work on the following aspects regarding Weeks on Chart (WoC):

- basic statistics on WoC
- influence of performer type
- influence of decade
- influence of peak-position
- influence of (month of) first appearance
- influence of time-to-christmas of first appearance
- influence of "Christmas" in Title
- influence of instance
- run a ML model to predict weeks on chart and get the influence factors

There is a problem that some data is only available on song-ID level. For instance, the peak-position is the same for one song no matter whether it was more than once or just once on the charts (instance =1 or instance > 1). The same is true of the number of weeks on the charts: it is an overall sum not the number of weeks on chart for one season. This makes it difficult to compare the data is some ways.

This artefact needs to be kept in mind for the following investigations and is illustrated in the following query:

In [3787]:

```
n = sqldf("""select count(*) as weeksOnChartPerSeason,season, weeks_on_chart ,instan
          from dfXmasSongsRaw
          group by songid,season order by songid asc,1 desc;""")
rSeason.info())
n[51]

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106 entries, 0 to 105
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   weeksOnChartPerSeason  106 non-null    int64 
 1   season                 106 non-null    int64 
 2   weeks_on_chart         106 non-null    int64 
 3   instance               106 non-null    int64 
 4   songid                 106 non-null    object
 5   peak_position          106 non-null    int64 
 6   date                   106 non-null    object
 7   year                   106 non-null    int64 
dtypes: int64(6), object(2)
memory usage: 6.8+ KB
None
```

Out[3787]:

In [3788]:

```
#do some data wrangling for future analysis
dfXmasSongsRaw['match'] = dfXmasSongsRaw['songid'] + dfXmasSongsRaw['season'].as

sqlWoCPerSeason['matchMe'] = sqlWoCPerSeason['songid'] + sqlWoCPerSeason['seasor

sqlWoCPerSeasonSub = sqlWoCPerSeason[['matchMe','weeksOnChartPerSeason']]
dfXmasSongsRaw = dfXmasSongsRaw.merge(sqlWoCPerSeasonSub,left_on='match',right_c
#dfXmasSongsRaw.drop(columns=['match','matchMe'],inplace=True)
dfXmasSongsRaw[:3]

```

Out[3788]:

	url	weekid	week_position	song	performer	performerGroup
0	http://www.billboard.com/charts/hot-100/2000-0...	2000-01-01 00:00:00	53	THIS GIFT	98 Degrees	1
1	http://www.billboard.com/charts/hot-100/2000-0...	2000-08-01 00:00:00	49	THIS GIFT	98 Degrees	1
2	http://www.billboard.com/charts/hot-100/2000-0...	1/15/2000	79	THIS GIFT	98 Degrees	1

3 rows × 28 columns

Basic statistics on Weeks on Chart (WoC)

In [3789]:

```
dfXmasSongsRaw['weeks_on_chart'].describe()
```

Out[3789]:

```
count    387.000000
mean       9.645995
std        6.142627
min        1.000000
25%        5.000000
50%        8.000000
75%       15.000000
max       20.000000
Name: weeks_on_chart, dtype: float64
```

In [3790]:

```
dfXmasSongsRaw['weeks_on_chart'].hist(bins=7,color="g")
plt.title('Histogram of Weeks of Presence')
plt.xlabel('Weeks')
plt.ylabel('Count')
```

Out[3790]:

```
<AxesSubplot:>
```

Out[3790]:

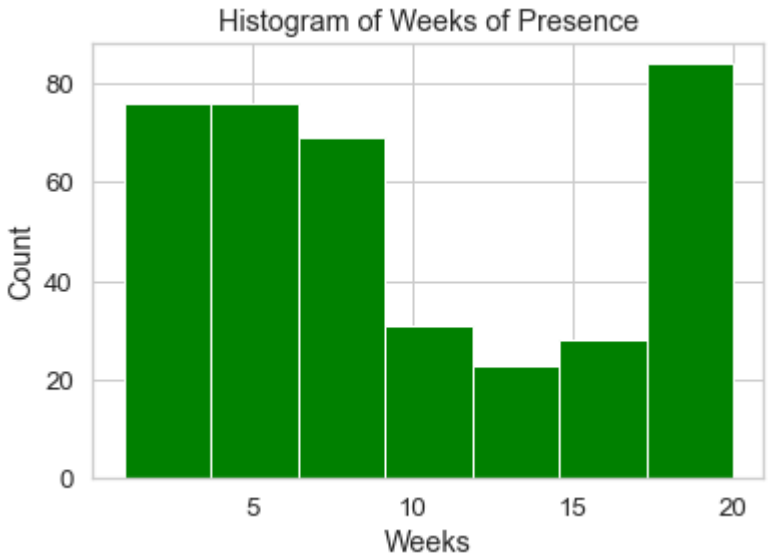
```
Text(0.5, 1.0, 'Histogram of Weeks of Presence')
```

Out[3790]:

```
Text(0.5, 0, 'Weeks')
```

Out[3790]:

```
Text(0, 0.5, 'Count')
```



In [3791]:

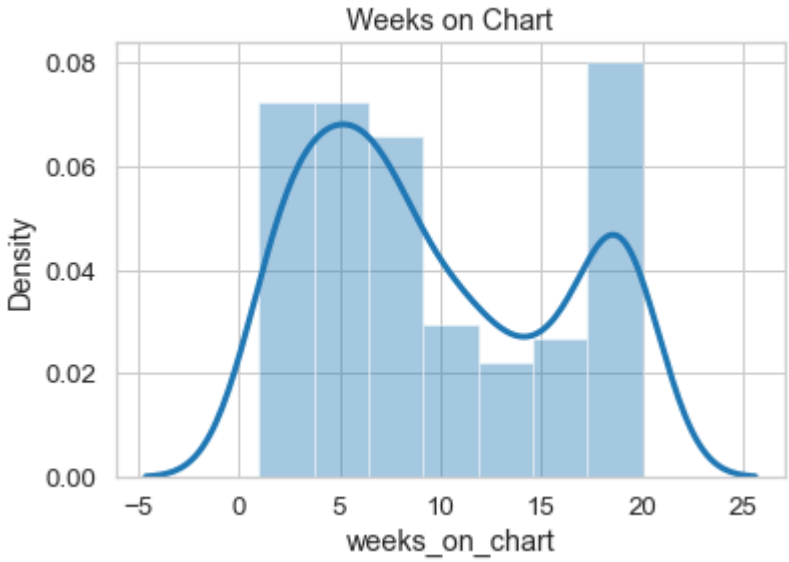
```
sns.set_context("notebook", font_scale=1.2, rc={"lines.linewidth": 3.0})
```

In [3792]:

```
1 sns.distplot(dfXmasSongsRaw['weeks_on_chart']).set_title("Weeks on Chart")
```

Out[3792]:

```
Text(0.5, 1.0, 'Weeks on Chart')
```

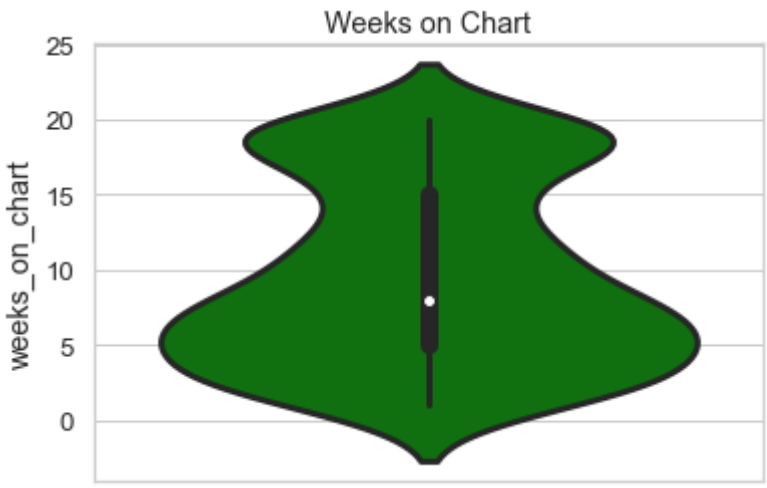


In [3793]:

```
1 sns.violinplot(y=dfXmasSongsRaw['weeks_on_chart'], color='g').set_title("Weeks c
2
```

Out[3793]:

```
Text(0.5, 1.0, 'Weeks on Chart')
```



Do the same basic analysis on the number of weeks on the chart by season

In [3794]:

```
1 sqlWoCPerSeason['weeksOnChartPerSeason'].describe()
```

Out[3794]:

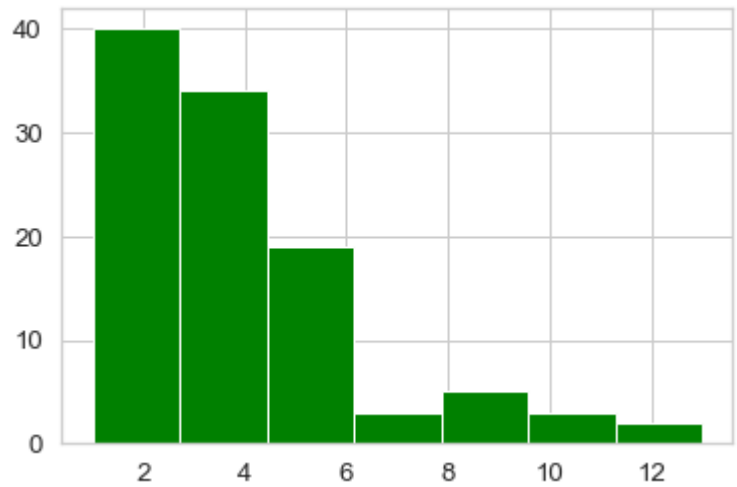
```
count    106.000000
mean       3.650943
std        2.665628
min        1.000000
25%        1.250000
50%        3.000000
75%        5.000000
max       13.000000
Name: weeksOnChartPerSeason, dtype: float64
```

In [3795]:

```
1 sqlWoCPerSeason['weeksOnChartPerSeason'].hist(bins=7,color="g")
```

Out[3795]:

```
<AxesSubplot:>
```

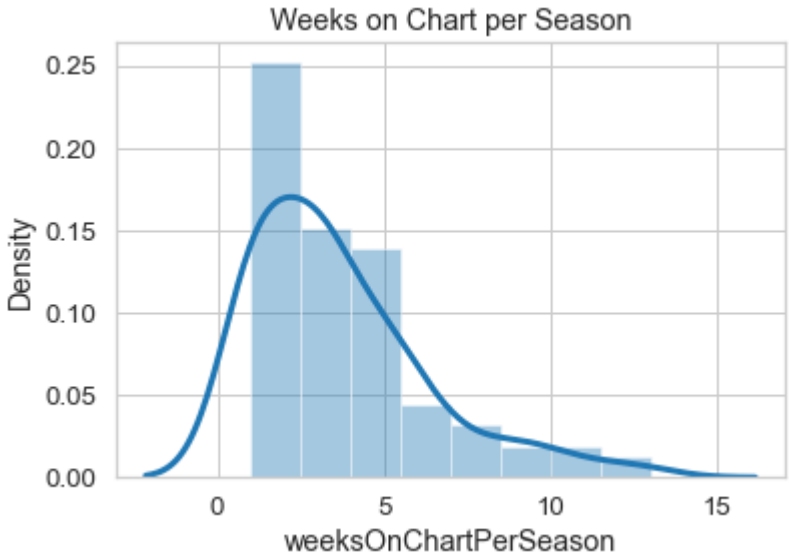


In [3796]:

```
1 sns.distplot(sqlWoCPerSeason['weeksOnChartPerSeason']).set_title("Weeks on Chart
```

Out[3796]:

```
Text(0.5, 1.0, 'Weeks on Chart per Season')
```

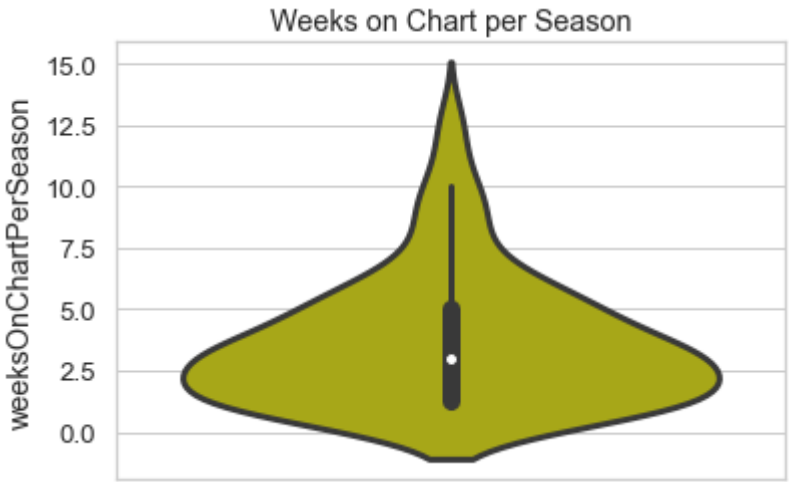


In [3797]:

```
1 sns.violinplot(y=sqlWoCPerSeason['weeksOnChartPerSeason'], color='y').set_title(
2
```

Out[3797]:

```
Text(0.5, 1.0, 'Weeks on Chart per Season')
```

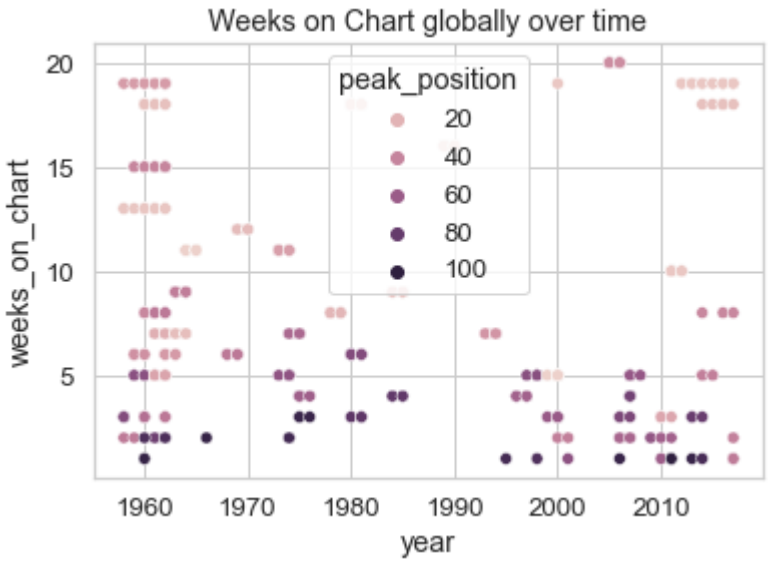


In [3798]:

```
#Scatterplot of date vs weeks on chart
sns.scatterplot(data=dfXmasSongsRaw, x='year', y='weeks_on_chart',hue='peak_positic
```

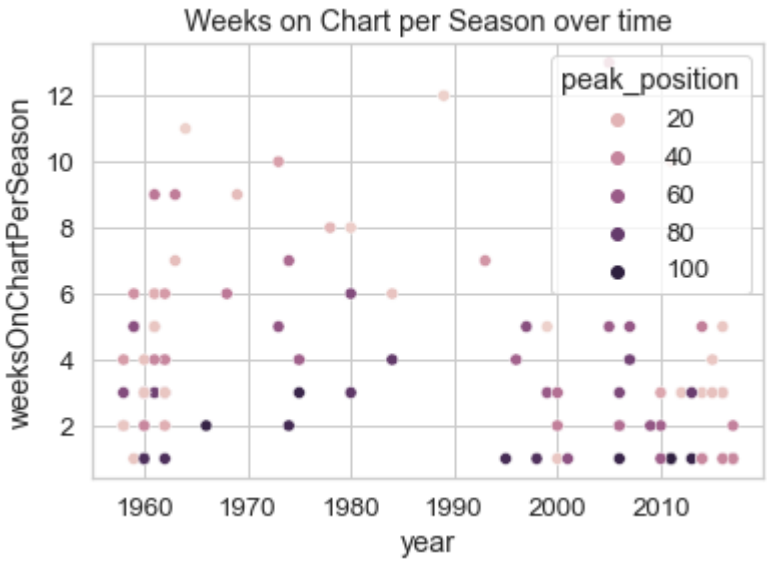
Out[3798]:

```
Text(0.5, 1.0, 'Weeks on Chart globally over time')
```



In [3799]:

```
#Scatterplot of date vs weeks on chart
plot = sns.scatterplot(data=sqlWoCPerSeason, x='year', y='weeksOnChartPerSeason'
```



In [3800]:

```
#Scatterplot of date vs weeks on chart
sns.scatterplot(data=sqlWoCPerSeason, x='year', y='weeks_on_chart',size='instanc
plt.xlabel('Year')
plt.ylabel('Count')
```

Out[3800]:

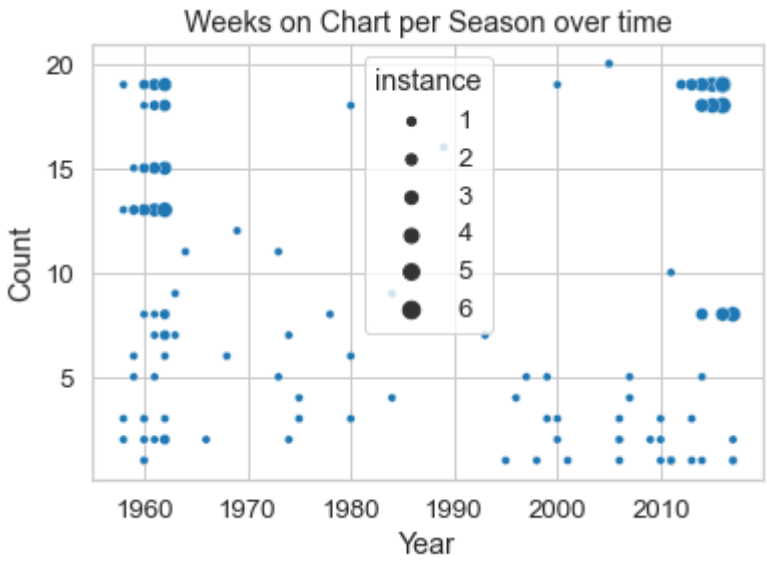
```
Text(0.5, 1.0, 'Weeks on Chart per Season over time')
```

Out[3800]:

```
Text(0.5, 0, 'Year')
```

Out[3800]:

```
Text(0, 0.5, 'Count')
```

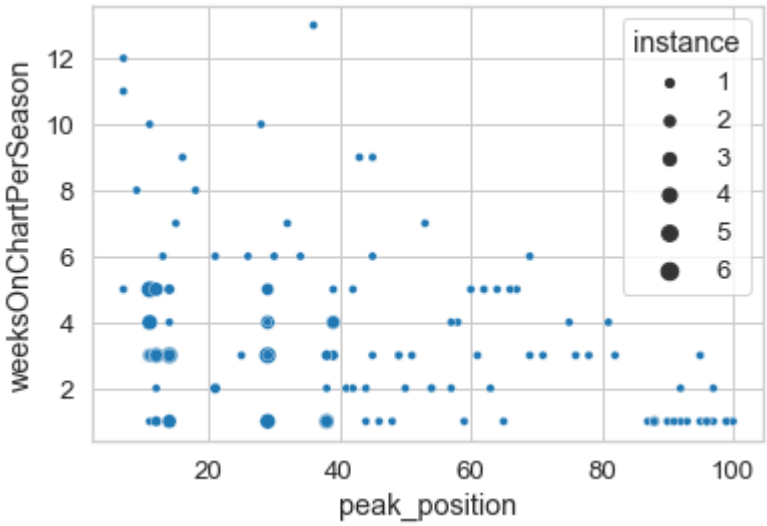


In [3801]:

```
#Scatterplot of date vs weeks on chart
sns.scatterplot(data=sqlWoCPerSeason, x='peak_position', y='weeksOnChartPerSeasc
```

Out[3801]:

```
<AxesSubplot:xlabel='peak_position', ylabel='weeksOnChartPerSeason'>
```



In [3802]:

```
sns.regplot(x="weeks_on_chart",y="peak_position", data=dfXmasSongsRaw);
plt.ylabel('Peak Position')
plt.xlabel('Weeks on Chart')
```

Out[3802]:

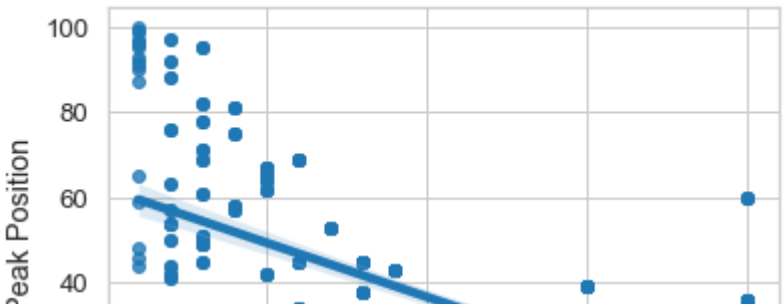
```
<AxesSubplot:xlabel='weeks_on_chart', ylabel='peak_position'>
```

Out[3802]:

```
Text(0, 0.5, 'Peak Position')
```

Out[3802]:

```
Text(0.5, 0, 'Weeks on Chart')
```



In [3803]:

```
sns.regplot(x="peak_position",y="weeksOnChartPerSeason", data=sqlWoCPerSeason);
plt.ylabel('Peak Position (globally)')
plt.xlabel('Weeks on Chart per Season')
```

Out[3803]:

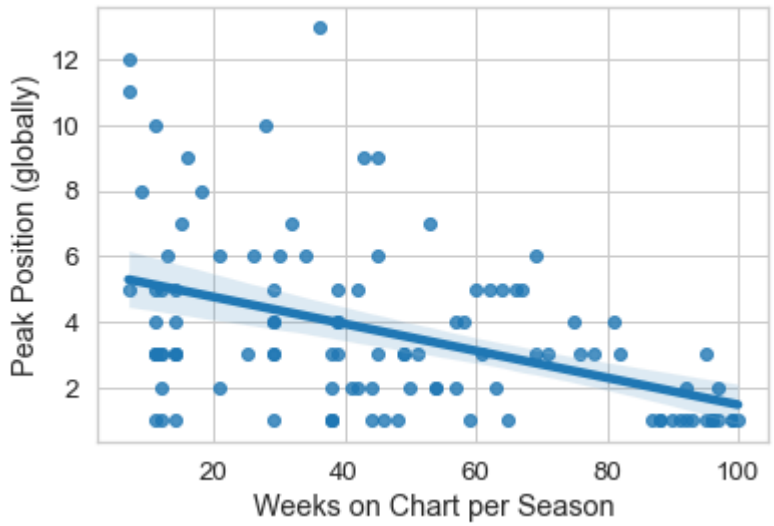
```
<AxesSubplot:xlabel='peak_position', ylabel='weeksOnChartPerSeason'>
```

Out[3803]:

```
Text(0, 0.5, 'Peak Position (globally)')
```

Out[3803]:

```
Text(0.5, 0, 'Weeks on Chart per Season')
```



Influence of performer type on WoC:

Null Hypothesis H0: **There is no signifcant difference in the means of the songs performed by a group or an individual.**

Run independent t-test.

In [3804]:

```
#split dataframe into 2 parts: one that contains songs performed by a group,
#the other part performed by individuals
groupPerformer = dfXmasSongsRaw[dfXmasSongsRaw['performerGroup']==1]
individualPerformer = dfXmasSongsRaw[dfXmasSongsRaw['performerGroup']==0]

# t-test for independent samples
# calculate the t test
data1 = groupPerformer['weeks_on_chart']
data2 = individualPerformer['weeks_on_chart']
alpha = 0.05
t_stat, df, cv, p = independent_ttest(data1, data2, alpha)
print('t=%.3f, df=%d, cv=%.3f, p=%.3f' % (t_stat, df, cv, p))
# interpret via critical value
if abs(t_stat) <= cv:
    print('Accept null hypothesis H0 that the means are equal.')
else:
    print('Reject the null hypothesis H0 that the means are equal.')
# interpret via p-value
if p > alpha:
    print('Accept null hypothesis H0 that the means are equal.')
else:
    print('Reject the null hypothesis H0 that the means are equal.')
```

```
t=-1.298, df=385, cv=1.649, p=0.195
Accept null hypothesis H0 that the means are equal.
Accept null hypothesis H0 that the means are equal.
```

With p=0.195 H0 is accepted!

Influence of decade on WoC.

Null Hypothesis H0: **There is no siginifcant influence of decade of the Song position on the weeks of presence in the Chart.**

Run ANOVA t-test with IV: decade as categorical variable (ordinal) with 7 levels DV: weeks of presence as continuous variable

In [3805]:

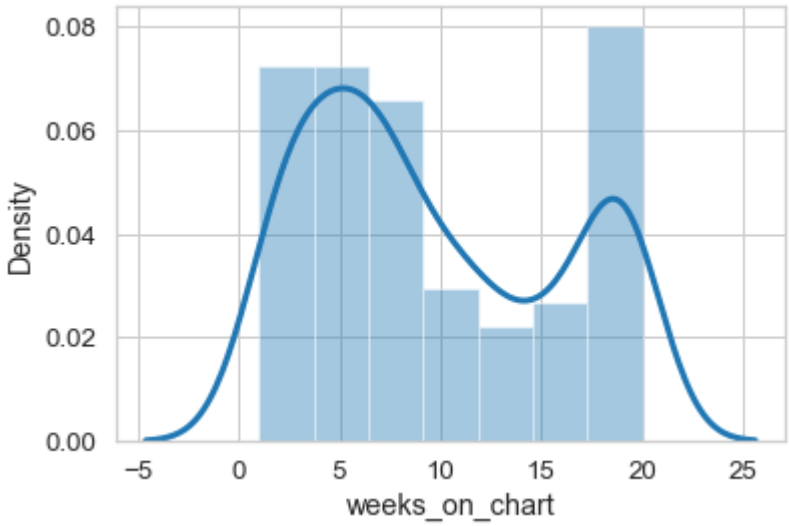
```
#Only Keep the 2 Varaibles that are necessary
xmasSongsDecadeVsWoC = dfXmasSongsRaw[['decade', 'weeks_on_chart']]

```

In [3806]:

```
#Test for normality
sns.distplot(xmasSongsDecadeVsWoC['weeks_on_chart'])
```

Out[3806]:

```
<AxesSubplot:xlabel='weeks_on_chart', ylabel='Density'>
```



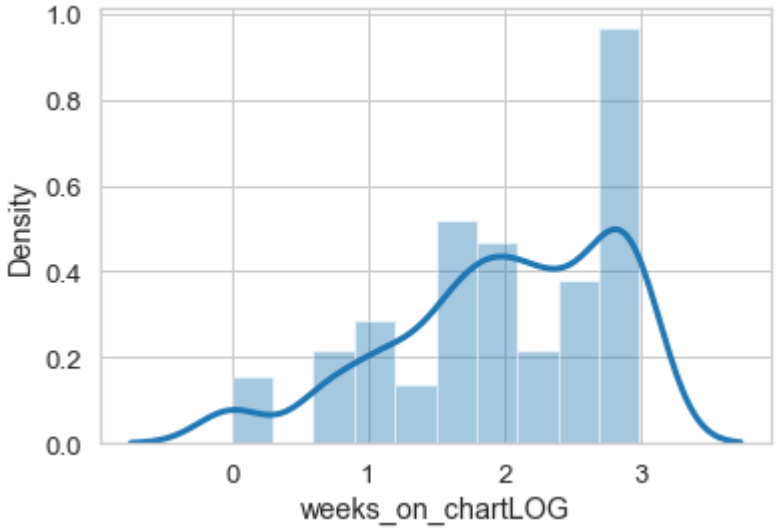
Not really normally distributed...

In [3807]:

```
xmasSongsDecadeVsWoC['weeks_on_chartLOG'] = np.log(xmasSongsDecadeVsWoC['weeks_c
sns.distplot(xmasSongsDecadeVsWoC['weeks_on_chartLOG'])
```

Out[3807]:

```
<AxesSubplot:xlabel='weeks_on_chartLOG', ylabel='Density'>
```

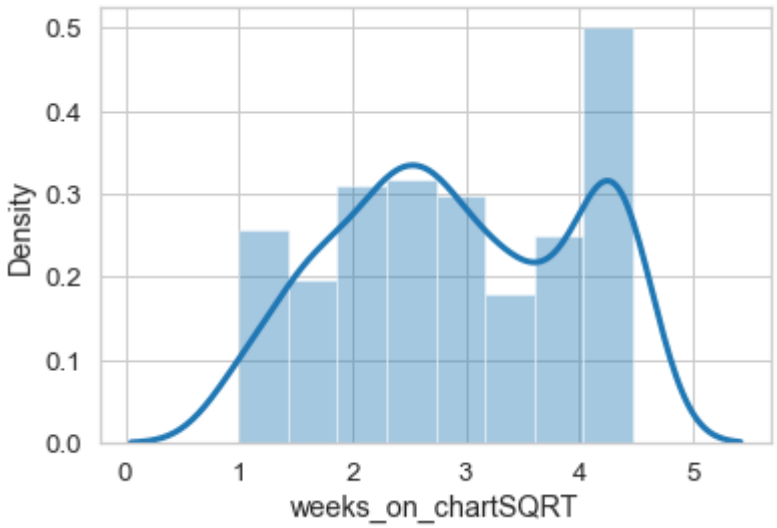


In [3808]:

```
xmasSongsDecadeVsWoC['weeks_on_chartSQRT'] = np.sqrt(xmasSongsDecadeVsWoC['weeks
sns.distplot(xmasSongsDecadeVsWoC['weeks_on_chartSQRT'])
```

Out[3808]:

```
<AxesSubplot:xlabel='weeks_on_chartSQRT', ylabel='Density'>
```

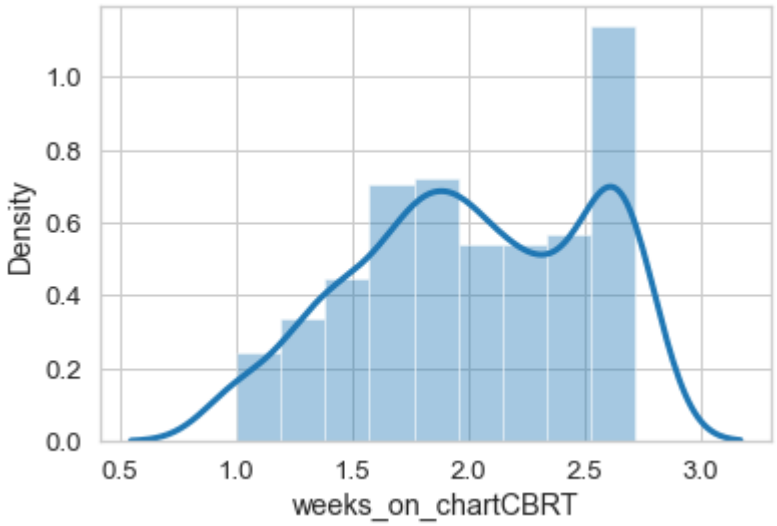


In [3809]:

```
xmasSongsDecadeVsWoC['weeks_on_chartCBRT'] = np.cbrt(xmasSongsDecadeVsWoC['weeks
sns.distplot(xmasSongsDecadeVsWoC['weeks_on_chartCBRT'])

```

Out[3809]:

```
<AxesSubplot:xlabel='weeks_on_chartCBRT', ylabel='Density'>
```



Neither datawrangling really leads to a normally-like distribution - but the cube-root does so the most. Go further with this.

In [3810]:

```
xmasSongsDecadeVsWoC.drop(['weeks_on_chartSQRT', 'weeks_on_chartLOG'], axis=1,ir
#Test for Homogeneity of Variance
scipy.stats.bartlett(xmasSongsDecadeVsWoC['weeks_on_chartCBRT'], xmasSongsDecade
```

Out[3810]:

```
BartlettResult(statistic=2335.1791198158403, pvalue=0.0)
```

Unfortunately there is a violation of assumption of Homogeneity of Variance!

In [3811]:

```
#Recode decade information
def recode (dec):
    if dec == 1950:
        return "1950ies"
    if dec == 1960:
        return "1960ies"
    if dec == 1970:
        return "1970ies"
    if dec == 1980:
        return "1980ies"
    if dec == 1990:
        return "1990ies"
    if dec == 2000:
        return "2000ies"
    if dec == 2010:
        return "2010ies"

xmasSongsDecadeVsWoC['decadeR'] = xmasSongsDecadeVsWoC['decade'].apply(recode)
```

In [3812]:

```
#PostHoc
postHoc = MultiComparison(xmasSongsDecadeVsWoC['weeks_on_chart'], xmasSongsDecac
postHocResults = postHoc.tukeyhsd()
print(postHocResults)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
 group1  group2 meandiff p-adj   lower  upper  reject
----------------------------------------------------
1950ies 1960ies   0.0387    0.9 -4.1729 4.2503  False
1950ies 1970ies  -2.1981 0.7978 -6.9801  2.584  False
1950ies 1980ies      1.2    0.9 -3.7767 6.1767  False
1950ies 1990ies  -3.0238 0.6216 -8.4113 2.3637  False
1950ies 2000ies  -0.2171    0.9 -4.9056 4.4713  False
1950ies 2010ies   0.7371    0.9 -3.7564 5.2305  False
1960ies 1970ies  -2.2367 0.3344 -5.3425  0.869  False
1960ies 1980ies   1.1613    0.9 -2.2365 4.5592  False
1960ies 1990ies  -3.0625 0.2545 -7.0377 0.9127  False
1960ies 2000ies  -0.2558    0.9 -3.2154 2.7037  False
1960ies 2010ies   0.6984    0.9 -1.9415 3.3382  False
1970ies 1980ies   3.3981 0.1748 -0.6856 7.4817  False
1970ies 1990ies  -0.8258    0.9  -5.401 3.7495  False
1970ies 2000ies   1.9809 0.6732  -1.746 5.7078  False
1970ies 2010ies   2.9351 0.1618 -0.5433 6.4135  False
1980ies 1990ies  -4.2238 0.1231 -9.0022 0.5546  False
1980ies 2000ies  -1.4171    0.9 -5.3907 2.5565  False
1980ies 2010ies  -0.4629    0.9 -4.2045 3.2786  False
1990ies 2000ies   2.8067 0.5079 -1.6707  7.284  False
1990ies 2010ies   3.7609 0.1257 -0.5119 8.0336  False
2000ies 2010ies   0.9542    0.9 -2.3944 4.3028  False
----------------------------------------------------
```

Because of violation of the assumption of homegeineity run Welch's ANOVA

#Source: https://www.statology.org/welchs-anova-in-python/ (https://www.statology.org/welchs-anova-in-python/)

In [3813]:

```
import pingouin as pg
pg.welch_anova(dv='weeks_on_chartCBRT', between='decade', data=xmasSongsDecadeVs
```

Out[3813]:

	Source	ddof1	ddof2	F	p-unc	np2
0	decade	6	104.169835	2.545635	0.024323	0.027191

The overall p-value (0.024323) from the ANOVA table is less than α = .05, which means reject the null hypothesis that the weeks on charts are not being significantly influenced by decades.

In [3814]:

```
#Perform a Games-Howell post-hoc test to determine exactly which group means are
pg.pairwise_gameshowell(dv='weeks_on_chartCBRT', between='decade', data=xmasSong
```

Out[3814]:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	hed
0	1950	1960	2.031489	2.066530	-0.035041	0.115885	-0.302381	24.242897	0.900000	-0.07(
1	1950	1970	2.031489	1.927808	0.103681	0.119317	0.868957	26.931311	0.900000	0.22
2	1950	1980	2.031489	2.148121	-0.116632	0.132318	-0.881455	37.108025	0.900000	-0.23
3	1950	1990	2.031489	1.813400	0.218089	0.138072	1.579533	38.322909	0.672758	0.46
4	1950	2000	2.031489	1.937693	0.093796	0.142300	0.659142	46.837971	0.900000	0.16
5	1950	2010	2.031489	2.012137	0.019352	0.135717	0.142594	42.412082	0.900000	0.03
6	1960	1970	2.066530	1.927808	0.138723	0.057344	2.419148	99.855174	0.201557	0.41
7	1960	1980	2.066530	2.148121	-0.081591	0.080992	-1.007394	51.046900	0.900000	-0.18
8	1960	1990	2.066530	1.813400	0.253131	0.090087	2.809861	31.881580	0.104628	0.61(
9	1960	2000	2.066530	1.937693	0.128837	0.096441	1.335911	64.717113	0.811828	0.21
10	1960	2010	2.066530	2.012137	0.054394	0.086433	0.629314	95.964968	0.900000	0.09
11	1970	1980	1.927808	2.148121	-0.220314	0.085832	-2.566812	58.376828	0.155410	-0.57
12	1970	1990	1.927808	1.813400	0.114408	0.094461	1.211161	36.986919	0.879067	0.30
13	1970	2000	1.927808	1.937693	-0.009886	0.100540	-0.098327	71.780781	0.900000	-0.02(
14	1970	2010	1.927808	2.012137	-0.084329	0.090984	-0.926858	102.531054	0.900000	-0.17
15	1980	1990	2.148121	1.813400	0.334722	0.110428	3.031127	51.502563	0.054821	0.79
16	1980	2000	2.148121	1.937693	0.210428	0.115671	1.819199	82.949967	0.533772	0.39
17	1980	2010	2.148121	2.012137	0.135985	0.107469	1.265344	95.074611	0.853024	0.26(
18	1990	2000	1.813400	1.937693	-0.124294	0.122211	-1.017040	65.977260	0.900000	-0.24
19	1990	2010	1.813400	2.012137	-0.198737	0.114479	-1.736018	65.405832	0.582532	-0.40
20	2000	2010	1.937693	2.012137	-0.074443	0.119544	-0.622729	107.669912	0.900000	-0.11

Interpretation: Interestingly there is according to the p-value no significant difference between the means of the

decades. The closest to a signifcant difference is between the 80ies and 90ies.

In [3815]:

```
#Examine means
xmasSongsDecadeVsWoC.groupby('decadeR').mean()
```

Out[3815]:

decadeR	decade	weeks_on_chart	weeks_on_chartCBRT
1950ies	1950.0	9.857143	2.031489
1960ies	1960.0	9.895833	2.066530
1970ies	1970.0	7.659091	1.927808
1980ies	1980.0	11.057143	2.148121
1990ies	1990.0	6.833333	1.813400
2000ies	2000.0	9.640000	1.937693
2010ies	2010.0	10.594203	2.012137

Influence of peak-position

Null Hypothesis H0: **There is no siginifcant influence of peak-position of the Song on the weeks of presence in the Chart.**

Run Linear Regression.
IV: Peak-Position as continuous variable
DV: weeks of presence as continuous variable

In [3816]:

```
#extract only necessary data into a separate dataframe
dfPPvsWoC = dfXmasSongsRaw[['peak_position','weeks_on_chart']]
dfPPvsWoC[:4]
```

Out[3816]:

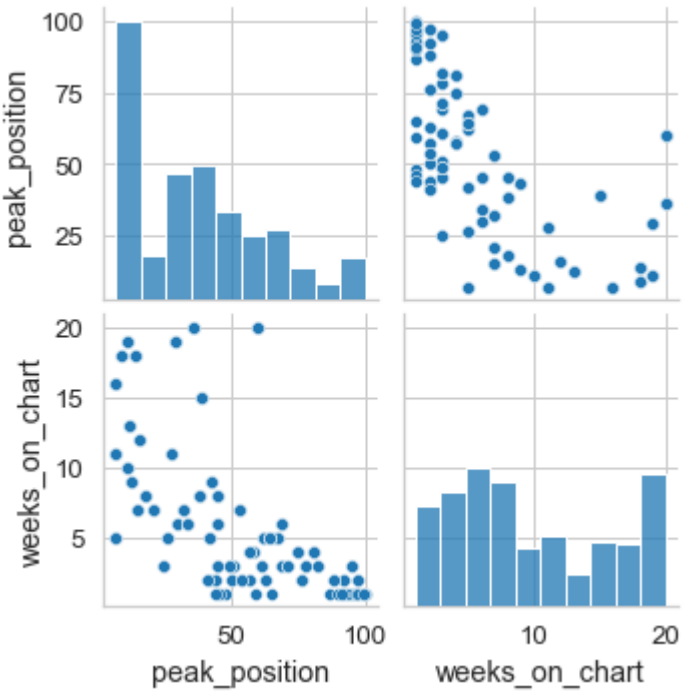
	peak_position	weeks_on_chart
0	49	3
1	49	3
2	49	3
3	96	1

In [3817]:

```
#Assumption Linearity
sns.pairplot(dfPPvsWoC)

```

Out[3817]:

```
<seaborn.axisgrid.PairGrid at 0x7fb9c87a2370>
```

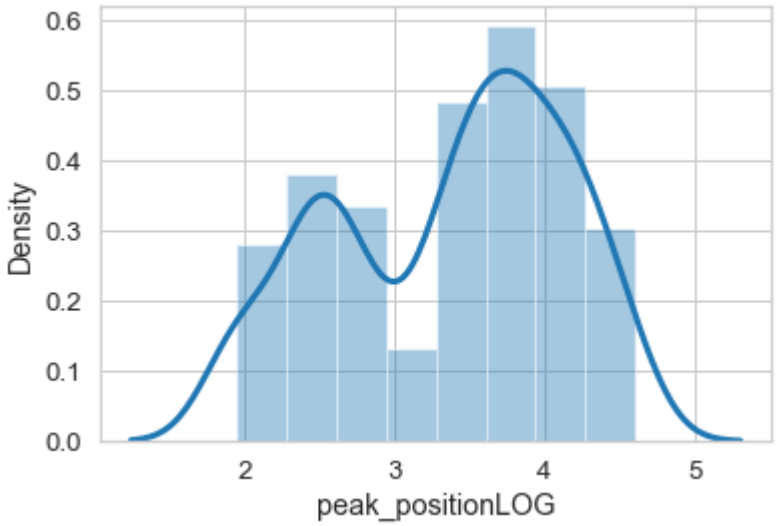


In [3818]:

```
#need some data wrangling
dfPPvsWoC['peak_positionLOG'] = np.log(dfPPvsWoC['peak_position'])
sns.distplot(dfPPvsWoC['peak_positionLOG'])
```

Out[3818]:

```
<AxesSubplot:xlabel='peak_positionLOG', ylabel='Density'>
```

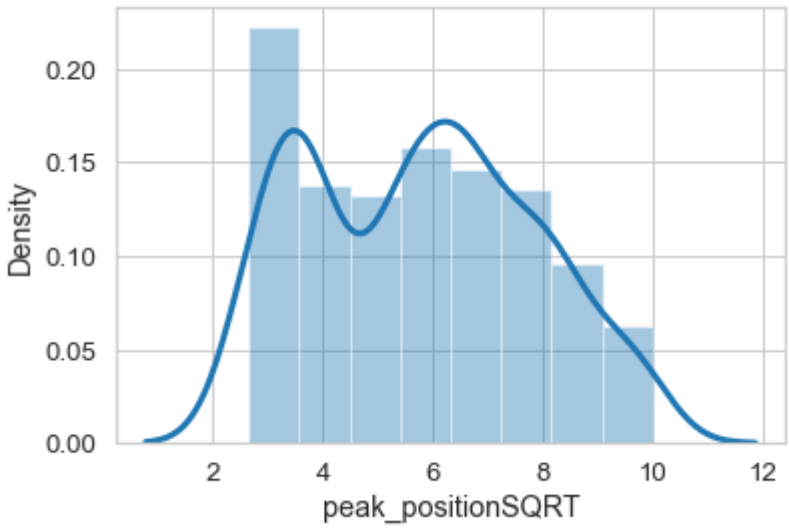


In [3819]:

```
dfPPvsWoC['peak_positionSQRT'] = np.sqrt(dfPPvsWoC['peak_position'])
sns.distplot(dfPPvsWoC['peak_positionSQRT'])
```

Out[3819]:

```
<AxesSubplot:xlabel='peak_positionSQRT', ylabel='Density'>
```

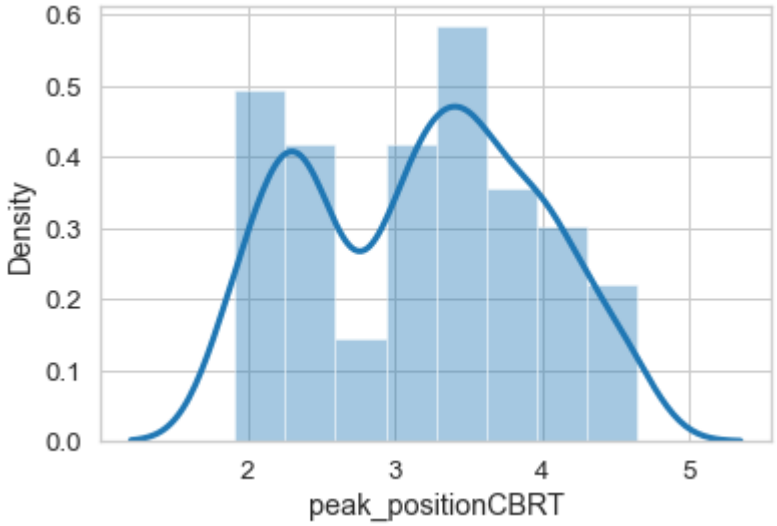


In [3820]:

```
dfPPvsWoC['peak_positionCBRT'] = np.cbrt(dfPPvsWoC['peak_position'])
sns.distplot(dfPPvsWoC['peak_positionCBRT'])
```

Out[3820]:

```
<AxesSubplot:xlabel='peak_positionCBRT', ylabel='Density'>
```



Keep SQRT

In [3821]:

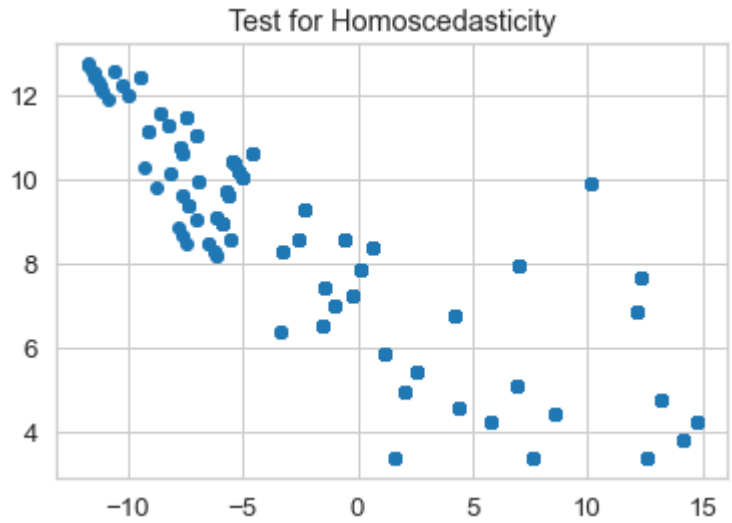
```
dfPPvsWoC.drop(['peak_positionLOG', 'peak_positionCBRT'], axis=1,inplace=True)
```

In [3822]:

```
#Testing for Homoscedasticity
x = dfPPvsWoC['peak_positionSQRT']
y = dfPPvsWoC['weeks_on_chart']
model = sm.OLS(y,x).fit()
pred_val = model.fittedvalues.copy()
true_val = dfPPvsWoC['weeks_on_chart'].values.copy()
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6, 4))
_ = ax.scatter(residual, pred_val)
plt.title('Test for Homoscedasticity')
```

Out[3822]:

```
Text(0.5, 1.0, 'Test for Homoscedasticity')
```



In [3823]:

```
sms.diagnostic.het_breuschpagan(residual, dfPPvsWoC[['peak_positionSQRT']])
```

Out[3823]:

```
(147.6126091376583, nan, 238.0178292677964, 3.582541644447623e-42)
```

Violation of the assumption of homoscedasticity, since this p-value is significant!

In [3824]:

```
sms.linear_harvey_collier(model)
```

Out[3824]:

```
Ttest_1sampResult(statistic=3.043024111303093, pvalue=0.00250366566961
15936)
```

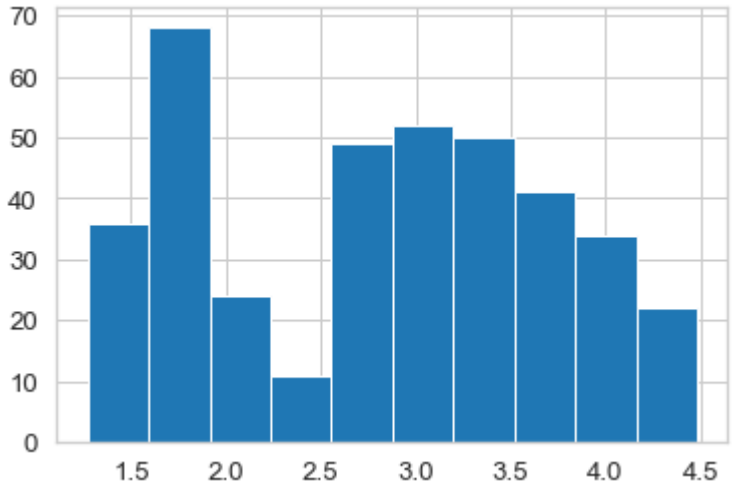
Still violation of the assumption of homoscedasticity, although less strong. Fix the problem:

In [3825]:

```
transformed, _ = boxcox(dfPPvsWoC['peak_positionSQRT'])
plt.hist(transformed)
```

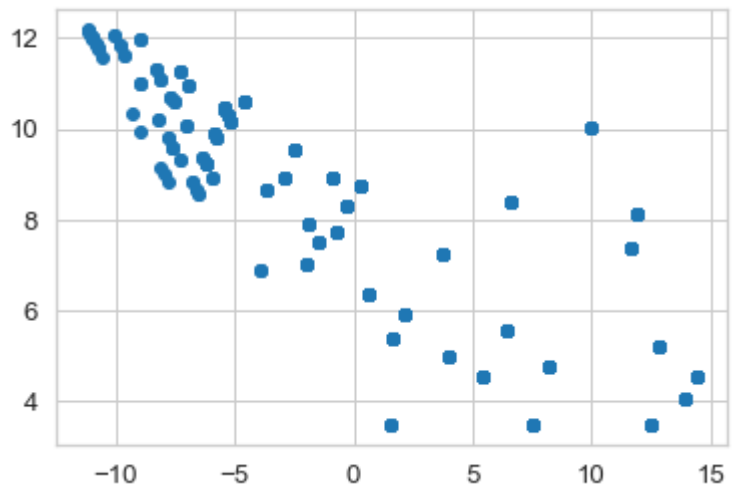
Out[3825]:

```
(array([36., 68., 24., 11., 49., 52., 50., 41., 34., 22.]),
 array([1.27045831, 1.59160458, 1.91275085, 2.23389712, 2.55504339,
        2.87618966, 3.19733593, 3.51848221, 3.83962848, 4.16077475,
        4.48192102]),
 <BarContainer object of 10 artists>)
```



In [3826]:

```
x = transformed
model1 = sm.OLS(y,x).fit()
pred_val = model1.fittedvalues.copy()
true_val = dfPPvsWoC['weeks_on_chart'].values.copy()
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6,4))
_ = ax.scatter(residual, pred_val)
```



In [3827]:

```
sms.diagnostic.het_breuschpagan(residual, dfPPvsWoC[['peak_positionSQRT']])
```

Out[3827]:

```
(149.34218659833198, nan, 242.5591786857345, 8.78829037643179e-43)
```

In [3828]:

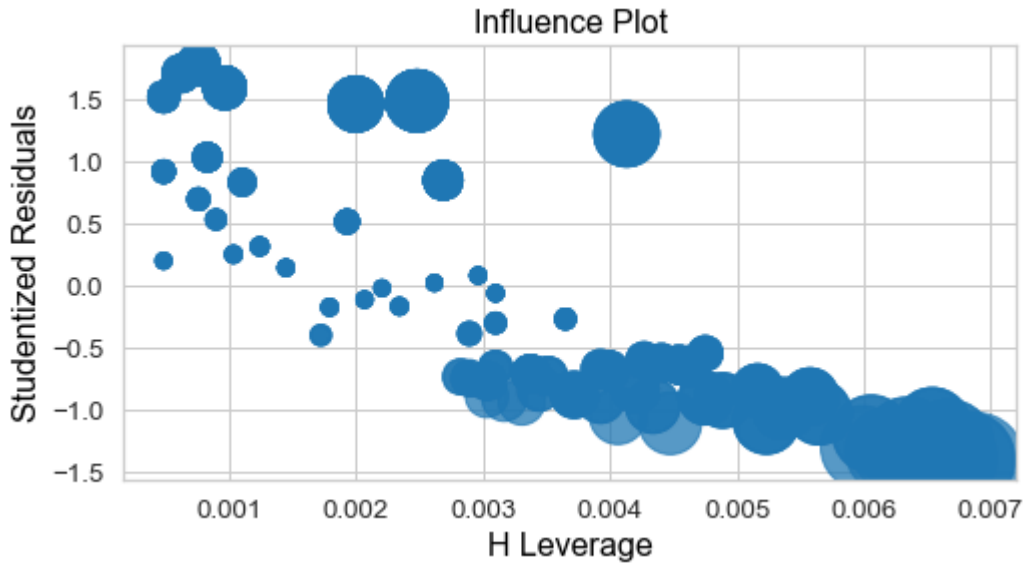
```
sms.linear_harvey_collier(model)
```

Out[3828]:

```
Ttest_1sampResult(statistic=3.043024111303093, pvalue=0.00250366566961
15936)
```

In [3829]:

```
#Screening for outliers
fig, ax = plt.subplots(figsize=(8,4))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



Looks like there are no outliers, at least none labelled!

In [3830]:

```
infl = model.get_influence()
infl.summary_frame()[:5]
```

Out[3830]:

	dfb_peak_positionSQRT	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid	
0	-0.041336	0.001711	-0.710958	0.003373	-0.041362	-0.710502	-0.0
1	-0.041336	0.001711	-0.710958	0.003373	-0.041362	-0.710502	-0.0
2	-0.041336	0.001711	-0.710958	0.003373	-0.041362	-0.710502	-0.0
3	-0.112772	0.012688	-1.380982	0.006609	-0.112639	-1.382611	-0.1
4	-0.054103	0.002928	-0.939762	0.003304	-0.054111	-0.939619	-0.0

In [3831]:

```
infl.summary_frame()['dfb_peak_positionSQRT'][:5]
# --> no problems here!
```

Out[3831]:

```
0   -0.041336
1   -0.041336
2   -0.041336
3   -0.112772
4   -0.054103
Name: dfb_peak_positionSQRT, dtype: float64
```

In [3832]:

```
infl.summary_frame()['dffits'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3832]:

```
106    0.078223
105    0.078223
104    0.078223
103    0.078223
102    0.078223
Name: dffits, dtype: float64
```

In [3833]:

```
infl.summary_frame()['hat_diag'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3833]:

```
252    0.006884
282    0.006815
372    0.006815
336    0.006678
363    0.006678
Name: hat_diag, dtype: float64
```

In [3834]:

```
infl.summary_frame()['student_resid'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3834]:

```
264    1.773563
260    1.773563
266    1.773563
265    1.773563
268    1.773563
Name: student_resid, dtype: float64
```

In [3835]:

```
model.summary()
```

Out[3835]:

OLS Regression Results

Dep. Variable:	weeks_on_chart	**R-squared (uncentered):**	0.467
Model:	OLS	**Adj. R-squared (uncentered):**	0.466
Method:	Least Squares	**F-statistic:**	338.7
Date:	Fri, 31 Dec 2021	**Prob (F-statistic):**	9.42e-55
Time:	18:53:28	**Log-Likelihood:**	-1370.1
No. Observations:	387	**AIC:**	2742.
Df Residuals:	386	**BIC:**	2746.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
peak_positionSQRT	1.2756	0.069	18.404	0.000	1.139	1.412

Omnibus:	518.565	**Durbin-Watson:**	0.377
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	27.467
Skew:	0.052	**Prob(JB):**	1.09e-06
Kurtosis:	1.699	**Cond. No.**	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Interpretation: **With p-value = 9.42e-55 reject H0 that peak-position has no significant influence on the weeks on the Chart of the song!**

In [3836]:

```
sqlPPvsWoC = sqldf("select distinct songid,peak_position, weeks_on_chart from df
sqlPPvsWoC[:5]
```

Out[3836]:

	songid	peak_position	weeks_on_chart
0	A Great Big SledThe Killers Featuring Toni Hal...	54	2
1	A Holly Jolly ChristmasBurl Ives	46	1
2	All I Want For Christmas Is YouMariah Carey	11	19
3	All I Want For Christmas Is YouMichael Buble	99	1
4	AmenThe Impressions	7	11

Create a Scatterplot of peak-position vs weeks on chart:

In [3837]:

```
sns.regplot(x="weeks_on_chart",
            y="peak_position",
            data=dfXmasSongsRaw);
plt.ylabel('Peak Position (globally)')
plt.xlabel('Weeks on Chart (globally)')
```

Out[3837]:

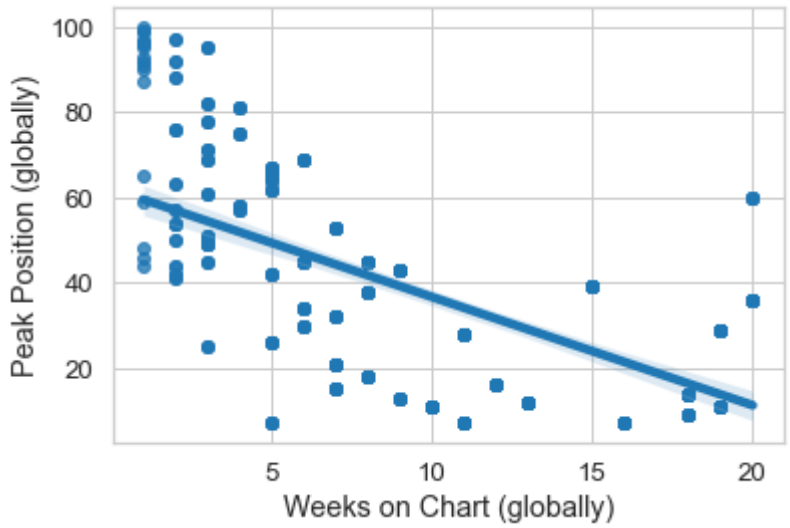
```
<AxesSubplot:xlabel='weeks_on_chart', ylabel='peak_position'>
```

Out[3837]:

```
Text(0, 0.5, 'Peak Position (globally)')
```

Out[3837]:

```
Text(0.5, 0, 'Weeks on Chart (globally)')
```



The plot confirms what could be expected: There is a negative slope of the linear regression, i.e. a negative correlation between the number of weeks of a song on the chart and its peak position. In other

words: the better a song was placed, the longer it stayed on the chart.

Influence of (month of) first appearance

Null Hypothesis H0: **There is no signifcant influence of the month of first appearance on the chart of the Song on the weeks of presence in the Chart.**

Run ANOVA.
IV: Month of first appearance as categorical variable (3 Levels) DV: weeks of presence as continuous variable

In [3838]:

```
sqlMofavsWoC = sqldf("select distinct date, songid,month, weeks_on_chart from df
sqlMofavsWoC[-30:-25]
```

Out[3838]:

	date	songid	month	weeks_on_chart
357	2014-12-27 00:00:00.000000	Have Yourself A Merry Little ChristmasSam Smith	12	1
358	2015-01-03 00:00:00.000000	Santa Tell MeAriana Grande	1	5
359	2015-01-03 00:00:00.000000	All I Want For Christmas Is YouMariah Carey	1	19
360	2015-01-10 00:00:00.000000	Santa Tell MeAriana Grande	1	5
361	2015-12-19 00:00:00.000000	All I Want For Christmas Is YouMariah Carey	12	19

In [3839]:

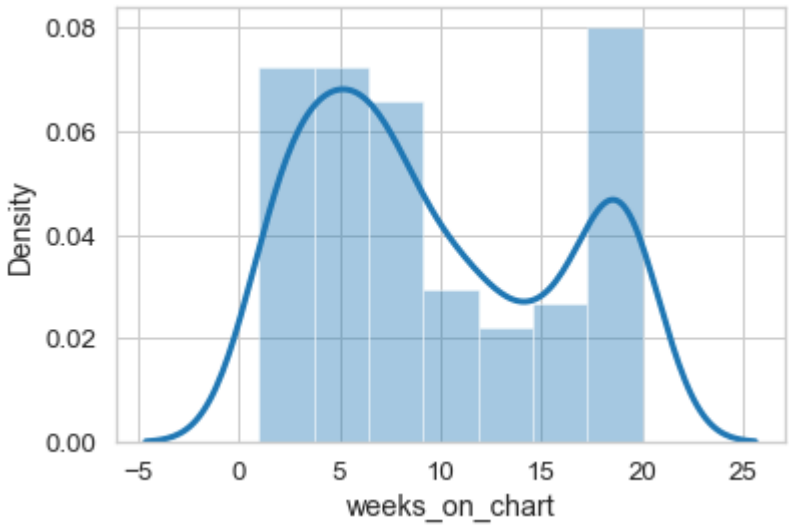
```
dfMofavsWoC = sqlMofavsWoC[['month','weeks_on_chart']]
```

In [3840]:

```
#Test for normality
#dfPPvsWoC['peak_positionSQRT'] = np.sqrt(dfPPvsWoC['peak_position'])
sns.distplot(dfMofavsWoC['weeks_on_chart'])

```

Out[3840]:

```
<AxesSubplot:xlabel='weeks_on_chart', ylabel='Density'>
```

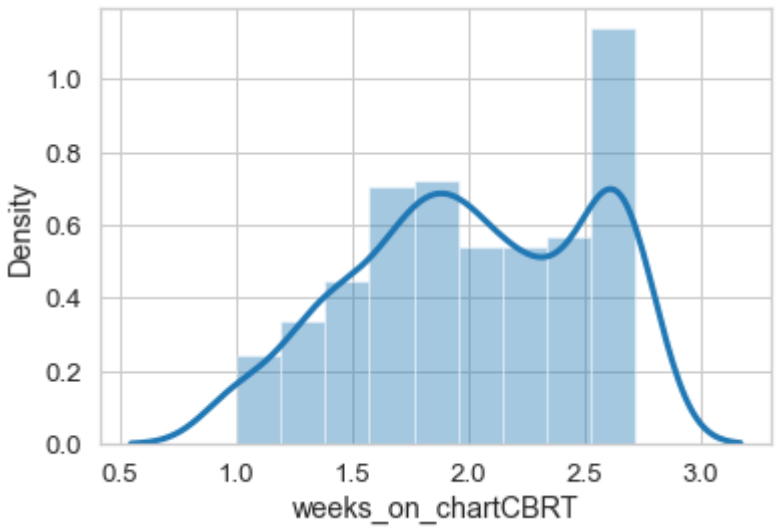


In [3841]:

```
#as done already prior to this analysis let's go with cube root
dfMofavsWoC['weeks_on_chartCBRT'] = np.cbrt(dfMofavsWoC['weeks_on_chart'])
sns.distplot(dfMofavsWoC['weeks_on_chartCBRT'])
```

Out[3841]:

```
<AxesSubplot:xlabel='weeks_on_chartCBRT', ylabel='Density'>
```



In [3842]:

```
#Test for Homogeneity of Variance
scipy.stats.bartlett(dfMofavsWoC['weeks_on_chartCBRT'], dfMofavsWoC['month'])
```

Out[3842]:

```
BartlettResult(statistic=1297.2255112190692, pvalue=4.5306188055798135
e-284)
```

Unfortunately there is a violation of assumption of Homogeneity of Variance!

In [3843]:

```
#Recode month information
def recode (month):
    if month == 11:
        return "Nov"
    if month == 12:
        return "Dec"
    if month == 1:
        return "Jan"

dfMofavsWoC['monthR'] = dfMofavsWoC['month'].apply(recode)
```

In [3844]:

```
#PostHoc
postHoc = MultiComparison(dfMofavsWoC['weeks_on_chart'], dfMofavsWoC['monthR'])
postHocResults = postHoc.tukeyhsd()
print(postHocResults)
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj  lower  upper  reject
-------------------------------------------------
   Dec    Jan  -0.5321 0.6725 -2.051 0.9868  False
   Dec    Nov   4.1792 0.0283 0.3549 8.0035   True
   Jan    Nov   4.7113 0.0127 0.8258 8.5967   True
-------------------------------------------------
```

Because of violation of the assumption of homegeineity run Welch's ANOVA

#Source: https://www.statology.org/welchs-anova-in-python/ (https://www.statology.org/welchs-anova-in-python/)

In [3845]:

```
import pingouin as pg
pg.welch_anova(dv='weeks_on_chart', between='month', data=dfMofavsWoC)
```

Out[3845]:

	Source	ddof1	ddof2	F	p-unc	np2
0	month	2	40.701053	6.730975	0.002982	0.020819

The overall p-value (0.002982) from the ANOVA table is less than α = .05, which means reject the null hypothesis that the month of first appearance on the chart does not influence significantly the number

of weeks on the chart.

In [3846]:

```
#Perform a Games-Howell post-hoc test to determine exactly which group means are
pg.pairwise_gameshowell(dv='weeks_on_chart', between='month', data=dfMofavsWoC)
```

Out[3846]:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	hedges
0	1	11	9.155405	13.866667	-4.711261	1.277051	-3.689171	19.238816	0.004161	-0.99498
1	1	12	9.155405	9.687500	-0.532095	0.645030	-0.824914	324.378940	0.672241	-0.08720
2	11	12	13.866667	9.687500	4.179167	1.250446	3.342140	17.718434	0.009785	0.88853

The p-values in the table above show:

- **the mean difference between January and November are significantly different**
- **the mean difference between November and December are significantly different**

Also: the average duration of presence on the chart is longest if the song first appeared in November, then in December and finally in January - which was to be expected. Although the difference between December and January is minimal.

Influence of time-to-christmas of first appearance on weeks on the Chart

Null Hypothesis H0: **There is no significant influence of of time-to-christmas of first appearance of the Song on the weeks of presence in the Chart.**

Run Linear Regression.
IV: Days to Christmas as continuous variable
DV: weeks of presence as continuous variable

In [3847]:

```
# get the songs with the relevant information
sqlUniqueSongsT2Xmas = sqldf("""select distinct songid,date,numberOfDaysToXmas,w
                            from dfXmasSongsRaw
                            group by songid order by 3 asc;""")
print(sqlUniqueSongsT2Xmas.info())
sqlUniqueSongsT2Xmas[:5]

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 4 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   songid              78 non-null     object
 1   date                78 non-null     object
 2   numberOfDaysToXmas  78 non-null     int64 
 3   weeks_on_chart      78 non-null     int64 
dtypes: int64(2), object(2)
memory usage: 2.6+ KB
None
```

Out[3847]:

	songid	date	numberOfDaysToXmas	weeks_on_chart
0	Grandma Got Run Over By A ReindeerElmo & Patsy	1998-01-10 00:00:00.000000	-16	1
1	All I Want For Christmas Is YouMariah Carey	2000-01-08 00:00:00.000000	-14	19
2	Shake Up ChristmasTrain	2011-01-08 00:00:00.000000	-14	1
3	A Holly Jolly ChristmasBurl Ives	2017-01-07 00:00:00.000000	-13	1
4	Feliz NavidadJose Feliciano	2017-01-07 00:00:00.000000	-13	1

In [3848]:

```
#extract only necessary data into a separate dataframe
dfUniqueSongsT2Xmas = sqlUniqueSongsT2Xmas[['numberOfDaysToXmas','weeks_on_chart
dfUniqueSongsT2Xmas[:4]
```

Out[3848]:

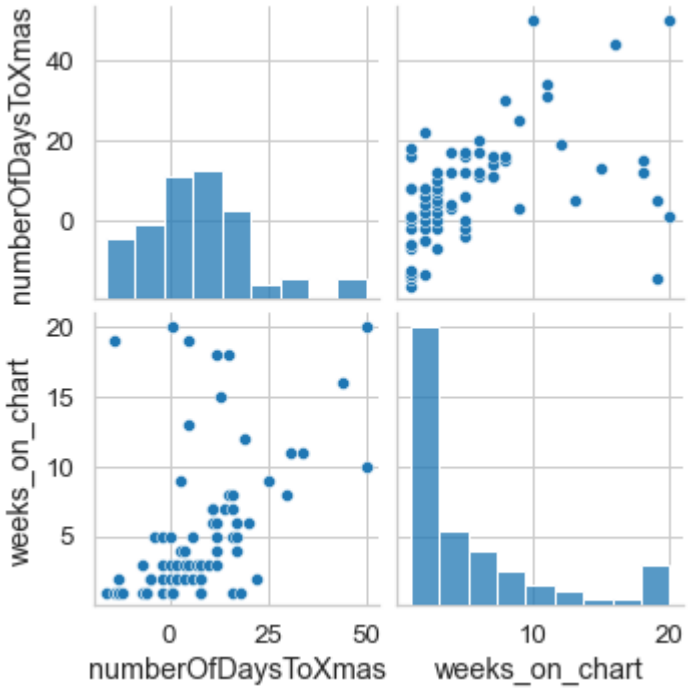
	numberOfDaysToXmas	weeks_on_chart
0	-16	1
1	-14	19
2	-14	1
3	-13	1

In [3849]:

```
#Assumption Linearity
sns.pairplot(dfUniqueSongsT2Xmas)
```

Out[3849]:

```
<seaborn.axisgrid.PairGrid at 0x7fb9b8639760>
```

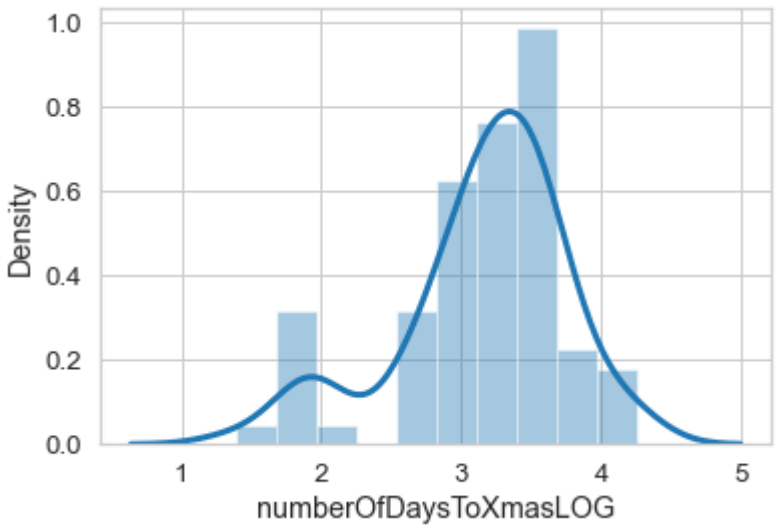


In [3850]:

```
#need some data wrangling - try Log with adding 20 to avoid log on negative numb
dfUniqueSongsT2Xmas['numberOfDaysToXmasLOG'] = np.log(dfUniqueSongsT2Xmas['numbe
sns.distplot(dfUniqueSongsT2Xmas['numberOfDaysToXmasLOG'])
```

Out[3850]:

```
<AxesSubplot:xlabel='numberOfDaysToXmasLOG', ylabel='Density'>
```

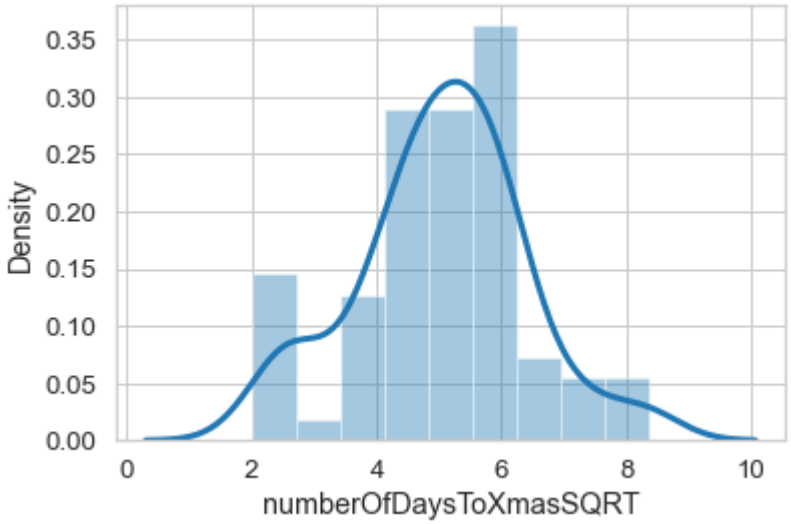


In [3851]:

```
#need some data wrangling - try SQRT with adding 20 to avoid sqrt on negative nu
dfUniqueSongsT2Xmas['numberOfDaysToXmasSQRT'] = np.sqrt(dfUniqueSongsT2Xmas['num
sns.distplot(dfUniqueSongsT2Xmas['numberOfDaysToXmasSQRT'])
```

Out[3851]:

```
<AxesSubplot:xlabel='numberOfDaysToXmasSQRT', ylabel='Density'>
```



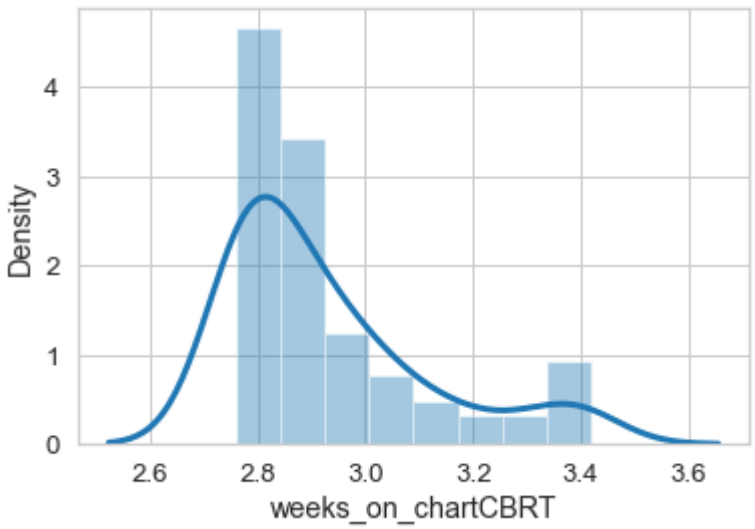
#SQRT looks better than log! --> Keep SQRT

In [3852]:

```
#as for weeks on chart - keep with cube root as explored in prior analysis
dfUniqueSongsT2Xmas['weeks_on_chartCBRT'] = np.cbrt(dfUniqueSongsT2Xmas['weeks_c
sns.distplot(dfUniqueSongsT2Xmas['weeks_on_chartCBRT'])
```

Out[3852]:

```
<AxesSubplot:xlabel='weeks_on_chartCBRT', ylabel='Density'>
```



In [3853]:

```
1 dfUniqueSongsT2Xmas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   numberOfDaysToXmas      78 non-null     int64  
 1   weeks_on_chart          78 non-null     int64  
 2   numberOfDaysToXmasLOG   78 non-null     float64
 3   numberOfDaysToXmasSQRT  78 non-null     float64
 4   weeks_on_chartCBRT      78 non-null     float64
dtypes: float64(3), int64(2)
memory usage: 3.2 KB
```

In [3854]:

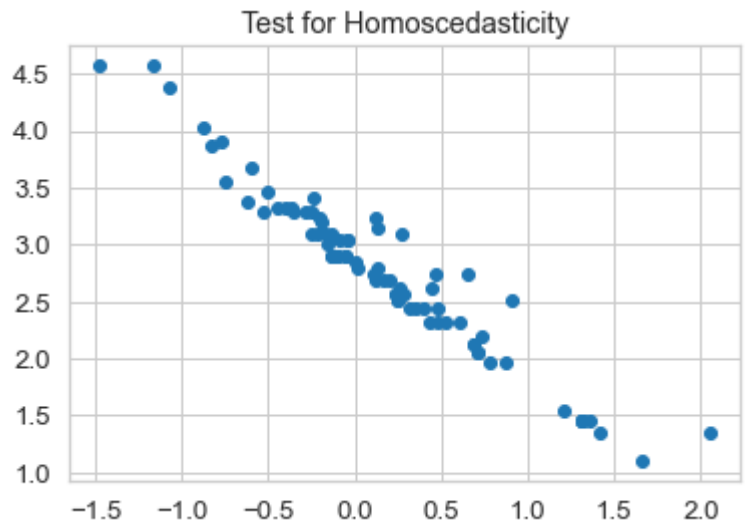
```
1 dfUniqueSongsT2Xmas.drop(['numberOfDaysToXmasLOG'], axis=1,inplace=True)
```

In [3855]:

```
1  #Testing for Homoscedasticity
2  x = dfUniqueSongsT2Xmas['numberOfDaysToXmasSQRT']
3  y = dfUniqueSongsT2Xmas['weeks_on_chartCBRT']
4  model = sm.OLS(y,x).fit()
5  pred_val = model.fittedvalues.copy()
6  true_val = dfUniqueSongsT2Xmas['weeks_on_chartCBRT'].values.copy()
7  residual = true_val - pred_val
8  fig, ax = plt.subplots(figsize=(6, 4))
9  _ = ax.scatter(residual, pred_val)
10 plt.title('Test for Homoscedasticity')
```

Out[3855]:

```
Text(0.5, 1.0, 'Test for Homoscedasticity')
```



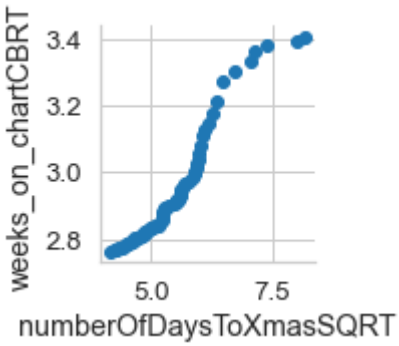
Looks fairly linear!

In [3856]:

```
from seaborn_qqplot import pplot
pplot(dfUniqueSongsT2Xmas, x = "numberOfDaysToXmasSQRT", y = "weeks_on_chartCBRT
```

Out[3856]:

```
<seaborn.axisgrid.PairGrid at 0x7fb9edfe4850>
```



In [3857]:

```
sms.diagnostic.het_breuschpagan(residual, dfUniqueSongsT2Xmas[['numberOfDaysToXm
```

Out[3857]:

```
(14.181343929027204, nan, 17.1104117473223, 8.919127906422687e-05)
```

Violation of the assumption of homoscedasticity, since this p-value with 8.919e-05 is significant!

In [3858]:

```
sms.linear_harvey_collier(model)
```

Out[3858]:

```
Ttest_1sampResult(statistic=-22.675055942190262, pvalue=4.836463533115
069e-35)
```

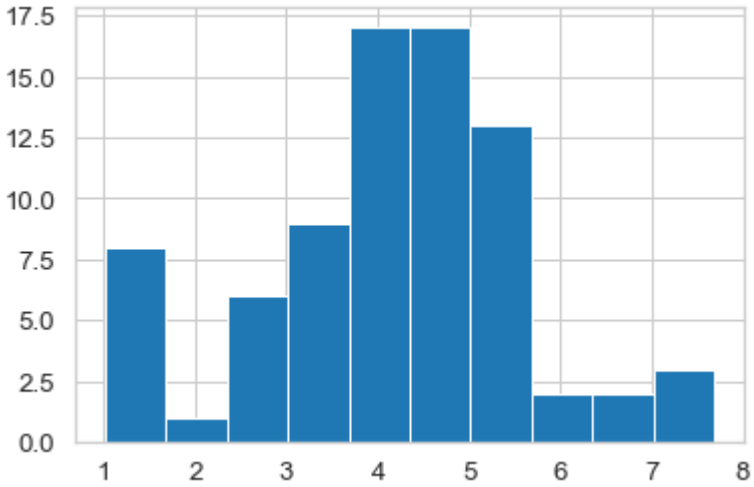
Still violation of the assumption of homoscedasticity. Fix the problem:

In [3859]:

```
transformed, _ = boxcox(dfUniqueSongsT2Xmas['numberOfDaysToXmasSQRT'])
plt.hist(transformed)
```

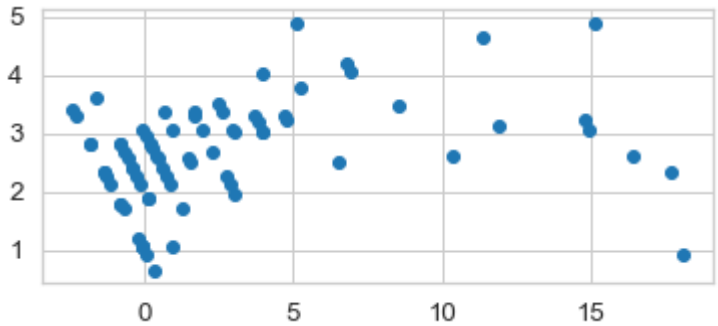
Out[3859]:

```
(array([ 8.,  1.,  6.,  9., 17., 17., 13.,  2.,  2.,  3.]),
 array([1.01161627, 1.67894782, 2.34627936, 3.01361091, 3.68094246,
        4.348274  , 5.01560555, 5.6829371 , 6.35026864, 7.01760019,
        7.68493173]),
 <BarContainer object of 10 artists>)
```



In [3860]:

```
x = transformed
model1 = sm.OLS(y,x).fit()
pred_val = model1.fittedvalues.copy()
true_val = dfUniqueSongsT2Xmas['weeks_on_chart'].values.copy()
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6,2.5))
_ = ax.scatter(residual, pred_val)
```



In [3861]:

```
sms.diagnostic.het_breuschpagan(residual, dfUniqueSongsT2Xmas[['numberOfDaysToXm
```

Out[3861]:

```
(13.641973241888072, nan, 16.321692763723227, 0.0001254620084345871)
```

Better - but still violation of homoscedasticity.

In [3862]:

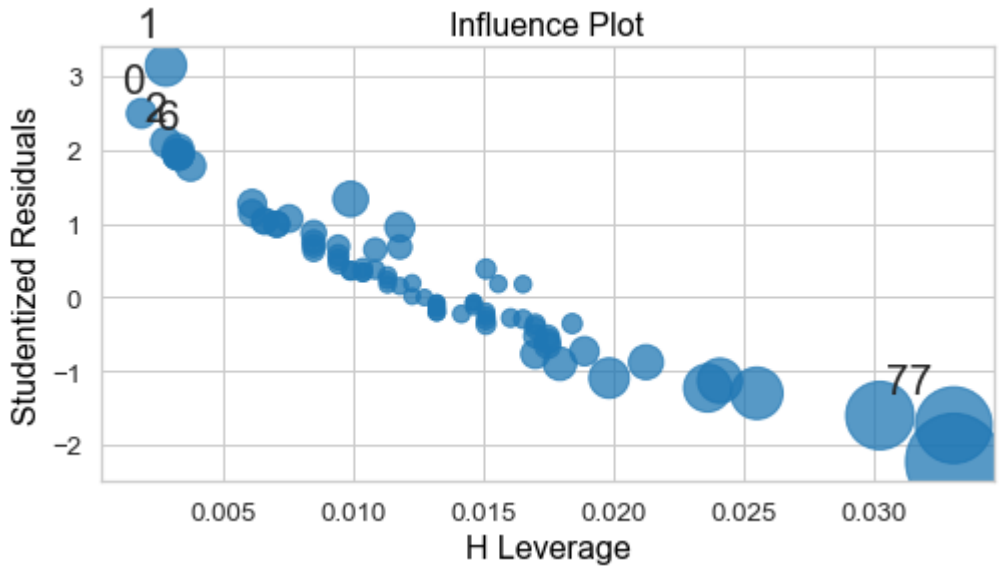
```
sms.linear_harvey_collier(model)
```

Out[3862]:

```
Ttest_1sampResult(statistic=-22.675055942190262, pvalue=4.836463533115
069e-35)
```

In [3863]:

```
#Screening for outliers
fig, ax = plt.subplots(figsize=(8,4))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



Looks like there are a few outliers. Remove them! Keep on repeating this process until all outliers are identified and removed!

In [3864]:

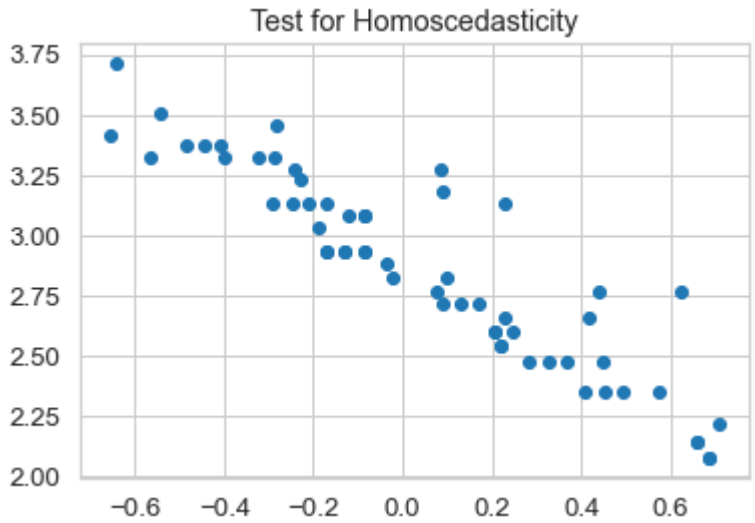
```
exclList=[0,1,2,3,4,5,6,7,8,9,10,24,70,72,73,74,75,76,77]
dfUniqueSongsT2XmasCleanedFromOutliers = dfUniqueSongsT2Xmas.drop(exclList)
```

In [3865]:

```
#re-create model:
#Testing for Homoscedasticity
x = dfUniqueSongsT2XmasCleanedFromOutliers['numberOfDaysToXmasSQRT']
y = dfUniqueSongsT2XmasCleanedFromOutliers['weeks_on_chartCBRT']
model = sm.OLS(y,x).fit()
pred_val = model.fittedvalues.copy()
true_val = dfUniqueSongsT2XmasCleanedFromOutliers['weeks_on_chartCBRT'].values.c
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6, 4))
_ = ax.scatter(residual, pred_val)
plt.title('Test for Homoscedasticity')
```

Out[3865]:

```
Text(0.5, 1.0, 'Test for Homoscedasticity')
```



In [3866]:

```
sms.diagnostic.het_breuschpagan(residual, dfUniqueSongsT2XmasCleanedFromOutliers
# --> p-Value = 0.0057, still < 0.05, assumption of Homoscedasticity not met
```

Out[3866]:

```
(7.334163019210957, nan, 8.2333216680958, 0.00572904828315419)
```

In [3867]:

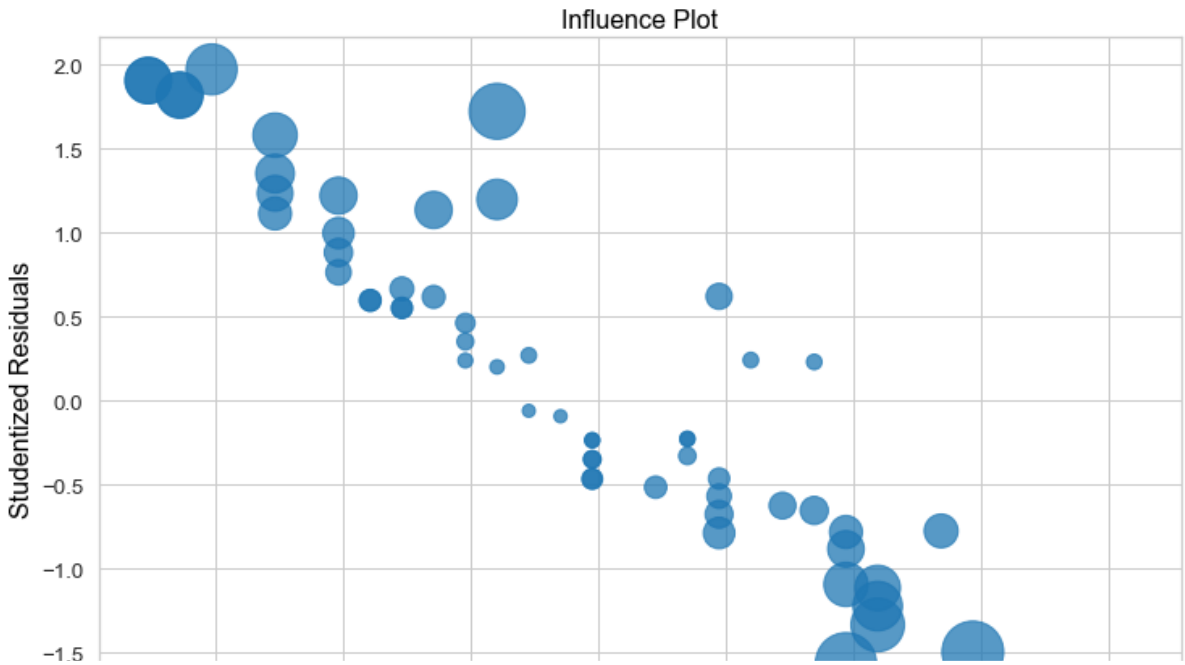
```
# Testing for Multicollinearity:
dfUniqueSongsT2XmasCleanedFromOutliers.corr()
```

Out[3867]:

	numberOfDaysToXmas	weeks_on_chart	numberOfDaysToXmasSQRT
numberOfDaysToXmas	1.000000	0.401677	0.997158
weeks_on_chart	0.401677	1.000000	0.403531
numberOfDaysToXmasSQRT	0.997158	0.403531	1.000000
weeks_on_chartCBRT	0.423257	0.997900	0.424786

In [3868]:

```
#Screening for Outliers
fig, ax = plt.subplots(figsize=(12,8))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



In [3869]:

```
infl = model.get_influence()
infl.summary_frame()[:5]
```

Out[3869]:

	dfb_numberOfDaysToXmasSQRT	cooks_d	standard_resid	hat_diag	dffits_internal	student_r
11	0.178393	0.030444	1.863532	0.008690	0.174481	1.905
12	0.178393	0.030444	1.863532	0.008690	0.174481	1.905
13	0.176324	0.029900	1.783643	0.009311	0.172917	1.818
14	0.176324	0.029900	1.783643	0.009311	0.172917	1.818
15	0.197487	0.037151	1.924450	0.009932	0.192746	1.971

In [3870]:

```
infl.summary_frame()['dfb_numberOfDaysToXmasSQRT'].sort_values(ascending=False)[
# --> no problem here

```

Out[3870]:

```
36    0.216206
15    0.197487
11    0.178393
12    0.178393
13    0.176324
Name: dfb_numberOfDaysToXmasSQRT, dtype: float64
```

In [3871]:

```
infl.summary_frame()['dffits'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3871]:

```
36    0.216206
15    0.197487
11    0.178393
12    0.178393
13    0.176324
Name: dffits, dtype: float64
```

In [3872]:

```
infl.summary_frame()['hat_diag'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3872]:

```
71    0.027933
69    0.024829
68    0.024209
67    0.023588
66    0.022967
Name: hat_diag, dtype: float64
```

In [3873]:

```
infl.summary_frame()['student_resid'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3873]:

```
15    1.971779
11    1.905318
12    1.905318
14    1.818785
13    1.818785
Name: student_resid, dtype: float64
```

In [3874]:

```
model.summary()
```

Out[3874]:

OLS Regression Results

Dep. Variable:	weeks_on_chartCBRT	**R-squared (uncentered):**	0.984
Model:	OLS	**Adj. R-squared (uncentered):**	0.984
Method:	Least Squares	**F-statistic:**	3612.
Date:	Fri, 31 Dec 2021	**Prob (F-statistic):**	6.09e-54
Time:	18:53:30	**Log-Likelihood:**	-24.538
No. Observations:	59	**AIC:**	51.08
Df Residuals:	58	**BIC:**	53.15
Df Model:	1		
Covariance Type:	nonrobust		

Interpretation:

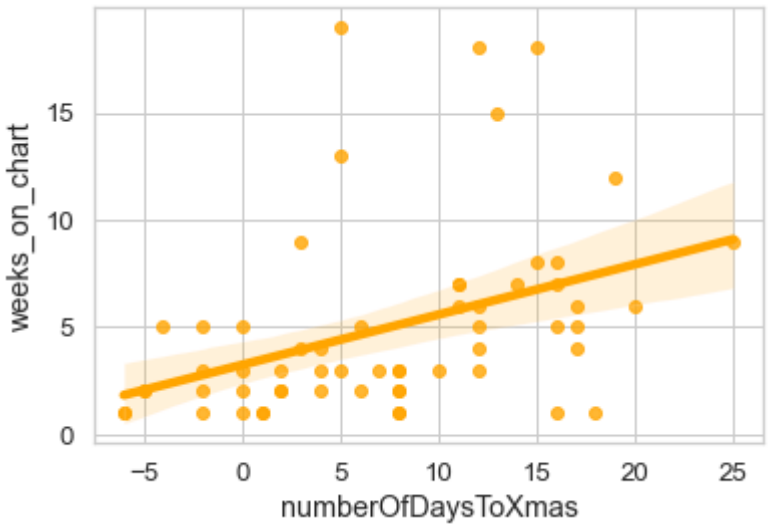
In this model (exclusion of outliers!):

- **The number of days to (or from) Christmas explains 98.4% (!) of the variability in number of weeks on chart (R-squared value).**
- **Reject H0: p-value = 6.09e-54; The number of days to (or from) Christmas significantly influence the number of weeks on chart.**

Create a Scatterplot of the dataset without the detected outliers:

In [3875]:

```
sns.regplot(y="weeks_on_chart", x="numberOfDaysToXmas", data=dfUniqueSongsT2Xmas
```



The plot confirms what could be expected: There is a slight positive slope of the linear regression, i.e. a

positive correlation between the number of weeks of a song on the chart and the number of days to or from Christmas. In other words: The further ahead from Christmas the song first appeared in the chart, the longer it stayed on the chart.

Run another linear regression model...

In [3876]:

```
y = dfUniqueSongsT2Xmas['weeks_on_chart'] # dependent variable
x = dfUniqueSongsT2Xmas['numberOfDaysToXmas'] # independent variable
lm = sm.OLS(y,x).fit() # fitting the model
lm.predict(x)
```

Out[3876]:

```
0     -4.912945
1     -4.298827
2     -4.298827
3     -3.991768
4     -3.991768
        ...
73     9.518831
74    10.440009
75    13.510599
76    15.352954
77    15.352954
Length: 78, dtype: float64
```

In [3877]:

```
1 lm.summary()
```

Out[3877]:

OLS Regression Results

Dep. Variable:	weeks_on_chart	**R-squared (uncentered):**	0.398
Model:	OLS	**Adj. R-squared (uncentered):**	0.390
Method:	Least Squares	**F-statistic:**	50.89
Date:	Fri, 31 Dec 2021	**Prob (F-statistic):**	4.64e-10
Time:	18:53:30	**Log-Likelihood:**	-248.20
No. Observations:	78	**AIC:**	498.4
Df Residuals:	77	**BIC:**	500.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
numberOfDaysToXmas	0.3071	0.043	7.134	0.000	0.221	0.393

Omnibus:	41.875	**Durbin-Watson:**	1.180
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	111.612
Skew:	1.817	**Prob(JB):**	5.80e-25
Kurtosis:	7.597	**Cond. No.**	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

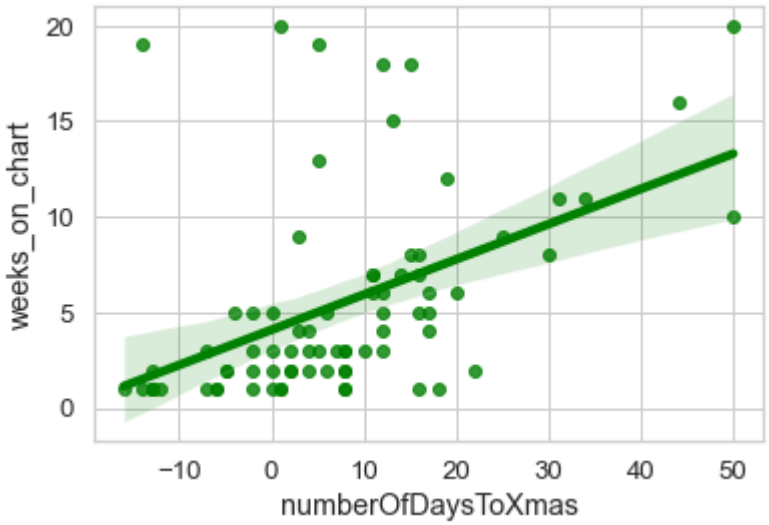
Interpretation:

- **The number of days to (or from) Christmas explains 39.8% of the variability in number of weeks on chart (R-squared value).**
- **Reject H0: p-value = 4.64e-10; The number of days to (or from) Christmas significantly influence the number of weeks on chart.**

Create a Scatterplot of days to Christmas versus weeks on chart:

In [3878]:

```
1 sns.regplot(y="weeks_on_chart", x="numberOfDaysToXmas", data=dfUniqueSongsT2Xmas
```



Influence of instances of a song

Null Hypothesis H0: **There is no significant influence of instances of a Song on the weeks of presence in the Chart.**

Run Linear Regression.
IV: Instance as continuous variable
DV: weeks of presence as continuous variable

In [3879]:

```
# get the songs with the relevant information
sqlUniqueSongsInstance = sqldf("select songid,max(instance) as maxinstance,weeks_on
print(sqlUniqueSongsInstance.info())
sqlUniqueSongsInstance[:5]

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   songid          78 non-null     object
 1   maxinstance     78 non-null     int64 
 2   weeks_on_chart  78 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 2.0+ KB
None
```

Out[3879]:

	songid	maxinstance	weeks_on_chart
0	Rockin' Around The Christmas TreeBrenda Lee	6	18
1	Jingle Bell RockBobby Helms	6	19
2	All I Want For Christmas Is YouMariah Carey	6	19
3	White ChristmasBing Crosby	5	13
4	The Christmas Song (Merry Christmas To You)Nat...	5	8

In [3880]:

```
#extract only necessary data into a separate dataframe
dfUniqueSongsMaxInstance = sqlUniqueSongsInstance[['maxinstance','weeks_on_chart
dfUniqueSongsMaxInstance[:4]
```

Out[3880]:

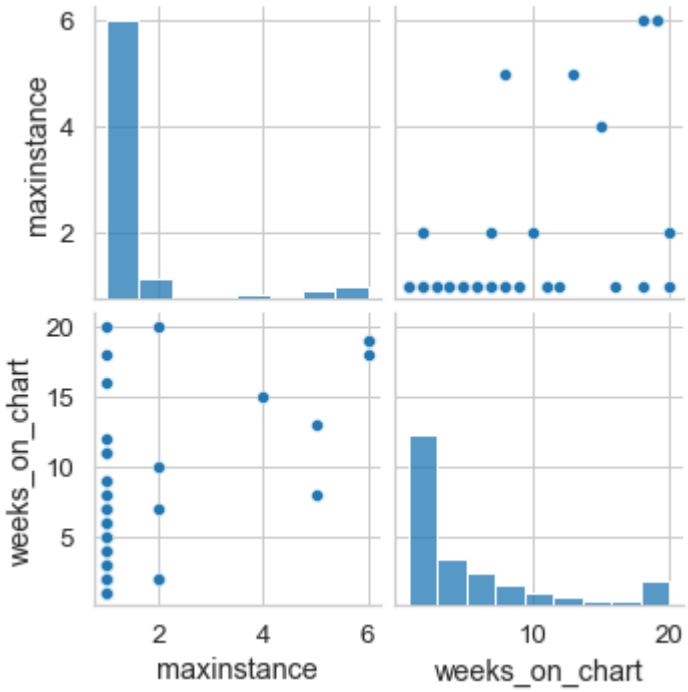
	maxinstance	weeks_on_chart
0	6	18
1	6	19
2	6	19
3	5	13

In [3881]:

```
#Assumption Linearity
sns.pairplot(sqlUniqueSongsInstance)
```

Out[3881]:

```
<seaborn.axisgrid.PairGrid at 0x7fb9c8c7efa0>
```

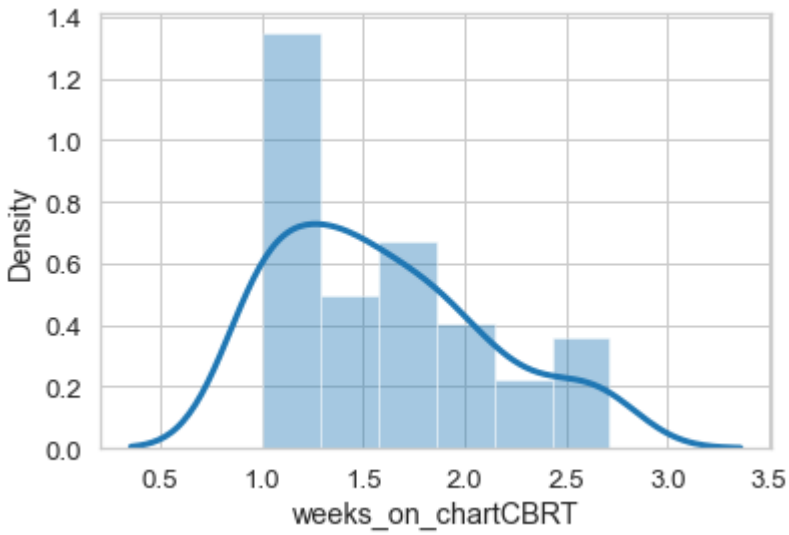


In [3882]:

```
#need some data wrangling - try Log
dfUniqueSongsMaxInstance['weeks_on_chartCBRT'] = np.cbrt(dfUniqueSongsMaxInstanc
sns.distplot(dfUniqueSongsMaxInstance['weeks_on_chartCBRT'])
```

Out[3882]:

```
<AxesSubplot:xlabel='weeks_on_chartCBRT', ylabel='Density'>
```

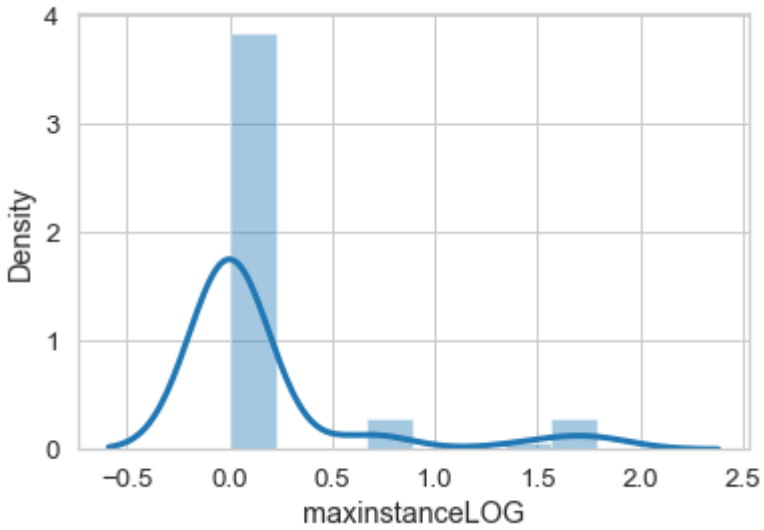


In [3883]:

```
#need some data wrangling - try Log
dfUniqueSongsMaxInstance['maxinstanceLOG'] = np.log(dfUniqueSongsMaxInstance['ma
sns.distplot(dfUniqueSongsMaxInstance['maxinstanceLOG'])
```

Out[3883]:

```
<AxesSubplot:xlabel='maxinstanceLOG', ylabel='Density'>
```

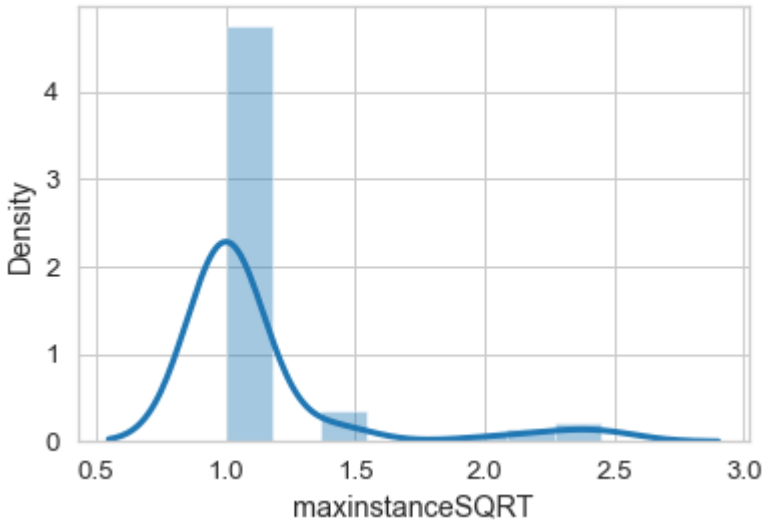


In [3884]:

```
#need some data wrangling - try sqrt
dfUniqueSongsMaxInstance['maxinstanceSQRT'] = np.sqrt(dfUniqueSongsMaxInstance['
sns.distplot(dfUniqueSongsMaxInstance['maxinstanceSQRT'])
```

Out[3884]:

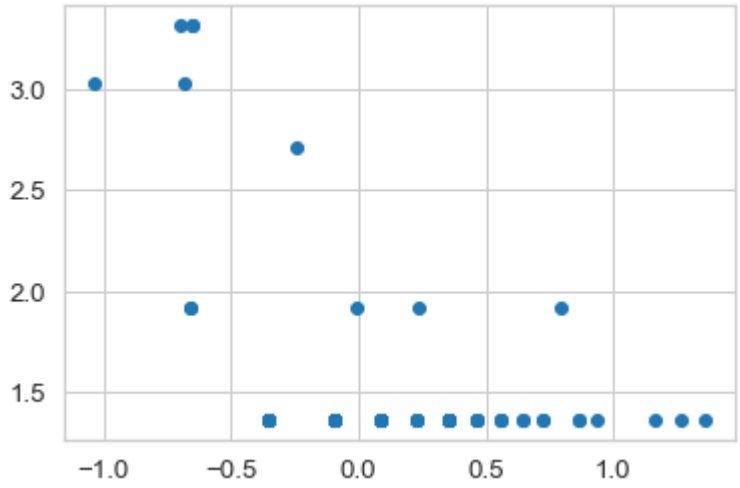
```
<AxesSubplot:xlabel='maxinstanceSQRT', ylabel='Density'>
```



SQRT is better - keep it!

In [3885]:

```
#Testing for Homoscedasticity
x = dfUniqueSongsMaxInstance['maxinstanceSQRT']
y = dfUniqueSongsMaxInstance['weeks_on_chartCBRT']
model = sm.OLS(y,x).fit()
pred_val = model.fittedvalues.copy()
true_val = dfUniqueSongsMaxInstance['weeks_on_chartCBRT'].values.copy()
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6, 4))
_ = ax.scatter(residual, pred_val)
```



In [3886]:

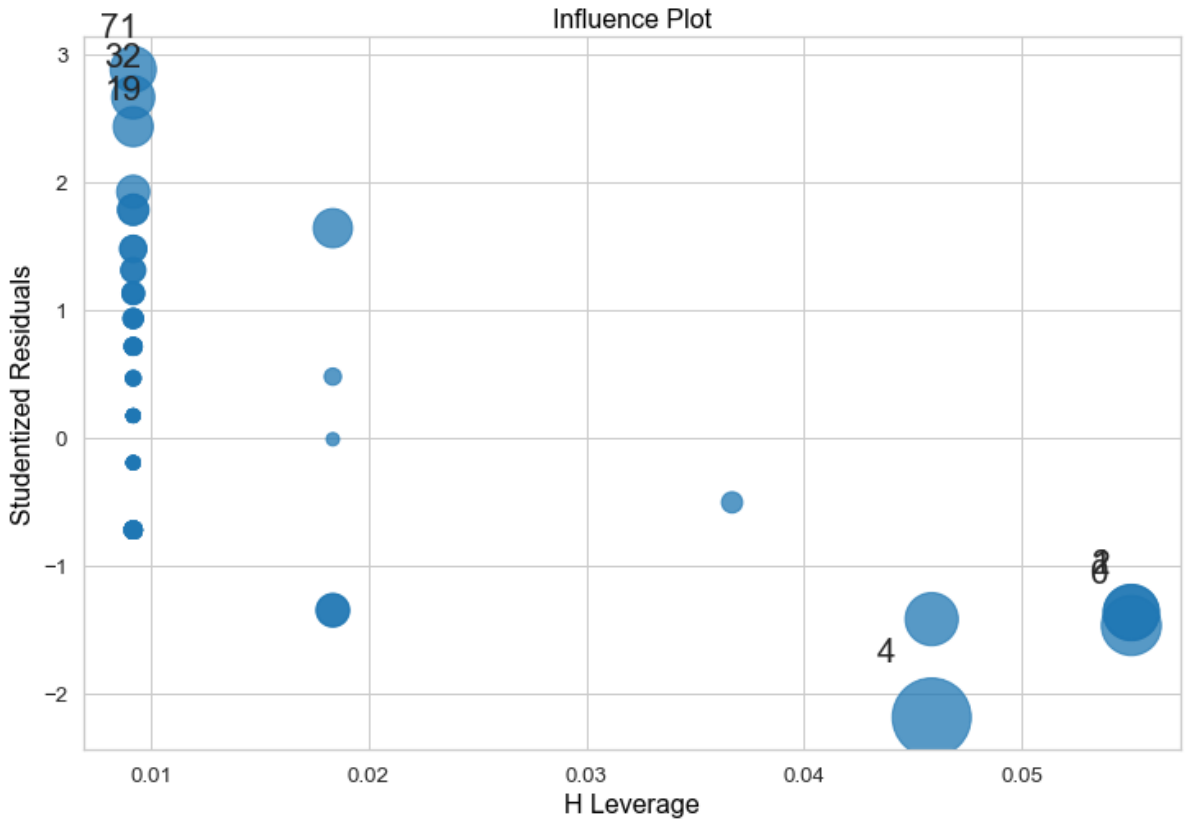
```
sms.diagnostic.het_breuschpagan(residual, dfUniqueSongsMaxInstance[['maxinstance
# --> p-Value = 7.101e-09, still < 0.05, assumption of Homoscedasticity not met
```

Out[3886]:

```
(27.658842963597795, nan, 42.305958654406766, 7.100955999260789e-09)
```

In [3887]:

```
#Screening for Outliers
fig, ax = plt.subplots(figsize=(12,8))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



In [3888]:

```
#Looks like there are a few outliers. Remove them! Keep on repeating this proces
exclList=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,19,24,32,36,67,70,71,72,73,74,75
exclList=[0,1,2,3,4,6,9,10,11,19,23,32,52,60,63,71,75]
dfUniqueSongsMaxInstanceCleanedFromOutliers = dfUniqueSongsMaxInstance.drop(excl

```

In [3889]:

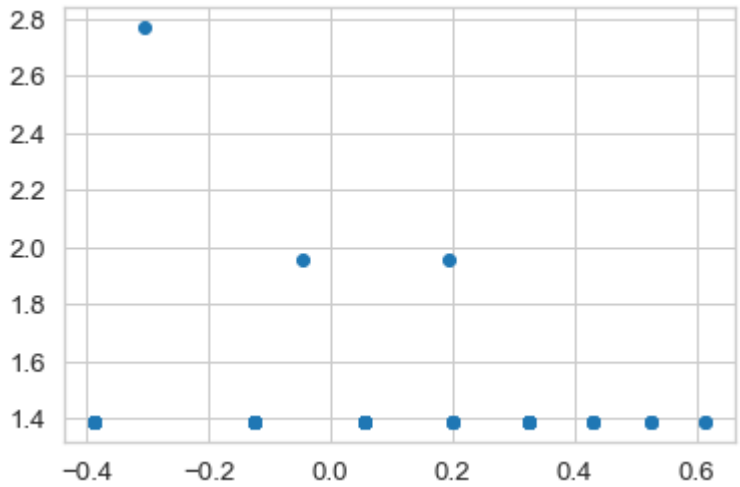
```
# Testing for Multicollinearity:
dfUniqueSongsMaxInstanceCleanedFromOutliers.corr()
```

Out[3889]:

	maxinstance	weeks_on_chart	weeks_on_chartCBRT	maxinstanceLOG	ma
maxinstance	1.000000	0.653350	0.472139	0.984228	
weeks_on_chart	0.653350	1.000000	0.961911	0.658188	
weeks_on_chartCBRT	0.472139	0.961911	1.000000	0.487389	
maxinstanceLOG	0.984228	0.658188	0.487389	1.000000	
maxinstanceSQRT	0.995889	0.658418	0.481820	0.996206	

In [3890]:

```
#re-create model:
#Testing for Homoscedasticity
x = dfUniqueSongsMaxInstanceCleanedFromOutliers['maxinstanceSQRT']
y = dfUniqueSongsMaxInstanceCleanedFromOutliers['weeks_on_chartCBRT']
model = sm.OLS(y,x).fit()
pred_val = model.fittedvalues.copy()
true_val = dfUniqueSongsMaxInstanceCleanedFromOutliers['weeks_on_chartCBRT'].val
residual = true_val - pred_val
fig, ax = plt.subplots(figsize=(6, 4))
_ = ax.scatter(residual, pred_val)
```



In [3891]:

```
sms.diagnostic.het_breuschpagan(residual, dfUniqueSongsMaxInstanceCleanedFromOut
# --> p-Value = 1.125e-11, still < 0.05, assumption of Homoscedasticity not met
```

Out[3891]:

```
(31.48240498463831, nan, 63.99384157466913, 4.9076794700324825e-11)
```

In [3892]:

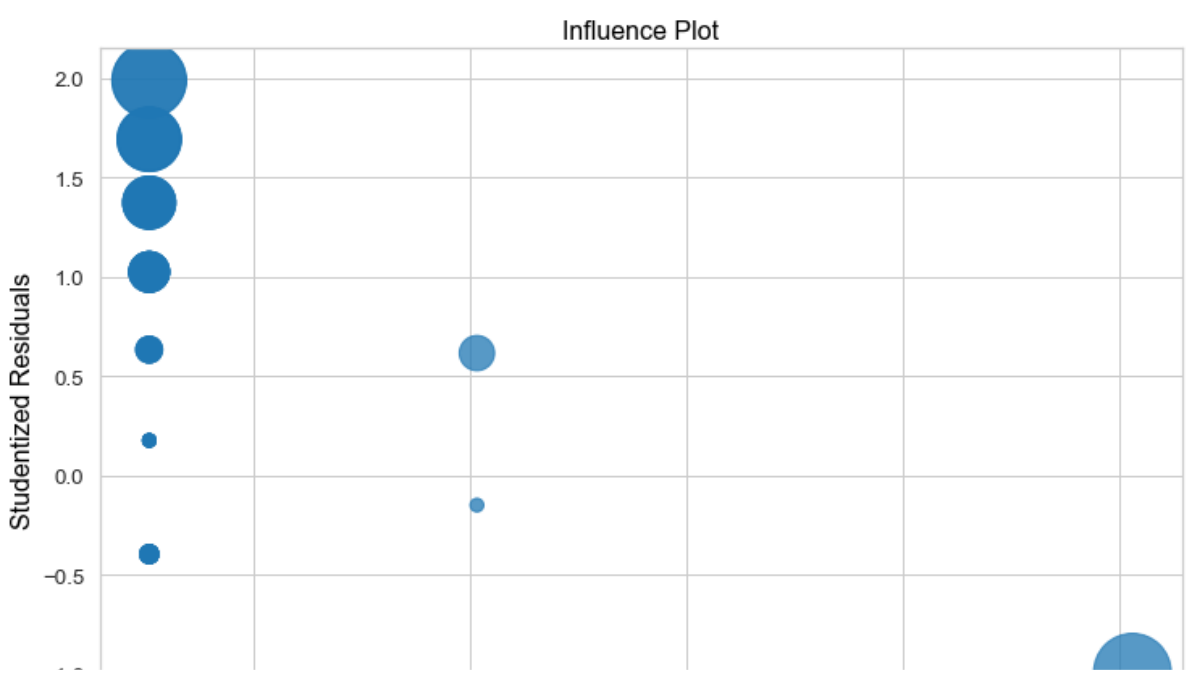
```
# Testing for Multicollinearity:
dfUniqueSongsMaxInstanceCleanedFromOutliers.corr()
```

Out[3892]:

	maxinstance	weeks_on_chart	weeks_on_chartCBRT	maxinstanceLOG	ma
maxinstance	1.000000	0.653350	0.472139	0.984228	
weeks_on_chart	0.653350	1.000000	0.961911	0.658188	
weeks_on_chartCBRT	0.472139	0.961911	1.000000	0.487389	
maxinstanceLOG	0.984228	0.658188	0.487389	1.000000	
maxinstanceSQRT	0.995889	0.658418	0.481820	0.996206	

In [3893]:

```
#Screening for Outliers
fig, ax = plt.subplots(figsize=(12,8))
fig = sm.graphics.influence_plot(model, alpha = .05, ax = ax, criterion="cooks")
```



In [3894]:

```
infl = model.get_influence()
infl.summary_frame()[:5]
```

Out[3894]:

	dfb_maxinstanceSQRT	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid	
5	-0.251524	0.063285	-0.990412	0.060606	-0.251565	-0.990252	-0.2
7	0.108966	0.011998	0.619616	0.030303	0.109534	0.616407	0.1
8	-0.026357	0.000706	-0.150326	0.030303	-0.026574	-0.149096	-0.0
12	-0.152090	0.022939	-1.221075	0.015152	-0.151456	-1.226188	-0.1
13	0.209953	0.042751	1.666986	0.015152	0.206764	1.692699	0.2

In [3895]:

```
infl.summary_frame()['dfb_maxinstanceSQRT'].sort_values(ascending=False)[:5]
# --> no problem here

```

Out[3895]:

```
36    0.246798
44    0.246798
13    0.209953
45    0.209953
35    0.209953
Name: dfb_maxinstanceSQRT, dtype: float64
```

In [3896]:

```
infl.summary_frame()['dffits'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3896]:

```
36    0.246798
44    0.246798
13    0.209953
45    0.209953
35    0.209953
Name: dffits, dtype: float64
```

In [3897]:

```
infl.summary_frame()['hat_diag'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3897]:

```
5     0.060606
8     0.030303
7     0.030303
76    0.015152
74    0.015152
Name: hat_diag, dtype: float64
```

In [3898]:

```
infl.summary_frame()['student_resid'].sort_values(ascending=False)[:5]
# --> no problem here
```

Out[3898]:

```
44    1.989751
36    1.989751
45    1.692699
13    1.692699
35    1.692699
Name: student_resid, dtype: float64
```

In [3899]:

```
1 model.summary()
```

Out[3899]:

OLS Regression Results

Dep. Variable:	weeks_on_chartCBRT	**R-squared (uncentered):**	0.954
Model:	OLS	**Adj. R-squared (uncentered):**	0.953
Method:	Least Squares	**F-statistic:**	1250.
Date:	Fri, 31 Dec 2021	**Prob (F-statistic):**	7.10e-42
Time:	18:53:32	**Log-Likelihood:**	-16.264
No. Observations:	61	**AIC:**	34.53
Df Residuals:	60	**BIC:**	36.64
Df Model:	1		
Covariance Type:	nonrobust		

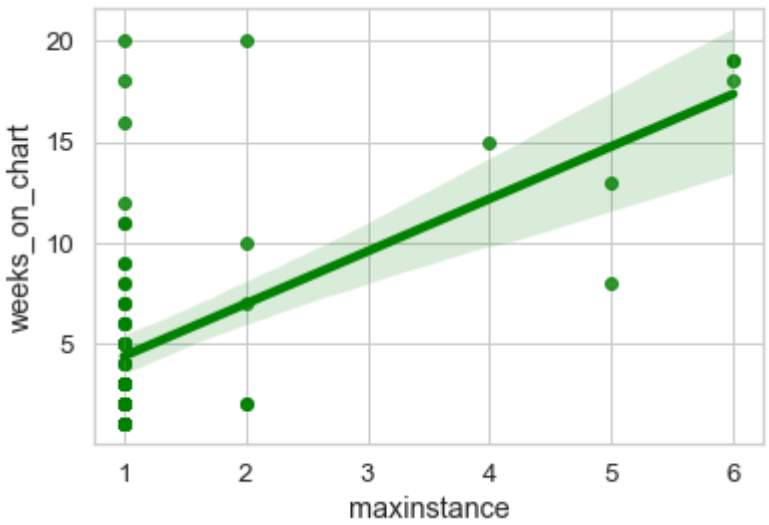
Interpretation:

- **The (max) number of instances explains 95.4% of the variability in number of weeks on chart (R-squared value).**
- **Reject H0: p-value = 7.10e-42; The (max) number of instances significantly influence the number of weeks on chart.**

Create a Scatterplot of days to Christmas versus weeks on chart:

In [3900]:

```
1 sns.regplot(y="weeks_on_chart", x="maxinstance", data=dfUniqueSongsMaxInstance,c
```



Get some statistics on song counts per season

In [3901]:

```
sqlSongsPerSeason = sqldf("""select count(*) as seasonCount, sum(weeksOnChartPer
                             from dfXmasSongsRaw
                             group by season
                             order by season;""")
sqlSongsPerSeason
```

Out[3901]:

	seasonCount	seasonSum	season
0	11	33	1958
1	17	87	1959
2	26	72	1960
3	43	253	1961
4	29	109	1962
5	16	130	1963
6	11	121	1964
7	2	4	1966
8	6	36	1968
9	9	81	1969
10	15	125	1973
11	9	53	1974
12	7	25	1975
13	8	64	1978
14	17	109	1980
15	10	52	1984
16	12	144	1989
17	7	49	1993
18	1	1	1994
19	4	16	1996
20	6	26	1997
21	12	44	1999
22	3	5	2000
23	18	194	2005
24	8	18	2006
25	9	41	2007
26	2	4	2009
27	8	16	2010
28	14	104	2011
29	3	9	2012
30	9	21	2013

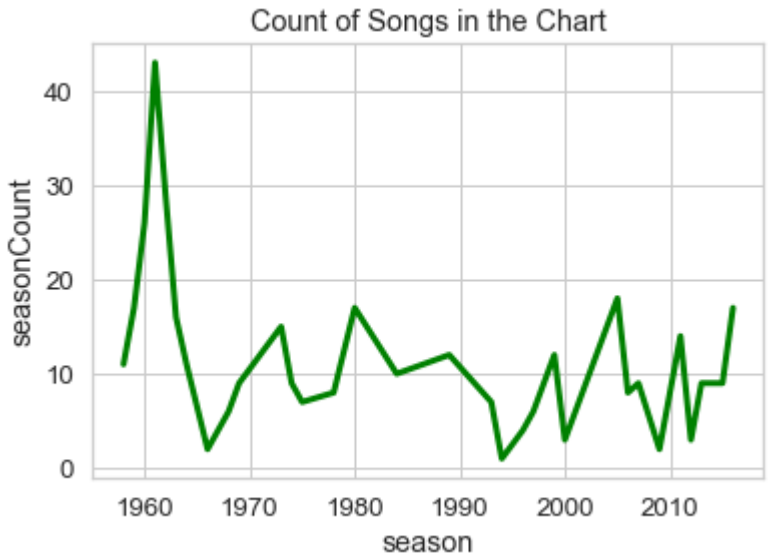
	seasonCount	seasonSum	season
31	9	35	2014
32	9	27	2015
33	17	51	2016

In [3902]:

```
#Plot the result: Counts of songs per season
sns.set_style("whitegrid")
sns.lineplot(data=sqlSongsPerSeason, x="season", y="seasonCount",color='g').set(
```

Out[3902]:

```
[Text(0.5, 1.0, 'Count of Songs in the Chart')]
```



In [3903]:

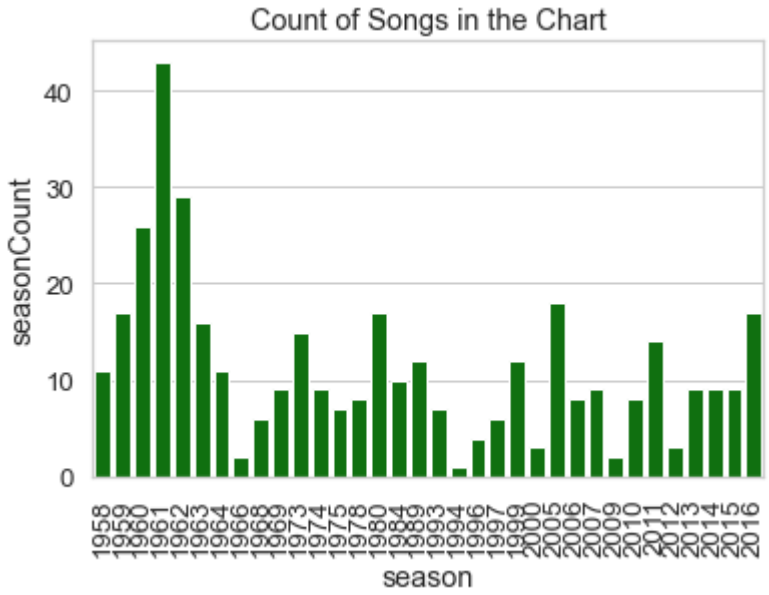
```
sns.barplot(data=sqlSongsPerSeason, x="season", y="seasonCount",color='g').set(t
plt.xticks(rotation = 'vertical')
plt.show()

```

Out[3903]:

```
[Text(0.5, 1.0, 'Count of Songs in the Chart')]
```

Out[3903]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 1
5, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33]),
 [Text(0, 0, '1958'),
  Text(1, 0, '1959'),
  Text(2, 0, '1960'),
  Text(3, 0, '1961'),
  Text(4, 0, '1962'),
  Text(5, 0, '1963'),
  Text(6, 0, '1964'),
  Text(7, 0, '1966'),
  Text(8, 0, '1968'),
  Text(9, 0, '1969'),
  Text(10, 0, '1973'),
  Text(11, 0, '1974'),
  Text(12, 0, '1975'),
  Text(13, 0, '1978'),
  Text(14, 0, '1980'),
  Text(15, 0, '1984'),
  Text(16, 0, '1989'),
  Text(17, 0, '1993'),
  Text(18, 0, '1994'),
  Text(19, 0, '1996'),
  Text(20, 0, '1997'),
  Text(21, 0, '1999'),
  Text(22, 0, '2000'),
  Text(23, 0, '2005'),
  Text(24, 0, '2006'),
  Text(25, 0, '2007'),
  Text(26, 0, '2009'),
  Text(27, 0, '2010'),
  Text(28, 0, '2011'),
  Text(29, 0, '2012'),
  Text(30, 0, '2013'),
  Text(31, 0, '2014'),
  Text(32, 0, '2015'),
  Text(33, 0, '2016')])
```

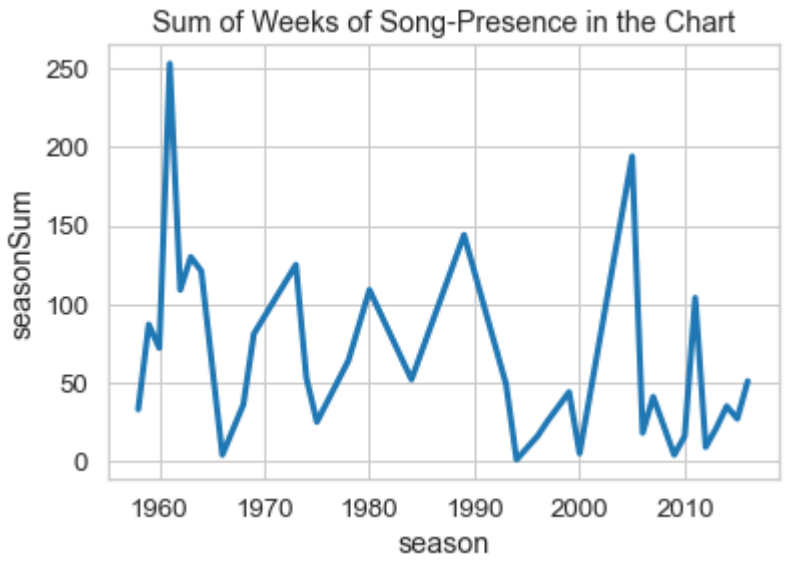


In [3904]:

```
#Plot the sum of song-presence per season
sns.lineplot(data=sqlSongsPerSeason, x="season", y="seasonSum",palette='pastel')

```

Out[3904]:

```
[Text(0.5, 1.0, 'Sum of Weeks of Song-Presence in the Chart')]
```



In [3905]:

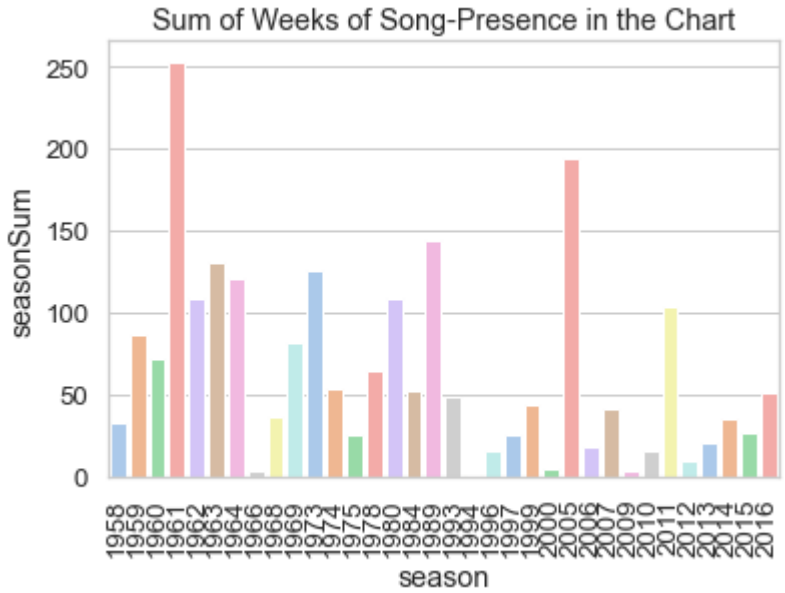
```
1
2 sns.barplot(data=sqlSongsPerSeason, x="season", y="seasonSum",palette='pastel').
3 plt.xticks(rotation = 'vertical')
4 plt.show()
5
```

Out[3905]:

```
[Text(0.5, 1.0, 'Sum of Weeks of Song-Presence in the Chart')]
```

Out[3905]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 1
5, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33]),
 [Text(0, 0, '1958'),
  Text(1, 0, '1959'),
  Text(2, 0, '1960'),
  Text(3, 0, '1961'),
  Text(4, 0, '1962'),
  Text(5, 0, '1963'),
  Text(6, 0, '1964'),
  Text(7, 0, '1966'),
  Text(8, 0, '1968'),
  Text(9, 0, '1969'),
  Text(10, 0, '1973'),
  Text(11, 0, '1974'),
  Text(12, 0, '1975'),
  Text(13, 0, '1978'),
  Text(14, 0, '1980'),
  Text(15, 0, '1984'),
  Text(16, 0, '1989'),
  Text(17, 0, '1993'),
  Text(18, 0, '1994'),
  Text(19, 0, '1996'),
  Text(20, 0, '1997'),
  Text(21, 0, '1999'),
  Text(22, 0, '2000'),
  Text(23, 0, '2005'),
  Text(24, 0, '2006'),
  Text(25, 0, '2007'),
  Text(26, 0, '2009'),
  Text(27, 0, '2010'),
  Text(28, 0, '2011'),
  Text(29, 0, '2012'),
  Text(30, 0, '2013'),
  Text(31, 0, '2014'),
  Text(32, 0, '2015'),
  Text(33, 0, '2016')])
```


Sum of Weeks of Song-Presence in the Chart
seasonSum
0
50
100
150
200
250
1958
1959
1960
1961
1962
1963
1964
1966
1968
1969
1973
1974
1975
1978
1980
1984
1989
1993
1994
1996
1997
1999
2000
2005
2006
2007
2009
2010
2011
2012
2013
2014
2015
2016
season

In [3906]:

```
fig, axes = plt.subplots(2, 2)
fig.suptitle('Season Statistics of Presence in the Chart')
axes[0,0].set_title("Sum of weeks")
axes[0,1].set_title("Sum of weeks")
axes[1,0].set_title("Count of titles")
axes[1,1].set_title("Count of titles")

sns.violinplot(ax=axes[0,0],y="seasonSum",data=sqlSongsPerSeason, palette="paste
sns.boxplot(ax=axes[0,1], y=sqlSongsPerSeason["seasonSum"], palette="pastel" )
sns.violinplot(ax=axes[1,0],y="seasonCount",data=sqlSongsPerSeason, palette="col
sns.boxplot(ax=axes[1,1], y=sqlSongsPerSeason["seasonCount"], palette="colorblir


```

Out[3906]:

```
Text(0.5, 0.98, 'Season Statistics of Presence in the Chart')
```

Out[3906]:

```
Text(0.5, 1.0, 'Sum of weeks')
```

Out[3906]:

```
Text(0.5, 1.0, 'Sum of weeks')
```

Out[3906]:

```
Text(0.5, 1.0, 'Count of titles')
```

Out[3906]:

```
Text(0.5, 1.0, 'Count of titles')
```

Out[3906]:

```
<AxesSubplot:title={'center':'Sum of weeks'}, ylabel='seasonSum'>
```

Out[3906]:

```
<AxesSubplot:title={'center':'Sum of weeks'}, ylabel='seasonSum'>
```

Out[3906]:

```
<AxesSubplot:title={'center':'Count of titles'}, ylabel='seasonCount'>
```

Out[3906]:

```
<AxesSubplot:title={'center':'Count of titles'}, ylabel='seasonCount'>
```

Do the same analysis for decades:

In [3907]:

```
sqlSongsPerDecade = sqldf("""select count(*) as decadeCount, sum(weeksOnChartPer
                             from dfXmasSongsRaw
                             group by decade
                             order by decade;""")
sqlSongsPerDecade
```

Out[3907]:

	decadeCount	decadeSum	decade
0	21	83	21400
1	144	798	21500
2	44	312	21600
3	35	257	21700
4	24	148	21800
5	50	298	21900
6	69	263	22000

In [3908]:

```
#Plot the result: Counts of songs per season
sns.set_style("whitegrid")

```

In [3909]:

```
fig, axes = plt.subplots(2, 2)
fig.set_size_inches(15, 10, forward=True)
fig.suptitle('Decade Statistics of Presence in the Chart',fontweight='bold',font
fig.tight_layout(h_pad=7, w_pad=3)
axes[0,0].set_title("Sum of Decades")
axes[0,1].set_title("Sum of Decades")
axes[1,0].set_title("Count of titles")
axes[1,1].set_title("Count of titles")


sns.lineplot(ax=axes[0,0],linewidth=8,data=sqlSongsPerDecade, x="decade", y="dec
sns.barplot(ax=axes[0,1],data=sqlSongsPerDecade, x="decade", y="decadeCount",pal
sns.lineplot(ax=axes[1,0],linewidth=8,data=sqlSongsPerDecade, x="decade", y="dec
sns.barplot(ax=axes[1,1],data=sqlSongsPerDecade, x="decade", y="decadeSum",palet

axes[0][1].tick_params(axis='x', rotation=90)
axes[1][1].tick_params(axis='x', rotation=90)

```

Out[3909]:

```
Text(0.5, 1.01, 'Decade Statistics of Presence in the Chart')
```

Out[3909]:

```
Text(0.5, 1.0, 'Sum of Decades')
```

Out[3909]:

```
Text(0.5, 1.0, 'Sum of Decades')
```

Out[3909]:

```
Text(0.5, 1.0, 'Count of titles')
```

Out[3909]:

```
Text(0.5, 1.0, 'Count of titles')
```

Out[3909]:

```
[Text(0.5, 1.0, 'Count of Songs in the Chart')]
```

Out[3909]:

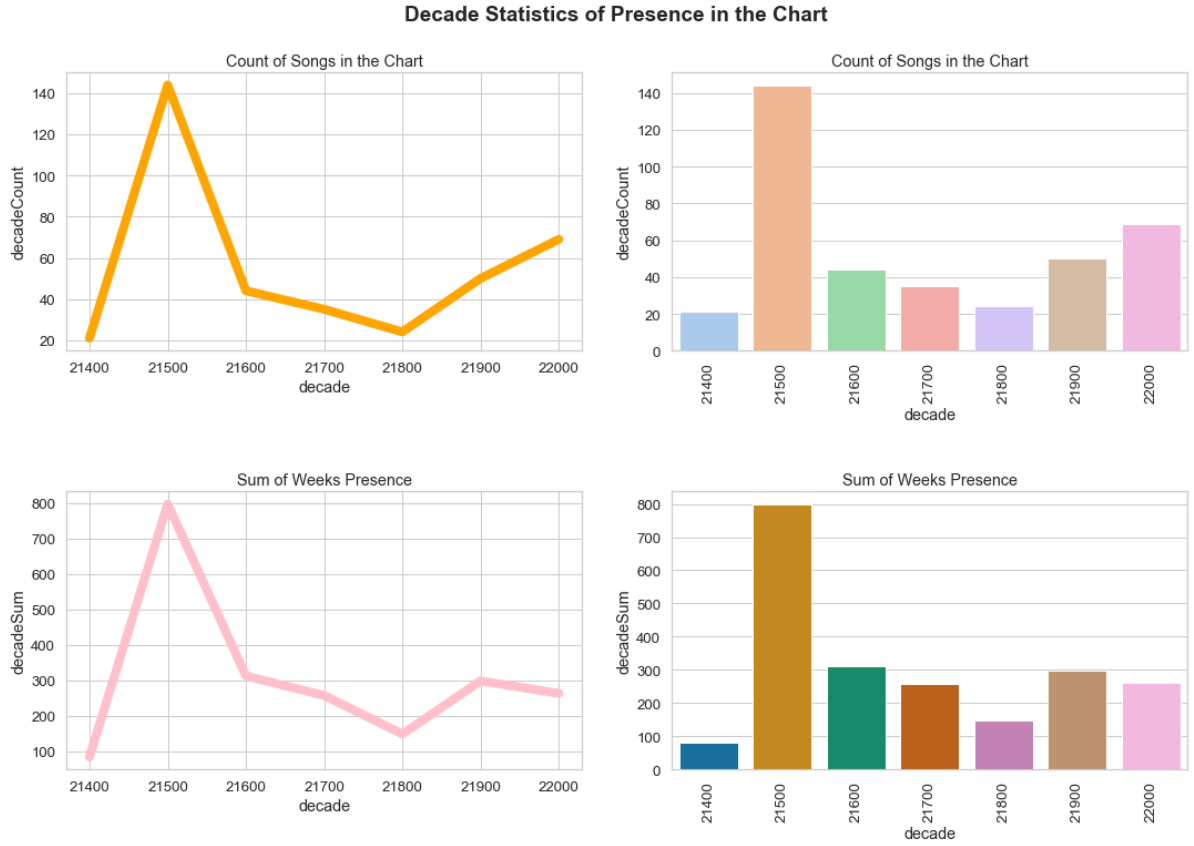
```
[Text(0.5, 1.0, 'Count of Songs in the Chart')]
```

Out[3909]:

```
[Text(0.5, 1.0, 'Sum of Weeks Presence')]
```

Out[3909]:

```
[Text(0.5, 1.0, 'Sum of Weeks Presence')]
```



It seems like Christmas-Songs were the most popular in the 1960ies while the least in 1990ies with an increasing trend in the last decade (only in terms of number of songs but not of presence in the chart!).

In [3910]:

```
1 dfXmasSongsRaw[:1]
```

Out[3910]:

	url	weekid	week_position	song	performer	performerGroup
0	http://www.billboard.com/charts/hot-100/2000-0...	2000-01-01 00:00:00	53	THIS GIFT	98 Degrees	1

1 rows × 28 columns

In [3911]:

```
dfXmasSongsRaw.info()
```

```
 9   peak_position              387 non-null    int64
 10  weeks_on_chart             387 non-null    int64
 11  year                       387 non-null    int64
 12  month                      387 non-null    int64
 13  day                        387 non-null    int64
 14  date                       387 non-null    datetime64[ns]
 15  season                     387 non-null    int64
 16  positionChangeWithinWeek   279 non-null    float64
 17  xmasDate                   387 non-null    datetime64[ns]
 18  numberOfDaysToXmas         387 non-null    int64
 19  initialStartPosition       387 non-null    int64
 20  BeforeXmas                 387 non-null    int64
 21  xmasInSong                 387 non-null    bool
 22  decade                     387 non-null    int64
 23  weekPosCat                 387 non-null    int64
 24  weekPosCat50               387 non-null    int64
 25  match                      387 non-null    object
 26  matchMe                    387 non-null    object
 27  weeksOnChartPerSeason      387 non-null    int64
```

Machine Learning Model to predict the duration on chart

Run a machine learning model to investigate whether it is appropriate to predict weeks on chart per season and check which factor has how much influence. Because there are 13 different classes of week-duration (1-13) and too few datasets reclassify the duration as:

- presence of 1 week only
- presence of 1 month (2-4 weeks)
- presence of more than a month (5 weeks or more)

In [3913]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc

#just for fun: add another variable: length of songname
dfXmasSongsRaw['songnamelength'] = dfXmasSongsRaw['song'].str.len()
dfXmasSongsRaw['xmasInSongR'] = np.where(dfXmasSongsRaw['xmasInSong'] == True,1,
dfXmasSongsRaw['positionChangeWithinWeek'] = dfXmasSongsRaw['positionChangeWithi

#Reclassification of duration
dfXmasSongsRaw['weeksOnChartPerSeasonR'] = np.where(dfXmasSongsRaw['weeksOnChart
dfXmasSongsRaw['weeksOnChartPerSeasonR'] = np.where(dfXmasSongsRaw['weeksOnChart
dfXmasSongsRaw['weeksOnChartPerSeasonR'].value_counts()

```

Out[3913]:

```
2    222
1    138
0     27
Name: weeksOnChartPerSeasonR, dtype: int64
```

In [3914]:

```
dfXmasSongsRaw[:3]
```

Out[3914]:

	url	weekid	week_position	song	performer	performerGroup
0	http://www.billboard.com/charts/hot-100/2000-0...	2000-01-01 00:00:00	53	THIS GIFT	98 Degrees	1
1	http://www.billboard.com/charts/hot-100/2000-0...	2000-08-01 00:00:00	49	THIS GIFT	98 Degrees	1
2	http://www.billboard.com/charts/hot-100/2000-0...	1/15/2000	79	THIS GIFT	98 Degrees	1

3 rows × 31 columns

In [3915]:

```

ecade','previous_week_position','weekPosCat','weekPosCat50','match','matchMe','weeksC

```

In [3916]:

```
y.unique()
```

Out[3916]:

```
array([1, 0, 2])
```

In [3917]:

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 387 entries, 0 to 386
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   week_position             387 non-null    int64  
 1   performerGroup            387 non-null    int64  
 2   year                      387 non-null    int64  
 3   month                     387 non-null    int64  
 4   day                       387 non-null    int64  
 5   season                    387 non-null    int64  
 6   positionChangeWithinWeek  387 non-null    float64
 7   numberOfDaysToXmas        387 non-null    int64  
 8   initialStartPosition      387 non-null    int64  
 9   BeforeXmas                387 non-null    int64  
 10  songnamelength            387 non-null    int64  
 11  xmasInSongR               387 non-null    int64  
dtypes: float64(1), int64(11)
memory usage: 47.4 KB
```

In [3918]:

```
#Train Test Split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.33, random_
#Initial Random Forest Model
forest = RandomForestClassifier(n_estimators=500, random_state=76)
forest.fit(x_train, y_train)
```

Out[3918]:

```
RandomForestClassifier(n_estimators=500, random_state=76)
```

In [3919]:

```
#Evaluate Model Fit
forestPredictions = forest.predict(x_test)
print(confusion_matrix(y_test, forestPredictions))
print(classification_report(y_test, forestPredictions))
```

```
[[ 5  7  2]
 [ 0 38 10]
 [ 0  8 58]]
              precision    recall  f1-score   support

           0       1.00      0.36      0.53        14
           1       0.72      0.79      0.75        48
           2       0.83      0.88      0.85        66

    accuracy                           0.79       128
   macro avg       0.85      0.68      0.71       128
weighted avg       0.81      0.79      0.78       128
```

Interpretation:

- The model is 81% accurate (weighted avg)
- There is 100% accuracy for songs that are only 1 week present in the charts; for 2-4 weeks 72% and for more 83%
- 5 songs were correctly predicted as 1-week-presence-songs, 7 were erroneously classified as 2-4-week-presence-songs and 2 erroneously as more-than-a-month-songs
- 38 songs were correctly predicted as 2-4-week-presence-songs, 0 were erroneously classified as 1-week-presence-songs and 10 erroneously as more-than-a-month-songs
- 58 songs were correctly predicted as more-than-a-month-songs, 0 were erroneously classified as 1-week-presence-songs and 8 erroneously as 2-4-week-presence-songs

Hyperparameter Tuning

In [3920]:

```
from sklearn.model_selection import RandomizedSearchCV
n_estimators_array = [1, 4, 5, 7, 8, 9, 10, 20, 50, 75, 100, 250, 500]
results = []
bestn = -1
bestEstimator = -1
for n in n_estimators_array:
    forest = RandomForestClassifier(n_estimators=n, random_state=76)
    forest.fit(x_train, y_train)
    result = accuracy_score(y_test, forest.predict(x_test))
    results.append(result)
    if result > bestEstimator:
        bestEstimator = result
        bestn = n
    print(n, ':', result)

print("\nbest result at n={} with estimator={}".format(bestn,bestEstimator))
```

Out[3920]:

```
RandomForestClassifier(n_estimators=1, random_state=76)
```

```
1 : 0.703125
```

Out[3920]:

```
RandomForestClassifier(n_estimators=4, random_state=76)
```

```
4 : 0.734375
```

Out[3920]:

```
RandomForestClassifier(n_estimators=5, random_state=76)
```

```
5 : 0.7734375
```

Out[3920]:

```
RandomForestClassifier(n_estimators=7, random_state=76)
```

```
7 : 0.7890625
```

Out[3920]:

```
RandomForestClassifier(n_estimators=8, random_state=76)
```

```
8 : 0.8046875
```

Out[3920]:

```
RandomForestClassifier(n_estimators=9, random_state=76)
```

```
9 : 0.796875
```

Out[3920]:

```
RandomForestClassifier(n_estimators=10, random_state=76)
```

```
10 : 0.78125
```

Out[3920]:

```
RandomForestClassifier(n_estimators=20, random_state=76)
```

```
20 : 0.78125
```

Out[3920]:

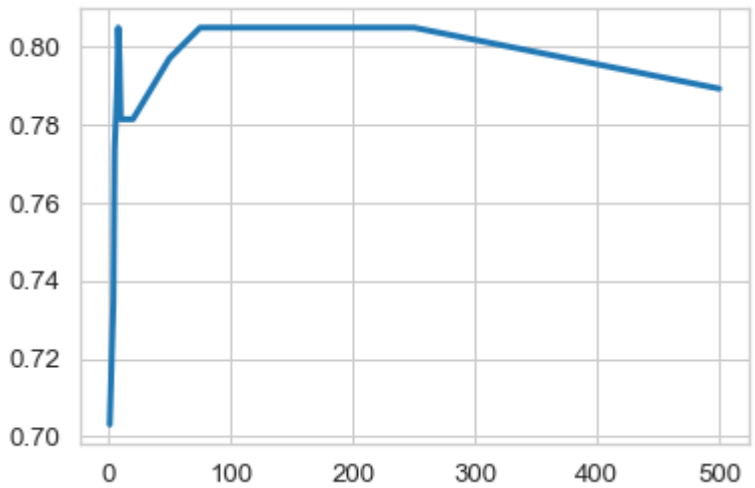
```
Out[3920]:
RandomForestClassifier(n_estimators=50, random_state=76)
50 : 0.796875
Out[3920]:
RandomForestClassifier(n_estimators=75, random_state=76)
75 : 0.8046875
Out[3920]:
RandomForestClassifier(random_state=76)
100 : 0.8046875
Out[3920]:
RandomForestClassifier(n_estimators=250, random_state=76)
250 : 0.8046875
Out[3920]:
RandomForestClassifier(n_estimators=500, random_state=76)
500 : 0.7890625
best result at n=8 with estimator=0.8046875
```

In [3921]:

```
1 plt.plot(n_estimators_array, results)
2
```

Out[3921]:

```
[<matplotlib.lines.Line2D at 0x7fb9ff1bba60>]
```



Tuning the Remaining Tree

In [3922]:

```
# Number of features to consider at every split
max_features = ['auto', None, 'log2']
# Maximum number of levels in tree
max_depth = [10, 20, 30, 40, 50, 60, 70, 80, 90, None]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
random_grid = {'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_leaf': min_samples_leaf}

rf = RandomForestClassifier(n_estimators=bestn)
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_gric
rf_random.fit(x_train, y_train)
minSamplesLeaf = rf_random.best_params_['min_samples_leaf']
maxFeatures = rf_random.best_params_['max_features']
maxDepth = rf_random.best_params_['max_depth']
rf_random.best_params_


```

Out[3922]:

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(n_estimators
=8),
                   n_iter=90,
                   param_distributions={'max_depth': [10, 20, 30, 40,
50, 60,
                                                      70, 80, 90, Non
e],
                                        'max_features': ['auto', None,
'log2'],
                                        'min_samples_leaf': [1, 2,
4]},
                   random_state=42)
```

Out[3922]:

```
{'min_samples_leaf': 2, 'max_features': None, 'max_depth': 60}
```

In [3923]:

```
ndomForestClassifier(n_estimators=bestn, min_samples_leaf=minSamplesLeaf, max_feature
x_train, y_train)
```

Out[3923]:

```
RandomForestClassifier(max_depth=60, max_features=None, min_samples_le
af=2,
                       n_estimators=8)
```

In [3924]:

```
forestPredictions = forest.predict(x_test)
print(confusion_matrix(y_test, forestPredictions))
print(classification_report(y_test, forestPredictions))

```

```
[[ 4  7  3]
 [ 0 40  8]
 [ 0  9 57]]
              precision    recall  f1-score   support

           0       1.00      0.29      0.44        14
           1       0.71      0.83      0.77        48
           2       0.84      0.86      0.85        66

    accuracy                           0.79       128
   macro avg       0.85      0.66      0.69       128
weighted avg       0.81      0.79      0.78       128
```

Although many different parameter changes were tried, the model could not be improved.

Finally look at **feature importance**:

In [3925]:

```
feature_importances = pd.Series(forest.feature_importances_, index=x.columns)
feature_importances.sort_values(inplace=True, ascending=False)
print(feature_importances[:][:])
```

```
season                      0.198436
week_position               0.173208
numberOfDaysToXmas          0.164639
songnamelength              0.156286
positionChangeWithinWeek    0.062544
xmasInSongR                 0.061088
initialStartPosition        0.057681
year                        0.055640
day                         0.035147
performerGroup              0.025133
month                       0.006948
BeforeXmas                  0.003248
dtype: float64
```

Interpretation:

- season seems to be the most influential variable with nearly 20%
- week position, year and number of Days to Christmas are also influential with each more than 10%
- the remaining 8 variables are between nearly 2% and 9%
- whether the song first appeared before or after Christmas seems to have the least impact

In [3926]:

```
feature_importances.plot(kind='bar', figsize=(7,6),color='g')

```

Out[3926]:

<AxesSubplot:>