Case Study 4

MSDS 7333 - 4043 - Quantifying the World

Team: Samuel Arellano, Daniel Clark, Dhyan Shah, Chandler Vaughn

In [ ]:

```
%matplotlib inline

import csv
import os
import requests
import warnings
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pylab as py
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm
from bs4 import BeautifulSoup
from datetime import datetime
from requests import get
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from urllib.request import urlopen

sns.set(color_codes=True)
```

In [ ]:

```
os.getcwd()
```

Out[ ]:

```
'C:\\Users\\gouki\\ML1\\QTW2'
```

Abstract

The goal of this assignment has 3 key components. The first is to research and get the relevant data from a website by using web scraping techniques. The second is to clean, parse and organize the scraped data into a single dataframe. The final step is to perform an exploratory and statistical analysis on the dataset to help answer a question. The dataset and website in question is The Credit Union Cherry Blossom Ten Mile Run and 5K Run-Walk race results for the female runners in the 10-mile races. The website www.cherryblossom.org contains race results from 1999 - 2019, however we will be focusing on the results from 1999 to 2013. Using this specific race and time frame, we are focusing on reviewing whether the age distribution changed among the female race contestants over the years. Using a series of EDA, linear regression, change point analysis, and ANOVA(Analysis of Variance) we found that there was some significant movement in age over the span of the years.

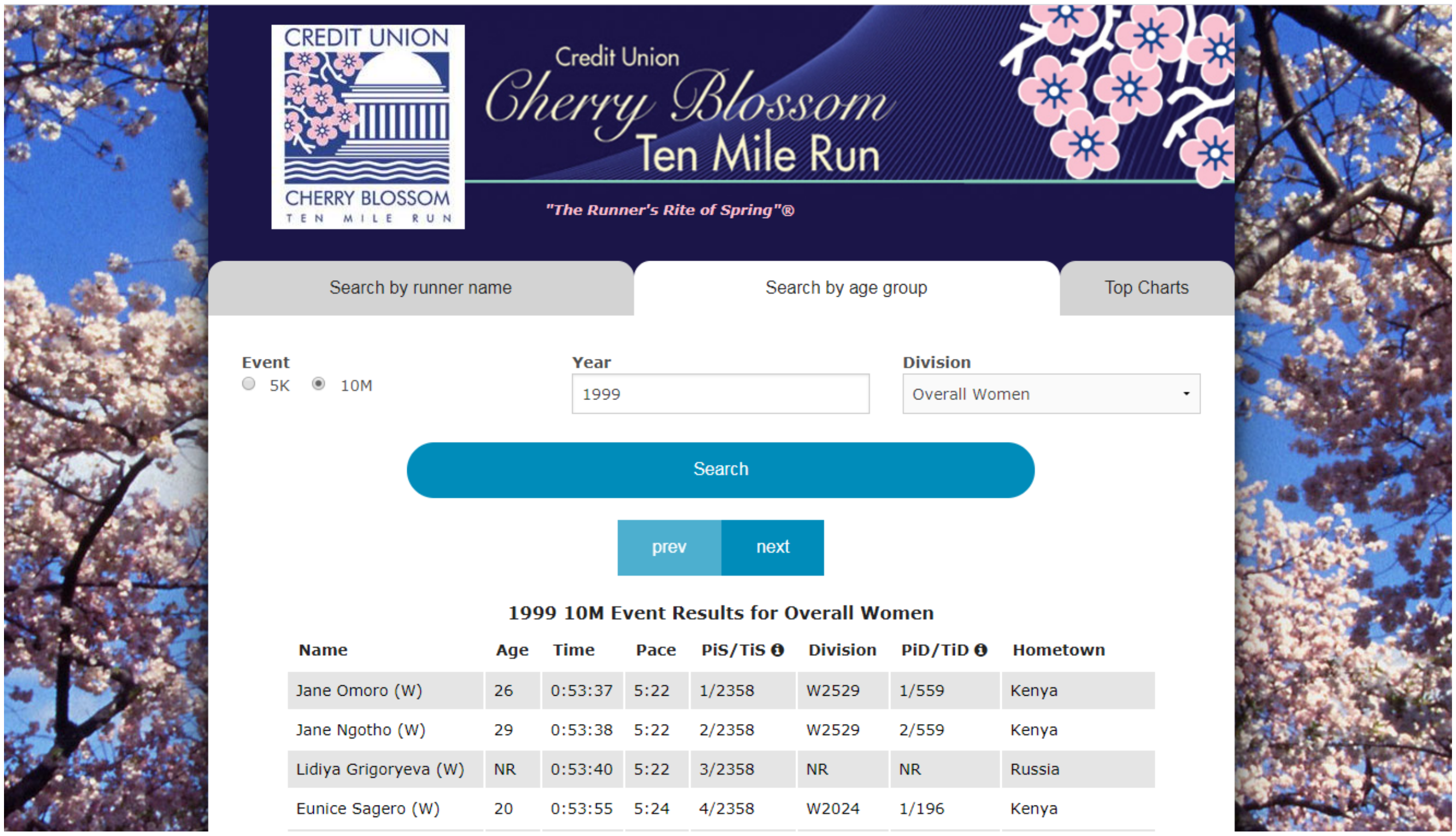
Introduction

The Credit Union Cherry Blossom Ten Mile Run is an annual run held in Washington, D.C. that brings together credit unions from across the country with a shared vision of fundraising and helping children. The proceeds from donations, registration fees, and merchandise sales from the event support Children's Hospitals that belong to the non-profit Children's Miracle Network.

Since the inaugural run in 1973, hundreds of thousands of runners from around the world have participated in the event. Registration of each runner's name, age, hometown, and race times give an enormous trove of information regarding potential trends in the age and performance of runners over the years. The organizations website http://www.cherryblossom.org/ (http://www.cherryblossom.org/) provides records of this data going back to 1999. This analysis will use web scraping to collect data from the website, inspect and clean the data, convert the data into a clean data frame, conduct exploratory data analysis and determine if the age distribution of participating runners has changed significantly over the years.

Data

Data for this analysis was taken from the Credit Union Cherry Blossom Ten Mile Run sites searcheable results web page using the URL http://www.cballtimeresults.org/performances?division=Overall+Women&page=1&section=10M&sex=W&utf8=%E2%9C%93&year=1999 (http://www.cballtimeresults.org/performances?division=Overall+Women&page=1&section=10M&sex=W&utf8=%E2%9C%93&year=1999) as our initial reference for our web scraping function.



Event ○ 5K ● 10M

Year: 1999

Division: Overall Women

1999 10M Event Results for Overall Women

Name	Age	Time	Pace	PiS/TiS	Division	PiD/TiD	Hometown
Jane Omoro (W)	26	0:53:37	5:22	1/2358	W2529	1/559	Kenya
Jane Ngotho (W)	29	0:53:38	5:22	2/2358	W2529	2/559	Kenya
Lidiya Grigoryeva (W)	NR	0:53:40	5:22	3/2358	NR	NR	Russia
Eunice Sagero (W)	20	0:53:55	5:24	4/2358	W2024	1/196	Kenya

Each page yielded only 20 runners and the number of female runners per year varied from 2,166 to 11,042, however from page to page the URL remained consistent making it relatively simple to iteratively cycle through the **page=** and **year=** variables of the URL. After an examination of the page source it was determined that the values: *"Year","Name", "Age", "Time","Pace","PiS/TiS", "Division", "PiD/TiD", and "Hometown"* could be obtained for each runner. A CSV file was created with the appropriate headers to store the scraped data.

In [ ]:

```
#create outputfile
outputFile = "CherryBlossomWomens10M.csv"

#create headers
file = open("CherryBlossomWomens10M.csv", "w",encoding='utf8' )
writer = csv.DictWriter(
    file, fieldnames=["Year","Name", "Age", "Time","Pace","PiS/TiS", "Division",
"PiD/TiD", "Hometown", 'State','Blank1','Blank2',])
writer.writeheader()
file.close()
```

In [ ]:

```
#split the marathon url into two parts to format page number and year
websiteURLpart1 = "http://www.cballtimeresults.org/performances?division=Overall
+Women&page="
websiteURLpart2 = "&section=10M&sex=W&utf8=%E2%9C%93&year="
```

Through an examination of the page source it was determined that 'tr class='print-link-color' indicated entries for a new runner and two functions were created. The first function would update the URL **page=** and **year=** and the second function would parse the URL calling the first function to update the URL when no more 'tr class='print-link-color' entries were found on the page and update the year when the end page was reached.

In [ ]:

```
#create url function to format the url that is then passed to requests to get th
e html page
def createURL(url1, url2, pageNum, year):
    return(url1 + str(pageNum) + url2 + str(year))

#create parsing function that uses createURL to iterate through pages
def getRunners(url1, url2):
    year = 1999
    #iterate through all years from 1999 to 2018
    while(year < 2019):
        pageNum = 1
        morePages = True
        #iterate through all pages in each year
        while(morePages):
            websiteURL = createURL(url1, url2, pageNum, year)
            with open(outputFile, "a", encoding='utf8') as f:
                #requests return the html of the page in a raw object
                page = requests.get(websiteURL)
                #Beautiful Soup parses the requests object into a better formatt
ed html object
                soup = BeautifulSoup(page.content, "html.parser")
                newLine = ""
                contents = soup.find_all("tr", class_="print-link-color")
                #check if the table is empty, if so, go to the next year
                if not contents:
                    morePages = False
                    year += 1
                #if the table has content
                else:
                    #iterate through all of the table rows that have class=print
-link-color
                    for tr in soup.find_all("tr", class_="print-link-color"):
                        #add the previously parsed line to the file
                        if(newLine != ""):
                            f.write(newLine.rstrip(","))
                        newLine = ""
                        f.write("\n")
                        firsta = True
                        for a in tr.select("td a"):
                            if(firsta):
                                f.write(a.getText().split()[0])
                                firsta = False
                            else:
                                f.write(a.getText())
                            f.write(",")
                    #once all of the table rows for this page are parsed, go to
the next page
                    pageNum += 1
                    f.close()
    return(True)
```

In [ ]:

```
#run getRunners function to scrape website for women runners
getRunners(websiteURLpart1, websiteURLpart2)
```

Out[ ]:

```
True
```

Data Preperation

In [ ]:

```
df = pd.read_csv('CherryBlossomWomens10M.csv', header=0)
```

After the data is scraped and appended to the CSV file, a dataframe is created that consists of 13,8265 observations and 12 fields. The dataframe has 2 blank fields that were created so that the CSV would have enough indexes. Those 2 blank fields along with 6 other unnecessary fields are dropped and a new dataframe is created. A review of the dataframes first 5 rows indicates that records with no collected data are marked NR.

In [ ]:

```
df.shape
```

Out[ ]:

```
(138265, 12)
```

```
In [ ]:
```

```
df.head(5)
```

```
Out[ ]:
```

	Year	Name	Age	Time	Pace	PiS/TiS	Division	PiD/TiD	Hometown	State	Blank1
0	1999	Jane Omoro (W)	26	0:53:37	5:22	1/2358	W2529	1/559	Kenya	NaN	NaN
1	1999	Jane Ngotho (W)	29	0:53:38	5:22	2/2358	W2529	2/559	Kenya	NaN	NaN
2	1999	Lidiya Grigoryeva (W)	NR	0:53:40	5:22	3/2358	NR	NR	Russia	NaN	NaN
3	1999	Eunice Sagero (W)	20	0:53:55	5:24	4/2358	W2024	1/196	Kenya	NaN	NaN
4	1999	Alla Zhilyayeva (W)	29	0:54:08	5:25	5/2358	W2529	3/559	Russia	NaN	NaN

```
In [ ]:
```

```
df1 = df.drop(['Pace','PiS/TiS', 'Division', 'PiD/TiD', 'Hometown', 'State','Bla
nk1','Blank2'], axis=1)
```

```
In [ ]:
```

```
df1.shape
```

```
Out[ ]:
```

```
(138265, 4)
```

After our dropping, we are left with just 4 columns.

```
In [ ]:
```

```
df1['Age'].value_counts()['NR']
```

```
Out[ ]:
```

```
20
```

We search for the number of instances of NR and determine that NR is present in 20 rows of the "Age" field. Because there is no way to accurately impute data and because they account for such a small portion of the nearly 14K observations, we decide it is best to drop the 20 records.

In [ ]:

```
df1.head(5)
```

Out[ ]:

	Year	Name	Age	Time
0	1999	Jane Omoro (W)	26	0:53:37
1	1999	Jane Ngotho (W)	29	0:53:38
3	1999	Eunice Sagero (W)	20	0:53:55
4	1999	Alla Zhilyayeva (W)	29	0:54:08
5	1999	Teresa Wanjiku (W)	24	0:54:10

The new dataframe is checked for null values and the datatypes are examined.

In [ ]:

```
print(df1.isnull().sum())
```

```
Year     0
Name     0
Age      0
Time     0
dtype: int64
```

No null values are detected, so now we are free to move forward into adjusting our column data types.

In [ ]:

```
df1.dtypes
```

Out[ ]:

```
Year      int64
Name     object
Age      object
Time     object
dtype: object
```

"Age" is converted to an integer value and "Time" is converted to a float variable that represents the minutes it took each runner to finnish the race.

In [ ]:

```
df1['Age'] = df1['Age'].astype(str).astype(int)
df1['Time'] = df1['Time'].astype(str)
```

In [ ]:

```
#convert string time and pace to float minutes
dftime = df1["Time"].str.split(":", n = 2, expand = True).astype(float)
```

In [ ]:

```
dftime['Time'] = (dftime[0]*60)+dftime[1]+(dftime[2]/60)
df1['Time'] = dftime["Time"]
```

In [ ]:

```
dftime['Time'] = (dftime[0]*60)+dftime[1]+(dftime[2]/60)
df1['Time'] = dftime["Time"]
```

In [ ]:

```
df1.head(5)
```

Out[ ]:

	Year	Name	Age	Time	Pace
0	1999	Jane Omoro (W)	26	53.616667	00:05:22
1	1999	Jane Ngotho (W)	29	53.633333	00:05:22
2	1999	Eunice Sagero (W)	20	53.916667	00:05:24
3	1999	Alla Zhilyayeva (W)	29	54.133333	00:05:25
4	1999	Teresa Wanjiku (W)	24	54.166667	00:05:25

After our conversion, you can see that we are converting our time value to a float that will allow us to more easily aggregate into future EDA and modeling. While our key focus is on age, we are going to also be interested in seeing how time is related to age and how that could possibly change year over year.

In [ ]:

```
df1.to_csv ('CherryBlossomWomens10MTidy.csv', index = False, header=True)
```

Exploratory Data Analysis

Group Age Over Time

In [ ]:

```
#import tidy dataset
df1 = pd.read_csv('CherryBlossomWomens10MTidy.csv', header=0)
```

```
In [ ]:
```

```
df1 = df1[df1.Year < 2014]
df1[["Age"]].describe()
```

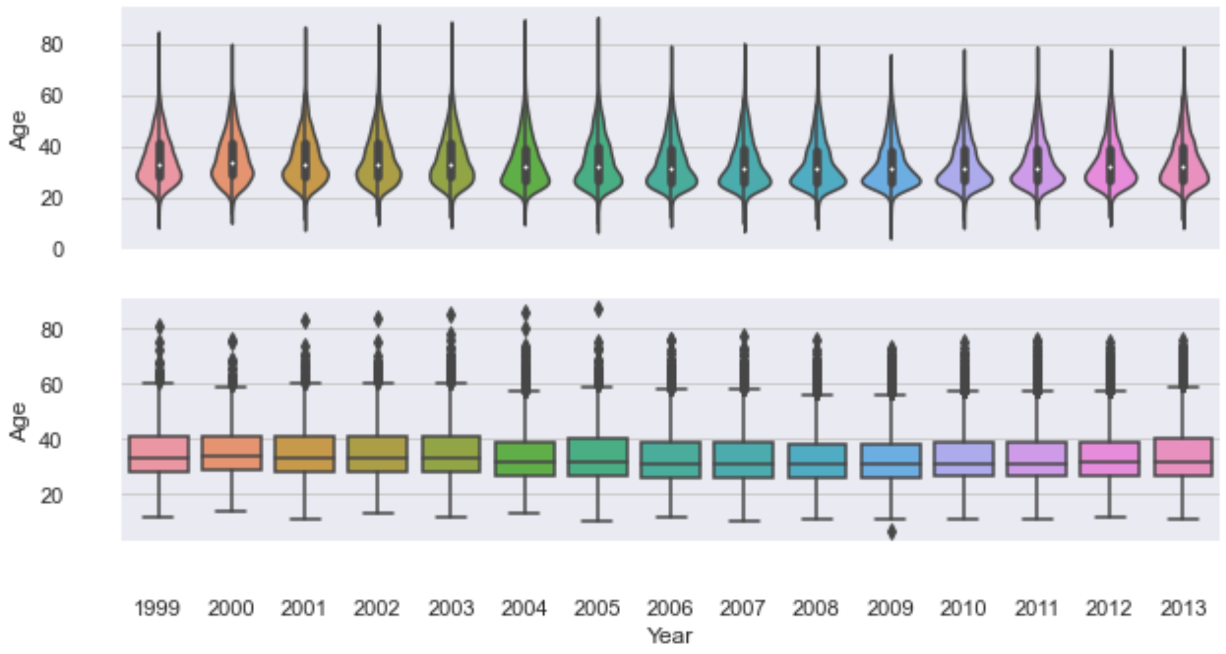
```
Out[ ]:
```

	Age
count	86169.000000
mean	33.919820
std	9.228929
min	7.000000
25%	27.000000
50%	32.000000
75%	39.000000
max	87.000000

Descriptive statistics of the "Age" variable indicate that outliers may be present in the dataset. While the difference between the quantile range from 25% to 75% is only 1.30 standard deviations, the difference between the mean and the minimum values is 2.92 standard deviations and the difference between the mean and the maximum values is 5.75 standard deviations. Besides indicating the potential presence of outliers this also indicates that our distribution may be right skewed.

In [ ]:

```
f, axes = plt.subplots(2, 1, figsize=(10, 5), sharex=True)
sns.set(style="ticks", palette="pastel")
sns.boxplot(x="Year", y="Age", data=df1, ax=axes[1])
sns.violinplot(x="Year", y="Age", data=df1, ax=axes[0])
sns.despine(offset=20, trim=True)
```



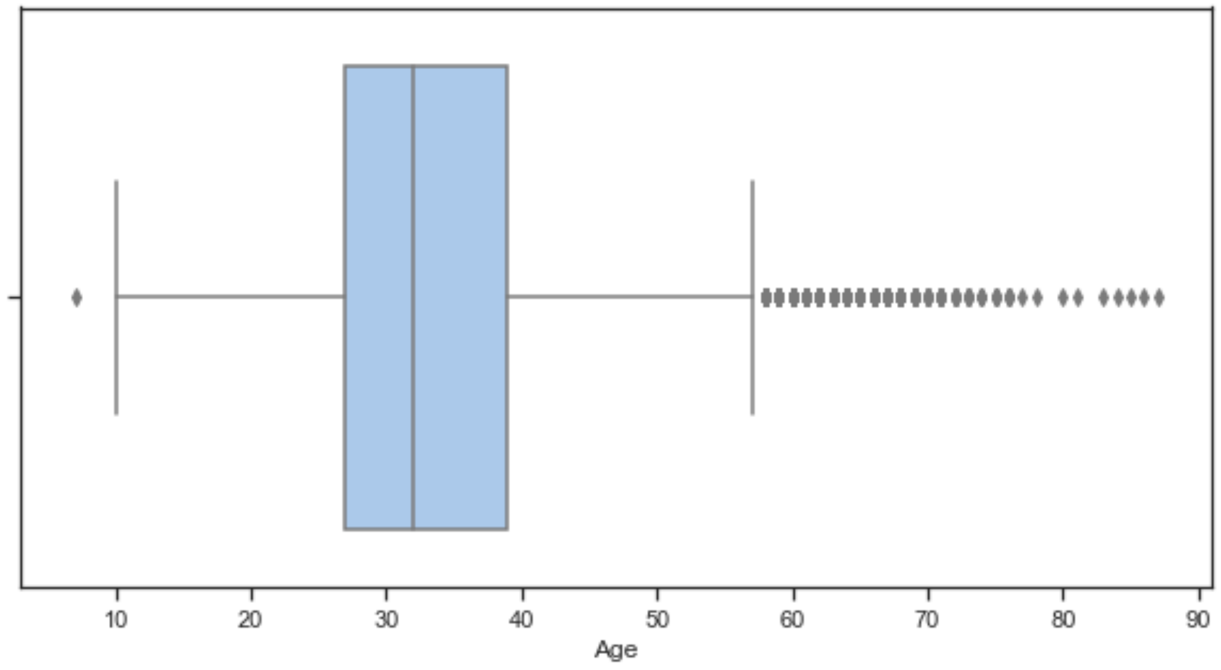
Box plots of the age distributions both by year and in aggregate seem to visually validate the presence of outliers that could possibly be skewing the data. As measures of mean are sensitive to the presence of outliers we must determine if the outliers need to be removed or if we need to use a method that does not rely on mean when testing the null hypothesis that: There is no significant difference in the distribution of ages over time.

In [ ]:

```
plt.figure(figsize=(10,5))
sns.boxplot(x=df1["Age"])
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff2f8de48d0>
```



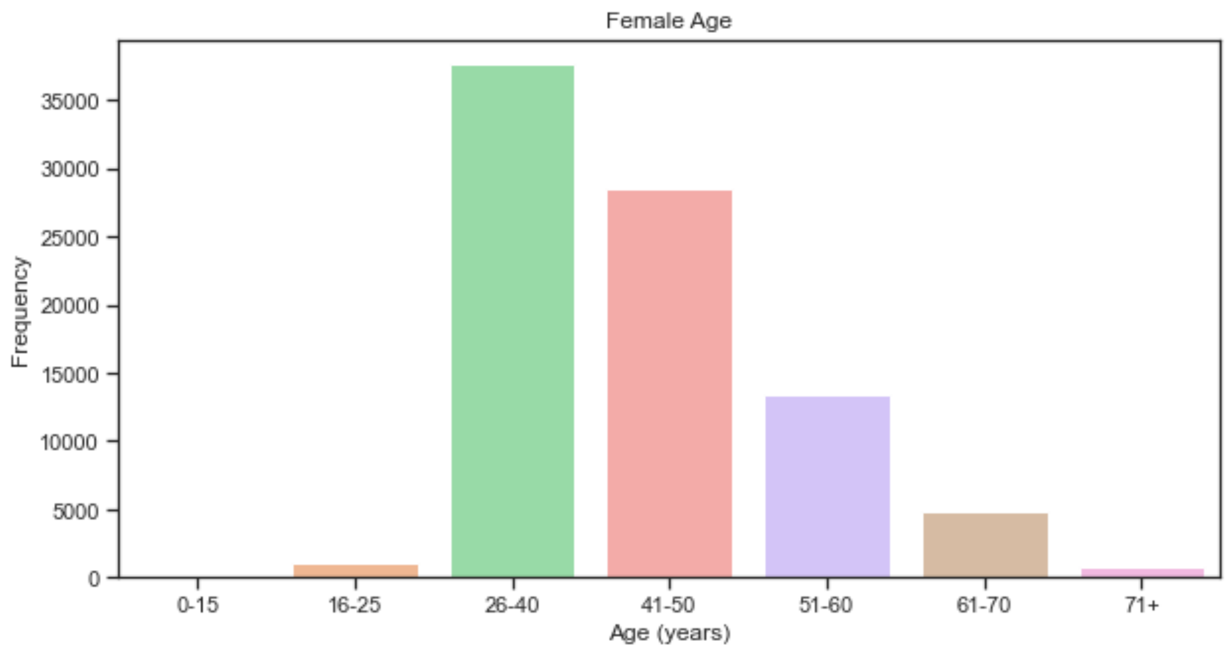
The plot above provides us a distribution of the age groupings across all 13 years. On the high side of the mean, we can see there's a number of participants over the age of 60 which heavily outweighs the number of outliers below the whisker at below 10 years old.

In [ ]:

```
bins = [0, 10, 20, 30, 40, 50, 60, 70]
names = ['0-15', '16-25', '26-40', '41-50','51-60', '61-70','71+']

df1['AgeRange'] = pd.cut(df1['Age'], bins, labels=names)

plt.figure(figsize=(10,5))
#sns.set(style="ticks", palette="pastel")
sns.countplot('AgeRange', data=df1)

plt.title('Female Age'); plt.xlabel('Age (years)'); plt.ylabel('Frequency')
```

Out[ ]:

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



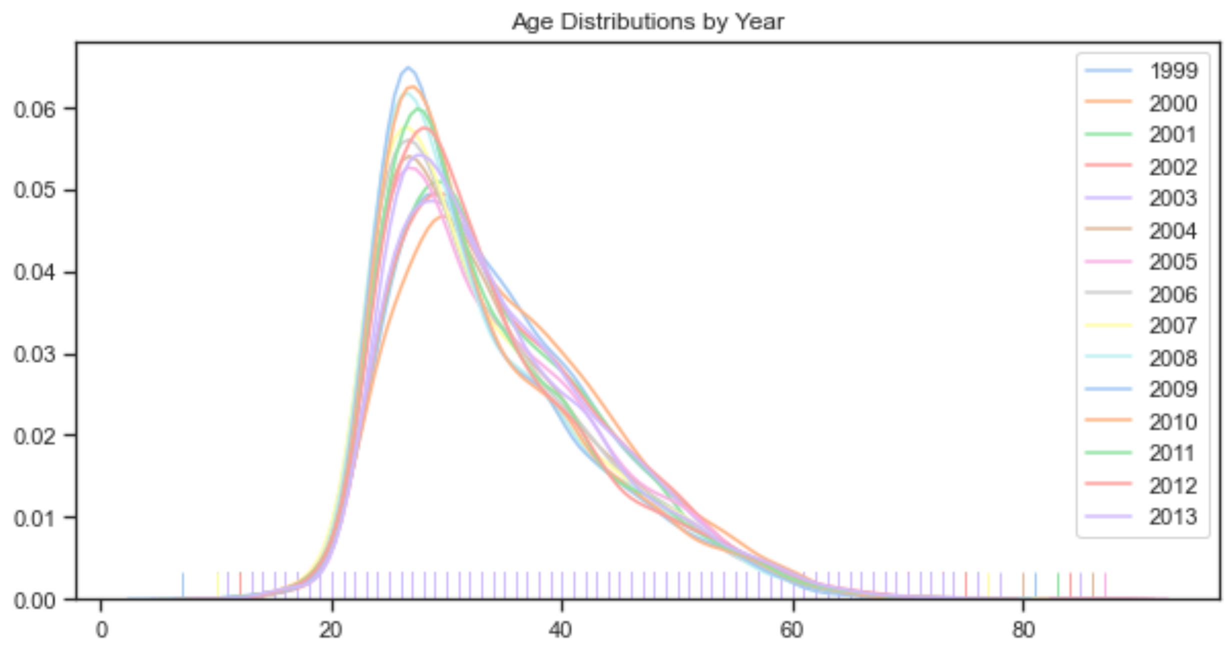
We created a series of bins within our age groupings so that we can more easily visualize our skewness in a histogram. Here, we can see the 26-40 grouping has the most representatives followed by the 41-50 group. As discussed previously, our theme of right skewness is apparent here as well.

In [ ]:

```python
plt.figure(figsize=(10,5))
n = 1999
while n < 2014:
    YEAR = df1.loc[df1['Year'] == n]
    sns.distplot(YEAR[['Age']], hist=False, rug=True, label=n)
    n+=1
plt.title("Age Distributions by Year")
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7ff2da4a9f90>
```



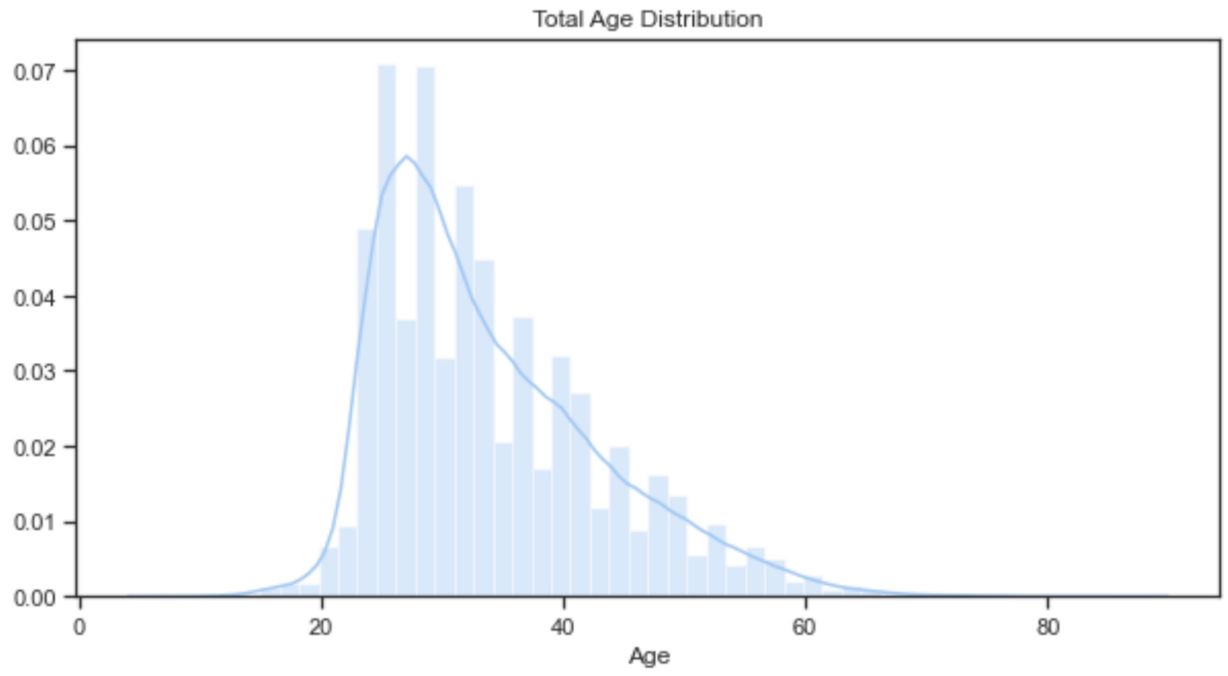
Review of the distribution plots validate that "Age" among female runners is not normally distributed and variations of the mean of the distributions from year to year seems to be more pronounced in the distribution plots. While the distributions indicate that there may be a shift in mean age over time it is difficult to conclusively determine the direction of the shift.

In [ ]:

```
plt.figure(figsize=(10,5))
sns.distplot(df1['Age'])
plt.title("Total Age Distribution")
```

Out[ ]:

```
Text(0.5, 1.0, 'Total Age Distribution')
```



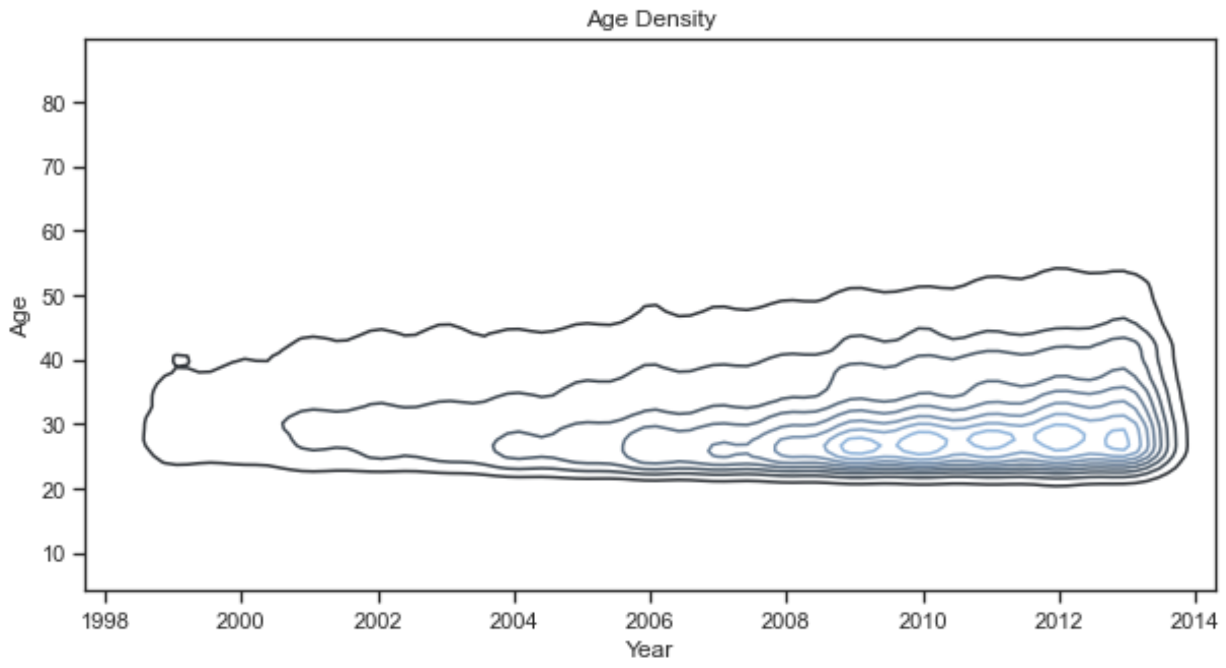
Review of the kernel density plot seems to indicate that there is a change in the distribution of "Age" over time. Additionally the plot seems to indicate a gradual increase in the density of runners under 30 years of age.

In [ ]:

```
f, ax = plt.subplots(figsize=(10, 5))
sns.kdeplot(df1.Year, df1.Age, ax=ax)
plt.title("Age Density")
```

Out[ ]:

```
Text(0.5, 1.0, 'Age Density')
```



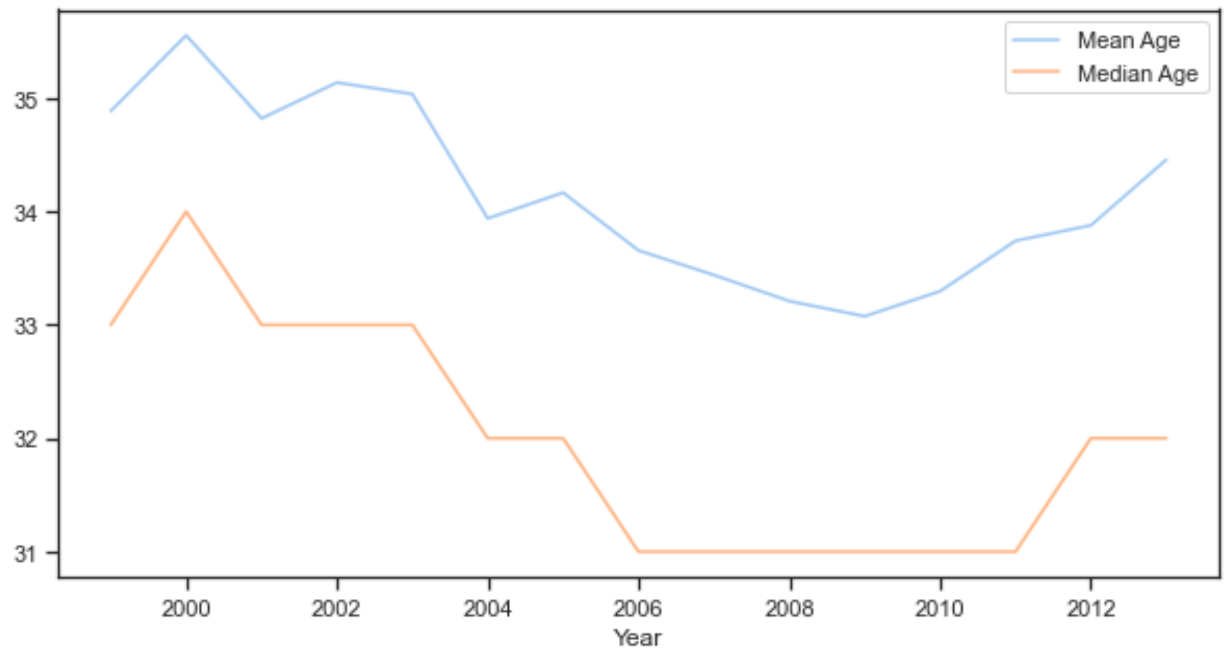
By comparing the mean age by year to the median age by year the effects of the outliers on mean age are more noticeable. When looking at the range of the age axis it appears that the range between the lowest mean and the highest mean is less than 3 years and the range between the lowest median and the highest median is about 3 years. Whether this variation is significant or not will be determined in the statistical testing section of our analysis.

In [ ]:

```
df1.groupby('Year')['Age'].mean().plot(kind='line', figsize=(10,5), label="Mean
Age")
df1.groupby('Year')['Age'].median().plot(kind='line', figsize=(10,5),label="Medi
an Age")
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7ff2e9a98c10>
```



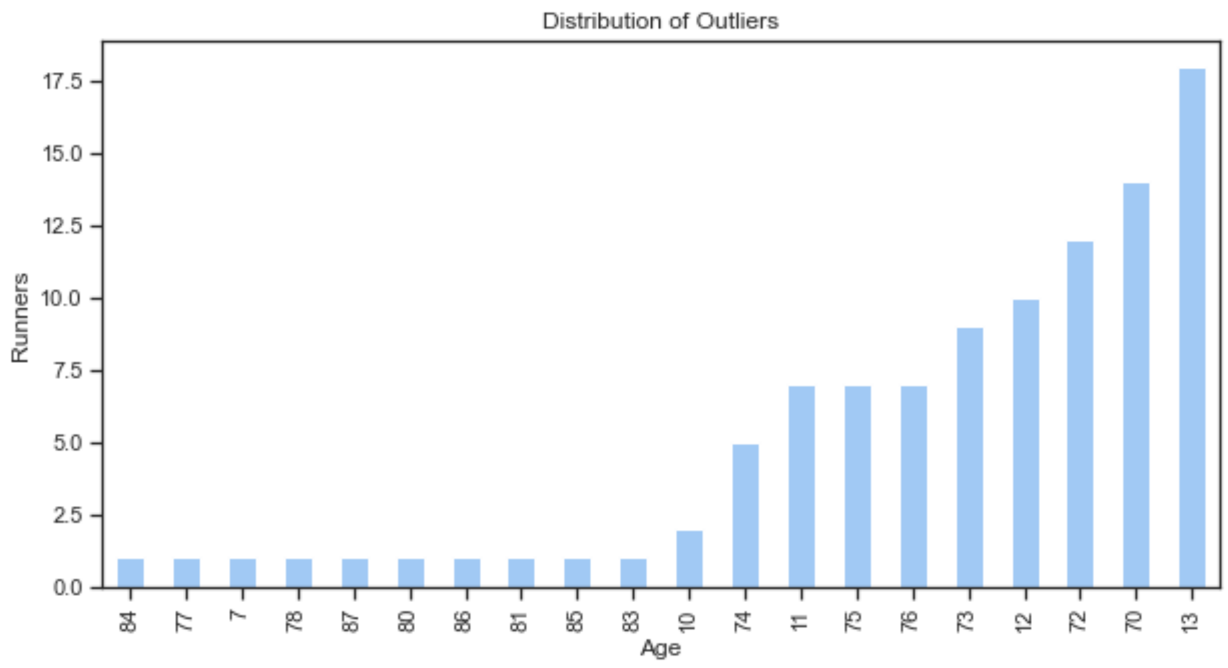
The 20 ages with the lowest incidence rate were counted and plotted. All together they accounted for only 0.12% of the entire dataset. To analyze the impact of removing the outliers from the dataset a new data set was created that omitted all ages greater than 70 or less than 13. This process was repeated with the lowest 25 and lowest 30 leading to new datasets that omitted 0.29% and 0.72% of the observations respectively. We chose this method because it detects anomalous data points on both sides of the mean and determines outliers by their frequency.

In [ ]:

```
df1.Age.value_counts().nsmallest(20).plot(kind='bar', figsize=(10,5))
plt.title("Distribution of Outliers")
plt.ylabel("Runners")
plt.xlabel("Age")
```

Out[ ]:

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



In [ ]:

```
df1.Age.value_counts().nsmallest(20).sum()/df1.Age.count()
```

Out[ ]:

```
0.0011721152618691176
```

In [ ]:

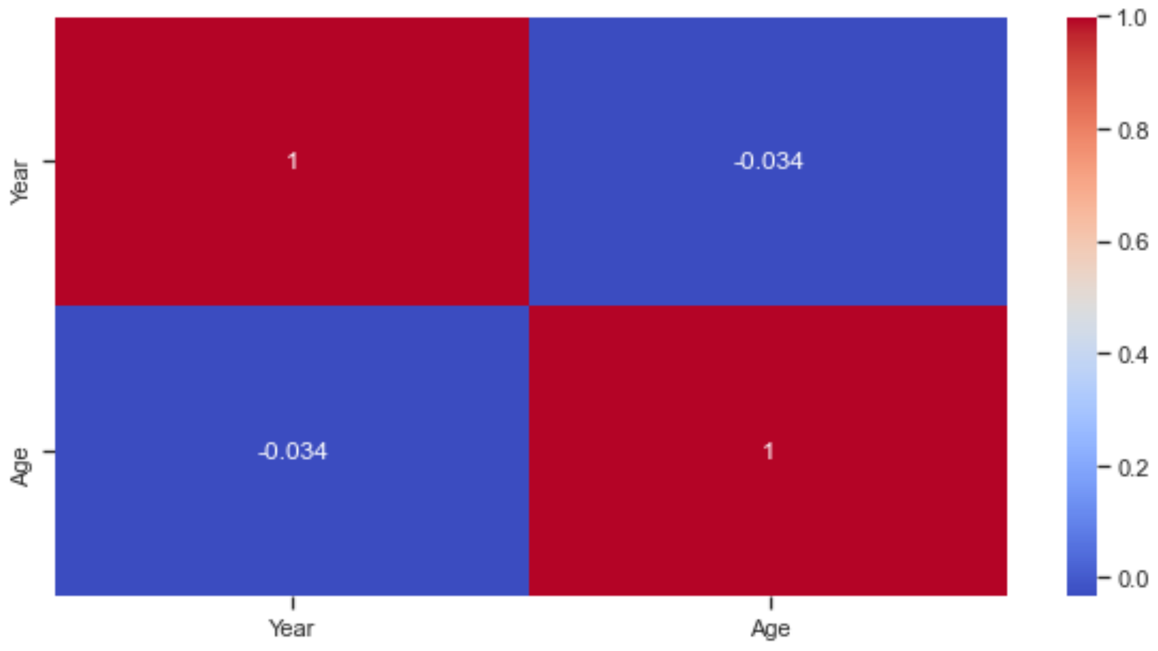
```
df20 = df1[df1.Age < 70]
df20 = df20[df20.Age > 13]
df25 = df1[df1.Age < 67]
df25 = df25[df25.Age > 14]
df30 = df1[df1.Age < 63]
df30 = df30[df30.Age > 15]
```

In [ ]:

```
plt.figure(figsize=(10,5))
c= df1.corr()
sns.heatmap(c,cmap='coolwarm',annot=True)
c
```

Out[ ]:

	Year	Age
Year	1.000000	-0.033776
Age	-0.033776	1.000000



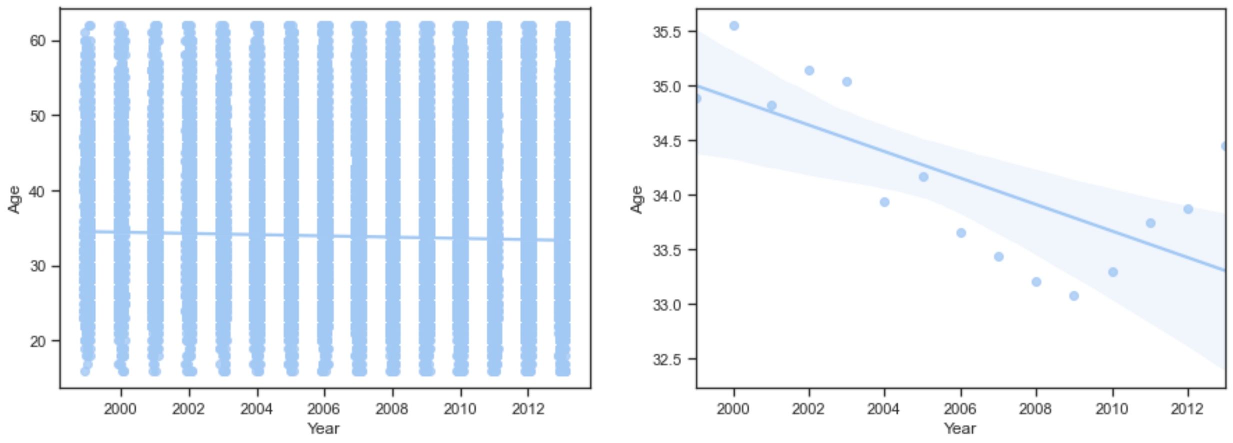
Review of the correlation plot indicates a very weak negative correlation between year and age. Review of regression plot seems to indicate that overall the trend is for the average age to decrease over time. This trend is much more pronounced when only the mean age of each year is plotted.

In [ ]:

```
g = df1.groupby('Year')['Age'].mean().reset_index()
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15, 5), sharey=False)
sns.regplot(x=g["Year"], y=g['Age'], ax=ax2)
sns.regplot(x=df30["Year"], y=df30['Age'],x_jitter=.1, ax=ax1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff288137610>
```

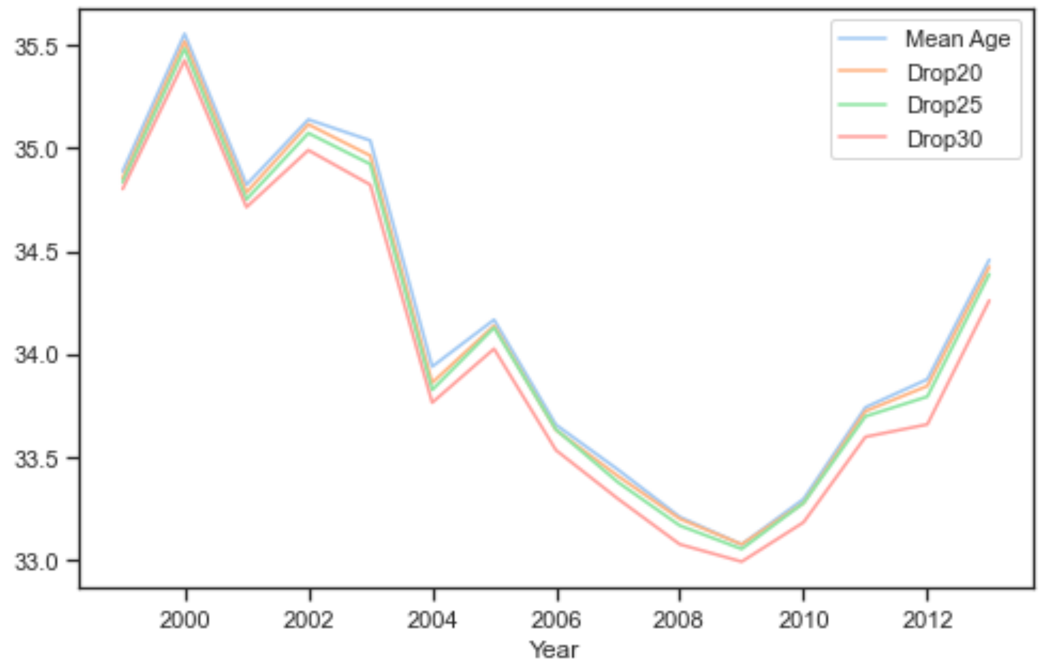


```
In [ ]:
```

```
df1.groupby('Year')['Age'].mean().plot(kind='line', figsize=(8,5), label="Mean A
ge")
df20.groupby('Year')['Age'].mean().plot(kind='line', figsize=(8,5), label="Drop2
0")
df25.groupby('Year')['Age'].mean().plot(kind='line', figsize=(8,5), label="Drop2
5")
df30.groupby('Year')['Age'].mean().plot(kind='line', figsize=(8,5), label="Drop3
0")
plt.legend()
```

```
Out[ ]:
```

```
<matplotlib.legend.Legend at 0x7ff2b9c03f10>
```



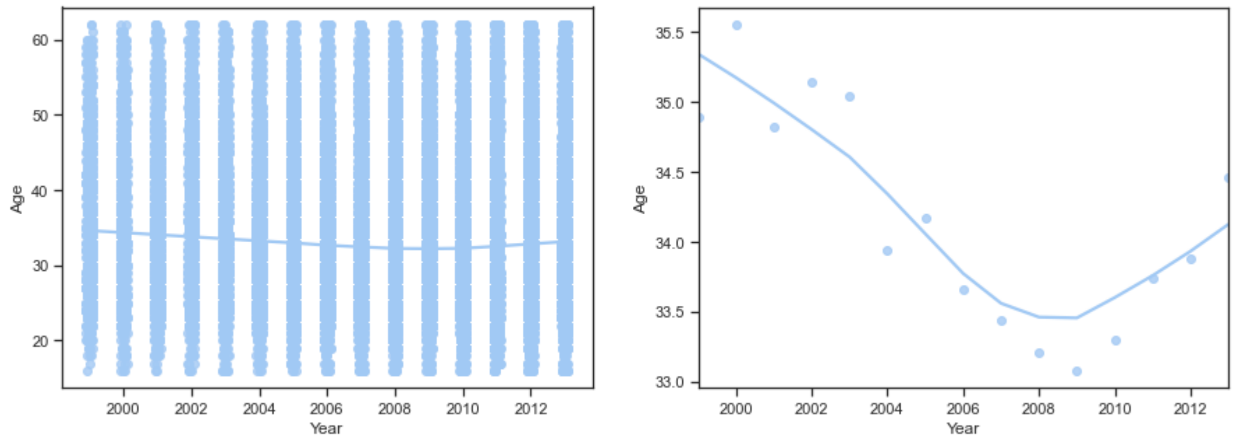
However, when we look at the distribution of mean age for each year, there appears to be change points in the data. To better capture this we plot a linear model plot with Locally Weighted Scatterplot Smoothing. This indicates that there may be a change point that occurs around 2009 where the general decrease in mean age starts an upward trend. Like all the other observations, tests for statistical significance have to be conducted to validate the visual analysis.

In [ ]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15, 5), sharey=False)
sns.regplot(x=g["Year"], y=g['Age'], lowess = True, ax=ax2)
sns.regplot(x=df30["Year"], y=df30['Age'],x_jitter=.1, lowess = True, ax=ax1)
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff29900f2d0>
```



In [ ]:

```
df1[["Pace","Time"]].describe()
```

Out[ ]:

	Pace	Time
count	86169	86169
unique	651	5199
top	00:09:52	1:36:34
freq	477	62

In [ ]:

```
df1['RunTime'] = [int(a) * 60 + int(b) for a,b,c in df1['Time'].str.split(':')]
```

In [ ]:

```
#create PaceTime variable
df1['PaceTime'] = [int(a) * 60 + int(b)* 60 + int(c) for a,b,c in df1['Pace'].st
r.split(':')]
```

Summmary of EDA

Overall, the average age of the runners in these races over the years hovers around the early to mid 30s. Each year, we found that our distribution of ages around the mean is generally right skewed, with most participants being in the age range of 26-50 with participants being as young as 8 and as old as 87. Each year, the distribution of ages changes relatively with the height of the bell curve changing every year. This would suggest that when more/less participants enter the race each year, these participants are most likely going to be very close to the mean. After reviewing the correlation between our remaining time, age and year variables, we saw that there was a very slight negative correlation between age and year, while a slightly more positive correlation between age and time. This gives us evidence to form a hypothesis around our key project question that the age distribution is getting younger each year, despite the fact that repeat participants are going to be getting older with the passing year. Reviewing a line plot showing the age each year, we can see our hypothesis being supported as the average age is generally decreasing from 2000 to 2009 until we see an upswing where the average age is getting older.

Question of Interest

Has the age distribution of Female runners significantly changed over time? If so has the change occurred over time or is there an evident break point?

Hypothesis

Null Hypotheses: There is no change in the Age distributions of each Year.

Alternative Hypotheses: At least, one Year has a different Age distribution.

ANOVA

As the fundamental question of interest is determining if there is a difference in the age distributions over time and each year essentially represents an independent group of runners, we decided to start with ANOVA which compares two means from independent groups using the F-distribution. ANOVA tests against the null hypothesis that there is no significant difference in the means of classes.

Assumptions:

Residuals are normally distributed (Shapiro Wilks Test)

Homogeneity of variances (Bartlett Test)

Observations are sampled independently from each other

Concerns

Our ANOVA dataset consists of 15 unbalanced classes. While there is a slight concern that this will adversely affect homogeneity of variances, we hope that the effects of the unbalanced classes will be mitigated by the large samples per group and the fact that we are testing against only one variable. While we assume that the age distribution of one year should be independent of another we found that of the total of 138,265 registered female runners there were only 82,157 unique names, indicating that up to 40.1% of the runners participated in multiple runs. As runners naturally age from year to year this makes a runners age in one year dependent on their age in a previous year.

In order to conduct ANOVA we convert the Year values in the df20 dataframe from integer to classes by casting them as string objects. We also create a data set of the summary statistics for each year.

In [ ]:

```
#create dateframe of summary statistics by year
dfAgeStats = df1.groupby('Year')['Age'].describe().reset_index()
#create dataframe where Year is a class variable
df20["Year"] = df20["Year"].astype(str)
df1.groupby('Year')['Age'].describe()
```

Out[ ]:

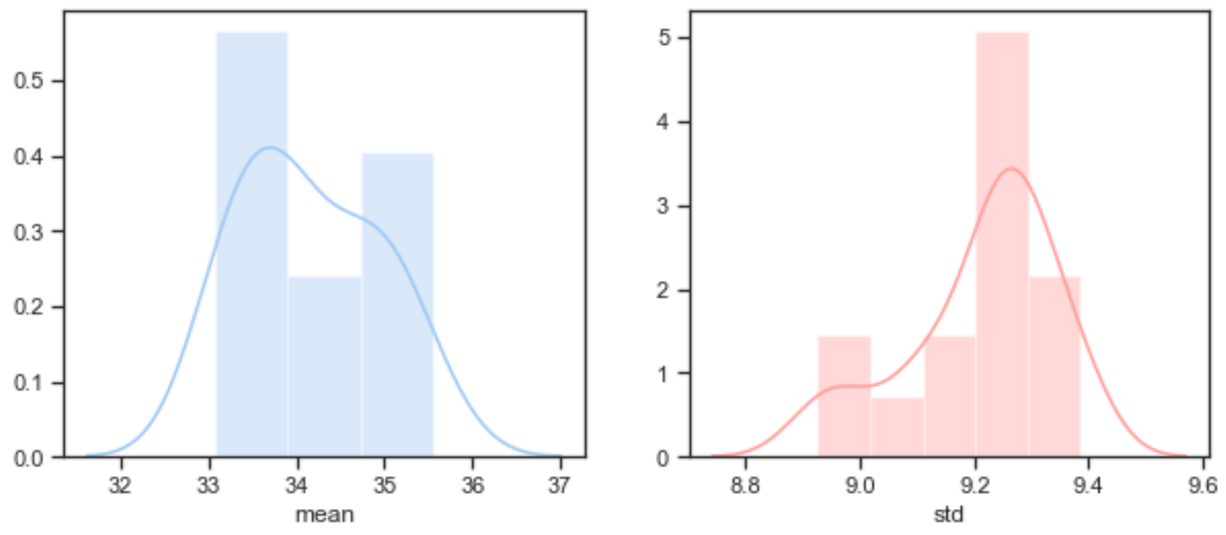
Year	count	mean	std	min	25%	50%	75%	max
1999	2354.0	34.886576	8.925263	12.0	28.0	33.0	41.0	81.0
2000	2166.0	35.553555	9.319658	14.0	29.0	34.0	41.0	76.0
2001	2971.0	34.821272	9.186583	11.0	28.0	33.0	41.0	83.0
2002	3330.0	35.137838	9.231994	13.0	28.0	33.0	41.0	84.0
2003	3525.0	35.036028	9.370971	12.0	28.0	33.0	41.0	85.0
2004	3885.0	33.939254	9.289703	13.0	27.0	32.0	39.0	86.0
2005	4323.0	34.166551	9.385567	10.0	27.0	32.0	40.0	87.0
2006	5435.0	33.657038	9.255073	12.0	26.0	31.0	39.0	76.0
2007	5530.0	33.439964	9.255429	10.0	26.0	31.0	39.0	77.0
2008	6395.0	33.210008	9.117958	11.0	26.0	31.0	38.0	76.0
2009	8322.0	33.076304	8.981805	7.0	26.0	31.0	38.0	73.0
2010	8853.0	33.297074	9.091657	11.0	27.0	31.0	39.0	75.0
2011	9030.0	33.741528	9.212669	11.0	27.0	31.0	39.0	76.0
2012	9727.0	33.877763	9.276222	12.0	27.0	32.0	39.0	75.0
2013	10323.0	34.457231	9.283621	11.0	27.0	32.0	40.0	76.0

In [ ]:

```
#distribution of standard deviations
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4), sharey=False)
#sns.regplot(x="Year", y="std", data=dfAgeStats, ax=ax2)
sns.distplot(dfAgeStats['mean'], ax=ax1)
sns.distplot(dfAgeStats['std'], ax=ax2, color='r')
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff298f97510>
```



We take a quick look at the distribution of the mean and standard deviations of the grouped ages and run our ANOVA. It returns an F score of 31.13 and a p-value significantly lower than our significance threshold of .05 indicating that the mean distribution of at least one group is significantly different than that of the other groups.

In [ ]:

```
#one-way anova model age grouped across year
model = ols('Age ~ Year', data=df20).fit()
table = sm.stats.anova_lm(model, typ=2)
print("           One-way ANOVA of Age distibutions for Years 1999-2013","\n\n",
table)
```

```
          One-way ANOVA of Age distibutions for Years 1999-2013

                 sum_sq       df          F        PR(>F)
Year      3.624647e+04     14.0  31.128203  6.005014e-84
Residual  7.155726e+06  86034.0        NaN           NaN
```

Normal Distribution of Residuals

```
In [ ]:
```

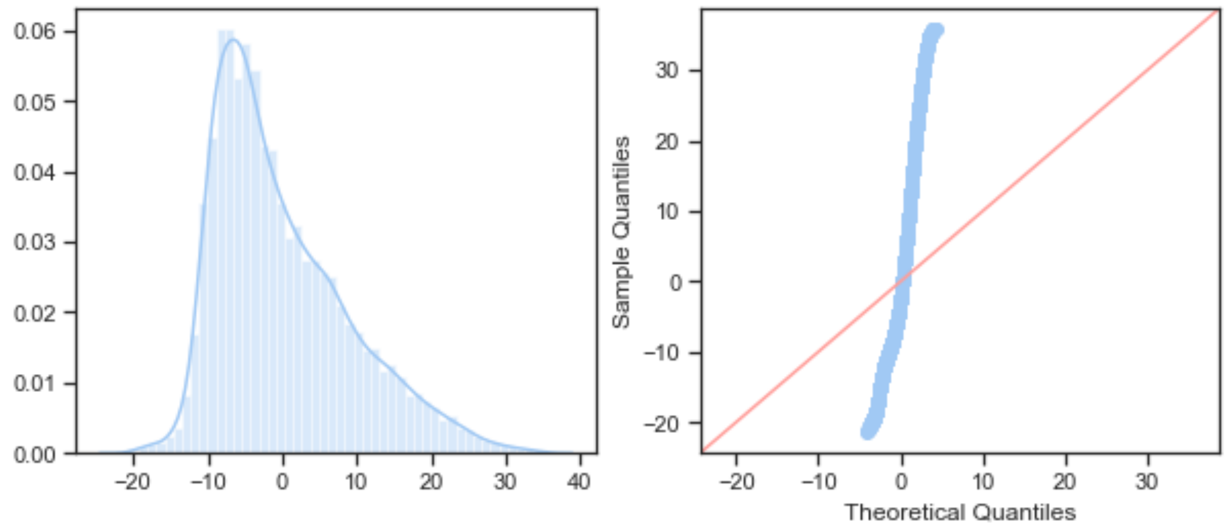
```
#Shapiro-Wilks tests for normality of model residuals
w, pvalue = stats.shapiro(model.resid)
print("Test Statistic against Null Hypothesis of Normal Distribution: ", round(w
,4),'\n' "p=value: ", round(pvalue,4))
```

```
Test Statistic against Null Hypothesis of Normal Distribution:  0.93
58
p=value:  0.0

/Users/chandlervaughn/anaconda3/lib/python3.7/site-packages/scipy/st
ats/morestats.py:1660: UserWarning: p-value may not be accurate for
N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

We run a Shapiro-Wilks test for normality but due to the high sample size the p-value is not conclusive. So we plot the residuals on a distribution plot and a QQ plot. The plots indicate that the residuals are not evenly distributed.

```
In [ ]:
```

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 4), sharey=False)
#plt.figure(figsize=(10,5))
sns.distplot(model.resid, ax=ax1)
sm.qqplot(model.resid, line ='45',ax=ax2)
py.show()
```



Homogeneity of Variances

In [ ]:

```
#bartlett test for homogeneity of variances
w, pvalue = stats.bartlett( df20.loc[df20['Year'] == "1999", 'Age'], df20.loc[df
20['Year'] == "2000", 'Age'], df20.loc[df20['Year'] == "2001", 'Age'],
                            df20.loc[df20['Year'] == "2003", 'Age'], df20.loc[df
20['Year'] == "2004", 'Age'], df20.loc[df20['Year'] == "2005", 'Age'],
                            df20.loc[df20['Year'] == "2006", 'Age'], df20.loc[df
20['Year'] == "2007", 'Age'], df20.loc[df20['Year'] == "2008", 'Age'],
                            df20.loc[df20['Year'] == "2009", 'Age'], df20.loc[df
20['Year'] == "2010", 'Age'], df20.loc[df20['Year'] == "2011", 'Age'],
                            df20.loc[df20['Year'] == "2012", 'Age'], df20.loc[df
20['Year'] == "2013", 'Age'])
print("Test Statistic against Null Hypothesis of equal variance: ", round(w,4),'
\n' "p=value: ", round(pvalue,4))
```

```
Test Statistic against Null Hypothesis of equal variance:  27.1705
p=value:  0.0118
```

We go on to test the homogeneity of variances using a Bartlett test and the resulting p-value of 0.0118 indicates that we cannot assume homogeneity of variances. Though our assumptions were not properly satisfied we still go on and conduct a multiple pairwise comparison of the groups as the EDA still validates the results of the ANOVA.

Tukey HD

In [ ]:

```
# perform multiple pairwise comparison (Tukey HSD)
m_comp = pairwise_tukeyhsd(endog=df20['Age'], groups=df20['Year'], alpha=0.05)
print(m_comp)
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower   upper  reject
----------------------------------------------------
  1999   2000   0.6635 0.4855 -0.2581  1.5851  False
  1999   2001  -0.0697    0.9  -0.924  0.7847  False
  1999   2002   0.2609    0.9 -0.5728  1.0945  False
  1999   2003   0.1096    0.9 -0.7146  0.9338  False
  1999   2004  -0.9919 0.0029 -1.8007 -0.1832   True
  1999   2005  -0.7155 0.1305 -1.5085  0.0775  False
  1999   2006  -1.2216  0.001 -1.9855 -0.4578   True
  1999   2007  -1.4444  0.001 -2.2062 -0.6825   True
  1999   2008  -1.6518  0.001 -2.3982 -0.9055   True
  1999   2009  -1.7763  0.001 -2.4991 -1.0535   True
  1999   2010   -1.575  0.001 -2.2929  -0.857   True
  1999   2011  -1.1301  0.001 -1.8466 -0.4136   True
  1999   2012   -1.008  0.001 -1.7192 -0.2969   True
  1999   2013  -0.4298 0.7307  -1.137  0.2773  False
  2000   2001  -0.7332  0.224 -1.6077  0.1413  False
  2000   2002  -0.4027    0.9 -1.2569  0.4516  False
  2000   2003  -0.5539 0.6267 -1.3989  0.2912  False
  2000   2004  -1.6554  0.001 -2.4854 -0.8254   True
  2000   2005   -1.379  0.001 -2.1937 -0.5644   True
  2000   2006  -1.8851  0.001 -2.6714 -1.0988   True
  2000   2007  -2.1079  0.001 -2.8923 -1.3235   True
  2000   2008  -2.3154  0.001 -3.0847 -1.5461   True
  2000   2009  -2.4398  0.001 -3.1863 -1.6933   True
  2000   2010  -2.2385  0.001 -2.9803 -1.4967   True
  2000   2011  -1.7936  0.001  -2.534 -1.0533   True
  2000   2012  -1.6715  0.001 -2.4067 -0.9363   True
  2000   2013  -1.0933  0.001 -1.8247  -0.362   True
  2001   2002   0.3305    0.9 -0.4508  1.1118  False
  2001   2003   0.1793    0.9 -0.5919  0.9504  False
  2001   2004  -0.9223 0.0031 -1.6769 -0.1676   True
  2001   2005  -0.6459  0.166 -1.3836  0.0919  False
  2001   2006   -1.152  0.001 -1.8583 -0.4457   True
  2001   2007  -1.3747  0.001 -2.0789 -0.6706   True
  2001   2008  -1.5822  0.001 -2.2695 -0.8949   True
  2001   2009  -1.7067  0.001 -2.3684  -1.045   True
  2001   2010  -1.5053  0.001 -2.1617 -0.8489   True
  2001   2011  -1.0605  0.001 -1.7153 -0.4057   True
  2001   2012  -0.9384  0.001 -1.5873 -0.2894   True
  2001   2013  -0.3602 0.8377 -1.0047  0.2844  False
  2002   2003  -0.1512    0.9 -0.8994  0.5969  False
  2002   2004  -1.2528  0.001 -1.9839 -0.5217   True
  2002   2005  -0.9764  0.001 -1.6901 -0.2627   True
  2002   2006  -1.4825  0.001 -2.1636 -0.8014   True
  2002   2007  -1.7052  0.001 -2.3841 -1.0263   True
  2002   2008  -1.9127  0.001 -2.5741 -1.2513   True
  2002   2009  -2.0372  0.001 -2.6719 -1.4024   True
  2002   2010  -1.8358  0.001  -2.465 -1.2066   True
  2002   2011   -1.391  0.001 -2.0185 -0.7634   True
  2002   2012  -1.2689  0.001 -1.8903 -0.6474   True
  2002   2013  -0.6907 0.0123 -1.3075 -0.0738   True
  2003   2004  -1.1016  0.001 -1.8219 -0.3813   True
  2003   2005  -0.8252  0.006 -1.5277 -0.1226   True
  2003   2006  -1.3313  0.001 -2.0007 -0.6618   True
  2003   2007   -1.554  0.001 -2.2212 -0.8868   True
  2003   2008  -1.7615  0.001 -2.4109  -1.112   True
  2003   2009  -1.8859  0.001 -2.5082 -1.2637   True
  2003   2010  -1.6846  0.001 -2.3012  -1.068   True
  2003   2011  -1.2398  0.001 -1.8547 -0.6249   True
  2003   2012  -1.1176  0.001 -1.7263  -0.509   True
  2003   2013  -0.5394 0.1419 -1.1434  0.0646  False
  2004   2005   0.2764    0.9  -0.408  0.9608  False
  2004   2006  -0.2297    0.9 -0.8801  0.4207  False
  2004   2007  -0.4524 0.5336 -1.1005  0.1956  False
  2004   2008  -0.6599 0.0294 -1.2896 -0.0302   True
  2004   2009  -0.7844  0.001  -1.386 -0.1827   True
  2004   2010   -0.583  0.063 -1.1789  0.0128  False
  2004   2011  -0.1382    0.9 -0.7323  0.4559  False
  2004   2012  -0.0161    0.9 -0.6037  0.5715  False
  2004   2013   0.5621 0.0728 -0.0206  1.1449  False
  2005   2006  -0.5061 0.2921 -1.1368  0.1246  False
  2005   2007  -0.7288 0.0073 -1.3572 -0.1005   True
  2005   2008  -0.9363  0.001 -1.5457 -0.3269   True
  2005   2009  -1.0608  0.001 -1.6411 -0.4804   True
  2005   2010  -0.8594  0.001 -1.4337 -0.2852   True
  2005   2011  -0.4146 0.4756  -0.987  0.1578  False
  2005   2012  -0.2925    0.9 -0.8582  0.2733  False
  2005   2013   0.2857    0.9  -0.275  0.8464  False
  2006   2007  -0.2227    0.9 -0.8138  0.3683  False
  2006   2008  -0.4302 0.4044 -1.0011  0.1407  False
  2006   2009  -0.5547 0.0371 -1.0945 -0.0149   True
  2006   2010  -0.3533 0.6111 -0.8866  0.1799  False
  2006   2011   0.0915    0.9 -0.4398  0.6228  False
  2006   2012   0.2136    0.9 -0.3104  0.7377  False
  2006   2013   0.7918  0.001  0.2732  1.3105   True
  2007   2008  -0.2075    0.9 -0.7757  0.3608  False
  2007   2009  -0.3319 0.7082 -0.8689  0.2051  False
  2007   2010  -0.1306    0.9  -0.661  0.3998  False
  2007   2011   0.3142 0.7595 -0.2142  0.8427  False
  2007   2012   0.4364  0.226 -0.0848  0.9575  False
  2007   2013   1.0146  0.001  0.4988  1.5303   True
  2008   2009  -0.1245    0.9 -0.6392  0.3903  False
  2008   2010   0.0769    0.9  -0.431  0.5847  False
  2008   2011   0.5217 0.0355  0.0159  1.0275   True
  2008   2012   0.6438 0.0011  0.1456   1.142   True
  2008   2013    1.222  0.001  0.7296  1.7145   True
  2009   2010   0.2013    0.9 -0.2713  0.6739  False
  2009   2011   0.6462  0.001  0.1758  1.1166   True
  2009   2012   0.7683  0.001  0.3061  1.2305   True
  2009   2013   1.3465  0.001  0.8904  1.8026   True
  2010   2011   0.4448 0.0755  -0.018  0.9077  False
  2010   2012    0.567 0.0022  0.1124  1.0215   True
  2010   2013   1.1452  0.001  0.6969  1.5935   True
  2011   2012   0.1221    0.9 -0.3301  0.5744  False
  2011   2013   0.7003  0.001  0.2544  1.1463   True
  2012   2013   0.5782  0.001  0.1409  1.0155   True
----------------------------------------------------
```

<u>Change Point / Mean Diff</u>

2004: -0.9919 2008: -0.6599 2011: 0.5217

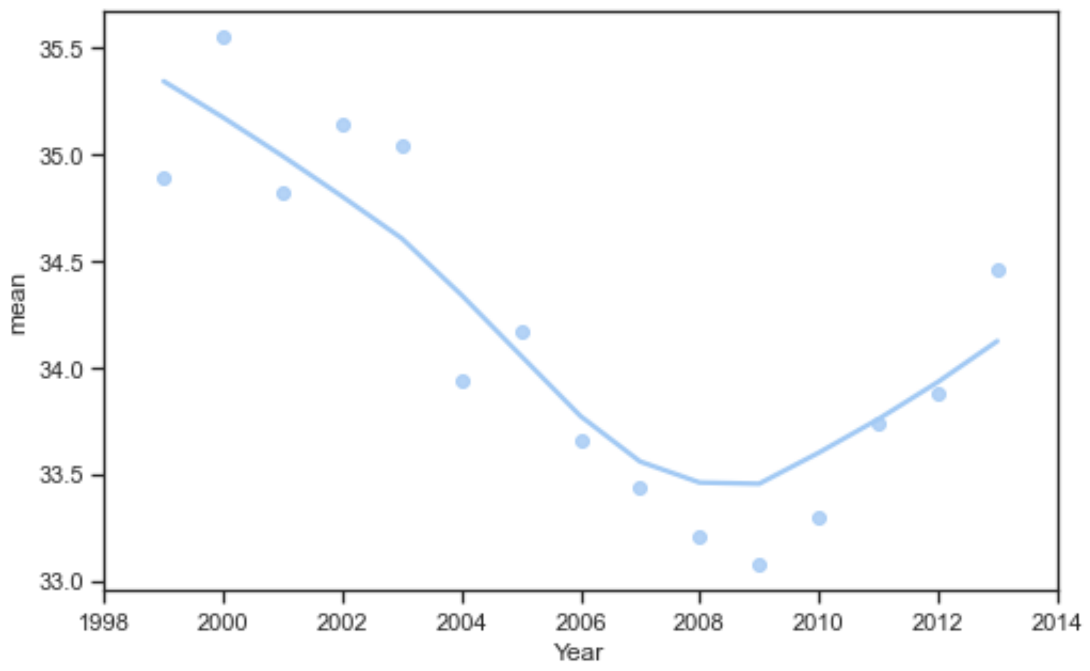
The pairwise comparison identifies 3 possible change points. Through the points seem to correlate with our visual analysis, the Tukey HD test requires that the same assumptions be met as ANOVA, so we merely note these points but move on to a nonparametric comparison of distributions and do a pairwise comparison using the Kolmogorov-Smirnov test of equal distribution.

```
In [ ]:
```

```
plt.figure(figsize=(8,5))
sns.regplot(x="Year", y="mean", data=dfAgeStats, lowess=True)
plt.xlim(1998, 2014)
```

```
Out[ ]:
```

```
(1998.0, 2014.0)
```



In [ ]:

```
#Kolmogorov-Smirnov test of equal distribution
from scipy.stats import ks_2samp
y1=1999
y2=2000

while y1 < 2013:
    x = df30.loc[df30['Year'] == y1, 'Age']; y = df30.loc[df30['Year'] == y2, 'A
ge']
    print(y1,"/",y2, " p-value", round(ks_2samp(x, y)[1],8))
    y1+=1
    y2+=1
```

```
1999 / 2000  p-value 0.10283367
2000 / 2001  p-value 0.99561583
2001 / 2002  p-value 1.0
2002 / 2003  p-value 1.0
2003 / 2004  p-value 1.0
2004 / 2005  p-value 1.0
2005 / 2006  p-value 1.0
2006 / 2007  p-value 1.0
2007 / 2008  p-value 1.0
2008 / 2009  p-value 1.0
2009 / 2010  p-value 1.0
2010 / 2011  p-value 1.0
2011 / 2012  p-value 1.0
2012 / 2013  p-value 5.2e-07
```

Using the Kolmogorov-Smirnov test we identify a change point as a year pairing that is significantly different, but where neither the previous pairing nor subsequent pairing are significantly different and where three consecutive similar pairings are considered a trend. The results of the tests suggest that at a significance level of $p < 0.01$; 2000:2001, 2003:2004, and 2010:2011 indicate that a change point may have occurred.

Linear Regression

Next we will explore linear regression techniques to see if we can forecast future average ages as well as see if we can predict the ages of the runners based on the year.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

In [ ]:

```
#separating input data into two parts X (features) and Y (target)
features = ['Year', 'RunTime', 'PaceTime']

X = df1[features].copy()

#This makes our model's coefficients take on the same scale for accurate feature
importance analysis
#Notice we scaled the data before the cross validation

Y= df1[['Age']].copy()
Y.shape
```

Out[ ]:

```
(86169, 1)
```

We will first set up our cross validation procedure and assign our X and Y variables, using Age as our Y and the Year, Runtime, and Pacetime as our X values.

This set up will allow us to use 80% of our data to train the model, which will be adequate to get a representative sample of our year over year data to test against.

Cross Validation

In [ ]:

```
#Divide data into test and training splits
from sklearn.model_selection import ShuffleSplit
cv = ShuffleSplit(n_splits=10, test_size=0.20, random_state=0)
```

In [ ]:

```
#Use mean absolute error (MAE) to score the regression models created
#(the scale of MAE is identical to the response variable)
from sklearn.metrics import mean_absolute_error, make_scorer, mean_squared_error

#Function for Root mean squared error
#https://stackoverflow.com/questions/17197492/root-mean-square-error-in-python
def rmse(y_actual, y_predicted):
    return np.sqrt(mean_squared_error(y_actual, y_predicted))

#Function for Mean Absolute Percentage Error (MAPE) - Untested
#Adapted from - https://stackoverflow.com/questions/42250958/how-to-optimize-map
e-code-in-python
def mape(y_actual, y_predicted):
    mask = y_actual != 0
    return (np.fabs(y_actual - y_predicted)/y_actual)[mask].mean() * 100

#Create scorers for rmse and mape functions
mae_scorer = make_scorer(score_func=mean_absolute_error, greater_is_better=False
)
rmse_scorer = make_scorer(score_func=rmse, greater_is_better=False)
mape_scorer = make_scorer(score_func=mape, greater_is_better=False)

#Make scorer array to pass into cross_validate() function for producing mutiple
scores for each cv fold.
errorScoring = {'MAE':  mae_scorer,
                'RMSE': rmse_scorer,
                'MAPE': mape_scorer
               }
```

```
In [ ]:
```

```
from sklearn.model_selection import cross_validate

def EvaluateRegressionEstimator(regEstimator, X, y, cv):

    scores = cross_validate(regEstimator, X, y, scoring=errorScoring, cv=cv, ret
urn_train_score=True)

    #cross val score sign-flips the outputs of MAE
    # https://github.com/scikit-learn/scikit-learn/issues/2439
    scores['test_MAE'] = scores['test_MAE'] * -1
    scores['test_MAPE'] = scores['test_MAPE'] * -1
    scores['test_RMSE'] = scores['test_RMSE'] * -1

    #print mean MAE for all folds
    maeAvg = scores['test_MAE'].mean()
    print_str = "The average MAE for all cv folds is: \t\t\t {maeAvg:.5}"
    print(print_str.format(maeAvg=maeAvg))

    #print mean test_MAPE for all folds
    scores['test_MAPE'] = scores['test_MAPE']
    mape_avg = scores['test_MAPE'].mean()
    print_str = "The average MAE percentage (MAPE) for all cv folds is: \t {mape
_avg:.5}"
    print(print_str.format(mape_avg=mape_avg))

    #print mean MAE for all folds
    RMSEavg = scores['test_RMSE'].mean()
    print_str = "The average RMSE for all cv folds is: \t\t\t {RMSEavg:.5}"
    print(print_str.format(RMSEavg=RMSEavg))
    print('*********************************************************')

    print('Cross Validation Fold Mean Error Scores')
    scoresResults = pd.DataFrame()
    scoresResults['MAE'] = scores['test_MAE']
    scoresResults['MAPE'] = scores['test_MAPE']
    scoresResults['RMSE'] = scores['test_RMSE']
    return scoresResults
```

Making Custom Estimators

In [ ]:

```
#Make new estimator compatible for use with GridSearchCV() and cross_validate()
# -  Cap predict function for LinearRegression between 0 and 100
# -  See: Roll your own estimator links above for details.
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.linear_model import LinearRegression

class CappedLinearRegression(LinearRegression):

    def predict(self, X):
        return np.clip(super(CappedLinearRegression, self).predict(X), 0, 100)
```

Next we will build a custom estimator using the Base linear regression procedure.

Baseline LInear Regression Grid Search

In [ ]:

```
#Create a Linear Regression object and perform a grid search to find the best pa
rameters
linreg = CappedLinearRegression()
parameters3 = {'normalize':(True,False), 'fit_intercept':(True,False)}

#Create a grid search object using the
from sklearn.model_selection import GridSearchCV
regGridSearchbase = GridSearchCV(estimator=linreg
                     , verbose=1 # low verbosity
                     , param_grid=parameters3
                     , cv=cv # KFolds = 10
                     , scoring=mae_scorer)

#Perform hyperparameter search to find the best combination of parameters for ou
r data
regGridSearchbase.fit(X, Y)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:    0.5s finishe
d
```

Out[ ]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=0, test_size=
0.2, train_size=None),
             error_score=nan,
             estimator=CappedLinearRegression(copy_X=True, fit_inter
cept=True,
                                               n_jobs=None, normalize
=False),
             iid='deprecated', n_jobs=None,
             param_grid={'fit_intercept': (True, False),
                         'normalize': (True, False)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring=make_scorer(mean_absolute_error, greater_is_bet
ter=False),
             verbose=1)
```

In [ ]:

```
#Print the parameterization of the best estimator
regGridSearchbase.best_estimator_
```

Out[ ]:

```
CappedLinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                       normalize=True)
```

In [ ]:

```
#Create CappedLinearRegression predictions between 0 and 100% using the best par
ameters for our Linear Regression object
regEstimatorbase = regGridSearchbase.best_estimator_

#Evaluate the regression estimator above using our pre-defined cross validation
and scoring metrics.
EvaluateRegressionEstimator(regEstimatorbase, X, Y, cv)
```

```
The average MAE for all cv folds is:                     7.3529
The average MAE percentage (MAPE) for all cv folds is:   22.332
The average RMSE for all cv folds is:                    9.1564
*********************************************************
Cross Validation Fold Mean Error Scores
```

Out[ ]:

	MAE	MAPE	RMSE
0	7.362397	22.203248	9.202555
1	7.346731	22.245239	9.178634
2	7.417983	22.592230	9.213945
3	7.367599	22.418641	9.145784
4	7.361852	22.244816	9.170254
5	7.372857	22.381290	9.192536
6	7.296773	22.271703	9.061802
7	7.335179	22.303049	9.122885
8	7.302884	22.333188	9.088240
9	7.364664	22.329016	9.187160

After running our linear regression baseline procedure, we are getting a root mean square error of just over 9 years. This means we are off by an average of 9 years with our estimations, which is not great considering our distribution field.

Next we will see if we can improve our score using other regression methods.

Ridge Regression

This model will use Lasso Regressions (L2 Norm) for regression of continuous variables. Documentation below:

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)

In [ ]:

```
#Create a regression object and perform a grid search to find the best parameter
s
from sklearn.linear_model import Ridge

regrid = Ridge(fit_intercept=True, normalize=True,copy_X=True
          , max_iter=1000, tol=0.0001, random_state=0)

#Test parameters
alpharid = [0.001, 0.1, 1, 5, 10, 20]
solverrid = [ 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']
parametersrid = {'alpha': alpharid, 'solver': solverrid}

#Create a grid search object using the parameters above
from sklearn.model_selection import GridSearchCV
regGridSearchrid = GridSearchCV(estimator=regrid
                   , n_jobs=8 # jobs to run in parallel
                   , verbose=1 # low verbosity
                   , param_grid=parametersrid
                   , cv=cv # KFolds = 10
                   , scoring=mae_scorer)

#Perform hyperparameter search to find the best combination of parameters for ou
r data
regGridSearchrid.fit(X, Y)
```

```
Fitting 10 folds for each of 36 candidates, totalling 360 fits

[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent wo
rkers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    1.8s
/Users/chandlervaughn/anaconda3/lib/python3.7/site-packages/joblib/e
xternals/loky/process_executor.py:706: UserWarning: A worker stopped
while some jobs were given to the executor. This can be caused by a
too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=8)]: Done 320 tasks      | elapsed:    8.8s
[Parallel(n_jobs=8)]: Done 360 out of 360 | elapsed:   10.5s finishe
d
```

Out[ ]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=0, test_size=
0.2, train_size=None),
             error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=T
rue,
                             max_iter=1000, normalize=True, random_s
tate=0,
                             solver='auto', tol=0.0001),
             iid='deprecated', n_jobs=8,
             param_grid={'alpha': [0.001, 0.1, 1, 5, 10, 20],
                         'solver': ['svd', 'cholesky', 'lsqr', 'spar
se_cg',
                                    'sag', 'saga']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring=make_scorer(mean_absolute_error, greater_is_bet
ter=False),
             verbose=1)
```

In [ ]:

```
#Display the best estimator parameters
regGridSearchrid.best_estimator_
```

Out[ ]:

```
Ridge(alpha=0.001, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=True, random_state=0, solver='saga', tol=0.0001)
```

In [ ]:

```
#Create a regression estimator with best parameters for cross validation
regEstimatorrid = regGridSearchrid.best_estimator_

#Evaluate the regression estimator above using our pre-defined cross validation
and scoring metrics.
EvaluateRegressionEstimator(regEstimatorrid, X, Y, cv)
```

```
The average MAE for all cv folds is:                       7.353
The average MAE percentage (MAPE) for all cv folds is:     22.333
The average RMSE for all cv folds is:                      9.1564
*********************************************************
Cross Validation Fold Mean Error Scores
```

Out[ ]:

	MAE	MAPE	RMSE
0	7.361206	22.199728	9.201621
1	7.346964	22.245987	9.178145
2	7.418005	22.593024	9.213746
3	7.367753	22.418738	9.146120
4	7.362280	22.244970	9.170503
5	7.372500	22.380783	9.192101
6	7.297203	22.272974	9.062266
7	7.335715	22.305025	9.123511
8	7.303124	22.333775	9.088812
9	7.365075	22.330452	9.187287

After performing a Ridge Regression, we didn't perform any better than our baseline linear regression. This is likely due to our age group and race times being highly varied. So the relationship between age and race time is going to be muddied by the variances.

Next let's do a simple linear regression to predict the average age of 2013's data using the data from 1999 - 2012.

Now let's set up our dataframe so that we are grouping the ages by year.

In [ ]:

```
dfage = df1.groupby('Year')['Age'].mean().reset_index()
```

In [ ]:

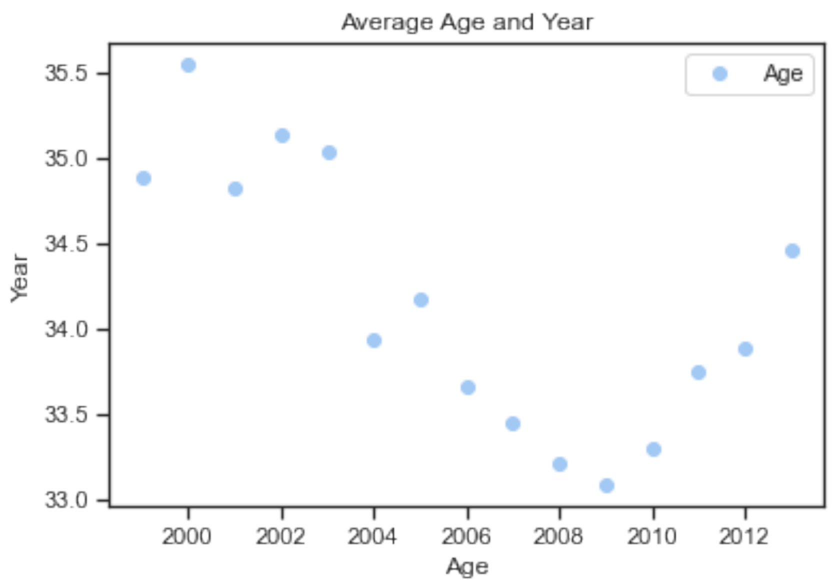
```
dfage
```

Out[ ]:

	Year	Age
0	1999	34.886576
1	2000	35.553555
2	2001	34.821272
3	2002	35.137838
4	2003	35.036028
5	2004	33.939254
6	2005	34.166551
7	2006	33.657038
8	2007	33.439964
9	2008	33.210008
10	2009	33.076304
11	2010	33.297074
12	2011	33.741528
13	2012	33.877763
14	2013	34.457231

In [ ]:

```
dfage.plot(x='Year', y='Age', style='o')
plt.title('Average Age and Year')
plt.xlabel('Age')
plt.ylabel('Year')
plt.show()
```



In [ ]:

```
list(dfage.columns.values)
```

Out[ ]:

```
['Year', 'Age']
```

In [ ]:

```
x = dfage['Year'].values.reshape(-1,1)
y = dfage['Age'].values.reshape(-1,1)
```

We will set our training and testing split so that it will only test our final row, the 2013 data.

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.06, shuffl
e=False)
```

In [ ]:

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
```

Out[ ]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, norma
lize=False)
```

In [ ]:

```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[352.95251056]
[[-0.15897334]]
```

In [ ]:

```
y_pred = regressor.predict(X_test)
```

In [ ]:

```
dfreg = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()}
)
dfreg
```

Out[ ]:

	Actual	Predicted
0	34.457231	32.939182

In [ ]:

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_p
red)))
```

```
Mean Absolute Error: 1.518049114227935
Mean Squared Error: 2.304473113208218
Root Mean Squared Error: 1.518049114227935
```

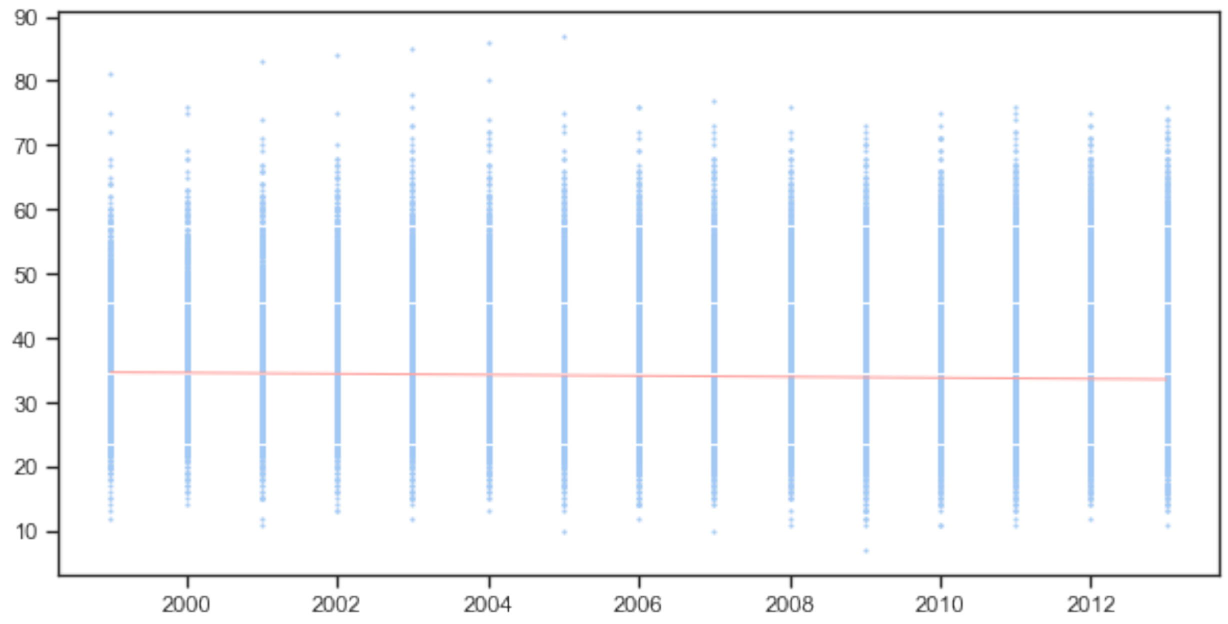
After Running our Linear Regression model against the aggregate average for each year, we found that our difference between actual and predicted average age is less than 1.5 years. While this seems good, in context to the distribution of 30 to 33.5, we can see that the prediction model is not really helping to forecast new average ages in the upcoming year.

Simple Linear Regression

In [ ]:

```
import statsmodels.formula.api as smf
plt.rcParams['figure.figsize'] = (10, 5)
x = df1["Year"]
y = df1["Age"]

# linear regression
f1 = smf.ols(formula = 'Age ~ Year', data = df1).fit()
df1['f1_pred'] = f1.predict()
print(f1.summary())

fig = plt.figure()
fig.patch.set_alpha = 0.5

ax = fig.add_subplot(111)
plt.scatter(x, y, s = 5, marker = '.')
ax.plot(df1.Year, df1.f1_pred, linestyle = '-', linewidth = 1, marker = '', labe
l = r'lin-reg', color='r')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
==========
Dep. Variable:                    Age   R-squared:
0.001
Model:                            OLS   Adj. R-squared:
0.001
Method:                 Least Squares   F-statistic:
98.42
Date:                Mon, 01 Jun 2020   Prob (F-statistic):
3.49e-23
Time:                        15:06:50   Log-Likelihood:                -
3.1372e+05
No. Observations:               86169   AIC:
6.274e+05
Df Residuals:                   86167   BIC:
6.275e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
----------
Intercept    192.0655     15.941     12.048      0.000     160.821
223.310
Year          -0.0788      0.008     -9.920      0.000      -0.094
-0.063
==============================================================================
==========
Omnibus:                    10192.192   Durbin-Watson:
1.914
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
14228.722
Skew:                           0.947   Prob(JB):
0.00
Kurtosis:                       3.616   Cond. No.
1.02e+06
==============================================================================
==========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
[2] The condition number is large, 1.02e+06. This might indicate tha
t there are
strong multicollinearity or other numerical problems.
```



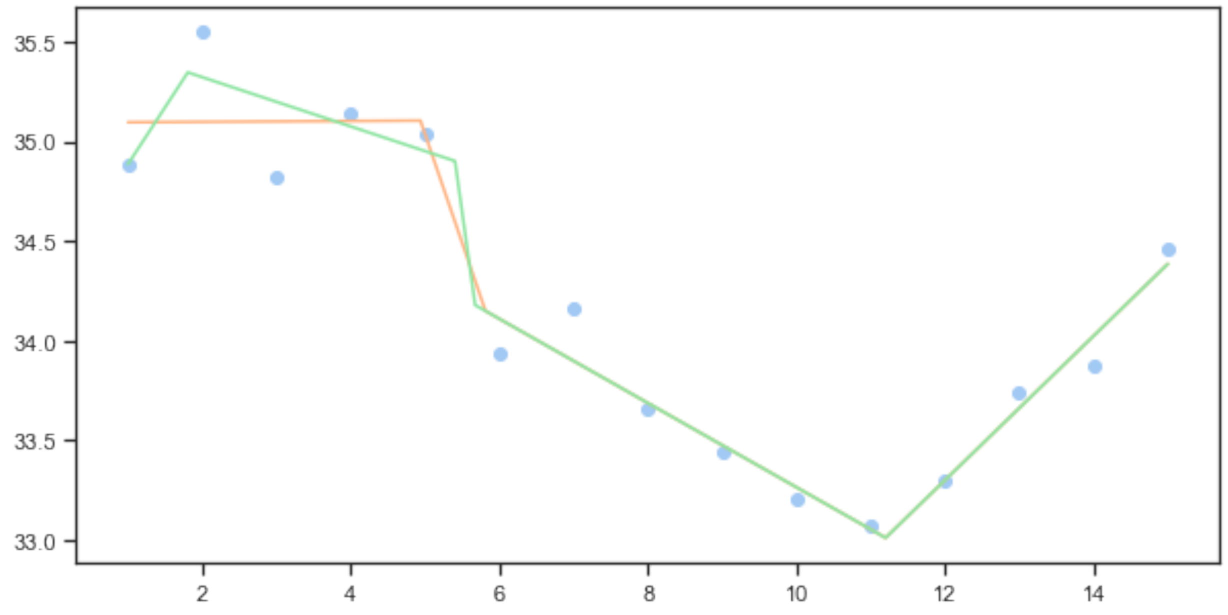
We also performed a statistical analysis of linear regression across the total data set using year and the distribution of age on each. We can tell those this simple linear regression that the slope of the average age per year is decreasing 0.07 years for the passing year with a significant p-value.

Piecewise Regression

Next, we will look into piecewise regression as a type of regression that breaks down the domain into multiple segments and fits a separate line into each. In the case below, we established a piecewise regression using instances with 4 and 5 lines. Here we can better see that we are more closely fitting our distribution of average age over the years. Interestingly, our two lines diverge with the first 7 years, but get much closer together for the remaining 8 years.

In [ ]:

```
import pwlf
# initialize piecewise linear fit with your x and y data
my_pwlf = pwlf.PiecewiseLinFit(x, y)

# fit the data for four line segments
res = my_pwlf.fit(4)

# predict for the determined points
xHat = np.linspace(min(x), max(x), num=10000)
yHat = my_pwlf.predict(xHat)


# fit the data for five line segments
res = my_pwlf.fit(5)
yHat2 = my_pwlf.predict(xHat)

# plot the results for both 4 segments and 5 segments
plt.figure()
plt.plot(x, y, 'o')
plt.plot(xHat, yHat, '-')
plt.plot(xHat, yHat2, '-')
plt.show()
```



Conclusion

Overall, we were tasked with scraping the cherryblossom.org website for the relevant data and statistically evaluate whether the age distribution each year changed over time. After programmatically scraping and cleaning 14 years worth of Women's 10k race results data, we found that there was a significant change in year over year age distribution from 1999 to 2013. We were able to come to this conclusion using an Anova test which was designed to test the mean differences between more than two categories of variables, as well as the simple linear regression test. Additionally, we were able to also visualize the results of the race age data to bolster our statistical analysis that the average age of the racers each year decreased over time. This effect was not linear as we started to see a small increase in the final 3 years, but overall we had a negative relationship between age and the passing year, at least per our time frame.

Deployment

While we had a very specific use case of scraping and analyzing the 1999 - 2013 data of a women's 10k race, the skills gained in web scraping and analysis can have many translations in building programmatic models to constantly evaluate and reevaluate performance after every time a site updates with new data to feed into their time series. This is commonly used in the case of sports betting and daily fantasy sports, where users are scaping up-to-the-minute stats to use in statistical models that refresh constantly. As long as you are taking the proper steps in dealing with your time series, you can use these techniques to build a programmatic and constantly updating machine learning model that changes based on the new data that feeds it.

References

Deborah Nolan and Duncan Temple Lang, "Case Studies in Data Science with R". University of California, Berkeley and University of California, Davis. 2015. http://www.rdatasciencecases.org (http://www.rdatasciencecases.org)