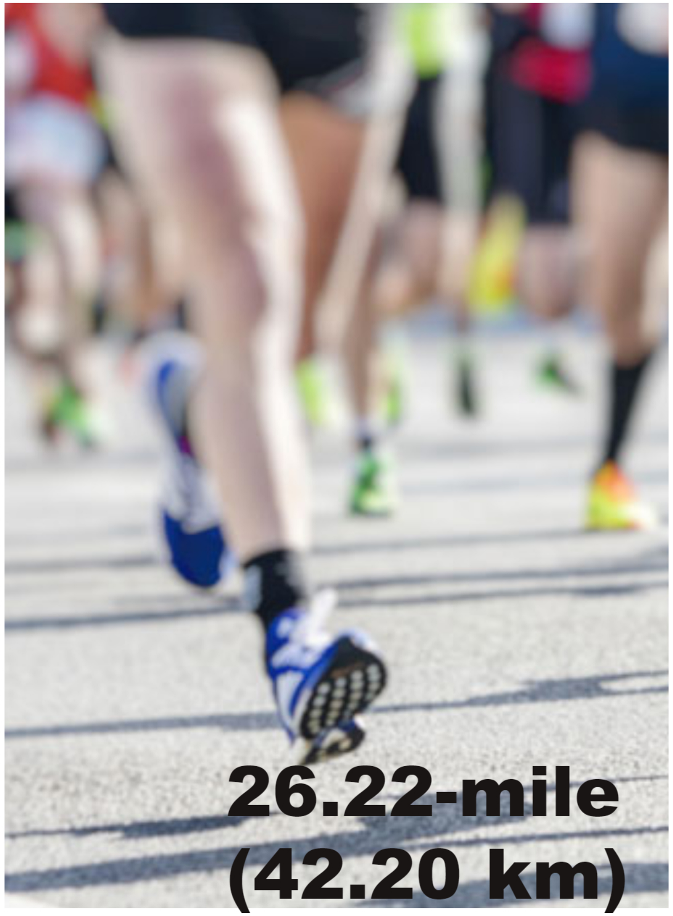
Ironman Triathlon Competition Analysis

- Sheldon Sebastian



An Ironman Triathlon is one of a series of long-distance triathlon races organized by the World Triathlon Corporation (WTC), consisting of a 2.4-mile (3.86 km) swim, a 112-mile (180.25 km) bicycle ride and a marathon 26.22-mile (42.20 km) run, raced in that order.

All the 3 events are all conducted on same day and hence it is widely considered as one of the most difficult one-day sporting events in the world. Any participant who manages to complete the triathlon within these time constraints is designated an Ironman.

Ironman Triathlons are hosted across the world and competitors collect points to participate in **Ironman World Championship** held annually in Hawaii.

Table of Content

In [1]:
```
# import statements
import pandas as pd
import os
import numpy as np
import pycountry  # used this library to get full country name from only 1st 3 letters of country
import plotly.graph_objs as go  # for plotting charts using plotly
from plotly.offline import iplot, init_notebook_mode  # used for plotly offline charts
from plotly.subplots import make_subplots  # used for creating subplots
```

In [2]:
```
# set this property to plot charts offline without chart studio
init_notebook_mode(connected=True)
```

1. Source and Description of data

The data is sourced from following website: http://academictorrents.com/details/2269d7d1c77375aea732eea0905e370d4741575f (http://academictorrents.com/details/2269d7d1c77375aea732eea0905e370d4741575f)

The dataset has records for Ironman *Qualifying* races and *Championship* races from year 2002 to 2016. We are only focusing on Ironman Triathlons and not considering Ironman 70.3 and Ultra-Triathlons competitions.

Back to table of contents

2. Preprocessing and cleaning

When we download the data we notice that there are individual CSV files for each competition based on year and location where competition was held.

All these individual CSV files are stored in *Individual Datasets* Folder.

In [3]:
```
# get all csv names from Individual Datasets directory
fileNames = os.listdir("Individual Datasets/")
```

We will combine all these individual files into 1 CSV file for ease of analysis. We notice that the host location and year is present in the name of individual CSV file.

For example the file name _im_wisconsin*2010.csv* means the competition was held in Wisconsin in year 2010.

We also have championship ironman races named as _im_world-championships*2004.csv* , we will use this to distinguish between championship and qualifying races.

In [4]:
```
fileNames[-20:-5]
```

Out[4]:
```
['im_wisconsin_2010.csv',
 'im_wisconsin_2011.csv',
 'im_wisconsin_2012.csv',
 'im_wisconsin_2013.csv',
 'im_wisconsin_2014.csv',
 'im_wisconsin_2015.csv',
 'im_wisconsin_2016.csv',
 'im_world-championships_2003.csv',
 'im_world-championships_2004.csv',
 'im_world-championships_2005.csv',
 'im_world-championships_2006.csv',
 'im_world-championships_2007.csv',
 'im_world-championships_2008.csv',
 'im_world-championships_2009.csv',
 'im_world-championships_2010.csv']
```

Using the host location and year from file name we add new columns "Host location" and "Year". To distinguish between championship and non championship races we check if file name contains the keyword championship. We then add another new column called "Championship Competition" which will contain boolean values.

In [5]:
```
# create empty dataframe
ironManRaw = pd.DataFrame()

for fileName in fileNames:
    # read individual csv file
    individualFile = pd.read_csv("Individual Datasets/" + fileName)

    # split file name by _ sign
    # using the filename add host_location
    individualFile["Host location"] = fileName.split("_")[1]

    # using the filename add year
    individualFile["Year"] = fileName.split("_")[2].replace(".csv", "")

    # using the filename add championship columns
    individualFile["Championship Competition"] = (
        # check if the word 'championships' present
        "championships"
        in fileName.split("_")[1]
    )

    # append data to common dataframe
    ironManRaw = ironManRaw.append(individualFile)

# write combined data frame into a new CSV
ironManRaw.to_csv("IronManCombined.csv", index=False)
```

We now read the combined IronManCombined.csv file into our pandas dataframe ironManRaw.

In [6]:
```
# reading input files
ironManRaw = pd.read_csv("IronManCombined.csv")
```

The total number of rows present in dataset is ::

In [7]:
```
len(ironManRaw)
```

Out[7]:
```
436131
```

We do initial analysis by seeing the columns present in the dataframe.

In [8]:
```
ironManRaw.columns
```

Out[8]:
```
Index(['name', 'genderRank', 'divRank', 'overallRank', 'bib', 'division',
       'age', 'state', 'country', 'profession', 'points', 'swim',
       'swimDistance', 't1', 'bike', 'bikeDistance', 't2', 'run',
       'runDistance', 'overall', 'Host location', 'Year',
       'Championship Competition'],
      dtype='object')
```

Here the column

- "name" contains name of participant,
- "genderRank" is the rank of pariticipant based on sex,
- "divRank" is rank of pariticipant based on division he belongs to,
- "overallRank" is overall rank in a race,
- "bib" is the bib number,
- "division" is the division the participant belongs to,
- "age" is age of participant,
- "state" is state where participant stays,
- "country" is country where participant stays,
- "profession" is profession of participant,
- "points" is points scored in a competition,
- "swim" is swim time taken by participant,
- "swimDistance" is the distance swam by participant,
- "t1" is transition time from swimming event to biking event by participant,
- "bike" is bike time taken by participant,
- "bikeDistance" is the distance biked by participant,
- "t2" is transition time from biking event to running event by participant,
- "run" is run time taken by participant,
- "runDistance" is the distance ran by participant,
- "overall" is the overall time taken by participant,
- "Host location" is the location where race was held,
- "Year" is the year when the race was held,
- "Championship Competition" specifies whether a race was championship competition or not.

The dataset contains information about ironman races from 2002 to 2016.

In [75]:
```
ironManRaw.sort_values("Year")["Year"].unique()
```

Out[75]:
```
array([2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012,
       2013, 2014, 2015, 2016], dtype=int64)
```

In [9]:
```
ironManRaw.head()
```

Out[9]:

	name	genderRank	divRank	overallRank	bib	division	age	state	country	profession	...	t1	bike	bikeDistance	t2	run	runD
0	Koppo (andrew)	---	---	DNF	830	30-34	---	---	AUS	---	...	---	---	180.2 km	---	---	
1	Hanspeter Abegg	---	---	DNS	50	25-29	---	---	CHE	---	...	---	---	180.2 km	---	---	
2	Alex Abell	---	339	---	51	18-24	---	---	AUS	---	...	---	05:31:53	180.2 km	---	04:05:52	
3	Michael Abrahams	---	405	---	52	30-34	---	---	AUS	---	...	---	---	180.2 km	---	10:47:47	
4	Mike Adair	---	743	---	53	50-54	---	---	NZL	---	...	---	05:44:58	180.2 km	---	04:57:03	

5 rows × 23 columns

Here the missing values are specified by "---", we replace all of them with NaN's.

In [10]:
```
# replace all --- with NaN's
ironManRaw.replace("---", np.NaN, inplace=True)
```

On analyzing the runDistance, swimDistance and bikeDistance we notice that there are some rows, which have 21.1 km, 1.9 km, and 90.1 km values, which are half ironman competition data.

In [11]:
```
print(ironManRaw["runDistance"].unique())
print(ironManRaw["swimDistance"].unique())
print(ironManRaw["bikeDistance"].unique())
```

```
['42.2 km' '26.2 mi' '21.1 km']
['3.9 km' '2.4 mi' '1.9 km']
['180.2 km' '112 mi' '90.1 km']
```

We are only interested in **full ironman competitions** so we delete the rows which contain records for half triathlons.

In [12]:
```
ironManRaw.drop(
    ironManRaw[
        (ironManRaw["swimDistance"] == "1.9 km")
        | (ironManRaw["bikeDistance"] == "90.1 km")
        | (ironManRaw["runDistance"] == "21.1 km")
    ].index,
    inplace=True,
)
```

After deleting those records we print the unique column records.

In [13]:
```
print(ironManRaw["runDistance"].unique())
print(ironManRaw["swimDistance"].unique())
print(ironManRaw["bikeDistance"].unique())
```

```
['42.2 km' '26.2 mi']
['3.9 km' '2.4 mi']
['180.2 km' '112 mi']
```

Deleting columns which are not useful for analysis:

- "name" : We delete this column, since a participants name is not useful for analysis.
- "state" : state column is not useful for analysis
- "genderRank" : The gender of participant is not given, thus gender rank does not provide any useful information.
- "bib" : We delete this column too as it is not useful for analysis.
- "swimDistance" : The swimming distance is not useful as it only specifies whether the participant has swam the entire distance or not.
- "bikeDistance" : The biking distance is not useful as it only specifies whether the participant has biked the entire distance or not.
- "runDistance" : The running distance is not useful as it only specifies whether the participant has ran the entire distance or not.
- "profession" : All profession values are NaN, thus delete this column.
- "points" : Points not useful for analysis
- "t1" : Transition time not useful for analysis
- "t2" : Transition time not useful for analysis

In [14]:
```
# drop unnecessary columns
ironManRaw.drop(
    [
        "name",
        "state",
        "genderRank",
        "bib",
        "runDistance",
        "swimDistance",
        "bikeDistance",
        "profession",
        "points",
        "t1",
        "t2",
    ],
    axis=1,
    inplace=True,
)
```

On analyzing the divisions we notice that there are values for XC which is executive division, PC which is physically challenged division, nknown which is unknown values. We **delete rows** which contain these values as we are not interested in these divisions.

In [15]:
```
ironManRaw.division.unique()
```

Out[15]:
```
array(['30-34', '25-29', '18-24', '50-54', '40-44', '35-39', '45-49',
       '55-59', '60-64', 'PRO', '65-69', '75-79', '70-74', 'XC', 'PC',
       '80+', 'nknown', '80-84', '85-89'], dtype=object)
```

In [16]:
```
ironManRaw.drop(ironManRaw[ironManRaw["division"] == "PC"].index, inplace=True)
ironManRaw.drop(ironManRaw[ironManRaw["division"] == "XC"].index, inplace=True)
ironManRaw.drop(ironManRaw[ironManRaw["division"] == "nknown"].index, inplace=True)
```

Thus preprocessing and cleaning of data is completed and final dataset looks as below:

In [76]:
```
ironManRaw.head()
```

Out[76]:

	divRank	overallRank	division	age	country	swim	bike	run	overall	Host location	Year	Championship Competition
0	NaN	DNF	30-34	NaN	AUS	01:01:25	NaN	NaN	DNF	australia	2005	False
1	NaN	DNS	25-29	NaN	CHE	NaN	NaN	NaN	DNS	australia	2005	False
2	339	NaN	18-24	NaN	AUS	01:01:29	05:31:53	04:05:52	10:39:14	australia	2005	False
3	405	NaN	30-34	NaN	AUS	01:06:17	NaN	10:47:47	10:47:47	australia	2005	False
4	743	NaN	50-54	NaN	NZL	01:03:11	05:44:58	04:57:03	11:45:12	australia	2005	False

Back to table of contents

3. Utility Functions

We are using plotly library to plot charts.

To modularize and reuse code we have created utility functions to plot Pie charts, Line and Bar charts and Choropleth charts. These functions act as wrapper methods to plotly libraries.

3.1 PieMaker Function

We have created a class called PieTemplate in which we pass the input to create a pie chart, and then using this template we call our pieMaker function to plot a pie chart. There is provisionality to also plot **subplots** using the pieMaker function.

In [17]:

```python
class PieTemplate:
    def __init__(self, labels, values, colorList, rowIndex=1, colIndex=1):
        self.labels = labels  # labels to display on pie chart
        self.values = values  # values used to create pie chart
        self.rowIndex = rowIndex  # subplot row position to place pie chart
        self.colIndex = colIndex  # subplot column position to place pie chart
        self.colorList = colorList  # set the color of each marker


def pieMaker(
    listOfPieTemplates, titleForPlot, subplotRow=1, subplotCol=1, subPlotTitle=None
):
    specArr = []

    # based on subplot rows and columns we create the 2D specs array
    for row in range(0, subplotRow):
        rowArr = []
        for col in range(0, subplotCol):
            # when making subplots for PIE charts the type needs to be specified as domain
            rowArr.append({"type": "domain"})
        specArr.append(rowArr)

    # creating graph object Figure using plotly
    pieFig = make_subplots(
        rows=subplotRow, cols=subplotCol, specs=specArr, subplot_titles=subPlotTitle
    )

    # iterating over pieTemplate list
    for pieTemplate in listOfPieTemplates:
        # adding pie Trace
        pieFig.add_trace(
            go.Pie(
                labels=pieTemplate.labels,
                values=pieTemplate.values,
                hoverinfo="label+value",
                marker=dict(colors=pieTemplate.colorList, line=dict(width=1)),
            ),
            pieTemplate.rowIndex,  # specifying row index for subplot
            pieTemplate.colIndex,  # specifying column index for subplot
        )
```

```python
    # adding title for plot using layout
    pieFig.update_layout(title_text=titleForPlot, template="plotly_white")

    return pieFig
```

Thus this is how pieMaker is implemented.

[Back to table of contents](#)

3.2 PlotMaker Function

We have created a class called PlotTemplate in which we pass the input to create either a line chart or bar chart, and then using this template we call our plotMaker function to plot either a line chart or bar chart. There is provisionality to add **drop down buttons** to the charts using plotMaker. In PlotTemplate class the visibility of every trace object by default is False i.e by default all graph objects will be hidden, we need to specify which objects need to be visible, by setting PlotTemplate.visible = True.

In [18]:
```python
# for line and bar charts
class PlotTemplate:
    def __init__(self, x, y, name, color, visible=False):
        self.x = x
        self.y = y
        self.name = name
        self.color = color
        self.visible = visible  # should the graph object be visible or not


def plotMaker(
    listOfPlotTemplates,
    titleOfPlot,
    typeOfChart,  # can be line or bar
    xLabel,  # x axis label
    yLabel,  # y axis label
    listOfButtonNames=None,
    plotsPerButton=None,  # each button controls how many graph objects
):
    # create plotly figure
    plotFig = go.Figure()

    # iterate over evervy plotTemplate in list
    for plotTemplate in listOfPlotTemplates:

        # add traces to figure for each template
        # if typeOfChart is line chart then use go.Scatter
        if typeOfChart.lower() == "line":
            plotFig.add_trace(
                go.Scatter(
                    x=plotTemplate.x,
                    y=plotTemplate.y,
                    name=plotTemplate.name,
                    visible=plotTemplate.visible,
                    marker_color=plotTemplate.color,
                )
            )

        # if typeOfChart is bar chart then use go.Bar
        elif typeOfChart.lower() == "bar":
            plotFig.add_trace(
```

```python
            go.Bar(
                x=plotTemplate.x,
                y=plotTemplate.y,
                name=plotTemplate.name,
                visible=plotTemplate.visible,
                marker_color=plotTemplate.color,
            )
        )

# if buttons are passed as input, then add drop down buttons
if listOfButtonNames != None and plotsPerButton != None:

    buttonsCreated = []

    for index, buttonName in enumerate(listOfButtonNames):
        # initialize visibleArr to size of templates present
        visibleArr = [False] * len(listOfPlotTemplates)

        lower = index * plotsPerButton
        upper = (index + 1) * plotsPerButton

        # set visibleArr from lower index to upper index as True
        for trueIndex in range(lower, upper):
            visibleArr[trueIndex] = True

        buttonsCreated.append(
            dict(label=buttonName, method="update", args=[{"visible": visibleArr}])
        )

    # update layout with buttons
    plotFig.update_layout(
        updatemenus=[go.layout.Updatemenu(buttons=buttonsCreated)],
    )

# update layout with plot title
plotFig.update_layout(
    title=titleOfPlot,
    xaxis=dict(title=xLabel),
    yaxis=dict(title=yLabel),
    template="plotly_white",
)
```

```
    return plotFig
```

This is how plotMaker is implemented. If any doubt refer supplemental content "plotMaker Logic Explanation.txt".

Back to table of contents

3.3 ChoroplethMaker Function

The choroplethMaker is a wrapper function which uses Choropleth graph object.

In [19]:
```python
def choroplethMaker(
    locationsToShow,
    textOnHover,
    colorIntensityFactor,
    colorTheme,
    reverseColorTheme,
    titleForPlot,
):
    # create graph object figure
    choroFig = go.Figure()

    # add choropleth trace
    choroFig.add_trace(
        go.Choropleth(
            locations=locationsToShow,  # country names which are given as 3 letter alphabets
            text=textOnHover,
            z=colorIntensityFactor,  # this determines how dark or light a country's color will be
            colorscale=colorTheme,  # color scale which will be used
            marker_line_color="darkgray",
            reversescale=reverseColorTheme,  # should the colorscale be inverted or not
            marker_line_width=0.5,
        )
    )

    # add layout
    choroFig.update_layout(
        title_text=titleForPlot,
        geo=dict(showframe=True, showcoastlines=True, projection_type="natural earth"),
    )

    return choroFig
```

This is how choroplethMaker is implemented.

Back to table of contents

4. Analysis

We now perform analysis on Ironman triathlon competition.

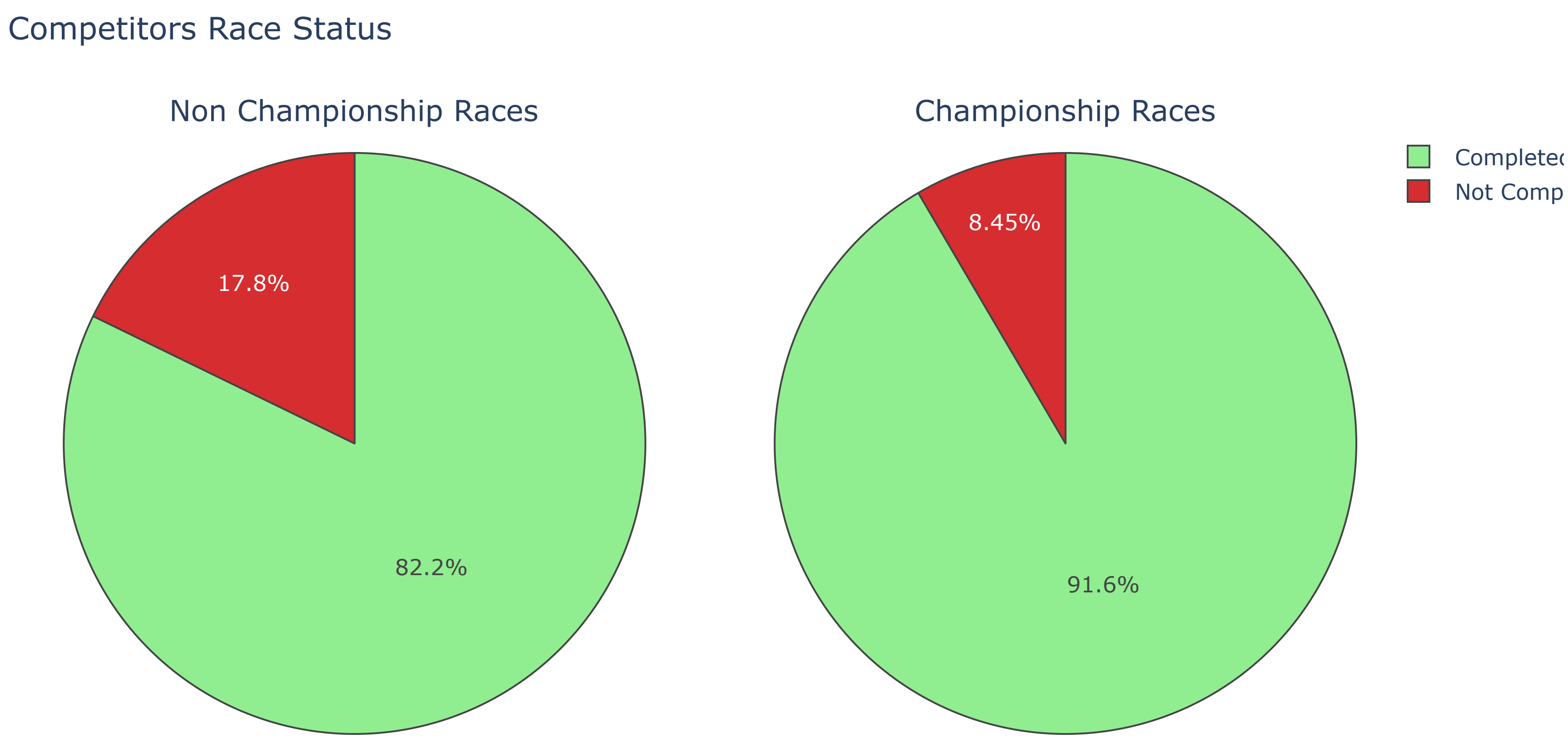
4.1 What percentage of competitors finish the race?

As ironman competitions are very tough and grueling we want to find the percentage of competitors who finished the race. We get count of participants who did not complete race by filtering overall column as *DNS* or Did not start, *DNF* = Did not finish and *DQ* = Disqualified.

We want to see how many percentage of people who completed championship and non championship races, so we divide our dataset into 2 parts, championship and non championship races.

In [20]:
```
# Prepare data
# non championship
notFinishedNonChampion = ironManRaw[
    (
        (ironManRaw["overall"] == "DNS")
        | (ironManRaw["overall"] == "DNF")
        | (ironManRaw["overall"] == "DQ")
    )
    & ~(ironManRaw["Championship Competition"])
]

finishedNonChampion = ironManRaw[
    ~(ironManRaw["Championship Competition"])
    & (
        (ironManRaw["overall"] != "DNS")
        & (ironManRaw["overall"] != "DNF")
        & (ironManRaw["overall"] != "DQ")
    )
]

# championship
notFinishedChampion = ironManRaw[
    (
        (ironManRaw["overall"] == "DNS")
        | (ironManRaw["overall"] == "DNF")
        | (ironManRaw["overall"] == "DQ")
    )
    & (ironManRaw["Championship Competition"])
]

finishedChampion = ironManRaw[
    (ironManRaw["Championship Competition"])
    & (
        (ironManRaw["overall"] != "DNS")
        & (ironManRaw["overall"] != "DNF")
        & (ironManRaw["overall"] != "DQ")
    )
]
```

In [21]:
```
# create pieTemplates
pieNonChampionship = PieTemplate(
    ["Completed Race", "Not Completed Race"],
    [len(finishedNonChampion), len(notFinishedNonChampion)],
    ["lightgreen", "#D62D30"],
    1,
    1,
)
pieChampionship = PieTemplate(
    ["Completed Race", "Not Completed Race"],
    [len(finishedChampion), len(notFinishedChampion)],
    ["lightgreen", "#D62D30"],
    1,
    2,
)

# pass template to pieMaker
raceCompletePie = pieMaker(
    [pieNonChampionship, pieChampionship],
    "Competitors Race Status",
    1,
    2,
    ["Non Championship Races", "Championship Races"],
)

iplot(raceCompletePie)
```



From above pie plots, we conclude that majority of the participants who take part complete the ironman race. We also notice that there is higher race completion in championship races.

Next we want to see completion of ironman races over the years in absolute and percentages.

In [22]:

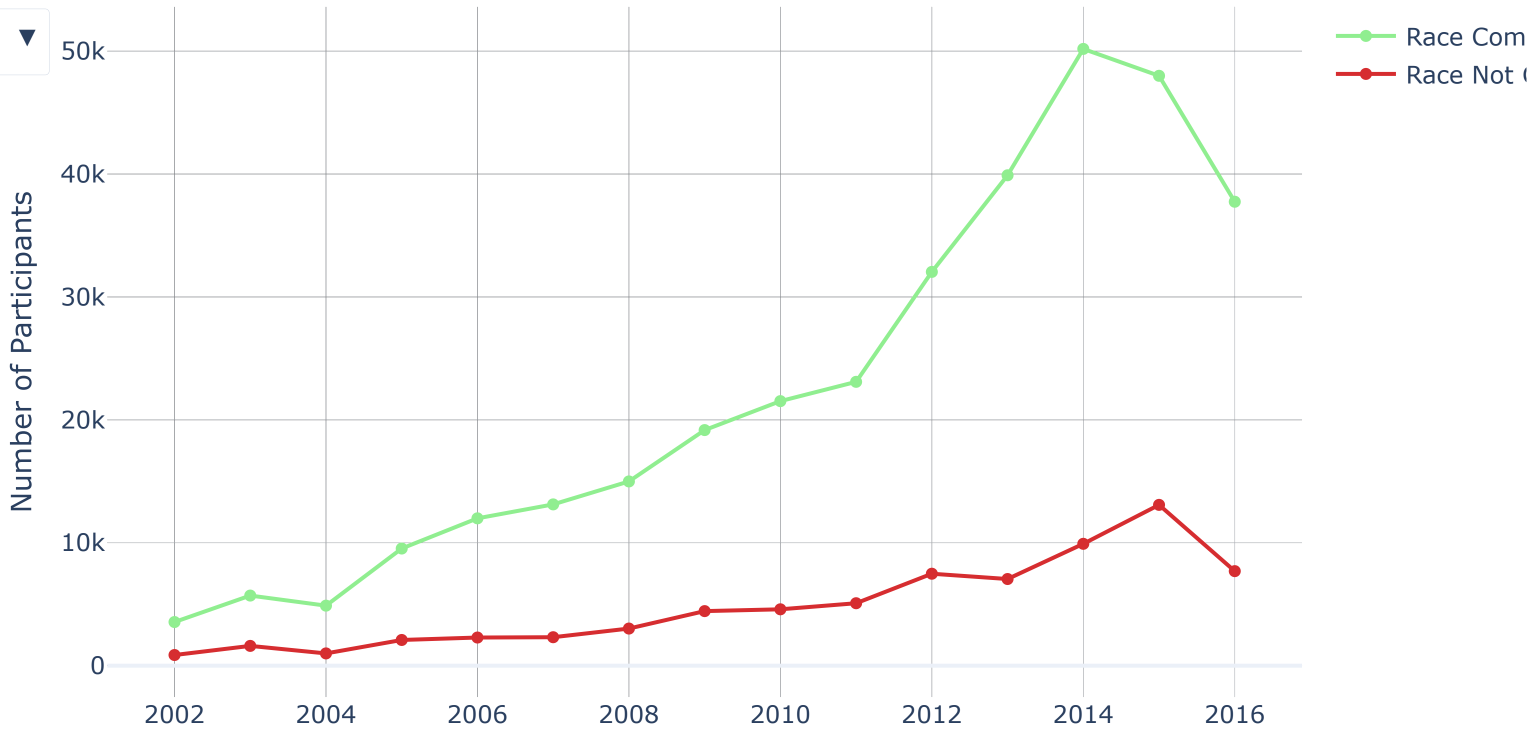
```
# Prepare data
# all championship years
yearChampionship = (
    finishedChampion.groupby("Year").size().reset_index(name="count")["Year"].to_list()
)

# all non championship years
yearNonChampionship = (
    notFinishedNonChampion.groupby("Year")
    .size()
    .reset_index(name="count")["Year"]
    .to_list()
)

# non championship absolute values
notCompletedNonChampionship = (
    notFinishedNonChampion.groupby("Year")
    .size()
    .reset_index(name="count")["count"]
    .to_list()
)
completedNonChampionship = (
    finishedNonChampion.groupby("Year")
    .size()
    .reset_index(name="count")["count"]
    .to_list()
)

# championship absolute values
notCompletedChampionship = (
    notFinishedChampion.groupby("Year")
    .size()
    .reset_index(name="count")["count"]
    .to_list()
)
completedChampionship = (
    finishedChampion.groupby("Year").size().reset_index(name="count")["count"].to_list()
)
```

In [23]:

```python
# color, x-axis and y-axis
# create template
plot1 = PlotTemplate(
    x=yearNonChampionship,
    y=completedNonChampionship,
    name="Race Completed",
    color="lightgreen",
    visible=True,
)
plot2 = PlotTemplate(
    x=yearNonChampionship,
    y=notCompletedNonChampionship,
    name="Race Not Completed",
    color="#D62D30",
    visible=True,
)
plot3 = PlotTemplate(
    x=yearChampionship,
    y=completedChampionship,
    color="lightgreen",
    name="Race Completed",
)
plot4 = PlotTemplate(
    x=yearChampionship,
    y=notCompletedChampionship,
    name="Race Not Completed",
    color="#D62D30",
)

# pass template to plotMakers
finisherAbsoluteFig = plotMaker(
    [plot1, plot2, plot3, plot4],
    "Racers Completing Ironman over the years in absolute values",
    "line",
    "Years",
    "Number of Participants",
    ["Non Championship", "Championship"],
    2,
)

iplot(finisherAbsoluteFig)
```

Racers Completing Ironman over the years in absolute values

Non Championship ▼

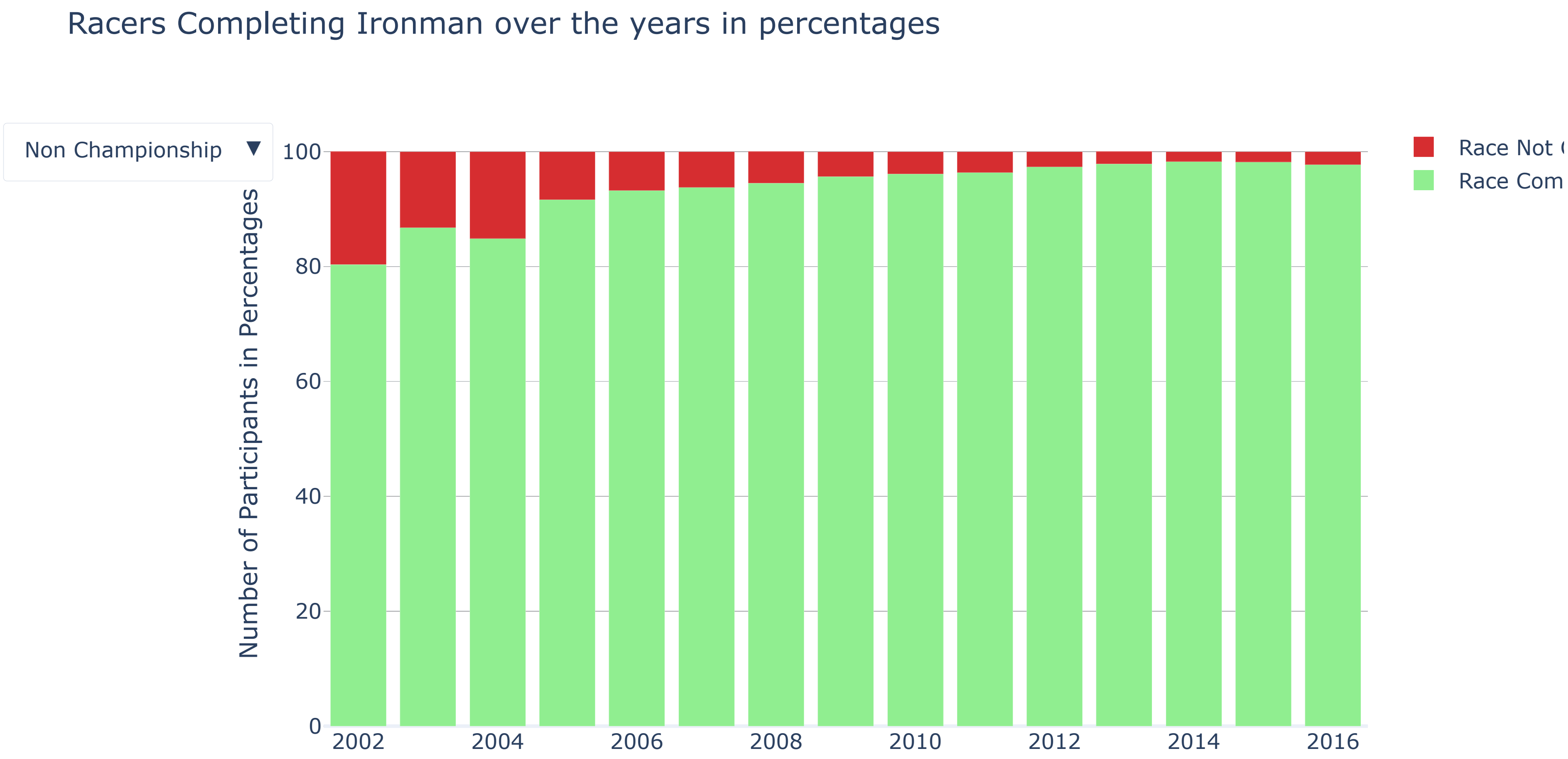


From above graph we see **number of participants increasing over the years** for both non championship and championship races. We also notice that number of people completing the race in non championship is increasing, we now plot the completion of ironman races over the years in percentages.

In [24]:
```
# Prepare data
# non championship
notCompletedNonChampionshipPercentage = [
    (x / (x + y)) * 100
    for x in notCompletedNonChampionship
    for y in completedNonChampionship
]
completedNonChampionshipPercentage = [
    (y / (x + y)) * 100
    for x in notCompletedNonChampionship
    for y in completedNonChampionship
]

# championship
notCompletedChampionshipPercentage = [
    (x / (x + y)) * 100 for x in notCompletedChampionship for y in completedChampionship
]
completedChampionshipPercentage = [
    (y / (x + y)) * 100 for x in notCompletedChampionship for y in completedChampionship
]
```

In [25]:
```
# create template
plot1 = PlotTemplate(
    x=yearNonChampionship,
    y=completedNonChampionshipPercentage,
    name="Race Completed",
    color="lightgreen",
    visible=True,
)
plot2 = PlotTemplate(
    x=yearNonChampionship,
    y=notCompletedNonChampionshipPercentage,
    name="Race Not Completed",
    color="#D62D30",
    visible=True,
)

plot3 = PlotTemplate(
    x=yearChampionship,
    y=completedChampionshipPercentage,
    name="Race Completed",
    color="lightgreen",
)
plot4 = PlotTemplate(
    x=yearChampionship,
    y=notCompletedChampionshipPercentage,
    name="Race Not Completed",
    color="#D62D30",
)

# pass template to plotMakers
finisherPercentFig = plotMaker(
    [plot1, plot2, plot3, plot4],
    "Racers Completing Ironman over the years in percentages",
    "bar",
    "Years",
    "Number of Participants in Percentages",
    ["Non Championship", "Championship"],
    2,
)
finisherPercentFig.update_layout(barmode="stack")
```

```
iplot(finisherPercentFig)
```



From above graph we see that number of participants completing non-championship races are increasing over the years over the years, but for championship races the completion percentage is constant at 91-93%.

[Back to table of contents]

4.2 What is the change in overall competition time over the years?

In our ironManRaw dataframe , we see that we have swim time, bike time and run time as HH:MM:SS, we convert all these values into **minutes** for ease of analysis.

In [26]:
```
ironManRaw.head()
```

Out[26]:

	divRank	overallRank	division	age	country	swim	bike	run	overall	Host location	Year	Championship Competition
0	NaN	DNF	30-34	NaN	AUS	01:01:25	NaN	NaN	DNF	australia	2005	False
1	NaN	DNS	25-29	NaN	CHE	NaN	NaN	NaN	DNS	australia	2005	False
2	339	NaN	18-24	NaN	AUS	01:01:29	05:31:53	04:05:52	10:39:14	australia	2005	False
3	405	NaN	30-34	NaN	AUS	01:06:17	NaN	10:47:47	10:47:47	australia	2005	False
4	743	NaN	50-54	NaN	NZL	01:03:11	05:44:58	04:57:03	11:45:12	australia	2005	False

We create **finisherData** dataframe, that contains only those participants who have finished the ironman triathlon.

```
In [27]: # remove NaN rows
finisherData = ironManRaw.dropna(
    # drop rows based on na's in any of selected columns
    subset=["swim", "bike", "run", "overall"],
    how="any",
)
finisherData.reset_index(inplace=True, drop=True)

# remove participants who did not finish(DNF) or did not start(DNS) or Disqualified(DQ)
finisherData = finisherData.drop(
    finisherData[
        (finisherData["overall"] == "DNF")
        | (finisherData["overall"] == "DNS")
        | (finisherData["overall"] == "DQ")
    ].index
)
```

We use the **convertHHMMSSToMinutes** function which will convert the string timestamps to minutes.

```
In [28]: def convertHHMMSSToMinutes(timeAsString):
    timeArr = timeAsString.split(":")
    return int(timeArr[0]) * 60 + int(timeArr[1]) + int(timeArr[2]) / 60
```

```
In [29]: finisherData["swim"] = finisherData["swim"].apply(convertHHMMSSToMinutes)
finisherData["bike"] = finisherData["bike"].apply(convertHHMMSSToMinutes)
finisherData["run"] = finisherData["run"].apply(convertHHMMSSToMinutes)
finisherData["overall"] = finisherData["overall"].apply(convertHHMMSSToMinutes)
```

After converting to minutes, the finisherData dataframe looks as follows::

In [30]:
```
finisherData.head()
```

Out[30]:

	divRank	overallRank	division	age	country	swim	bike	run	overall	Host location	Year	Championship Competition
0	339	NaN	18-24	NaN	AUS	61.483333	331.883333	245.866667	639.233333	australia	2005	False
1	743	NaN	50-54	NaN	NZL	63.183333	344.966667	297.050000	705.200000	australia	2005	False
2	786	NaN	40-44	NaN	NaN	58.100000	361.983333	292.016667	712.100000	australia	2005	False
3	215	NaN	35-39	NaN	AUS	65.866667	344.033333	204.916667	614.816667	australia	2005	False
4	476	NaN	18-24	NaN	AUS	53.200000	348.383333	255.400000	656.983333	australia	2005	False

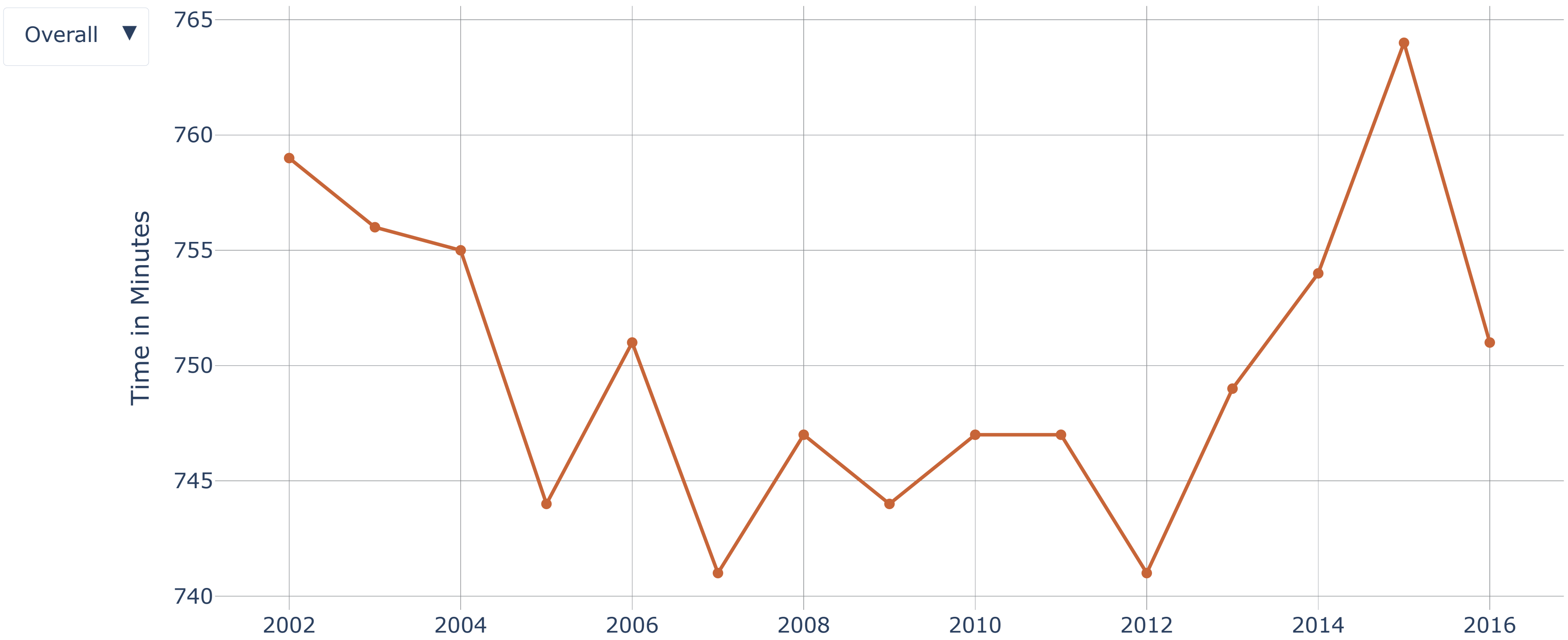
We want to analyze the change in overall competition time, swim time, bike time and run time over the years. For this we group by the finisherData by year and take the average time in minutes. The output of this operation is below:

In [31]:
```
timeOverYears = (
    finisherData.groupby("Year")
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)
```

In [32]:
```
# prepare data
yearForCompetitionTime = timeOverYears.index.to_list()
swimTimeYears = timeOverYears["swim"].to_list()
bikeTimeYears = timeOverYears["bike"].to_list()
runTimeYears = timeOverYears["run"].to_list()
overallTimeYears = timeOverYears["overall"].to_list()
```

In [33]:
```
# create Templates
line1 = PlotTemplate(yearForCompetitionTime, overallTimeYears, "Overall Time", "#C76538", True)
line2 = PlotTemplate(yearForCompetitionTime, swimTimeYears, "Swim Time", "#52C738")
line3 = PlotTemplate(yearForCompetitionTime, bikeTimeYears, "Bike Time", "#389AC7")
line4 = PlotTemplate(yearForCompetitionTime, runTimeYears, "Run Time", "#AD38C7")

# use plotMaker
competitionTimeFig = plotMaker(
    [line1, line2, line3, line4],
    "Competition time over the years(Less is better)",
    "line",
    "Years",
    "Time in Minutes",
    ["Overall", "Swim", "Bike", "Run"],
    1,
)

iplot(competitionTimeFig)
```

Competition time over the years(Less is better)



We notice there is a downward trend in overall time for a participant to complete the race. There is very small variation of swim time over the years, it is in range of 74 to 78 minutes. Similarly bike time is in range of 370 to 380 minutes. The run time decreases over the years too, being the lowest at 278 minutes in 2012.

[Back to table of contents]

4.3 What is the performance difference between championship races and qualifying races?

For this analysis we group the finisherData dataframe by Year and Championship Competition, aggregate using mean and then perform unstacking operations. We can now access each column using multi level index.

In [34]:
```
champVsNonChampRace = (
    finisherData.groupby(["Year", "Championship Competition"])
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
    .unstack()
)
champVsNonChampRace.dropna(inplace=True)
champVsNonChampRace
```

Out[34]:

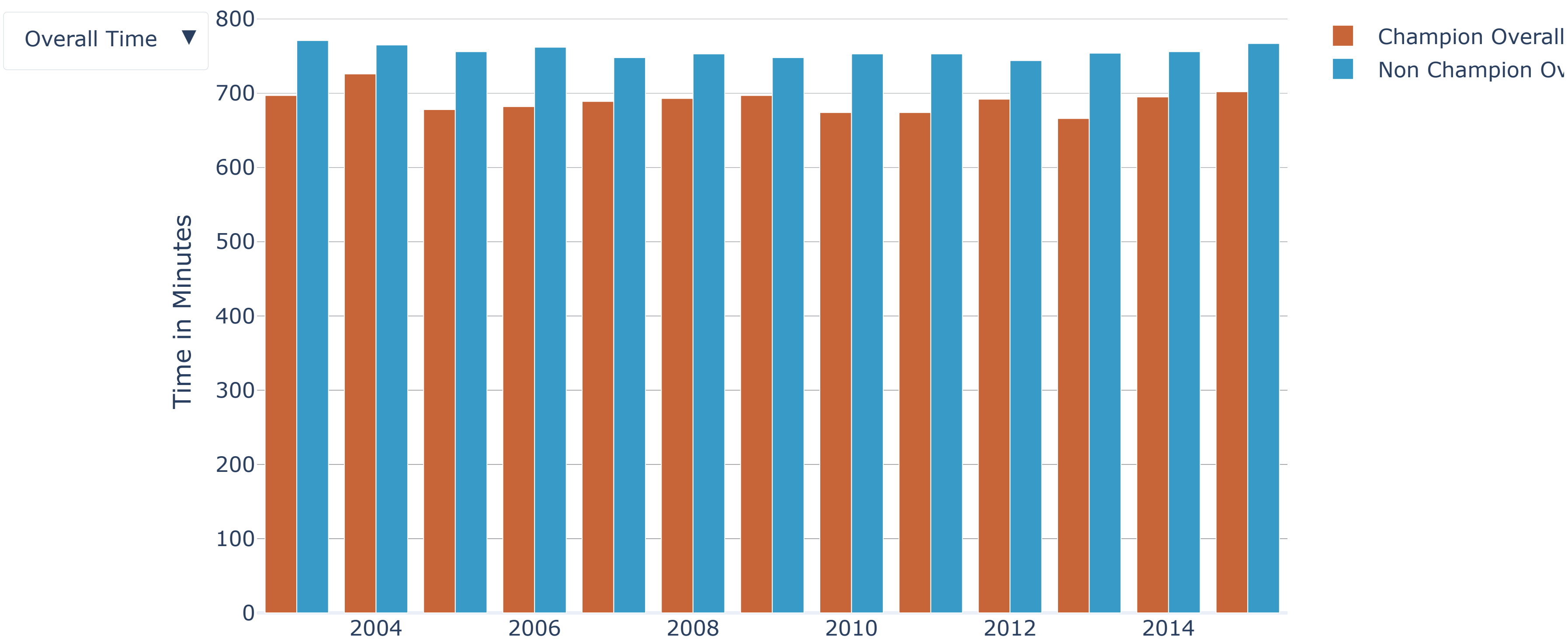
	swim		bike		run		overall	
Championship Competition	**False**	**True**	**False**	**True**	**False**	**True**	**False**	**True**
Year								
2003	74.0	71.0	377.0	357.0	306.0	260.0	771.0	697.0
2004	75.0	73.0	377.0	382.0	297.0	259.0	765.0	726.0
2005	74.0	74.0	373.0	344.0	297.0	250.0	756.0	678.0
2006	76.0	76.0	382.0	347.0	291.0	250.0	762.0	682.0
2007	76.0	73.0	376.0	359.0	284.0	247.0	748.0	689.0
2008	77.0	74.0	374.0	360.0	288.0	250.0	753.0	693.0
2009	77.0	73.0	371.0	356.0	287.0	257.0	748.0	697.0
2010	77.0	74.0	374.0	345.0	288.0	246.0	753.0	674.0
2011	76.0	74.0	377.0	342.0	285.0	248.0	753.0	674.0
2012	78.0	75.0	371.0	353.0	280.0	254.0	744.0	692.0
2013	76.0	72.0	377.0	336.0	286.0	249.0	754.0	666.0
2014	74.0	75.0	380.0	360.0	287.0	250.0	756.0	695.0
2015	77.0	76.0	382.0	352.0	292.0	262.0	767.0	702.0

In [35]:
```python
# prepare data

# championship
yearsPerformance = champVsNonChampRace.index.to_list()
swimTimeChampion = champVsNonChampRace["swim"][True].to_list()
bikeTimeChampion = champVsNonChampRace["bike"][True].to_list()
runTimeChampion = champVsNonChampRace["run"][True].to_list()
overallTimeChampion = champVsNonChampRace["overall"][True].to_list()

# Non championship
swimTimeNonChampion = champVsNonChampRace["swim"][False].to_list()
bikeTimeNonChampion = champVsNonChampRace["bike"][False].to_list()
runTimeNonChampion = champVsNonChampRace["run"][False].to_list()
overallTimeNonChampion = champVsNonChampRace["overall"][False].to_list()
```

In [36]:

```
bar1 = PlotTemplate(
    yearsPerformance,
    overallTimeChampion,
    "Champion Overall Time",
    "#C76538",
    visible=True,
)

bar2 = PlotTemplate(
    yearsPerformance, overallTimeNonChampion, "Non Champion Overall Time", "#389AC7", visible=True
)

bar3 = PlotTemplate(yearsPerformance, swimTimeChampion, "Champion Swim Time", "#79D629")
bar4 = PlotTemplate(
    yearsPerformance, swimTimeNonChampion, "Non Champion Swim Time", "#8629D6"
)

bar5 = PlotTemplate(yearsPerformance, bikeTimeChampion, "Champion Bike Time", "#52ACAD")
bar6 = PlotTemplate(
    yearsPerformance, bikeTimeNonChampion, "Non Champion Bike Time", "#AD5352"
)

bar7 = PlotTemplate(yearsPerformance, runTimeChampion, "Champion Run Time", "#DF20A0")
bar8 = PlotTemplate(
    yearsPerformance, runTimeNonChampion, "Non Champion Run Time", "#20DF5F"
)

champVsNonChampFig = plotMaker(
    [bar1, bar2, bar3, bar4, bar5, bar6, bar7, bar8],
    "Championship V/S Non Championship Performance (Less is Better)",
    "bar",
    "Years",
    "Time in Minutes",
    ["Overall Time", "Swim Time", "Bike Time", "Run Time"],
    2,
)

iplot(champVsNonChampFig)
```

Championship V/S Non Championship Performance (Less is Better)



We notice that overall time, swim time, bike time and run time for championship races are always less than non championship races over the years. A possible reason could be that in championship races, participants have already practiced ironman competition in non championship races, thus their performance could improve in championship races.

[Back to table of contents]

4.4 What are the performance differences between amateur competitors and professional competitors?

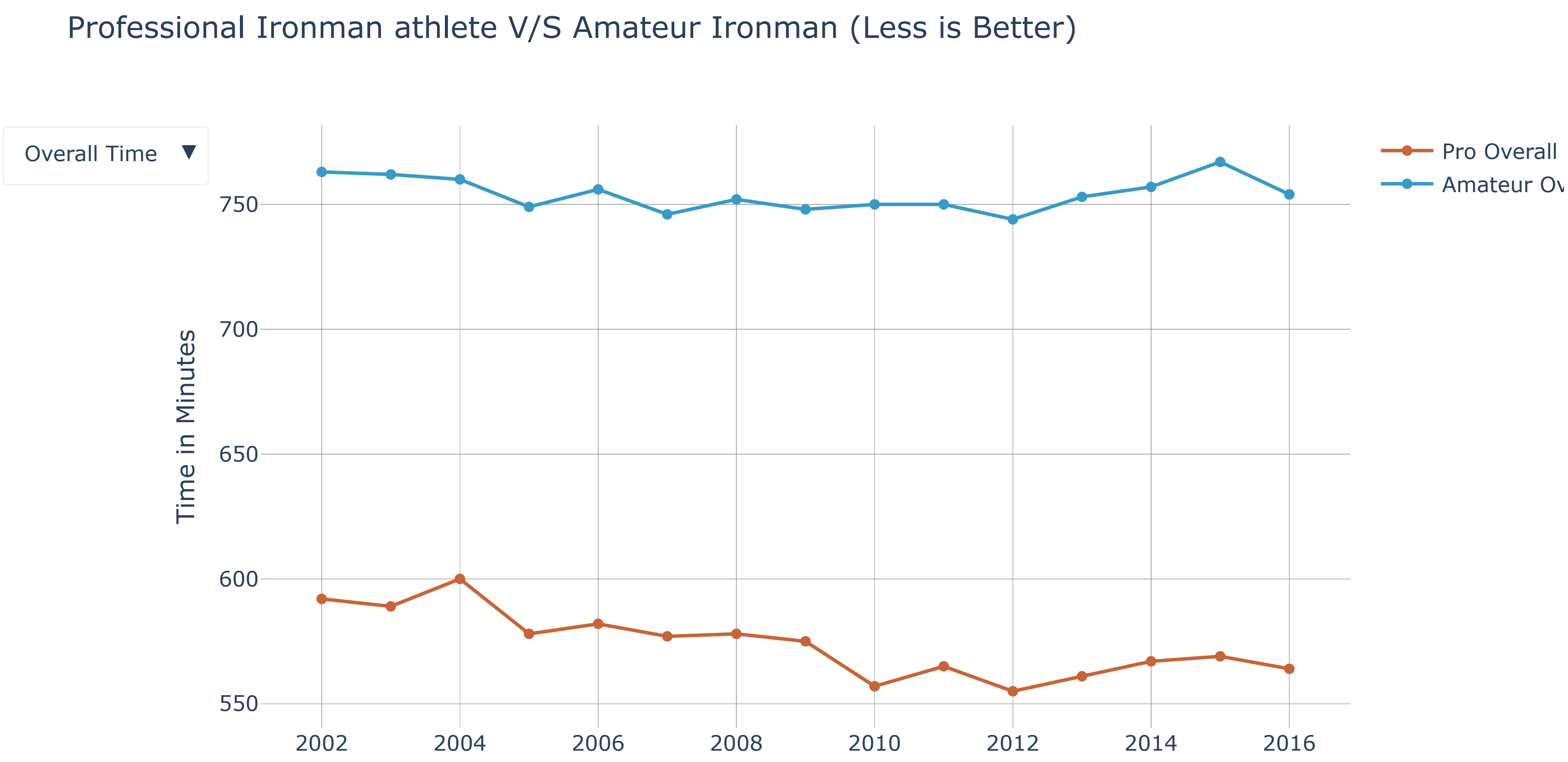
Since we have professional triathlon athletes competing with amateur competitors, we want to see the performance difference between them. For this we split finisherData into PRO and amateur Racers.

In [37]:
```
# categorize into pro and amateur
proRacers = (
    finisherData[finisherData["division"] == "PRO"]
    .groupby("Year")
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)
amateurRacers = (
    finisherData[finisherData["division"] != "PRO"]
    .groupby("Year")
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)
```

In [38]:
```
# prepare data
yearAmateurVsPro = proRacers.index.to_list()
swimTimePro = proRacers["swim"].to_list()
bikeTimePro = proRacers["bike"].to_list()
runTimePro = proRacers["run"].to_list()
overallTimePro = proRacers["overall"].to_list()

swimTimeNonPro = amateurRacers["swim"].to_list()
bikeTimeNonPro = amateurRacers["bike"].to_list()
runTimeNonPro = amateurRacers["run"].to_list()
overallTimeNonPro = amateurRacers["overall"].to_list()
```

In [39]:

```python
# create plotTemplates
line1 = PlotTemplate(yearAmateurVsPro, overallTimePro, "Pro Overall Time", "#C76538", visible=True)
line2 = PlotTemplate(
    yearAmateurVsPro, overallTimeNonPro, "Amateur Overall Time", "#389AC7", visible=True
)
line3 = PlotTemplate(yearAmateurVsPro, swimTimePro, "Pro Swim Time", "#79D629")
line4 = PlotTemplate(yearAmateurVsPro, swimTimeNonPro, "Amateur Swim Time", "#8629D6")
line5 = PlotTemplate(yearAmateurVsPro, bikeTimePro, "Pro Bike Time", "#52ACAD")
line6 = PlotTemplate(yearAmateurVsPro, bikeTimeNonPro, "Amateur Bike Time", "#AD5352")
line7 = PlotTemplate(yearAmateurVsPro, runTimePro, "Pro Run Time", "#DF20A0")
line8 = PlotTemplate(yearAmateurVsPro, runTimeNonPro, "Amateur Run Time", "#20DF5F")

# use plotTemplates in plotMaker
proVsAmateur = plotMaker(
    [line1, line2, line3, line4, line5, line6, line7, line8],
    "Professional Ironman athlete V/S Amateur Ironman (Less is Better)",
    "line",
    "Years",
    "Time in Minutes",
    ["Overall Time", "Swim Time", "Bike Time", "Run Time"],
    2,
)
iplot(proVsAmateur)
```



We notice that overall time, swim time, bike time and run time for professional athlete is always less than Amateur athlete over the years by a very significant margin.

[Back to table of contents](#)

4.5 What percentage of amateur racers overtook professional racers?

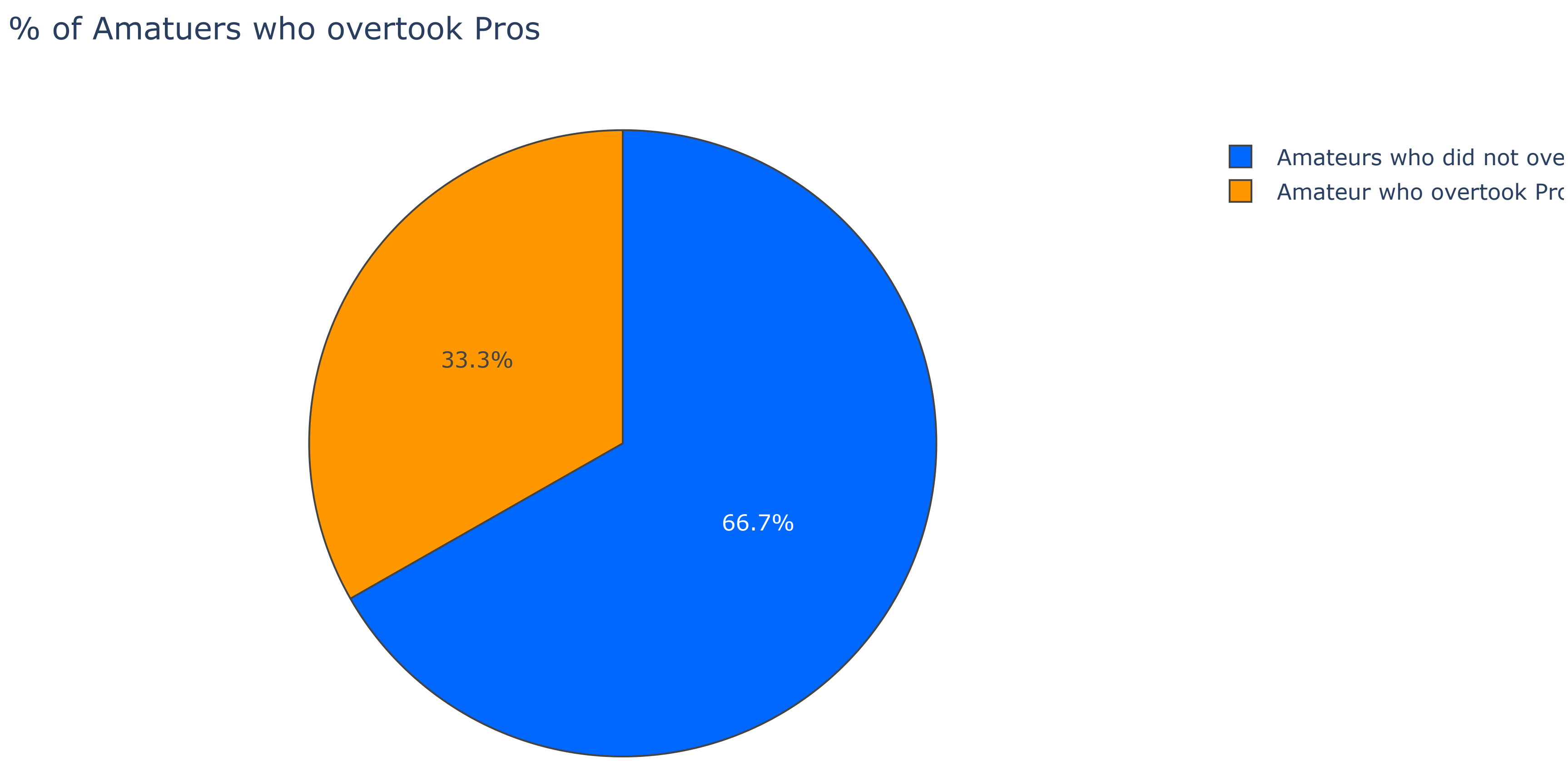
Although the average time over the years of a professional athlete is significantly less than an amateur we want to analyze, whether an amateur can overtake a professional and get a better overall rank. We compute the number of amateurs who got better overall rank than professional, for this we first drop rows which do not have overall rank value.

In [40]:
```
# remove nan overall ranks as they may skew the results
rankPresent = finisherData.dropna(subset=["overallRank"])
```

We then group the rankPresent data frame by Year and host location to get each individual race to compare overall rank of amateur and professional.

In [41]:
```
amateurOvertakeCount = 0
amateurTotalCount = 0

for name, individualRace in rankPresent.groupby(["Year", "Host location"]):

    # if in an individual race, if we have no professional, then skip that race
    if len(individualRace[individualRace["division"] == "PRO"]["overallRank"]) > 0:

        # get total amateur count
        amateurTotalCount += len(individualRace[individualRace["division"] != "PRO"])

        for amateurRank in individualRace[individualRace["division"] != "PRO"][
            "overallRank"
        ]:
            for proRank in individualRace[individualRace["division"] == "PRO"][
                "overallRank"
            ]:

                # if amateurRank less than proRank, then increment amateurOvertakeCount
                if int(amateurRank) < int(proRank):
                    amateurOvertakeCount += 1
                    break
```

In [42]:
```
# create template
pieOvertaker = PieTemplate(
    ["Amateur who overtook Pros", "Amateurs who did not overtake Pros"],
    [amateurOvertakeCount, amateurTotalCount - amateurOvertakeCount],
    ["#FF9800", "#0067FF"],
)

# use template in pieMaker
pieOvertakerFig = pieMaker([pieOvertaker], "% of Amatuers who overtook Pros")

iplot(pieOvertakerFig)
```



Even though on average amateurs do not perform as good as professionals, from our analysis we notice that **33%** the time, an amateur can overtake a pro in a race.

Back to table of contents

4.6 How many participants are from various divisions?

There are many age divisions in ironman competition, and there is also a separate division for PRO athletes. We want to analyze the number of participants from each division. Here we are only analyzing the participation count and not the participants finishing the competition.

In [43]:

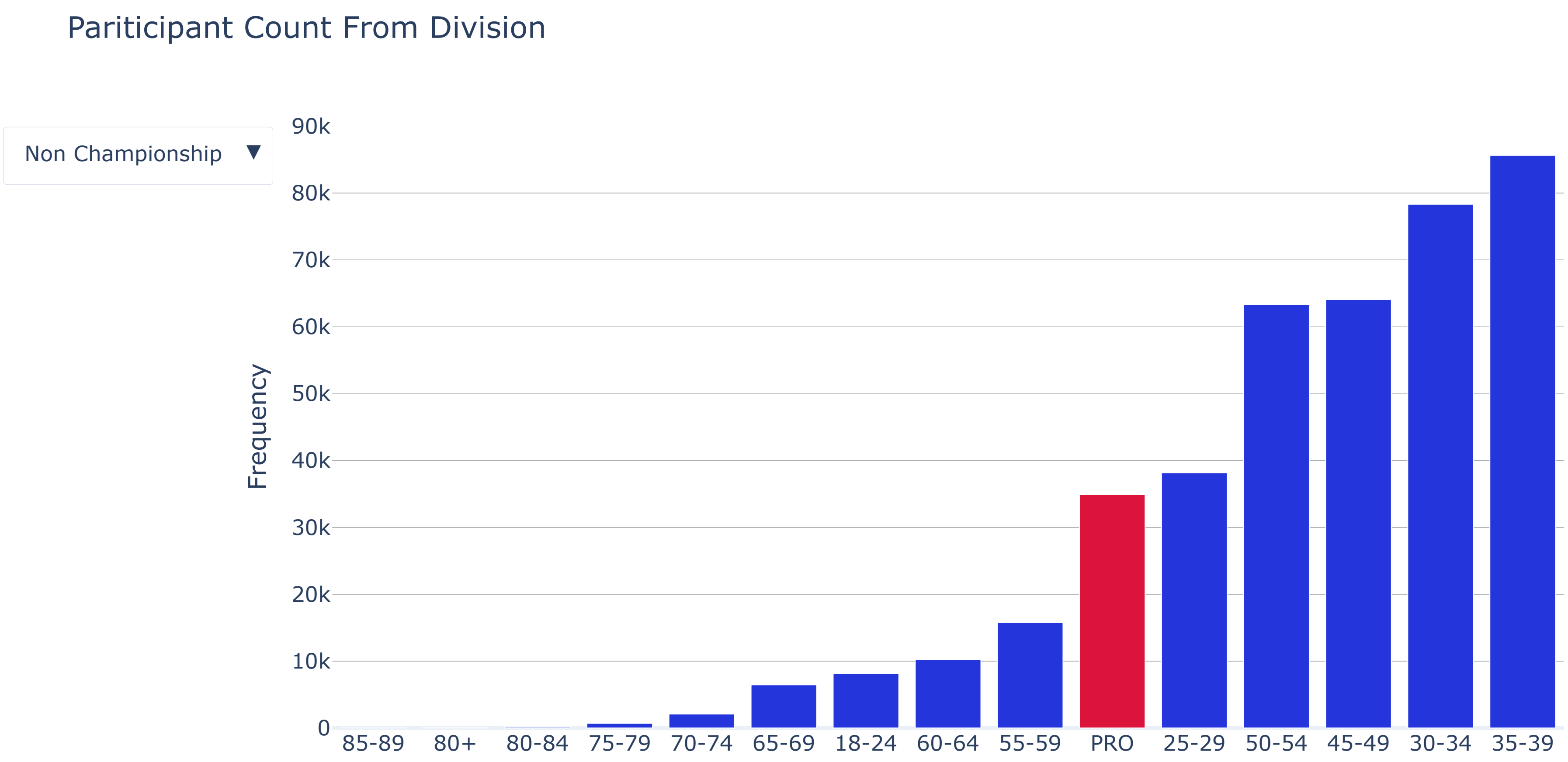
```
# prepare data
catergoryDf = (
    ironManRaw.groupby(["division", "Championship Competition"]) # group by divsion and Championship Competition
    .size()
    .reset_index(name="counts")
    .sort_values("counts")
)
catergoryDf
```

Out[43]:

	division	Championship Competition	counts
27	85-89	True	1
24	80+	False	1
25	80-84	False	22
26	80-84	True	24
23	75-79	True	120
22	75-79	False	160
21	70-74	True	299
19	65-69	True	538
20	70-74	False	687
1	18-24	True	847
17	60-64	True	919
15	55-59	True	1371
29	PRO	True	1579
3	25-29	True	1948
18	65-69	False	2082
13	50-54	True	2243
11	45-49	True	3278
5	30-34	True	3352
7	35-39	True	4114
9	40-44	True	4361
16	60-64	False	6452
28	PRO	False	8121
0	18-24	False	10241
14	55-59	False	15789
2	25-29	False	34910

	division	Championship Competition	counts
12	50-54	False	38166
10	45-49	False	63296
4	30-34	False	64077
6	35-39	False	78328
8	40-44	False	85638

In [44]:
```
# custom colorscale
customColor1 = ["#2436DB"] * len(catergoryDf["division"].unique())
customColor1[9] = "crimson"

# create template
plot1 = PlotTemplate(
    x=catergoryDf["division"].unique(),
    y=catergoryDf[~catergoryDf["Championship Competition"]]["counts"],
    name="Non Championship",
    color=customColor1,
    visible=True,
)

# custom colorscale
customColor2 = ["#DBC924"] * len(catergoryDf["division"].unique())
customColor2[9] = "crimson"

# create template
plot2 = PlotTemplate(
    x=catergoryDf["division"].unique(),
    y=catergoryDf[catergoryDf["Championship Competition"]]["counts"],
    name="Championship",
    color=customColor2,
)

# use plotMaker
participantFig = plotMaker(
    [plot1, plot2],
    "Pariticipant Count From Division",
    "bar",
    "Division",
    "Frequency",
    ["Non Championship", "Championship"],
    1,
)

iplot(participantFig)
```



We notice from above graph that most number of participants are from age division of 35-39 years, followed by 30-34 years. PRO athletes participating in race are 6th highest value and highlighted in red.

[Back to table of contents](#)

4.7 What is performance of a division?

We want to know how a particular division performs. we group the **finishedData** dataframe by divsion and then aggregate the columns using mean function.

In [45]:
```
# prepare data
divisionWise = (
    finisherData.groupby("division")
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)
divisionWise
```

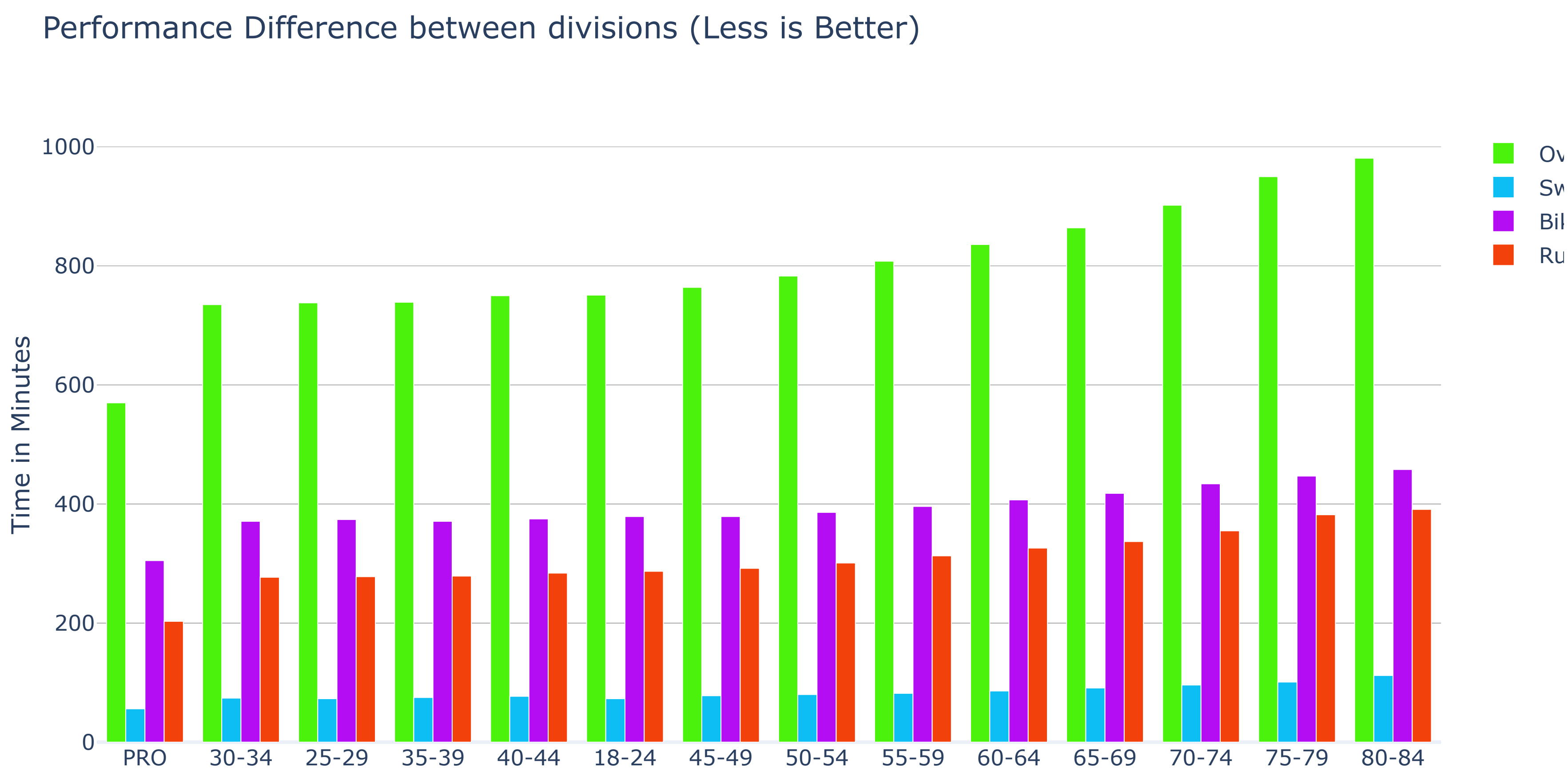
Out[45]:

division	swim	bike	run	overall
18-24	73.0	379.0	287.0	751.0
25-29	73.0	374.0	278.0	738.0
30-34	74.0	371.0	277.0	735.0
35-39	75.0	371.0	279.0	739.0
40-44	77.0	375.0	284.0	750.0
45-49	78.0	379.0	292.0	764.0
50-54	80.0	386.0	301.0	783.0
55-59	82.0	396.0	313.0	808.0
60-64	86.0	407.0	326.0	836.0
65-69	91.0	418.0	337.0	864.0
70-74	96.0	434.0	355.0	902.0
75-79	101.0	447.0	382.0	950.0
80-84	112.0	458.0	391.0	981.0
PRO	56.0	305.0	203.0	570.0

In [46]:

```
# create templates
plot1 = PlotTemplate(
    x=divisionWise.sort_values("overall").index.to_list(),
    y=divisionWise.sort_values("overall")["overall"].to_list(),
    name="Overall Time",
    color="#4BF30C",
    visible=True,
)

plot2 = PlotTemplate(
    x=divisionWise.sort_values("swim").index.to_list(),
    y=divisionWise.sort_values("swim")["swim"].to_list(),
    name="Swim Time",
    color="#0CBEF3",
    visible=True,
)

plot3 = PlotTemplate(
    x=divisionWise.sort_values("bike").index.to_list(),
    y=divisionWise.sort_values("bike")["bike"].to_list(),
    name="Bike Time",
    color="#B40CF3",
    visible=True,
)

plot4 = PlotTemplate(
    x=divisionWise.sort_values("run").index.to_list(),
    y=divisionWise.sort_values("run")["run"].to_list(),
    name="Run Time",
    color="#F3410C",
    visible=True,
)

# use plotMaker
figDivision = plotMaker(
    [plot1, plot2, plot3, plot4],
    "Performance Difference between divisions (Less is Better)",
    "bar",
    "Divisions",
    "Time in Minutes",
```

```
)
iplot(figDivision)
```



From above graph we conclude that PRO athletes are the best performing division, followed by 30-34 age division, followed by 25-29 age division. Age cannot be a determining factor alone for performance, since here older participants perform better than younger participants.

Also the mean age of PRO athletes is :

In [47]:
```
round(finisherData[finisherData["division"] == "PRO"]["age"].apply(float).mean(), 2)
```

Out[47]:
```
37.35
```

We now analyze the *performance of a division over the years* and compare it with other divisions. For this we group the finisherData by division and Year. After grouping operation we perform mean aggregation and then finally perform unstacking operations.

In [48]:
```
divisionYearWise = (
    finisherData.groupby(["division", "Year"])
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)
divisionYearWise = divisionYearWise.unstack().dropna()
# prepare data
divisionYearWise
```

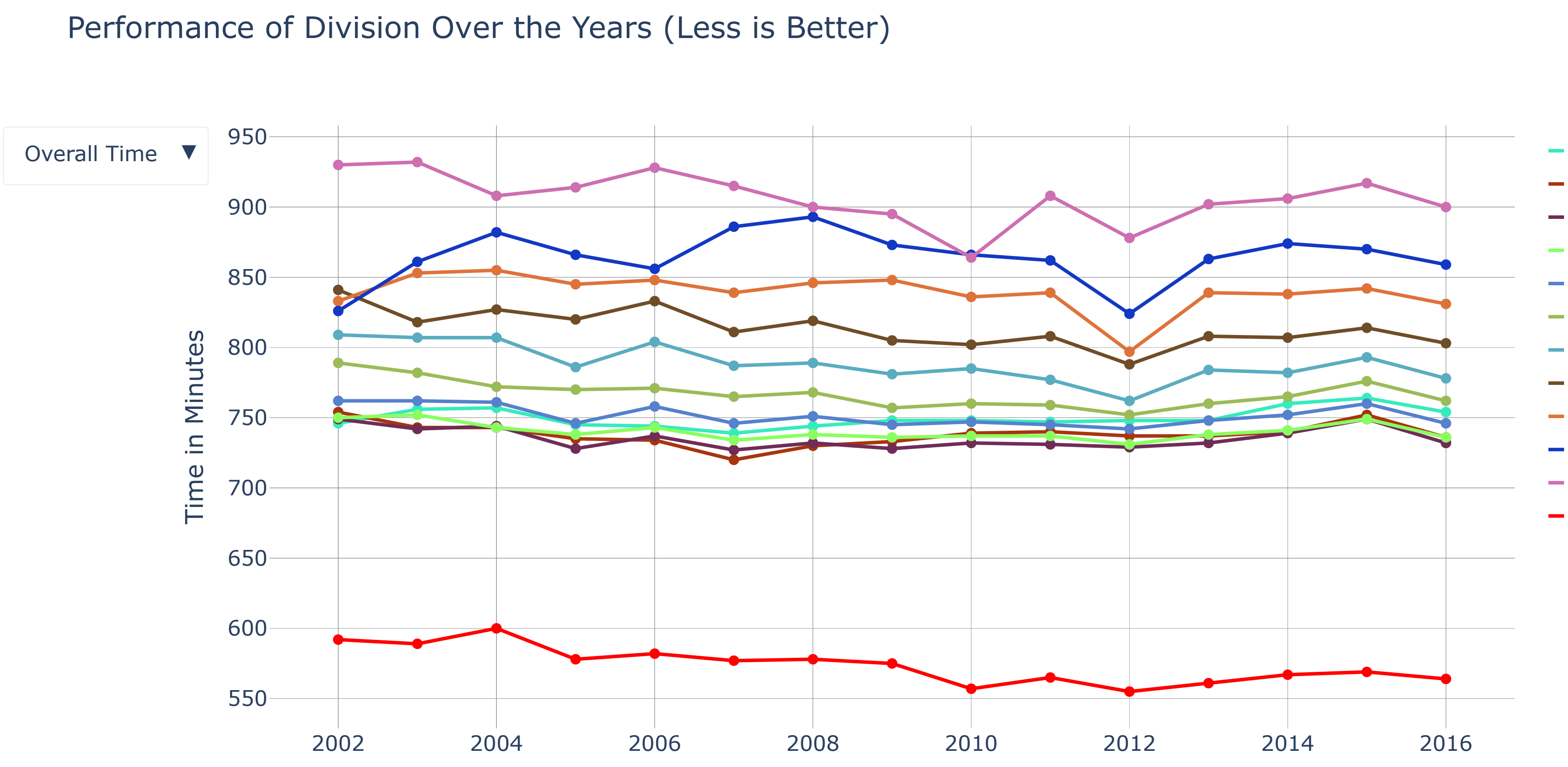
Out[48]:

	swim										...	overall									
Year	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
division																					
18-24	71.0	70.0	70.0	70.0	72.0	71.0	73.0	75.0	73.0	72.0	...	739.0	744.0	748.0	748.0	747.0	748.0	748.0	760.0	764.0	754.(
25-29	74.0	70.0	72.0	71.0	72.0	72.0	73.0	74.0	73.0	73.0	...	720.0	730.0	733.0	739.0	740.0	737.0	737.0	740.0	752.0	736.(
30-34	76.0	72.0	73.0	72.0	74.0	73.0	75.0	74.0	74.0	73.0	...	727.0	732.0	728.0	732.0	731.0	729.0	732.0	739.0	749.0	732.(
35-39	77.0	73.0	74.0	74.0	75.0	75.0	76.0	76.0	75.0	74.0	...	734.0	738.0	736.0	737.0	737.0	731.0	738.0	741.0	749.0	736.(
40-44	79.0	75.0	76.0	75.0	77.0	76.0	78.0	78.0	77.0	76.0	...	746.0	751.0	745.0	747.0	745.0	742.0	748.0	752.0	760.0	746.(
45-49	82.0	77.0	78.0	78.0	79.0	79.0	80.0	79.0	79.0	78.0	...	765.0	768.0	757.0	760.0	759.0	752.0	760.0	765.0	776.0	762.(
50-54	85.0	80.0	82.0	79.0	83.0	81.0	82.0	82.0	81.0	79.0	...	787.0	789.0	781.0	785.0	777.0	762.0	784.0	782.0	793.0	778.(
55-59	89.0	83.0	85.0	83.0	86.0	84.0	85.0	85.0	83.0	83.0	...	811.0	819.0	805.0	802.0	808.0	788.0	808.0	807.0	814.0	803.(
60-64	93.0	89.0	91.0	89.0	91.0	88.0	90.0	90.0	88.0	86.0	...	839.0	846.0	848.0	836.0	839.0	797.0	839.0	838.0	842.0	831.(
65-69	90.0	96.0	98.0	97.0	97.0	96.0	96.0	95.0	94.0	91.0	...	886.0	893.0	873.0	866.0	862.0	824.0	863.0	874.0	870.0	859.(
70-74	100.0	96.0	94.0	106.0	100.0	99.0	104.0	102.0	97.0	99.0	...	915.0	900.0	895.0	864.0	908.0	878.0	902.0	906.0	917.0	900.(
PRO	60.0	56.0	57.0	56.0	58.0	57.0	58.0	57.0	55.0	55.0	...	577.0	578.0	575.0	557.0	565.0	555.0	561.0	567.0	569.0	564.(

12 rows × 60 columns

In [49]:
```
divisionYear = []
metrics = ["overall", "swim", "bike", "run"]
firstFlag = True
#scale
colorScale = [
    "#36ebbd",
    "#a8350d",
    "#702c56",
    "#8afe60",
    "#5581cd",
    "#9bbb57",
    "#5bacc0",
    "#6f4d27",
    "#de733a",
    "#1338c3",
    "#cd6fb1",
    "red",
]

for metric in metrics:
    for index, eachCategory in enumerate(divisionYearWise[metric].index):
        divisionYear.append(
            # create plotTemplates
            PlotTemplate(
                x=divisionYearWise[metric].columns.to_list(),
                y=divisionYearWise[metric].loc[eachCategory].to_list(),
                name=eachCategory,
                color=colorScale[index],
                visible=True if firstFlag else False,
            )
        )

    firstFlag = False

figDivisionYear = plotMaker(
    divisionYear,
    "Performance of Division Over the Years (Less is Better)",
    "line",
    "Years",
    "Time in Minutes",
    ["Overall Time", "Swim Time", "Bike Time", "Run Time"],
```

```
    len(divisionYearWise["swim"]),
)

iplot(figDivisionYear)
```



We notice that PRO division consistently performs better than other divisions, also overall, swim, bike and run times are slightly decreasing over the years across all categories.

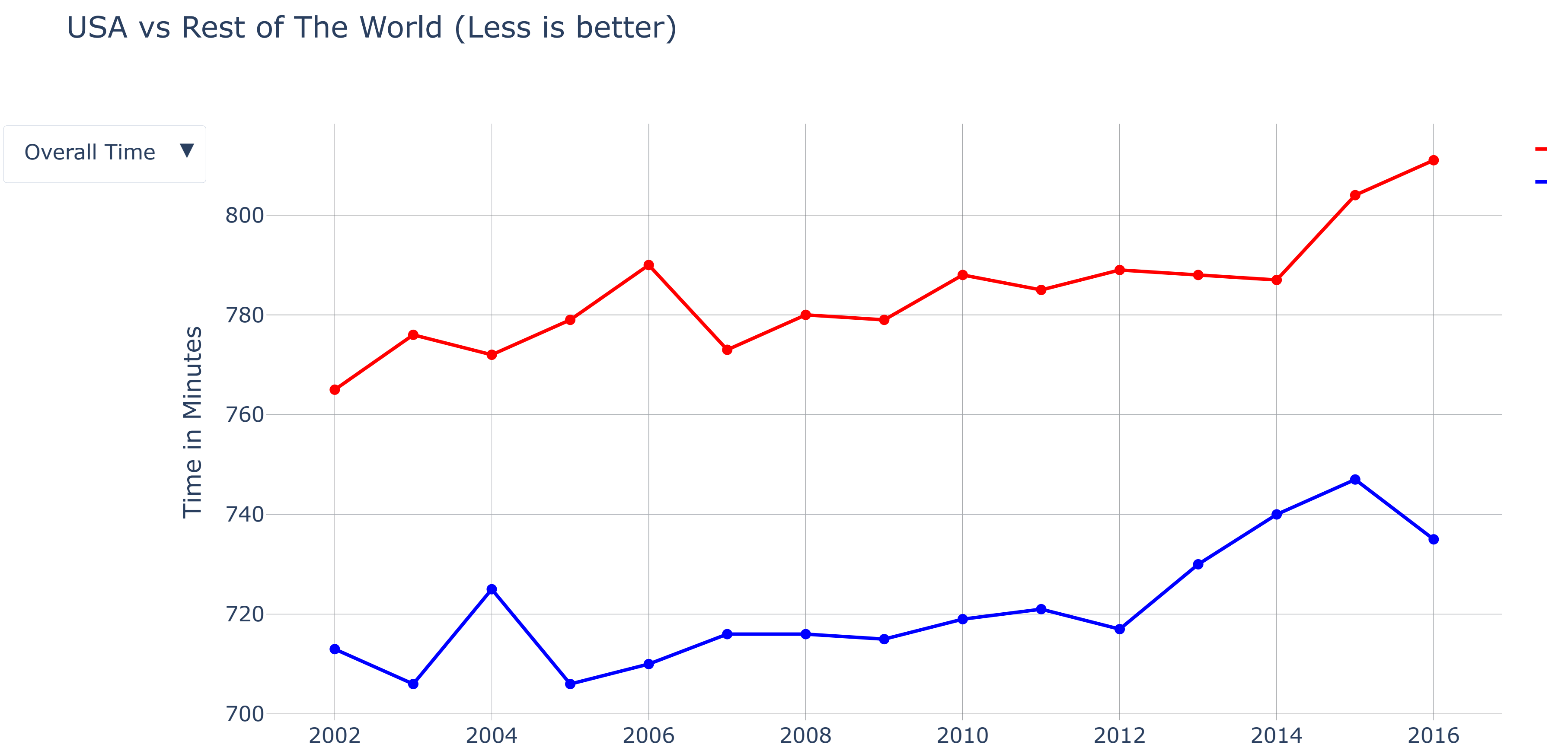
[Back to table of contents](#)

4.8 How does USA perform in comparison to the rest of the world?

We now compare the overall performance time of USA with rest of the world. We create 2 different dataframes, one which contains information about USA and other which contains information for rest of the world.

In [50]:
```
USA = (
    finisherData[finisherData["country"] == "USA"]
    .groupby(["Year"])
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)

world = (
    finisherData[finisherData["country"] != "USA"]
    .groupby(["Year"])
    .aggregate({"swim": "mean", "bike": "mean", "run": "mean", "overall": "mean"})
    .round(0)
)
```

In [51]:
```
USAvsOtherArr = []

metrics = ["overall", "swim", "bike", "run"]
firstFlag = True

for metric in metrics:
    USAvsOtherArr.append(
        PlotTemplate(
            x=USA.index.to_list(),
            y=USA[metric].to_list(),
            name="USA",
            color="red",
            visible=True if firstFlag else False,
        )
    )
    USAvsOtherArr.append(
        PlotTemplate(
            x=world.index.to_list(),
            y=world[metric].to_list(),
            name="World",
            color="blue",
            visible=True if firstFlag else False,
        )
    )
    firstFlag = False

figUSAvsOther = plotMaker(
    USAvsOtherArr,
    "USA vs Rest of The World (Less is better)",
    "line",
    "Years",
    "Time in Minutes",
    ["Overall Time", "Swim Time", "Bike Time", "Run Time"],
    2,
)

iplot(figUSAvsOther)
```



From above charts we infer that USA has higher overall time, swim time, bike time and run time in comparison with rest of the world.

[Back to table of contents]

4.9 Which countries participate the most?

We want to know from which countries participate the most in ironman competition. We already have the first 3 letter country name, we then use **pycountry** library to get full country name and then use that to create choropleth hover text. We group by country and find how many participants from each country by aggregating using the size function which will return number of rows per group.

In [52]:
```
worldParticipationChoropleth = (
    ironManRaw.groupby(["country"]).size().reset_index(name="Participants")
)
```

In [53]:
```
def getFullCountryName(alpha_3Name):
    # use this library to get full country name
    country = pycountry.countries.get(alpha_3=alpha_3Name)
    return country.name if country != None else None
```

In [54]:
```
# applying function on 3 alpha country code
worldParticipationChoropleth["Full Country Name"] = worldParticipationChoropleth[
    "country"
].apply(getFullCountryName)
```

The result of above operations are shown below::

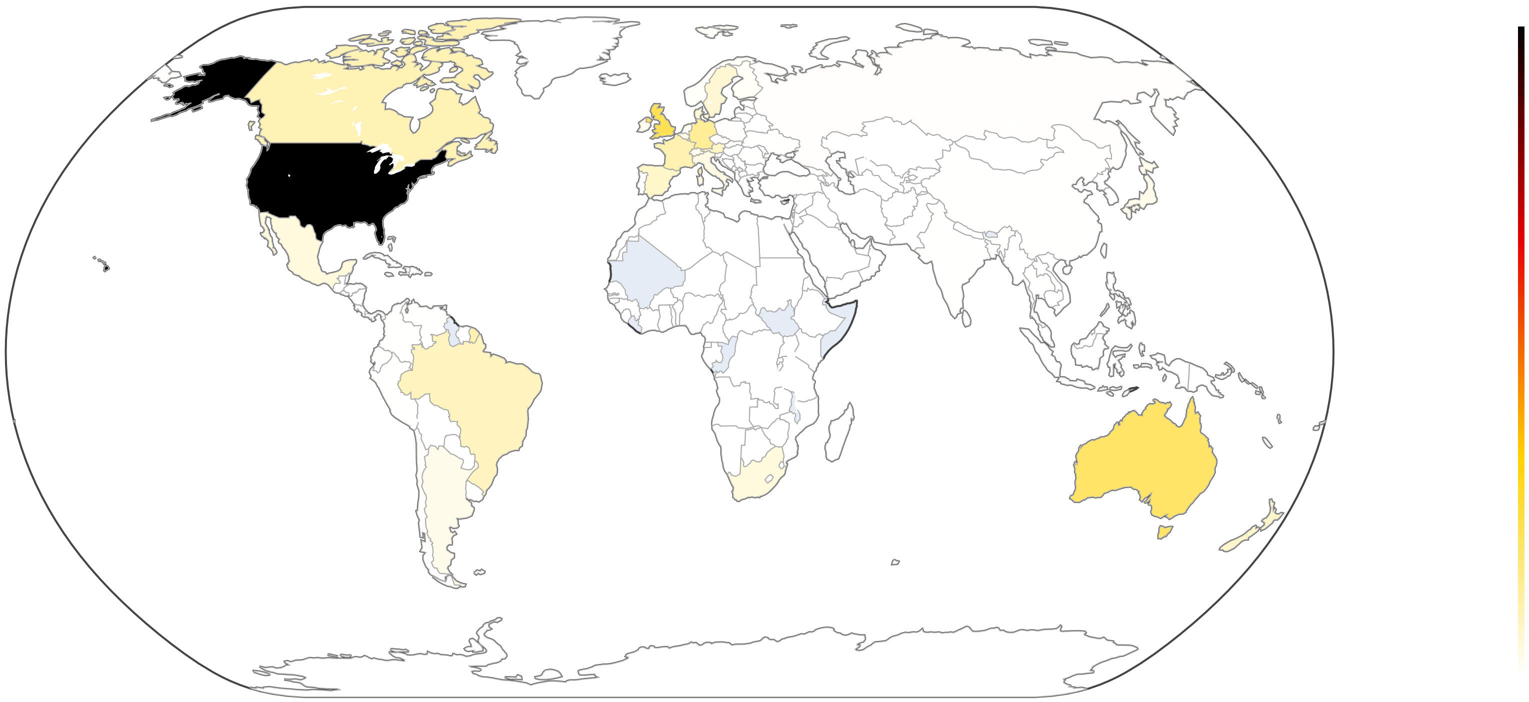
In [55]:
```
worldParticipationChoropleth.head()
```

Out[55]:

	country	Participants	Full Country Name
0	ABW	37	Aruba
1	AFG	8	Afghanistan
2	AGO	22	Angola
3	ALA	1	Åland Islands
4	ALB	14	Albania

In [56]:
```
# use choroplethMaker
participationFig = choroplethMaker(
    locationsToShow=worldParticipationChoropleth["country"],
    textOnHover=worldParticipationChoropleth["Full Country Name"],
    colorIntensityFactor=worldParticipationChoropleth["Participants"],
    colorTheme="hot",
    reverseColorTheme=True,
    titleForPlot="Participation by each Country",
)

iplot(participationFig)
```

Participation by each Country



From choropleth we observe the highest participating countries are USA, Canada, Brazil, Australia, United Kingdom, Spain, France, Germany, Argentina and South Africa.

Back to table of contents

4.10 Which countries perform the best in terms of overall time?

We want to know from which countries perform the best in ironman competition. We already have the first 3 letter country name, we then use **pycountry** library to get full country name and then use that to create choropleth hover text. We group by country and aggregate overall time using the mean function.

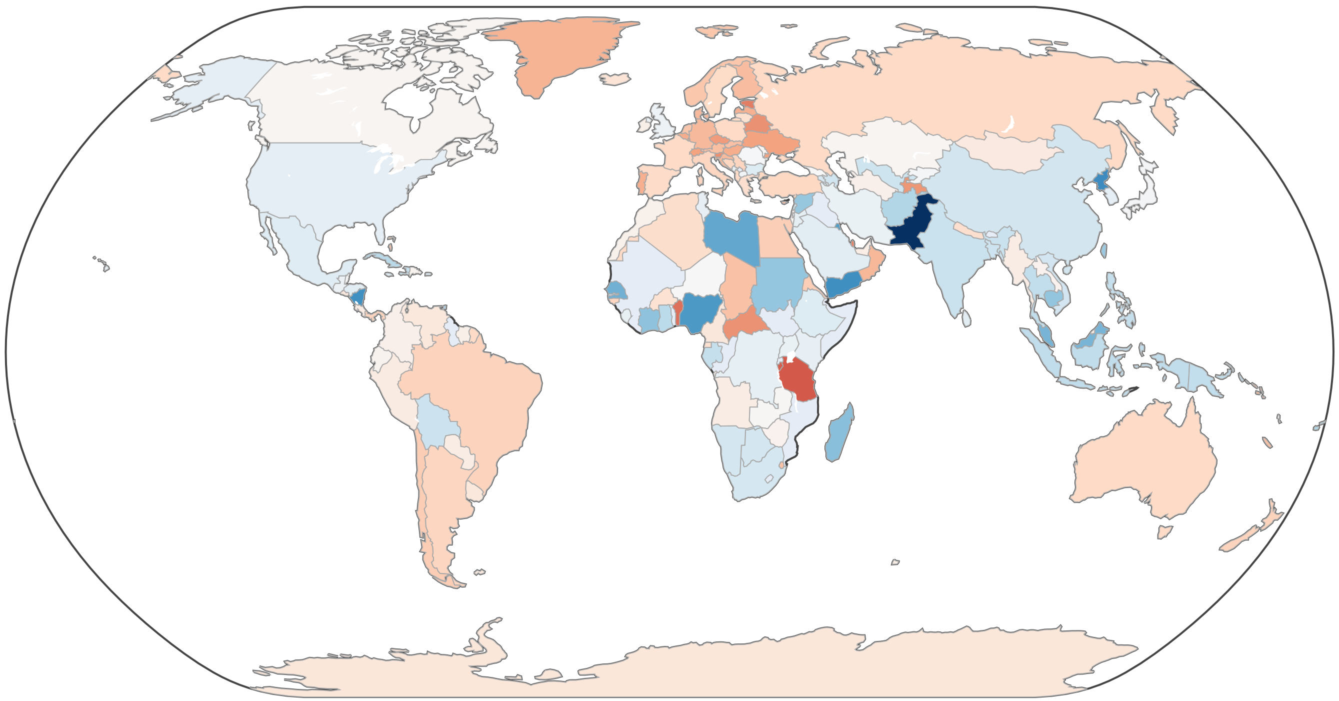
In [57]:
```
worldPerformanceChoropleth = (
    finisherData.groupby(["country"])
    .aggregate({"overall": "mean"})
    .round(0)
    .reset_index()
)
```

In [58]:
```
worldPerformanceChoropleth["Full Country Name"] = worldPerformanceChoropleth[
    "country"
].apply(getFullCountryName)
```

In [59]:

```
# use choroplethMaker
performanceFig = choroplethMaker(
    locationsToShow=worldPerformanceChoropleth["country"],
    textOnHover=worldPerformanceChoropleth["Full Country Name"],
    colorIntensityFactor=worldPerformanceChoropleth["overall"],
    colorTheme="rdbu",
    reverseColorTheme=False,
    titleForPlot="Overall Time in Minutes by each Country (Less is Better)",
)
iplot(performanceFig)
```

Overall Time in Minutes by each Country (Less is Better)



From choropleth the highest performing countries are colored in red which are Tanzania, Central Africa Republic, Greenland and most of the European countries like Germany, Ukraine, Belarus, Switzerland.

[Back to table of contents]

5. Conclusion

In conclusion, about 80% of participants complete the Ironman competition and in recent years this number is strongly growing. The overall completion time of the race is also decreasing and participants perform better in championship races than in non-championship races.

The PRO athlete performs better than an amateur athlete on average, but there is still a 33% chance that an amateur can overtake a professional in a race.

The most number of participants are from the 35-39-year-old age division, followed by 30-34-year-old age division. Other than the PRO division, the 30-34 year age division performs the best with the lowest race times. Thus an individual's age should not restrict oneself from participating in ironman.

Countries like the USA, Canada, Brazil, Australia, United Kingdom, Germany participate in higher numbers. Countries like Tanzania, Greenland, Australia, Germany, the United Kingdom have better overall race times.

Back to table of contents