In [1]:
```
# Import packages
import pandas as pd
import plotly.express as px
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import seaborn as sns
```

Exploratory analysis

In [2]:
```
Fall2019 = pd.read_csv('RecCen2019.csv')
```

In [3]:
```
Fall2022 = pd.read_csv('RecCen_Fall2022 .csv')
```

In [4]:
```
Fall2022.head()
```

Out[4]:

	Date	Time	Visits	Day
0	9/18/22	Time	0	Sunday
1	9/18/2022	6:00 a.m.	0	Sunday
2	9/18/2022	7:00 a.m.	0	Sunday
3	9/18/2022	8:00 a.m.	0	Sunday
4	9/18/2022	9:00 a.m.	0	Sunday

In [5]:
```
# Remove the first row
Fall2022 = Fall2022.tail(-1)
```

In [6]:
```
Fall2022.head()
```

Out[6]:

	Date	Time	Visits	Day
1	9/18/2022	6:00 a.m.	0	Sunday
2	9/18/2022	7:00 a.m.	0	Sunday
3	9/18/2022	8:00 a.m.	0	Sunday
4	9/18/2022	9:00 a.m.	0	Sunday
5	9/18/2022	10:00 a.m.	0	Sunday

In [7]:
```
Fall2019.head()
```

Out[7]:

	Date	Time	Visits	Day
0	9/29/2019	5:00 a.m.	0	Sunday
1	9/29/2019	6:00 a.m.	0	Sunday
2	9/29/2019	7:00 a.m.	0	Sunday
3	9/29/2019	8:00 a.m.	4	Sunday
4	9/29/2019	9:00 a.m.	225	Sunday

In [8]:
```
Fall2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1428 entries, 1 to 1428
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Date    1428 non-null   object
 1   Time    1428 non-null   object
 2   Visits  1428 non-null   int64 
 3   Day     1428 non-null   object
dtypes: int64(1), object(3)
memory usage: 44.8+ KB
```

In [9]:
```
Fall2019.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1386 entries, 0 to 1385
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Date        1386 non-null   object
 1   Time        1386 non-null   object
 2   Visits      1386 non-null   int64 
 3   Day         1386 non-null   object
dtypes: int64(1), object(3)
memory usage: 43.4+ KB
```

In [10]:
```
Fall2022.columns
```

Out[10]:
```
Index(['Date', 'Time', 'Visits', 'Day'], dtype='object')
```

In [11]:
```
Fall2019.columns
```

Out[11]:
```
Index(['Date      ', 'Time      ', 'Visits ', 'Day'], dtype='object')
```

In [12]:
```
# Rename columns so there is no weird spaces after
Fall2019 = Fall2019.rename(columns = {'Visits ': 'Visits', 'Date      ': 'Dat
```

In [13]:
```
Fall2022.isnull().sum()
```

Out[13]:
```
Date       0
Time       0
Visits     0
Day        0
dtype: int64
```

In [14]:
```
Fall2019.isnull().sum()
```

Out[14]:
```
Date       0
Time       0
Visits     0
Day        0
dtype: int64
```

Take out all the values where there is 0 visits, this accounts for how the hours are different for weekdays compared to weekends. For example 6-8 AM on sunday all have 0 visits even though the rec center is not open during that time which would throw off the mean since its counted as an (n) value but it adds 0 visits, also accounts for holidays where rec center is not open

In [15]:
```
Fall2022 = Fall2022[Fall2022.Visits != 0]
Fall2019 = Fall2019[Fall2019.Visits != 0]
```

In [16]:
```
Fall2022.groupby(['Day']).sum()
```

Out[16]:

Day	Visits
Friday	22334
Monday	36363
Saturday	17219
Sunday	20664
Thursday	29676
Tuesday	35830
Wednesday	32497

In [17]:
```
Fall2019.groupby(['Day']).sum()
```

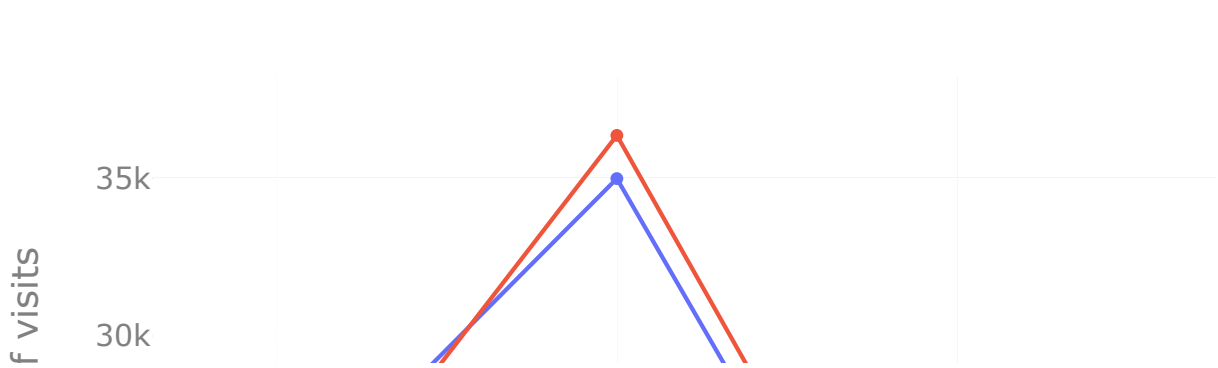
Out[17]:

Day	Visits
Friday	24129
Monday	34993
Saturday	16293
Sunday	17745
Thursday	30478
Tuesday	36794
Wednesday	32876

Comparison between Fall 2019 and Fall 2022

In [18]:
```
# Make days in right order
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturd

# For 2019
Fall2019 = Fall2019.reset_index()
Fall2019 = Fall2019.set_index('Day').loc[day_order]
Fall2019 = Fall2019.reset_index()
# Same for 2022
Fall2022 = Fall2022.reset_index()
Fall2022 = Fall2022.set_index('Day').loc[day_order]
Fall2022 = Fall2022.reset_index()
# Group by day and get sums
days2019 = Fall2019.groupby(['Day']).sum()
days2022 = Fall2022.groupby(['Day']).sum()

# Make plots
fig = go.Line(x = Fall2022.Day.unique() , y = days2022['Visits'], name = 'Fa
fig2 = go.Line(x = Fall2019.Day.unique(), y = days2019['Visits'], name = 'Fa

# Overlay them
data = [fig2,fig]
layout = go.Layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0
fig3 = go.Figure(data = data, layout = layout)

# Add titles
fig3.update_layout(
    title="Comparison between Fall 2019 and Fall 2022 by day",
    xaxis_title="Day",
    yaxis_title="Total amount of visits",
    legend_title="Year",
    font=dict(
        size=14,
        color="Grey"
    )
)

# Show
fig3.show()
```

```
/Users/jakejensema/opt/anaconda3/lib/python3.9/site-packages/plotly/graph_ob
js/_deprecations.py:378: DeprecationWarning: plotly.graph_objs.Line is depre
cated.
Please replace it with one of the following more specific types
  - plotly.graph_objs.scatter.Line
  - plotly.graph_objs.layout.shape.Line
  - etc.

  warnings.warn(
```

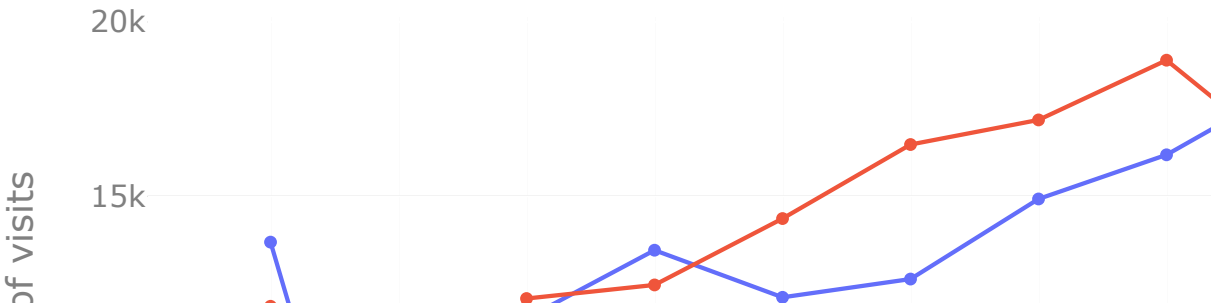


We can see the trend is identical for both 2019 and 2022, what about for by time instead of day?

In [19]:

```
# Sort so times are in order
time_order = ['6:00 a.m.', '7:00 a.m.', '8:00 a.m.', '9:00 a.m.', '10:00 a.m
              '1:00 p.m.', '2:00 p.m.', '3:00 p.m.', '4:00 p.m.', '5:00 p.m.'
              '9:00 p.m.', '10:00 p.m.']

# For 2019
Fall2019 = Fall2019.reset_index()
Fall2019 = Fall2019.set_index('Time').loc[time_order]
Fall2019 = Fall2019.reset_index()
# Same for 2022
Fall2022 = Fall2022.reset_index()
Fall2022 = Fall2022.set_index('Time').loc[time_order]
Fall2022 = Fall2022.reset_index()

# Make overlaying plot

# Group by time to and get sum
times2019 = Fall2019.groupby(['Time']).sum()
times2022 = Fall2022.groupby(['Time']).sum()

# Make 2 plots one for 2019 and one for 2022
fig_1 = go.Line(x = Fall2022.Time.unique() , y = times2022['Visits'], name =
fig_2 = go.Line(x = Fall2019.Time.unique(), y = times2019['Visits'], name =

# Overlay them
data = [fig_2,fig_1]
layout = go.Layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0
fig = go.Figure(data = data, layout = layout)

# Add titles
fig.update_layout(
    title="Comparison between Fall 2019 and Fall 2022 by day",
    xaxis_title="Time",
    yaxis_title="Total amount of visits",
    legend_title="Year",
    font=dict(
        size=14,
        color="Grey"
    )
)

# Show
fig.show()
```

```
/Users/jakejensema/opt/anaconda3/lib/python3.9/site-packages/plotly/graph_ob
js/_deprecations.py:378: DeprecationWarning:

plotly.graph_objs.Line is deprecated.
Please replace it with one of the following more specific types
  - plotly.graph_objs.scatter.Line
  - plotly.graph_objs.layout.shape.Line
  - etc.
```

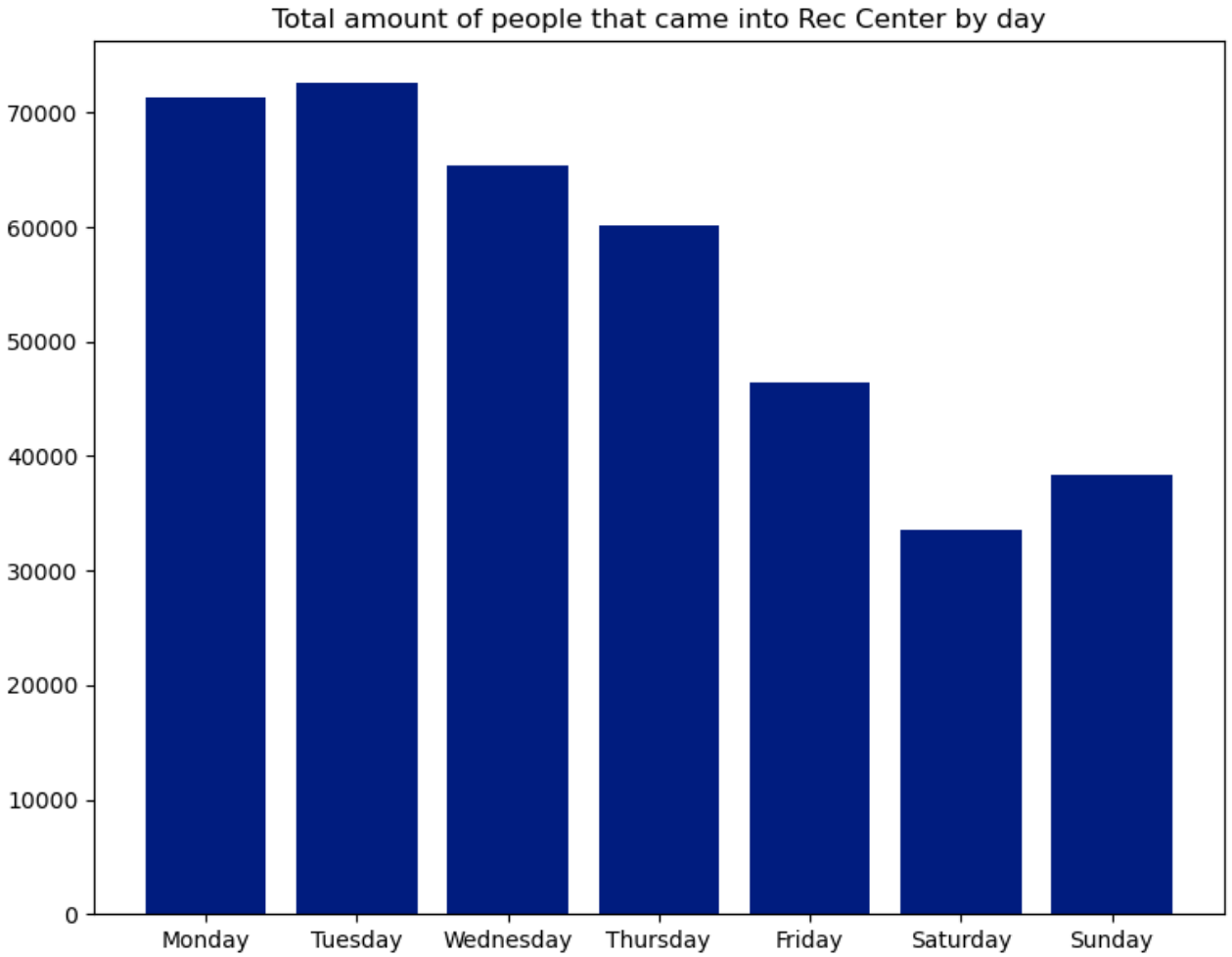


We can see a very similar pattern for Fall 2019 and Fall 2022 so it would not hurt to combind the data so we have more data to use. Which will help improving the accuracy of our data

In [20]:
```
frames = [Fall2019, Fall2022]

df_new = pd.concat(frames)
```

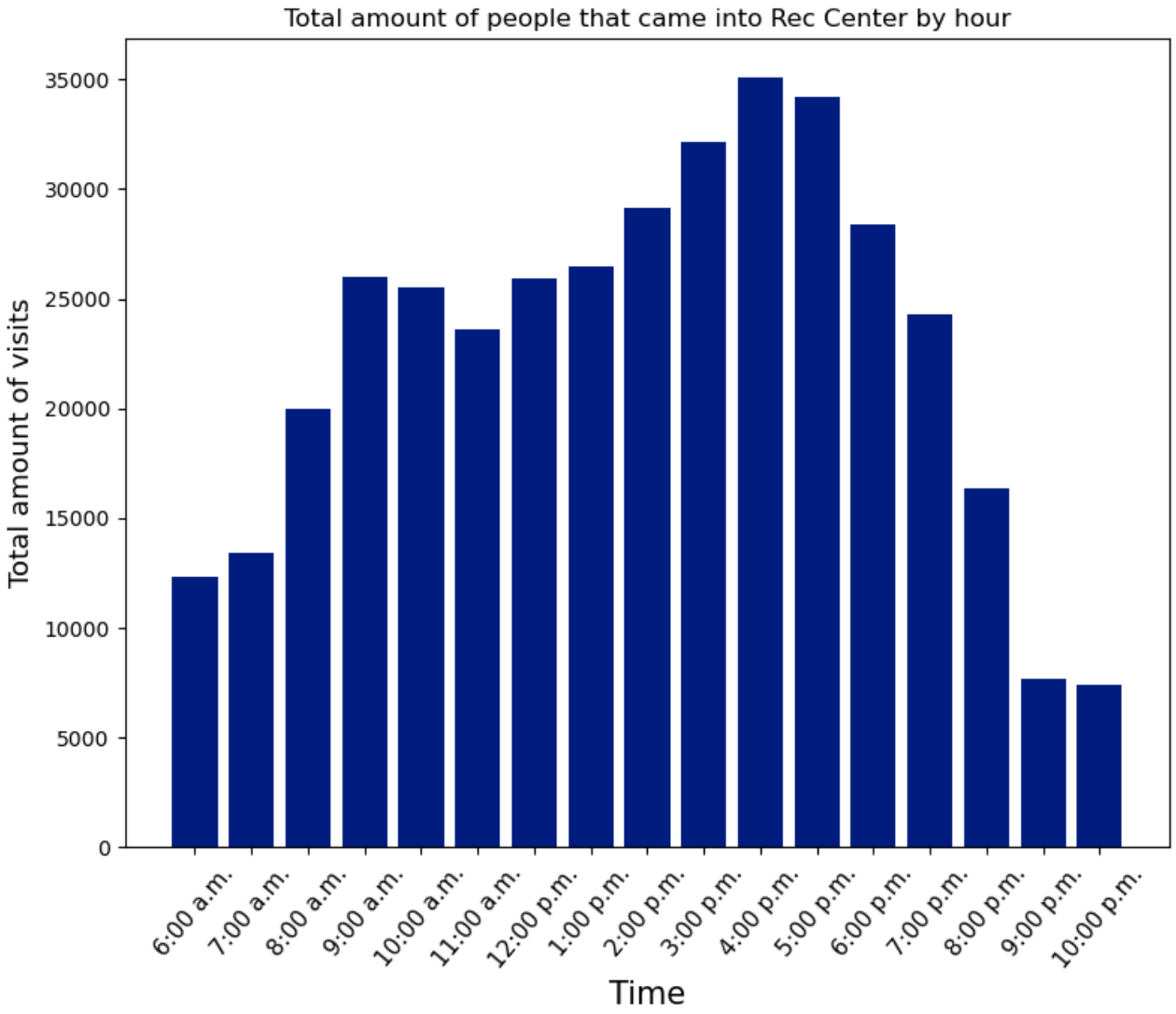
In [21]:

```
plt.style.use('seaborn-dark-palette')
# Take out 5 AM from the dataset because the rec center is not open at 5 AM
df_new = df_new.loc[df_new['Time'] != '5:00 a.m.']

# Get total amount of visits by day
df_day = pd.DataFrame(df_new.groupby('Day').sum())
df_day = df_day.reset_index()

# Set days in order for index
day_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturd

# Set inde=x
df_day = df_day.set_index('Day').loc[day_order]
df_day = df_day.reset_index()

# Make fig
fig = plt.figure(figsize=(9,7))
plt.bar(df_day['Day'], df_day['Visits'])
plt.title('Total amount of people that came into Rec Center by day')

# Show
plt.show()
```



In [22]:

```
# Get data sorted
plt.style.use('seaborn-dark-palette')

# Take out 5 AM from the dataset because the rec center is not open at 5 AM

Visits = df_new.groupby('Time').mean()

# Get total amount of visits by time
df_time = pd.DataFrame(df_new.groupby('Time').sum())
df_time = df_time.reset_index()

# Set new index
df_time = df_time.set_index('Time').loc[time_order]
df_time = df_time.reset_index()

# Make fig
plt.figure(figsize=(9,7))
plt.bar(df_time['Time'], df_time['Visits'])
plt.xlabel('Time', fontsize = 15)
plt.ylabel('Total amount of visits', fontsize = 13)
plt.xticks(fontsize = 11, rotation = 50)
plt.title('Total amount of people that came into Rec Center by hour')

# Show
plt.show()
```



Charts for article

Chart displaying the best time to go to the rec cen

In [23]:
```
# Gradient Color Bar Plots
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
from matplotlib import colors as mcolors, path
```

In [24]:
```
# Get data in correct form to be plotted
df_time_mean = pd.DataFrame(df_new.groupby('Time').mean())
df_time_mean = df_time_mean.reset_index()

df_time_mean = df_time_mean.set_index('Time').loc[time_order]
df_time_mean = df_time_mean.reset_index()

# Plot
fig, ax = plt.subplots()
bars = ax.bar(df_time_mean['Time'], df_time_mean['Visits'], color = 'blue')
```

```
# Make plot Title/Tick marks
plt.title('Average amount of people coming into rec cen per hour', fontsize
plt.xticks(fontsize = 10, rotation = 70)
plt.yticks(fontsize = 10,)

# Change style of graph
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['bottom'].set_color('#DDDDDD')
ax.tick_params(bottom=False, left=False)
ax.set_axisbelow(True)
ax.yaxis.grid(True, color='w', linestyle = '--')
ax.xaxis.grid(False)

# Add labels to bars for exact number
bar_color = bars[0].get_facecolor()

for bar in bars:
  ax.text(
      bar.get_x() + bar.get_width() / 2,
      bar.get_height() + 3,
      int(round(bar.get_height(),1)),
      horizontalalignment='center',
      color='black',
      weight='heavy',
      fontsize = 8.3
  )

# Apply gradient for bars
def gradientbars(bars,ydata,cmap):
    ax = bars[0].axes
    lim = ax.get_xlim()+ax.get_ylim()
    ax.axis(lim)
    for bar in bars:
        bar.set_facecolor("none")
        x,y = bar.get_xy()
        w, h = bar.get_width(), bar.get_height()
        grad = np.atleast_2d(np.linspace(0,1*h/max(ydata),256)).T
        ax.imshow(grad, extent=[x,x+w,y,y+h], origin='lower', aspect="auto",
                  norm=cm.colors.NoNorm(vmin=0,vmax=1), cmap=plt.get_cmap(cm

# Pass chart through function
gradientbars(bars, df_time_mean['Visits'], 'cool')

# Show
plt.tight_layout()
```

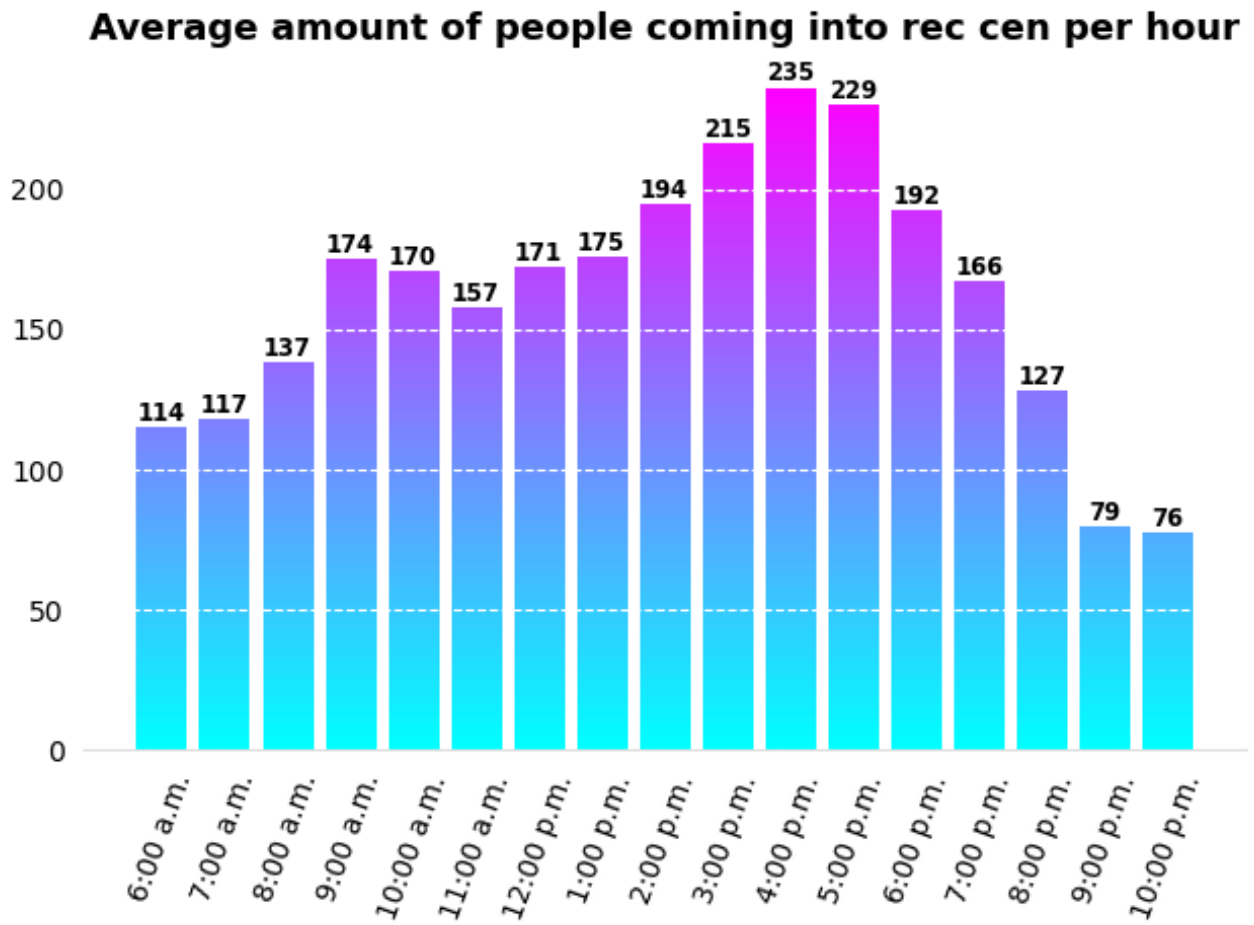
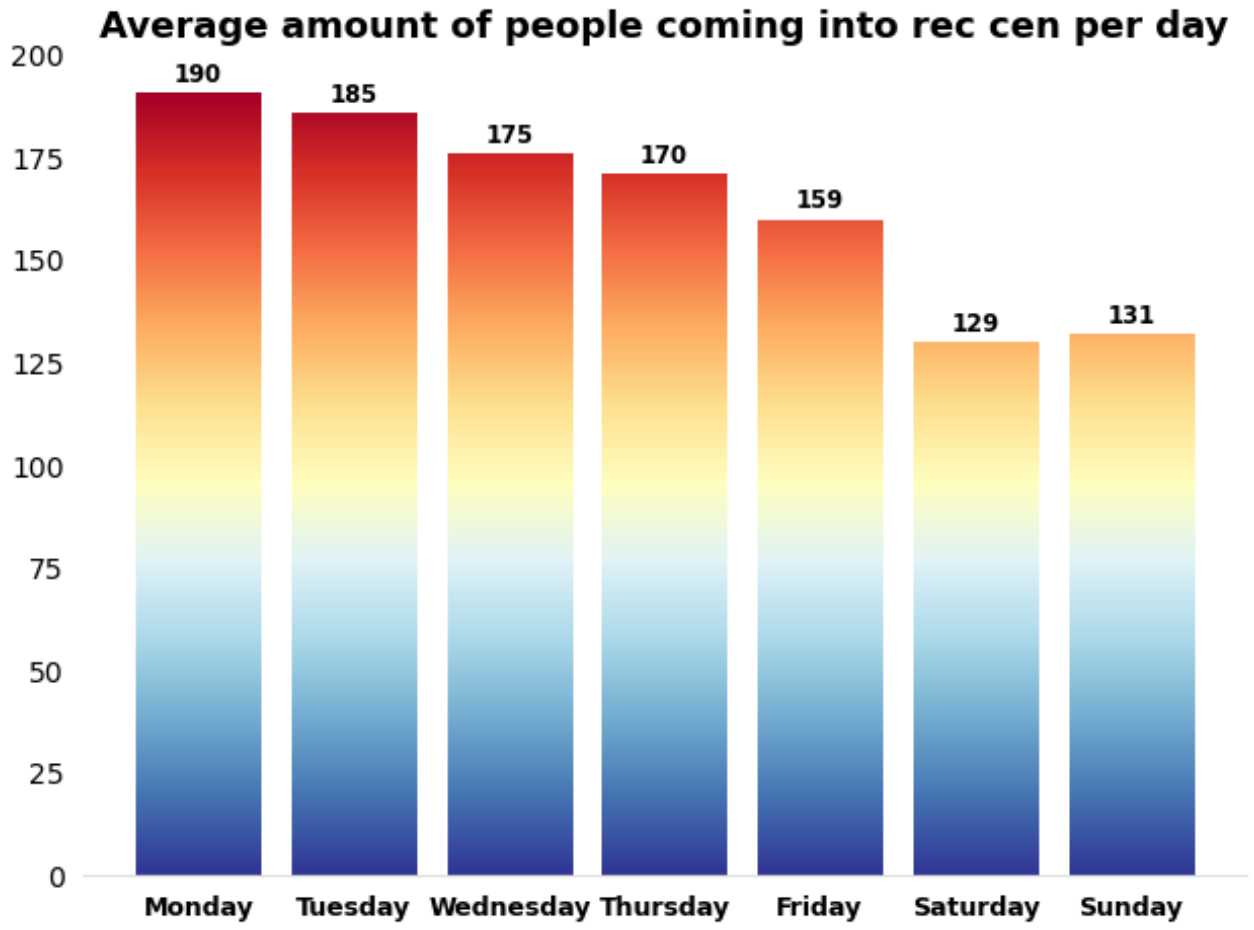


Chart showing most popular days

In [25]:
```
# Get data in correct form to be plotted
Visits = df_new.groupby('Day').mean()

df_day_mean = pd.DataFrame(df_new.groupby('Day').mean())
df_day_mean = df_day_mean.reset_index()

day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturd

df_day_mean = df_day_mean.set_index('Day').loc[day_order]
df_day_mean = df_day_mean.reset_index()

# Plot
fig, ax = plt.subplots()
bars = ax.bar(df_day_mean['Day'], df_day_mean['Visits'], color = 'blue')

# Make plot Title/Tick marks
plt.title('Average amount of people coming into rec cen per day', fontsize =
plt.xticks(fontsize = 9, weight = 'bold')
plt.yticks(fontsize = 10,)

# Change style of graph
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
```

```
ax.spines['bottom'].set_color('#DDDDDD')
ax.tick_params(bottom=False, left=False)
ax.set_axisbelow(True)
ax.xaxis.grid(False)

# Add labels to bars for exact number
bar_color = bars[0].get_facecolor()

for bar in bars:
  ax.text(
      bar.get_x() + bar.get_width() / 2,
      bar.get_height() + 3,
      int(round(bar.get_height(),1)),
      horizontalalignment='center',
      color='black',
      weight='heavy',
      fontsize = 8.3
  )

def gradientbars(bars,ydata,cmap):
    ax = bars[0].axes
    lim = ax.get_xlim()+ax.get_ylim()
    ax.axis(lim)
    for bar in bars:
        bar.set_facecolor("none")
        x,y = bar.get_xy()
        w, h = bar.get_width(), bar.get_height()
        grad = np.atleast_2d(np.linspace(0,1*h/max(ydata),256)).T
        ax.imshow(grad, extent=[x,x+w,y,y+h], origin='lower', aspect="auto",
                  norm=cm.colors.NoNorm(vmin=0,vmax=1), cmap=plt.get_cmap(cm

# Pass chart through function
gradientbars(bars, df_day_mean['Visits'], 'RdYlBu_r')

# Show
plt.tight_layout()
```



Heatmaps between day, time, and amount of visits.

In [26]:

```
df = df_new.drop(['Date'], axis = 1)

# Set times in order
df['Time'] = df['Time'].replace({'6:00 a.m.': 6, '7:00 a.m.': 7, '8:00 a.m.'
             '10:00 a.m.': 10, '11:00 a.m.': 11, '12:00 p.m.': 12, '1:00 p.m
             '3:00 p.m.': 15, '4:00 p.m.': 16, '5:00 p.m.': 17, '6:00 p.m.':
             '8:00 p.m.':20, '9:00 p.m.': 21, '10:00 p.m.': 22})

# Make new dataframe for heatmap
heatmap1_data = pd.pivot_table(df, values='Visits',
                     index=['Day'],
                     columns='Time')

heatmap1_data = heatmap1_data.reset_index()

heatmap_data = heatmap1_data.set_index(['Day']).loc[day_order]

# Make plot
fig = sns.heatmap(heatmap_data, cmap="Reds")
fig.tick_params(left=False, bottom=False)

# Show
fig
```

Out[26]: <AxesSubplot:xlabel='Time', ylabel='Day'>

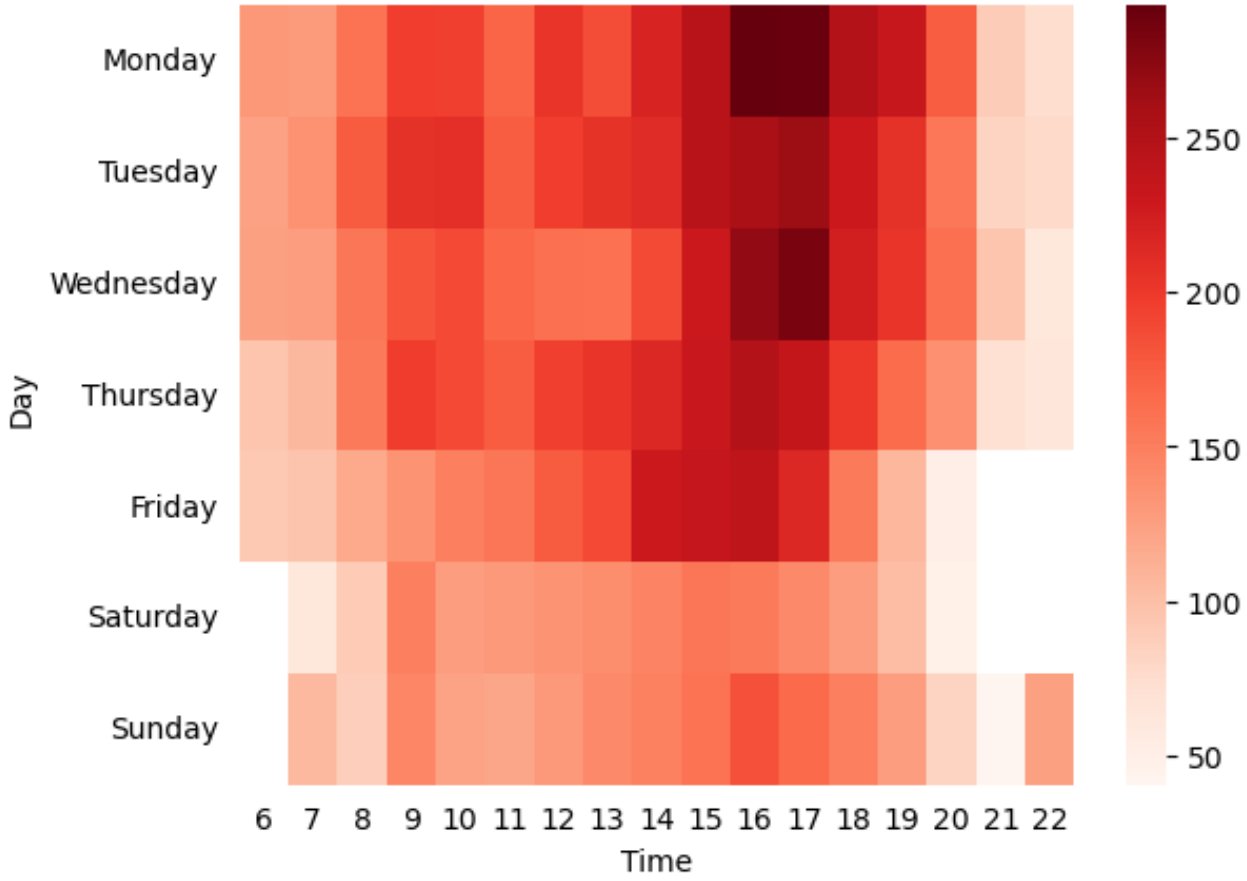
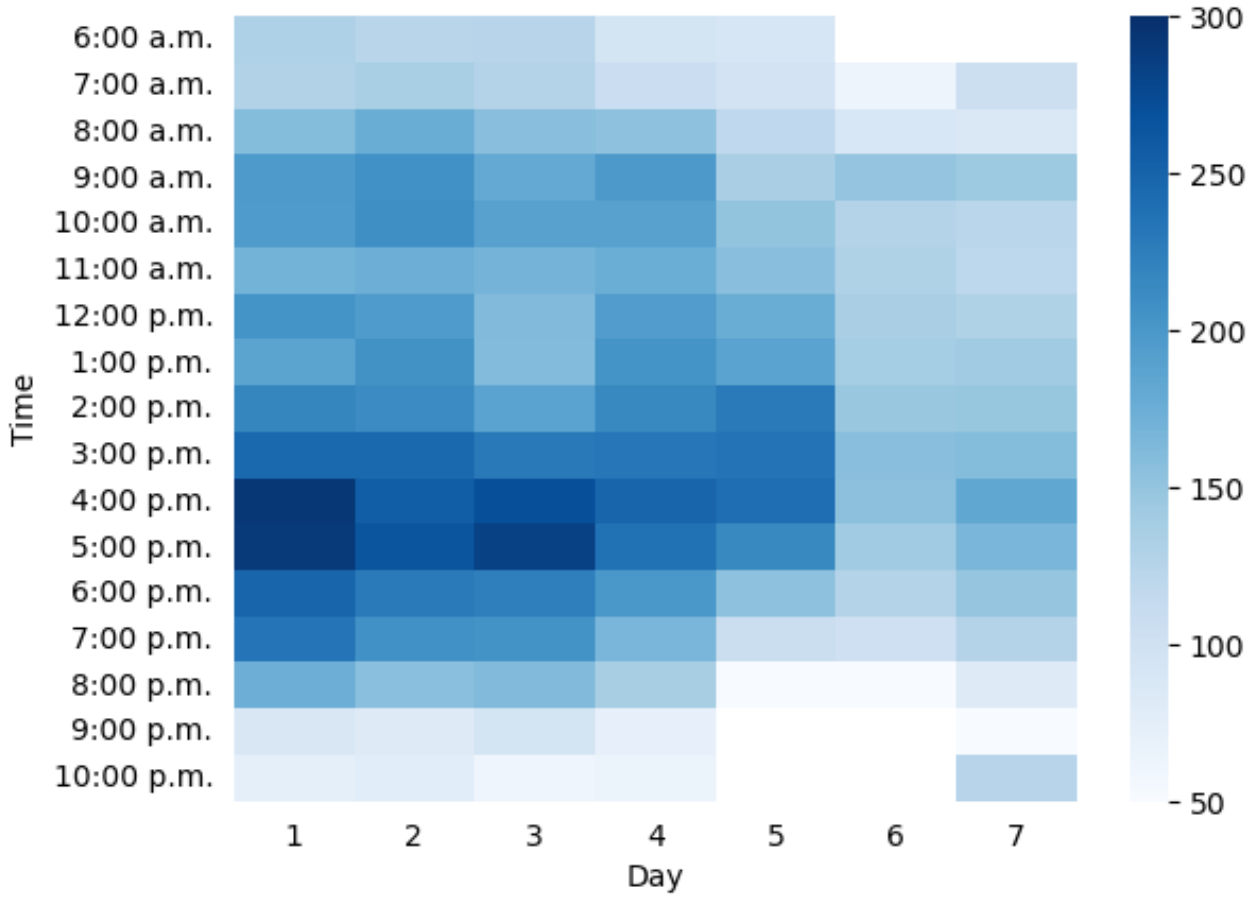


Chart is hard to read the data easily. Putting the times on the y axis would make it easy to see what time is the best

In [27]:

```
df = df_new.drop(['Date'], axis = 1)

# Set days in order
df['Day'] = df['Day'].replace({'Monday': 1, 'Tuesday': 2, 'Wednesday': 3, 'T

# Make dataset
heatmap2_data = pd.pivot_table(df, values='Visits',
                     index=['Time'],
                     columns='Day')

heatmap3_data = heatmap2_data.reset_index()

heatmap4_data = heatmap3_data.set_index(['Time']).loc[time_order]


# Loop through cmaps to see what looks best

colors = plt.colormaps()
#for i in colors:
    #fig2 = sns.heatmap(heatmap4_data, cmap=i, vmin = 50, vmax = 300)
    #print(i)
    #plt.show()

# Make and display chart
fig3 = sns.heatmap(heatmap4_data, cmap='Blues', vmin = 50, vmax = 300)
fig3.tick_params(left=False, bottom=False)
plt.savefig('Heatmap_RecCen')
```



Compare Finals / Midterm week to normal weeks

In [28]:

```
frames = [Fall2019, Fall2022]
df_new = pd.concat(frames)
df_new = df_new.drop(columns = ['level_0', 'index'])

# Make dataframes for midterm and finals week for 2019 and 2022
df_midterm_2019 = df_new.loc[(df_new['Date'] == '10/27/2019') | (df_new['Dat
                        (df_new['Date'] == '10/29/2019') | (df_new['Date'] ==
                       (df_new['Date'] == '10/31/2019') | (df_new['Date'] ==

df_finals_2019 = df_new.loc[(df_new['Date'] == '12/7/2019') | (df_new['Date'
                        (df_new['Date'] == '12/9/2019') | (df_new['Date'] ==
                       (df_new['Date'] == '12/11/2019') | (df_new['Date'] ==
                            (df_new['Date'] == '12/12/2019')]

df_midterm_2022 = df_new.loc[(df_new['Date'] == '10/23/2022') | (df_new['Dat
                             (df_new['Date'] == '10/25/2022') | (df_new['Date
                             (df_new['Date'] == '10/27/2022') | (df_new['Date

df_finals_2022 = df_new.loc[(df_new['Date'] == '12/3/2022') | (df_new['Date'
                           (df_new['Date'] == '12/5/2022') | (df_new['Date']
                           (df_new['Date'] == '12/7/2022') | (df_new['Date']
                           (df_new['Date'] == '12/9/2022')]

frames = [df_midterm_2019, df_midterm_2022, df_finals_2019, df_finals_2022]

test_df = pd.concat(frames)

# Make dataframe that does not include these days

df_not_midterm_2019 = df_new.loc[(df_new['Date'] != '10/27/2019') | (df_new[
                        (df_new['Date'] != '10/29/2019') | (df_new['Date'] !=
                       (df_new['Date'] != '10/31/2019') | (df_new['Date'] !=

df_not_finals_2019 = df_new.loc[(df_new['Date'] != '12/7/2019') | (df_new['D
                        (df_new['Date'] != '12/9/2019') | (df_new['Date'] !=
                       (df_new['Date'] != '12/11/2019') | (df_new['Date'] !=
                            (df_new['Date'] != '12/12/2019')]

df_not_midterm_2022 = df_new.loc[(df_new['Date'] != '10/23/2022') | (df_new[
                             (df_new['Date'] != '10/25/2022') | (df_new['Date
                             (df_new['Date'] != '10/27/2022') | (df_new['Date

df_not_finals_2022 = df_new.loc[(df_new['Date'] != '12/3/2022') | (df_new['D
                           (df_new['Date'] != '12/5/2022') | (df_new['Date']
                           (df_new['Date'] != '12/7/2022') | (df_new['Date']
                           (df_new['Date'] != '12/9/2022')]

frames2 = [df_not_midterm_2019, df_not_midterm_2022, df_not_finals_2019, df_

nontest_df = pd.concat(frames2)
test_df.head()
```

Out[28]:

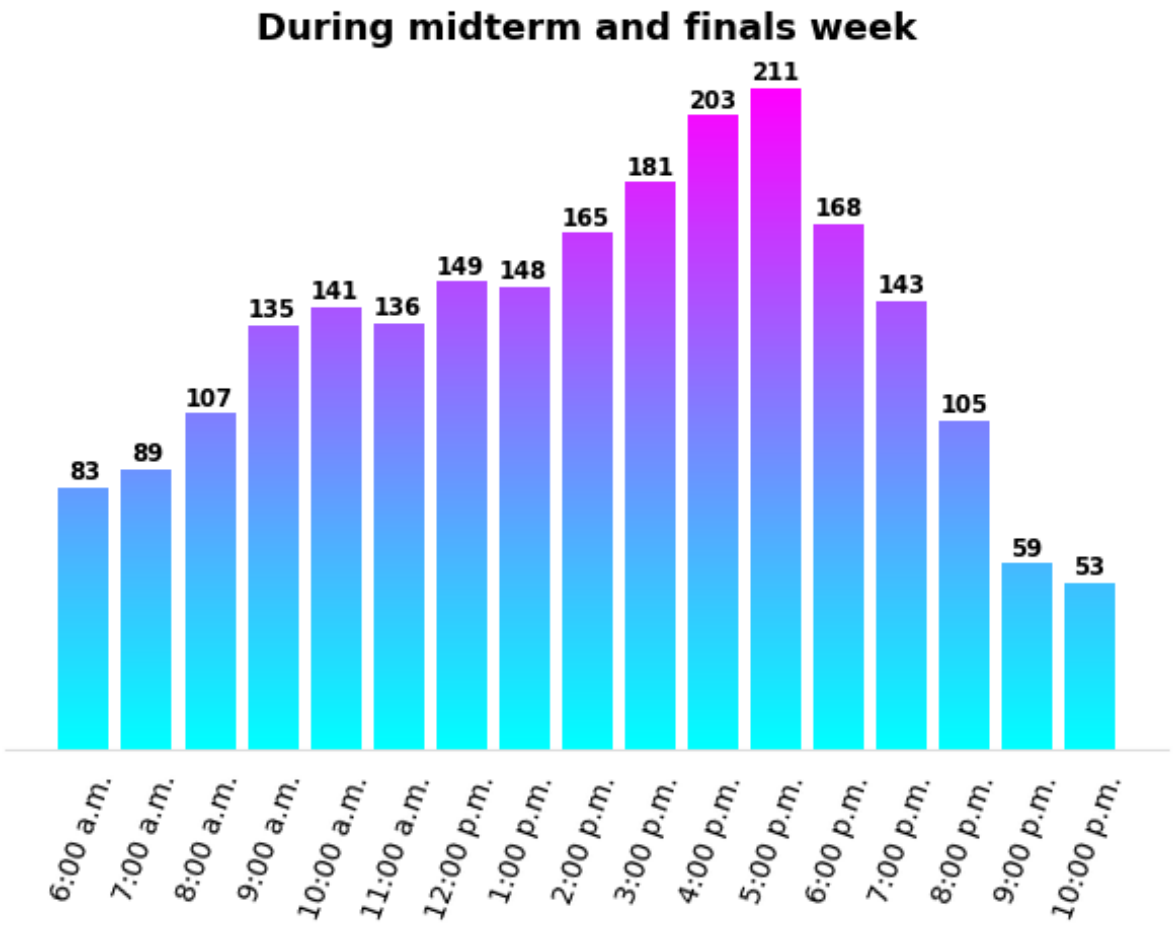
	Time	Day	Date	Visits
4	6:00 a.m.	Monday	10/28/2019	141
14	6:00 a.m.	Tuesday	10/29/2019	116
25	6:00 a.m.	Wednesday	10/30/2019	143
36	6:00 a.m.	Thursday	10/31/2019	67
46	6:00 a.m.	Friday	11/1/2019	73

Create visual comparing the two

In [29]:

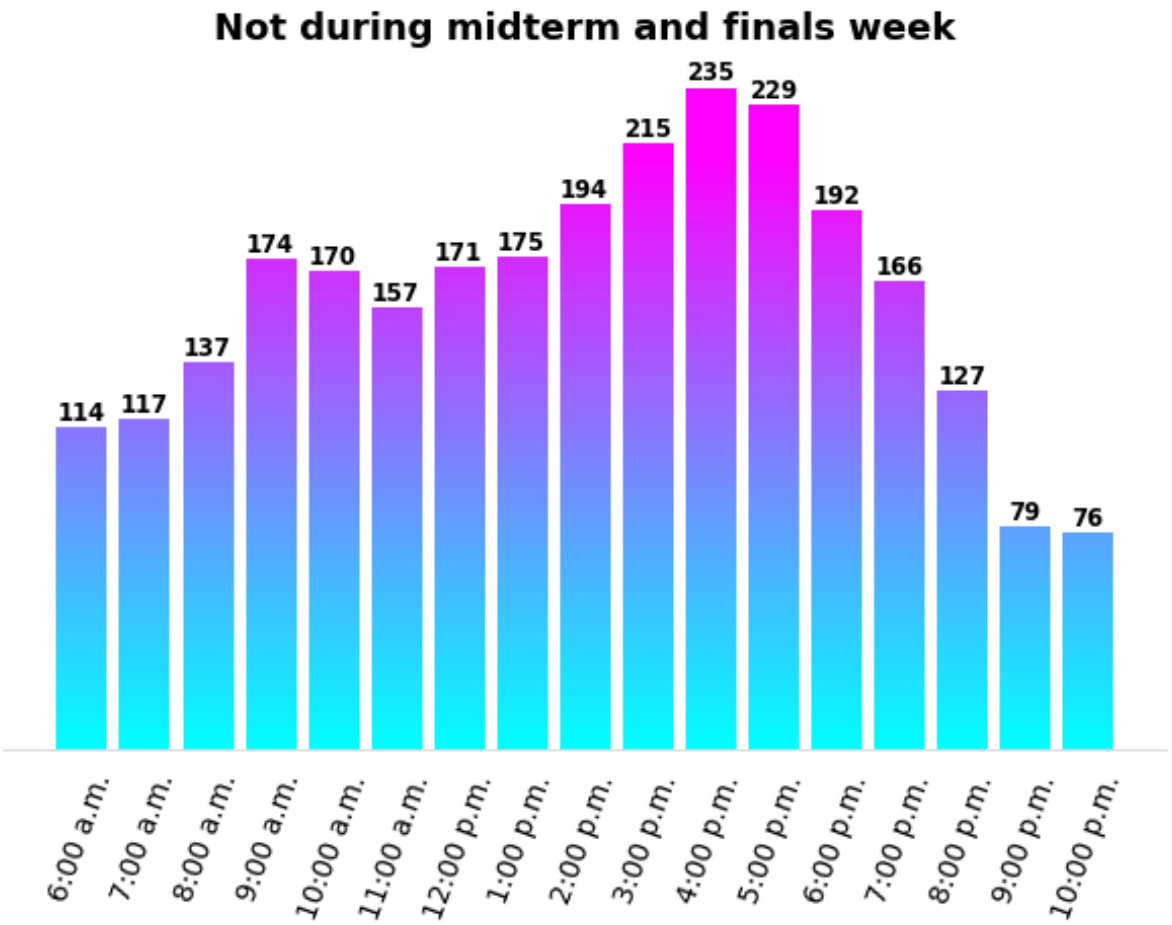
```
# Get data in correct form to be plotted
df_test_time_mean = pd.DataFrame(test_df.groupby('Time').mean())
df_test_time_mean = df_test_time_mean.reset_index()

df_test_time_mean = df_test_time_mean.set_index('Time').loc[time_order]
df_test_time_mean = df_test_time_mean.reset_index()

# Plot
fig, ax = plt.subplots()
bars = ax.bar(df_test_time_mean['Time'], df_test_time_mean['Visits'], color

# Make plot Title/Tick marks
plt.title('During midterm and finals week', fontsize = 13, weight = 'bold')
plt.xticks(fontsize = 10, rotation = 70)
plt.yticks(fontsize = 10, color = 'w')

# Change style of graph
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['bottom'].set_color('#DDDDDD')
ax.tick_params(bottom=False, left=False)
ax.set_axisbelow(True)
ax.yaxis.grid(False)
ax.xaxis.grid(False)

# Add labels to bars for exact number
bar_color = bars[0].get_facecolor()

for bar in bars:
  ax.text(
      bar.get_x() + bar.get_width() / 2,
      bar.get_height() + 3,
      int(round(bar.get_height(),1)),
      horizontalalignment='center',
      color='black',
      weight='heavy',
      fontsize = 8.3
  )

# Apply gradient for bars
```

```
def gradientbars(bars,ydata,cmap):
    ax = bars[0].axes
    lim = ax.get_xlim()+ax.get_ylim()
    ax.axis(lim)
    for bar in bars:
        bar.set_facecolor("none")
        x,y = bar.get_xy()
        w, h = bar.get_width(), bar.get_height()
        grad = np.atleast_2d(np.linspace(0,1*h/max(ydata),256)).T
        ax.imshow(grad, extent=[x,x+w,y,y+h], origin='lower', aspect="auto",
                  norm=cm.colors.NoNorm(vmin=0,vmax=1), cmap=plt.get_cmap(cm

# Pass chart through function
gradientbars(bars, df_test_time_mean['Visits'], 'cool')

plt.tight_layout()
plt.savefig('Finals.png')
```



In [30]:
```
# Get data in correct form to be plotted
df_nontest_time_mean = pd.DataFrame(nontest_df.groupby('Time').mean())
df_nontest_time_mean = df_nontest_time_mean.reset_index()

df_nontest_time_mean = df_nontest_time_mean.set_index('Time').loc[time_order
df_nontest_time_mean = df_nontest_time_mean.reset_index()

# Plot
fig, ax = plt.subplots()
bars = ax.bar(df_nontest_time_mean['Time'], df_nontest_time_mean['Visits'],
```

```
# Make plot Title/Tick marks
plt.title('Not during midterm and finals week', fontsize = 13, weight = 'bol
plt.xticks(fontsize = 10, rotation = 70)
plt.yticks(fontsize = 10, color = 'w')

# Change style of graph
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['bottom'].set_color('#DDDDDD')
ax.tick_params(bottom=False, left=False)
ax.set_axisbelow(True)
ax.yaxis.grid(False)
ax.xaxis.grid(False)

# Add labels to bars for exact number
bar_color = bars[0].get_facecolor()

for bar in bars:
  ax.text(
      bar.get_x() + bar.get_width() / 2,
      bar.get_height() + 3,
      int(round(bar.get_height(),1)),
      horizontalalignment='center',
      color='black',
      weight='heavy',
      fontsize = 8.3
  )

# Apply gradient for bars
def gradientbars(bars,ydata,cmap):
    ax = bars[0].axes
    lim = ax.get_xlim()+ax.get_ylim()
    ax.axis(lim)
    for bar in bars:
        bar.set_facecolor("none")
        x,y = bar.get_xy()
        w, h = bar.get_width(), bar.get_height()
        grad = np.atleast_2d(np.linspace(0,1*h/max(ydata),256)).T
        ax.imshow(grad, extent=[x,x+w,y,y+h], origin='lower', aspect="auto",
                  norm=cm.colors.NoNorm(vmin=0,vmax=1), cmap=plt.get_cmap(cm

# Pass chart through function
gradientbars(bars, df_test_time_mean['Visits'], 'cool')

plt.tight_layout()
plt.savefig('NonFinals.png')
```



Use linear regression model to see if we can get an accuration prediction on number of visitors

Scale the data

In [31]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

In [32]:
```
df = df_new.copy()
train_data = df
cols = train_data.columns
```

In [33]:
```
train_data = pd.get_dummies(df)
```

Train test split

In [34]:
```
#import train_test_split
from sklearn.model_selection import train_test_split
```

In [35]:
```
X = train_data.drop(columns = 'Visits')
y = train_data['Visits']
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

Linear regression model

In [37]:
```
from sklearn.linear_model import LinearRegression
```

In [38]:
```
LR = LinearRegression()
```

In [39]:
```
LR.fit(X_train,y_train)
```

Out[39]:
```
LinearRegression()
```

In [40]:
```
pred = LR.predict(X_test)
# Round up
pred = np.around(pred)
```

Accuracy score

In [41]:
```
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score,confusion_matrix
```

R^2 score

In [42]:
```
r2_score(y_test, pred)
```

Out[42]:
```
0.6830996042755412
```

Accuracy score

In [43]:
```
lr_accuracy=accuracy_score(y_test,pred)
lr_accuracy
```

Out[43]:
```
0.008583690987124463
```

In [44]:
```
data = {'Visits':y_test, 'Prediction': pred}
comp = pd.DataFrame(data)
comp
```

Out[44]:

	Visits	Prediction
855	151	184.0
944	174	137.0
1105	24	55.0
20	114	140.0
587	251	231.0
...	...	...
502	164	207.0
769	344	268.0
778	305	273.0
448	61	37.0
728	165	188.0

466 rows × 2 columns

Conclusion

We can see the model did a good job with taking the structure of the data, but it was way too centered around the average amount of people and is not able to tell the "peak" hours from the normal ones. A better approach we could use for this would involve looking at forecasting a time series.

Using an ARIMA (AutoRegressive Integrated Moving Average Model) which helps with time forecasting to give a better prediction based on the values from previous times

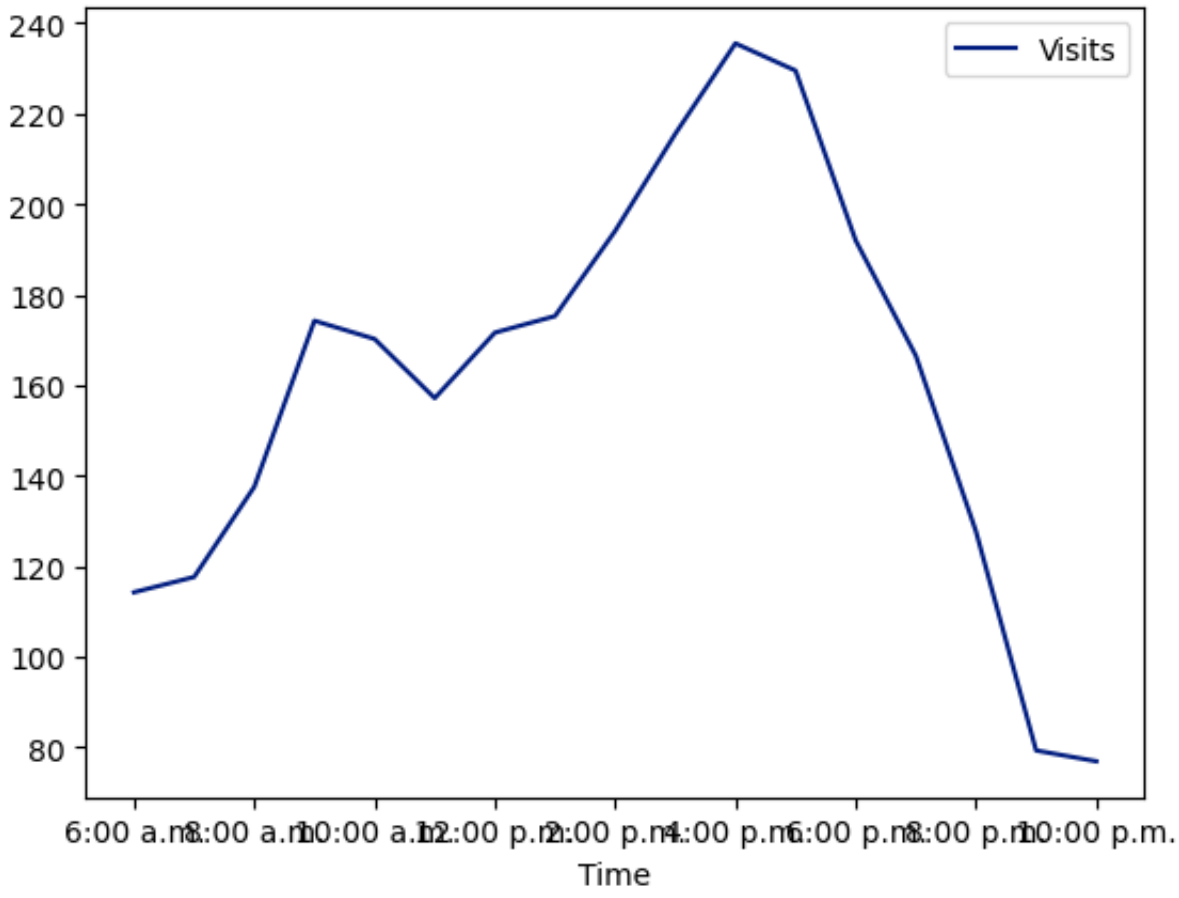
In [45]:
```
# import libraries
from statsmodels.tsa.arima.model import ARIMA

df_arima = df.drop(columns = ['Day', 'Date'])

df_arima['datetime'] = pd.to_datetime(df_arima['Time'])

df_arima['time'] = pd.to_datetime(df_arima['datetime']).dt.time
df_arima = df_arima.drop(columns = ['datetime', 'Time'])

df_arima = df_arima.set_index('time')
model = ARIMA(df_arima, order=(2,1,2))

results = model.fit()

fit = results.fittedvalues

fit = fit.reset_index()

fit3 = fit
fit2 = fit

fit = fit.groupby('time').mean()

fit = fit.rename(columns = {0: 'Visits'})
```

```
/Users/jakejensema/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa
/base/tsa_model.py:471: ValueWarning:

An unsupported index was provided and will be ignored when e.g. forecasting.

/Users/jakejensema/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa
/base/tsa_model.py:471: ValueWarning:

An unsupported index was provided and will be ignored when e.g. forecasting.

/Users/jakejensema/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa
/base/tsa_model.py:471: ValueWarning:

An unsupported index was provided and will be ignored when e.g. forecasting.
```

Seeing how the ARIMA model did

In [46]:
```
fit2 = fit2.rename(columns = {0: 'Visits'})

df_mean = df.groupby('Time').mean()
df_mean = df_mean.reindex(index = time_order)
fit_plot = fit2.groupby('time').mean()
fit3[0] = fit3[0].round().astype(int)
df['Predicted'] = fit3[0]

# Show plot
df_mean.plot()
```

Out[46]: <AxesSubplot:xlabel='Time'>



Metric scores

In [47]:
```
r2_score(df['Visits'], df['Predicted'])
```

Out[47]: 0.3398511516066497

In [48]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

mse = mean_squared_error(df['Visits'], df['Predicted'])

print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 3715.401287553648
```

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

We see a very high Mean squared error(MSE) because of the outliers in our data.