In [1]:
```
# initializing otter-grader
import otter
grader = otter.Notebook()
```

Homework 3: Bike Sharing

Exploratory Data Analysis (EDA) and Visualization

Due Date: Sunday 5/3, 11:59 PM

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** below.

Collaborators: *list collaborators here*

Introduction

Bike sharing systems are new generation of traditional bike rentals where the process of signing up, renting and returning is automated. Through these systems, users are able to easily rent a bike from one location and return them to another. We will be analyzing bike sharing data from Washington D.C.

In this assignment, you will perform tasks to clean, visualize, and explore the bike sharing data. You will also investigate open-ended questions. These open-ended questions ask you to think critically about how the plots you have created provide insight into the data.

After completing this assignment, you should be comfortable with:

- reading plaintext delimited data into `pandas`
- wrangling data for analysis
- using EDA to learn about your data
- making informative plots with Altair

We recommend reading through the entire assignment first to get an idea of your goals.

Grading

Grading is broken down into autograded answers and free response.

For autograded answers, the results of your code are compared to provided and/or hidden tests.

For free response, readers will evaluate how well you answered the question and/or fulfilled the requirements of the question.

For plots, your plots should be *similar* to the given examples. We will tolerate small variations such as color differences or slight variations in scale. However it is in your best interest to make the plots as similar as possible, as similarity is subject to the readers.

Note that for ALL plotting questions from here on out, we will expect appropriate titles, axis labels, legends, etc. The following question serves as a good guideline on what is "enough": If I directly downloaded the plot and viewed it, would I be able to tell what was being visualized without knowing the question?

In [2]:
```
# Run this cell to set up your notebook.
# Make sure ds100_utils.py is in this assignment's folder
import csv
import numpy as np
import pandas as pd
import altair as alt
alt.data_transformers.disable_max_rows()

import zipfile
from pathlib import Path
import ds100_utils # custom file to add helpful routines

# Default plot configurations

from IPython.display import display, Latex, Markdown
```

Loading Bike Sharing Data

The data we are exploring is collected from a bike sharing system in Washington D.C.

The variables in this data frame are defined as:

Variable	Description
instant	record index
dteday	date
season	1. spring 2. summer 3. fall 4. winter
yr	year (0: 2011, 1:2012)
mnth	month (1 to 12)
hr	hour (0 to 23)
holiday	whether day is holiday or not
weekday	day of the week
workingday	if day is neither weekend nor holiday
weathersit	1. clear or partly cloudy 2. mist and clouds 3. light snow or rain 4. heavy rain or snow
temp	normalized temperature in Celsius (divided by 41)
atemp	normalized "feels-like" temperature in Celsius (divided by 50)
hum	normalized percent humidity (divided by 100)
windspeed	normalized wind speed (divided by 67)
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes including casual and registered

Download the Data

```
In [3]: data_dir = Path('data')
In [4]: # Run this cell to look at the top of the file.  No further action is n
        eeded
        for line in ds100_utils.head(data_dir/'bikeshare.txt'):
            print(line,end="")
```

```
instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathersit,
temp,atemp,hum,windspeed,casual,registered,cnt
1,2011-01-01,1,0,1,0,0,6,0,1,0.24,0.2879,0.81,0,3,13,16
2,2011-01-01,1,0,1,1,0,6,0,1,0.22,0.2727,0.8,0,8,32,40
3,2011-01-01,1,0,1,2,0,6,0,1,0.22,0.2727,0.8,0,5,27,32
4,2011-01-01,1,0,1,3,0,6,0,1,0.24,0.2879,0.75,0,3,10,13
```

Size

Is the file big? How many records do we expect to find?

Let's find out.

```
In [5]: # Run this cell to view some metadata.  No further action is needed
        print("Size:", (data_dir/"bikeshare.txt").stat().st_size, "bytes")
        print("Line Count:", ds100_utils.line_count(data_dir/"bikeshare.txt"),
        "lines")
```

```
Size: 1156736 bytes
Line Count: 17380 lines
```

Loading the data

The following code loads the data into a Pandas DataFrame.

In [6]:
```
# Run this cell to load the data.  No further action is needed
bike = pd.read_csv(data_dir/'bikeshare.txt')
bike.head(30)
```

Out[6]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	at
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2
5	6	2011-01-01	1	0	1	5	0	6	0	2	0.24	0.2
6	7	2011-01-01	1	0	1	6	0	6	0	1	0.22	0.2
7	8	2011-01-01	1	0	1	7	0	6	0	1	0.20	0.2
8	9	2011-01-01	1	0	1	8	0	6	0	1	0.24	0.2
9	10	2011-01-01	1	0	1	9	0	6	0	1	0.32	0.3
10	11	2011-01-01	1	0	1	10	0	6	0	1	0.38	0.3
11	12	2011-01-01	1	0	1	11	0	6	0	1	0.36	0.3
12	13	2011-01-01	1	0	1	12	0	6	0	1	0.42	0.4
13	14	2011-01-01	1	0	1	13	0	6	0	2	0.46	0.4
14	15	2011-01-01	1	0	1	14	0	6	0	2	0.46	0.4
15	16	2011-01-01	1	0	1	15	0	6	0	2	0.44	0.4
16	17	2011-01-01	1	0	1	16	0	6	0	2	0.42	0.4
17	18	2011-01-01	1	0	1	17	0	6	0	2	0.44	0.4
18	19	2011-01-01	1	0	1	18	0	6	0	3	0.42	0.4
19	20	2011-01-01	1	0	1	19	0	6	0	3	0.42	0.4
20	21	2011-01-01	1	0	1	20	0	6	0	2	0.40	0.4
21	22	2011-01-01	1	0	1	21	0	6	0	2	0.40	0.4
22	23	2011-01-01	1	0	1	22	0	6	0	2	0.40	0.4

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	at
23	24	2011-01-01	1	0	1	23	0	6	0	2	0.46	0.4
24	25	2011-01-02	1	0	1	0	0	0	0	2	0.46	0.4
25	26	2011-01-02	1	0	1	1	0	0	0	2	0.44	0.4
26	27	2011-01-02	1	0	1	2	0	0	0	2	0.42	0.4
27	28	2011-01-02	1	0	1	3	0	0	0	2	0.46	0.4
28	29	2011-01-02	1	0	1	4	0	0	0	2	0.46	0.4
29	30	2011-01-02	1	0	1	6	0	0	0	3	0.42	0.4

Below, we show the shape of the file. You should see that the size of the DataFrame matches the number of lines in the file, minus the header row.

In [7]:
```
bike.shape
```

Out[7]:
```
(17379, 17)
```

0: Examining the Data

Before we start working with the data, let's examine its granularity.

Question 0A

Look at the data stored in the dataframe `bike` . What is the granularity of these data (i.e. what does each row represent)?

how many bikes where rented, along with weather status etc, for intervals in a day

In [8]:
```
# Use this cell for scratch work.
# If you need to add more cells for scratch work, add them BELOW this c
ell.
```

Question 0B

For this assignment, we'll be using this data to study bike usage in Washington D.C. Based on the granularity and the variables present in the data, what might some limitations of using these data be? What would be two additional data categories/variables that you can collect to address some of these limitations?

we do not know the time intervals, nor the locations.

In [9]:
```
# Use this cell for scratch work. If you need to add more cells for scr
atch work, add them BELOW this cell.
```

1: Data Preparation

A few of the variables that are numeric/integer actually encode categorical data. These include `holiday` , `weekday` , `workingday` , and `weathersit` . In the following problem, we will convert these four variables to strings specifying the categories. In particular, use 3-letter labels (`Sun` , `Mon` , `Tue` , `Wed` , `Thu` , `Fri` , and `Sat`) for `weekday` . You may simply use `yes` / `no` for `holiday` and `workingday` .

In this exercise we will *mutate* the data frame, **overwriting the corresponding variables in the data frame.** However, our notebook will effectively document this in-place data transformation for future readers. Make sure to leave the underlying datafile `bikeshare.txt` unmodified.

Question 1a (Decoding `weekday` , `workingday` , and `weathersit`)

Decode the `holiday` , `weekday` , `workingday` , and `weathersit` fields:

1. `holiday` : Convert to `yes` and `no` . Hint: There are fewer holidays...
2. `weekday` : It turns out that Monday is the day with the most holidays. Mutate the `'weekday'` column to use the 3-letter label (`'Sun'` , `'Mon'` , `'Tue'` , `'Wed'` , `'Thu'` , `'Fri'` , and `'Sat'` ...) instead of its current numerical values. Assume `0` corresponds to `Sun` , `1` to `Mon` and so on.
3. `workingday` : Convert to `yes` and `no` .
4. `weathersit` : You should replace each value with one of `Clear` , `Mist` , `Light` , or `Heavy` . Hint: Use the `'hum'` (humidity) column to determine whether your value should be `Clear` (less humid), `Mist` (slightly more humid), `Light` (more humid), or `Heavy` (very humid).

Note: In this exercise you need to *mutate* the data frame, **overwriting the corresponding variables in the data frame**. If you want to revert changes, run the cell that reloads the csv.

Hint: One approach is to use the [replace (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html) method of the pandas DataFrame class. We haven't discussed how to do this so you'll need to look at the documentation. The most concise way is to use the approach described in the documentation as "nested-dictonaries" (which you can generate manually to describe the mapping of the values), though there are many possible solutions.

```
In [10]: bike.head(50)
```

Out[10]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	at
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2
5	6	2011-01-01	1	0	1	5	0	6	0	2	0.24	0.2
6	7	2011-01-01	1	0	1	6	0	6	0	1	0.22	0.2
7	8	2011-01-01	1	0	1	7	0	6	0	1	0.20	0.2
8	9	2011-01-01	1	0	1	8	0	6	0	1	0.24	0.2
9	10	2011-01-01	1	0	1	9	0	6	0	1	0.32	0.3
10	11	2011-01-01	1	0	1	10	0	6	0	1	0.38	0.3
11	12	2011-01-01	1	0	1	11	0	6	0	1	0.36	0.3
12	13	2011-01-01	1	0	1	12	0	6	0	1	0.42	0.4
13	14	2011-01-01	1	0	1	13	0	6	0	2	0.46	0.4
14	15	2011-01-01	1	0	1	14	0	6	0	2	0.46	0.4
15	16	2011-01-01	1	0	1	15	0	6	0	2	0.44	0.4
16	17	2011-01-01	1	0	1	16	0	6	0	2	0.42	0.4
17	18	2011-01-01	1	0	1	17	0	6	0	2	0.44	0.4
18	19	2011-01-01	1	0	1	18	0	6	0	3	0.42	0.4
19	20	2011-01-01	1	0	1	19	0	6	0	3	0.42	0.4
20	21	2011-01-01	1	0	1	20	0	6	0	2	0.40	0.4
21	22	2011-01-01	1	0	1	21	0	6	0	2	0.40	0.4
22	23	2011-01-01	1	0	1	22	0	6	0	2	0.40	0.4

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	at
23	24	2011-01-01	1	0	1	23	0	6	0	2	0.46	0.4
24	25	2011-01-02	1	0	1	0	0	0	0	2	0.46	0.4
25	26	2011-01-02	1	0	1	1	0	0	0	2	0.44	0.4
26	27	2011-01-02	1	0	1	2	0	0	0	2	0.42	0.4
27	28	2011-01-02	1	0	1	3	0	0	0	2	0.46	0.4
28	29	2011-01-02	1	0	1	4	0	0	0	2	0.46	0.4
29	30	2011-01-02	1	0	1	6	0	0	0	3	0.42	0.4
30	31	2011-01-02	1	0	1	7	0	0	0	2	0.40	0.4
31	32	2011-01-02	1	0	1	8	0	0	0	3	0.40	0.4
32	33	2011-01-02	1	0	1	9	0	0	0	2	0.38	0.3
33	34	2011-01-02	1	0	1	10	0	0	0	2	0.36	0.3
34	35	2011-01-02	1	0	1	11	0	0	0	2	0.36	0.3
35	36	2011-01-02	1	0	1	12	0	0	0	2	0.36	0.3
36	37	2011-01-02	1	0	1	13	0	0	0	2	0.36	0.3
37	38	2011-01-02	1	0	1	14	0	0	0	3	0.36	0.3
38	39	2011-01-02	1	0	1	15	0	0	0	3	0.34	0.3
39	40	2011-01-02	1	0	1	16	0	0	0	3	0.34	0.3
40	41	2011-01-02	1	0	1	17	0	0	0	1	0.34	0.3
41	42	2011-01-02	1	0	1	18	0	0	0	2	0.36	0.3
42	43	2011-01-02	1	0	1	19	0	0	0	1	0.32	0.2
43	44	2011-01-02	1	0	1	20	0	0	0	1	0.30	0.2
44	45	2011-01-02	1	0	1	21	0	0	0	1	0.26	0.2
45	46	2011-01-02	1	0	1	22	0	0	0	1	0.24	0.2

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	at
46	47	2011-01-02	1	0	1	23	0	0	0	1	0.22	0.2
47	48	2011-01-03	1	0	1	0	0	1	1	1	0.22	0.1
48	49	2011-01-03	1	0	1	1	0	1	1	1	0.20	0.1
49	50	2011-01-03	1	0	1	4	0	1	1	1	0.16	0.1

In [11]:
```
# Modify holiday weekday, workingday, and weathersit here
bike['holiday'].replace(0, "no", inplace=True)
bike['holiday'].replace(1, "yes", inplace=True)

bike['weekday'].replace(0, "Sun", inplace=True)
bike['weekday'].replace(1, "Mon", inplace=True)
bike['weekday'].replace(2, "Tue", inplace=True)
bike['weekday'].replace(3, "Wed", inplace=True)
bike['weekday'].replace(4, "Thu", inplace=True)
bike['weekday'].replace(5, "Fri", inplace=True)
bike['weekday'].replace(6, "Sat", inplace=True)

bike['workingday'].replace(0, "no", inplace=True)
bike['workingday'].replace(1, "yes", inplace=True)

bike['weathersit'].replace(4, "Heavy", inplace=True)
bike['weathersit'].replace(3, "Light", inplace=True)
bike['weathersit'].replace(2, "Mist", inplace=True)
bike['weathersit'].replace(1, "Clear", inplace=True)
```

Question 1b (Holidays)

How many entries in the data correspond to holidays? Set the variable `num_holidays` to this value.

In [12]:
```
num_holidays = len(bike.loc[bike['holiday'] == "yes"])
```

Question 1c (Computing Daily Total Counts)

In the next few questions we will be analyzing the daily number of registered and unregistered users.

Construct a data frame named `daily_counts` indexed by `dteday` with the following columns:

- `casual` : total number of casual riders for each day
- `registered` : total number of registered riders for each day
- `workingday` : whether that day is a working day or not (`yes` or `no`)
- `weathersit` : what was the weather situation like (`Clear` , `Mist` , `Light` , or `Heavy`)
- `season` : what season the day falls on

Hint: `groupby` and `agg` . For the `agg` method, please check the [documentation (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.core.groupby.DataFrameGroupBy.agg.html)](https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.core.groupby.DataFrameGroupBy.agg.html) for examples on applying different aggregations per column. If you use the capability to do different aggregations by column, you can do this task with a single call to `groupby` and `agg` . For the `workingday` column we can take any of the values since we are grouping by the day, thus the value will be the same within each group. Take a look at the `'first'` or `'last'` aggregation functions.

In [13]:
```
daily_counts = bike.groupby(['dteday']).agg({'casual':[sum],
                                             'registered':[sum],
                                             'workingday':['first'],
                                             'weathersit':['first'],
                                             'season':['last']
                                             })
daily_counts.columns = ['casual', 'registered', 'workingday', 'weathers
it','season']
daily_counts.head()
```

Out[13]:

	casual	registered	workingday	weathersit	season
dteday					
2011-01-01	331	654	no	Clear	1
2011-01-02	131	670	no	Mist	1
2011-01-03	120	1229	yes	Clear	1
2011-01-04	108	1454	yes	Clear	1
2011-01-05	82	1518	yes	Clear	1

2: Exploring the Distribution of Riders

In this question we'll begin by comparing the distribution of the daily counts of casual and registered riders.

Question 2a

Usually there are multiple ways to solve a problem. In this and the next question, create the same visualization using `pandas` methods and the built-in Altair transformations.

For this question, create a long table using the pandas function melt (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.melt.html) that has one column indicating the rider type (`casual` vs `registered`) and another column with the number of riders.

The temporal granularity of the records should be *daily counts*, which you should have after completing question 1c.

As always, include a properly-labeled legend, xlabel, ylabel, and a title.

After creating the plot, look at it and make sure you understand what the plot is actually telling us, e.g., on a given day, the most likely number of registered riders we expect is ~4000, but it could be anywhere from nearly 0 to 7000. (Note that our example uses a `bin=alt.Bin(maxbins=100)` parameter.)

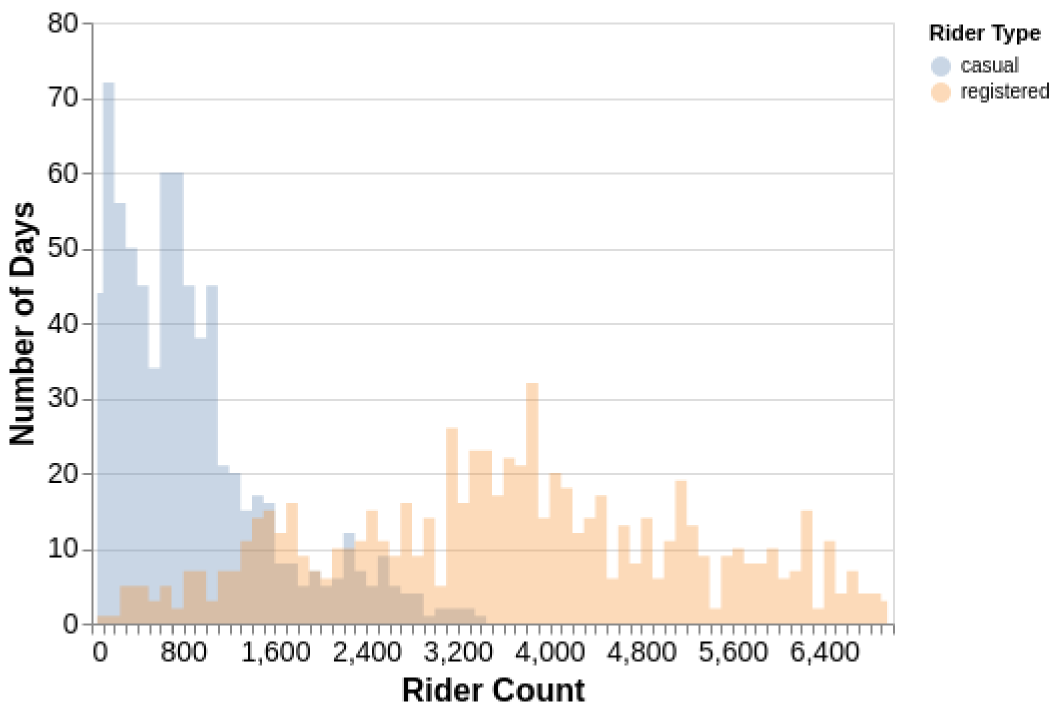
In [14]:
```
## A long table with 2 columns
daily_counts_long = pd.melt(daily_counts.reset_index(), value_vars=['ca
sual','registered'])
daily_counts_long.head()
```

Out[14]:

	variable	value
0	casual	331
1	casual	131
2	casual	120
3	casual	108
4	casual	82

In [15]:
```
## A long table with 2 columns
daily_counts_long = pd.melt(daily_counts.reset_index(), value_vars=['ca
sual','registered'])
daily_counts_long

## Call alt.Chart on daily_counts_long to make the look like the provid
ed figure

alt.Chart(daily_counts_long).mark_area(
    opacity=0.3,
    interpolate='step'
).encode(
    x=alt.X('value:Q', title='Rider Count', bin=alt.Bin(maxbins=100)),
    y=alt.Y('count()', title='Number of Days', stack=None),
    color=alt.Color("variable", legend=alt.Legend(title='Rider Type'))
).configure_axis(
    labelFontSize=14, # change axes label font size
    titleFontSize=16  # change axes title font size
)
```

Out[15]:



Question 2b

Use the layered_histogram (https://altair-viz.github.io/gallery/layered_histogram.html) function to create a plot that overlays the histograms of the daily counts of bike users, using one color to represent `casual` riders, and another to represent `registered` riders. The temporal granularity of the records should be *daily counts*, which you should have after completing question 1c. The example on the layered_histogram (https://altair-viz.github.io/gallery/layered_histogram.html) page should be very helpful for this question.

As always, include a properly-labeled legend, xlabel, ylabel, and a title.

After creating the plot, look at it and make sure you understand what the plot is actually telling us, e.g., on a given day, the most likely number of registered riders we expect is ~4000, but it could be anywhere from nearly 0 to 7000. (Note that our example uses a `bin=alt.Bin(maxbins=100)` parameter.)

Your figure should look identical to the figure in 2a.

Hint: You will need to use the `transform_fold` (https://altair-viz.github.io/user_guide/transform/fold.html) function shown in the layered_histogram (https://altair-viz.github.io/gallery/layered_histogram.html) example. This function is similar to `pd.melt` discussed in class, in that it takes a wide data frame and converts it to a long format. Altair works best with long format data (https://altair-viz.github.io/user_guide/transform/fold.html).

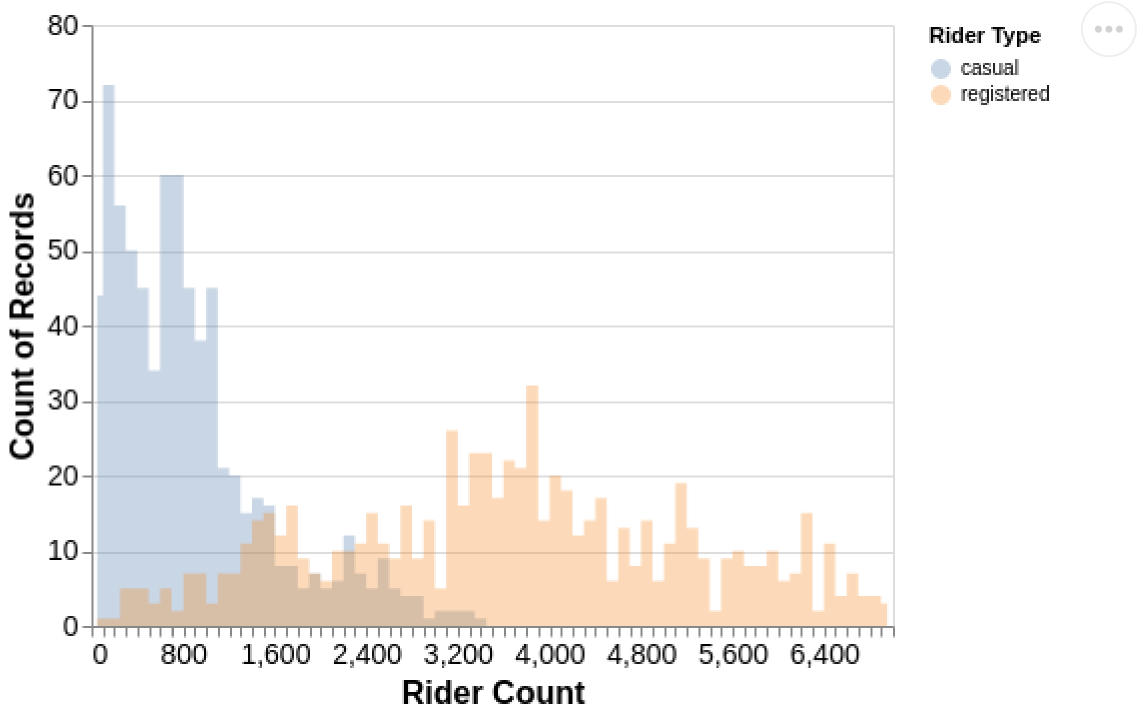
In [16]:
```
d_counts = daily_counts.reset_index()
d_counts.head()
```

Out[16]:

	dteday	casual	registered	workingday	weathersit	season
0	2011-01-01	331	654	no	Clear	1
1	2011-01-02	131	670	no	Mist	1
2	2011-01-03	120	1229	yes	Clear	1
3	2011-01-04	108	1454	yes	Clear	1
4	2011-01-05	82	1518	yes	Clear	1

In [17]:
```
alt.Chart(d_counts).transform_fold(
    ['casual', 'registered'],
    as_=['Experiment', 'Measurement']
).mark_area(
    opacity=0.3,
    interpolate='step'
).encode(
    x = alt.X('Measurement:Q', title='Rider Count', bin=alt.Bin(maxbins
=100)),
    y = alt.Y('count()', stack=None),
    color = alt.Color('Experiment:N', title='Rider Type')
).configure_axis(
    labelFontSize=14, # change axes label font size
    titleFontSize=16  # change axes title font size
)
```

Out[17]:



Question 2c

In the cell below, describe the differences you notice between the histograms for casual and registered riders. Consider concepts such as modes, symmetry, skewness, tails, gaps and outliers. Include a comment on the spread of the distributions.

rider count for registered is a gaussian distribution, with a mean of around 4,000, and a large variance. much greater range of rider counts

for casual users, distribution is more concentrated from 0-800 values, and does not exceed 3,500.

Question 2d

The density plots do not show us how the counts for registered and casual riders vary together. Use mark_point () to make a scatter plot to investigate the relationship between casual and registered counts. This time, let's use the `bike` DataFrame to plot hourly counts instead of daily counts. Color the points in the scatterplot according to whether or not the day is a working day (your colors do not have to match ours exactly, but they should be different based on whether the day is a working day). In addition, there are many points in the scatter plot, so make them small to help reduce overplotting (https://www.data-to-viz.com/caveat/overplotting.html).

The transform_regression (https://altair-viz.github.io/user_guide/transform/regression.html) function will also try to draw a linear regression line. In order to add two regression lines, set the `groupby` argument of the `transform_regression` to the working day variable.

Your image should look something like this:

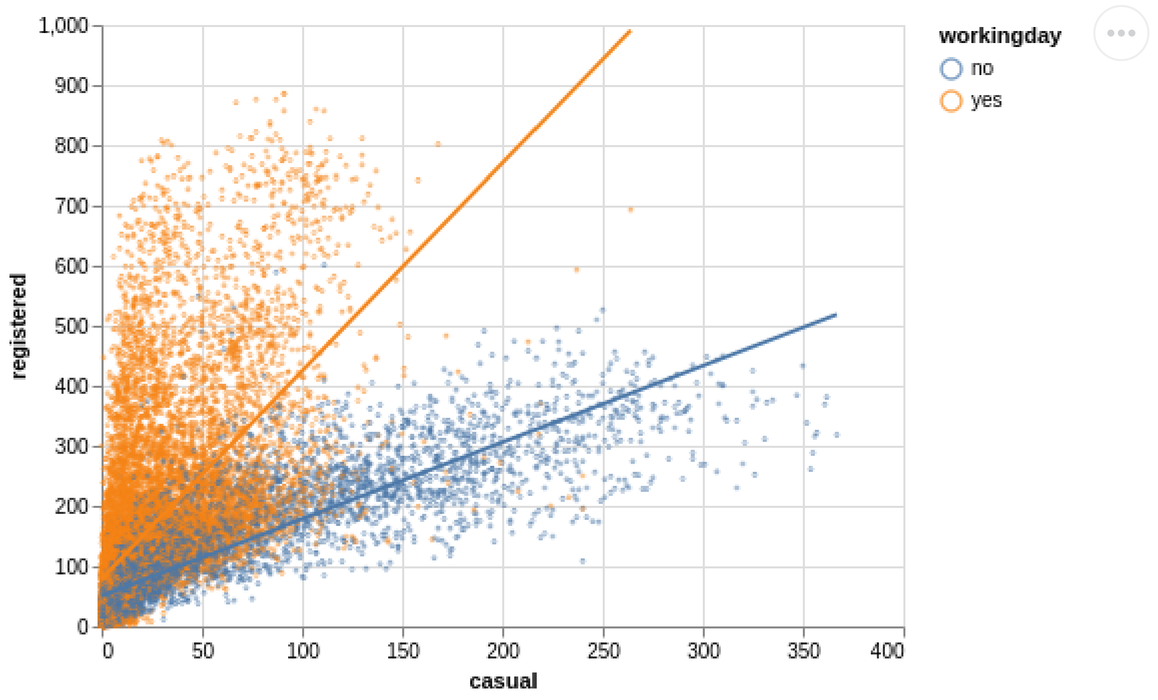
In [18]:
```
bike_long = pd.melt(bike, value_vars=['casual','registered'])
bike.head()
```

Out[18]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	ater
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24	0.28
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22	0.27
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22	0.27
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24	0.28
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24	0.28

In [19]:
```
chart = alt.Chart(bike).mark_point().encode(
    y=alt.Y('registered:Q'),
    x=alt.X('casual:Q'),
    color=alt.Color('workingday')
)

chart2 = chart.transform_regression('casual', 'registered', groupby=['w
orkingday']).mark_line()

alt.layer(chart, chart2).configure_point(size=0.5)
```

Out[19]:



Question 2e

What does this scatterplot reveal about the relationship (if any) between casual and registered riders and whether or not the day they ride is on the weekend? What effect does [overplotting (https://www.data-to-viz.com/caveat/overplotting.html)](https://www.data-to-viz.com/caveat/overplotting.html) have on your ability to describe this relationship?

registered users mostly use the service on working days, while casual users mostly use the service on non-working days

3 Interactive Visualizations of Seasonal Patterns

Question 3a

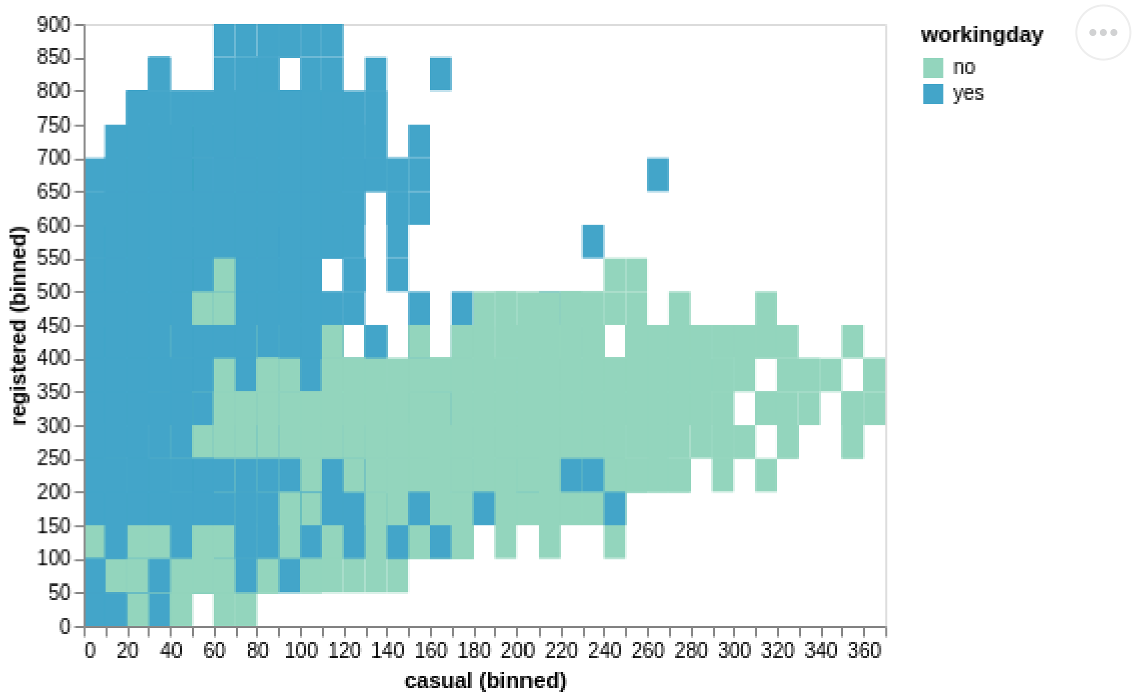
In this question we'll try visualizing the relationship between registered and casual riders using a heatmap. In Altair the heat map uses the mark_rect (https://altair-viz.github.io/gallery/simple_heatmap.html#gallery-simple-heatmap) property. Create a heatmap with causal riders on the x-axis and registered riders on the y-axis. Use any color scheme you like.

Save you chart in a variable named `rect` (which will be used in the next question) and display it.

This example (https://altair-viz.github.io/gallery/binned_heatmap.html) may be particularly helpful.

In [20]:
```
rect = alt.Chart(bike).mark_rect().encode(
    x=alt.X('casual:Q', bin=alt.Bin(maxbins=60)),
    y=alt.Y('registered:Q', bin=alt.Bin(maxbins=40)),
    color=alt.Color('workingday', scale=alt.Scale(scheme='greenblue'))
)

rect
```

Out[20]:



Question 3b

Now, let's see if we can create an interactive visualization around the heatmap we just drew. To do that we'll follow the iteractive cross highlight (https://altair-viz.github.io/gallery/interactive_cross_highlight.html) example in the documentation. In our example we'll make a bar plot indicating the total number of riders per season.

When complete, if you hold the shift key and click on the bars, you can select which observation appear in the heat map.

1. Add a column to `daily_counts` called `total` which is the total number of riders each day (sum of `registered` and `casual`).
2. Make a barplot called `bar_plot` , which has the total riders in each season.

   - Hint: the y encoding will need be the sum of total daily counts. See the documentation on useful encoding shortands (https://altair-viz.github.io/user_guide/encoding.html#encoding-shorthands)
   - Follow the interactive cross highilght example to set the color of the bars to change when you click on them.

Use the interactive visualization to quickly detect which season had the day with the most casual riders. Save your answer as either "spring", "summer", "winter" or "fall" in a variable calle `most_casual` .

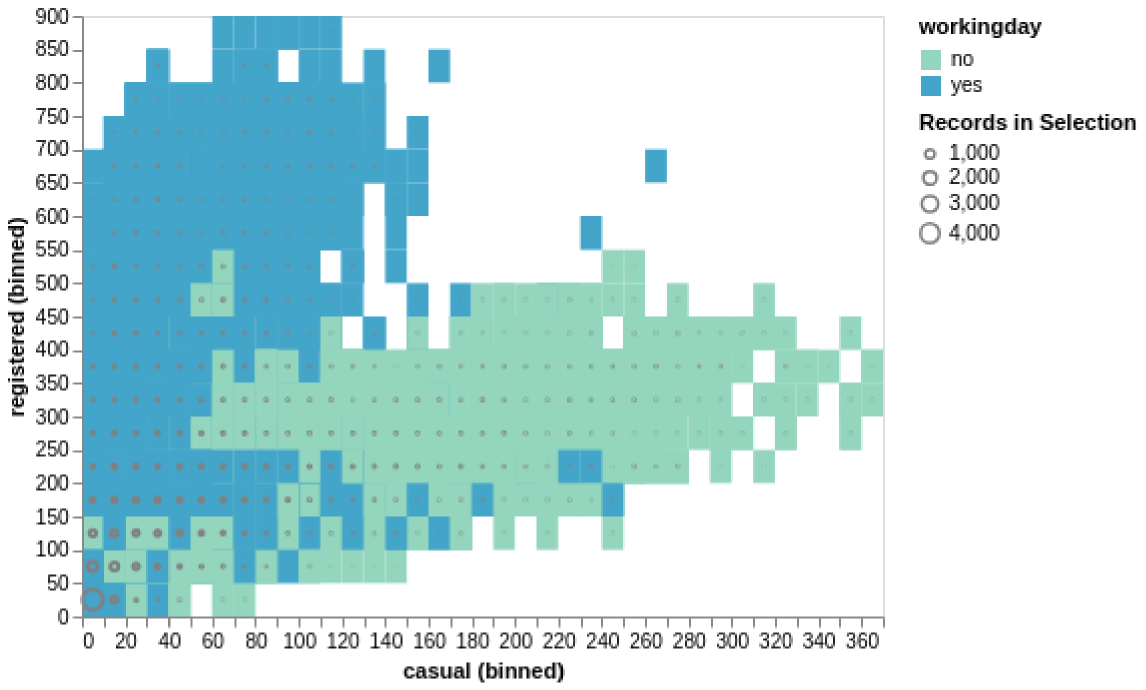
In [21]:
```
daily_counts['total'] = daily_counts['casual'] + daily_counts['registered']
daily_counts.head()
```

Out[21]:

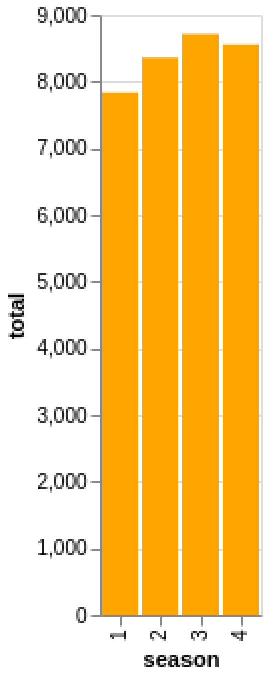
	casual	registered	workingday	weathersit	season	total
dteday						
2011-01-01	331	654	no	Clear	1	985
2011-01-02	131	670	no	Mist	1	801
2011-01-03	120	1229	yes	Clear	1	1349
2011-01-04	108	1454	yes	Clear	1	1562
2011-01-05	82	1518	yes	Clear	1	1600

In [22]:
```
# This creates a selection tool
pts = alt.selection(type="multi", encodings=['x'])

## This is a point plot where the size corresponds to the number of rid
ers in each rectangle
## We'll only show the circle if the relevant bar has been selected
## This is done with `transform_filter on out pts selection tool
circ = rect.mark_point().encode(
    alt.ColorValue('grey'),
    alt.Size('count()',
        legend=alt.Legend(title='Records in Selection')
    )
).transform_filter(
    pts
)

## Make a bar plot of total riders in each season and add the selection
 tool
bar_plot = alt.Chart(daily_counts).mark_bar().encode(
    x=alt.X('season:O'),
    y=alt.Y('total'),
    color=alt.condition(pts, alt.ColorValue("orange"), alt.ColorValue(
"purple"))
).add_selection(pts)

## Put all of the plots together.
interactive_plot = alt.vconcat(
    rect + circ,
    bar_plot
).resolve_legend(
    color="independent",
    size="independent"
)

# Use the interactive visualization to quickly see which season had the
 day with most casual riders
# You response should be either "spring", "summer", "winter" or "fall"
most_casual = "fall"

interactive_plot
```

Out[22]:



workingday
no
yes
Records in Selection
1,000
2,000
3,000
4,000
registered (binned)
casual (binned)
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
20
40
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360



total
season
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1
2
3
4

4: Understanding Daily Patterns

Question 4a

Let's examine the behavior of riders by plotting the average number of riders for each hour of the day over the **entire dataset**, stratified by rider type.

Your plot should look like the plot below. While we don't expect your plot's colors to match ours exactly, your plot should have different colored lines for different kinds of riders.

From the `bike` data frame, compute a new data frame called `hourly_means` with the average number of riders at each hour of the day.

Hint: By default Altair cannot include indices in the channel encodings. If `hr` is an index in your data frame, make it a column using the `reset_index` function. See the note on [including index data in Altair (https://altair-viz.github.io/user_guide/data.html#including-index-data)](https://altair-viz.github.io/user_guide/data.html#including-index-data).

In [23]:
```
bike.head()
```

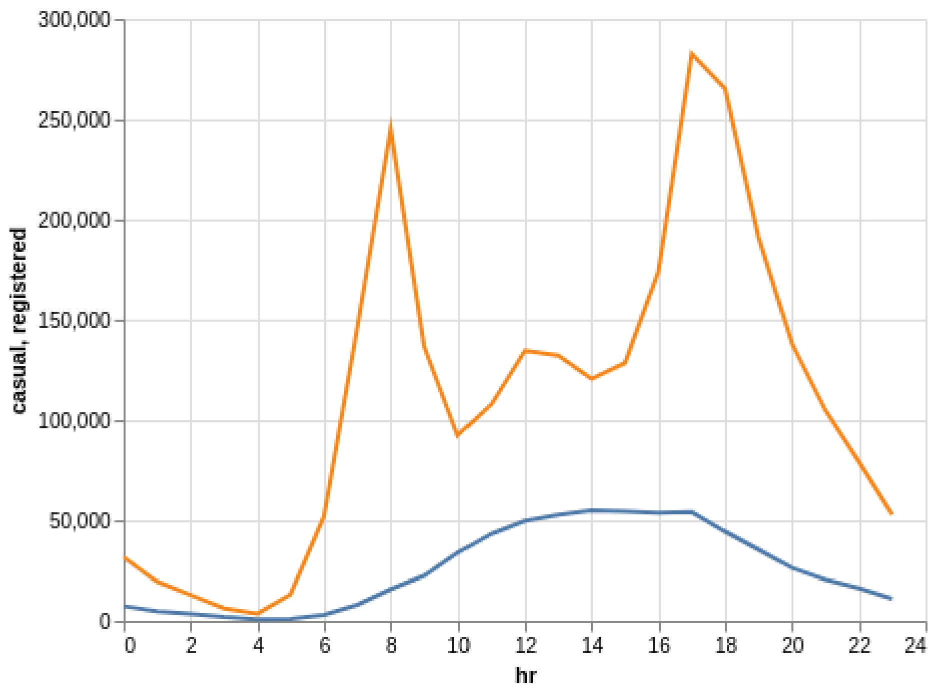
Out[23]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	ater
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24	0.28
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22	0.27
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22	0.27
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24	0.28
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24	0.28

In [74]:
```
hourly_means = bike.groupby(['hr']).agg({'casual':[sum],
                                          'registered':[sum],
                                          })
hourly_means.columns = ['casual', 'registered']
hourly_means = hourly_means.reset_index()
```

In [75]:
```
alt.Chart(pd.melt(hourly_means.reset_index(), id_vars=['hr'], value_var
s=['casual','registered'])).mark_line().encode(
    x=alt.X('hr'),
    y=alt.Y('value', title="casual, registered"),
    color=alt.Color('variable', legend=None),
)
```

Out[75]:



Question 4b

What can you observe from the plot? Hypothesize about the meaning of the peaks in the registered riders' distribution.

Peaks are around the time most people finish/start work - this is inline with my previous hypothesis that registered users use bikes mostly on work days.

the peak around 12-14 hours for the casual riders, combined with the previous trend of using the bikes on weekend, is inline with a nice afternoon weekend ride on a bike!

5: Exploring Ride Sharing and Weather

In this question we'll start examining how the weather is affecting rider's behavior.

Question 5a

We can quickly view relationships between many variables using what is often called a "pairs plot". In Altair they call this [repeated-charts (https://altair-viz.github.io/user_guide/compound_charts.html#repeated-charts)](https://altair-viz.github.io/user_guide/compound_charts.html#repeated-charts). Make a 4x4 grid of plots showing the relationship between the `casual` , `hum` , `windspeed` , and `temp` variables. Set the color of the points to the `season` variables. Do the number of casual riders seem related to weather? Are any of the weather variables related?

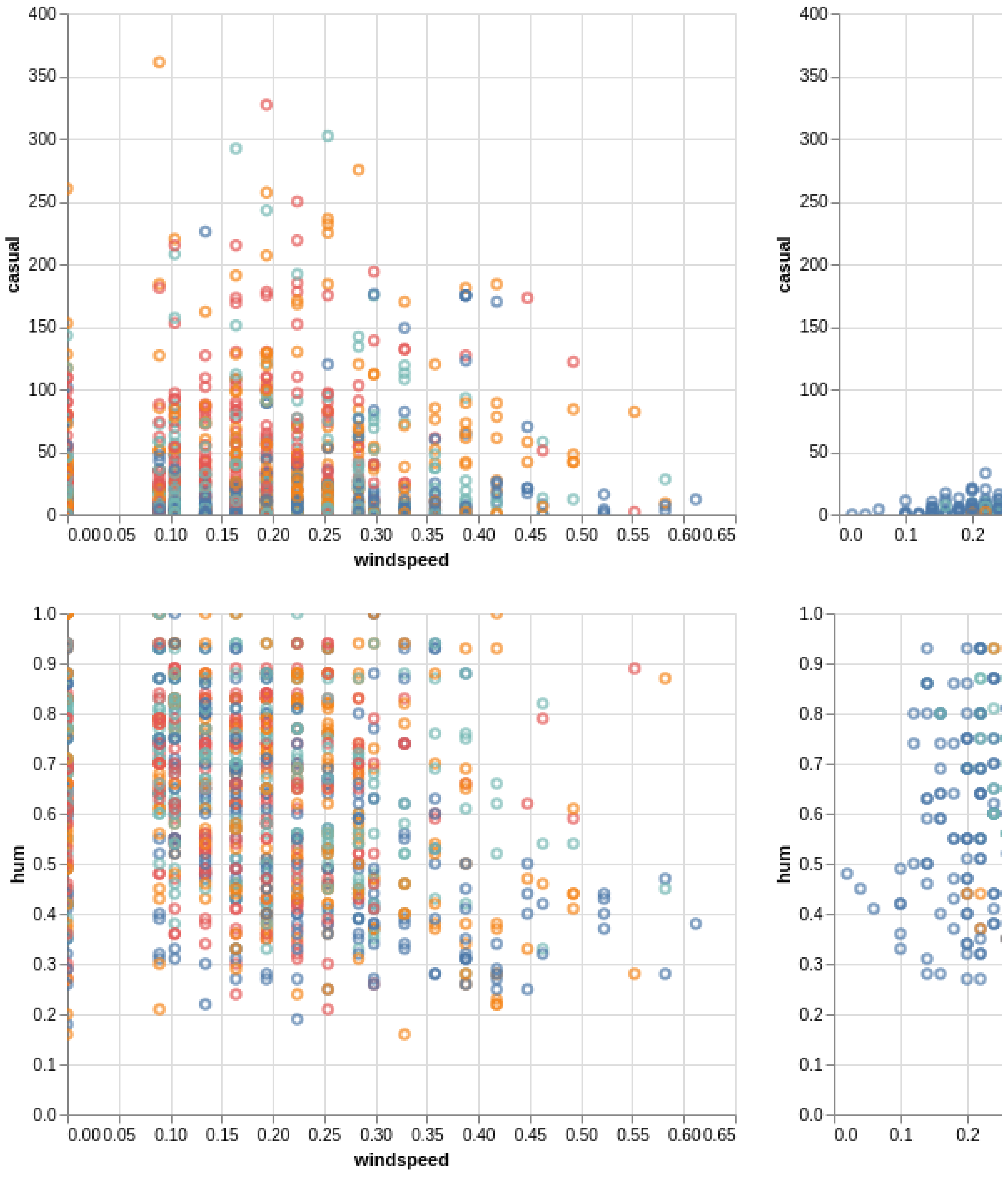
To limit the computation time and memory required to make the plot, make the plot using `bike_sample` , which is a random sample of 1000 rows of `bike` . You may want to first define `row_sample` as 1000 random row indices that you can use to make the selection.

In [ ]:

```

```

In [26]:
```
import altair as alt
import random

row_sample = [random.randint(0, 17379) for i in range(1000)]
bike_sample = bike.iloc[row_sample]

base = alt.Chart().mark_point().encode(
    color='season:N'
).interactive()

chart = alt.vconcat(data=bike_sample)
for y_encoding in ['casual:Q', 'hum:Q']:
    row = alt.hconcat()
    for x_encoding in ['windspeed:Q', 'temp:Q']:
        row |= base.encode(x=x_encoding, y=y_encoding)
    chart &= row
chart
```

Out[26]:



Question 5b

Next, let's look at how the proportion of casual riders changes as weather changes. Create a new column in the `bike` DataFrame calle `prop_casual` which represents the proportion of casual riders out of all riders for each record.

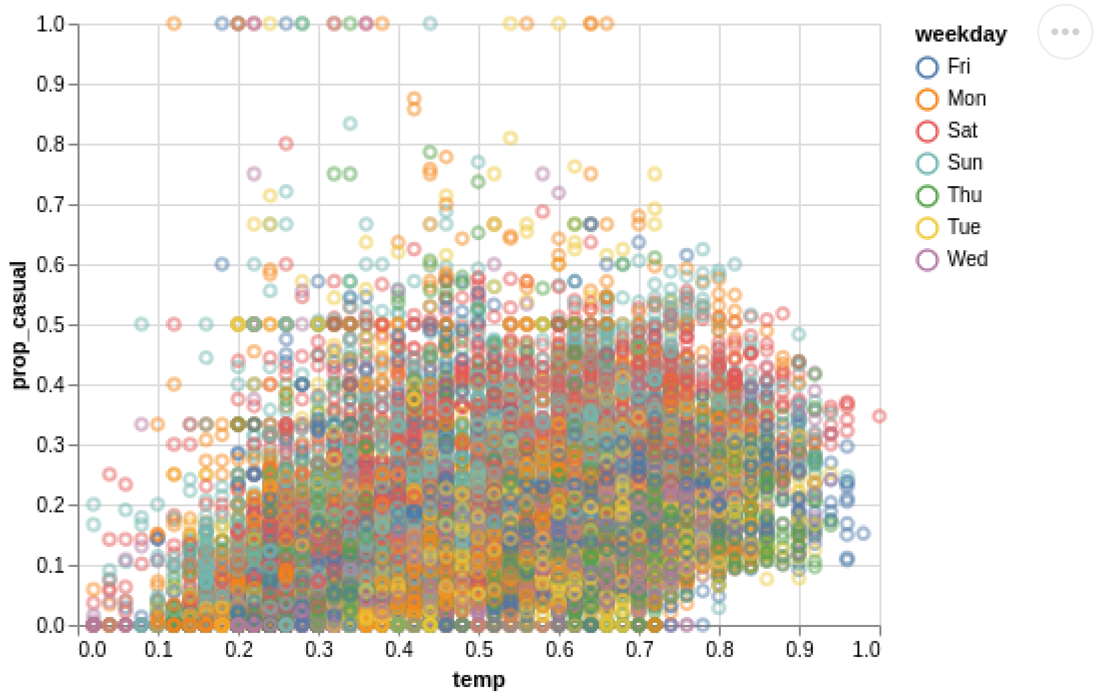
In [27]:
```
bike['prop_casual'] = bike['casual'] / (bike['casual'] + bike['register
ed'])
```

Question 5c

In order to examine the relationship between proportion of casual riders and temperature, we can again create a scatterplot using `mark_point`. We can even use color/hue to encode the information about day of week. Run the cell below, and you'll see we end up with a big mess that is impossible to interpret.

In [28]:
```
alt.Chart(bike).mark_point().encode(
     x='temp',
    y='prop_casual',
    color='weekday'
).configure_mark(
    opacity=0.5,
)
```

Out[28]:



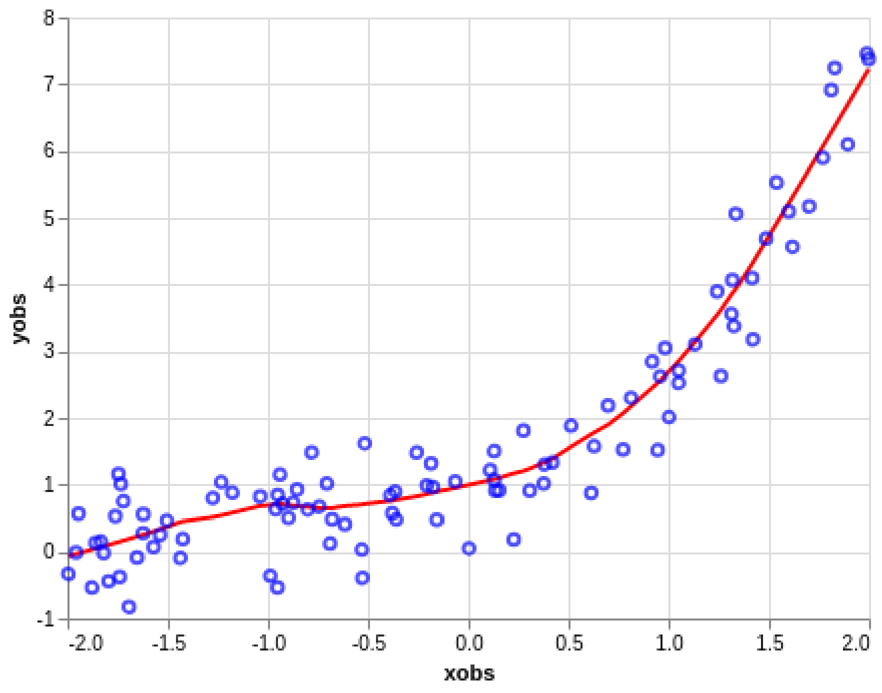
Question 5d

A better approach is to use local smoothing. The basic idea is that for each x value, we compute some sort of representative y value that captures the data close to that x value. One technique for local smoothing is LOESS transform (LOcally Estimated Scatterplot Smoothing), which is also known as "Locally Weighted Scatterplot Smoothing" (LOWESS).

See an example is below. The blue curve shown is a smoothed version of the scatterplot. To include the LOESS fit to the scatter plot, we can use the [transform_loess (https://altair-viz.github.io/user_guide/transform/loess.html)](https://altair-viz.github.io/user_guide/transform/loess.html) function in Altair.

In [29]:
```
# Make noisy data
xobs = np.sort(np.random.rand(100)*4.0 - 2)
noisy_data = pd.DataFrame({
    'xobs' : xobs,
    'yobs' : np.exp(xobs) + np.random.randn(100) / 2.0
})

chart = alt.Chart(noisy_data).transform_loess('xobs', 'yobs').mark_line
(size=2, color="red").encode(
    x='xobs',
    y='yobs'
)

chart + alt.Chart(noisy_data).mark_point(color="blue").encode(
    x = "xobs",
    y = "yobs"
)
```

Out[29]:



In our case with the bike ridership data, we want 7 curves, one for each day of the week. The x-axis will be the temperature and the y-axis will be a smoothed version of the proportion of casual riders.

You should use [transform_loess (https://altair-viz.github.io/user_guide/transform/loess.html)](https://altair-viz.github.io/user_guide/transform/loess.html) just like the example above. Unlike the example above, plot ONLY the lowess curve. Do not plot the actual data points, which would result in a mess of points covering the smooth line fits.

You do not need to match the colors on our sample plot as long as the colors in your plot make it easy to distinguish which day they represent.

Set the plot title to "Temperature vs Casual Rider Proportion by Weekday".

Hints:

- Start by creating a new column in the `bike` data frame, called `temp_f`. Look at the top of this homework notebook for a description of the temperature field to know how to convert to Fahrenheit. By default, the temperature field ranges from 0.0 to 1.0. In case you need it, $\text{Fahrenheit} = \text{Celsius} * \frac{9}{5} + 32$.

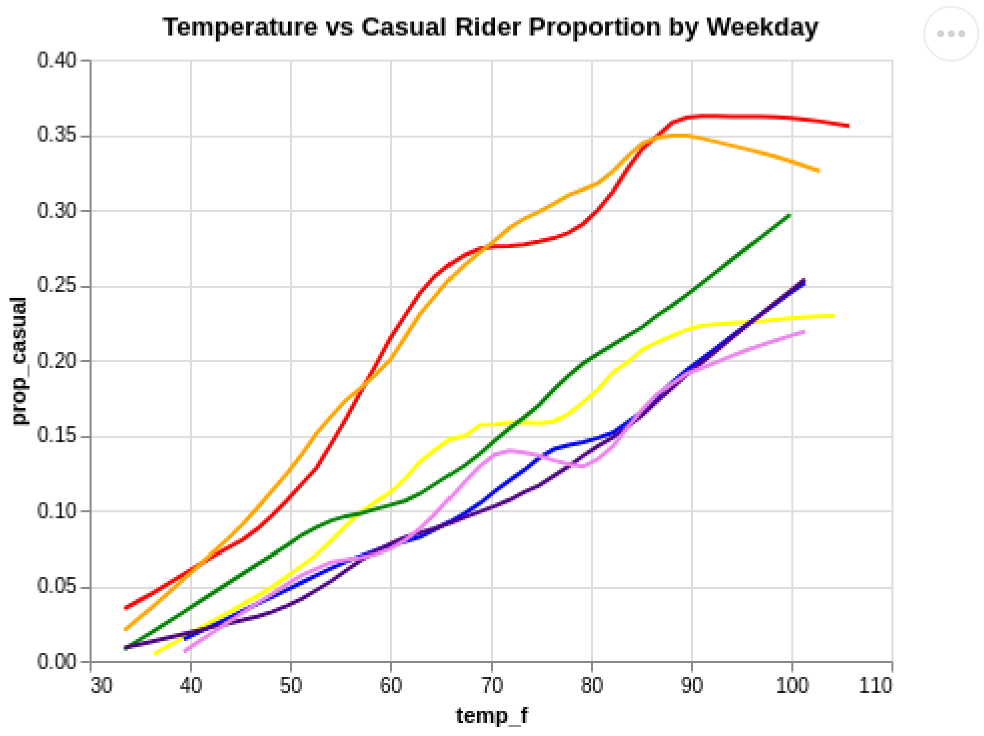
```
In [30]: bike.head()
```

Out[30]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	ater
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24	0.28
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22	0.27
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22	0.27
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24	0.28
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24	0.28

In [31]:
```
bike['temp_f'] = (bike['temp']*(41))*(9/5) + 32

bike_fri = bike[bike['weekday'] == 'Fri']
bike_sat = bike[bike['weekday'] == 'Sat']
bike_sun = bike[bike['weekday'] == 'Sun']
bike_mon = bike[bike['weekday'] == 'Mon']
bike_tue = bike[bike['weekday'] == 'Tue']
bike_wed = bike[bike['weekday'] == 'Wed']
bike_thu = bike[bike['weekday'] == 'Thu']

chart_fri = alt.Chart(bike_fri).transform_loess('temp_f', 'prop_casual'
).mark_line(size=2, color="yellow").encode(
    x='temp_f',
    y='prop_casual',
)
chart_sat = alt.Chart(bike_sat).transform_loess('temp_f', 'prop_casual'
).mark_line(size=2, color="red").encode(
    x='temp_f',
    y='prop_casual',
)
chart_sun = alt.Chart(bike_sun).transform_loess('temp_f', 'prop_casual'
).mark_line(size=2, color="orange").encode(
    x='temp_f',
    y='prop_casual',
)
chart_mon = alt.Chart(bike_mon).transform_loess('temp_f', 'prop_casual'
).mark_line(size=2, color="green").encode(
    x='temp_f',
    y='prop_casual',
)
chart_tue = alt.Chart(bike_tue).transform_loess('temp_f', 'prop_casual'
).mark_line(size=2, color="blue").encode(
    x='temp_f',
    y='prop_casual',
)
chart_wed = alt.Chart(bike_wed).transform_loess('temp_f', 'prop_casual'
).mark_line(size=2, color="indigo").encode(
    x='temp_f',
    y='prop_casual',
)
chart_thu= alt.Chart(bike_thu).transform_loess('temp_f', 'prop_casual')
.mark_line(size=2, color="violet").encode(
    x='temp_f',
    y='prop_casual',
)

alt.layer(chart_fri, chart_sat, chart_sun, chart_mon,
         chart_tue, chart_wed, chart_thu ).properties(
    title="Temperature vs Casual Rider Proportion by Weekday")
```

Out[31]:



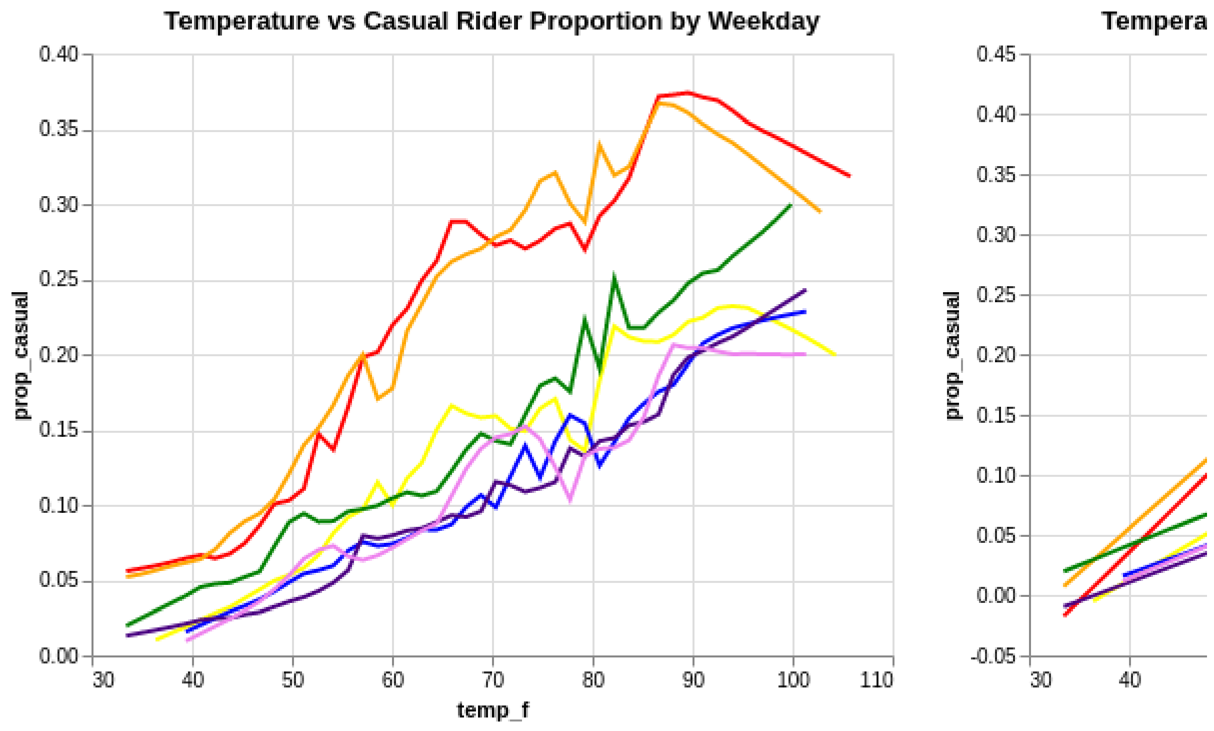
Question 5e

Repeat the above plot, but this change the `bandwith` argument for the `transform_loess` function. The bandwidth parameter controls the smoothness of the interpolating line, with smaller numbers leading to less smoothness and larger numbers to more smoothness. Create one chart with `bandwith=0.1` (call it `chart1`) and another with `bandwidth=0.8` (call it `chart8`). [Use the `|` operator to plot `chart1` and `chart8` side by side. (https://altair-viz.github.io/user_guide/compound_charts.html#horizontal-concatenation)](https://altair-viz.github.io/user_guide/compound_charts.html#horizontal-concatenation)

In [32]:
```
chart_fri = alt.Chart(bike_fri).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.1).mark_line(size=2, color="yellow").encode(
    x='temp_f',
    y='prop_casual',
)
chart_sat = alt.Chart(bike_sat).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.1).mark_line(size=2, color="red").encode(
    x='temp_f',
    y='prop_casual',
)
chart_sun = alt.Chart(bike_sun).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.1).mark_line(size=2, color="orange").encode(
    x='temp_f',
    y='prop_casual',
)
chart_mon = alt.Chart(bike_mon).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.1).mark_line(size=2, color="green").encode(
    x='temp_f',
    y='prop_casual',
)
chart_tue = alt.Chart(bike_tue).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.1).mark_line(size=2, color="blue").encode(
    x='temp_f',
    y='prop_casual',
)
chart_wed = alt.Chart(bike_wed).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.1).mark_line(size=2, color="indigo").encode(
    x='temp_f',
    y='prop_casual',
)
chart_thu= alt.Chart(bike_thu).transform_loess('temp_f', 'prop_casual',
 bandwidth=0.1).mark_line(size=2, color="violet").encode(
    x='temp_f',
    y='prop_casual',
)

chart_fri1 = alt.Chart(bike_fri).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="yellow").encode(
    x='temp_f',
    y='prop_casual',
)
chart_sat1 = alt.Chart(bike_sat).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="red").encode(
    x='temp_f',
    y='prop_casual',
)
chart_sun1 = alt.Chart(bike_sun).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="orange").encode(
    x='temp_f',
    y='prop_casual',
)
chart_mon1 = alt.Chart(bike_mon).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="green").encode(
    x='temp_f',
    y='prop_casual',
)
```

```
chart_tue1 = alt.Chart(bike_tue).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="blue").encode(
    x='temp_f',
    y='prop_casual',
)
chart_wed1 = alt.Chart(bike_wed).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="indigo").encode(
    x='temp_f',
    y='prop_casual',
)
chart_thu1 = alt.Chart(bike_thu).transform_loess('temp_f', 'prop_casua
l', bandwidth=0.8).mark_line(size=2, color="violet").encode(
    x='temp_f',
    y='prop_casual',
)
chart1 = alt.layer(chart_fri, chart_sat, chart_sun, chart_mon,
         chart_tue, chart_wed, chart_thu).properties(
    title="Temperature vs Casual Rider Proportion by Weekday")
chart8 =  alt.layer(chart_fri1, chart_sat1, chart_sun1, chart_mon1,
         chart_tue1, chart_wed1, chart_thu1).properties(
    title="Temperature vs Casual Rider Proportion by Weekday")

chart1 | chart8
```

Out[32]:



Question 5f

The default bandwidth for `transform_loess` is 0.3. Among the 0.1, 0.3 and 0.8, which amount of smoothness was most informative about the relationship between casual riders and temperature? Why? In your answer, you should try to address which patterns in the lowess fits you think are robust or likely to be repeated in future years.

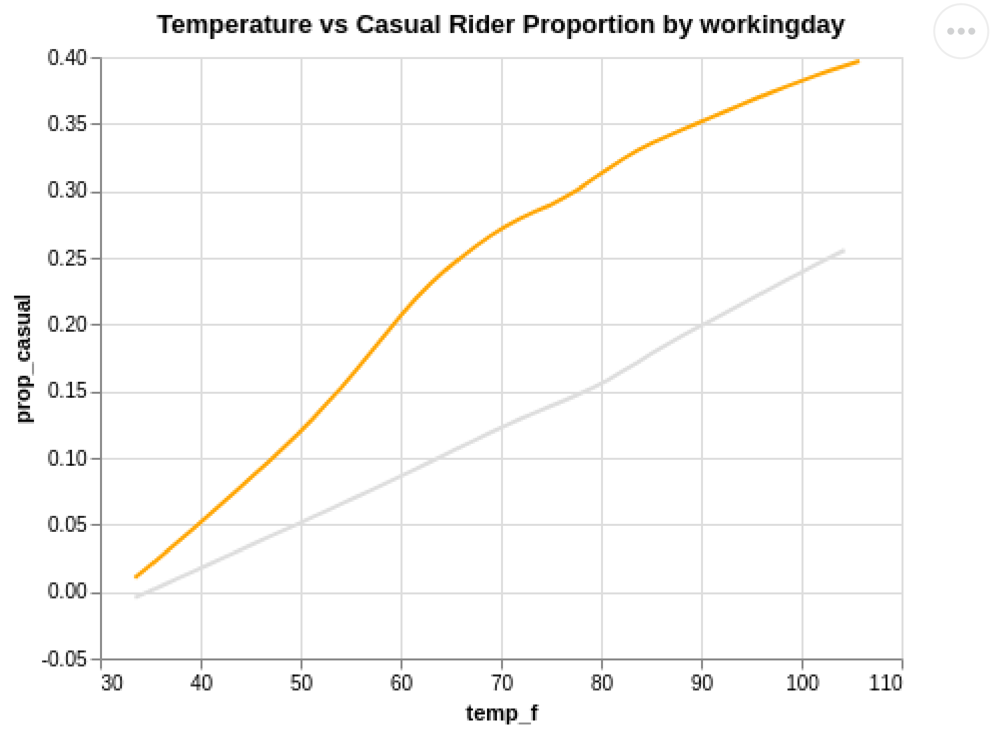
I would say 0.3. It has a healthy mix between showing the proper trend while filtering out the randomness/noise that you can see with 0.8. With 0.1 you lose too much information, but you can see the overall trend clearly, so that is a good graph too.

Question 5g

Remake the above plot, this time grouping by `workingday` rather than `weekday`. Use a bandwidth of your choice.

In [33]:
```
bike_yes = bike[bike['workingday'] == 'yes']
bike_no = bike[bike['workingday'] == 'no']

chart_yes = alt.Chart(bike_yes).transform_loess('temp_f', 'prop_casual'
, bandwidth=0.5).mark_line(size=2, color="puple").encode(
    x='temp_f',
    y='prop_casual',
).properties(
    title="Temperature vs Casual Rider Proportion by workingday")
chart_no = alt.Chart(bike_no).transform_loess('temp_f', 'prop_casual',
bandwidth=0.5).mark_line(size=2, color="orange").encode(
    x='temp_f',
    y='prop_casual',
).properties(
    title="Temperature vs Casual Rider Proportion by workingday")

alt.layer(chart_yes, chart_no)
```

Out[33]:



In [34]:
```
bike.head()
bike_no
```

Out[34]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24
...	...	...	...	...	...	...	...	...	...	...	...
17350	17351	2012-12-30	1	1	12	19	no	Sun	no	Clear	0.34
17351	17352	2012-12-30	1	1	12	20	no	Sun	no	Clear	0.22
17352	17353	2012-12-30	1	1	12	21	no	Sun	no	Clear	0.20
17353	17354	2012-12-30	1	1	12	22	no	Sun	no	Clear	0.20
17354	17355	2012-12-30	1	1	12	23	no	Sun	no	Clear	0.20

5514 rows × 19 columns

Question 5h

What do you see from the plots above? How is `prop_casual` changing as a function of temperature? Do you notice anything else interesting? Which plot do you think most clearly tells the story about bike share use and temperature and days of the week. Your answer should the plots the previous three parts.

both plots above show that with increasing tempereature, proportion of casual riders increases.

I think the plot we made in 5d (the first one) tells the story pretty perfectly. Not only can you see the increasing trend, but you see that at some point the proportion plateaus and even decreases, so basically there is a "sweet spot" temperature that motivates casual riders to pick up a bike.

Question 6 Expanding our Analysis

Question 6a

Imagine you are working for a Bike Sharing Company that collaborates with city planners, transportation agencies, and policy makers in order to implement bike sharing in a city. These stakeholders would like to reduce congestion and lower transportation costs. They also want to ensure the bike sharing program is implemented equitably. In this sense, equity is a social value that is informing the deployment and assessment of your bike sharing technology.

Equity in transportation includes: improving the ability of people of different socio-economic classes, genders, races, and neighborhoods to access and afford the transportation services, and assessing how inclusive transportation systems are over time.

Do you think the `bike` data as it is can help you assess equity? If so, please explain. If not, how would you change the dataset? You may discuss how you would change the granularity, what other kinds of variables you'd introduce to it, or anything else that might help you answer this question.

I think the fact that the proportion of casual riders varies is promising. it shows the transportation system is flexible and people with different motivations are able to access it when they feel they want to. it is an indicator of variability in the democraphic. on the other hand, we need more specific data to be certain of our deductions. for example, age and income, average distance travelled/money paid.

one important feature that could shine light on age,income,av.distance is location. if people in locations that are known to house lower-income people actively use the platform, then it would be a strong indicator that a wide demographic can access this form of transportation. end location could also be tracked, to see if people are going to a corporate area, a school, a hospital etc, shedding more light on the diversity of the people using the system.

```
In [35]: # Use this cell for scratch work. If you need to add more cells for scr
         atch work, add them BELOW this cell.
```

Question 6b

[Bike sharing is growing in popularity (https://www.bts.gov/newsroom/bike-share-stations-us)](https://www.bts.gov/newsroom/bike-share-stations-us) and new cities and regions are making efforts to implement bike sharing systems that complement their other transportation offerings. The [goals of these efforts (https://www.wired.com/story/americans-falling-in-love-bike-share/)](https://www.wired.com/story/americans-falling-in-love-bike-share/) are to have bike sharing serve as an alternate form of transportation in order to alleviate congestion, provide geographic connectivity, reduce carbon emissions, and promote inclusion among communities.

Bike sharing systems have spread to many cities across the country. The company you work for asks you to determine the feasibility of expanding bike sharing to additional cities of the U.S.

Based on your plots in this assignment, what would you recommend and why? Please list at least two reasons why, and mention which plot(s) you drew you analysis from.

Note: There isn't a set right or wrong answer for this question, feel free to come up with your own conclusions based on evidence from your plots!

I would recommend access points to the bikes close to residential hotspots and corporate hotspots. It seems as if people are using this bikes regularly to travel from home to work and back.

During fall, and warmer weather, the availability of bikes and diversity of locations should be increase. During these seasons casual users start using the bikes much more, introducing more variability and demand.

In [36]:
```
# Use this cell for scratch work. If you need to add more cells for scr
atch work, add them BELOW this cell.
```

Submit

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. **Please save before submitting!**

Running Built-in Tests

1. All tests are in `tests` directory
2. Each python file in `tests` is a test
3. `grader.check('testname')` runs test `'testname'` , e.g. `'q1'`
4. `grader.check_all()` runs all visible tests

In [76]:
```
# Run built-in checks
grader.check_all()
```

q1a

All tests passed!

q1b

All tests passed!

q1c

All tests passed!

q2c

All tests passed!

q3a

All tests passed!

q3b

All tests passed!

q4a

All tests passed!

q5a

All tests passed!

q5b

All tests passed!

In [ ]:
```
# Generate pdf in classic notebook (does not work in JupyterLab)
import nb2pdf
nb2pdf.convert('hw3.ipynb')

# To generate pdf using command-line, run in terminal,
# nb2pdf hw3.ipynb
```

Submission Checklist

1. Check filename is 'hw3.ipynb'
2. Save file to confirm all changes are on disk
3. Run *Kernel > Restart & Run All* to execute all code from top to bottom
4. Check `grader.check_all()` output
5. Save file again to write any new output to disk
6. Check generated pdf that all responses are displayed correctly
7. Submit to Gradescope