Instructions

The assignment is at the bottom!

This cell automatically downloads Capital Bikeshare data

And here we read in the data

In [1]:
```
# !pip install seaborn
```

In [2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = 20, 10
import pandas as pd
import numpy as np
bikes = pd.read_csv('../data/bikeshare.csv.gz')
bikes.head()
bikes['start'] = pd.to_datetime(bikes['Start date'], infer_datetime_format=True)
bikes['end'] = pd.to_datetime(bikes['End date'], infer_datetime_format=True)
bikes["dur"] = (bikes['Duration (ms)']/1000).astype(int)
bikes.head()
```

Out[2]:

	Duration (ms)	Start date	End date	Start station number	Start station	End station number	End station	Bike number	Member Type	s
0	301295	3/31/2016 23:59	4/1/2016 0:04	31280	11th & S St NW	31506	1st & Rhode Island Ave NW	W00022	Registered	20 03 23:59
1	557887	3/31/2016 23:59	4/1/2016 0:08	31275	New Hampshire Ave & 24th St NW	31114	18th St & Wyoming Ave NW	W01294	Registered	20 03 23:59
2	555944	3/31/2016 23:59	4/1/2016 0:08	31101	14th & V St NW	31221	18th & M St NW	W01416	Registered	20 03 23:59
3	766916	3/31/2016 23:57	4/1/2016 0:09	31226	34th St & Wisconsin Ave NW	31214	17th & Corcoran St NW	W01090	Registered	20 03 23:57
4	139656	3/31/2016 23:57	3/31/2016 23:59	31011	23rd & Crystal Dr	31009	27th & Crystal Dr	W21934	Registered	20 03 23:57

In [3]:
```
bikes.dur.mean()
```

Out[3]:
```
992.8716543657755
```

In [4]:
```
bikes.dur.std()
```

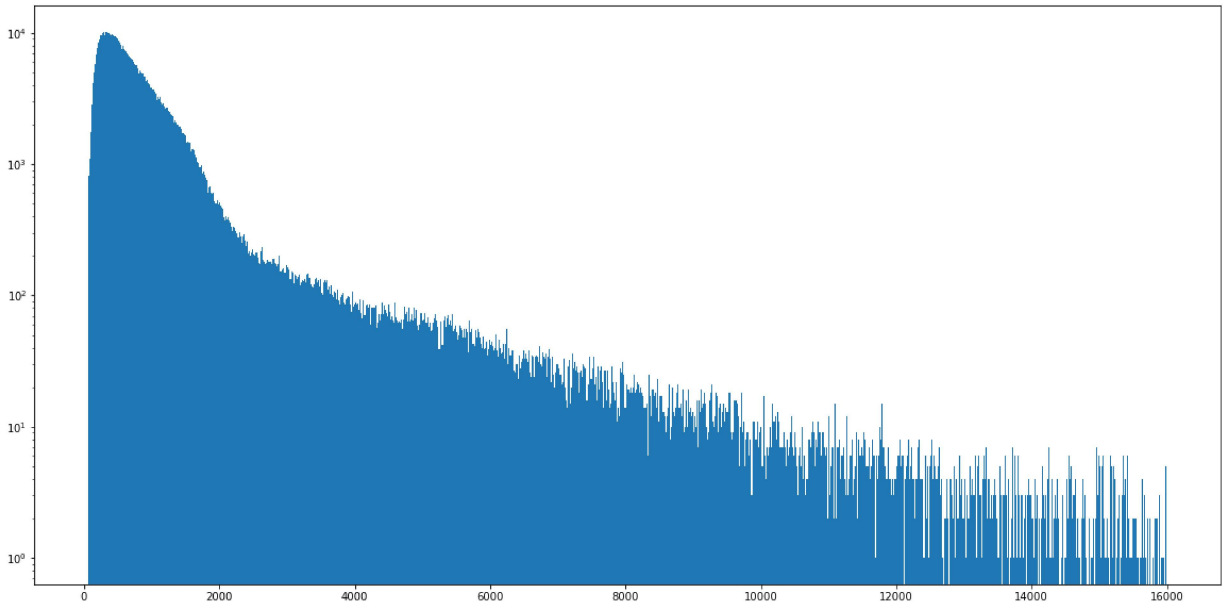
Out[4]:
```
2073.9809135296514
```

In [5]:
```
bikes[bikes.dur>16000].shape
```

Out[5]:
```
(973, 12)
```

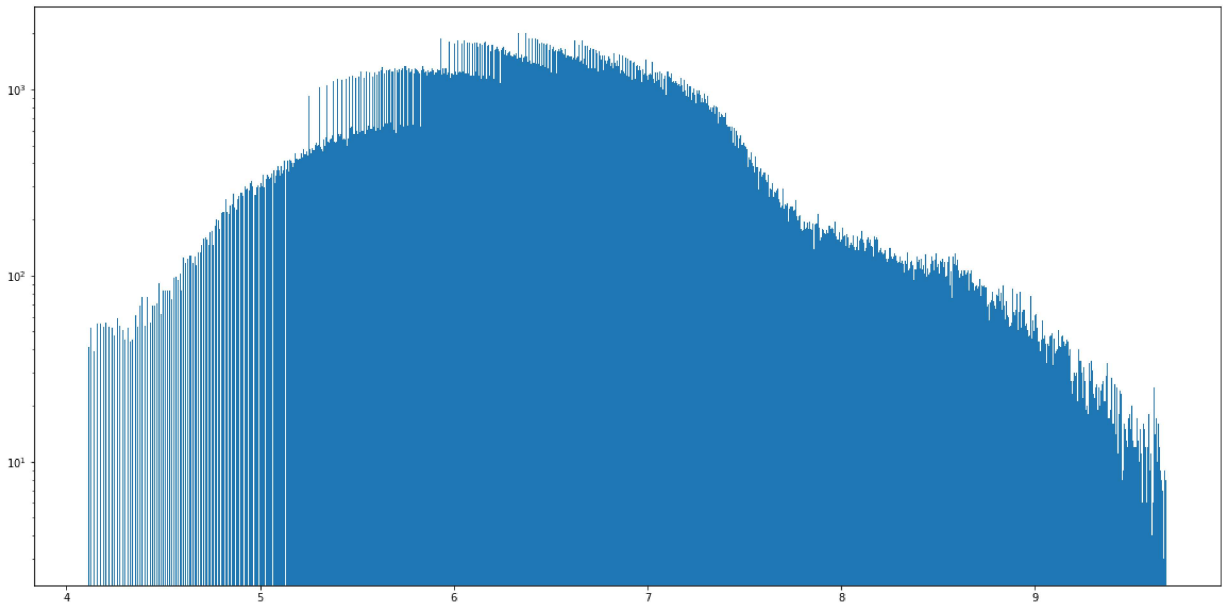
In [6]:
```
plt.rcParams['figure.figsize'] = 20, 10
```

In [7]:
```
_=plt.hist(bikes[bikes.dur<16000].dur, log=True, bins=1000)
```



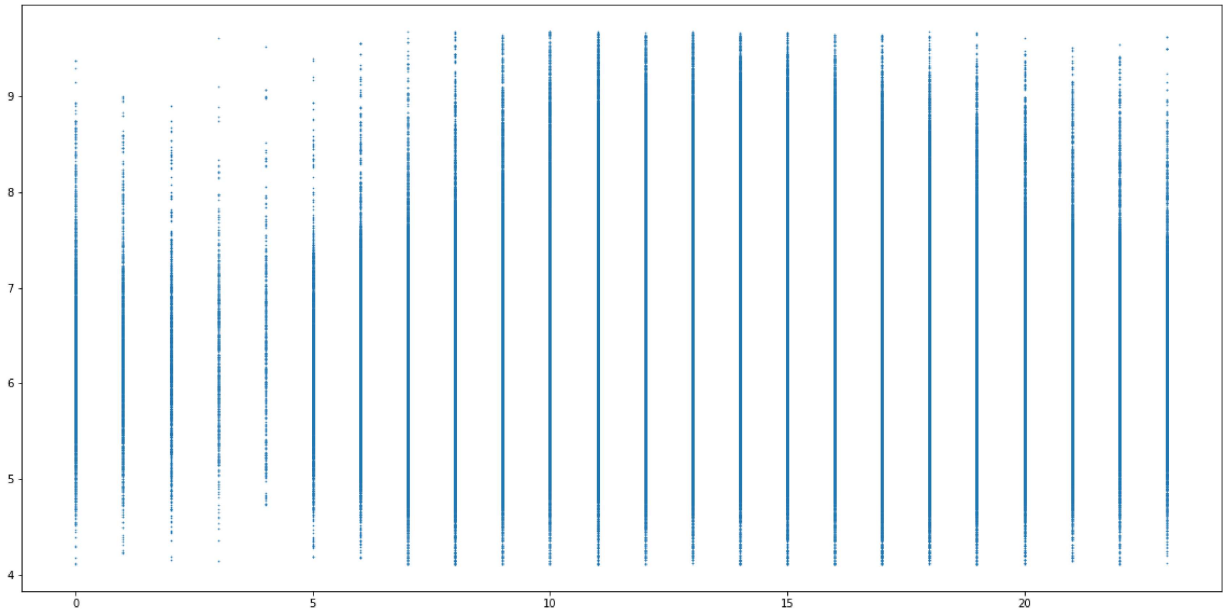
In [8]:
```
short = bikes[bikes.dur<16000]
```

In [9]:
```
_=plt.hist(np.log1p(short.dur), log=True, bins=1000)
```



In [10]:
```
plt.scatter(short.start.dt.hour, np.log1p(short.dur), s=.4)
```

Out[10]:
```
<matplotlib.collections.PathCollection at 0x243ceb24888>
```



In [11]:
```
np.log1p(0), np.log(0)
```

```
C:\Users\gdlev\anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarn
ing: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

Out[11]:
```
(0.0, -inf)
```

In [12]:
```
bikes['log_dur'] = np.round(np.log1p(bikes.dur), 1)
```

In [13]:
```
monday = bikes[bikes.start.dt.dayofweek==1]
```

In [14]:
```
dur_hour = monday.groupby(['log_dur', monday.start.dt.hour]).count()
```

In [15]:
```
dur_hour
```

Out[15]:

		Duration (ms)	Start date	End date	Start station number	Start station	End station number	End station	Bike number	Member Type	start
log_dur	start										
4.1	7	1	1	1	1	1	1	1	1	1	1
	9	2	2	2	2	2	2	2	2	2	2
	11	1	1	1	1	1	1	1	1	1	1
	14	2	2	2	2	2	2	2	2	2	2
	16	2	2	2	2	2	2	2	2	2	2
...	...	...	...	...	...	...	...	...	...	...	...
11.2	21	2	2	2	2	2	2	2	2	2	2
11.3	14	1	1	1	1	1	1	1	1	1	1
	17	1	1	1	1	1	1	1	1	1	1

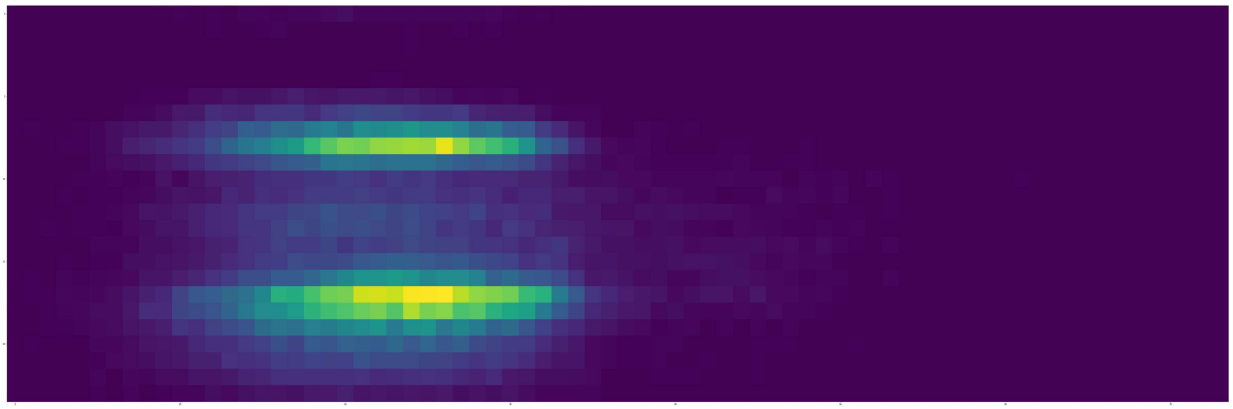
In [16]:
```
duration_hour = dur_hour.start.unstack().T.fillna(0)
duration_hour
```

Out[16]:

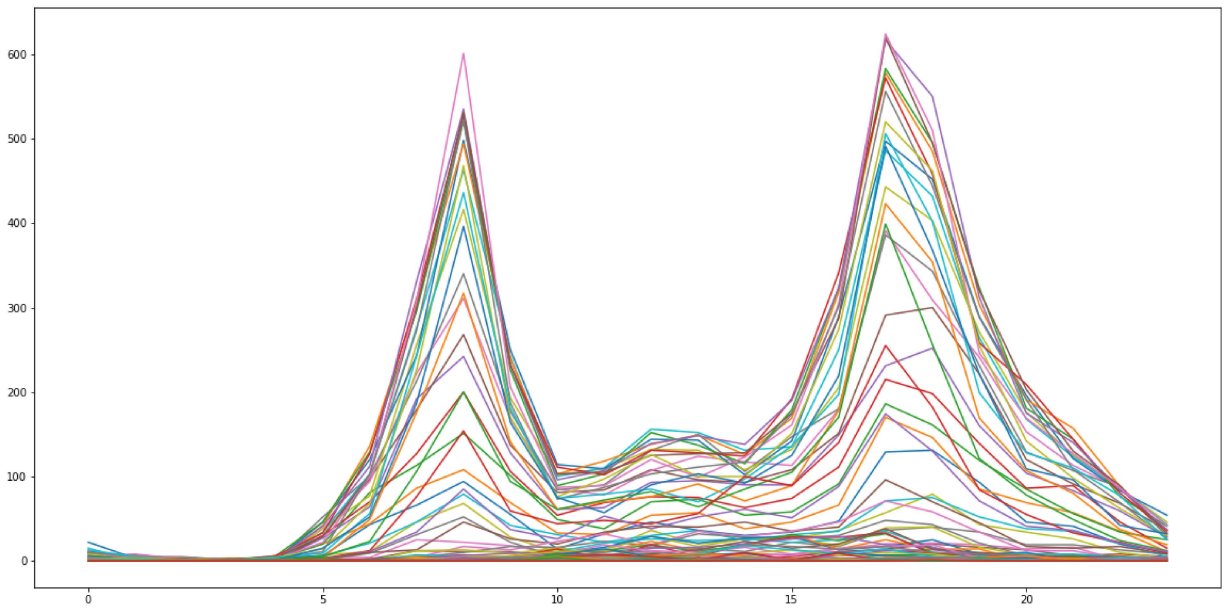
log_dur	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	5.0	...	10.5	10.6	10.7	10.8	10.9
start																
0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	2.0	3.0	...	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	3.0	1.0	...	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	1.0	0.0	0.0	1.0	4.0	1.0	7.0	6.0	...	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	2.0	1.0	2.0	4.0	9.0	11.0	21.0	...	0.0	0.0	0.0	1.0	0.0
7	1.0	5.0	4.0	1.0	5.0	12.0	25.0	31.0	46.0	46.0	...	0.0	1.0	1.0	0.0	0.0
8	0.0	3.0	2.0	6.0	7.0	11.0	22.0	52.0	68.0	79.0	...	4.0	2.0	1.0	0.0	0.0
9	2.0	3.0	2.0	4.0	3.0	11.0	18.0	22.0	28.0	42.0	...	1.0	1.0	0.0	0.0	0.0

In [17]:
```
plt.figure(figsize=(100,100))
plt.imshow(duration_hour)
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x243cec70708>
```



In [18]:
```
_=plt.plot(duration_hour)
```



In [19]:
```
bikes['Member Type'].value_counts()
```

Out[19]:
```
Registered    467432
Casual         84967
Name: Member Type, dtype: int64
```

Create a new column that represents the hour+minute of the day as a fraction (i.e. 1:30pm = 13.5)

In [20]:
```
np.round(.65, 1)
```

Out[20]:
```
0.6
```

In [21]:
```
37//6, (37//6)/10, 37/60
```

Out[21]:
```
(6, 0.6, 0.6166666666666667)
```

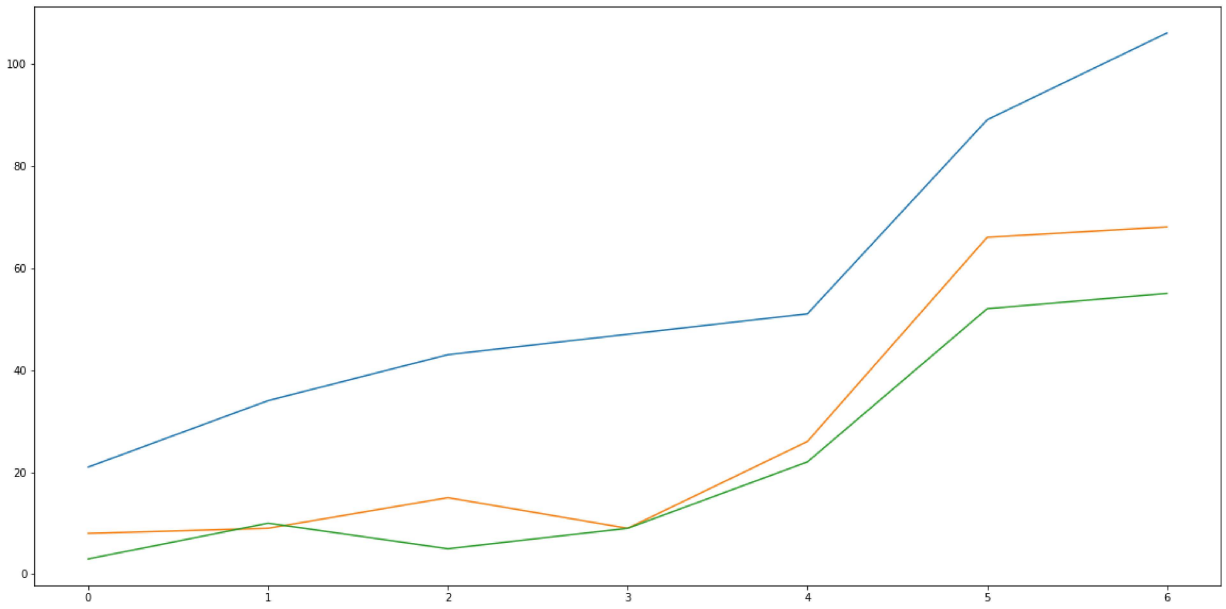
In [22]:
```
bikes['hour_of_day'] = (bikes.start.dt.hour + (bikes.start.dt.minute//6)/10)
```

In [23]:
```
bikes['roundhour_of_day'] = (bikes.start.dt.hour ) # keep the hour handy as well
```

Aggregate to get a count per hour/minute of the day across all trips

In [24]:
```
# reg_bikes = bikes[bikes['Member Type']=='Registered']
# hours = reg_bikes.groupby([reg_bikes.hour_of_day, reg_bikes.start.dt.dayofweek]
# hours['hour'] = hours.index

reg_bikes = bikes[bikes['Member Type']=='Registered']
hours = reg_bikes.groupby([reg_bikes.start.dt.dayofweek,reg_bikes.hour_of_day]).c
hours_2 = hours.start.unstack().T.fillna(0)
hours_2['hour'] = hours_2.index


day_hour_count = hours.dur.unstack()
plt.figure(figsize=(20,10))
plt.plot(day_hour_count.index, day_hour_count[0])
plt.plot(day_hour_count.index, day_hour_count[1])
plt.plot(day_hour_count.index, day_hour_count[2])
# plt.plot(y.index, day_hour_count[3])
# plt.plot(y.index, day_hour_count[4])
# plt.plot(y.index, day_hour_count[5])
# plt.plot(y.index, day_hour_count[6])
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x243ced843c8>]
```



In [25]:
```
day_hour_count
```

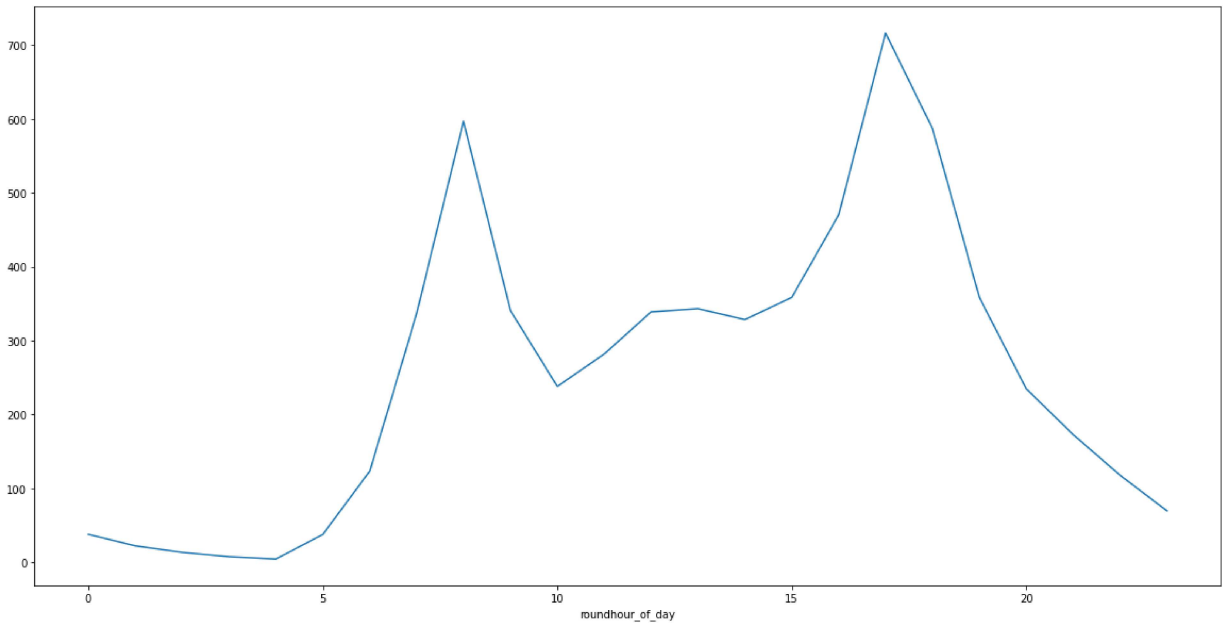
Out[25]:

hour_of_day	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	...	23.0	23.1	23.2
start														
0	21.0	39.0	31.0	26.0	19.0	16.0	21.0	16.0	11.0	11.0	...	82.0	71.0	74.0
1	34.0	22.0	24.0	27.0	24.0	23.0	14.0	10.0	8.0	8.0	...	111.0	115.0	62.0
2	43.0	27.0	26.0	25.0	29.0	20.0	22.0	17.0	13.0	6.0	...	111.0	132.0	105.0
3	47.0	37.0	42.0	29.0	29.0	25.0	25.0	17.0	17.0	15.0	...	140.0	122.0	121.0
4	51.0	56.0	50.0	52.0	50.0	52.0	42.0	32.0	27.0	33.0	...	122.0	93.0	99.0
5	89.0	87.0	98.0	99.0	98.0	70.0	58.0	64.0	71.0	66.0	...	117.0	108.0	121.0
6	106.0	100.0	77.0	87.0	69.0	58.0	83.0	61.0	61.0	63.0	...	86.0	61.0	46.0

7 rows × 240 columns

In [26]:
```
hoursn = bikes.groupby('roundhour_of_day').agg('count')
hoursn['hour'] = hoursn.index
(hoursn.start/90).plot() # 90 days in a quarter
```

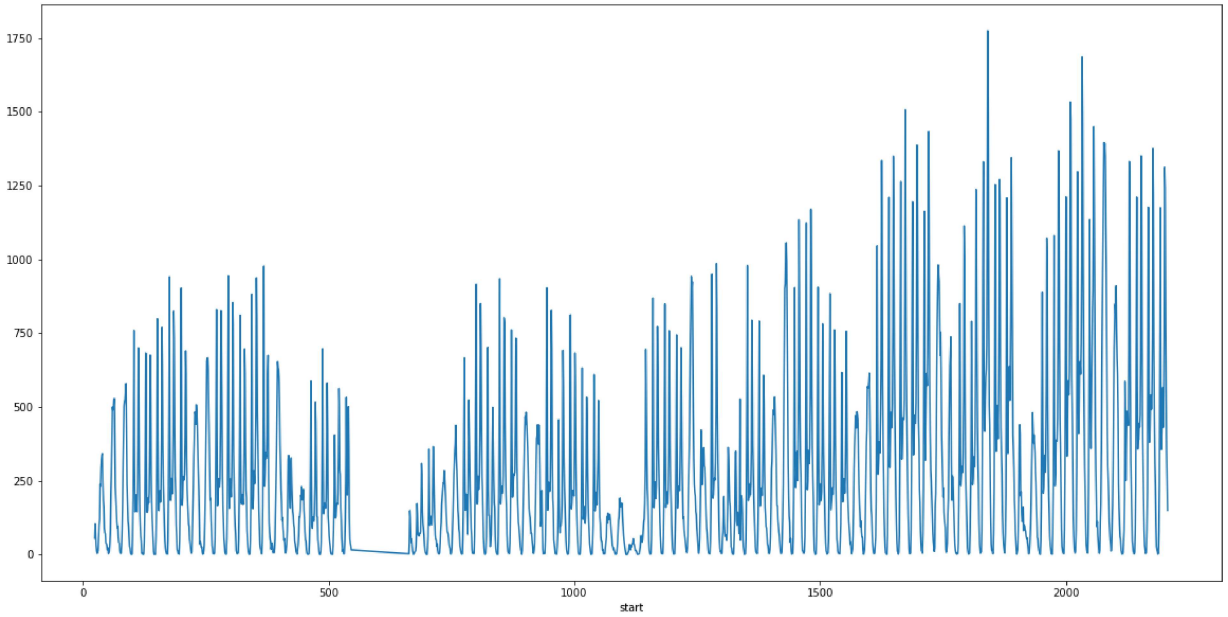
Out[26]:
```
<AxesSubplot:xlabel='roundhour_of_day'>
```



In [27]:
```
hour_count = bikes.groupby(bikes.start.dt.dayofyear*24 + bikes.start.dt.hour).cou
```

In [28]:
```
plt.figure(figsize=(20,10))
hour_count.start.plot()
```

Out[28]: <AxesSubplot:xlabel='start'>



In [29]:
```
day_count = bikes.groupby(bikes.start.dt.dayofyear).count()
```

In [30]:
```
day_hour = bikes.groupby([bikes.start.dt.dayofyear, bikes.start.dt.hour]).count()
```

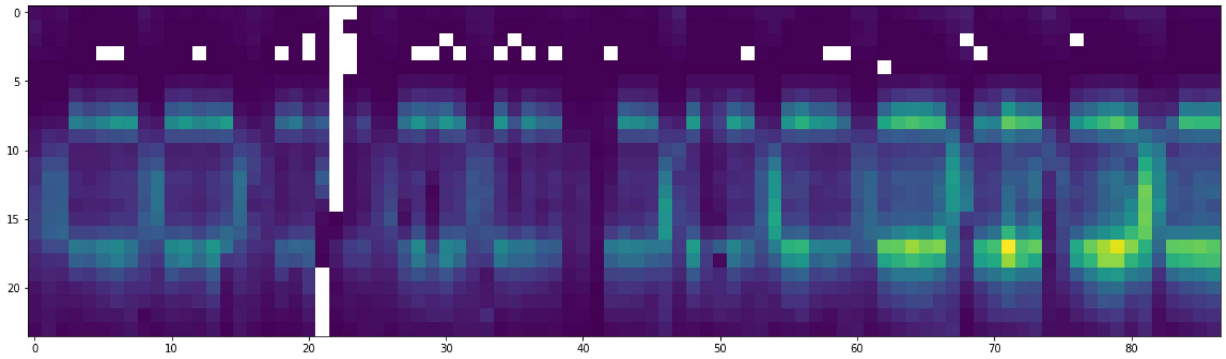
In [31]:
```
day_hour.start.unstack()
```

Out[31]:

start	0	1	2	3	4	5	6	7	8	9	...	14	15	16
start														
1	56.0	105.0	74.0	32.0	13.0	5.0	10.0	14.0	54.0	101.0	...	324.0	338.0	342.0
2	37.0	31.0	17.0	23.0	4.0	7.0	10.0	34.0	80.0	203.0	...	495.0	525.0	529.0
3	59.0	42.0	39.0	15.0	6.0	9.0	5.0	33.0	87.0	168.0	...	524.0	546.0	579.0
4	20.0	6.0	2.0	1.0	3.0	58.0	192.0	468.0	759.0	321.0	...	145.0	206.0	365.0
5	5.0	5.0	3.0	1.0	2.0	42.0	131.0	363.0	683.0	329.0	...	175.0	208.0	365.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
87	113.0	82.0	50.0	34.0	12.0	24.0	94.0	166.0	297.0	509.0	...	910.0	761.0	667.0
88	15.0	7.0	2.0	3.0	8.0	42.0	81.0	197.0	587.0	464.0	...	481.0	437.0	696.0
89	31.0	11.0	9.0	3.0	8.0	79.0	240.0	727.0	1211.0	564.0	...	433.0	473.0	700.0
90	31.0	18.0	4.0	6.0	7.0	79.0	215.0	703.0	1176.0	593.0	...	493.0	545.0	749.0

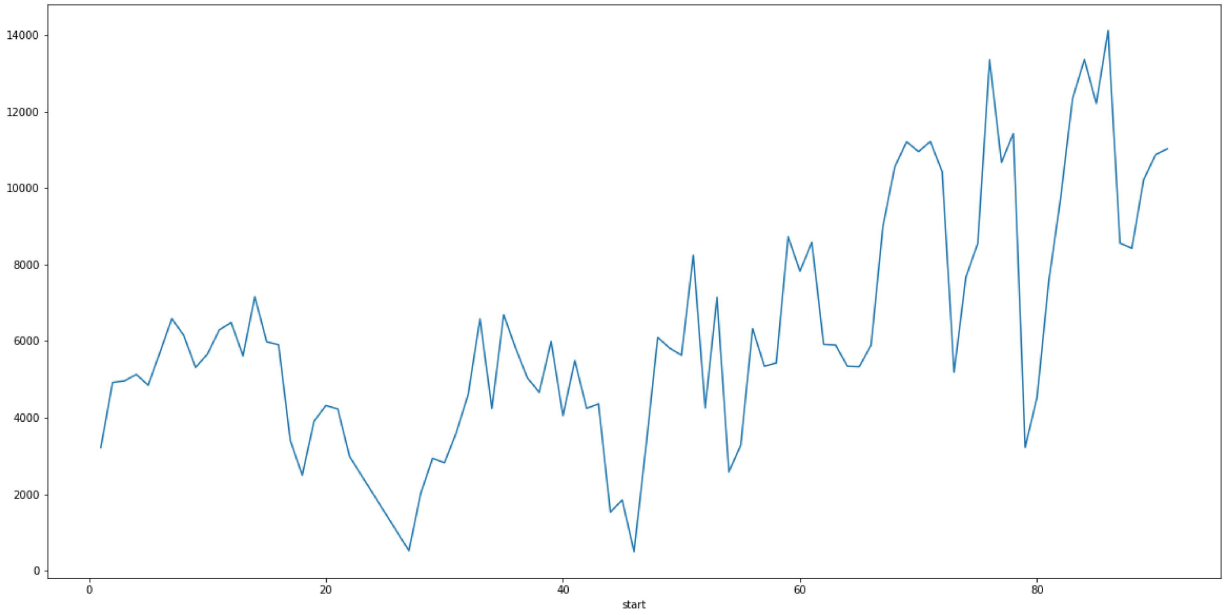
In [32]:
```
plt.figure(figsize=(20,10))
plt.imshow(day_hour.start.unstack().T)
```

Out[32]:
```
<matplotlib.image.AxesImage at 0x243d109ea88>
```



In [33]:
```
day_count.start.plot()
```

Out[33]:
```
<AxesSubplot:xlabel='start'>
```



In [34]:
```
bikes.start.dt.dayofyear
```

Out[34]:
```
0         91
1         91
2         91
3         91
4         91
          ..
552394     1
552395     1
552396     1
552397     1
552398     1
Name: start, Length: 552399, dtype: int64
```

In [35]:
```
bikes[bikes.start=="2016-01-10"].shape
```

Out[35]:
```
(1, 15)
```

Assignment 4

Explain the results in a **paragraph + charts** of to describe which model you'd recommend. This means show the data and the model's line on the same chart. The paragraph is a simple justification and comparison of the several models you tried.

1. Using the `day_hour_count` dataframe create two dataframes `monday` and `saturday` that represent the data for those days. (hint: Monday is day=0)

In [36]:
```
monday = day_hour_count[[0]].copy()
```

In [37]:
```
monday = day_hour_count.iloc[[0]].copy().T.fillna(0)
saturday = day_hour_count.iloc[[5]].copy().T.fillna(0)
```

In [38]:
```
monday
```

Out[38]:

hour_of_day	start
0.0	21.0
0.1	39.0
0.2	31.0
0.3	26.0
0.4	19.0
...	...
23.5	36.0
23.6	37.0
23.7	30.0
23.8	33.0
23.9	34.0

240 rows × 1 columns

In [39]:
```
saturday
```

Out[39]:

hour_of_day	start
0.0	89.0
0.1	87.0
0.2	98.0
0.3	99.0
0.4	98.0
...	...
23.5	93.0
23.6	95.0
23.7	105.0
23.8	93.0
23.9	111.0

240 rows × 1 columns

In [ ]:
```
```

In [40]:
```
monday["hour"] = monday.index
saturday["hour"] = saturday.index
```

In [41]:
```
monday
```

Out[41]:

hour_of_day	start	0	hour
0.0		21.0	0.0
0.1		39.0	0.1
0.2		31.0	0.2
0.3		26.0	0.3
0.4		19.0	0.4
...		...	...
23.5		36.0	23.5
23.6		37.0	23.6
23.7		30.0	23.7
23.8		33.0	23.8
23.9		34.0	23.9

240 rows × 2 columns

2a. Create 3 models fit to `monday.hour_of_day` with varying polynomial degrees (choose from `n=1,2,3,5,10,15`). (Repeat for `saturday` below)

Plot all the results for each polynomial.

In [42]:
```
from sklearn.linear_model import LinearRegression
```

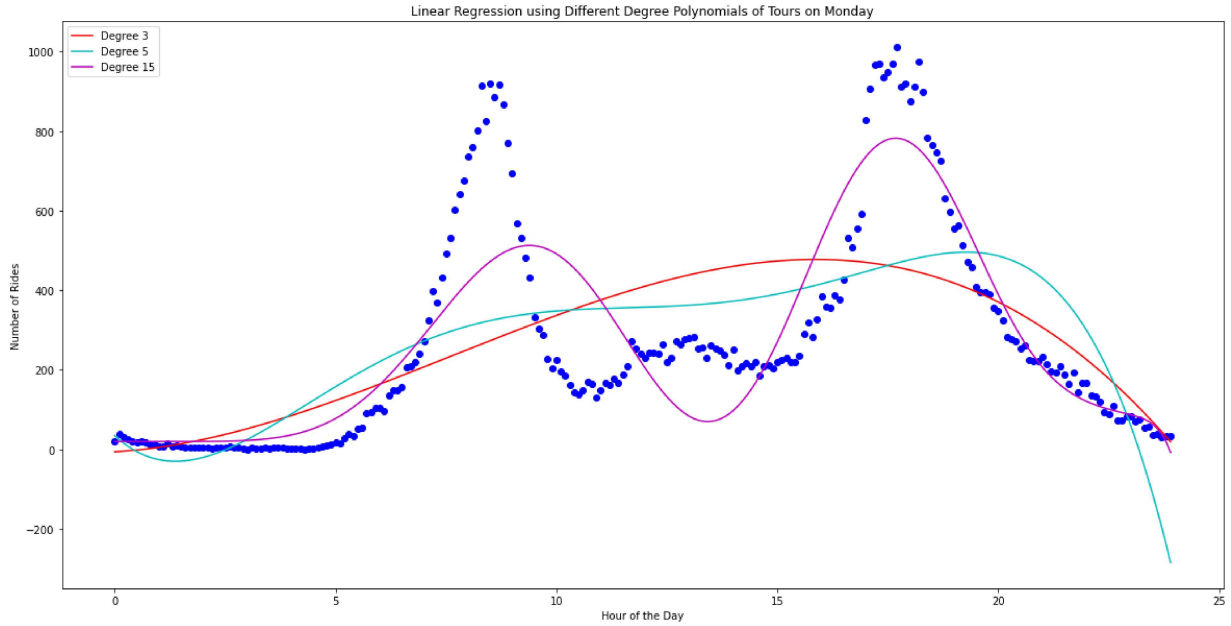
In [43]:
```
x = monday.hour.to_numpy().reshape(-1,1)
y = monday[0].to_numpy().reshape(-1,1)
```

In [44]:
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3)

x_m_3 = poly.fit_transform(x)
model_3 = LinearRegression()
model_3.fit(x_m_3, y)

poly = PolynomialFeatures(degree = 5)

x_m_5 = poly.fit_transform(x)
model_5 = LinearRegression()
model_5.fit(x_m_5, y)

poly = PolynomialFeatures(degree = 15)

x_m_15 = poly.fit_transform(x)
model_15 = LinearRegression()
model_15.fit(x_m_15, y)
```

Out[44]:
```
LinearRegression()
```

In [45]:
```
plt.scatter(x,y, c = "b")
plt.plot(x, np.dot(x_m_3, model_3.coef_.T) + model_3.intercept_, c = "r", label =
plt.plot(x, np.dot(x_m_5, model_5.coef_.T) + model_5.intercept_, c = "c", label =
plt.plot(x, np.dot(x_m_15, model_15.coef_.T) + model_15.intercept_, c = "m", labe
plt.legend(loc = "upper left")
plt.title("Linear Regression using Different Degree Polynomials of Tours on Monda
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Rides")
```

Out[45]:
```
Text(0, 0.5, 'Number of Rides')
```



Explanation

I would recommend modeling this data with a polynomial of degree 15. Graphically, it appears that this model models the data the best out of the choices given without overfitting. While, the magenta model (deg 15) doesn't accurately capture the peaks and valleys shown, by the data, it is the best choice out of all the models chosen for this analysis. The red model does not show a decreasing slope until about 15:00 or 3PM, while the data clearly shows a bi modal distribution with the number of tours decreasing after around 9:00 or 9AM and 18:00 or 6PM. The cyan model does show a relatively tame bimodal distribution, but the predictions are wildly inaccurate according to their residuals (which is observed by the distance between the line and dots at any point throughout the day). Due to these reasons, I would recommend using a polynomial of degree 15 to model the data preesent.

2b. Repeat 2a for `saturday.hour_of_day`

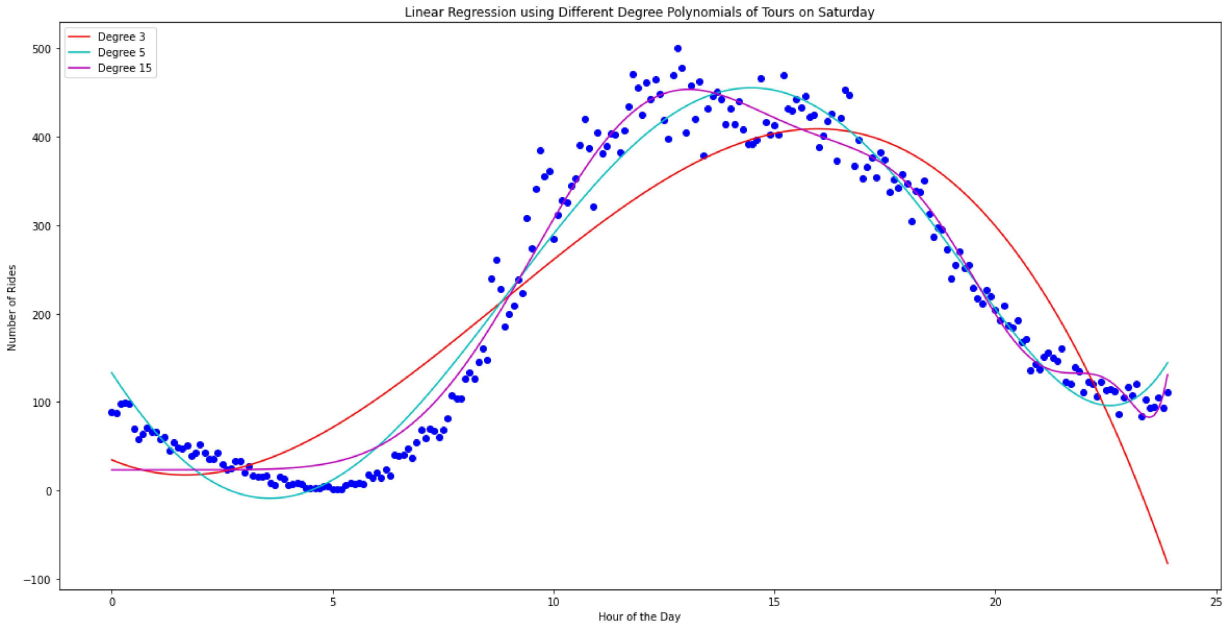
In [46]:
```
x = saturday.hour.to_numpy().reshape(-1,1)
y = saturday[5].to_numpy().reshape(-1,1)
```

In [47]:
```
poly = PolynomialFeatures(degree = 3)

x_s_3 = poly.fit_transform(x)
model_3 = LinearRegression()
model_3.fit(x_s_3, y)

poly = PolynomialFeatures(degree = 5)

x_s_5 = poly.fit_transform(x)
model_5 = LinearRegression()
model_5.fit(x_s_5, y)

poly = PolynomialFeatures(degree = 15)

x_s_15 = poly.fit_transform(x)
model_15 = LinearRegression()
model_15.fit(x_s_15, y)
```

Out[47]:
```
LinearRegression()
```

In [48]:
```
plt.scatter(x,y, c = "b")
plt.plot(x, np.dot(x_s_3, model_3.coef_.T) + model_3.intercept_, c = "r", label =
plt.plot(x, np.dot(x_s_5, model_5.coef_.T) + model_5.intercept_, c = "c", label =
plt.plot(x, np.dot(x_s_15, model_15.coef_.T) + model_15.intercept_, c = "m", labe
plt.legend(loc = "upper left")
plt.title("Linear Regression using Different Degree Polynomials of Tours on Satur
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Rides")
```

Out[48]:
```
Text(0, 0.5, 'Number of Rides')
```



Explanation

I would reccomend modeling this data with a degree 5 polynomial. Graphically, it appears that the degree 5 polynomial models the data the most accurately and does not overfit. The red model (degree 3) shows the overall trends of the distribution, but visually it appears that the predicted values are different from their actual values. The magenta model appears to be overfit and shows trends that visually do not appear in the data. This is most evident looking at 22:00 or 10PM, where

the magenta model shows the number if tours increasing, while the data shows a gradual decrease of tours between 16:00 or 4PM until 23:00 or 11PM. Due to these reasons, I would recommend choosing the degree 5 polynomial to model this data, because it most accurately reflects the data and does not overfit.

3. create 3 new models fit to `hour_of_day` with different Ridge Regression α (alpha) Ridge Coefficient values using the `monday` and `saturday` datasets.

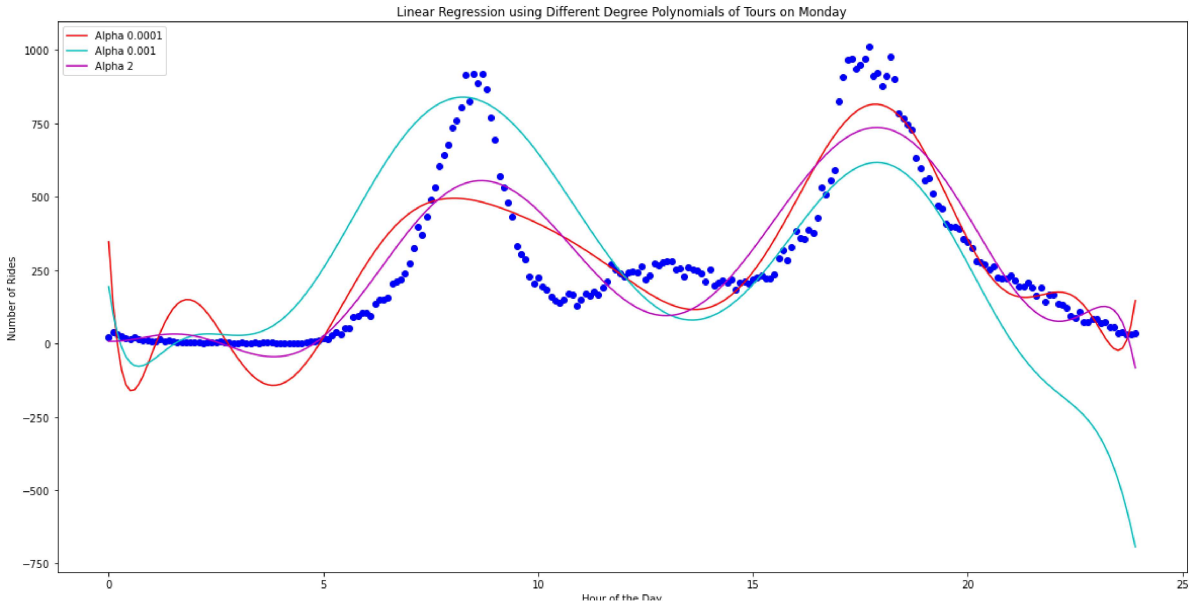
In [49]:
```
from sklearn.linear_model import Ridge
#Monday
x = monday.hour.to_numpy().reshape(-1,1)
y = monday[0].to_numpy().reshape(-1,1)

poly = PolynomialFeatures(degree = 12)


x_m_half = poly.fit_transform(x)
model_half = Ridge(alpha = 0.0001)
model_half.fit(x_m_half, y)


x_m_2 = poly.fit_transform(x)
model_2 = Ridge(alpha = 0.001)
model_2.fit(x_m_2, y)


x_m_10 = poly.fit_transform(x)
model_10 = Ridge(alpha = 2)
model_10.fit(x_m_10, y)
```

```
C:\Users\gdlev\anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:157:
LinAlgWarning: Ill-conditioned matrix (rcond=9.24055e-39): result may not be ac
curate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
C:\Users\gdlev\anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:157:
LinAlgWarning: Ill-conditioned matrix (rcond=1.37826e-34): result may not be ac
curate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

Out[49]:
```
Ridge(alpha=2)
```

In [50]:
```
plt.scatter(x,y, c = "b")
plt.plot(x, np.dot(x_m_half, model_half.coef_.T) + model_half.intercept_, c = "r"
plt.plot(x, np.dot(x_m_2, model_2.coef_.T) + model_2.intercept_, c = "c", label =
plt.plot(x, np.dot(x_m_10, model_10.coef_.T) + model_10.intercept_, c = "m", labe
plt.legend(loc = "upper left")
plt.title("Linear Regression using Different Degree Polynomials of Tours on Monda
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Rides")
```

Out[50]:
```
Text(0, 0.5, 'Number of Rides')
```



Explanation

I would reccomend choosing the degree 12 polynomial fitted to ridge regression with an alpha value of 2. Graphically, it appears that this model most accurately represents the data without overfitting. The cyan model (alpha = 0.01) appears to model the peaks and values quite well, but quickly becomes erratic, especially at the beginning and the end of the day where the predicted number of tours is drastically different than the actual tours at those times. The red model (alpha = 0.0001) appears to model the data quite well, however shows trends that do not exist visually. For example, the red model predicts the number of tours to increase between 1:00 or 1AM and 2:00 or 2AM and then decrease between 2:00 or 2AM until 5:00 or 5AM, visually the number of tours appears to be relatively constant between 12:00 or 12AM and 5:00 or 5AM. Due to these reasons, I would reccomend modeling the present data using a polynomila of degree 12 and an alpha value of 2.

In [51]:
```
#Saturday
x = saturday.hour.to_numpy().reshape(-1,1)
y = saturday[5].to_numpy().reshape(-1,1)
```

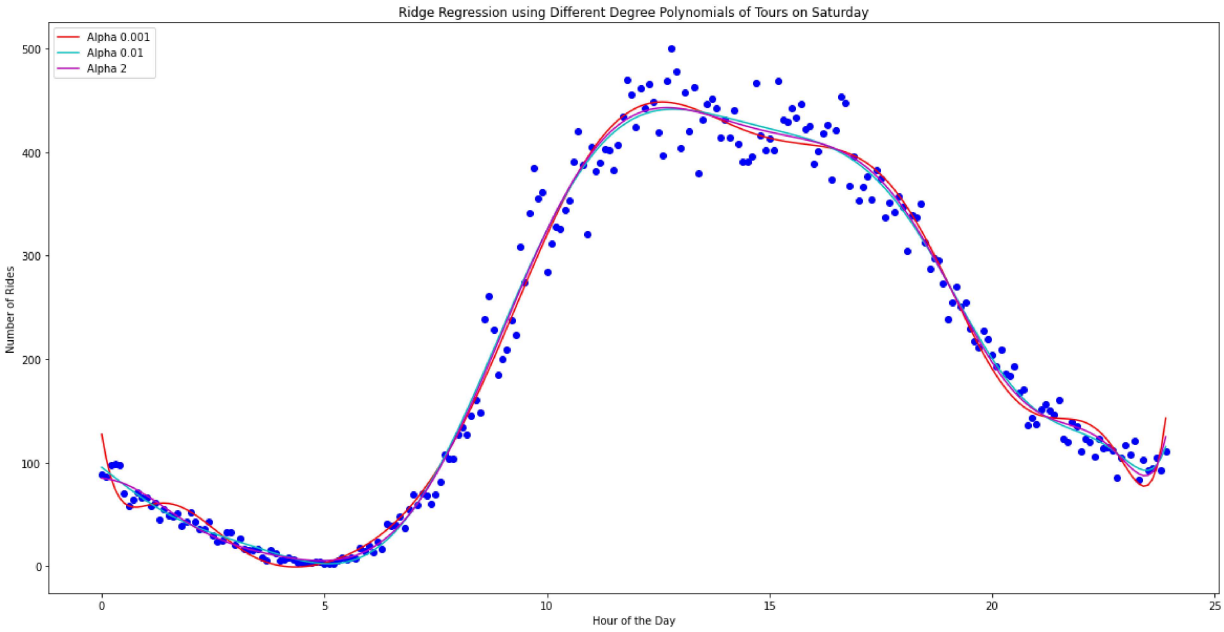
In [52]:
```
poly = PolynomialFeatures(degree = 12)

x_s_half = poly.fit_transform(x)
model_half = Ridge(alpha = 0.0001)
model_half.fit(x_s_half, y)

x_s_2 = poly.fit_transform(x)
model_2 = Ridge(alpha = 0.01)
model_2.fit(x_s_2, y)

x_s_10 = poly.fit_transform(x)
model_10 = Ridge(alpha = 2)
model_10.fit(x_s_10, y)
```

```
C:\Users\gdlev\anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:157:
LinAlgWarning: Ill-conditioned matrix (rcond=9.24055e-39): result may not be ac
curate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
C:\Users\gdlev\anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:157:
LinAlgWarning: Ill-conditioned matrix (rcond=9.24055e-37): result may not be ac
curate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
C:\Users\gdlev\anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:157:
LinAlgWarning: Ill-conditioned matrix (rcond=1.37826e-34): result may not be ac
curate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

Out[52]:
```
Ridge(alpha=2)
```

In [53]:
```
plt.scatter(x,y, c = "b")
plt.plot(x, np.dot(x_s_half, model_half.coef_.T) + model_half.intercept_, c = "r"
plt.plot(x, np.dot(x_s_2, model_2.coef_.T) + model_2.intercept_, c = "c", label =
plt.plot(x, np.dot(x_s_10, model_10.coef_.T) + model_10.intercept_, c = "m", labe
plt.legend(loc = "upper left")
plt.title("Ridge Regression using Different Degree Polynomials of Tours on Saturd
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Rides")
```

Out[53]:
```
Text(0, 0.5, 'Number of Rides')
```



Explanation

I would recommend using a polynomial of degree 12 fitted with a ridge regression that has an alpha value of 0.01. Graphically, it appears that this model seems to model the data the most accurately without overfitting. The ridge regression model with alpha value of 2 appears to overfit the data and by looking at early times in the day, the predictions appear to be poor; if the graph continued on the left side, the magenta (alpha = 2) model appears to make predictions that the number of tours would decrease, but visually it appears that the number of tours would increase. The red model (alpha = 0.0001) would not be chosen, because it could potentially be overfitting the data. The predictions for the red model are also poor; if the graph was extended on the right side, the model predicts that number of tours would drastically increase leading into the next day, however visually it appears that the number of tours would moderately increase. Obviously this data is aggregated over the whole year, however when I'm talking about extending the data, I'm referencing if for example the data was only showing January 5th, then extending the graph to the right would show January 4th, and extending the data to the left would show January 6th. The overall trend would appear periodic. Due to all these reasons, I believe that out of the model choices chosen, a polynomial of degree 12 fitted to ridge regression model with alpha value of 0.01 would be the most robust in terms of overfitting and robustness.

In [ ]:
```
```