Group 8 Final: Chicago Taxi Data

Andrew Shimshock, Claire Zyfers, Mina Tawfik, Robby Konrath, Travis Vickers

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import datetime
import math
import json
%matplotlib inline
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:

```
df=pd.read_csv("chicago_taxi_trips_2016_01.csv")
df2=pd.read_csv("chicago_taxi_trips_2016_02.csv")
df3=pd.read_csv("chicago_taxi_trips_2016_03.csv")
df4=pd.read_csv("chicago_taxi_trips_2016_04.csv")
df5=pd.read_csv("chicago_taxi_trips_2016_05.csv")
df6=pd.read_csv("chicago_taxi_trips_2016_06.csv")
df7=pd.read_csv("chicago_taxi_trips_2016_07.csv")
df8=pd.read_csv("chicago_taxi_trips_2016_08.csv")
df9=pd.read_csv("chicago_taxi_trips_2016_09.csv")
df10=pd.read_csv("chicago_taxi_trips_2016_10.csv")
df11=pd.read_csv("chicago_taxi_trips_2016_11.csv")
df12=pd.read_csv("chicago_taxi_trips_2016_12.csv")
#data set came with twelve months so opened all
#takes a couple minutes
```

In [3]:

```
df=df.append(df2,ignore_index=True)
df=df.append(df3,ignore_index=True)
df=df.append(df4,ignore_index=True)
df=df.append(df5,ignore_index=True)
df=df.append(df6,ignore_index=True)
df=df.append(df7,ignore_index=True)
df=df.append(df8,ignore_index=True)
df=df.append(df9,ignore_index=True)
df=df.append(df10,ignore_index=True)
df=df.append(df11,ignore_index=True)
df=df.append(df12,ignore_index=True)
#added them all together
#also takes a couple minutes
```

In [4]:

```
newdf=df.sample(frac=.1,random_state=0)
#took a 10% sample size because 20 million rows is too slow to work with in Juypter, but 2 million
is fast enough
```

Data Description

1. **taxi_id**: identification number of taxi
2. **trip_start_timestamp**: time taxi driver begins trip for passenger
3. **trip_end_timestamp**: time taxi driver ends trip for passenger
4. **trip_seconds**: total time of trip counted in seconds for passenger
5. **trip_miles**: total miles traveled during trip for passenger

6. **pickup_community_area**: specialized community area where passenger picked up
7. **dropoff_community_area**: specialized community area where passenger dropped off up
8. **fare**: amount to go from point a to b
9. **tips**: amount passenger tipped taxi driver
10. **tolls**: extra toll payments
11. **extras**: extra money spent by passenger for taxi ride
12. **trip_total**: total amount passenger spent on taxi ride
13. **payment_type**: payment method of customer
14. **company**: company taxi driver works for
15. **pickup_lattitude**: pickup destination lattitude point
16. **pickup_longitude**: pickup destination longitude point
17. **dropoff_lattitude**: dropoff destination lattitude point
18. **dropoff_longitude**: dropoff destination longitude point

In [5]:

```
newdf.shape
```

Out[5]:

```
(1986616, 20)
```

In [6]:

```
newdf.isna().sum()
```

Out[6]:

```
taxi_id                     283
trip_start_timestamp          0
trip_end_timestamp          250
trip_seconds                329
trip_miles                   22
pickup_census_tract       1986616
dropoff_census_tract       772817
pickup_community_area      275671
dropoff_community_area     308533
fare                           30
tips                           30
tolls                          30
extras                         30
trip_total                     30
payment_type                    0
company                    763833
pickup_latitude            275629
pickup_longitude           275629
dropoff_latitude           304747
dropoff_longitude          304747
dtype: int64
```

In [215]:

```
newdf.head()
```

Out[215]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016.0	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8.0	28.0
11107382	4387.0	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24.0	24.0
12285774	4865.0	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28.0	56.0
4176652	727.0	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6.0	3.0
18263903	2452.0	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	NaN	NaN

Data Cleanup

In [7]:

```
newdf=newdf.drop(columns={'pickup_census_tract','dropoff_census_tract'})
#drop these two columns because one is all nan
```

In [8]:

```
faremean=newdf.fare.mean()
newdf.fare.fillna(value=faremean,inplace=True)
tipsmean=newdf.tips.mean()
tollsmean=newdf.tolls.mean()
milesmean=newdf.trip_miles.mean()
secondsmean=newdf.trip_seconds.mean()
extrasmean=newdf.extras.mean()
newdf.tips.fillna(value=tipsmean,inplace=True)
newdf.tolls.fillna(value=tollsmean,inplace=True)
newdf.trip_miles.fillna(value=milesmean,inplace=True)
newdf.trip_seconds.fillna(value=secondsmean,inplace=True)
newdf.extras.fillna(value=extrasmean,inplace=True)
totalmean=newdf.trip_total.mean()
newdf.trip_total.fillna(value=totalmean,inplace=True)
#fill in the few missing numerical values with averages
```

In [9]:

```
newdf.isna().sum()
```

Out[9]:

```
taxi_id                      283
trip_start_timestamp           0
trip_end_timestamp           250
trip_seconds                   0
trip_miles                     0
pickup_community_area     275671
dropoff_community_area    308533
fare                           0
tips                           0
tolls                          0
extras                         0
trip_total                     0
payment_type                   0
company                   763833
pickup_latitude           275629
pickup_longitude          275629
dropoff_latitude          304747
dropoff_longitude         304747
dtype: int64
```

In [10]:

```
list=[]
for x in newdf[newdf.trip_end_timestamp.isna()].trip_start_timestamp.values:
    y=datetime.datetime.strptime(x,"%Y-%m-%d %H:%M:%S")+datetime.timedelta(0,newdf.trip_seconds.mea
n())
    y=y.strftime("%Y-%m-%d %H:%M:%S")
    list.append(y)
dfchange=newdf[newdf.trip_end_timestamp.isna()].copy()
dfchange.trip_end_timestamp=list
#create a new df that fills in the few missing end times by adding the average seconds to the star
t date.
#the timestamps in the whole df are not exactly accurate to the trip seconds per row. They are jus
t rounded to the nearest 15 min mark
```

In [11]:

```
dfchange.head()
```

Out[11]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
6094758	1445.0	2016-4-21 17:45:00	2016-04-21 17:57:47	767.551123	0.0	28.0	NaN
4217508	7517.0	2016-3-14 15:30:00	2016-03-14 15:42:47	767.551123	0.0	6.0	NaN
11236242	7099.0	2016-6-29 20:30:00	2016-06-29 20:42:47	767.551123	0.0	59.0	NaN
6661011	6966.0	2016-4-20 01:15:00	2016-04-20 01:27:47	767.551123	0.0	NaN	NaN
13288045	1603.0	2016-8-21 21:00:00	2016-08-21 21:12:47	767.551123	0.0	56.0	NaN

In [12]:

```
dfcopy=newdf.copy()
dfcopy.dropna(subset=['trip_end_timestamp'],axis=0,inplace=True)
dfcopy=dfcopy.append(dfchange)
#added on these changed nans to the bottom of the df and dropped the nans as they were
```

In [13]:

```
dfcopy.isna().sum()
```

Out[13]:

```
taxi_id                      283
trip_start_timestamp           0
trip_end_timestamp             0
trip_seconds                   0
trip_miles                     0
pickup_community_area     275671
dropoff_community_area    308533
fare                           0
tips                           0
tolls                          0
extras                         0
trip_total                     0
payment_type                   0
company                   763833
pickup_latitude           275629
pickup_longitude          275629
dropoff_latitude          304747
dropoff_longitude         304747
dtype: int64
```

In [14]:

```
dfcopy.company.fillna(value='Unknown', inplace=True)
dfcopy.pickup_community_area.fillna(value='Unknown',inplace=True)
dfcopy.dropoff_community_area.fillna(value='Unknown',inplace=True)
dfcopy.taxi_id.fillna(value='Unknown',inplace=True)
#fill in columns with categorical values with Unknown
#note taxi_id is not a number but a string, the original value before the owener transformed some
data is a very long number that makes no sense so we just kept these numbers as the id name
```

In [15]:

```
df=dfcopy.copy()
#change back to df after all these changes
```

This next part involves changing back some of the values to what they were. The maker of this dataset created a key in json format for company names and latitude and longitude values. This next part involves making them their actual correct values. They also had one for taxi_ids but the format was terrible so we stuck with string numbers to represent them

In [16]:

```
df.head()
#as you can see company is numerical and longitude and latitude don't make any sense because they
are all located around chicago
```

Out[16]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4046	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

In [17]:

```
with open("column_remapping.json") as f_open:
    column_id_map = json.load(f_open)
company_id = column_id_map['company']
list=df.company.tolist()
i=-1
for x in list:
    i=i+1
    if x=='Unknown':
        list[i]='10000'
for i in range(len(list)):
    list[i]=int(list[i])
for i in range(len(list)):
    list[i]=str(list[i])
#opens the json data codes for company
#puts the current dataframe's company column into a list
#changes the unknown values already put in the column to an id specifically for that
#changes each number to an int because right now they are floats
#changes the int back to a string to correspond with the json file as they are all string numbers
with no decimals
```

In [312]:

```
list[0]
#output to match with the json file
```

Out[312]:

```
'Unknown'
```

In [313]:

```
company_id['0']
#what the json file says for company
```

Out[313]:

```
'3623-Arrington Enterprises'
```

In [20]:

```
i=-1
for x in list:
    i=i+1
    for y in company_id:
        if x==y:
            list[i]=company_id[y]
for i in range(len(list)):
          if list[i]=='10000':
               list[i]='Unknown'

#loops through both the json and the company list and modifies them to the real value
#last loop changes back the 10000 to unknown, which is what it was
```

In [309]:

```
list[0]
#final output
```

Out[309]:

'Unknown'

In [22]:

```
df.company=list
df.head()
#shows result in df
```

Out[22]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

In [23]:

```
df.pickup_latitude.fillna(value='Unknown',inplace=True)
df.pickup_longitude.fillna(value='Unknown',inplace=True)
df.dropoff_latitude.fillna(value='Unknown',inplace=True)
df.dropoff_longitude.fillna(value='Unknown',inplace=True)
#make these unknown for now just to get their correct values from json file
```

In [24]:

```
df.head()
```

Out[24]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

In [25]:

```
dfbefore=df.copy()
```

same process is repeated for all four of pickup longitude and latitude and dropoff longitude and latitude

In [26]:

```
with open("column_remapping.json") as f_open:
    column_id_map = json.load(f_open)
pickup_latitude_id = column_id_map['pickup_latitude']
list2=df.pickup_latitude.tolist()
i=-1
for x in list2:
    i=i+1
    if x=='Unknown':
```

```
    if x== 'Unknown':
        list2[i]='10000'
for i in range(len(list2)):
    list2[i]=int(list2[i])
for i in range(len(list2)):
    list2[i]=str(list2[i])
i=-1
for x in list2:
    i=i+1
    for y in pickup_latitude_id:
        if x==y:
            list2[i]=pickup_latitude_id[y]
for i in range(len(list2)):
    list2[i]=float(list2[i])

#same exact process as before so won't explain details again
#except changing final answer from string to float so can graph these eventually
#all in one shot this time
```

In [308]:

```
list2[0]
```

Out[308]:

41.892507781

In [28]:

```
df.pickup_latitude=list2
```

In [30]:

```
pickuplatmean=df[df.pickup_latitude!=10000.0].pickup_latitude.mean()
```

In [33]:

```
for i in range(len(list2)):
          if list2[i]==10000.0:
               list2[i]=pickuplatmean
```

In [307]:

```
list2[0]
```

Out[307]:

41.892507781

In [35]:

```
df.pickup_latitude=list2
```

In [36]:

```
with open("column_remapping.json") as f_open:
    column_id_map = json.load(f_open)
pickup_longitude_id = column_id_map['pickup_longitude']
list3=df.pickup_longitude.tolist()
i=-1
for x in list3:
    i=i+1
    if x=='Unknown':
        list3[i]='10000'
for i in range(len(list3)):
    list3[i]=int(list3[i])
for i in range(len(list3)):
    list3[i]=str(list3[i])
i=-1
for x in list3:
```

```
    i=i+1
    for y in pickup_longitude_id:
        if x==y:
            list3[i]=pickup_longitude_id[y]
for i in range(len(list3)):
    list3[i]=float(list3[i])
```

In [306]:

```
list3[0]
```

Out[306]:

-87.626214906

In [38]:

```
df.pickup_longitude=list3
```

In [39]:

```
df[df.pickup_longitude!=10000.0].pickup_longitude.mean()
```

Out[39]:

-87.65903175066352

In [40]:

```
pickuplongmean=df[df.pickup_longitude!=10000.0].pickup_longitude.mean()
```

In [41]:

```
for i in range(len(list3)):
          if list3[i]==10000.0:
               list3[i]=pickuplongmean
```

In [305]:

```
list3[0]
```

Out[305]:

-87.626214906

In [43]:

```
df.pickup_longitude=list3
```

In [44]:

```
with open("column_remapping.json") as f_open:
    column_id_map = json.load(f_open)
dropoff_longitude_id = column_id_map['dropoff_longitude']
list4=df.dropoff_longitude.tolist()
i=-1
for x in list4:
    i=i+1
    if x=='Unknown':
        list4[i]='10000'
for i in range(len(list4)):
    list4[i]=int(list4[i])
for i in range(len(list4)):
    list4[i]=str(list4[i])
i=-1
for x in list4:
    i=i+1
    for y in dropoff_longitude_id:
```

```
        if x==y:
            list4[i]=dropoff_longitude_id[y]
for i in range(len(list4)):
    list4[i]=float(list4[i])
```

In [304]:

```
list4[0]
```

Out[304]:

-87.642648998

In [46]:

```
df.dropoff_longitude=list4
```

In [47]:

```
df[df.dropoff_longitude!=10000.0].dropoff_longitude.mean()
```

Out[47]:

-87.65373579490085

In [48]:

```
dropofflongmean=df[df.dropoff_longitude!=10000.0].dropoff_longitude.mean()
```

In [49]:

```
for i in range(len(list4)):
          if list4[i]==10000.0:
               list4[i]=dropofflongmean
```

In [303]:

```
list4[0]
```

Out[303]:

-87.642648998

In [51]:

```
df.dropoff_longitude=list4
```

In [52]:

```
with open("column_remapping.json") as f_open:
    column_id_map = json.load(f_open)
dropoff_latitude_id = column_id_map['dropoff_latitude']
list5=df.dropoff_latitude.tolist()
i=-1
for x in list5:
    i=i+1
    if x=='Unknown':
        list5[i]='10000'
for i in range(len(list5)):
    list5[i]=int(list5[i])
for i in range(len(list5)):
    list5[i]=str(list5[i])
i=-1
for x in list5:
    i=i+1
    for y in dropoff_latitude_id:
        if x==y:
            list5[i]=dropoff_latitude_id[y]
for i in range(len(list5)):
```

```
for i in range(len(list5)):
    list5[i]=float(list5[i])
```

In [302]:

```
list5[0]
```

Out[302]:

41.879255084

In [54]:

```
df.dropoff_latitude=list5
```

In [55]:

```
df[df.dropoff_latitude!=10000.0].dropoff_latitude.mean()
```

Out[55]:

41.900885537721386

In [56]:

```
dropofflatmean=df[df.dropoff_latitude!=10000.0].dropoff_latitude.mean()
```

In [57]:

```
for i in range(len(list5)):
          if list5[i]==10000.0:
               list5[i]=dropofflatmean
```

In [301]:

```
list5[0]
```

Out[301]:

41.879255084

In [59]:

```
df.dropoff_latitude=list5
```

In [60]:

```
df.head()
```

Out[60]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

In [189]:

```
dfbefore.head()
```

Out[189]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

In [62]:

```
df.isna().sum()
```

Out[62]:

```
taxi_id                   0
trip_start_timestamp      0
trip_end_timestamp        0
trip_seconds              0
trip_miles                0
pickup_community_area     0
dropoff_community_area    0
fare                      0
tips                      0
tolls                     0
extras                    0
trip_total                0
payment_type              0
company                   0
pickup_latitude           0
pickup_longitude          0
dropoff_latitude          0
dropoff_longitude         0
dtype: int64
```

NaN cleanup finished. Now lets add some new columns to make it easier to find interesting things

In [66]:

```
df[['date_start','time_start']]=df.trip_start_timestamp.str.split(" ",expand=True)
```

In [74]:

```
df[['date_end','time_end']]=df.trip_end_timestamp.str.split(" ",expand=True)
```

In [75]:

```
df.head()
#split up the timestamps into separate dates and times
```

Out[75]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

5 rows × 22 columns

In [80]:

```
datelist=[]
for x in range(len(df.date_start)):
    datelist.append(datetime.datetime.strptime(df.date_start.values[x],'%Y-%m-%d'))
for x in range(len(datelist)):
    datelist[x]=datetime.datetime.isoweekday(datelist[x])
for x in range(len(datelist)):
    if datelist[x]>5:
        datelist[x]='Weekend'
    else:
        datelist[x]='Weekday'
#make the start date column into a new list with datetime.dates
#use datetime isoweekday to specificy if one thing is a weekday
```

In [300]:

```
datelist[0]
```

Out[300]:

```
'Weekend'
```

In [82]:

```
df['Day_of_week']=datelist
```

In [83]:

```
df.groupby('Day_of_week')['Day_of_week'].count()
```

Out[83]:

```
Day_of_week
Weekday    1467801
Weekend     518815
Name: Day_of_week, dtype: int64
```

In [84]:

```
df.head()
```

Out[84]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

5 rows × 23 columns

Finding Number 1 - Best location to make the most money

In [85]:

```
import geopandas as gpd
import descartes
from shapely.geometry import Point, Polygon
#import geopandas to make a map with chicago to see density of where cabs are
```

In [86]:

```
map1=gpd.read_file('geo_export_14fc79b6-5fbf-426a-9691-2408ba6973c8.shp')
#shapefile downloaded from chicago url
```

In [87]:

```
geo=[Point(xy) for xy in zip(df.pickup_longitude,df.pickup_latitude)]
#using point to make a list of all the points for pickup
```

In [88]:

```
geo2=[Point(xy) for xy in zip(df.dropoff_longitude,df.dropoff_latitude)]
#using point to make a list of all the points for dropoff
```

In [89]:

```
geodf=gpd.GeoDataFrame(df,crs="EPSG:4326",geometry=geo)
#setting the right longitude/latitude identifier
```

In [90]:

```
geodf2=gpd.GeoDataFrame(df,crs="EPSG:4326",geometry=geo2)
#setting the right longitude/latitude identifier
```

In [91]:

```
fig,ax=plt.subplots(figsize=(17,17))
map1.plot(ax=ax,color='grey')
geodf.plot(ax=ax,markersize=20,color='red',marker="o",label='pickup')
geodf2.plot(ax=ax,markersize=20,color='blue',marker="^",label='dropoff')
plt.legend(prop={'size':10})
#chicago is located in northeast here
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10e46c224e0>
```

Out[91]:

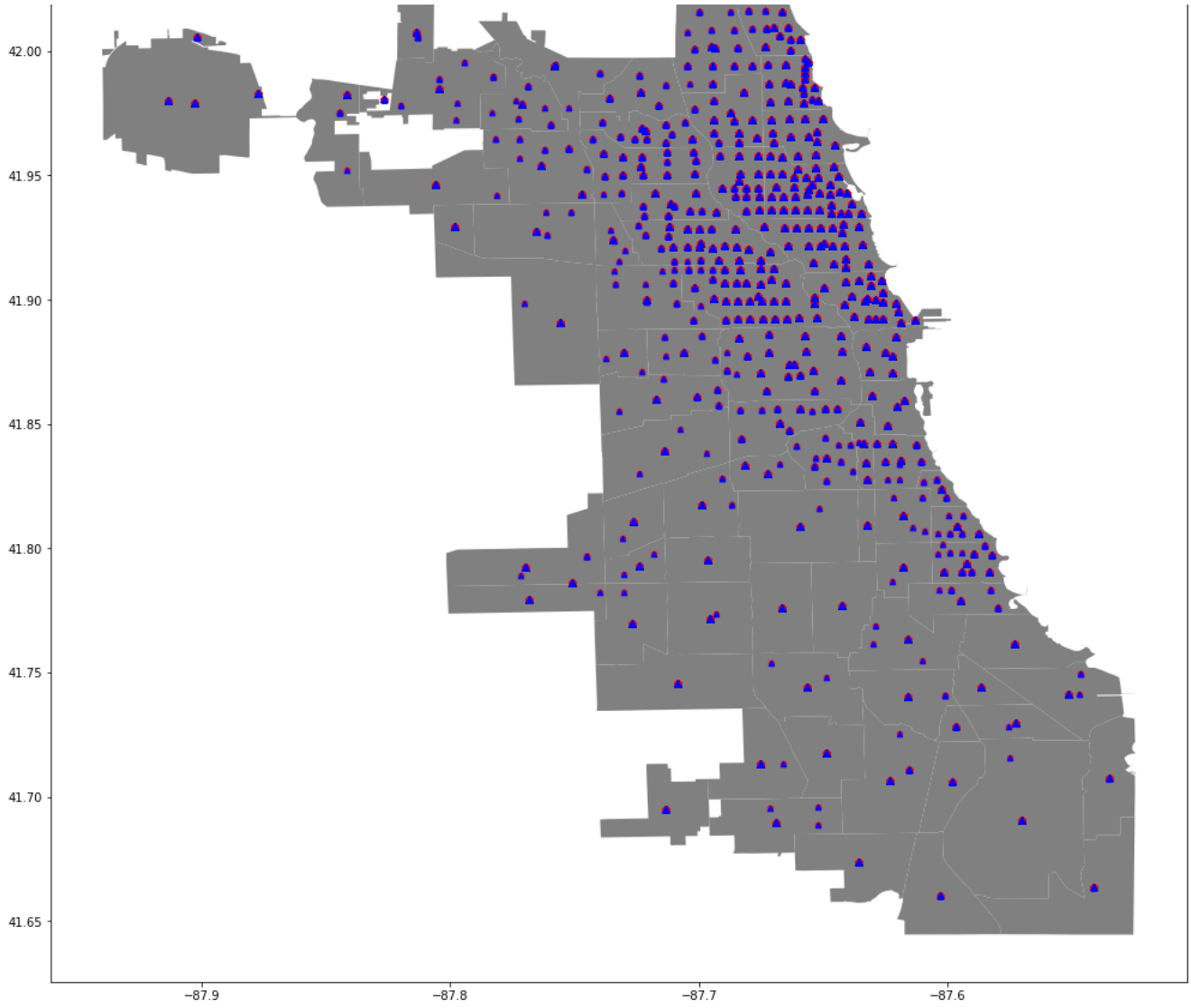
```
<matplotlib.axes._subplots.AxesSubplot at 0x10e46c224e0>
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10e46c224e0>
```

Out[91]:

```
<matplotlib.legend.Legend at 0x10f587d5eb8>
```

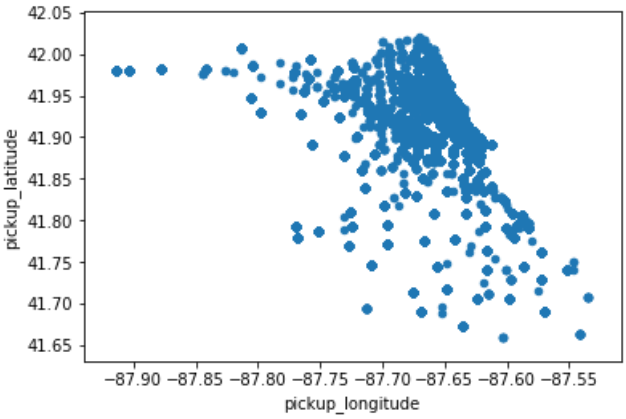


In [92]:

```
df.plot(kind='scatter',x='pickup_longitude',y='pickup_latitude')
#just to better show the density
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10f58b98a58>
```



From the previous charts, overall majority of rides and business occur in Chicago, which makes sense. Pickup and dropoff locations are pretty close to one another

In [93]:

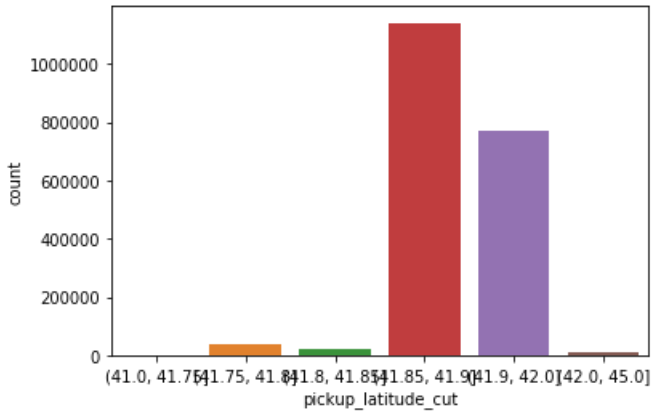
```
df['pickup_latitude_cut']=pd.cut(df.pickup_latitude, bins=(41,41.75,41.8,41.85,41.9,42,45))
```

In [216]:

```
sns.countplot(x='pickup_latitude_cut',data=df)
```

Out[216]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10e08a8aeb8>
```



Trips that start south of Chicago are about twice as long in length and seconds

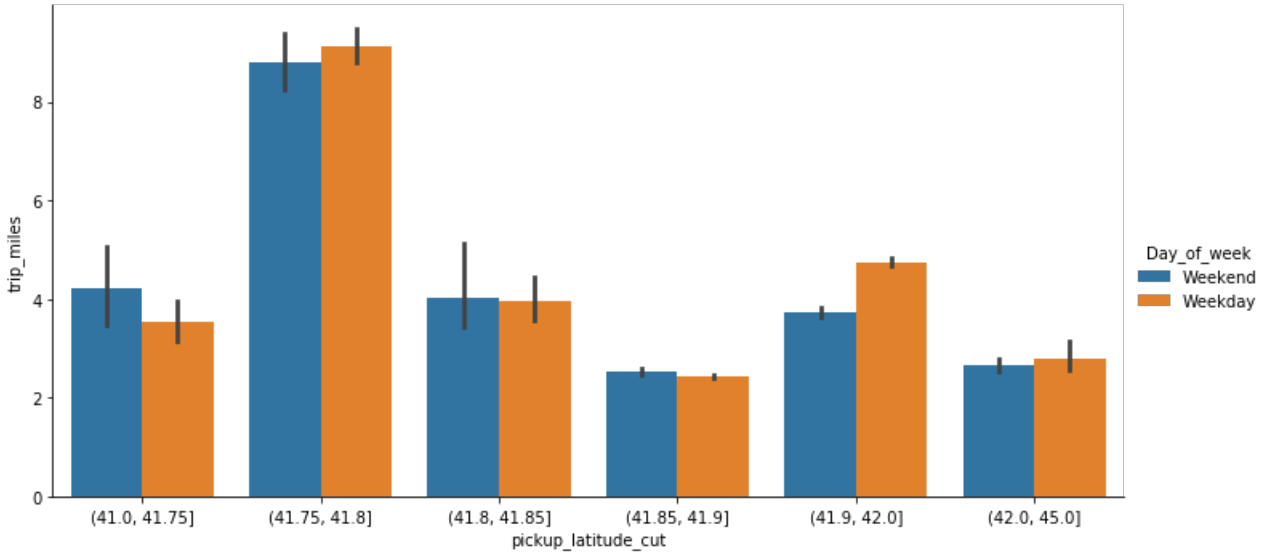
In [94]:

```
sns.catplot(x='pickup_latitude_cut',y='trip_miles',hue='Day_of_week',kind='bar',data=df,aspect=2)
```

```
C:\Users\Robby Konrath\Documents\Python\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[94]:

```
<seaborn.axisgrid.FacetGrid at 0x10f5a47c390>
```

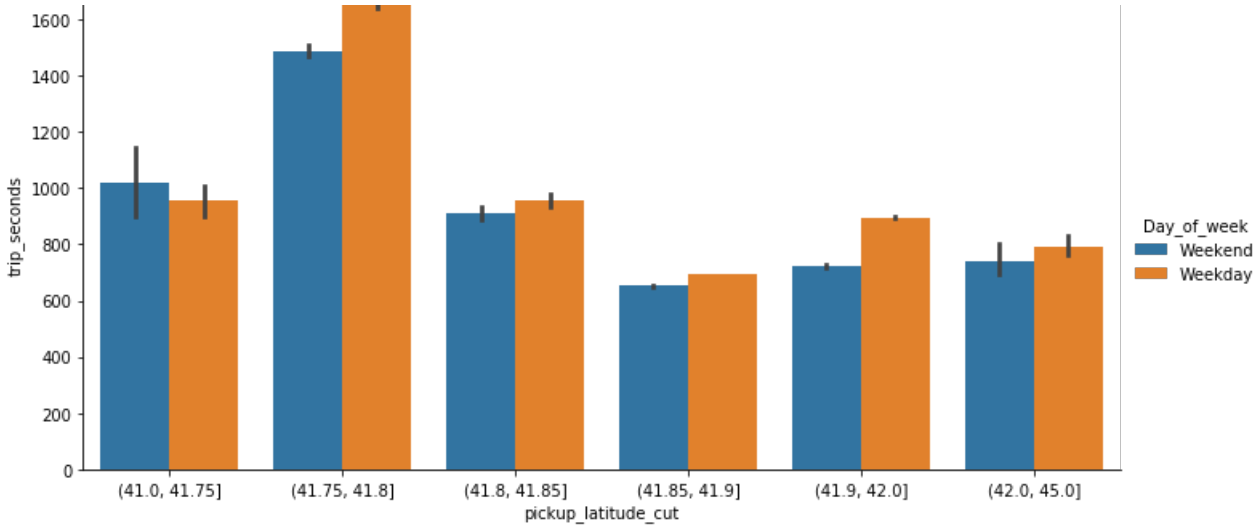


In [95]:

```
sns.catplot(x='pickup_latitude_cut',y='trip_seconds',hue='Day_of_week',kind='bar',data=df,aspect=2)
```

Out[95]:

```
<seaborn.axisgrid.FacetGrid at 0x10f73bff128>
```



In [217]:

```
dfr=df.copy()
```

In [218]:

```
dfr=df2[(df2.trip_seconds!=0) & (df2.fare!=0)]
#can't compute next things with zero in denom
```

In [219]:

```
dfr['miles_per_sec']=df2.trip_miles/df2.trip_seconds
```

In [220]:

```
dfr['revenue_per_sec']=df2.trip_total/df2.trip_seconds
```

In [223]:

```
dfr.groupby('pickup_latitude_cut')['miles_per_sec'].mean()
```

Out[223]:

```
pickup_latitude_cut
(41.0, 41.75]     0.003490
(41.75, 41.8]     0.013854
(41.8, 41.85]     0.011537
(41.85, 41.9]     0.004147
(41.9, 42.0]      0.007746
(42.0, 45.0]      0.004679
Name: miles_per_sec, dtype: float64
```

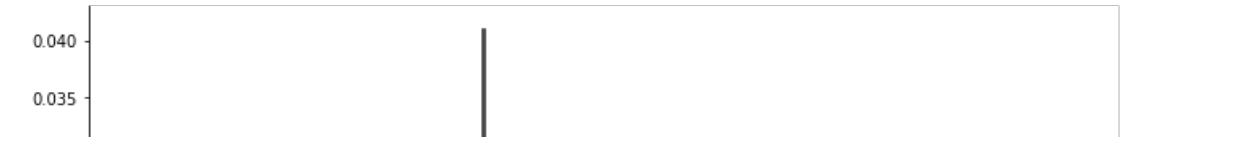
Revenue per sec while driving is about the same, however miles per second is about 3 times as much in south of Chicago than in Chicago probably due to traffic

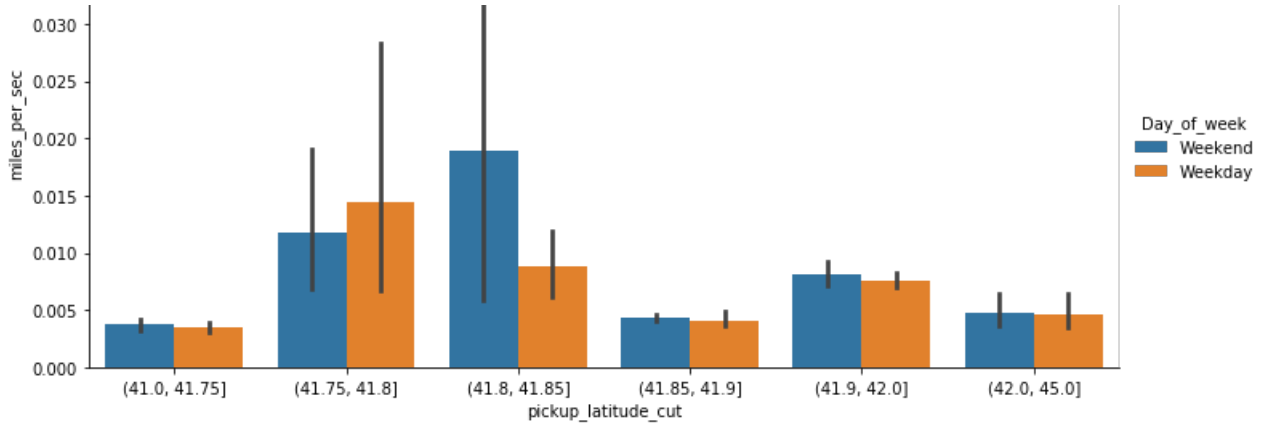
In [221]:

```
sns.catplot(x='pickup_latitude_cut',y='miles_per_sec',hue='Day_of_week',kind='bar',data=dfr,aspect=
2)
```

Out[221]:

```
<seaborn.axisgrid.FacetGrid at 0x10e08a33208>
```



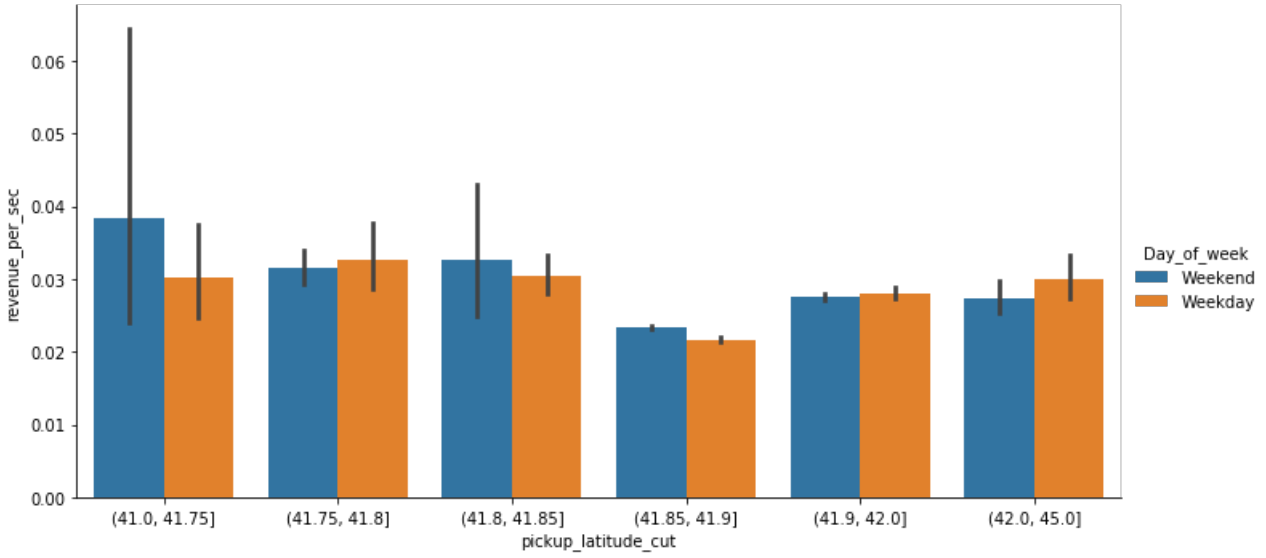


In [222]:

```
sns.catplot(x='pickup_latitude_cut',y='revenue_per_sec',hue='Day_of_week',kind='bar',data=dfr,aspect=2)
```

Out[222]:

```
<seaborn.axisgrid.FacetGrid at 0x10e08a2ecf8>
```

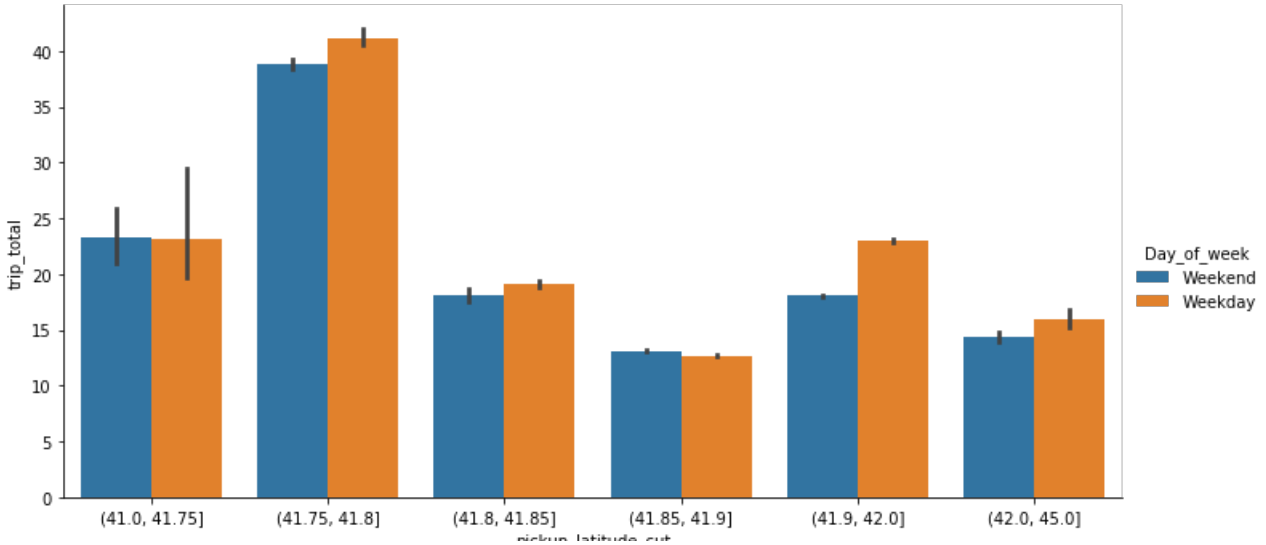


In [239]:

```
sns.catplot(x='pickup_latitude_cut',y='trip_total',hue='Day_of_week',kind='bar',data=dfr,aspect=2)
```

Out[239]:

```
<seaborn.axisgrid.FacetGrid at 0x10e08a588d0>
```



pickup_latitude_cut

CITY OF CHICAGO TAXICAB FARE RATES and INFORMATION

FARE RATES as of January 1, 2016

Flag Pull (Base Fare)	$ 3.25
Each additional mile	$ 2.25
Every 36 seconds of time elapsed	$ 0.20
First additional passenger	$ 1.00
Each additional passenger after first passenger*	$ 0.50
Vomit Clean-up Fee	$50.00
Illinois Airport Departure Tax**	$ 4.00

In [227]:

```
dfr.corr()
```

Out[227]:

	trip_seconds	trip_miles	fare	tips	tolls	extras	trip_total	pickup_latitude	pickup_longitude	drop
trip_seconds	1.000000	0.132681	0.368569	0.339868	0.042777	0.038256	0.337818	0.130653	-0.322486	
trip_miles	0.132681	1.000000	0.123905	0.125604	0.006092	0.017243	0.117004	0.058491	-0.130726	
fare	0.368569	0.123905	1.000000	0.309943	0.303833	0.043547	0.826877	0.123816	-0.303801	
tips	0.339868	0.125604	0.309943	1.000000	0.018512	0.063229	0.369049	0.131302	-0.393087	
tolls	0.042777	0.006092	0.303833	0.018512	1.000000	0.009024	0.248809	0.016436	-0.028723	
extras	0.038256	0.017243	0.043547	0.063229	0.009024	1.000000	0.590590	0.033882	-0.084586	
trip_total	0.337818	0.117004	0.826877	0.369049	0.248809	0.590590	1.000000	0.126649	-0.318198	
pickup_latitude	0.130653	0.058491	0.123816	0.131302	0.016436	0.033882	0.126649	1.000000	-0.569925	
pickup_longitude	-0.322486	-0.130726	-0.303801	-0.393087	-0.028723	-0.084586	-0.318198	-0.569925	1.000000	
dropoff_latitude	0.093108	0.032119	0.083585	0.073294	0.000680	0.003274	0.073152	0.353006	-0.127479	
dropoff_longitude	-0.256770	-0.089607	-0.236641	-0.222796	-0.003429	-0.007756	-0.207785	-0.133263	0.114434	
time_start_compare	0.033745	-0.001517	-0.000571	0.020360	-0.001011	0.004127	0.003766	-0.050413	-0.022736	
time_end_compare	0.040574	0.002278	0.007817	0.030633	-0.001504	0.004619	0.011483	-0.049489	-0.027981	
tips_true	0.097084	0.036161	0.087539	0.656834	0.001796	0.012140	0.137616	-0.002491	-0.106453	
big_tipper	0.373929	0.142337	0.340769	0.821286	0.015030	0.058662	0.372969	0.144493	-0.445556	
miles_per_sec	-0.003675	0.094120	0.013605	-0.000181	0.000073	0.000805	0.010915	0.000475	-0.005108	
revenue_per_sec	-0.022869	0.025309	0.215603	0.024704	0.087725	0.372807	0.374736	0.007813	-0.017429	
credit_card_true	0.098101	0.038361	0.092182	0.629218	0.002479	0.015485	0.140621	0.000468	-0.112582	

By utilizing the above information found on chicago.gov site, we can figure out how important each attribute is to how much money one cab driver can make

In [224]:

```
dfr.groupby('pickup_latitude_cut')['trip_seconds'].mean()
```

Out[224]:

```
pickup_latitude_cut
(41.0, 41.75]     1121.647563
(41.75, 41.8]     1698.328547
(41.8, 41.85]     1057.092571
```

```
(41.85, 41.9]      694.980061
(41.9, 42.0]      1020.161169
(42.0, 45.0]       882.568279
Name: trip_seconds, dtype: float64
```

In [229]:

```
dfr.groupby('pickup_latitude_cut')['trip_miles'].mean()
```

Out[229]:

```
pickup_latitude_cut
(41.0, 41.75]     4.228906
(41.75, 41.8]     9.541302
(41.8, 41.85]     4.423950
(41.85, 41.9]     2.498210
(41.9, 42.0]      5.363605
(42.0, 45.0]      3.160857
Name: trip_miles, dtype: float64
```

In [236]:

```
print ('The amount of rides one driver can do in Chicago versus south is ' + str(1698/694))
print ('The base rate amount for one ride south of Chicago is 3.25')
print ('The base rate amount for one ride in Chicago for as many trips as south of Chicago is ' +
str(1698/694*3.25))
```

```
The amount of rides one driver can do in Chicago versus south is 2.446685878962536
The base rate amount for one ride south of Chicago is 3.25
The base rate amount for one ride in Chicago for as many trips as south of Chicago is
7.9517291066282425
```

In [231]:

```
print ('The amount made total from miles south of Chicago would be ' + str(9.54*2.25))
print ('The amount made total from miles in Chicago would be ' + str(2.49*2.25))
```

```
The amount made total from miles south of Chicago would be 21.464999999999996
The amount made total from miles in Chicago would be 5.602500000000001
```

In [232]:

```
print ('The amount made total from trip seconds south of Chicago would be ' + str(1698.329/36*.2))
print ('The amount made total from trip seconds in Chicago would be ' + str(694.98/36*.2))
```

```
The amount made total from trip seconds south of Chicago would be 9.435161111111112
The amount made total from trip seconds in Chicago would be 3.861
```

In [296]:

```
print ('Most one could make in one trip on average south of Chicago is ' + str(3.25+21.465+9.4))
print ('Most one could make in Chicago on average compared to one trip south of Chicago is ' + str
(7.95+5.6+3.86))
```

```
Most one could make in one trip on average south of Chicago is 34.115
Most one could make in Chicago on average compared to one trip south of Chicago is 17.41
```

Traveling many miles fast is the best

Finding

Based on Chicago Taxi Cab Data most business is in Chicago, but can make much more per ride south of Chicago (about double based on rates). Also, day of week seems to have little effect

Insight

Large companies should stay in Chicago to maximize business opportunties due to many people. However, because long trips mainly occur south of Chicago and they are usually pretty fast, individual cab drivers may want to begin their trips South of Chicago to maximize total profit.

Finding 2- Taxi Efficiency Based on Time of Day

In [98]:

```
df.head()
```

Out[98]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

5 rows × 25 columns

Prepare dummies for time of day

In [240]:

```
dft=df.copy()
```

In [241]:

```
dft['starthour']=dft.time_start.str[0:2]
```

In [242]:

```
dft['starthour'] = dft.starthour.astype(int)
```

In [243]:

```
dft['time_0-6'] = dft.starthour.apply(lambda x: 1.0 if x<=6 else 0.0)
```

In [244]:

```
dft['time_6-12'] = dft.starthour.apply(lambda x: 1.0 if (x<=12 & x>6) else 0.0)
```

In [245]:

```
dft['time_12-18'] = dft.starthour.apply(lambda x: 1.0 if (x<=18 & x>12) else 0.0)
```

In [246]:

```
dft['time_18-24'] = dft.starthour.apply(lambda x: 1.0 if x>18 else 0.0)
```

In [248]:

```
dft.head()
```

Out[248]:

	taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_community_area	dropoff_community_area
2615738	4016	2016-2-13 17:15:00	2016-2-13 17:45:00	1080.0	2.8	8	28
11107382	4387	2016-6-16 20:30:00	2016-6-16 20:30:00	300.0	0.5	24	24
12285774	4865	2016-7-14 15:45:00	2016-7-14 16:15:00	2340.0	11.0	28	56
4176652	727	2016-3-14 03:00:00	2016-3-14 03:00:00	420.0	0.0	6	3
18263903	2452	2016-11-16 21:15:00	2016-11-16 21:15:00	600.0	1.6	Unknown	Unknown

5 rows × 37 columns

Prepare for clustering analysis

In [249]:

```
from sklearn.cluster import KMeans
```

In [250]:

```
clu = KMeans(n_clusters=3, random_state=0, max_iter=3000)
```

In [251]:

```
dft=dft.drop(['taxi_id','trip_start_timestamp','trip_end_timestamp','pickup_community_area','dropoff_community_area',\
         'payment_type','company','pickup_latitude','pickup_longitude','dropoff_latitude','dropoff_longitude',\
         'date_start', 'time_start', 'date_end', 'time_end',\
        'Day_of_week', 'geometry', 'pickup_latitude_cut', 'time_start_compare',\
          'time_end_compare'],axis=1)
```

In [252]:

```
dft=dft[(dft.trip_seconds!=0) & (dft.fare!=0)]
#can't compute next items with zero in denom
```

Create variable for whether or not a customer tips

In [253]:

```
dft['tipper']=dft.tips.apply(lambda x: 1.0 if x>0 else 0.0)
```

In [255]:

```
dft=dft.drop(['bin_tips','bin_time','tips_true','bin_fare','big_tipper'],axis=1)
```

In [256]:

```
dft.head()
```

Out[256]:

	trip_seconds	trip_miles	fare	tips	tolls	extras	trip_total	starthour	time_0-6	time_6-12	time_12-18	time_18-24	tipper
2615738	1080.0	2.8	12.75	2.0	0.0	1.0	15.75	17	0.0	0.0	0.0	0.0	1.0
11107382	300.0	0.5	5.00	0.0	0.0	1.0	6.00	20	0.0	0.0	0.0	1.0	0.0

12285774	2340.0	11.9	30.50	6.2	0.0	0.0	37.20	15	0.0	0.0	0.0	1.0	0.0	1.0
4176652	420.0	0.0	7.50	0.0	0.0	1.5	9.00	3	1.0	0.0	0.0	0.0	0.0	0.0
18263903	600.0	1.6	8.25	3.0	0.0	1.0	12.25	21	0.0	0.0	0.0	1.0	1.0	

Create efficieny variable of trip revenue/trip time

In [257]:

```
dft['revenue_per_sec']=dft.trip_total/dft.trip_seconds
```

In [258]:

```
dft['miles_per_sec']=dft.trip_miles/dft.trip_seconds
```

In [259]:

```
clu
```

Out[259]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=3000,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=0, tol=0.0001, verbose=0)
```

In [260]:

```
clu.fit(dft)
```

Out[260]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=3000,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=0, tol=0.0001, verbose=0)
```

In [261]:

```
clu.labels_
```

Out[261]:

```
array([0, 0, 2, ..., 0, 0, 0])
```

In [263]:

```
dft['cluster'] = clu.labels_
```

In [264]:

```
dft.groupby('cluster').mean()
```

Out[264]:

cluster	trip_seconds	trip_miles	fare	tips	tolls	extras	trip_total	starthour	time_0-6	time_6-12	time_12-18	time_1
0	594.985004	2.208847	10.033706	1.082239	0.001612	0.706507	11.873733	13.664939	0.143915	0.089942	0.118803	0.2767
1	52102.507194	13.187302	72.511583	1.132698	0.865108	0.678777	75.241223	12.082734	0.104317	0.169065	0.075540	0.1187
2	2421.573478	13.712285	39.445912	4.991370	0.010737	3.458138	47.990896	14.413481	0.057451	0.096227	0.167396	0.2123

Clusters 0 and 2 have the most efficiency and have the highest portions of their rides occuring in the time 0-6 and time 18-24 slots. These slots tend to have much lower trip times and and trip times suggesting that shorter trips may be more efficient

In [266]:

```
dft['rev_sec_bin']=pd.qcut(dft.revenue_per_sec, 5)
```

In [267]:

```
dft['timebin']=pd.cut(dft.starthour, bins=[0,6,12,18,24])
```

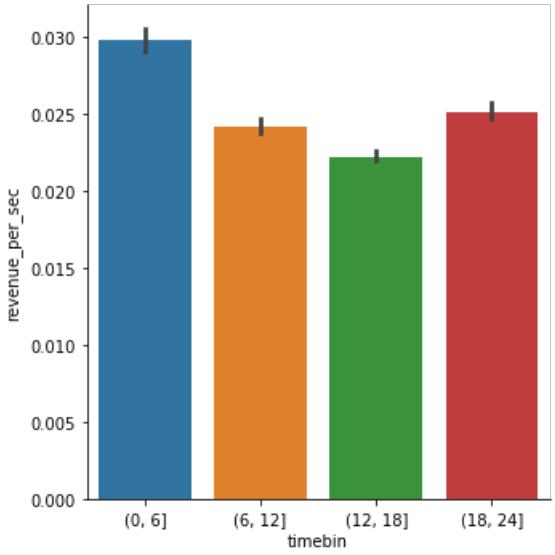
In [269]:

```
sns.catplot(y='revenue_per_sec',x='timebin', kind='bar', data=dft)
```

```
C:\Users\Robby Konrath\Documents\Python\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tu
ple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[269]:

```
<seaborn.axisgrid.FacetGrid at 0x10e0897b470>
```



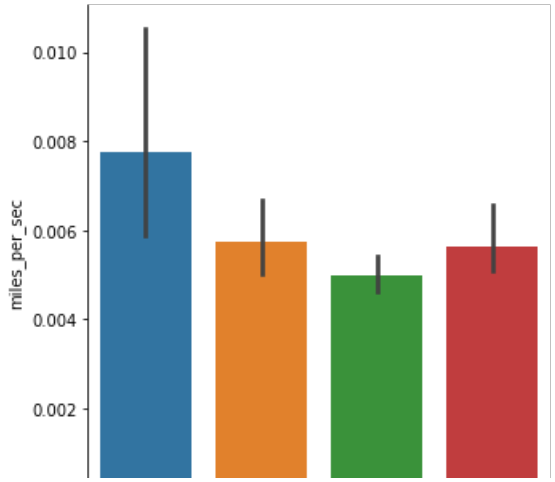
20% faster^

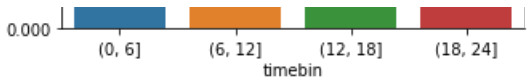
In [270]:

```
sns.catplot(y='miles_per_sec',x='timebin', kind='bar', data=dft)
```

Out[270]:

```
<seaborn.axisgrid.FacetGrid at 0x1104a03ba58>
```





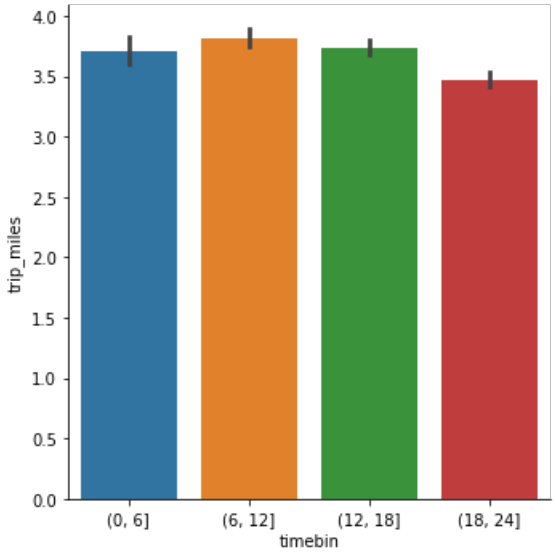
33% faster^

In [271]:

```
sns.catplot(y='trip_miles',x='timebin', kind='bar', data=dft)
```

Out[271]:

```
<seaborn.axisgrid.FacetGrid at 0x10e0ca6e5f8>
```



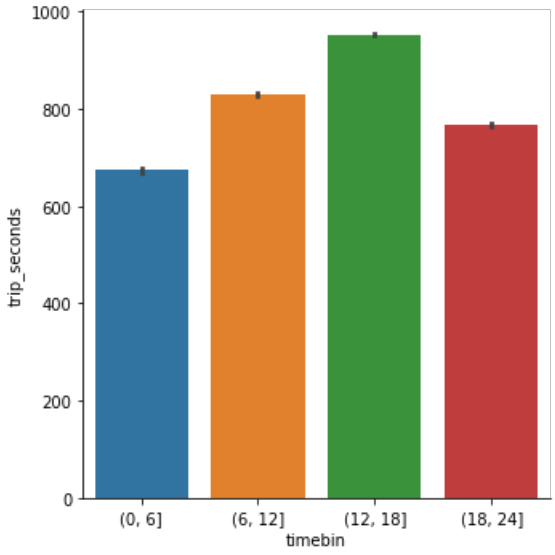
^same amount of miles usually

In [272]:

```
sns.catplot(y='trip_seconds',x='timebin', kind='bar', data=dft)
```

Out[272]:

```
<seaborn.axisgrid.FacetGrid at 0x10e0c77cc88>
```



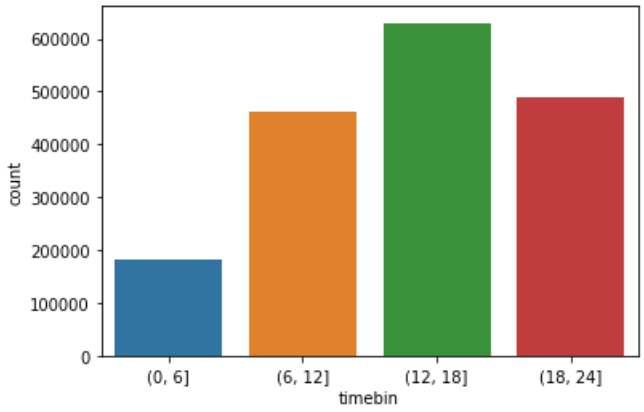
^much faster to go those miles

In [273]:

```
sns.countplot(x='timebin',data=dft)
```

Out[273]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x10e0c776e48>
```



Summary

Rides that occur between the hours of midnight to 6am and 6pm to midnight tend to be the most efficient in terms of revenue per second of ride time as well as miles per second. Miles per second is faster and rides that drive a large amount of miles fast tend to make the most as we know from finding one

Insight

Taxis that want to make money most efficiently should operate between 6pm and 6am rather than more normal business hours. However, after 6 pm may be best due to volume of people out

Finding 3- Credit Card Users and Tips

In [274]:

```
dfm=df.copy()
```

In [275]:

```
dfm = dfm[(dfm.payment_type == 'Cash') | (dfm.payment_type == 'Credit Card')]
```

In [277]:

```
dfm2 = dfm[['payment_type', 'trip_total', 'trip_miles']]
```

In [278]:

```
dfm2 = dfm2[dfm2.trip_miles != 0]
```

In [279]:

```
dfm2['binned_tripmiles']=pd.qcut(dfm2.trip_miles, 5)
```

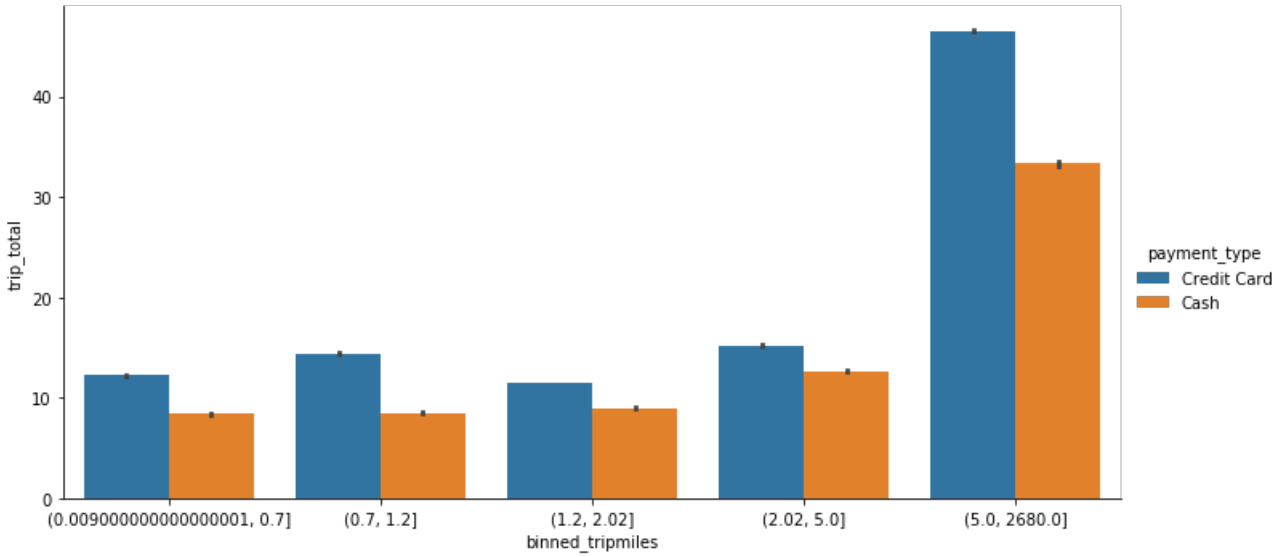
In [280]:

```
sns.catplot(y='trip_total', data=dfm2, x='binned_tripmiles', aspect=2, kind='bar', hue = 'payment_type')
```

```
C:\Users\Robby Konrath\Documents\Python\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index,
`arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[280]:

<seaborn.axisgrid.FacetGrid at 0x10e0c75ec18>



In [283]:

```
dfa=df.copy()
dfa = dfa[dfa.trip_seconds !=0]
dfa = dfa[dfa.trip_total != 0]
#can't compute next thing with zero in denom
```

In [284]:

```
dfa['cost_per_mile']=dfa.trip_total/dfa.trip_miles
```

In [289]:

```
dfaCost=dfa.copy()
```

In [290]:

```
dfaCost = dfaCost[(dfaCost.cost_per_mile > .01) & (dfaCost.cost_per_mile < .7)]
```

In [291]:

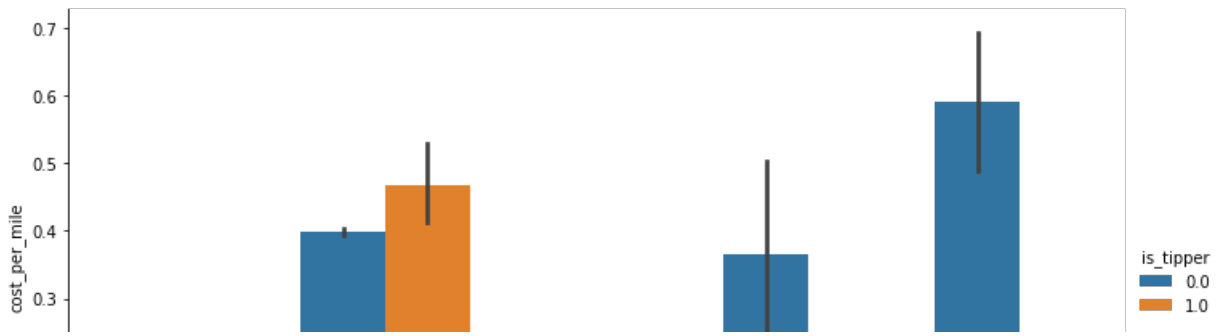
```
dfaCost['is_tipper'] = dfaCost.tips.apply(lambda x: 1.0 if x>0 else 0.0)
```

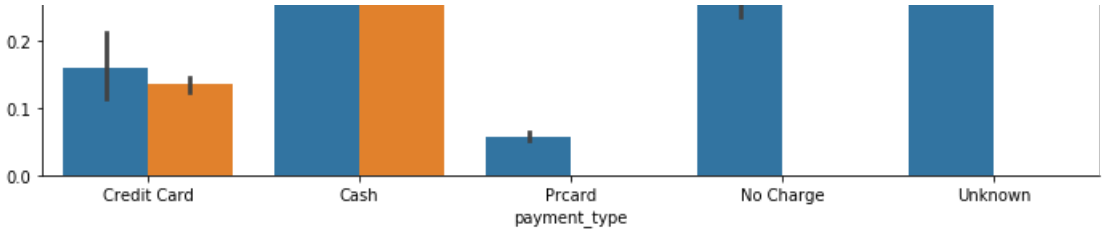
In [292]:

```
sns.catplot(x='payment_type', y='cost_per_mile', data=dfaCost, hue = 'is_tipper', kind='bar', aspect =2)
```

Out[292]:

<seaborn.axisgrid.FacetGrid at 0x10e0cb644a8>



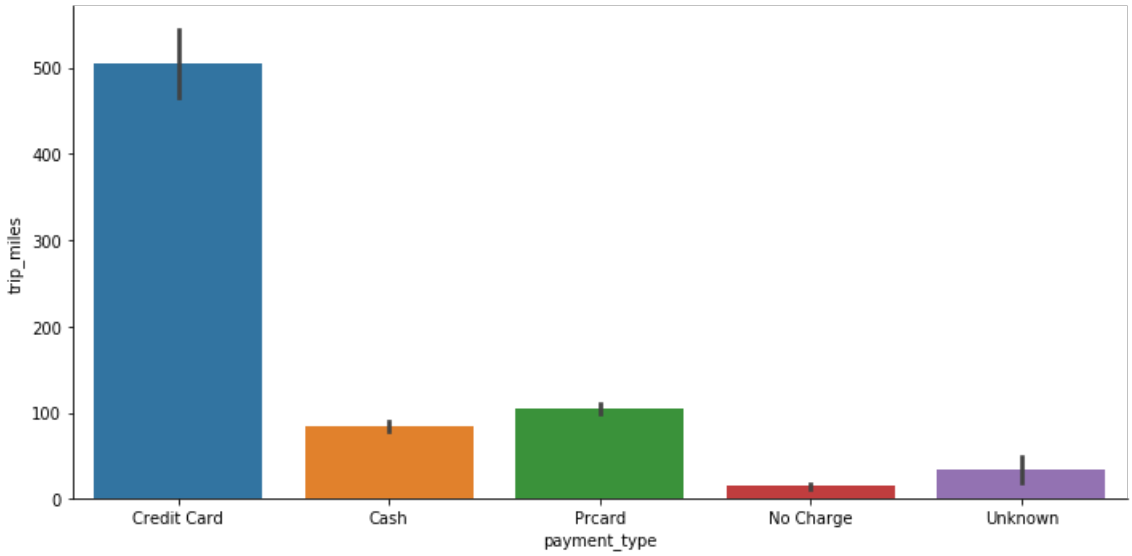


In [293]:

```
sns.catplot(x='payment_type',y='trip_miles',kind='bar',data=dfaCost,aspect=2)
```

Out[293]:

```
<seaborn.axisgrid.FacetGrid at 0x11149d5c668>
```

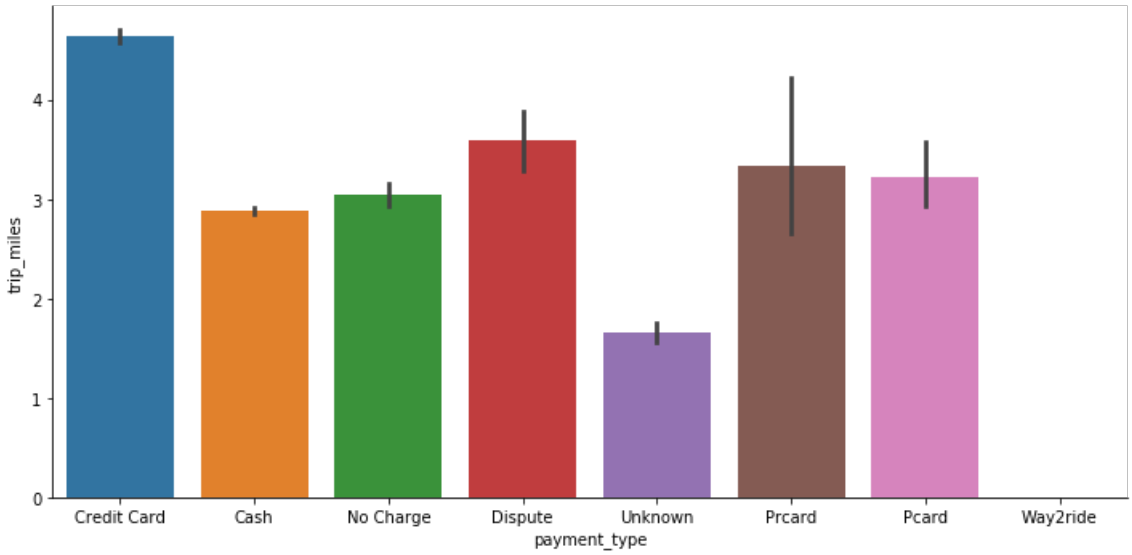


In [295]:

```
sns.catplot(x='payment_type',y='trip_miles',kind='bar',data=dfa,aspect=2)
```

Out[295]:

```
<seaborn.axisgrid.FacetGrid at 0x11149f62da0>
```

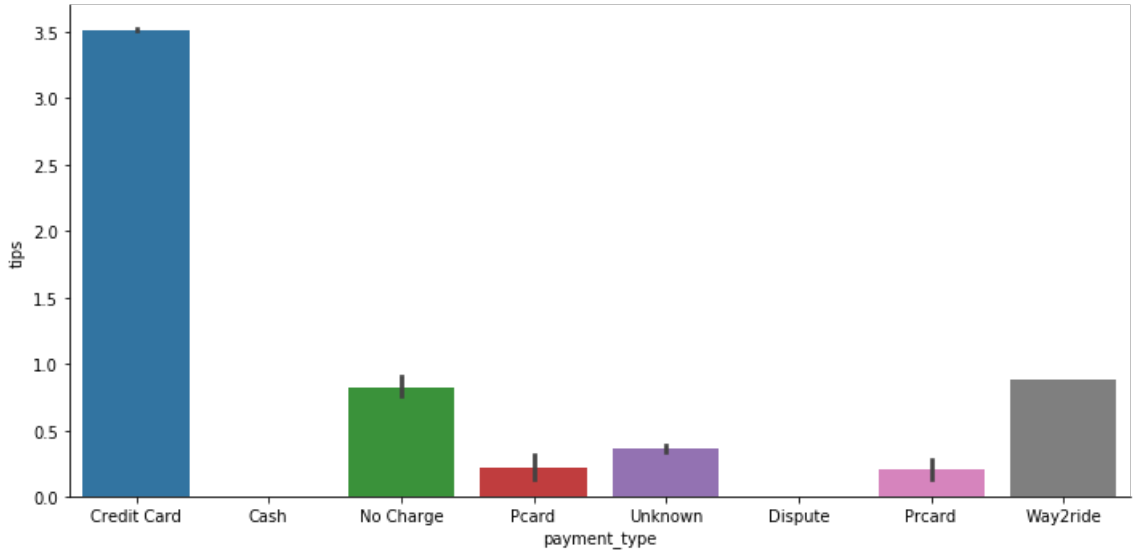


In [299]:

```
sns.catplot(x='payment_type',y='tips',kind='bar',data=df,aspect=2)
```

Out[299]:

```
<seaborn.axisgrid.FacetGrid at 0x11149be6f28>
```



Summary

The first barplot shows the relationship between trip_miles and trip_total paid with hues on Credit and Cash. The trip_miles are binned with equal frequencies. Credit card payments tended to range higher than cash payments for all distances.

The second barplot shows that Credit Card users typically have a lower cost per mile suggesting that they have longer rides and cash users have a very high cost per mile and this indicates they will spend more time in traffic.

Also, credit card users may see a screen with a tip amount when swiping that could lead them to tip more

Insight

Credit card users tend to both pay more for every binned distance traveled and on average have a longer trip distance, this leads us to say that credit card users are more likely to tip so all taxi companies should incentivize credit card payments with some sort of discount program.

df.to_csv('OMIS114bestclean.csv',index=0)