Instructions:

this dataset is a longitudinal survey of recipients of job training. They were given training on how to use freelance work to increase their income. The survey asks about gig work income, gig type, and opinions about the work compared to regular employee work.

Important notes:

1. All work should be done in this notebook. Everything should be replicable. It is live and monitored.
2. At the top of each cell, write your initials or some code indicating that this is your code. If you change someone else's code, notate that incomments
3. In contingency or other types of tables, include an N count. If it's a median of income drivers working 15 hours a week/ how many respondents is that? If its a median of 5, there should be a column in the table that has a value of N=5.
4. Update: when creating tables, please use pandas GroupBy feature

https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html

It's important because we want to be able to investigate by different types of subgroups and apply different functions (like, for instance, find out the interquartile range of Uber drivers working 15 hours a week)

-- The data for the notebook is in three google sheets. The first tab has question wording of each variable

data sorces. you will have to load them separate and maybe change the path name

https://docs.google.com/spreadsheets/d/1SrNdTp7mZAf6CdF6sWavZGMk3lhZgTxl8mDc6uUWdw0/edit?usp=sharing

https://docs.google.com/spreadsheets/d/1Y439FpgAzmec7VbLvdqsoI_GN4YNjNFTCzQtDx66W9k/edit?usp=sharing

https://docs.google.com/spreadsheets/d/11kyMbR_SpqA-aNdJ2A2cQnIwss1DuBM2x3FAM92Xqss/edit?usp=sharing

The goal of the survey is to see how income from gig work changes over time. Some respondents will do no gig work. Some will do a lot. This investigates how income changes, and how it is associated with type of gig work, hours work, if they have another job or their opinions on gig work.

load data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import re
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.options.mode.chained_assignment = None
pd.set_option('display.max_rows', 250)
pd.set_option('display.max_columns', 250)


#garrick code
#load dataframes and join

#df6 = pd.read_csv('/content/Samaschool_6 Month Data_T4A - 6Month.csv', index_col=0)
#df9 = pd.read_csv('/content/Samaschool_9 Month Data_T4A - 9Month.csv', index_col=0)
#df1 = pd.read_csv('/content/Samaschool_1 Month Data_T4A - 1Month.csv', index_col=0)

##greg addition file name

df6 = pd.read_csv('/content/Samaschool_6 Month Data_T4A - 6Month.csv', index_col=0)
df9 = pd.read_csv('/content/Samaschool_9 Month Data_T4A - 9Month.csv', index_col=0)
df1 = pd.read_csv('/content/Samaschool_1 Month Data_T4A - 1Month.csv', index_col=0)



df1 = df1.rename(columns=df1.iloc[1], index = {'6101460':'101460'})#stray 6?
df6 = df6.rename(columns=df6.iloc[1])
df9 = df9.rename(columns=df9.iloc[1])

df1.drop(df1.index[:3], inplace=True)
df6.drop(df6.index[:3], inplace=True)
df9.drop(df9.index[:3], inplace=True)

print(df1.shape)
print(df6.shape)
print(df9.shape)

df_merge = df1.join(df6, how='outer').join(df9, how='outer')

print(df_merge.shape)
#df_merge.head()
```

recode hours and income and merge files

```
##recode . with _

df_merge.columns = df_merge.columns.str.replace(".", "_")


#garrick code; converted reported hours to range
hours = {'35-50 hours per week':40, '15-34 hours per week':25,
         '1-14 hours per week':8, 'Varied a lot by week':np.nan,
         'More than 50 hours per week':60, np.nan:np.nan}

df_merge[['_emp_hours_6', '_emp_hours_9']] = df_merge[['_emp_hours_6', '_emp_hours_9']].apply
df_merge[['_emp_hours_6', '_emp_hours_9']].fillna(0, inplace=True)


#gregcode to check if hours coded didn't result in additional NaNs
#dfm = df_merge

df_merge['_emp_hours_9'].value_counts(dropna=False)
```

```
## greg code

df_merge.columns
```

```
# garrick code: normalize monthly income for gig and employer work
income = ['_indw_inc_total_1',
          '_emp_annsalary_6', '_emp_hourly_6',
          '_indw_inc_total_6', '_indw_month_6',
```

```
          '_emp_annsalary_9', '_emp_hourly_9',
          '_indw_inc_total_9', '_indw_month_9']




df_income = df_merge.copy()

df_income[income] = df_income[income].replace('\,', '', regex=True)
df_income[income] = df_income[income].replace('[0-9]+\-', '', regex=True) #replace ranges wit
df_income[income] = df_income[income].replace('[^.0-9]+', '', regex=True)
df_income[income] = df_income[income].astype(float)

df_income['gig_monthly_6'] = df_income['_indw_inc_total_6'] / df_income['_indw_month_6']
df_income['gig_monthly_9'] = df_income['_indw_inc_total_9'] / df_income['_indw_month_9']

##filling na's with mean? - greg
df_income[['_emp_annsalary_6', '_emp_annsalary_9']].fillna(
    df_income[['_emp_annsalary_6', '_emp_annsalary_9']].mean(),
    inplace=True)
df_income[['_emp_annsalary_6', '_emp_annsalary_9']].fillna(0, inplace=True)

##filling is questionable
emp6 = ['_emp_hourly_6','_emp_hours_6','_emp_annsalary_6']
df_income['emp_monthly_6'] = df_income[emp6].apply(lambda row:
                             max(row[emp6[0]] * row[emp6[1]] * 4, row[emp6[2]]/12),
                             axis = 1)

emp9 = ['_emp_hourly_9','_emp_hours_9','_emp_annsalary_9']
df_income['emp_monthly_9'] = df_income[emp9].apply(lambda row:
                             max(row[emp9[0]] * row[emp9[1]] * 4, row[emp9[2]]/12),
                             axis = 1)




##greg turn NAs to zeros
#df_income['_indw_inc_total_1'] = df_income['_indw_inc_total_1'].fillna(0)


# garrick code
# gig_income_13, gig_income_36, gig_income_19
```

```
#df_income['gig_income_16'] = (df_income['gig_monthly_6'] - df_income['_indw_inc_total_1'])/6


df_income['gig_income_69'] = (df_income['gig_monthly_9'] - df_income['gig_monthly_6'])/3
#df_income['gig_income_19'] = (df_income['gig_monthly_9'] - df_income['_indw_inc_total_1'])/9

#greg code replace nans in month 1 with zeros
df_income['gig_income_16'] = (df_income['gig_monthly_6'] - df_income['_indw_inc_total_1'].rep
df_income['gig_income_19'] = (df_income['gig_monthly_9'] - df_income['_indw_inc_total_1'].rep

income_changes = ['gig_income_16', 'gig_income_69', 'gig_income_19']
df_income['gig_income_change'] = df_income[income_changes].apply(lambda row:
        max(row['gig_income_16'], row['gig_income_69'], row['gig_income_19']),
        axis = 1)

df_income[['emp_monthly_6', 'emp_monthly_9']] = df_income[['emp_monthly_6',
                                                           'emp_monthly_9']].replace(
                                                               0,
                                                               np.nan)

df_income['gig_vs_emp_6'] = df_income['gig_monthly_6'] / df_income['emp_monthly_6']
df_income['gig_vs_emp_9'] = df_income['gig_monthly_9'] / df_income['emp_monthly_9']

print(df_income.shape)
income_changes += ['gig_income_change', 'gig_vs_emp_6', 'gig_vs_emp_9']
df_income[income_changes].describe().round(2)

#greg replacement zoers code in alt column

#dftt['_indw_inc_total_1'] = dftt['_indw_inc_total_1'].replace(np.nan, 0)
df_income['alt_gig_income_16'] = (df_income['gig_monthly_6'] - df_income['_indw_inc_total_1']

print(df_income['gig_income_16'].describe())
```

```
##greg looking at zeros status

dftt = df_income



dftt['gig_income_16'] = (dftt['gig_monthly_6'] - dftt['_indw_inc_total_1'].replace(np.nan,0))
```

```
#dftt['monthly']

#dftt['_indw_inc_total_1'].fillna('NaN', inplace=True)

dftt['_indw_inc_total_1'] = dftt['_indw_inc_total_1'].replace(np.nan, 0)

dftt['_indw_inc_total_1'].value_counts(dropna=False)

ls2 = (dftt['gig_monthly_6'] - dftt['_indw_inc_total_1'])/6

#(df_income['gig_monthly_6'] - df_income['_indw_inc_total_1'].replace(np.nan, 0))/6

df_income.iloc[1]

ls = (df_income['gig_monthly_6'] - df_income['_indw_inc_total_1'])/6

print(ls.describe())
#ls2.value_counts()
dftt['gig_income_16'].describe()
```

```
df_income['_indw_inc_total_6'].value_counts(dropna=False)
```

```
##greg code exploring first month gig income


#total after first month
#df_income['_indw.gigactivation_1'].value_counts()

#total income after first month
#df_income[df_income['_indw_inc_total_1'] > 0].describe()['_indw_inc_total_1']

#total income after 9 month
```

```
df_income[df_income['_indw_inc_total_6'] > 0].describe()['_indw_inc_total_6']
```

```
df_income.columns[df_income.columns.str.contains('hours', na=False)]
```

```
##greg code status##

def q1(x):
    return x.quantile(0.25)

def q2(x):
    return x.median()

def q3(x):
    return x.quantile(0.75)

#df_income.columns

#df_income.gig_services.value_counts()

df_income.groupby(['_indw_hours_6']).agg([np.size, np.median, np.std])['_indw_inc_total_6']

#df_income.groupby(['_indw_hours_6']).agg([np.size, np.median, np.std])['gig_income_16']

df_income.groupby(['_indw_hours_9']).agg([np.size, np.median, q1, q2, q3])[['gig_income_chang
```

```
df_income.groupby(['_indw_hours_6']).agg([np.size, np.median])[['gig_vs_emp_6','gig_income_16
```

```
#df_income.groupby(['_indw.gigactivation_1']).describe()['_indw_inc_total_9']

#df_income.reset_index(inplace=True)

(df_income
 .filter(['gig_income_16'])
 .head(3))
```

recode job categories

```
#df_income[(df_income['_indw.gigactivation_1'] == 'No') & (df_income['_indw_inc_total_6'] > 0
df_income[(df_income['_indw_inc_total_6'] > 0) & (df_income['_indw_inc_total_9'] > 0) ]
```

```
# garrick code:  gig categories
gig_cols1 = ['_indw_online_care_1', '_indw_online_field_1',
           '_indw_online_handy_1', '_indw_online_lyft_1', '_indw_online_post_1',
           '_indw_online_taskrab_1', '_indw_online_thumb_1', '_indw_online_upwork_1',
           '_indw_online_urbansit_1', '_indw_online_uber_1', '_indw_online_wonolo_1',
           '_indw_online_workmark_1', '_indw_online_other_1']

gig_cols6 =  ['_indw_service_care_6',
              '_indw_service_clean_6', '_indw_service_deliver_6',
              '_indw_service_maint_6', '_indw_service_personal_6',
              '_indw_service_it_6', '_indw_service_drive_6',
              '_indw_service_prof_6', '_indw_service_create_6',
              '_indw_service_other_6']

gig_cols9 =   ['_indw_service_care_9',
                '_indw_service_clean_9', '_indw_service_deliver_9',
                '_indw_service_maint_9', '_indw_service_personal_9',
                '_indw_service_it_9', '_indw_service_drive_9',
                '_indw_service_prof_9', '_indw_service_create_9',
                '_indw_service_other_9']

gig_cols = gig_cols1 + gig_cols6 + gig_cols9

# merge service columns into one
df_income['gig_services'] = df_income[gig_cols].apply(
    lambda x: '; '.join(list(set(x.dropna().astype(str)))),
    axis=1)

#remove prefixes i.e. '_indw_service_care_9' --> 'care_9'
df_gig = df_income.copy()
df_gig.rename(columns={x:re.sub('\_indw\.\w+e\.', '', x) for x in gig_cols}, inplace = True)

#greg code
#df_income.rename(columns=lambda x:re.sub('\_indw\.\w+e\.', '', x), inplace=True)



#garrick code
df_gig.columns.values


# garrick code
def contin(conditions, categories):
```

```
  table = df_gig.groupby(conditions).count()[categories]

  n_right = table.sum(axis = 1)
  n_right.name = 'SUM'
  table = pd.concat([table, n_right], axis=1, join='inner')
  n_bottom = table.sum(axis = 0)
  n_bottom.name = 'SUM'
  table = table.append(n_bottom)
  return table

# to generate contingency table
# step 1: create conditional columns
df_gig['increased_6-9'] = df_gig['gig_monthly_9'] > df_gig['gig_monthly_6']
# step 2: list conditional columns in groupby function
conditions = ['increased_6-9']
# step 3: list gig categories
categories = [ 'it_6','drive_6','maint_6']

contin(conditions, categories)
```

```
##greg code
#df_gig['gig_services'].value_counts()
df_gig['gig_levels'] = df_gig['gig_services']
df_gig['gig_levels'].value_counts()
```

```
#greg code status here
print(type(df_gig['gig_levels']))
#income_df['income'].replace(regex=True, inplace=True, to_replace=r'(\d+[\,][0-9]+\s)|(^[0-9]

routine = ['Lyft', 'Delivery', 'Uber', 'Postmates', 'Urban', 'Driving', 'shop', 'Door', 'Post
```

```
"""import pandas as pd
import re
pattern_1 = re.compile(r'\bstudent', re.IGNORECASE)
data = [['I am a teacher',0],['I am a student ',0],['Student group', 0]]

df = pd.DataFrame(data, columns =['A','B'])
print("orginal df:",df)
df['B'] = df.apply(lambda row: 1 if pattern_1.search(row.A) else row.B , axis=1)
print("\n\nmodified df:",df)


#from another answer
import re

dicti={'the':20, 'a':10, 'over':2}
patterns=['the', 'an?']
regex_matches = [re.compile("^"+pattern+"$").match for pattern in patterns]

extractddicti= {k:v for k,v in dicti.items()
                if any (regex_match(k) for regex_match in regex_matches)}


"""

import re
pattern_1 = [re.compile(r'\b'+pattern, re.IGNORECASE).match for pattern in routine]

df_gig['gig_levels_binary'] = df_gig.apply(lambda row: 1 if pattern_1 in row.gig_services els
print("\n\nmodified df:",df_gig['gig_levels_binary'].value_counts())

df_gig.loc[df_gig['gig_levels_binary'] == 1]['gig_services']


# another way to group if category has multiple values
# might be what you meant by the more usual way to use groupby
df_gig['_emp_hours_6'].fillna(df_gig['_emp_hours_6'].mean(), inplace = True)
table = df_gig.groupby(['_emp_hours_6'])['increased_6-9'].sum()
pd.DataFrame(table)


# grouping by the merged categories gets unwieldy because of the many unique mix of jobs
table = df_gig.groupby(['gig_services'])['increased_6-9'].sum()
pd.DataFrame(table)


#recode variables
```

Import Data old :

```
from google.colab import drive
drive.mount('/content/drive')


#shohan code
#import dataset
#df = pd.read_csv('/content/my_data.csv', index_col=0)

#data sorces. you will have to load them separate and maybe change the path name
#https://docs.google.com/spreadsheets/d/1SrNdTp7mZAf6CdF6sWavZGMk3lhZgTxl8mDc6uUWdw0/edit?usp
#https://docs.google.com/spreadsheets/d/1Y439FpgAzmec7VbLvdqsoI_GN4YNjNFTCzQtDx66W9k/edit?usp
#https://docs.google.com/spreadsheets/d/11kyMbR_SpqA-aNdJ2A2cQnIwss1DuBM2x3FAM92Xqss/edit?usp

df = pd.read_csv('/content/Samaschool_6 Month Data_T4A - 6Month.csv')
df2 = pd.read_csv('/content/Samaschool_9 Month Data_T4A - 9Month.csv')
df3 = pd.read_csv('/content/Samaschool_1 Month Data_T4A - 1Month.csv')

df = df.rename(columns=df.iloc[1])
df2 = df2.rename(columns=df2.iloc[1])
df3 = df3.rename(columns=df3.iloc[1])

df.drop(df.index[:3], inplace=True)
df2.drop(df2.index[:3], inplace=True)
df3.drop(df3.index[:3], inplace=True)



print('data 1 :', df.shape)
print('data 2 :', df2.shape)
print('data 3 :', df3.shape)


#from google.colab import drive
#drive.mount('/content/drive')
```

Clean Data :

2 cells hidden

Merge Data :

3 cells hidden

Apply proper data type on each columns

1 cell hidden

Seperate categorical data and neumreic data

4 cells hidden

Demography data

```
#Age
print('Age :\n\n', demo_df['_demo.age'].describe())


#gender
print('Gender : \n', demo_df['_demo.gender'].value_counts())


#education
tmpdf = demo_df.groupby('_demo.highestedu').count()
tmpdf.plot(kind='bar', y='_demo.age', title='Summary of Demography')
plt.show()

print(demo_df['_demo.highestedu'].value_counts())
```

Contingency Table of Demography data

```
demo_df.groupby(['_demo.highestedu','_demo.gender']).count()
```

Well Being data:

1 cell hidden

Harassment data :

1 cell hidden

Service Data :

1 cell hidden

Income Data:

These columns are messy. Does not have proper data type. So what I did here

- remove the tailing space
- replace special characters
- fill NULL value with default `0`
- match expression with regx (hardest part)
- add proper category

clean Income data

To clraify **income** and **hourly_income** columns:

- **income:** In the past 6 months, how much did you earn in total from your gig work, after deducting your business expenses from your income?
- **hourly_income:** *In the past 6 months, how much did you usually earn per hour for your gig work?*

```
#shohan code

income_df = service_df
income_df['income'] = merged_df['_indw_inc_total'] # _indw_inc_total_6
income_df['hourly_income'] = merged_df['_indw_hourly'] # _indw_hourlypay_6
# need to clean this data
income_df['work_hours'] = merged_df['_indw_hours'] #_indw_hours_6

#remove tailing space
income_df['income'] = income_df['income'].str.strip()
income_df['income'] = income_df['income'].str.replace(',','')

income_df['hourly_income'] = income_df['hourly_income'].str.strip()

#fill Null
income_df['income'] = income_df['income'].fillna(0)
income_df['hourly_income'] = income_df['hourly_income'].fillna(0)

#clean
income_df['income'].replace(regex=True, inplace=True, to_replace=r'(\d+[\,][0-9]+\s)|(^[0-9]+
income_df['hourly_income'].replace(regex=True, inplace=True, to_replace=r'^[0-9][0.0-9.0][\-]
income_df['hourly_income'].replace(regex=True, inplace=True, to_replace=r'\D+', value=r'') #2
```

```
#convert
income_df['income'] = income_df['income'].astype(float)
income_df['hourly_income'] = income_df['hourly_income'].astype(float)
income_df['work_hours'] = income_df['work_hours'].astype('category')

#income_df[income_df['income'] > 10000]['income']
#income_df['hourly_income']

#income_df.info()
#income_df.head()
#income_df.describe()
#income_df['work_hours'].unique()
```

Median Income data with N

```
#shohan code #(greg edit key)

# if the median income of uber driver is $100,
# i also need to know how many people that is collected from. If its the median of 5 drivers,

#tmpdf = pd.crosstab(index=[income_df.income, income_df.hourly_income], columns=[income_df.se

cols = ['service_care', 'service_clean', 'service_deliver', 'service_maintanance',
        'service_personal', 'service_it', 'service_driving', 'service_professional', 'service

income_median = income_df.groupby(['service_care'])[['income', 'hourly_income']].median()
income_median['N'] = income_df['service_care'].value_counts()
#print(income_median)

for col in cols:
  income_median = income_df.groupby([col])[['income', 'hourly_income']].median()
  income_median['N'] = income_df[col].value_counts()
  print(income_median)

#print(income_median)
```

Contengency Table of Income, Hourly-Income, Working-Hours

```
#shohans code

#print(income_df['work_hours','income','hourly_income'])['service_driving'].count())
#print(income_df['service_it'].count())

tt = income_df.loc[ income_df['service_driving'].notnull(), ['income', 'hourly_income', 'work
tmpdf = pd.crosstab(index=[tt.work_hours], columns=[tt.hourly_income, tt.income], margins=Tru
display(tmpdf)
```

```
#print(tt.groupby(['work_hours','income','hourly_income']).head())

#print(income_df['service_driving'].count())
```

What is the gig income by peole in different industries by hours-work ?

Median income of per month :

1 cell hidden

Graph of Median income

1 cell hidden

Summary of Mean Income data

1 cell hidden

Employment preference Data

1 cell hidden

Unemployment Data :

1. **unemp_earner :** In the past 6 months, did you have any periods of unemployment, meaning any time that you wanted to work to earn an income but were not able to?
2. **unemp_months**: How many months were you unemployed?
3. **unemp_reason:** Employment reason ?
4. **other_unemp_reason**: Other employment reason ?

1 cell hidden

Sell Data

1 cell hidden

Business Data

> mostly empty, only 2-3 field is not empty

1 cell hidden

Freelance/Gig Data:

1. Are you interested in more training on how to be a successful freelance, gig, or independent contract worker?
2. What topics are you most interested in for more training? You can choose more than one option, so choose as many as apply to you.

5 cells hidden

Benifit Data

3 cells hidden

Concern Data

- As a freelance, gig, or independent contract worker, what are your top 3 concerns?

```
cols = ['_indw.concern.findwork','_indw.concern.getwork', '_indw.concern.income',
 '_indw.concern.fairpay', '_indw.concern.latepay',
 '_indw.concern.retire', '_indw.concern.savemoney',
 '_indw.concern.benefit', '_indw.concern.taxrate',
 '_indw.concern.debt', '_indw.concern.taxlegal',
 '_indw.concern.career', '_indw.concern.discrimin',
 '_indw.concern.other',
 ]

concern_df = df[cols]
# '_indw.type_6' '_indw.searchtime_6'
concern_df.describe()
```

- On average, how long would it take you to find freelance, gig, or independent contract work, between looking for work and actually starting it?

```
#shohan code
search_time = df['_indw.searchtime']
search_time.value_counts()
```

Gray code below

```
#Read in data from spreadsheets

df = pd.read_csv('/content/Copy of Samaschool_1 Month Data_T4A - 1Month.csv')
df2 = pd.read_csv('/content/Copy of Samaschool_6 Month Data_T4A - 6Month.csv')
df3 = pd.read_csv('/content/Copy of Samaschool_9 Month Data_T4A - 9Month.csv')

df = df.rename(columns=df.iloc[1])
df2 = df2.rename(columns=df2.iloc[1])
df3 = df3.rename(columns=df3.iloc[1])

df.drop(df.index[:3], inplace=True)
df2.drop(df2.index[:3], inplace=True)
df3.drop(df3.index[:3], inplace=True)


print('data 1 :', df.shape)
print('data 2 :', df2.shape)
print('data 3 :', df3.shape)


#drop the columns where all elements are NaN:

df = df.dropna(axis=1, how='all')
df2 = df2.dropna(axis=1, how='all')
df3 = df3.dropna(axis=1, how='all')


print(df.shape)
print(df2.shape)
print(df3.shape)


#clean column names
df.columns = df.columns.str.replace('\d+', '')
df.columns = df.columns.str.rstrip('.')

df2.columns = df2.columns.str.replace('\d+', '')
df2.columns = df2.columns.str.rstrip('.')

df3.columns = df3.columns.str.replace('\d+', '')
df3.columns = df3.columns.str.rstrip('.')
```

Merge dataframe

```
#merge 3 df into one dataframe
merged = pd.merge(df, df2, how='outer')
merged = merged.merge(df3, how='outer')
```

```
merged = merged.merge(df3, how='outer')

merged.shape


#rename unnamed column to City
merged.rename(columns={merged.columns[50]: 'City'}, inplace=True)
```

Track income of gig work

```
#gregs code
# note from shohan : need to clean these data and convert dtypes
continuous = ['_emp.annsalary', '_emp_hourly', '_emp.earnings','_emp_hours']
emp_income =  merged_df[continuous]

emp_income['_emp.earnings'] = emp_income['_emp.earnings'].astype('category')
emp_income['_emp_hours'] = emp_income['_emp_hours'].astype('category')

#clean data
emp_income['_emp.annsalary'] = emp_income['_emp.annsalary'].fillna('0')
emp_income['_emp.annsalary'] = emp_income['_emp.annsalary'].str.replace(r'\D', '')
emp_income['_emp.annsalary'] = emp_income['_emp.annsalary'].str.strip()
emp_income['_emp.annsalary'] = emp_income['_emp.annsalary'].astype(float)

emp_income['_emp_hourly'] = emp_income['_emp_hourly'].fillna('0')
emp_income['_emp_hourly'].replace(regex=True, inplace=True, to_replace=r'^[0-9][0.0-9.0][\-]+
emp_income['_emp_hourly'] = emp_income['_emp_hourly'].str.strip()
emp_income['_emp_hourly'] = emp_income['_emp_hourly'].astype(float)

#emp_income['_emp_hourly'].unique()
#emp_income.head()
#disrtibution of _emp_hourly
sns.distplot(emp_income['_emp_hourly'], bins=10)
plt.show()

#print(emp_income['_emp.earnings'].unique())

emp_income.groupby(['_emp.earnings', '_emp_hours']).mean()



# #Dataframe with matching respondent ids
# common_respondents = df.merge(df2, how='left', on='_respondentid')
# common_respondents = common_respondents.merge(df3, how='left', on='_respondentid')


# common_respondents.shape


# common_respondents.head()
```

Code

```
##groupby attempts

#shohan code #(greg edit key)

# if the median income of uber driver is $100,
# i also need to know how many people that is collected from. If its the median of 5 drivers,

tmpdf = pd.crosstab(index=[income_df.work_hours], columns=[income_df.service_driving], values
#pd.crosstab(index=df_tips['day'], columns=df_tips['sex'], values=df_tips['total_bill'], coln


cols = ['service_care', 'service_clean', 'service_deliver', 'service_maintanance',
        'service_personal', 'service_it', 'service_driving', 'service_professional', 'service

income_median = income_df.groupby(['service_care'])[['income', 'hourly_income']].median()
#income_agg = income_df.groupby(['service_care'])[['income']].agg([pd.sum(), pd.median(), pd.

#x = income_df.groupby('service_driving').agg([np.sum, np.median, np.std])
#income_median['N'] = income_df['service_care'].value_counts()
#print(income_median)

#for col in cols:
#  income_median = income_df.groupby([col])[['work_hours']].median()
#  income_median['N'] = income_df[col].value_counts()
#  print(income_median)

tmpdf


income_df['work_hours']


#income_df.head()




df.columns




df.head()
```

```
df.head()


#df.groupby['_service.secondary.clean_6']
df.groupby('_service.secondary.clean_6').aggregate(np.median)


services = ['_service.secondary.care_6', '_service.secondary.clean_6',
       '_service.secondary.deliver_6', '_service.secondary.maint_6',
       '_service.secondary.personal_6', '_service.secondary.it_6',
       '_service.secondary.drive_6', '_service.secondary.prof_6',
       '_service.secondary.create_6', '_service.secondary.other_6',
       '_emp.service.clean_6', '_emp.service.security_6', '_emp.service.it_6',
       '_emp.service.warehouse_6', '_emp.service.other_6',
       '_indw.service.care_6', '_indw.service.clean_6',
       '_indw.service.deliver_6', '_indw.service.maint_6',
       '_indw.service.personal_6', '_indw.service.it_6',
       '_indw.service.drive_6', '_indw.service.prof_6',
       '_indw.service.create_6', '_indw.service.other_6',
       '_indw.train.custservice_6', '_rent.service.house_6',
       '_rent.service.car_6', '_rent.service.other_6']


##add merged columns

df = df_merge
df['tmp'] = df.index
#df.columns[df.columns.str.contains('service', na=False)]

#df['_indw.service.drive_9'].value_counts(dropna=False)

#for column in services:
#    df[column] = df[column].str.replace('X', column)
reshaped_df = pd.melt(df, value_vars=services)
def group_func(series):
        for val, idx in zip(series, series.index.values):
              values += [str(idx)]
        return " ".join(values)

df.loc[:,'Group'] = df.loc[:,services].apply(group_func, axis=1)


df['categorical'] = df[df.loc[:,services]].apply(lambda x: ','.join(x.dropna().astype(int).as



df['new'] = np.nan

value = []

for column in df.loc[:,services]:
   # Select column contents by column name using [] operator
   columnSeriesObj = df[column]
   #print('Colunm Name : ', column)
```

```
    #print('Column Contents : ', columnSeriesObj.values)
    if not columnSeriesObj.isnull().any():
      value.append(columnSeriesObj.values)


value


df.Group_head(25)
```

https://stackoverflow.com/questions/54448276/pandas-reshaping-data-from-multiple-columns-into-a-single-column