In [1]:

```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sqlalchemy
from sqlalchemy import create_engine
from scipy import stats

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```python
# Create connection to our database.
engine = create_engine("mysql://root:password@localhost/taskrabbit_db")
```

In [3]:

```python
# Read in all of the data.
data = pd.read_sql_query("SELECT * FROM taskrabbit", engine)
```

In [4]:

```python
# Preview the table.
data.head()
```

Out[4]:

	recommendation_id	created_at	tasker_id	position	hourly_rate	num_completed_tasks	hired	category
0	0-0-70cf97d7-37af-4834-901c-ce3ad4893b8c	2017-09-01 00:32:25	1009185352	1	38	151	0	Furniture Assembly
1	0-0-70cf97d7-37af-4834-901c-ce3ad4893b8c	2017-09-01 00:32:25	1006892359	2	40	193	0	Furniture Assembly
2	0-0-70cf97d7-37af-4834-901c-ce3ad4893b8c	2017-09-01 00:32:25	1012023956	3	28	0	0	Furniture Assembly
3	0-0-70cf97d7-37af-4834-901c-ce3ad4893b8c	2017-09-01 00:32:25	1009733517	4	43	303	0	Furniture Assembly
4	0-0-70cf97d7-37af-4834-901c-ce3ad4893b8c	2017-09-01 00:32:25	1013579273	5	29	39	0	Furniture Assembly

Question 1: How many recommendation sets are in this data sample?

Question 2: Each recommendation set shows from 1 to 15 Taskers, what is:

- average number of Taskers shown

- median number of Taskers shown

In [5]:

```python
# Get unique counts of recommendation ids and pass it through the describe function to get quick info on recommendations.
# Logic is that the however many times the recommendation id shows up, it corresponds to a tasker's position, which will
# tell us how many taskers are shown per recommendation.

pd.DataFrame(
    data["recommendation_id"]\
    .value_counts()\
    .describe()
)
```

Out[5]:

	recommendation_id
count	2100.000000
mean	14.285714
std	2.553531
min	1.000000
25%	15.000000
50%	15.000000
75%	15.000000
max	15.000000

For questions 1 and 2, the answers lie within the above dataframe.

1) There are 2100 recommendation sets.

2a) The average number of taskers per recommendation is 14.23 (rounded up to 2 decimal places).

2b) The median number of taskers is 15.

Question 3: How many total unique Taskers are there in this data sample?

In [6]:

```python
# Taking value_counts of each tasker id will give us the counts of each unique tasker.
# Similar to COUNT(distinct tasker_id).
# Calling on 'len' will give us the length of the column, which is how many unique taskers we have in our sample.

len(data["tasker_id"].value_counts())
```

Out[6]: 830

Question 4: Which tasker has been shown the most? Which tasker has been shown the least?

In [7]:

```python
# Set up query
query = '''
SELECT tasker_id,
count(tasker_id) as 'times_shown'
FROM taskrabbit
GROUP BY tasker_id
ORDER BY count(tasker_id) DESC
'''

# Read the query as a dataframe.
taskers = pd.read_sql_query(query, engine)

# Take a look at the first 5 entries in the dataframe.
taskers.head()
```

Out[7]:

	tasker_id	times_shown
0	1014508755	608
1	1012043028	438
2	1014675294	387
3	1014629676	311
4	1007283421	290

Tasker shown the most:

In [8]:

```python
# Print the first row in the above data frame.
print(taskers.iloc[0])
```

```
tasker_id      1014508755
times_shown           608
Name: 0, dtype: int64
```

Taskers shown the least:

There are no taskers that have been shown 0 times, therefore, the taskers that have been shown the least are the taskers who have been shown at least once.

In [9]:

```python
# Grabbing all taskers who have been shown once into a new dataframe.
taskers_min = taskers[(taskers["times_shown"] == 1)]

# Set a variable equal to taskers_min["tasker_id"] to grab the column and convert it to list to get the IDs only.
least_taskers = list(taskers_min["tasker_id"])

print(f"There are {len(least_taskers)} taskers who have been shown the least. The tasker IDs are shown below.\n")
print(least_taskers)
```

```
There are 68 taskers who have been shown the least. The tasker IDs are shown below.

[1013362004, 1010779242, 1012364558, 1010640007, 1014547884, 1011968845, 1009772528, 1012386513, 1014439502, 1011952623, 100
9603880, 1010681878, 1013830691, 1009641175, 1007472083, 1011957940, 1012678504, 1008604368, 1010042971, 1006899551, 1011901
532, 1014086818, 1013573125, 1009547227, 1011985968, 1009712638, 1009871933, 1013934937, 1006853970, 1009702351, 1009994950,
1010021990, 1012151299, 1010009736, 1013854788, 1012289475, 1013656032, 1012071620, 1007480912, 1008870833, 1014926743, 1012
805440, 1008368716, 1014593279, 1007295623, 1008033678, 1012348656, 1009461190, 1008469117, 1011972750, 1007923586, 10072461
22, 1011949117, 1014478773, 1009618500, 1008474216, 1012166729, 1007383273, 1009612428, 1013573988, 1009754999, 1007638825,
1009112003, 1006690425, 1008828652, 1014310300, 1008919567, 1013731883]
```

Question 5: Which tasker has been hired the most? Which tasker has been hired the least?

In [10]:

```python
# Grab all of the taskers who have been hired and sort by the number of times they have been hired.
query = '''
SELECT tasker_id,
count(hired) as 'times_hired'
FROM taskrabbit
WHERE hired = 1
GROUP BY tasker_id
ORDER BY count(hired) DESC'''

# Read query as dataframe.
hired = pd.read_sql_query(query, engine)

# Preview dataframe.
hired.head()
```

Out[10]:

	tasker_id	times_hired
0	1012043028	59
1	1013131759	39
2	1013359522	37
3	1013165984	36
4	1013794735	35

Tasker hired the most:

In [11]:
```python
# Print first row data
print(hired.iloc[0])
```

```
tasker_id      1012043028
times_hired            59
Name: 0, dtype: int64
```

Taskers hired the least:

Since we filtered by those taskers who have been hired, the least amout of times a tasker can be hired is 1, therefore, the taskers that have been shown the least are the taskers who have been shown at least once.

In [12]:
```python
# Grabbing all taskers who have been hired and shown once into a new dataframe.
hired_least = hired[(hired["times_hired"] == 1)]

# Set a variable equal to a list of only the tasker_ids that have been hired the least and print it.
least_hired = list(hired_least["tasker_id"])

print(f"There are {len(least_hired)} taskers who have been hired the least. The tasker IDs are shown below.\n")
print(least_hired)
```

```
There are 79 taskers who have been hired the least. The tasker IDs are shown below.

[1011985968, 1008890855, 1008162664, 1013305557, 1012278216, 1013443125, 1008887321, 1009638159, 1009917428, 1008111352, 100
9348455, 1009230070, 1014212647, 1009192067, 1008903001, 1014629676, 1013034579, 1012666042, 1009848016, 1012170634, 1009616
142, 1010801075, 1007955495, 1008473496, 1012229493, 1009729754, 1013865107, 1009693303, 1007898815, 1014832736, 1014100703,
1008782548, 1008008634, 1010752503, 1013434046, 1008962854, 1015009096, 1006720321, 1014157578, 1007164698, 1009865854, 1009
751605, 1008030151, 1007146669, 1008037901, 1009393887, 1010578972, 1009733517, 1007480912, 1012719266, 1009606144, 10088627
13, 1013852438, 1007477780, 1009638573, 1010438496, 1008762938, 1008930827, 1009703547, 1015020347, 1008966829, 1010008242,
1013696131, 1011920226, 1010565292, 1007702812, 1014740602, 1008790779, 1009966564, 1013745838, 1013745898, 1013711863, 1009
299585, 1013553854, 1011914012, 1013677268, 1014478773, 1008724560, 1014251534]
```

Question 6: How many taskers have a conversion rate of 100%? Conversion rate is times hired over times shown.

In [13]:
```python
# Recall our taskers dataframe:
taskers.head()
```

Out[13]:

	tasker_id	times_shown
0	1014508755	608
1	1012043028	438
2	1014675294	387
3	1014629676	311
4	1007283421	290

In [14]:
```python
# Recall our hired dataframe:
hired.head()
```

Out[14]:

	tasker_id	times_hired
0	1012043028	59
1	1013131759	39
2	1013359522	37
3	1013165984	36
4	1013794735	35

In [15]:
```python
# Do a Pandas merge (aka an inner join on tasker_id) on 'hired' and 'taskers' dataframes.
conversion = taskers.merge(hired, how="inner", on= "tasker_id")

# Calculate conversion rate as new column.
conversion["conversion_rate"] = conversion["times_hired"]/conversion["times_shown"]

# Preview our merged dataframe with calculations.
conversion.head()
```

Out[15]:

	tasker_id	times_shown	times_hired	conversion_rate
0	1014508755	608	7	0.011513
1	1012043028	438	59	0.134703
2	1014675294	387	5	0.012920
3	1014629676	311	1	0.003215
4	1007283421	290	2	0.006897

In [16]:
```python
# 100% conversion rate means conversion_rate == 1.0
# Equivalent to: SELECT 'tasker_id' FROM conversion WHERE conversion_rate = 1

conversion_100 = pd.DataFrame(
    conversion[conversion["conversion_rate"] == 1]["tasker_id"]
)
```

Taskers with 100% conversion rate:

In [17]:

```
print(f"There are {len(conversion_100)} taskers with a 100% conversion rate.")
conversion_100
```

```
There are 6 taskers with a 100% conversion rate.
```

Out[17]:

	tasker_id
274	1008861741
306	1008094420
307	1012369686
309	1011985968
310	1007480912
311	1014478773

Question 7: Would it be possible for all taskers to have a conversion rate of 100%?

In practice, no it would not. When one tasker is chosen over another in a recommendation, those who were not selected will have their conversion rate permanently lowered. Taskers' conversion rates can approach 100% after not being selected, but they will never reach 100%.

For example, if a tasker has been selected 9 out of 9 times in the span of their account being active, should they be not selected in the 10th time they are shown in a recommendation, their conversion rate will drop from 100% to 90%. They can be selected 90 more times and will still have a conversion rate of 99%. Repeat to infinity and you will still never reach 100%.

This will happen to each tasker that is chosen over another tasker, so in practice, it will not be possible for all taskers have a conversion of 100%.

In theory, yes, but only if one tasker is shown per recommendation and the tasker is hired every single time.

Question 8: For each category, what is the average position of the Tasker who is hired?

In [18]:

```
#
query = '''
SELECT category,
avg(position) as 'avg_position'
FROM taskrabbit
WHERE hired = 1
GROUP BY category
'''
position = pd.read_sql_query(query, engine)

position
```

Out[18]:

	category	avg_position
0	Furniture Assembly	3.6119
1	Mounting	4.5961
2	Moving Help	4.1454

Question 9: For each category, what is the average hourly rate and average number of completed tasks for the Taskers who are hired?

In [19]:

```
query = '''
SELECT category,
avg(hourly_rate) as "avg_rate",
avg(num_completed_tasks) as "avg_tasks"
FROM taskrabbit
WHERE hired = 1
GROUP BY category
'''

category = pd.read_sql_query(query, engine)
category
```

Out[19]:

	category	avg_rate	avg_tasks
0	Furniture Assembly	38.7010	249.0210
1	Mounting	50.1548	284.0961
2	Moving Help	63.0123	273.8827

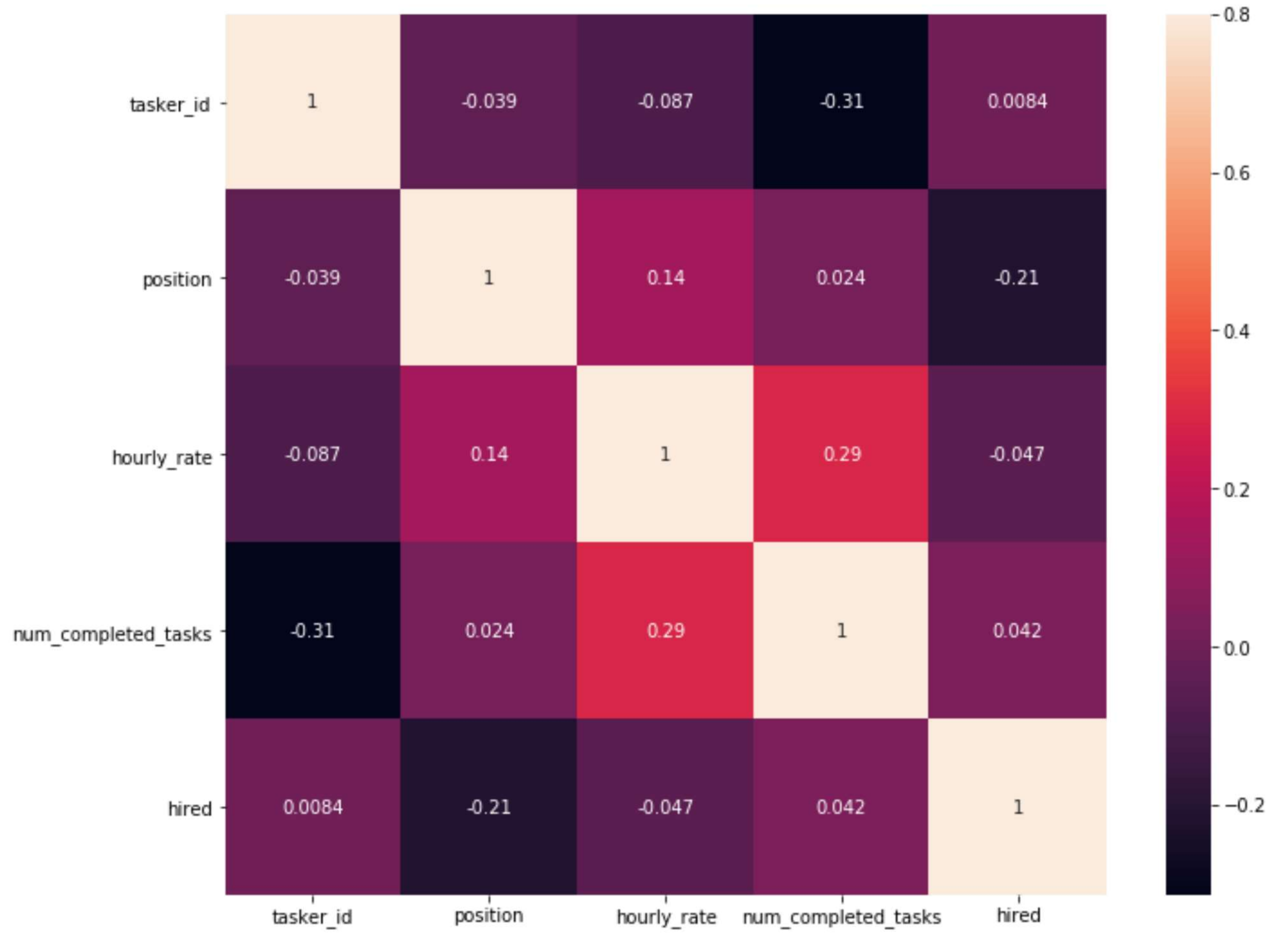
Question 10: Based on the previous, how would you suggest hourly rates to taskers to maximize opportunity to be hired?

We would want to find the distribution of hourly rates for those taskers that have been hired to visually spot any trends.

We begin by looking for any correlations between our values in the dataset:

In [20]:

```python
# Generate heatmap of related variables.
corrmat = data.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True, annot=True);
```



Looking at this heatmap, we see that the most related variables in the dataset are going to be the number of completed tasks and the hourly rate. Let's dive a little deeper in to the dataset and break the information down by category:

Defining a few functions to not repeat ourselves:

In [21]:

```python
def df_by_category(category):

    ''' Returns a dataframe from a given category in the dataset.'''

    # Query for all taskers' hourly rates who have been hired by category.
    query = f'''
    SELECT hourly_rate as '{category}_hourly_rate',
    num_completed_tasks
    FROM taskrabbit
    WHERE hired = 1 and category = '{category}'
    '''

    df = pd.read_sql_query(query, engine)

    return df
```

In [22]:

```python
def plot_by_category(df, category):

    '''Plots a dataframe by hourly rate vs tasks completed and a histogram for hourly_rate.'''

    # Set up scatterplot
    plt.grid(zorder=0)
    plt.title(f"{category}")
    plt.xlabel("Hourly Rate")
    plt.ylabel("Tasks Completed")
    plt.scatter(x=df[f"{category}_hourly_rate"],y=df["num_completed_tasks"], edgecolor="black",zorder=3)
    plt.show()

    # Set up histogram
    sns.distplot(df[f"{category}_hourly_rate"])
```

In [23]:

```python
def pearson(df, category):

    ''' Prints pearson coefficient and p-value for hourly_rate and num_completed_tasks'''

    x = df[f"{category}_hourly_rate"]
    y = df["num_completed_tasks"]

    data = stats.pearsonr(x,y)

    print("pearson coefficient: ", data[0])
    print("p-value: ", data[1])
```

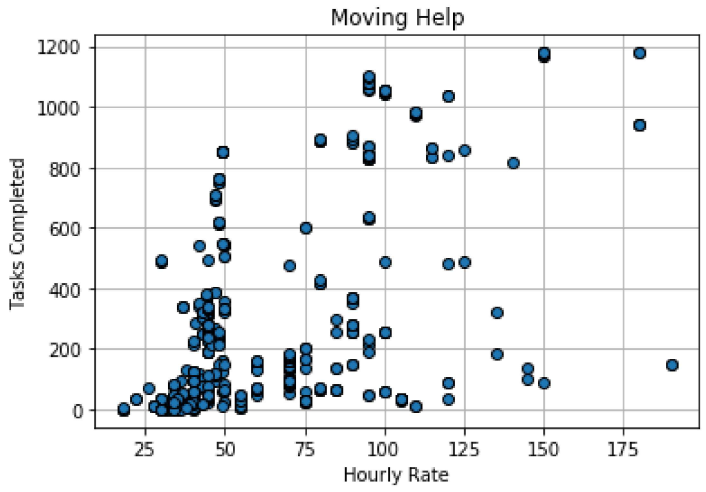
Plotting each category to see where our distrubution lies:

By looking at the taskers' hourly rates who have been hired and the number of tasks they have completed, we can get an idea of where the sweet spot is for an hourly rate that will increase their chances of being selected. Using the standard deviation, we can determine a range of hourly rates that will fall within the range of what we can consider a successful tasker.

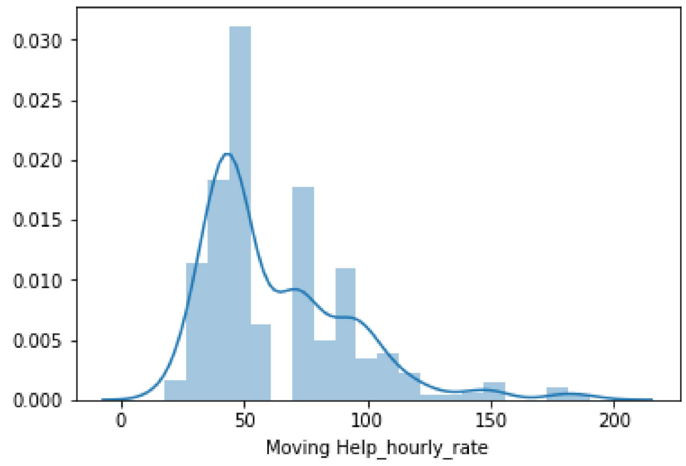
Moving Help:

In [24]:
```
moving_help = df_by_category("Moving Help")

plot_by_category(moving_help, "Moving Help")
pearson(moving_help, "Moving Help")
```




```
pearson coefficient:  0.5321601218393275
p-value:  4.549279071656759e-43
```



In [25]:
```
# Check our stats for the dataset we pulled. Round to 2 sig figs.
moving_help.describe().round(2)
```

Out[25]:

	Moving Help_hourly_rate	num_completed_tasks
count	571.00	571.00
mean	63.01	273.88
std	29.96	316.91
min	18.00	0.00
25%	43.00	36.00
50%	49.00	147.00
75%	80.00	334.50
max	190.00	1178.00

Looking at our scatterplot, we can visually see that the sweet spot in the graph for an hourly rate is around $50. This is across the board for the least experienced (0 tasks completed) to the more experienced (500+ tasks completed).

This is further confirmed in the histogram that accompanies this analysis. The frequency at which a $50 hourly rate occurs is very apparent, so we can choose this hourly rate as a good starting point for suggesting a rate for a tasker.

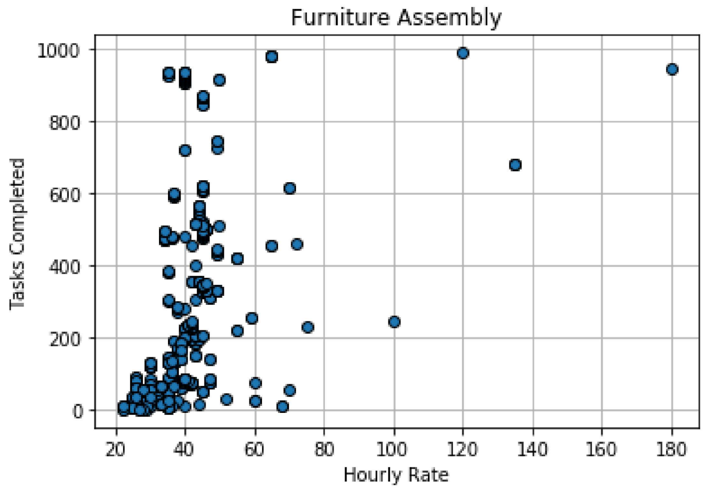
Checking out the distribution for the Moving Help category, we see that the median hourly rate is 49.00 with the mean being 63.34. Now as the number of tasks completed goes up, there is a weak positive correlation with the hourly rate as well. This tells us that as a tasker is more experienced, the tasker can command a higher hourly rate.

Furniture Assembly:

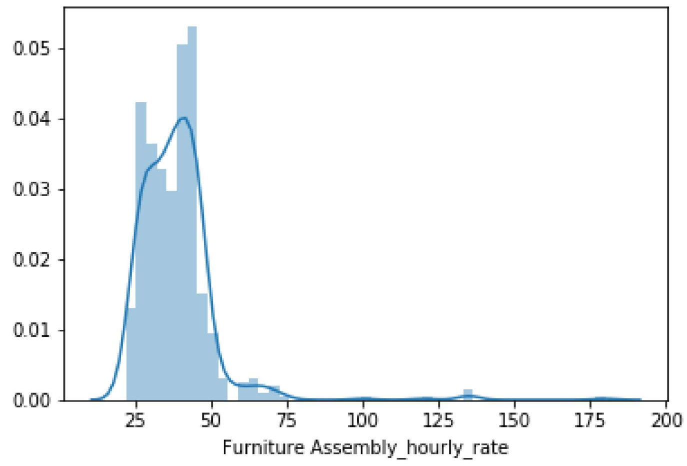
In [26]:

```
furn_assemb = df_by_category("Furniture Assembly")

plot_by_category(furn_assemb, "Furniture Assembly")
pearson(furn_assemb, "Furniture Assembly")
```




```
pearson coefficient:  0.4848399651267565
p-value:  4.6557924140171275e-35
```



In [27]:

```
furn_assemb.describe().round(2)
```

Out[27]:

	Furniture Assembly_hourly_rate	num_completed_tasks
count	572.00	572.00
mean	38.70	249.02
std	13.51	266.54
min	22.00	0.00
25%	30.00	48.50
50%	38.00	131.50
75%	44.00	420.50
max	180.00	988.00

In this distribution we can visually infer that the optimal rate for those who have been hired lies around 40.00. Looking at the median and the mean, we can confirm this as well.

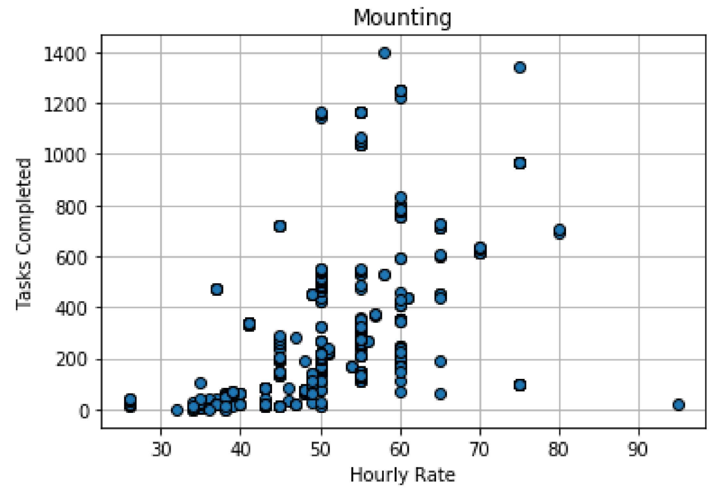
The suggested hourly rate for a tasker for Furniture Assembly should hover around 40.00 with the sliding scale being within one standard deviation to maximize their chances.

Mounting:

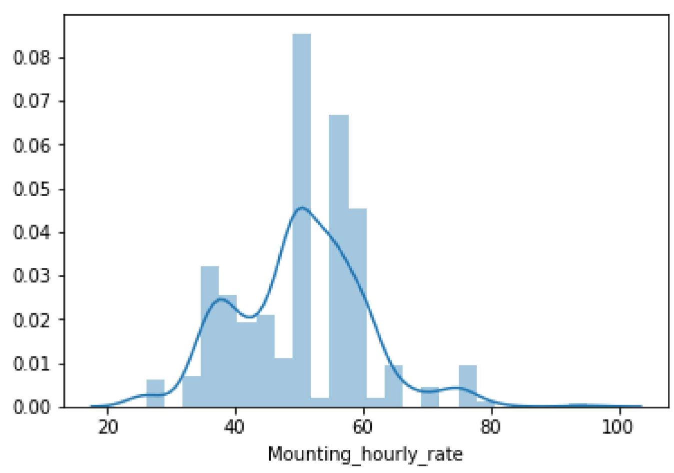
In [28]:

```python
mounting = df_by_category("Mounting")

plot_by_category(mounting, "Mounting")
pearson(mounting, "Mounting")
```




```
pearson coefficient:  0.5346245533600721
p-value:  7.29900532575875e-43
```



In [29]:

```python
mounting.describe().round(2)
```

Out[29]:

	Mounting_hourly_rate	num_completed_tasks
count	562.00	562.00
mean	50.15	284.10
std	10.04	290.96
min	26.00	0.00
25%	43.00	70.50
50%	50.00	190.00
75%	55.00	410.00
max	95.00	1397.00

This distribution is much different than the last two. We see a bimodal distribution in hourly_rate frequencies in our histogram and based off of the scatter plot, it is dependent on the number of tasks a tasker has completed. We do see a stronger positive correlation than the previous two categories and one can conclude that for this category, the best rate to suggest would be one that is dependent on how many tasks a tasker has completed.

For taskers that have completed less than 100 tasks, it would make sense to suggest an hourly rate of slightly less than 40.00, and those who have a bit more experience under their belt could have suggestions starting at 50.00 - selecting the medians for each distribution essentially and have a sliding scale based off of the standard deviation.

Summary

Based off of this analysis, we can conclude that in order to boost a tasker's chances of being hired, it is dependent on:

- The category in which the work falls in
- The amount of experience a tasker has (num_tasks_completed)
- The tasker's position in each recommendation set

Looking at the distribution by category, sliding scales based off 1 standard deviation from the median can be implemented to suggest a suitable range of hourly rates.