final-project-4-2

April 18, 2021

[94]:
```
capital_dict ={
    'Alabama': 'Montgomery',
    'Alaska': 'Juneau',
    'Arizona':'Phoenix',
    'Arkansas':'Little Rock',
    'California': 'Sacramento',
    'Colorado':'Denver',
    'Connecticut':'Hartford',
    'Delaware':'Dover',
    'Florida': 'Tallahassee',
    'Georgia': 'Atlanta',
    'Hawaii': 'Honolulu',
    'Idaho': 'Boise',
    'Illinois': 'Springfield',
    'Indiana': 'Indianapolis',
    'Iowa': 'Des Moines',
    'Kansas': 'Topeka',
    'Kentucky': 'Frankfort',
    'Louisiana': 'Baton Rouge',
    'Maine': 'Augusta',
    'Maryland': 'Annapolis',
    'Massachusetts': 'Boston',
    'Michigan': 'Lansing',
    'Minnesota': 'St. Paul',
    'Mississippi': 'Jackson',
    'Missouri': 'Jefferson City',
    'Montana': 'Helena',
    'Nebraska': 'Lincoln',
    'Neveda': 'Carson City',
    'New Hampshire': 'Concord',
    'New Jersey': 'Trenton',
    'New Mexico': 'Santa Fe',
    'New York': 'Albany',
    'North Carolina': 'Raleigh',
    'North Dakota': 'Bismarck',
    'Ohio': 'Columbus',
```

```
    'Oklahoma': 'Oklahoma City',
    'Oregon': 'Salem',
    'Pennsylvania': 'Harrisburg',
    'Rhoda Island': 'Providence',
    'South Carolina': 'Columbia',
    'South Dakoda': 'Pierre',
    'Tennessee': 'Nashville',
    'Texas': 'Austin',
    'Utah': 'Salt Lake City',
    'Vermont': 'Montpelier',
    'Virginia': 'Richmond',
    'Washington': 'Olympia',
    'West Virginia': 'Charleston',
    'Wisconsin': 'Madison',
    'Wyoming': 'Cheyenne'
}
```

[95]:
```
us_state_abbrev = {
    'Alabama': 'AL',
    'Alaska': 'AK',
    'American Samoa': 'AS',
    'Arizona': 'AZ',
    'Arkansas': 'AR',
    'California': 'CA',
    'Colorado': 'CO',
    'Connecticut': 'CT',
    'Delaware': 'DE',
    'District of Columbia': 'DC',
    'Florida': 'FL',
    'Georgia': 'GA',
    'Guam': 'GU',
    'Hawaii': 'HI',
    'Idaho': 'ID',
    'Illinois': 'IL',
    'Indiana': 'IN',
    'Iowa': 'IA',
    'Kansas': 'KS',
    'Kentucky': 'KY',
    'Louisiana': 'LA',
    'Maine': 'ME',
    'Maryland': 'MD',
    'Massachusetts': 'MA',
    'Michigan': 'MI',
    'Minnesota': 'MN',
    'Mississippi': 'MS',
    'Missouri': 'MO',
    'Montana': 'MT',
```

```
    'Nebraska': 'NE',
    'Nevada': 'NV',
    'New Hampshire': 'NH',
    'New Jersey': 'NJ',
    'New Mexico': 'NM',
    'New York': 'NY',
    'North Carolina': 'NC',
    'North Dakota': 'ND',
    'Northern Mariana Islands':'MP',
    'Ohio': 'OH',
    'Oklahoma': 'OK',
    'Oregon': 'OR',
    'Pennsylvania': 'PA',
    'Puerto Rico': 'PR',
    'Rhode Island': 'RI',
    'South Carolina': 'SC',
    'South Dakota': 'SD',
    'Tennessee': 'TN',
    'Texas': 'TX',
    'Utah': 'UT',
    'Vermont': 'VT',
    'Virgin Islands': 'VI',
    'Virginia': 'VA',
    'Washington': 'WA',
    'West Virginia': 'WV',
    'Wisconsin': 'WI',
    'Wyoming': 'WY'
}
```

[96]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
import re
import time
import sqlalchemy
import seaborn as sb
import matplotlib
from datetime import datetime
import sqlalchemy
from time import mktime
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

0.1 Data Processing

[97]:
```
# Set ability to scroll through 500 rows & columns
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
# Set matplotlib default settings
matplotlib.rc_file_defaults()


# API_KEY for Open Weather Map
API_KEY = '1a13247695e35ba6373838d8f1384ea5'

# Function to convert to Fahrenheit
def convert_to_fahrenheit(K):
    return (K-273.15) * 9/5 + 32


# Function to convert seconds to hours
def convert_to_hours(seconds):
    hour = seconds // 3600
    return float(hour)


# Function to call Open Weather Map API
def get_weather_data():
    temp_data = []
    for state in capital_dict:
        response = requests.get("https://api.openweathermap.org/data/2.5/
 ↪weather?q={}&appid={}".format(capital_dict[state], API_KEY))
        json = response.json()
        try:
            temp = convert_to_celsius(json['main']['temp'])
            desc = json['weather'][0]['description']
            wind = json['wind']['speed']
            humidity = json['main']['humidity']
            epoch_sunrise = int(json['sys']['sunrise'])
            epoch_sunset = int(json['sys']['sunset'])

            hours_of_sunlight = convert(epoch_sunset-epoch_sunrise)
            data = (temp, desc, wind, humidity, hours_of_sunlight, state,␣
 ↪capital_dict[state], us_state_abbrev[state])
            temp_data.append(data)
        except:
            continue
```

```
    temp_df = pd.DataFrame(temp_data, columns = ['temp', 'description',␣
 ↪'wind_speed', 'humidity', 'hours_of_sunlight', 'state', 'city',␣
 ↪'state_code'])
    return temp_df

# Function to read in mental health data and process
def read_mental_health_data():
    # Read in Mental Health Data from locally downloaded CSV file
    df = pd.read_csv('survey.csv')
    # Normalizing how data represents Male, Female (Ex: M to Male, F to Female)
    df["Gender"] = df['Gender'].apply(lambda s: re.sub("(^F)([A-Za-z]+)*",␣
 ↪"Female", s.strip().title()))
    df["Gender"] = df['Gender'].apply(lambda s: re.sub("(^M)([A-Za-z]+)*",␣
 ↪"Male", s.strip().title()))
    df["Gender"] = df['Gender'].apply(lambda s: "Other" if s != 'Male' and s !=␣
 ↪'Female' else s)
    # Filter out ages that don't make sense
    df = df[df["Age"] < 80]
    df = df[df["Age"] > 0]
    # Filtering out rows with null states
    df_state = df[~df["state"].isna()]
    return df_state

# Function to merge (inner join) dataframes
def join_dataframes(temp_df, df_state):
    # Merge the dataframes using an inner join
    df_joined = temp_df.merge(df_state, how = 'inner', left_on = 'state_code',␣
 ↪right_on = 'state')
    return df_joined

# Create the joined dataframe using the functions defined above
temp_df = get_weather_data()
df_state = read_mental_health_data()
df_joined = join_dataframes(temp_df, df_state)
```

[98]:
```
%reload_ext sql
%sql postgres://jovyan:si330studentuser@localhost:5432/si330
%sql drop table if exists "temp_mental_health"
engine = sqlalchemy.create_engine('postgres://jovyan:si330studentuser@localhost:
 ↪5432/si330')
# Push dataframe to SQL database
df_joined.to_sql("temp_mental_health",engine)
```

```
 * postgres://jovyan:***@localhost:5432/si330
Done.
```

1 Relationships to Analyze:

1.0.1 Weather vs. Mental Health Data

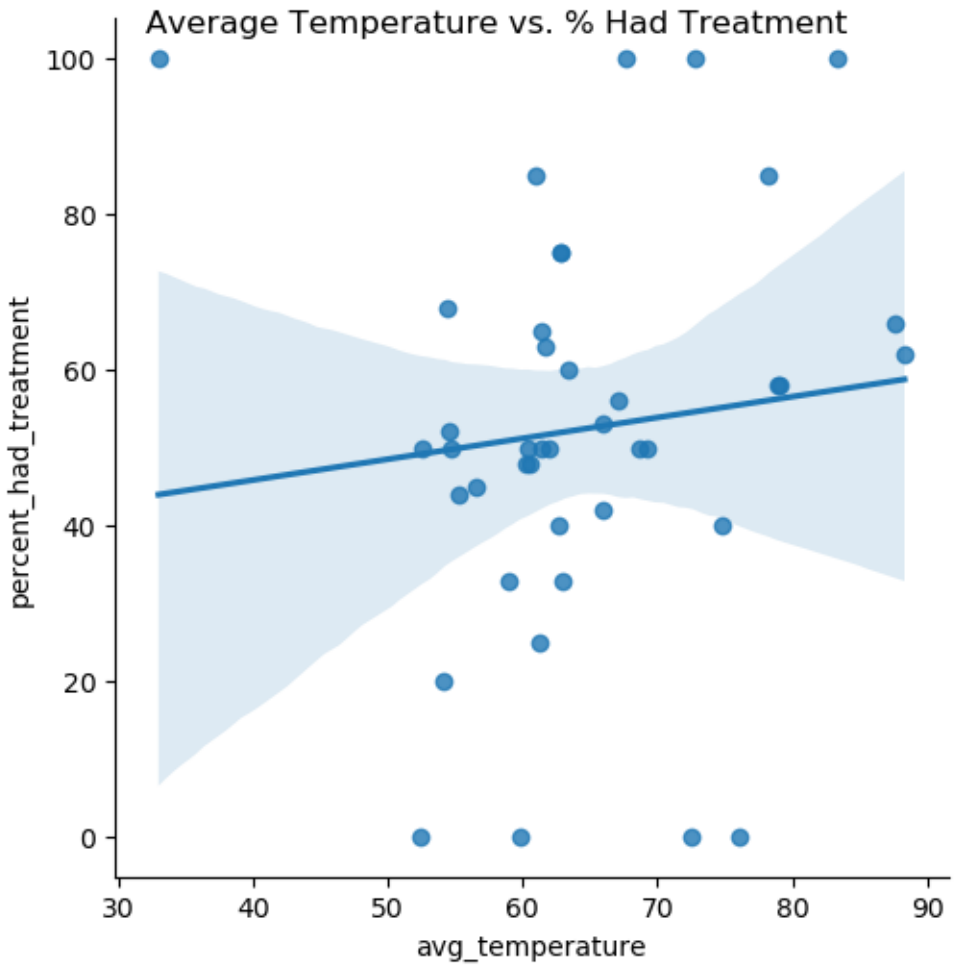
- temp vs. treatment
- hours_of_sunlight vs. treatment
- wind vs. treatment
- humidity vs. treatment
- state bar plot vs. treatment ### Mental Health Data vs. Mental Health Data
- age vs. treatment
- remote_work vs. treatment
- gender vs. treatment
- family_history vs. treatment

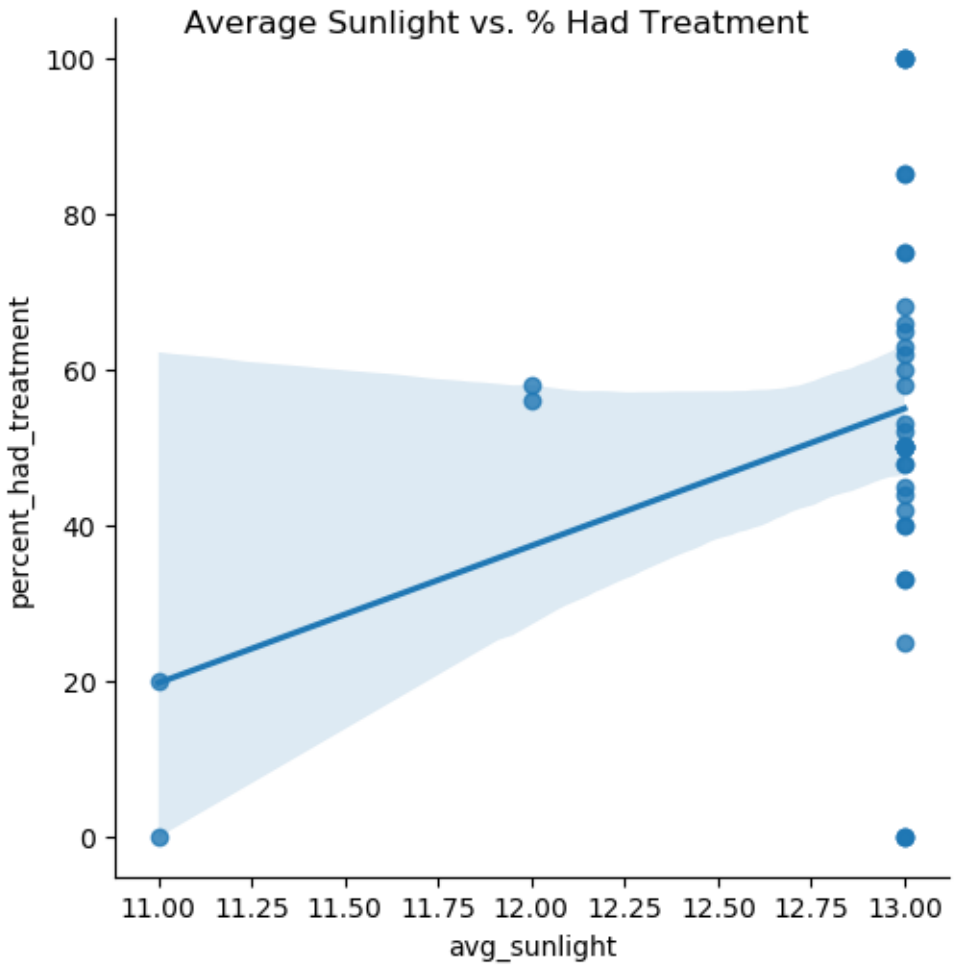
1.1 Querying and Plotting

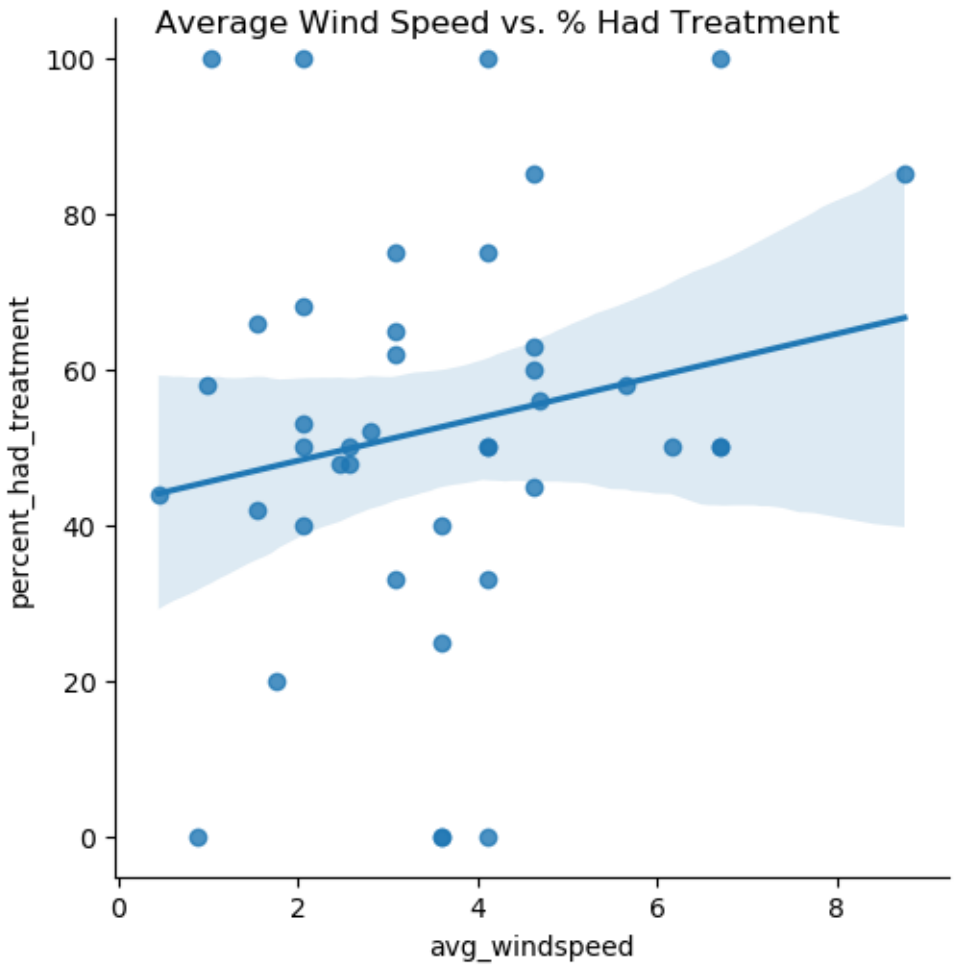
[99]:
```
# temperature vs treatment
temperature_vs_treatment_query = "SELECT avg(temp) as avg_temperature, SUM(case␣
 ↪when treatment = 'Yes' then 1 else 0 end)*100 / (count(state_x)) as␣
 ↪percent_had_treatment from temp_mental_health group by state_x"
temperature_vs_treatment = pd.read_sql_query(temperature_vs_treatment_query,␣
 ↪engine)
# sunlight vs treatment
sunlight_vs_treatment_query = "SELECT avg(hours_of_sunlight) as avg_sunlight,␣
 ↪SUM(case when treatment = 'Yes' then 1 else 0 end)*100 / (count(state_x)) as␣
 ↪percent_had_treatment from temp_mental_health group by state_x"
sunlight_vs_treatment = pd.read_sql_query(sunlight_vs_treatment_query, engine)
# wind speed vs treatment
windspeed_vs_treatment_query = "SELECT avg(wind_speed) as avg_windspeed,␣
 ↪SUM(case when treatment = 'Yes' then 1 else 0 end)*100 / (count(state_x)) as␣
 ↪percent_had_treatment from temp_mental_health group by state_x"
windspeed_vs_treatment = pd.read_sql_query(windspeed_vs_treatment_query, engine)
#humidity vs treatment
humidity_vs_treatment_query = "SELECT avg(humidity) as avg_humidity, SUM(case␣
 ↪when treatment = 'Yes' then 1 else 0 end)*100 / (count(state_x)) as␣
 ↪percent_had_treatment from temp_mental_health group by state_x"
humidity_vs_treatment = pd.read_sql_query(humidity_vs_treatment_query, engine)
all_query = "SELECT * from temp_mental_health"
all_query_df = pd.read_sql_query(all_query, engine)
grouped_state_query = "SELECT state_code as state, SUM(case when treatment =␣
 ↪'Yes' then 1 else 0 end) as num_had_treatment from temp_mental_health group␣
 ↪by state_code"
grouped_state_df = pd.read_sql_query(grouped_state_query, engine)
```

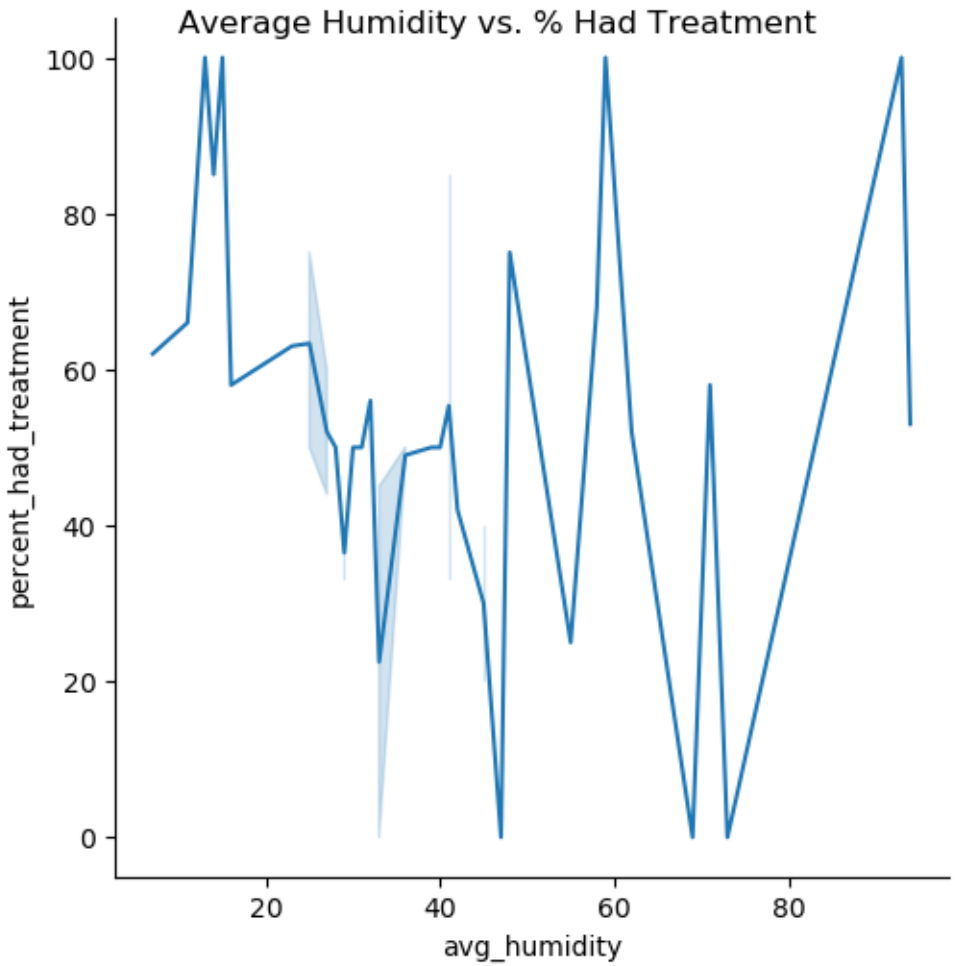
[100]:
```
#temperature vs treatment plot
temperature_vs_treatment_plot = sb.lmplot(x = "avg_temperature", y =␣
 ↪"percent_had_treatment", data = temperature_vs_treatment)
```

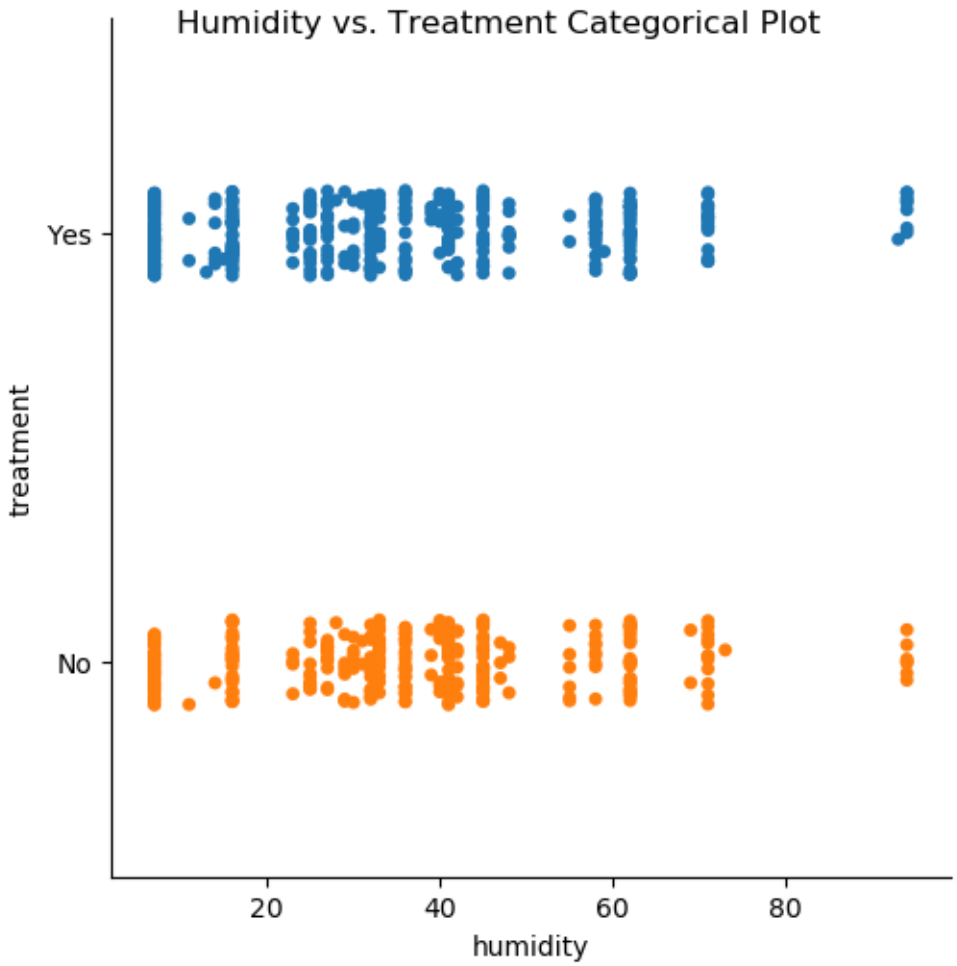
```
temperature_vs_treatment_plot.fig.suptitle("Average Temperature vs. % Had␣
 ↪Treatment")
temperature_vs_treatment_plot.savefig("temperature_vs_treatment.pdf")
#sunlight vs treatment plot
sunlight_vs_treatment_plot = sb.lmplot(x = "avg_sunlight", y =␣
 ↪"percent_had_treatment", data = sunlight_vs_treatment)
sunlight_vs_treatment_plot.fig.suptitle("Average Sunlight vs. % Had Treatment")
sunlight_vs_treatment_plot.savefig("sunlight_vs_treatment.pdf")
#windspeed vs treatment plot
windspeed_vs_treatment_plot = sb.lmplot(x = "avg_windspeed", y =␣
 ↪"percent_had_treatment", data = windspeed_vs_treatment)
windspeed_vs_treatment_plot.fig.suptitle("Average Wind Speed vs. % Had␣
 ↪Treatment")
windspeed_vs_treatment_plot.savefig("windspeed_vs_treatment.pdf")
#humidity vs treatment plot
humid_vs_treatment_plot = sb.relplot(x = "avg_humidity", y =␣
 ↪"percent_had_treatment", kind = "line", data = humidity_vs_treatment)
humid_vs_treatment_plot.fig.suptitle("Average Humidity vs. % Had Treatment")
humid_vs_treatment_plot.savefig("humid_vs_treatment_plot.pdf")
#humidity vs treatment categorical plot
humidity_cat_plot = sb.catplot(x = "humidity", y = "treatment", order = ["Yes",␣
 ↪"No"], data = all_query_df)
humidity_cat_plot.fig.suptitle("Humidity vs. Treatment Categorical Plot")
humidity_cat_plot.savefig("humid_cat_plot.pdf")
#age vs treatment plot
age_cat_plot = sb.catplot(x = "Age", y = "treatment", order = ["Yes", "No"],␣
 ↪kind = "box", data = all_query_df)
age_cat_plot.fig.suptitle("Age vs. Treatment Categorical Plot")
age_cat_plot.savefig("age_cat_plot.pdf")
```

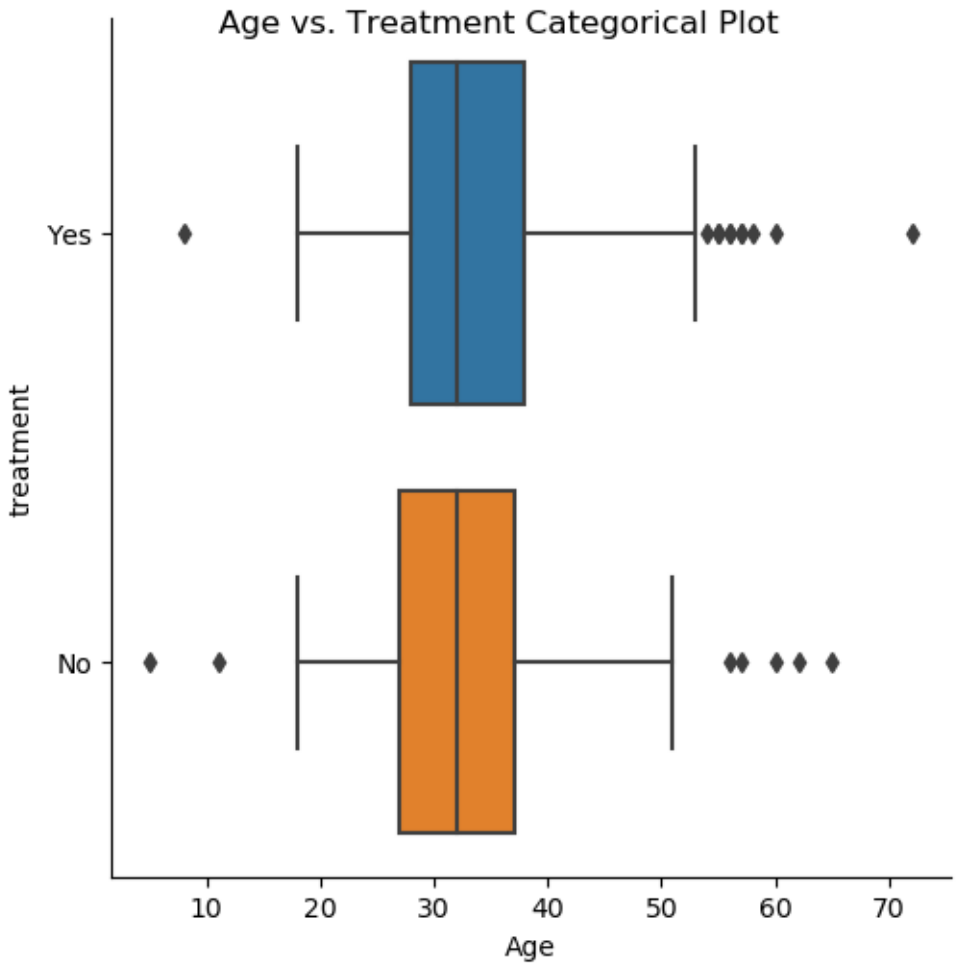

Average Temperature vs. % Had Treatment
percent_had_treatment
avg_temperature
100
80
60
40
20
0
30
40
50
60
70
80
90


Average Sunlight vs. % Had Treatment
percent_had_treatment
100
80
60
40
20
0
11.00
11.25
11.50
11.75
12.00
12.25
12.50
12.75
13.00
avg_sunlight

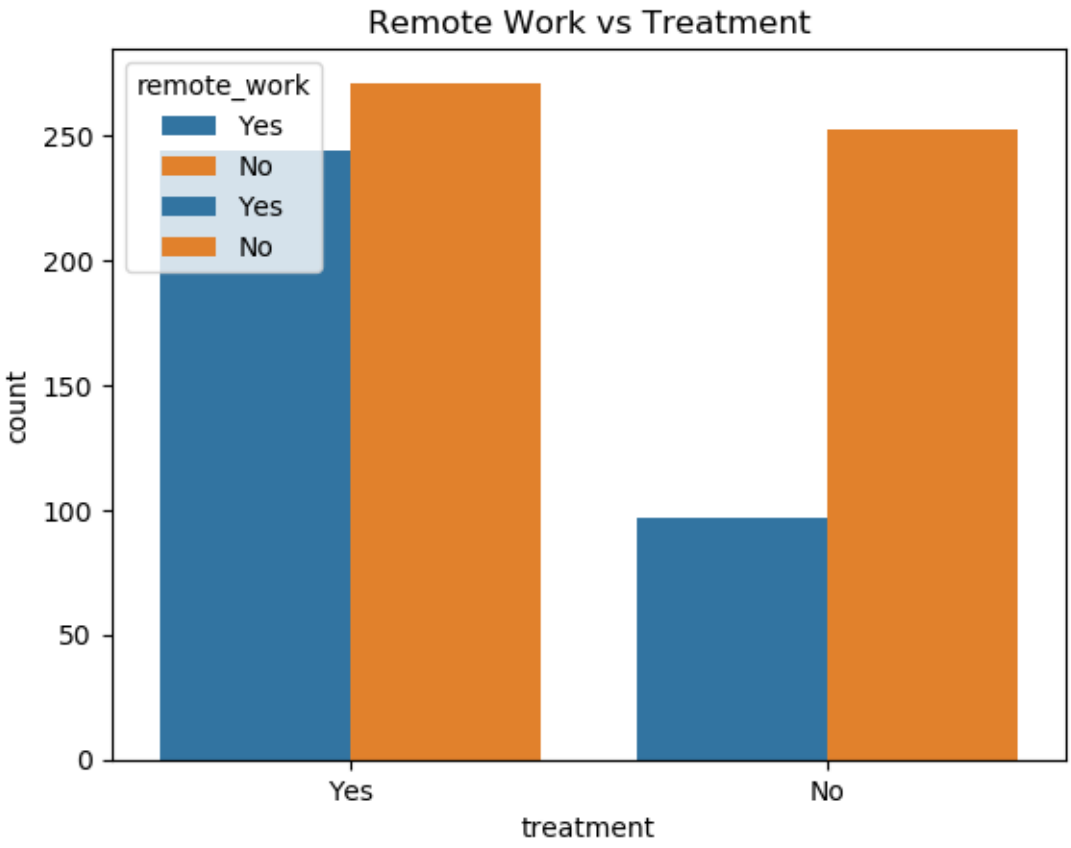

Average Wind Speed vs. % Had Treatment
percent_had_treatment
avg_windspeed
0
20
40
60
80
100
0
2
4
6
8

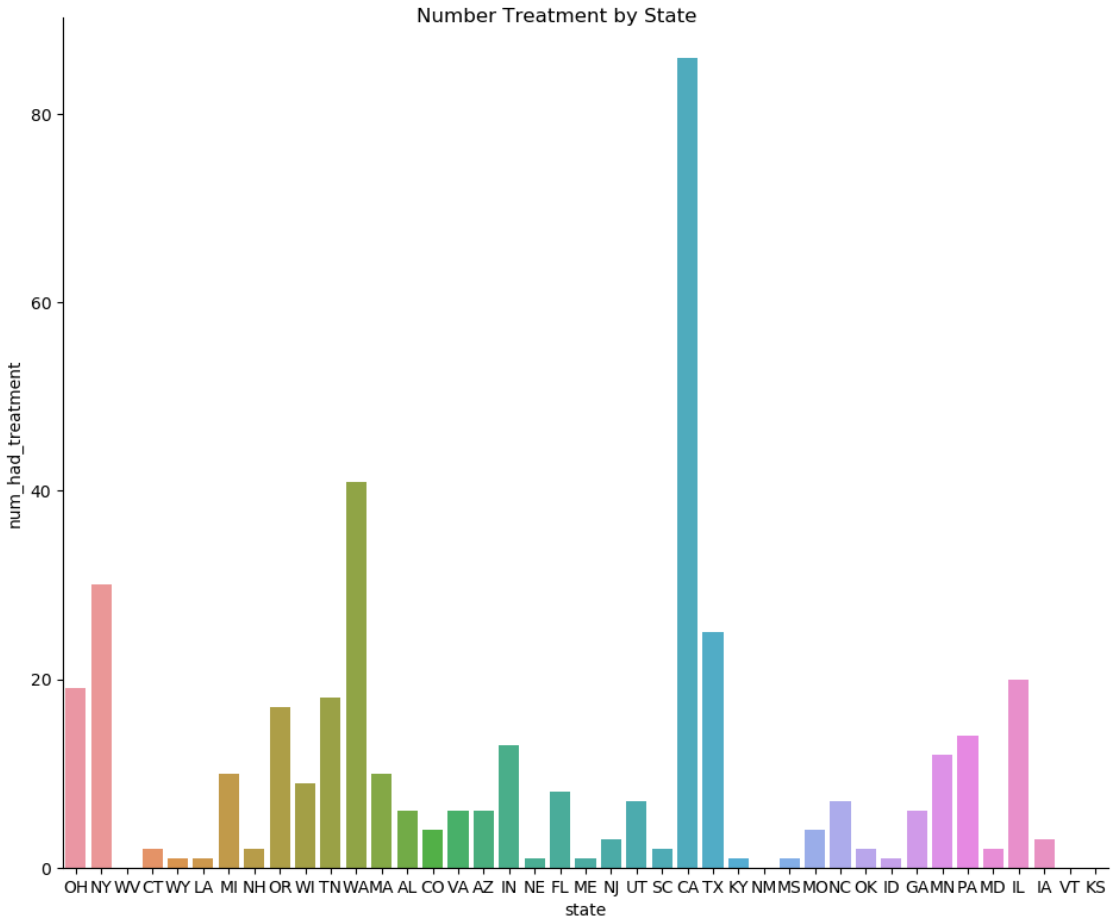

Average Humidity vs. % Had Treatment
percent_had_treatment
avg_humidity
0
20
40
60
80
100
20
40
60
80


Humidity vs. Treatment Categorical Plot
Yes
No
treatment
20
40
60
80
humidity

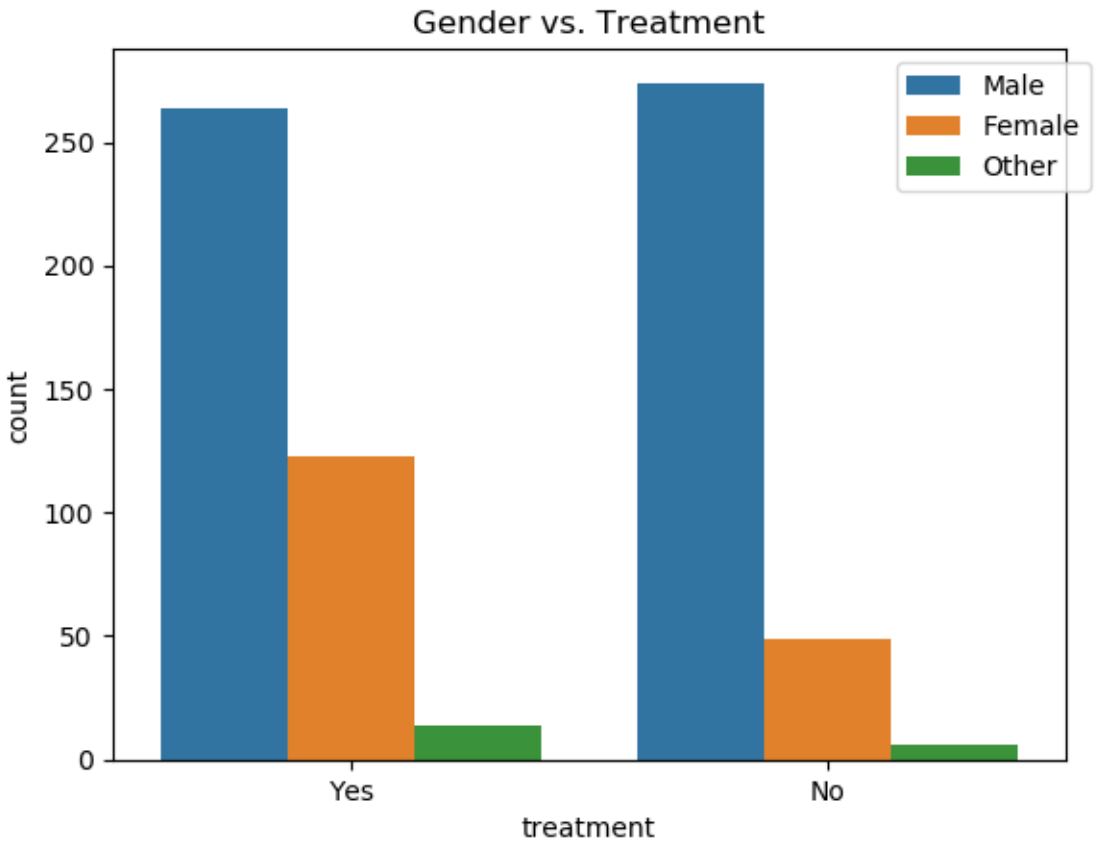


[101]:
```
# family history vs treatment plot
family_history_plot = sb.countplot(x = "treatment", hue = "family_history",␣
 ↪data = all_query_df).set_title("Family History vs Treatment")
family_history_plot.figure.savefig('family_history.pdf')
# remote work vs treatment plot
remote_work_plot = sb.countplot(x = "treatment", hue = "remote_work", data =␣
 ↪all_query_df).set_title("Remote Work vs Treatment")
remote_work_plot.figure.savefig('remote_work.pdf')
# treatment % by state plot
state_treatment_plot = sb.catplot(x= "state", y = "num_had_treatment", kind =␣
 ↪"bar", data = grouped_state_df)
state_treatment_plot.fig.suptitle("Number Treatment by State")
state_treatment_plot.fig.set_figwidth(10)
state_treatment_plot.fig.set_figheight(8)
```


Remote Work vs Treatment
remote_work
Yes
No
Yes
No
250
200
150
100
50
0
count
Yes
No
treatment



[102]:
```
# treatment % by gender
gender_plot = sb.countplot(x = "treatment", hue = "Gender", data = all_query_df)
gender_plot.legend(bbox_to_anchor=(1.04,1), loc="best")
gender_plot.set_title("Gender vs. Treatment")
gender_plot.figure.savefig('gender_vs_treatment.pdf')
```



1.2 Unit Testing

[103]:
```
import unittest
import numpy


class TestSum(unittest.TestCase):
    def test_convert_to_fahrenheit(self):
        self.assertEqual(int(convert_to_fahrenheit(300)), 80, "Should be 80")
        self.assertEqual(int(convert_to_fahrenheit(320)), 116, "Should be 116")
        self.assertEqual(int(convert_to_fahrenheit(340)), 152, "Should be 152")
    def test_convert_to_hours(self):
        self.assertEqual(convert_to_hours(3600), 1, "Should be 1")
        self.assertEqual(convert_to_hours(7200), 2, "Should be 2")
        self.assertEqual(convert_to_hours(7300), 2, "Should be 2")
        self.assertEqual(convert_to_hours(7400), 2, "Should be 2")
    def test_get_weather_data(self):
        df_test = get_weather_data()
        for i in range(len(df_test)):
```

```
            self.assertEqual(type(df_test.iloc[i][0]), numpy.float64, "Should␣
↪be str")
            self.assertEqual(type(df_test.iloc[i][1]), str, "Should be str")
            self.assertEqual(type(df_test.iloc[i][2]), numpy.float64, "Should␣
↪be float")
            self.assertEqual(type(df_test.iloc[i][3]), numpy.int64, "Should be␣
↪int")
            self.assertEqual(type(df_test.iloc[i][4]), numpy.float64, "Should␣
↪be float")
            self.assertEqual(type(df_test.iloc[i][5]), str, "Should be str")
            self.assertEqual(type(df_test.iloc[i][6]), str, "Should be str")
            self.assertEqual(type(df_test.iloc[i][7]), str, "Should be str")
    def test_read_mental_health_data(self):
        test_df = read_mental_health_data()
        for i in range(len(test_df)):
            self.assertEqual(type(test_df.iloc[i]['Age']), numpy.int64 or int,␣
↪"Should be int")
            self.assertEqual(type(test_df.iloc[i]['Gender']), str, "Should be␣
↪str")
            self.assertEqual(type(test_df.iloc[i]['state']), str, "Should be␣
↪str")
            self.assertEqual(type(test_df.iloc[i]['family_history']), str,␣
↪"Should be str")
            self.assertEqual(type(test_df.iloc[i]['remote_work']), str, "Should␣
↪be str")
            self.assertEqual(type(test_df.iloc[i]['treatment']), str, "Should␣
↪be str")
    def test_calc_linear_regression(self):
        self.assertEqual(calc_linear_regression([1,2,3], [4, 5, 6])[1], 1,␣
↪"Should be 1")
        self.assertEqual(calc_linear_regression([1,2,3], [4, 5, 6])[0], 3,␣
↪"Should be 3")

unittest = TestSum()
unittest.test_convert_to_fahrenheit()
unittest.test_convert_to_hours()
unittest.test_get_weather_data()
unittest.test_read_mental_health_data()
unittest.test_calc_linear_regression()
```

```
slope b1 is 1.0
intercept b0 is 3.0
squared error is 0.0
mean squared error is 0.0
root mean square error is:  0.0
R2 is:  1.0
```

```
slope b1 is 1.0
intercept b0 is 3.0
squared error is 0.0
mean squared error is 0.0
root mean square error is:  0.0
R2 is:  1.0
```

1.3 Regression Testing

[105]:
```
#creating a function to calculate linear regressions
def calc_linear_regression(x_list, y_list):
    #first find the values for the slope and intercept
    x = np.array(x_list)
    y = np.array(y_list)
    n = np.size(x)

    x_mean = np.mean(x)
    y_mean = np.mean(y)
    x_mean,y_mean

    Sxy = np.sum(x*y)- n*x_mean*y_mean
    Sxx = np.sum(x*x)-n*x_mean*x_mean

    b1 = Sxy/Sxx
    b0 = y_mean-b1*x_mean
    print('slope b1 is', b1)
    print('intercept b0 is', b0)

    #next calculate the mean squared error and R2
    y_pred = b1 * x + b0
    error = y - y_pred
    squared_error = np.sum(error**2)
    print('squared error is', squared_error)
    mean_square_error = squared_error/n
    print('mean squared error is', mean_square_error)
    root_mean_square_error = np.sqrt(mean_square_error)
    print('root mean square error is: ', root_mean_square_error)
    SSt = np.sum((y - y_mean)**2)
    R2 = 1- (squared_error/SSt)
    print('R2 is: ', R2)
    return (b0, b1, mean_square_error, root_mean_square_error, R2)
#linear regression of average temperature vs percent of people who have␣
 ↪received treatment
avg_temperature_list = temperature_vs_treatment["avg_temperature"].to_list()
percent_had_treatment_list = temperature_vs_treatment["percent_had_treatment"].
 ↪to_list()
calc_linear_regression(avg_temperature_list, percent_had_treatment_list)
```

```
#linear regression of average hours of sun light vs percent of people who have␣
 ↪received treatment
avg_hours_sun_list = sunlight_vs_treatment["avg_sunlight"].to_list()
percent_had_treatment_list = sunlight_vs_treatment["percent_had_treatment"].
 ↪to_list()
calc_linear_regression(avg_hours_sun_list, percent_had_treatment_list)
#linear regression of average wind speed vs percent of people who have received␣
 ↪treatment
avg_wind_speed_list = windspeed_vs_treatment["avg_windspeed"].to_list()
percent_had_treatment_list = windspeed_vs_treatment["percent_had_treatment"].
 ↪to_list()
calc_linear_regression(avg_wind_speed_list, percent_had_treatment_list)
```

```
slope b1 is 0.26752838092949816
intercept b0 is 35.14082249599085
squared error is 26970.15858241014
mean squared error is 657.8087459124424
root mean square error is:  25.6477824755366
R2 is:  0.011790752317921482
slope b1 is 17.5935828877007
intercept b0 is -173.72727272727482
squared error is 24468.395721925128
mean squared error is 596.7901395591495
root mean square error is:  24.429288560233378
R2 is:  0.10345744336404905
slope b1 is 2.714769888989919
intercept b0 is 42.86460338807278
squared error is 26316.73171581932
mean squared error is 641.8715052638859
root mean square error is:  25.33518315039159
R2 is:  0.035732861159287754
```

[105]:
```
(42.86460338807278,
 2.714769888989919,
 641.8715052638859,
 25.33518315039159,
 0.035732861159287754)
```

[ ]: