Isha Gosalia and Mengyi Li

Horse Racing and Weather Analysis

Github Repository: https://github.com/igosalia/si206-final-proj

Initial Goals

Our initial goals for this project were to gather data from two websites, Equibase and Wikipedia, and two APIs, the OpenWeather API and the Racing API. We wanted to scrape data from Equibase on the results of horse races in the United States over the past three years and scrape data from Wikipedia on major racing venues in the United States. From the OpenWeather API, we intended to get historical data on the weather conditions at these major horse racing venues, and gather data from the Racing API on horse racing courses and events, as well as information on the horses competing at these events.

In terms of calculations and visualizations, our initial plan was to calculate the average temperatures at races over the last three years, average pace per mile for major racing events, and conduct a hypothesis test and correlation coefficient analysis for multiple independent variables such as temperature, humidity, etc. We also planned to create graphs to visualize these calculated results using Matplotlib.

Goals Achieved

We were able to gather data on horses (weight, rating, age, previous performance) and race cards for today and tomorrow (name, course, date, off time, distance, region, type, track condition, and surface type) from the Racing API and weather data associated with the races on the race cards (location, track name, date, temperature, dew, humidity, windspeed, and visibility) from Visual Crossing Weather API.

From this gathered data, we were able to calculate the average historical temperature, humidity, windspeed, and visibility for each race course on today and tomorrow's race cards. For further analysis, a multiple regression model was fitted to the data to determine the impact each independent weather variable had on the horse rating. We did not use dew in our calculations and regression in order to avoid confounding our results due to its similarity and collinearity to temperature. To visualize the data, we constructed residual plots, scatterplots with the regression line, histograms to show the distribution of each variable, and a correlation matrix to identify variables that were related to each other.

Goals Not Achieved & Problems Faced

We were unable to use Equibase and other racing websites we attempted to substitute due to these websites blocking HTTP requests. As a result, we did not achieve our goal of scraping data from Equibase on the results of horse races in the United States over the past three years. In addition, without the detailed Equibase data, we were unable to calculate the average pace per mile for racing events.

Furthermore, while we originally aimed to scrape data from Wikipedia on major racing venues in the United States, the race card data gathered from the Racing API only had data on European races. We also had trouble identifying a way to use the data scraped from Wikipedia in our calculations, as the website only contained information on courses and their locations. In the end, we ended up not using Wikipedia due to the data falling short of 100 entries and its lack of relevant functionality.

We also ran into the issue of OpenWeather API only offering current data for free, not historical. As a result, we switched to the Visual Crossing weather API, which offered free historical data. For the Racing API, we ended up only able to gather data on today and tomorrow's races. As a result, we had to change the weather API locations in order to match the data gathered from Racing API.

Calculations

```
"""Calculations using weather and racing data"""

import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import numpy as py
import seaborn as sns
import statsmodels.api as sm

database_path = "weather_and_horse_race_data.db"

def load_data_from_db():
    conn = sqlite3.connect(database_path)

    #joining race_card, horse_info, and weather
    query = """
        SELECT rc.course_id, hi.horse_name, hi.horse_age, hi.horse_weight,
        hi.horse_rating, w.temperature, w.humidity, w.windspeed, w.visibility
        FROM race_cards rc
        JOIN horse_info hi ON rc.raceid = hi.raceid
        JOIN weather w ON rc.course_id = w.course_id AND rc.date = w.date
    """
    data = pd.read_sql_query(query, conn)
    conn.close()
    return data

def calculate_weather_info(data):
    #group by track and calculate mean for each weather attr
    weather_stats = data.groupby('course_id')[['temperature', 'humidity', 'windspeed', 'visibility']].mean()
    #add header to weather dataframe
    weather_stats.rename(columns={
        'temperature': 'average_temp',
        'humidity' : 'average_humidity',
        'windspeed' : 'average_windspeed',
        'visibility' : 'average_visibility'
    }, inplace=True)

    return weather_stats
```

We began with joining relevant columns from the race_cards and horse_info on raceid, then joining the weather and race_card tables on course_id and data. This was then added to a Pandas DataFrame for ease of manipulation for our calculations.

To calculate the mean for each weather attribute, we grouped the joined data by course_id, then found the mean for temperature, humidity, windspeed, and visibility, renaming the columns in the new DataFrame for clarity.

```python
def multiple_reg(x, y, x_label, y_label, title):
    #multiple regression
    x = sm.add_constant(x)
    model = sm.OLS(y, x).fit()
    predictions = model.predict(x)

    print(model.summary())
    write_regression_to_file(model.summary().as_text())

    plt.figure(figsize=(10, 6))

    for i, label in enumerate(x_label):
        plt.scatter(x.iloc[:, i + 1], y, label=label, alpha=0.5)

    #regression line
    plt.plot(x.index, predictions, color='red', linewidth=2, label='Regression Line')

    plt.xlabel('Independent Variables')
    plt.ylabel(y_label)
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()

    return model

def calculate_residuals(y, predictions):
    residuals = y - predictions
    return residuals
```

We then performed multiple regression using statsmodels.api. We calculated the residuals by subtracting the predicted values from actual values.

```python
def residual_plot(x, residuals, x_labels, title):
    for label in x_labels:
        plt.figure(figsize=(10, 6))
        plt.scatter(x[label], residuals, alpha=0.5, label=f'Residuals vs {label}')  # Changed to x[label] to match the DataFrame column names
        plt.axhline(0, color='red', linewidth=2)
        plt.xlabel(label)
        plt.ylabel('Residuals')
        plt.title(f'{title} ({label})')
        plt.legend()
        plt.grid(True)
        plt.show()

def histogram(data, column, title):
    plt.figure(figsize=(10, 6))
    plt.hist(data[column].dropna(), bins=20, color='blue', edgecolor='black', alpha=0.7)
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()

def scatter_plot(x, y, x_label, y_label, title):
    plt.figure(figsize=(10, 6))
    plt.scatter(x, y, alpha=0.5, color='blue')
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.grid(True)
    plt.show()

def correlation_plot(data):
    plt.figure(figsize=(10, 8))
    corr_matrix = data[['temperature', 'humidity', 'windspeed', 'visibility', 'horse_rating']].corr()
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)
    plt.title('Correlation Matrix')
    plt.show()
```

We plotted the residuals using Matplotlib, creating a scatterplot with the regression line as axhline.

We then similarly plotted histograms and scatterplots of the data, followed by a correlation matrix made using Seaborn.

```python
def write_stats_to_file(weather_stats, filename='calculated_stats.txt'):
    with open(filename, 'w') as file:
        file.write("Weather Statistics:\n")
        file.write(weather_stats.to_string())
        file.write("\n\n")

def write_regression_to_file(regression_summary, filename='calculated_stats.txt'):
    with open(filename, 'a') as file:
        file.write('\n')
        file.write(regression_summary)
```

We then wrote the weather_stats and regression analysis to the same txt file.

```python
def main():
    # Load data
    data = load_data_from_db()
    print(data.head())

    # Convert relevant columns to numeric
    data[['temperature', 'humidity', 'windspeed', 'visibility', 'horse_rating']] = data[['temperature', 'humidity', 'windspeed', 'visibility', 'horse_rating']].apply(pd.to_numeric, errors

    # Drop NA
    data.dropna(subset=['temperature', 'humidity', 'windspeed', 'visibility', 'horse_rating'], inplace=True)

    # Weather calculations
    weather_stats = calculate_weather_info(data)
    write_stats_to_file(weather_stats)
    print(weather_stats)

    # Vars for regression
    independent_vars = data[['temperature', 'humidity', 'windspeed', 'visibility']]
    dependent_var = data['horse_rating']

    # Multiple regression
    model = multiple_reg(independent_vars, dependent_var, ['temperature', 'humidity', 'windspeed', 'visibility'], 'Horse Rating', 'Multiple Regression for Horse Rating vs Weather Conditio

    # Residuals
    residuals = calculate_residuals(dependent_var, model.predict(sm.add_constant(independent_vars)))
    residual_plot(independent_vars, residuals, ['temperature', 'humidity', 'windspeed', 'visibility'], 'Residuals of Horse Rating vs Weather Conditions')  # Changed labels to match DataFr

    # Scatterplots to show relationships
    scatter_plot(data['temperature'], data['horse_rating'], 'Temperature', 'Horse Rating', 'Horse Rating vs Temperature')
    scatter_plot(data['humidity'], data['horse_rating'], 'Humidity', 'Horse Rating', 'Horse Rating vs Humidity')
    scatter_plot(data['windspeed'], data['horse_rating'], 'Windspeed', 'Horse Rating', 'Horse Rating vs Windspeed')
    scatter_plot(data['visibility'], data['horse_rating'], 'Visibility', 'Horse Rating', 'Horse Rating vs Visibility')

    # Histograms to show distributions
    histogram(data, 'temperature', 'Distribution of Temperature')
    histogram(data, 'humidity', 'Distribution of Humidity')
    histogram(data, 'windspeed', 'Distribution of Windspeed')
    histogram(data, 'visibility', 'Distribution of Visibility')
    histogram(data, 'horse_rating', 'Distribution of Horse Ratings')

    # Correlation plot
    correlation_plot(data)

if __name__ == "__main__":
    main()
```

Finally, first loaded and joined the data, then conducted data preprocessing, making sure that all numeric columns were converted to the right data type (numeric) and that NaN values were dropped.

We then called the calculate_weather_info and write_stats_to_file functions to add our calculated means to the txt file. We then performed the multiple regression, setting temperature, humidity, windspeed, and visibility as our independent variables and horse_rating as the dependent variable, then writing to our txt file.

We then called the scatter_plot and histogram functions, creating scatterplots of our independent variables against the dependent variable and histograms of variable distributions. Finally, we called the correlation_plot function and then called the main function.

Calculations File:

Weather Statistics:

course_id	average_temp	average_humidity	average_windspeed	average_visibility
1	40.360000	88.600000	13.960000	12.180000
2	40.526667	89.473333	15.673333	12.426667
3	41.146667	85.393333	13.460000	12.513333
4	40.866667	87.953333	15.300000	12.553333
5	41.226667	89.493333	13.033333	12.706667
6	40.566667	86.213333	14.080000	12.820000
7	71.406667	53.560000	14.060000	14.800000
11	42.620000	79.713333	15.706667	12.040000
12	40.526667	88.006667	13.813333	12.686667
13	40.193333	87.673333	14.873333	12.800000

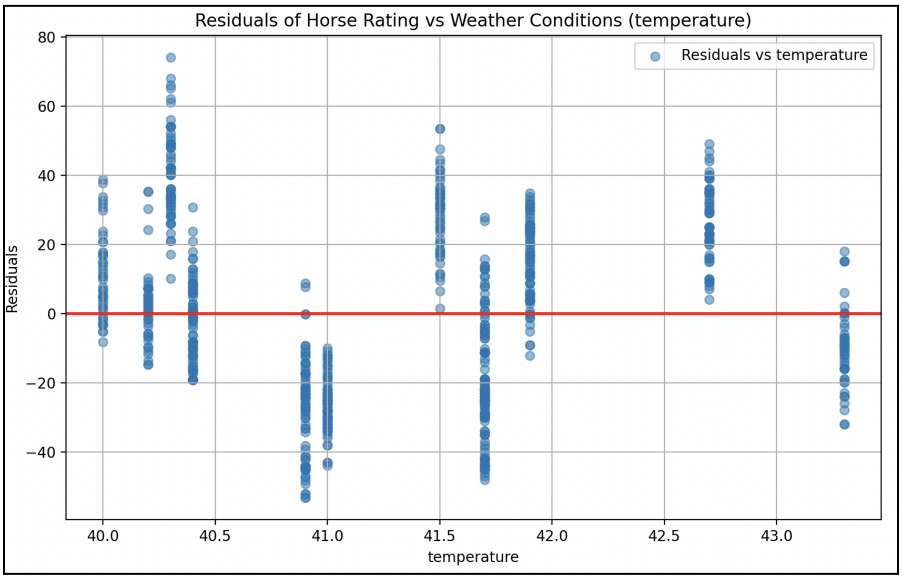
OLS Regression Results

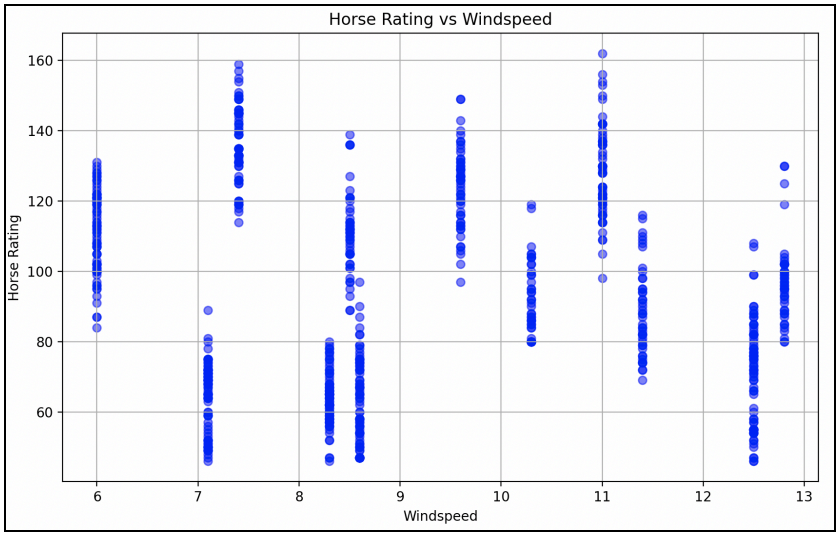
Dep. Variable:	horse_rating	R-squared:	0.019
Model:	OLS	Adj. R-squared:	0.019
Method:	Least Squares	F-statistic:	59.39
Date:	Fri, 13 Dec 2024	Prob (F-statistic):	9.53e-50
Time:	16:07:35	Log-Likelihood:	-57898.
No. Observations:	12180	AIC:	1.158e+05
Df Residuals:	12175	BIC:	1.158e+05
Df Model:	4		
Covariance Type:	nonrobust		

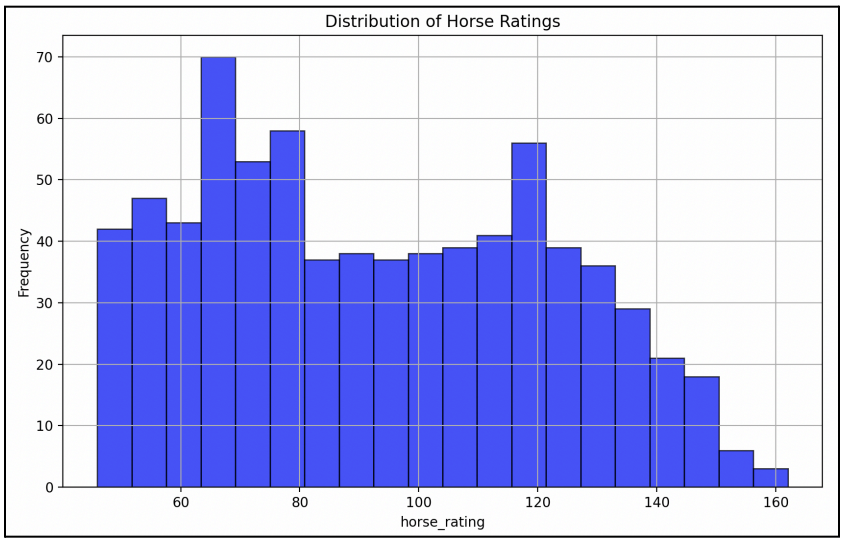
	coef	std err	t	P>|t|	[0.025	0.975]
const	64.1484	4.441	14.445	0.000	55.444	72.853
temperature	-0.0715	0.034	-2.096	0.036	-0.138	-0.005
humidity	0.3335	0.035	9.564	0.000	0.265	0.402
windspeed	0.1490	0.044	3.413	0.001	0.063	0.235
visibility	0.0450	0.085	0.529	0.597	-0.122	0.212

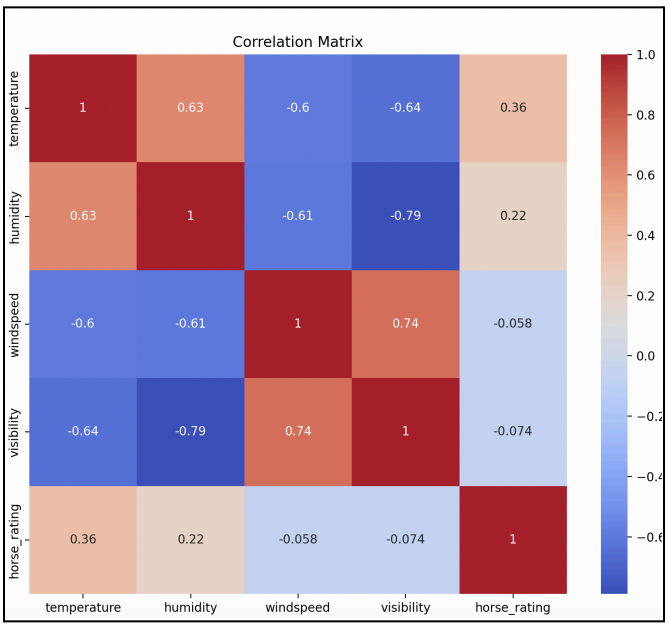
Omnibus:	2998.832	Durbin-Watson:	0.013
Prob(Omnibus):	0.000	Jarque-Bera (JB):	597.547
Skew:	0.200	Prob(JB):	1.75e-130
Kurtosis:	1.991	Cond. No.	1.70e+03

Visualizations




Horse Rating vs Windspeed
Horse Rating
Windspeed
160
140
120
100
80
60
6
7
8
9
10
11
12
13


Distribution of Horse Ratings
Frequency
horse_rating
70
60
50
40
30
20
10
0
60
80
100
120
140
160


Correlation Matrix
temperature
humidity
windspeed
visibility
horse_rating
1
0.63
-0.6
-0.64
0.36
0.63
1
-0.61
-0.79
0.22
-0.6
-0.61
1
0.74
-0.058
-0.64
-0.79
0.74
1
-0.074
0.36
0.22
-0.058
-0.074
1
1.0
0.8
0.6
0.4
0.2
0.0

Instructions for Running Code

To run our code, the first file to run is create_database.py, which creates the database file (weather_and_horse_race_data.db) and each database table if it does not already exist. It also loads the courses and locations into the database so that we can avoid duplicate string data when storing data from the weather and horse racing APIs.

After that, the next file to run is weather_api.py, which makes API requests to the Visual Crossing weather API and adds 15 rows to the database each time, as the Visual Crossing API provides weather forecasting data for the next 15 days. Once all of the weather data has been added to the database, the next file to run is racing_api.py, which makes the API requests if we have not already made them, and adds information on the races to the race_cards table as well as information about the horses competing at each race into the horse_info table. This file has to be run many times as it first adds 25 rows at a time into the race_cards table, and once all of the data has been loaded into the race_cards table it then adds all of the horse information into the horse_info table, adding 25 rows at a time as well.

The last file to run is the calculations_and_graphs.py file, which does the database JOIN calculations and calculates information regarding the average temperature, visibility, windspeed, and humidity as well as comparing horse ratings and performance to the weather conditions. This file also does calculations regarding linear regression and creates various visualizations using these calculations. The calculations are written to the file calculated_stats.txt, as shown above.

Function Documentation

1. Create Database Functions (create_database.py)
    a. def create_db( ):
        i. Creates each table in our database
        ii. No input, returns nothing.
    b. def insert_locations_into_db( ):
        i. Inserts the racing locations we are focusing on into the location_names table in our database, this allows us to avoid having duplicate string data in our other database tables.
        ii. No input, returns nothing.
    c. def insert_courses_into_db( ):
        i. Inserts the race course names into the course_names table in our database, this allows us to avoid having duplicate string data in our other database tables.
        ii. No input, returns nothing.
2. Visual Crossing Weather API functions (weather_api.py)
    a. def get_weather_data( ):
        i. Each time the file is run, this function makes an API request to the next major racing location and gets data on the next 15 days of weather forecasts for that location. Once it has the data, it converts the date into a Unix

timestamp and also retrieves the corresponding race course id from the course_names table in the database and the corresponding location id from the location_names database. It then inserts the weather data into our database.

ii. No input, returns nothing.

3. Racing API functions (racing_api.py)
    a. def make_api_requests( ):
        i. Makes two API requests to get information from the Racing API on today's racecards and tomorrow's racecards. It adds all of the racecards retrieved from the API to a list and saves all of the API data to a json file. This allows us to keep track of the data we have retrieved so that every time we run this file, we can insert the next 25 rows of race data into our database.
        ii. No input, returns nothing.
    b. def get_data( ):
        i. Opens the json file containing the API data and loads it into a list.
        ii. No input, returns the list containing the API data.
    c. def get_racecards( ):
        i. Calls get_data() to get the API data on today and tomorrow's racecards, and then adds the next 25 racecards to the database.
        ii. No input, returns nothing.
    d. def get_runners( ):
        i. Calls get_data to get the API data on today and tomorrow's racecards. Since each race card has information about each horse competing in that race, we add all of the horse information to a list. We then insert 25 rows into the horse_info table in the database each time we run this file.
        ii. No input, returns nothing.
    e. def is_empty( ):
        i. Checks if we have already made API requests and saved the data to the json file. If we have not made any API requests yet, it calls the make_api_requests() function.
        ii. No input, returns nothing.
4. Calculations and Graphs Functions (calculations_and_graphs.py)
    a. def load_data_from_db( ):
        i. Does database JOIN to join the horse_info, weather, and race_cards tables in our database, and reads the SQL query results into a pandas DataFrame.
        ii. No input, returns the DataFrame containing the data.
    b. def calculate_weather_info(data):
        i. Calculates the average temperature, humidity, windspeed, and visibility at each racing course.
        ii. Input is the data from load_data_from_db( ), returns the DataFrame containing these average statistics.
    c. def multiple_reg(x, y, x_label, y_label, title):

    i. Conducts multiple regression analysis using statsmodels.api and calls the write_regression_to_file function to write these statistics to the calculation file (calculated_stats.txt). Then, plots the results using Matplotlib.
    ii. Input: x, y, x_label, y_label, title; returns the fitted regression model.
d. def calculate_residuals(y, predictions):
    i. Calculates the residuals from the regression model.
    ii. Input: y and predictions, returns the calculated residuals.
e. def residual_plot(x, residuals, x_labels, title):
    i. Plots the residuals using Matplotlib.
    ii. Inputs: x, residuals, x_labels, title; returns nothing.
f. def histogram(data, column, title):
    i. Plots a histogram of the input column (temperature, horse rating, humidity, etc.)
    ii. Inputs: data, column, title; returns nothing.
g. def scatter_plot(x, y, x_label, y_label, title):
    i. Plots a scatter plot of the inputs x and y.
    ii. Inputs: x, y, x_label, y_label, title; returns nothing.
h. def correlation_plot(data):
    i. Creates a correlation matrix using the data on horse rating, humidity, visibility, wind speed and temperature.
    ii. Input: data, returns nothing.
i. def write_stats_to_file(weather_stats, filename= 'calculated_stats.txt'):
    i. Writes the calculated weather statistics from the calculate_weather_info function to the file "calculated_stats.txt".
    ii. Inputs: weather_stats, filename; returns nothing.
j. def write_regression_to_file(regression_summary, filename= 'calculated_stats.txt'):
    i. Writes the calculated regression results from the multiple_reg function to the file "calculated_stats.txt"
    ii. Inputs: regression_summary, filename; returns nothing.
k. def main( ):
    i. Calls the load_data_from_db function to get the data, then calls the weather_stats function to get the weather calculations and the write_stats_to_file function to write the weather calculations to the calculations file. The main function then calls multiple_reg, calculate_residuals, residual_plot, scatter_plot, histogram, and correlation_plot functions to generate our visualizations and calculations.
    ii. No inputs, returns nothing.

Resources

Date	Issue Description	Location of Resource	Result
12/13/2024	No prior experience using statsmodels, needed guidance on how to create regression models	https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html	Understood the information on OLS
12/13/2024	Unsure how to use statsmodels with Pandas	https://stackoverflow.com/questions/29186436/multiple-linear-regression-in-pandas-statsmodels-valueerror	Was able to create OLS for multiple independent variables
12/13/2024	No prior experience using Seaborn, needed guidance on how to create correlation matrix	https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html	Was able to successfully create correlation matrix
12/13/2024	Duplicate string data in multiple databases due to the date column being a string, so needed to convert this to a Unix timestamp.	https://docs.python.org/3/library/datetime.html#datetime.datetime.strptime	Was able to convert the dates into Unix timestamps before inserting data into the database.