Infrastructure Factors Affecting National Park Visitation

Amy N. Aragon

Western Governors University

Table of Contents

A. Project Overview

Research Question

This project addresses the research question, "How does proximity to major transportation hubs and accommodations affect National Park visitation numbers?" Understanding the extent to which these factors influence visitation can help authorities improve accessibility, promote under-visited parks, and enhance visitors' experiences. This project analyzes National Park visitation data alongside geographic proximity metrics to gain insights into how accessibility impacts park popularity.

Scope

The scope of the project is limited only to proximity metrics including distance to major highways, distance to airports, and amount of available lodging within a radius of each park. This project does not consider any other individual park characteristics. The analysis does not include environmental factors, such as climate, nor does it consider any visitor demographic information.

Analytics Solution

For the first step of the project, I produced a clean dataset containing National Park visitation totals for the past five years, the latitude and longitude of each park, the distance to a highway in miles, the distance to an airport in miles, and the number of available lodging accommodations within a 31-mile radius of each park. Next, an exploratory analysis was performed to identify missing data and examine descriptive statistics. Lastly, Pearson correlation and multiple linear regression were used to identify the significance of each variable and its

effect on visitation numbers. Python was used for data cleaning and statistical modeling, Jupyter Notebook was used for iterative coding and analysis, and Tableau to visualize geographic data.

B. Project Execution

Project Plan

The execution of the project followed the general outline from the initial proposal with a few differences. Originally it was stated that the latitude and longitude of each National Park would be manually added to the dataset. However, to increase efficiency, I implemented a Python function using Nominatum from the geopy library. This function automatically fetched the latitude and longitude for each park using the park's name, saving time and reducing manual input errors. This adjustment significantly improved the overall workflow by automating a time-consuming step.

Project Planning Methodology

The project adhered to an Agile methodology which allowed for iterative progress and flexibility. The phases from the project proposal were completed in order and as planned, revisiting phases to make refinements as applicable. For instance, revisiting the data extraction phase to implement the above stated Python function greatly improved efficiency.

Project Time and Milestones

The timeline of the project was condensed compared to the original proposal. Data extraction and cleaning was the most extensive step of the project, taking one week. The data cleaning report was completed simultaneously. Once the cleaned dataset was obtained, conducting the statistical analysis and generating visualizations took only two days compared to the two weeks from the project proposal.

C. Methodology

Data Selection and Collection

The data selection and collection process proceeded largely as planned, with one notable change: instead of manually inputting latitude and longitude for each National Park, I used an automated function written in Python that utilized Nominatum from the geopy library. The collection of the other data, such as distance to highways, airports, and amount of available lodging, proceeded without issue. However, the Google Places API, which was used to obtain proximity to airports and amount of lodging, limited the search radius to 50,000 meters, or approximately 31 miles.

Handling Obstacles

No obstacles were encountered during the extraction phase. However, the 50,000-meter search radius limit from Google Places API was an unexpected restriction. Given the large size of the parks, it's possible that not all airports or accommodations within proximity to the parks were captured. I opted to use the maximum allowed radius, which seemed reasonable given the scope of this study and made sure to document the possible effect on the completeness of the data.

Data Governance Issues

The project did not encounter any unexpected data governance issues. Data quality was ensured by using appropriate and reputable data sources, such as obtaining visitation data directly from the National Park Service, following thorough data cleaning steps, and handling missing values. Transparency in the methodology was ensured by documenting constraints, such as the Google Places API limitations.

D. Data Extraction and Preparation Processes

Data Extraction

The data extraction process involved gathering National Park visitation data and proximity metrics, including the distance to highways, airports, and number of lodging accommodations near each park.

1. National Park Visitation Data

   I began by downloading yearly National Park visitation data as a CSV file directly from the National Park Service (NPS) Visitor Use Statistics portal. The data provided yearly visitation numbers for each park. I imported and converted the CSV file into a dataframe using Python and the pandas library.

2. Conversion and Data Cleaning

   After downloading the CSV, I used Python and the pandas library to import and convert the file into a dataframe. The conversion allowed me to manipulate and analyze the data more easily. I then removed rows and columns that contained only missing (NaN) values to maintain the integrity of the dataset. To simplify the dataset, I summed the visitation numbers for the past five years into a single column (representing total visitation over this period). The remaining columns were not necessary for the analysis and were dropped.

Data Preparation

Once the visitation data was cleaned and prepared, I focused on calculating proximity metrics for each National Park, as these were needed to answer the research question.

1. Latitude and Longitude Data

The NPS data did not include geographic coordinates for the parks, which were necessary for calculating proximity metrics. To obtain this data, I implemented a Python function using Nominatum API from the geopy library. This function searched for each park's name and retrieved its latitude and longitude. Once the geographic data was added to the dataframe, I plotted the data on a map to visually verify its accuracy.

2. Proximity to Highways

   To calculate the distance from each park to the nearest highway, I used the OpenStreetMap Overpass API to extract the highway data for a 100-mile radius. A Python function was written to compute the shortest distance between each park and nearest highway, which was then added as a new column to the dataframe.

3. Proximity to Airports

   To calculate the distance to the nearest airport, I used the Google Places API to search for airports within the vicinity of each park (the radius was limited to 50,000 meters, approximately 31 miles). A Python function was used to calculate the distance from each park to its nearest airport and add this information to the dataframe.

4. Lodging Accommodations

   To determine the number of available lodging accommodations within a 31-mile radius of each park, I used the Google Places API again, specifically to query lodging accommodations. The total number of accommodations was added as an additional column.

Tools and Techniques

- The pandas library was used extensively to manipulate data, clean it, and calculate key information such as the total visitation for each park.
- The geopy library and Nominatum API were used to retrieve geographic coordinates (latitude and longitude).
- OpenStreetMap Overpass API was used to obtain highway proximity data.
- Google Places API was used to gather information on airports and lodging accommodations.

The use of geopy and OpenStreetMap was more efficient and scalable than manually entering latitude, longitude, and highway proximity data. Both OpenStreetMap and Google Places API were suitable for high quality, real-time geographic data that was accurate and easy to integrate into the Python workflow. The use of a 100-mile proximity for highway data and a 31-mile radius for airports and lodging was meant to capture a meaningful area around each park, although it was limited by the maximum radius of the Google Places API. Overall, the data extraction and preparation processes were efficient and leveraged Python libraries and APIs for automated data collection and accurate calculations.

E. Data Analysis Process

Methods

The primary methods used to analyze the data were exploratory data analysis (EDA), correlation analysis, and multiple linear regression. These methods were chosen to identify the relationships and significance of proximity metrics (distance to highways, distance to airports, and number of lodging accommodations) and National Park visitation numbers.

- Exploratory data analysis: I performed EDA on the cleaned and completed dataset. I computed basic statistics such as mean and standard deviation.
- Correlation analysis: I calculated Pearson correlation coefficients to assess the linear relationships between proximity metrics and the dependent variable (visitation numbers) and their statistical significance.
- Multiple linear regression: I performed multiple linear regression to model the relationship between the dependent variable and the independent variables. This analysis was conducted using the statsmodels library, which provided detailed results.

Advantages and Disadvantages

Advantages:

- Python Libraries: The libraries used, such as pandas and statsmodels, are powerful and efficient. They allow for easy handling of large datasets and provide robust, comprehensive analyses.
- Pearson Correlation: The Pearson correlation coefficient is an effective method for quantifying the linear relationship between variables.
- Multiple Linear Regression: This method is widely used in data analysis and helps assess the influence of multiple variables on a dependent variable.

Limitations:

- Linearity: The Pearson correlation and multiple linear regression methods assume a linear relationship between variables. If non-linear relationships exist, these methods may not capture the true patterns.
- Multicollinearity: If the independent variables are highly correlated with each other, it could lead to multicollinearity, which reduces the accuracy of the regression coefficients.

- Data Quality: The accuracy of the analysis is dependent on the data quality. While I used reliable APIs for data collections, limitations in the geographic data (such as the 31-mile radius limit) could have impacted the results.
- Assumption for Regression: Multiple linear regression assumes a normal distribution. If this assumption is violated, the results may not be accurate.

Step-by-Step Analysis

1. Exploratory Data Analysis:
    a. Data cleaning: I removed rows with missing values and dropped unnecessary columns.
    b. Descriptive statistics: I calculated basic statistics using pandas' describe() function. I calculated z-scores and identified parks with z-scores greater than 3 or less than -3 to find outliers. I found that Great Smoky Mountains National Park was an outlier, with much higher visitation than the other parks.
    c. Visualizations: I used a heat map visualization to identify missing values. The only column with missing values was the "Distance to Nearest Airport (miles)" column, so I used a box plot to visualize the differences in visitation numbers between parks with no nearby airport and parks that did have a nearby airport to see if that relationship was worth exploring.
2. Correlation Analysis:
    a. I calculated Pearson correlation coefficients between the independent and dependent variables and p-values using df.corr(). This step provided an initial understanding of which variables might have a significant linear relationship with visitation, as well as statistical significance.

3. Multiple Linear Regression
    a. I used statsmodels library to fit the multiple linear regression model. I defined the dependent variable y (total visitation) and the independent variables X (distance to highway, distance to airport, and number of accommodations).
    b. I used scatterplots to visualize any linear relationship between the variables.
    c. After fitting the model, I calculated the Variance Inflation Factor (VIF) for each independent variable using statsmodels to rule out multicollinearity. There was no significant multicollinearity influencing the results.
4. Evaluation and Verification
    a. The regression output provided coefficients for each independent variable. I checked the p-values for each coefficient to determine statistical significance of the variables.
    b. I reviewed the R-squared and Adjusted R-squared values to assess how well the model explain the variation in total visitation.
    c. I re-ran the multiple linear regression model without the outlier (Great Smoky Mountains National Park) to see if this had significantly influenced the results. The removal of this park had minimal effect on the coefficients, confirming it was not a driving factor in the overall regression.

F. Results

Output of Analytics Solution

An exploratory analysis of the data revealed that the only column with any missing values was the “Distance to Nearest Airport (miles)” column, as shown in Figure 1. I was interested in whether the parks with no nearby airport had a significant difference in visitation

numbers compared to the parks that did have an airport in the vicinity. An independent t-test showed a p-value of 0.0000418781, much less than the <0.05 threshold. This indicates that having a nearby airport versus having no airport has a significant impact on visitation as visualized in Figure 2. Of note, the Pearson correlation and multiple linear regression analyses excluded the parks with missing airports, as NaN values could not be included.

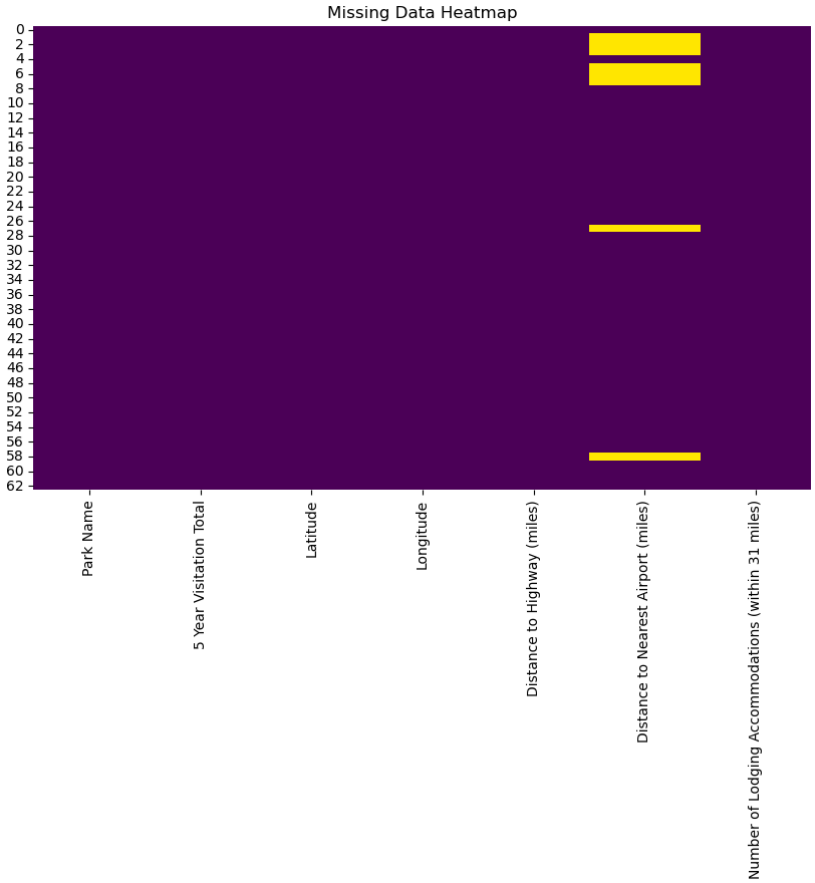


Figure 1. Missing Data Heatmap

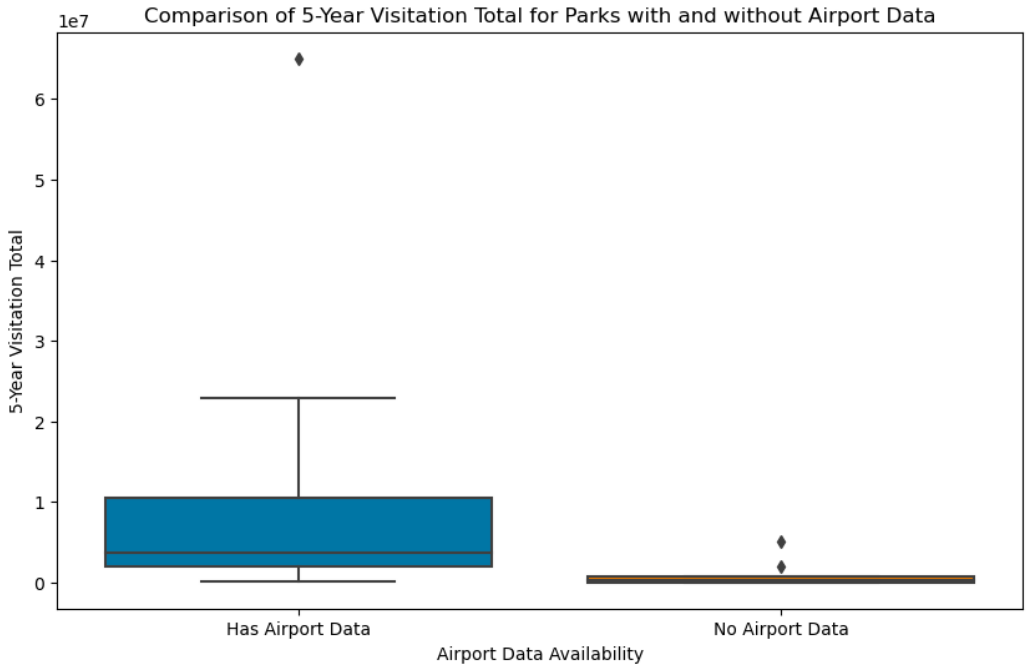


Figure 2. Comparison of Parks With and Without Airports

Table 1 shows the results of the Pearson correlation analysis. An r-value of 1 to -1 indicates the strength of the linear relationship. The results indicated that the independent variables of distance to highways, distances to airports, and amount of lodging accommodations did not have a strong linear relationship.

Variable	Pearson Correlation	P-value
Distance to Highway (miles)	-0.0296	0.8299
Distance to Airport (miles)	0.0952	0.4894
Number of Lodging Accommodations	0.1552	0.2579

Table 1. Pearson Correlation

The results of the multiple linear regression had an R-squared of 0.031 and an Adjusted R-squared of -0.026. An R-squared value of greater than 0.6 would indicate strong explanatory power, meaning that the proximity metrics did not have a strong explanatory power on visitation numbers. As we can see in Figure 3, no linear relationship is observed. The individual p-values

were 0.953 for Distance to Highway (miles), 0.555 for Distance to Nearest Airport (miles), and 0.291 for Number of Lodging Accommodations (within 31 miles). A p-value of less than 0.05 would indicate statistical significance, so these results suggest that the proximity metrics do not have a significant impact on visitation numbers.

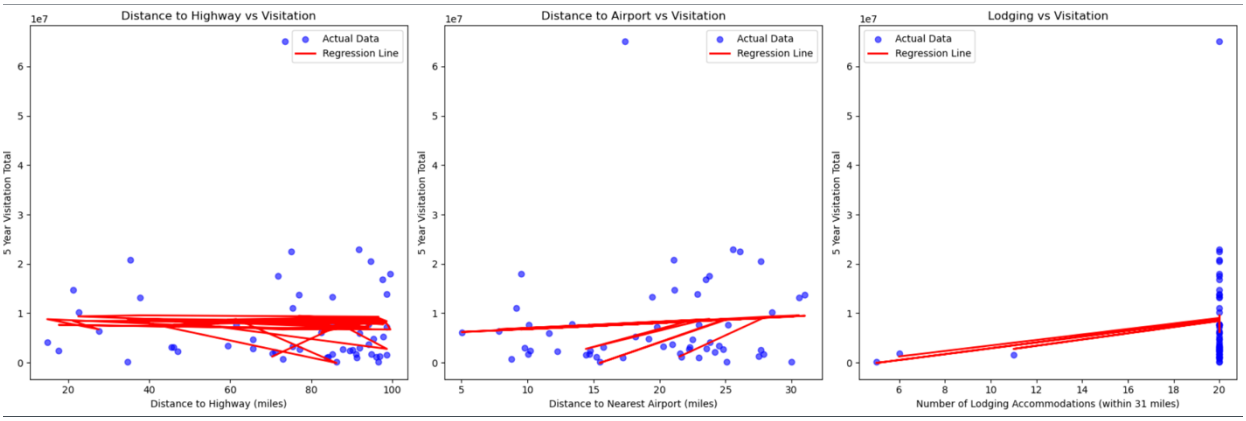


Figure 3. Results of Multiple Linear Regression Analysis

Practical Significance

The project's results indicate that the proximity to infrastructure is not a predictor of National Park visitation. This may point to the importance of other factors that were not included in the analysis, such as visitor demographics, park amenities, or environmental conditions. For example, a park located far away from a major highway may still see high visitation numbers due to the activities available in the park. The only statistically significant result was the comparison between parks with and without airports, meaning that the availability of an airport, not necessarily the distance, could contribute to visitation.

Overall Success

The project successfully highlighted the need for further research into additional factors impacting visitation numbers. The analysis of proximity metrics provided some insights, though

these metrics did not prove to be strong predictors. The t-test results on the impact of nearby airports were significant, adding valuable context. The effectiveness of the project could be improved by using a different method to gather proximity data that is not limited by the 50,000-meter radius of the Google Places API. Inclusion of a larger area may have affected the analysis. Incorporating other factors such as park-specific attractions, weather, and visitor demographics may also have yielded more insightful results.

G. Summary

Conclusions

The analysis revealed that proximity metrics such as distance to highways, airports, and the number of lodging accommodations did not have a strong linear relationship to National Park visitation numbers. Parks with nearby airports had significantly higher visitation compared to those without, indicating that the availability of airports may play a role in attracting visitors. The proximity metrics did not show strong relationships with visitation as evidenced by the low Pearson correlation values and poor explanatory power of the multiple linear regression model. The analysis suggests that other factors that were not included in the study may influence visitation numbers more significantly than proximity to infrastructure.

Effectiveness of Tools

Python libraries such as pandas and statsmodels allowed for efficient data cleaning, analysis, and modeling. They provided a robust and detailed analysis of the dataset. Descriptive statistics allowed for quick insights into outliers and missing data, guiding further analysis, such as the comparison between parks with and without nearby airports. The use of visualizations such as heatmaps for missing data and a box plot for visualizing the impact of airport proximity helped communicate the findings clearly for effective storytelling. For instance, the scatterplots

in Figure 3 make the multiple linear regression analysis easier to explain and understand by showing the lack of linear relationships.

Recommendations

1. Explore Other Factors Affecting Visitation

   Since proximity metrics did not show a strong relationship with visitation, further research is needed to identify other factors that might influence park visitation. Visitor demographics, environmental factors, and marketing efforts are all ideas for future analysis. Exploring other variables could help build a more comprehensive model for predicting visitation patterns.

2. Focus on Airport Accessibility

   Due to the significant difference in visitation for parks with and without airports, consider developing or improving airport infrastructure to increase accessibility. Targeting investments in airports or improving connections between airports and more remote parks may boost visitation.

References

Google. (n.d.). *Distance matrix API overview*. Google Developers. Retrieved December 21, 2024, from https://developers.google.com/maps/documentation/distance-matrix/overview

Google. (n.d.). *Places API: Nearby search*. Google Developers. Retrieved December 21, 2024, from https://developers.google.com/maps/documentation/places/web-service/nearby-search

National Park Service. (n.d.). *Visitor use statistics*. U.S. Department of the Interior. Retrieved December 21, 2024, from https://www.nps.gov/subjects/socialscience/visitor-use-statistics.htm

OpenStreetMap Wiki. (n.d.). *Overpass API.* OpenStreetMap. Retrieved December 25, 2024, from https://wiki.openstreetmap.org/wiki/Overpass_API

Statsmodels Documentation. (n.d.). *Regression models.* Statsmodels. Retrieved December 21, 2024, from https://www.statsmodels.org/stable/regression.html

Appendix A: Complete Dataset

Park Name	5 Year Visitation Total	Latitude	Longitude	Distance to Highway (miles)	Distance to Nearest Airport (miles)	Number of Lodging Accommodations (within 31 miles)
Denali National Park	1811807	63.23166	-151.0405554	70.40398458	21.52003081	6
Gates of the Arctic National Park	41254	67.75084	-153.2474557	88.33611608		0
Glacier Bay National Park	2017020	58.81418	-136.8720935	90.1585901		0
Katmai National Park	228113	58.50498	-155.0896226	66.31897838		3
Kenai Fjords National Park	1661733	59.86856	-150.0257714	90.98039594	27.85451891	20
Kobuk Valley National Park	73032	67.38982	-159.0574319	100.4272502		0
Lake Clark National Park	75298	60.70315	-153.2659216	65.75853634		3
Wrangell-St. Elias National Park	284903	61.1809	-143.827636	85.06179202		3
Arches National Park	7647347	38.72653	-109.5630201	89.65354877	10.11570657	20
Big Bend National Park	2462195	29.33325	-103.1941721	17.69465095	14.71634714	20
Black Canyon of the Gunnison National Parl	1737674	38.57981	-107.743697	94.78888425	10.05503387	20
Bryce Canyon National Park	10980088	37.57079	-112.1855939	75.42003206	9.135515892	20
Canyonlands National Park	3718973	38.23326	-109.9206627	94.14866429	20.98879505	20
Capitol Reef National Park	6109379	38.06703	-111.1552562	82.42063061	5.021952739	20
Carlsbad Caverns National Park	1758823	32.12125	-104.6012306	85.24730549	14.70693558	20
Glacier National Park	13672433	48.61749	-113.7608851	76.96641455	31.0094322	20
Grand Canyon National Park	22869992	36.30785	-112.292896	91.67473159	25.54900136	20
Grand Teton National Park	16803811	43.81057	-110.6487948	97.52914086	23.47928158	20
Great Sand Dunes National Park	2597338	37.79119	-105.6122672	90.23179446	27.66410568	20
Guadalupe Mountains National Park	1030707	31.91611	-104.900737	91.21259049	17.20999012	20
Mesa Verde National Park	2397141	37.25378	-108.4555964	89.53664426	10.22440179	20
Petrified Forest National Park	2644105	34.97448	-109.7079943	87.82525457	24.79134711	20
Rocky Mountain National Park	20826361	40.35539	-105.7176957	35.30417428	21.0862594	20
Saguaro National Park	4781338	32.16838	-110.6183452	95.38500775	19.17432279	20
White Sands National Park	3240860	32.76418	-106.331193	75.33772001	20.30604572	20
Yellowstone National Park	20478460	44.62009	-110.5606893	94.73821572	27.68394651	20
Zion National Park	22435012	37.32474	-113.0048036	75.06225942	26.09173129	20
Badlands National Park	5165365	43.52711	-102.3616629	40.09086006		18
Cuyahoga Valley National Park	13342271	41.25509	-81.53764172	85.26029882	19.46112686	20
Gateway Arch National Park	7728021	38.62451	-90.18568335	94.1300639	13.39465453	20
Hot Springs National Park	10127352	34.52907	-93.03014017	22.52856524	28.53551582	20
Indiana Dunes National Park	13204673	41.61527	-87.20726193	37.67206161	30.56888502	20
Isle Royale National Park	113166	48.00738	-88.82898752	96.61788511	30.00904136	20
Theodore Roosevelt National Park	3454587	46.95848	-103.4650052	59.47205905	24.50257804	20
Voyageurs National Park	1181366	48.46555	-92.88539732	96.02916057	21.64565964	20
Denali National Park	10932230.35	34.08272	-83.38419555	71.45402665	25.24119903	24.80672269
Gates of the Arctic National Park	11176557.44	33.51435	-81.64854959	71.09127533		25.27535014
Glacier Bay National Park	11420884.53	32.94599	-79.91290363	70.72852401		25.74397759
Katmai National Park	11665211.61	32.37763	-78.17725768	70.3657727		26.21260504
Kenai Fjords National Park	11909538.7	31.80927	-76.44161172	70.00302138	26.43616609	26.68123249
Kobuk Valley National Park	12153865.78	31.24091	-74.70596576	69.64027006	26.73490785	27.14985994
Lake Clark National Park	12398192.87	30.67255	-72.9703198	69.27751874	27.03364962	27.61848739
Wrangell-St. Elias National Park	12642519.96	30.10419	-71.23467384	68.91476743	27.33239138	28.08711485
Arches National Park	12886847.04	29.53582	-69.49902788	68.55201611	27.63113315	28.5557423
Big Bend National Park	13131174.13	28.96746	-67.76338192	68.18926479	27.92987491	29.02436975
Black Canyon of the Gunnison National Parl	13375501.22	28.3991	-66.02773596	67.82651348	28.22861668	29.4929972
Bryce Canyon National Park	13619828.3	27.83074	-64.29209	67.46376216	28.52735844	29.96162465
Canyonlands National Park	13864155.39	27.26238	-62.55644404	67.10101084	28.82610021	30.4302521
Capitol Reef National Park	14108482.47	26.69402	-60.82079809	66.73825953	29.12484197	30.89887955
Carlsbad Caverns National Park	14352809.56	26.12566	-59.08515213	66.37550821	29.42358374	31.367507
Glacier National Park	14597136.65	25.55729	-57.34950617	66.01275689	29.7223255	31.83613445
Grand Canyon National Park	14841463.73	24.98893	-55.61386021	65.65000557	30.02106726	32.3047619
Grand Teton National Park	15085790.82	24.42057	-53.87821425	65.28725426	30.31980903	32.77338936
Great Sand Dunes National Park	15330117.91	23.85221	-52.14256829	64.92450294	30.61855079	33.24201681
Guadalupe Mountains National Park	15574444.99	23.28385	-50.40692233	64.56175162	30.91729256	33.71064426
Mesa Verde National Park	15818772.08	22.71549	-48.67127637	64.19900031	31.21603432	34.17927171
Petrified Forest National Park	16063099.16	22.14713	-46.93563041	63.83624899	31.51477609	34.64789916
Rocky Mountain National Park	16307426.25	21.57877	-45.19998445	63.47349767	31.81351785	35.11652661
Saguaro National Park	16551753.34	21.0104	-43.46433849	63.11074636	32.11225962	35.58515406
White Sands National Park	16796080.42	20.44204	-41.72869254	62.74799504	32.41100138	36.05378151
Yellowstone National Park	17040407.51	19.87368	-39.99304658	62.38524372	32.70974315	36.52240896
Zion National Park	17284734.6	19.30532	-38.25740062	62.0224924	33.00848491	36.99103641

Appendix B: Links to Data Sources

National Park Visitation Reports

Geopy Documentation

OpenStreetMap Overpass API

Google Places API

Appendix C: Data Cleaning Report

Data Collection & Cleaning

National Parks Visitation Numbers

For the first step of this project, we will import visitation data from the National Park Service Visitor Use Portal that has been manually extracted as a csv file and convert it to a dataframe.

In [ ]:
```
import pandas as pd
df = pd.read_csv('/Users/amyaragon/Desktop/NPS Data/annual_visitation_nps.csv'
df.head()
```

Out[ ]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	U
0	Annual Visitation and Record Year by Park (190...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	Region	Park Name	Park Type	2023	2022	NaN	2021	2020	
3	Alaska Region	Denali NP & PRES	National Park	498,722	427,562	NaN	229,521	54,850	
4	NaN	Gates of the Arctic NP & PRES	National Park	11,045	9,457	NaN	7,362	2,872	

5 rows × 124 columns

We can see that there is some clean-up that needs to be done. Many columns contain NaN values and Row 1 is entirely empty. The headers are also unnamed. We will drop the empty row and the first column, which provides regional information that is unneeded for the project.

In [ ]:
```
df = df.iloc[2:].reset_index(drop=True)
df = df.drop(df.columns[0], axis=1)
df.head()
```

Out[ ]:

	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	L
0	Park Name	Park Type	2023	2022	NaN	2021	2020	2019	
1	Denali NP & PRES	National Park	498,722	427,562	NaN	229,521	54,850	601,152	
2	Gates of the Arctic NP & PRES	National Park	11,045	9,457	NaN	7,362	2,872	10,518	
3	Glacier Bay NP & PRES	National Park	703,659	545,758	NaN	89,768	5,748	672,087	
4	Katmai NP & PRES	National Park	33,763	33,908	NaN	24,764	51,511	84,167	

5 rows × 123 columns

Next, we will remove column 2 which shows park types as the only information imported from NPS was for National Parks, so the park types are all the same. We will also remove column 5 which shows entirely NaN values.

In [ ]:
```
df = df.drop(df.columns[[1,4]], axis=1)
df.head()
```

Out[ ]:

	Unnamed: 1	Unnamed: 3	Unnamed: 4	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	L
0	Park Name	2023	2022	2021	2020	2019	2018	2017	
1	Denali NP & PRES	498,722	427,562	229,521	54,850	601,152	594,660	642,809	
2	Gates of the Arctic NP & PRES	11,045	9,457	7,362	2,872	10,518	9,591	11,177	
3	Glacier Bay NP & PRES	703,659	545,758	89,768	5,748	672,087	597,915	547,057	
4	Katmai NP & PRES	33,763	33,908	24,764	51,511	84,167	37,818	37,818	

5 rows × 121 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65 entries, 0 to 64
Columns: 121 entries, Unnamed: 1 to Unnamed: 123
dtypes: object(121)
memory usage: 61.6+ KB
```

Upon exploring the data types in the dataframe, all of the values are object data types. We will need to remove the commas from the numerical values and convert them to integers. We will also replace NaN values with 0.

In [ ]:
```
# Replae NaN with 0
df = df.fillna(0)
# Remove all commas, spaces, and non-numeric characters
df.iloc[:, 1:] = df.iloc[:, 1:].replace({',': '', ' ': '', '$': ''}, regex=Tru
df.head()
```

Out[ ]:

	Unnamed: 1	Unnamed: 3	Unnamed: 4	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	U
0	Park Name	2023	2022	2021	2020	2019	2018	2017	
1	Denali NP & PRES	498722	427562	229521	54850	601152	594660	642809	
2	Gates of the Arctic NP & PRES	11045	9457	7362	2872	10518	9591	11177	
3	Glacier Bay NP & PRES	703659	545758	89768	5748	672087	597915	547057	
4	Katmai NP & PRES	33763	33908	24764	51511	84167	37818	37818	

5 rows × 121 columns

We are planning to look at the visitation totals for the past 5 years and store the totals in the "5 Year Visitation Column." I am going to set up my column names, remove unneeded columns, and convert the park visitation numbers to integer types so that they can be summed.

In [ ]:
```
# Renaming the first two columns and making them the header
df.rename(columns={df.columns[0]: "Park Name"}, inplace=True)
df.rename(columns={df.columns[1]: "5 Year Visitation Total"}, inplace=True)
df = df.drop(index=0).reset_index(drop=True)
print(df)
```

```
                        Park Name 5 Year Visitation Total Unnamed: 4  \
0                  Denali NP & PRES                498722     427562
1   Gates of the Arctic NP & PRES                 11045       9457
2            Glacier Bay NP & PRES                703659     545758
3                 Katmai NP & PRES                 33763      33908
4                  Kenai Fjords NP                387525     389943
..                             ...                   ...        ...
59                   Everglades NP                810189    1155193
60        Great Smoky Mountains NP               13297647   12937633
61                  Mammoth Cave NP                654450     663147
62                Virgin Islands NP                343685     196752
63                                0                     0          0

   Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9 Unnamed: 10 Unnamed: 11  \
0      229521      54850     601152     594660      642809      587412
1        7362       2872      10518       9591       11177       10047
2       89768       5748     672087     597915      547057      520171
3       24764      51511      84167      37818       37818       37818
4      411782     115882     356601     321596      303598      346534
..        ...        ...        ...        ...         ...         ...
59     942130     702319    1118300     597124     1018557      930907
60   14161548   12095720   12547743   11421200    11338893    11312786
61     515774     290392     551590     533206      587853      586514
62     323999     167540     133398     112287      304408      411343
63          0          0          0          0           0           0

   Unnamed: 12  ... Unnamed: 114 Unnamed: 115 Unnamed: 116 Unnamed: 117  \
0       560757  ...            0            0           0            0
1        10745  ...            0            0           0            0
2       551353  ...            0            0           0            0
3        37818  ...            0            0           0            0
4       296697  ...            0            0           0            0
..         ...  ...          ...          ...         ...          ...
59     1077427  ...            0            0           0            0
60    10712674  ...            0            0           0            0
61      566895  ...            0            0           0            0
62      438372  ...            0            0           0            0
63           0  ...            0            0           0            0

   Unnamed: 118 Unnamed: 119 Unnamed: 120 Unnamed: 121 Unnamed: 122  \
0             0            0            0           0            0
1             0            0            0           0            0
2             0            0            0           0            0
3             0            0            0           0            0
4             0            0            0           0            0
..          ...          ...          ...         ...          ...
59            0            0            0           0            0
60            0            0            0           0            0
61            0            0            0           0            0
62            0            0            0           0            0
63            0            0            0           0            0

   Unnamed: 123
0             0
1             0
2             0
3             0
4             0
..          ...
59            0
```

```
60            0
61            0
62            0
63            0

[64 rows x 121 columns]
```

In [ ]:
```
# Keep only the first 6 columns, retaining only the data for the past 5 years
df = df.iloc[:, :6]
# Drop row 63 as it only contains 0 values
df = df.drop(index=63).reset_index(drop=True)
# Convert all visitation numbers to numeric data type
df.iloc[:, 1:] = df.iloc[:, 1:].apply(lambda col: pd.to_numeric(col, errors='c
```

In [ ]:
```
 # Examine the data types in each column to confirm that it worked
print(df.iloc[:, :].apply(lambda x: x.map(type)).value_counts())
```

```
Park Name      5 Year Visitation Total  Unnamed: 4     Unnamed: 6     Unnamed:
7     Unnamed: 8
<class 'str'>  <class 'int'>           <class 'int'>  <class 'int'>  <class
'int'>  <class 'int'>    63
Name: count, dtype: int64
```

In [ ]:
```
# Sum the values in all columns except the first column (excluding the header)
df['5 Year Visitation Total'] = df.iloc[:, 1:].sum(axis=1)
```

In [ ]:
```
# Drop all columns except the park name and total visitation for 5 years
df = df[['Park Name', '5 Year Visitation Total']]
# Verify the results
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total
0	Denali NP & PRES	1811807
1	Gates of the Arctic NP & PRES	41254
2	Glacier Bay NP & PRES	2017020
3	Katmai NP & PRES	228113
4	Kenai Fjords NP	1661733

Now we have a dataframe consisting of a "Park Name" column with all national parks listed, and a "5 Year Visitation Total" that contains the sum of the visitation totals for each park from the past 5 years. In the "Park Name" column, some parks end in "& PRES", and all end with "NP." I am going to remove "& PRES" and expand "NP" to "National Park" for tidiness and ease of interpretation.

In [ ]:
```
# Remove "& PRES" from the "Park Name" column
df['Park Name'] = df['Park Name'].str.replace(r'& PRES$', '', regex=True)

# Expand "NP" to "National Park" in the "Park Name" column
df['Park Name'] = df['Park Name'].str.replace(r'\bNP\b', 'National Park', rege

# Verify the result by displaying the first few rows
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total
0	Denali National Park	1811807
1	Gates of the Arctic National Park	41254
2	Glacier Bay National Park	2017020
3	Katmai National Park	228113
4	Kenai Fjords National Park	1661733

Now, we need to add our additional data, the first of which is latitude and longitude for each park. To do this I will use Nominatum from geopy and create a function that searches for the latitude and longitude using the park names and adds it to the dataset.

In [ ]:
```
import geopy
```

In [ ]:
```
from geopy.geocoders import Nominatim
import time
```

In [ ]:
```
# Initialize geolocator
geolocator = Nominatim(user_agent="park_locator", timeout=10)

# Function to get latitude and longitude
def get_lat_lon(park_name):
    try:
        location = geolocator.geocode(park_name)
        if location:
            return location.latitude, location.longitude
        else:
            return None, None
    except Exception as e:
        print(f"Error with {park_name}: {e}")
        return None, None
```

In [ ]:
```
# Apply function to the 'Park Name' column
df[['Latitude', 'Longitude']] = df['Park Name'].apply(lambda x: pd.Series(get_

# Pause for 1 second between requests to avoid overloading the geocoding servi
time.sleep(1)
```

In [ ]:
```
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total	Latitude	Longitude
0	Denali National Park	1811807	63.231662	-151.040555
1	Gates of the Arctic National Park	41254	67.750841	-153.247456
2	Glacier Bay National Park	2017020	58.814175	-136.872094
3	Katmai National Park	228113	58.504984	-155.089623
4	Kenai Fjords National Park	1661733	59.868563	-150.025771

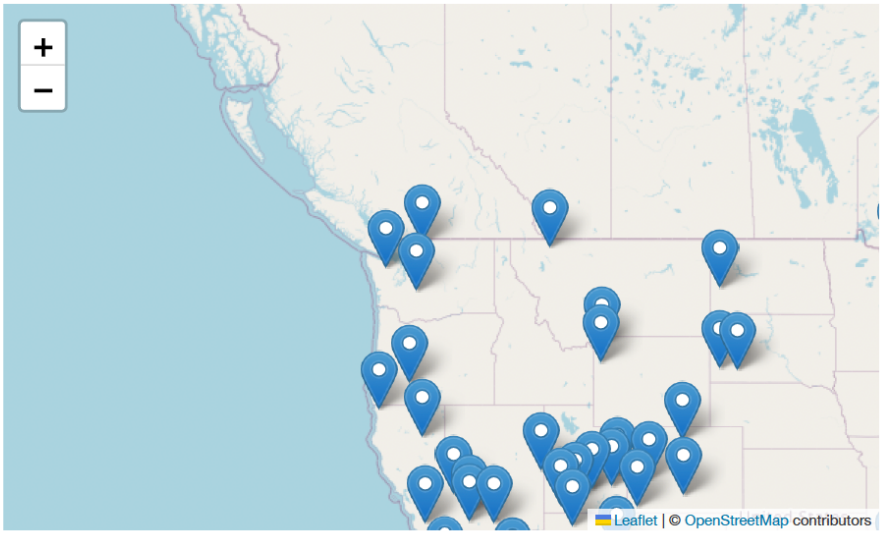
The geolocator has now pulled a latitude and longitude for each park. Just to double check, let's visualize these points on a map to ensure that there are no outliers (for instance, any

points outside of the United States).

In [ ]:
```
import folium
```

In [ ]:
```
# Create a base map centered on the United States
m = folium.Map(location=[37.0902, -95.7129], zoom_start=4)
# Create a base map centered on the United States (or a specific area)
m = folium.Map(location=[37.0902, -95.7129], zoom_start=4)

# Loop through the dataframe and add a marker for each park
for index, row in df.iterrows():
    lat = row['Latitude']
    lon = row['Longitude']
    park_name = row['Park Name']

    # Add a marker for each park if latitude and longitude are available
    if pd.notna(lat) and pd.notna(lon):
        folium.Marker([lat, lon], popup=park_name).add_to(m)

# Save the map to an HTML file to view in the browser
m.save('parks_map.html')

# Display the map in a Jupyter notebook
m
```

Out[ ]:



Exploring the map above shows us that the markers appear to be in the correct locations, meaning our latitude and longitudinal data is accurate. This is very important for the rest of the analysis since the project is based on proximity data.

Highway Proximity Data

Now that we have a clean dataset of National Park locations and visitation numbers, we are now going to obtain the distances from each park to a major highway using OpenStreetMap.

In [ ]:
```
import requests
import geopy.distance
```

In [ ]:
```
# Function to get the nearest major highway using OSM Overpass API, radius of 1
def get_nearest_highway(lat, lon):
    overpass_url = "http://overpass-api.de/api/interpreter"
    overpass_query = f"""
    [out:json];
    (
      way["highway"](around:160934, {lat}, {lon});
      node["highway"](around:160934, {lat}, {lon});
    );
    out body;
    """

    response = requests.get(overpass_url, params={'data': overpass_query})
    data = response.json()

    # Check if we have results and if the data structure contains the expected
    if 'elements' in data and len(data['elements']) > 0:
        nearest_highway = data['elements'][0]  # Get the first highway element
        # Ensure we're working with a node, not a way (nodes have lat/lon, ways
        if 'lat' in nearest_highway and 'lon' in nearest_highway:
            highway_lat = nearest_highway['lat']
            highway_lon = nearest_highway['lon']
            return highway_lat, highway_lon
        elif 'nodes' in nearest_highway:  # In case it's a 'way', retrieve a n
            node_id = nearest_highway['nodes'][0]  # Get the first node
            node_url = f"http://overpass-api.de/api/interpreter?data=[out:json
            node_response = requests.get(node_url)
            node_data = node_response.json()
            if 'elements' in node_data and len(node_data['elements']) > 0:
                node = node_data['elements'][0]
                return node['lat'], node['lon']

    return None, None  # Return None if no highway found or data is incomplete
```

In [ ]:
```
# Function to calculate distance between park and nearest highway
def calculate_distance_to_highway(park_lat, park_lon):
    highway_lat, highway_lon = get_nearest_highway(park_lat, park_lon)

    if highway_lat and highway_lon:
        # Calculate the distance using geopy
        park_coords = (park_lat, park_lon)
        highway_coords = (highway_lat, highway_lon)
        return geopy.distance.geodesic(park_coords, highway_coords).miles
    else:
        return None
```

In [ ]:
```
# Add distance to a major highway to the dataframe
df['Distance to Highway (miles)'] = df.apply(
    lambda row: calculate_distance_to_highway(row['Latitude'], row['Longitude']
)
```

```
# Verify the result by displaying the first few rows
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total	Latitude	Longitude	Distance to Highway (miles)
0	Denali National Park	1811807	63.231662	-151.040555	70.403985
1	Gates of the Arctic National Park	41254	67.750841	-153.247456	88.336116
2	Glacier Bay National Park	2017020	58.814175	-136.872094	90.158590
3	Katmai National Park	228113	58.504984	-155.089623	66.318978
4	Kenai Fjords National Park	1661733	59.868563	-150.025771	90.980396

We utilized a 100 mile search radius for the nearest highway due to the large size of the National Parks. After running the functions to obtain this data from OpenStreetMap, there is a new column in the dataset showing the distance to the nearest highway in miles for each park.

Airport Proximity Data

Now we will utilize the Google Places API to calculate the distance from each park to an airport.

In [ ]:
```
import googlemaps
from geopy.distance import geodesic
```

In [ ]:
```
api_key = 'AIzaSyB_29pevIGtixOw3WfU1Av7-1lRzzR5SQk'
gmaps = googlemaps.Client(key=api_key)
```

In [ ]:
```
# Function to find the nearest airport using the Places API
def find_nearest_airport_distance(lat, lon, radius=50000):
    # Search for airports within a given radius (50 km in this case)
    places_result = gmaps.places_nearby(location=(lat, lon), radius=radius, ty

    # If airports are found, calculate the distance to the nearest one
    if places_result['results']:
        airport = places_result['results'][0]
        airport_lat = airport['geometry']['location']['lat']
        airport_lon = airport['geometry']['location']['lng']

        # Calculate the distance to the nearest airport using geodesic
        park_location = (lat, lon)
        airport_location = (airport_lat, airport_lon)
        distance = geodesic(park_location, airport_location).miles  # Distance

        return distance
    else:
        return None  # Return None if no airport is found within the radius
```

In [ ]:
```
# Loop through each park and calculate the distance to the nearest airport
for index, row in df.iterrows():
    lat = row['Latitude']  # Adjust to the actual column name for Latitude
    lon = row['Longitude']  # Adjust to the actual column name for Longitude

    # Calculate the distance to the nearest airport
    distance_to_airport = find_nearest_airport_distance(lat, lon)

    # Add the distance to the DataFrame
    df.at[index, 'Distance to Nearest Airport (miles)'] = distance_to_airport
```

In [ ]:
```
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total	Latitude	Longitude	Distance to Highway (miles)	Distance to Nearest Airport (miles)
0	Denali National Park	1811807	63.231662	-151.040555	70.403985	21.520031
1	Gates of the Arctic National Park	41254	67.750841	-153.247456	88.336116	NaN
2	Glacier Bay National Park	2017020	58.814175	-136.872094	90.158590	NaN
3	Katmai National Park	228113	58.504984	-155.089623	66.318978	NaN
4	Kenai Fjords National Park	1661733	59.868563	-150.025771	90.980396	27.854519

In [ ]:
```
# Count the NaN values in a specific column
nan_count = df['Distance to Nearest Airport (miles)'].isna().sum()
print(f"No airport within 50,000 meters: {nan_count}")
```

```
No airport within 50,000 meters: 8
```

Of note, the maximum radius for the Google Places API is 50,000 meters, or approximately 31 miles. There are eight National Parks with no airport within 31 miles. For all others, the distance in miles to the nearest airport is now added to the dataframe.

Number of Accommodations

We will utilize the Google Places API once more to determine the number of 'lodging' category locations there are within the maximum radius (50,000 meters) of each park. Instead of establishing the distance to nearest lodging, I will store the number of accoommdations found within the radius to explore how many options are available.

In [ ]:
```
# Function to find the number of lodging accommodations within a given radius
def find_lodging_count(lat, lon, radius=50000):
    # Search for lodging within a given radius (50 km in this case)
    places_result = gmaps.places_nearby(location=(lat, lon), radius=radius, ty

    # Return the number of lodging establishments found
```

```
    if places_result['results']:
        return len(places_result['results'])
    else:
        return 0  # Return 0 if no lodging is found within the radius
```

In [ ]:
```
# Loop through each park and calculate the number of lodging establishments wi
for index, row in df.iterrows():
    lat = row['Latitude']  # Adjust to the actual column name for Latitude
    lon = row['Longitude']  # Adjust to the actual column name for Longitude

    # Calculate the number of lodging accommodations
    lodging_count = find_lodging_count(lat, lon)

    # Add the lodging count to the DataFrame
    df.at[index, 'Number of Lodging Accommodations (within 31 miles)'] = lodgi
```

In [ ]:
```
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total	Latitude	Longitude	Distance to Highway (miles)	Distance to Nearest Airport (miles)	Number of Lodging Accommodations (within 31 miles)
0	Denali National Park	1811807	63.231662	-151.040555	70.403985	21.520031	6.0
1	Gates of the Arctic National Park	41254	67.750841	-153.247456	88.336116	NaN	0.0
2	Glacier Bay National Park	2017020	58.814175	-136.872094	90.158590	NaN	0.0
3	Katmai National Park	228113	58.504984	-155.089623	66.318978	NaN	3.0
4	Kenai Fjords National Park	1661733	59.868563	-150.025771	90.980396	27.854519	20.0

In [ ]:
```
# Convert the lodging count column to integers
df['Number of Lodging Accommodations (within 31 miles)'] = df['Number of Lodgi
df.head()
```

Out[ ]:

	Park Name	5 Year Visitation Total	Latitude	Longitude	Distance to Highway (miles)	Distance to Nearest Airport (miles)	Number of Lodging Accommodations (within 31 miles)
0	Denali National Park	1811807	63.231662	-151.040555	70.403985	21.520031	6
1	Gates of the Arctic National Park	41254	67.750841	-153.247456	88.336116	NaN	0
2	Glacier Bay National Park	2017020	58.814175	-136.872094	90.158590	NaN	0
3	Katmai National Park	228113	58.504984	-155.089623	66.318978	NaN	3
4	Kenai Fjords National Park	1661733	59.868563	-150.025771	90.980396	27.854519	20

In [ ]:
```
# Export the DataFrame to a CSV file
df.to_csv('/Users/amyaragon/Desktop/NPS Data/park_visitation_data.csv', index=
```

The complete, clean dataset can now be exported and used for analysis.

Appendix D: Code Used in Analysis

```
# First, we must load the cleaned dataset as a dataframe and begin exploring the data.

import pandas as pd
df = pd.read_csv('/Users/amyaragon/Desktop/NPS Data/park_visitation_data.csv')
df.head()

df.describe()

# We will use a heat map visualization to easily see where the missing values are within this dataset.

import seaborn as sns
import matplotlib.pyplot as plt

# Plot missing data heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Data Heatmap')
plt.show()

# The only category with missing values is the "Distance to Nearest Airport (miles)", meaning that there are National Parks in the dataset that do not have an airport within 31 miles. We will explore whether this impacts visitation numbers compared to parks that do have a nearby airport.

# Create a new column to label parks with missing airport data
df['Missing Airport Data'] = df['Distance to Nearest Airport (miles)'].isnull()

# Check if the new column is created correctly
df[['Park Name', 'Distance to Nearest Airport (miles)', 'Missing Airport Data']].head()

# Visitation comparison
# Boxplot comparing visitation between parks with and without airport data
plt.figure(figsize=(10, 6))
sns.boxplot(x='Missing Airport Data', y='5 Year Visitation Total', data=df)
plt.xticks([0, 1], ['Has Airport Data', 'No Airport Data'])  # Adjust labels for x-axis
plt.title('Comparison of 5-Year Visitation Total for Parks with and without Airport Data')
plt.xlabel('Airport Data Availability')
plt.ylabel('5-Year Visitation Total')
plt.show()

from scipy import stats

# Checking for a significant difference between both groups
has_airport_data = df[df['Missing Airport Data'] == False]['5 Year Visitation Total']
no_airport_data = df[df['Missing Airport Data'] == True]['5 Year Visitation Total']

# Independent t-test
t_stat, p_value = stats.ttest_ind(has_airport_data, no_airport_data, equal_var=False)
```

```
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

# The p-value is much smaller than 0.05, meaning that there is a statistically significant difference between visitation at parks with no nearby airport, and parks that do have an airport within 31 miles.

from scipy.stats import pearsonr

# Will not include the parks with missing airport data in the calculation
df_no_na = df.dropna(subset=['Distance to Nearest Airport (miles)'])
correlations = ['Distance to Highway (miles)', 'Distance to Nearest Airport (miles)', 'Number of Lodging Accommodations (within 31 miles)']

# Create an empty dictionary to store results
correlation_results = {}

for var in correlations:
    # Calculate the Pearson correlation coefficient and p-value
    corr, p_val = pearsonr(df_no_na[var], df_no_na['5 Year Visitation Total'])

    # Store the results in the dictionary
    correlation_results[var] = {'correlation': corr, 'p_value': p_val}

# Display the results
for var, result in correlation_results.items():
    print(f"{var}: Pearson correlation = {result['correlation']:.4f}, p-value = {result['p_value']:.4f}")


# For the p-values to show significance, they would need to be less than 0.05. Based on the Pearson correlation, none are showing statistical significance.

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import statsmodels.api as sm

# Define the independent variables (X) and dependent variable (y)
X = df_no_na[['Distance to Highway (miles)', 'Distance to Nearest Airport (miles)', 'Number of Lodging Accommodations (within 31 miles)']]
y = df_no_na['5 Year Visitation Total']

# Add constant to the model for intercept
X = sm.add_constant(X)

# Fit the model using statsmodels
model = sm.OLS(y, X).fit()

# Display the regression results
print(model.summary())
```

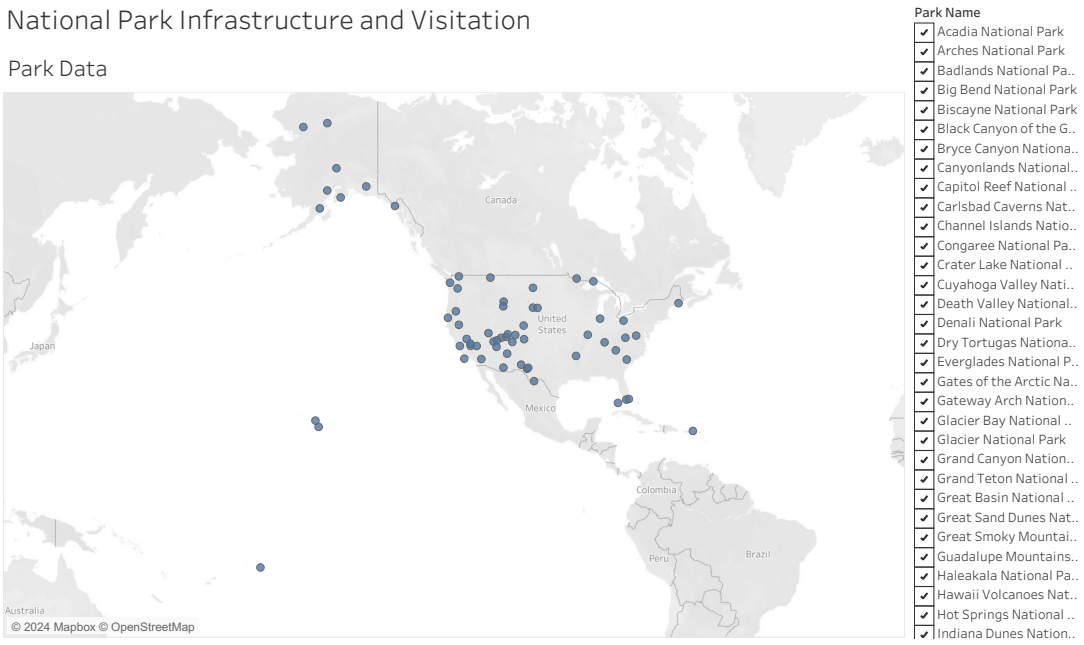
```
# Predict the values using the model
y_pred = model.predict(X)

# Create a subplot for each variable
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot for 'Distance to Highway (miles)'
axes[0].scatter(df_no_na['Distance to Highway (miles)'], y, color='blue', label='Actual Data', alpha=0.6)
axes[0].plot(df_no_na['Distance to Highway (miles)'], y_pred, color='red', label='Regression Line',
linewidth=2)
axes[0].set_xlabel('Distance to Highway (miles)')
axes[0].set_ylabel('5 Year Visitation Total')
axes[0].set_title('Distance to Highway vs Visitation')
axes[0].legend()

# Plot for 'Distance to Nearest Airport (miles)'
axes[1].scatter(df_no_na['Distance to Nearest Airport (miles)'], y, color='blue', label='Actual Data',
alpha=0.6)
axes[1].plot(df_no_na['Distance to Nearest Airport (miles)'], y_pred, color='red', label='Regression Line',
linewidth=2)
axes[1].set_xlabel('Distance to Nearest Airport (miles)')
axes[1].set_ylabel('5 Year Visitation Total')
axes[1].set_title('Distance to Airport vs Visitation')
axes[1].legend()

# Plot for 'Number of Lodging Accommodations (within 31 miles)'
axes[2].scatter(df_no_na['Number of Lodging Accommodations (within 31 miles)'], y, color='blue',
label='Actual Data', alpha=0.6)
axes[2].plot(df_no_na['Number of Lodging Accommodations (within 31 miles)'], y_pred, color='red',
label='Regression Line', linewidth=2)
axes[2].set_xlabel('Number of Lodging Accommodations (within 31 miles)')
axes[2].set_ylabel('5 Year Visitation Total')
axes[2].set_title('Lodging vs Visitation')
axes[2].legend()

plt.tight_layout()

# Show plot
plt.show()

# No linear relationships are found with these variables and visitation numbers. Let's calculate a z-score
for the dataset to determine if there are outliers, as this may be skewing the results.

# %%
# Calculate the Z-scores for the '5 Year Visitation Total'
df_no_na['Z-Score'] = (df_no_na['5 Year Visitation Total'] - df_no_na['5 Year Visitation Total'].mean()) /
df_no_na['5 Year Visitation Total'].std()

# Identify the park(s) with Z-scores greater than 3 or less than -3
outliers = df_no_na[df_no_na['Z-Score'].abs() > 3]
print(outliers)
```

```
# Now, we will remove the outlier and re-run the model to see if this significantly impacts the results.

df_no_outlier = df_no_na[df_no_na['Park Name'] != 'Great Smoky Mountains National Park']

# Define the independent variables (X) and dependent variable (y)
X = df_no_outlier[['Distance to Highway (miles)', 'Distance to Nearest Airport (miles)', 'Number of
Lodging Accommodations (within 31 miles)']]
y = df_no_outlier['5 Year Visitation Total']

# Add constant to the model for intercept
X = sm.add_constant(X)

# Fit the model using statsmodels
model = sm.OLS(y, X).fit()

# Display the regression results
print(model.summary())

# Predict the values using the model
y_pred = model.predict(X)

# Create a subplot for each variable
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot for 'Distance to Highway (miles)'
axes[0].scatter(df_no_outlier['Distance to Highway (miles)'], y, color='blue', label='Actual Data',
alpha=0.6)
axes[0].plot(df_no_outlier['Distance to Highway (miles)'], y_pred, color='red', label='Regression Line',
linewidth=2)
axes[0].set_xlabel('Distance to Highway (miles)')
axes[0].set_ylabel('5 Year Visitation Total')
axes[0].set_title('Distance to Highway vs Visitation')
axes[0].legend()

# Plot for 'Distance to Nearest Airport (miles)'
axes[1].scatter(df_no_outlier['Distance to Nearest Airport (miles)'], y, color='blue', label='Actual Data',
alpha=0.6)
axes[1].plot(df_no_outlier['Distance to Nearest Airport (miles)'], y_pred, color='red', label='Regression
Line', linewidth=2)
axes[1].set_xlabel('Distance to Nearest Airport (miles)')
axes[1].set_ylabel('5 Year Visitation Total')
axes[1].set_title('Distance to Airport vs Visitation')
axes[1].legend()

# Plot for 'Number of Lodging Accommodations (within 31 miles)'
axes[2].scatter(df_no_outlier['Number of Lodging Accommodations (within 31 miles)'], y, color='blue',
label='Actual Data', alpha=0.6)
axes[2].plot(df_no_outlier['Number of Lodging Accommodations (within 31 miles)'], y_pred, color='red',
label='Regression Line', linewidth=2)
axes[2].set_xlabel('Number of Lodging Accommodations (within 31 miles)')
```

```
axes[2].set_ylabel('5 Year Visitation Total')
axes[2].set_title('Lodging vs Visitation')
axes[2].legend()

plt.tight_layout()

# Show plot
plt.show()

# To double check if multicollinearity is impacting the results, we will calculate the variance inflation factor.

from statsmodels.stats.outliers_influence import variance_inflation_factor

# Calculate VIF for each independent variable
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif_data)

# Multicollinearity is not in issues based on the VIF results. The VIF for all factors is close to 1, so there is no significant correlation between the predictor variables that would cause multicollinearity.
```

Appendix E: Tableau Visualization

National Park Infrastructure Map



Dashboard includes visitation numbers and all proximity metrics on hover, can be filtered by park name.