DTSA 5509 Final Project

Predicting Walkability of U.S. Counties

Rahul Cheeniyil

12-11-2024

In this project, I use the Walkability Index dataset (published by the U.S. Environmental Protection Agency) to build and test supervised learning models to predict walkability of U.S. Census Blocks. Predicting walkability is important for allocating infrastructure funding for areas with low pedestrian accessibility. Improving city walkability can promote for public health and safety as well as reduce automobile traffic and improve environmental health due to reduced emissions.

The models I will use are multilinear regression and gradient boosting. In this report, I will walk through the data I use, how I intend to use it, data cleaning, and exploratory data analysis prior to building the models themselves. Model performance will be assessed at the end of the report.

Data

The data consists of various metrics from Censis 2019 block groups. The data is accessed by a publicly available API. The code cell below loads the data via REST API requests and loads it into a Pandas DataFrame. The resulting DataFrame has 220,134 rows and 182 columns. Each row corresponds to a U.S. Census Block and each column responds to a metric associated with that block. The column "NatWalkInd" is the National Walkability Index score assigned to the block. This is the metric I will be predicting with machine learning models. All other columns reflect various metrics/statistics associated with the block. These 181 columns can be grouped into several categories.

- Geographic Identifiers: Spatial and administrative context for identifying the blocks
- Population and Demographics: Population of the block and various demographic distributions.
- Automobile Ownership Statistics: Car ownership/dependency of residents including percentage of households owning different numbers of vehicles.
- Employment and Wages: Job distributions and wage levels.
- Land Use and Density: Metrics related to the land itself.
- Transportation: Proximity to destinations and transit.
- Environmental Metrics: Assessments of emissions and vehicle usage.
- Geographic Measurements: Geometric characterization of the land.

In [2]:
```
import requests
import pandas as pd

# Publicly accessibly Esri REST API Endpoint
api_url = "https://geodata.epa.gov/arcgis/rest/services/OA/WalkabilityIndex/MapServer/0/query"

params = {
    "where": "1=1",  # Select all records
    "outFields": "*",  # Retrieve all fields
    "f": "json",  # Response format
    "resultRecordCount": 10000,  # Limit to 10000 records per request to avoid rate limits
    "returnGeometry": True,  # Include geographic data such as area/perimeter of census blocks
}

# Fetch the data in chunks
def fetch_data(api_url, params):
    offset = 0
    all_data = []

    while True:
        params["resultOffset"] = offset
        response = requests.get(api_url, params=params)

        if response.status_code == 200:
            data = response.json()
            features = data.get("features", [])

            if not features:
                break  # No more records

            all_data.extend([feature["attributes"] for feature in features])

            offset += len(features)
        else:
            print(f"Error: {response.status_code}")
            break

    return pd.DataFrame(all_data)

walkability_data = fetch_data(api_url, params)  # This took me about 7 minutes to run.

print(walkability_data.info)
```

```
# Report memory usage in MB
memory_B = walkability_data.memory_usage(deep=True).sum()
memory_MB = round(memory_B/1024/1024, 3)
print(f"The Complete DataFrame size is {memory_MB:,} MB.")
```

```
<bound method DataFrame.info of              GEOID10       GEOID20 STATEFP COUNTYFP TRACTCE BLKGRPCE   CSA
0       481130078254  481130078254      48      113  007825        4   206  \
1       481130078252  481130078252      48      113  007825        2   206
2       481130078253  481130078253      48      113  007825        3   206
3       481130078241  481130078241      48      113  007824        1   206
4       481130078242  481130078242      48      113  007824        2   206
...              ...           ...     ...      ...     ...      ...   ...
220129  780309610002  780309610002      78      030  961000        2  None
220130  780309610003  780309610003      78      030  961000        3  None
220131  780309610005  780309610005      78      030  961000        5  None
220132  780309607003  780309607003      78      030  960700        3  None
220133  780309602002  780309602002      78      030  960200        2  None

                       CSA_Name   CBSA                          CBSA_Name  ...
0       Dallas-Fort Worth, TX-OK  19100  Dallas-Fort Worth-Arlington, TX  ...  \
1       Dallas-Fort Worth, TX-OK  19100  Dallas-Fort Worth-Arlington, TX  ...
2       Dallas-Fort Worth, TX-OK  19100  Dallas-Fort Worth-Arlington, TX  ...
3       Dallas-Fort Worth, TX-OK  19100  Dallas-Fort Worth-Arlington, TX  ...
4       Dallas-Fort Worth, TX-OK  19100  Dallas-Fort Worth-Arlington, TX  ...
...                          ...    ...                              ...  ...
220129                      None                                   None  ...
220130                      None                                   None  ...
220131                      None                                   None  ...
220132                      None                                   None  ...
220133                      None                                   None  ...

        VMT_per_worker  VMT_tot_min  VMT_tot_max  VMT_tot_avg  GHG_per_worker
0            27.496405    11.442995    82.636303    25.659327       24.499297  \
1            26.465754    11.442995    82.636303    25.659327       23.580987
2            32.311348    11.442995    82.636303    25.659327       28.789412
3            26.277851    11.442995    82.636303    25.659327       23.413565
4            30.138550    11.442995    82.636303    25.659327       26.853448
...                ...          ...          ...          ...             ...
220129             NaN          NaN          NaN          NaN             NaN
220130             NaN          NaN          NaN          NaN             NaN
220131             NaN          NaN          NaN          NaN             NaN
220132             NaN          NaN          NaN          NaN             NaN
220133             NaN          NaN          NaN          NaN             NaN

         Annual_GHG  Shape_Length    Shape_Area  OBJECTID  SLC_score
0       6369.817221   3703.093137  4.242040e+05         1  77.450956
1       6131.056669   4183.883949  6.909034e+05         2  78.898635
2       7485.246991   2013.474140  1.527511e+05         3  70.687759
3       6087.527012   3481.320509  6.878808e+05         4  79.162569
4       6981.896477   4426.401418  9.776380e+05         5  73.739730
...             ...           ...           ...       ...        ...
220129          NaN   3529.071724  3.744628e+05    220735        NaN
220130          NaN   2567.207401  3.258587e+05    220736        NaN
220131          NaN   2063.598299  1.809511e+05    220737        NaN
220132          NaN  17692.559655  1.159002e+07    220738        NaN
220133          NaN   4071.307514  6.798168e+05    220739        NaN

[220134 rows x 182 columns]>
The Complete DataFrame size is 440.863 MB.
```

Data Cleaning

Upon examining the column data, it's fairly clear that not all are needed to build a model. Primary features of interest are going to be quantifiable, meaning we can drop all of our non-numeric datatype columns. Additionally, examining our column groupings, we likely don't need any administrative information or other identifiers, so it's safe to drop those columns. I also drop redudant columns that effecively show duplicate data that is presented differently. This could bias the models to more strongly depends on duplicated data. We can also discard any columns providing a "score" other than our target dependent variable "NatWalkInd." Scores are results from assessments and are not realistic to include in a model that is meant to infer walkability based on measurable metrics. From the remaining dataframe, I drop any rows with None or NaN values to ensure useability of data. After these steps, the data should be clean and usable for building a model. Many columns still remain, but those will be filtered down further after investigating which features best suit my models through EDA.

Remaining data size is 217,181 rows and 146 columns taking up 243.574 MB in memory.

In [3]:
```
# Remove columns with non-numeric values
numeric_walkability_data = walkability_data.select_dtypes(include=['number'])

# Manually drop columns with data irrelevant to model training
columns_to_drop = [
    "OBJECTID",  # Unique object identifier
    "Shape_Length",  # Metadata about polygons for mapping
    "Shape_Area",
    "Households",  # Redundant general counts already covered by more detailed columns
    "Residents",
    "Workers_1",
```

```
    "Drivers",
    "Vehicles",
    "SLC_score",  # Scores that are higher-level aggregations - not independent variables
    "D2A_Ranked",
    "D2B_Ranked",
    "D3B_Ranked",
    "D4A_Ranked",
    "B_C_*",  # Coefficients derived from statistical or predictive models of the population
    "B_N_*",
    "C_R_*",
    "GHG_per_worker", # Metrics focused on vehicle usage/emissions, likely resulting from vehicle ownership
    "Annual_GHG",
    "NonCom_VMT_Per_Worker",
    "Com_VMT_Per_Worker",
    "VMT_per_worker",
    "logd1a",  # Log/Scaled transformations of existing variables
    "logd1c",
    "logd3aao",
    "logd3apo",
    "logd4d",
    "d4bo25",
    "d5dei_1"
]

filtered_walkability_data = numeric_walkability_data.drop(columns=columns_to_drop, errors='ignore')

# Filter out rows with missing data
cleaned_walkability_data = filtered_walkability_data.dropna()

print(cleaned_walkability_data.info)

memory_B = cleaned_walkability_data.memory_usage(deep=True).sum()
memory_MB = round(memory_B/1024/1024, 3)
print(f"The Cleaned DataFrame size is {memory_MB:,} MB.")

del walkability_data, numeric_walkability_data, filtered_walkability_data  # Free up memory taken by large intermedia
```

```
<bound method DataFrame.info of         CBSA_POP  CBSA_EMP  CBSA_WRK       Ac_Total    Ac_Water
0        7189384   3545715   3364458      73.595028    0.000000  \
1        7189384   3545715   3364458     119.829909    0.000000
2        7189384   3545715   3364458      26.367053    0.000000
3        7189384   3545715   3364458     119.060687    0.000000
4        7189384   3545715   3364458     169.927211    0.000000
...          ...       ...       ...            ...         ...
217177     80610     39570     37773     193.420737    0.000000
217178     80610     39570     37773     495.303549   44.102902
217179     80610     39570     37773     126.513965    0.000000
217180         0         0         0  367912.651157  810.976517
217181         0         0         0  914821.703629  423.499748

             Ac_Land        Ac_Unpr  TotPop  CountHU     HH  ...
0          73.595028      73.595028    1202    460.0  423.0  ...  \
1         119.829909     119.214200     710    409.0  409.0  ...
2          26.367053      26.367050     737    365.0  329.0  ...
3         119.060687     119.060687     904    384.0  384.0  ...
4         169.927211     148.742920     948    343.0  343.0  ...
...              ...            ...     ...      ...    ...  ...
217177    193.420737     185.670240    1251    588.0  557.0  ...
217178    451.200646     376.627030    1292    501.0  473.0  ...
217179    126.513965     123.144540    1055    456.0  456.0  ...
217180  367101.674640  221749.898140     833    509.0  425.0  ...
217181  914398.203881  516485.052320     970    522.0  448.0  ...

        C_R_Vehicles  C_R_White  C_R_Male  C_R_Lowwage  C_R_Medwage
0          1394052.0   0.291340  0.493078     0.213768     0.339558  \
1          1394052.0   0.291340  0.493078     0.213768     0.339558
2          1394052.0   0.291340  0.493078     0.213768     0.339558
3          1394052.0   0.291340  0.493078     0.213768     0.339558
4          1394052.0   0.291340  0.493078     0.213768     0.339558
...              ...        ...       ...          ...          ...
217177       63039.0   0.866892  0.502185     0.225690     0.327138
217178       63039.0   0.866892  0.502185     0.225690     0.327138
217179       63039.0   0.866892  0.502185     0.225690     0.327138
217180        4240.0   0.925765  0.501845     0.276741     0.380064
217181        4240.0   0.925765  0.501845     0.276741     0.380064

        C_R_Highwage  C_R_DrmV  VMT_tot_min  VMT_tot_max  VMT_tot_avg
0           0.446673  0.393452    11.442995    82.636303    25.659327
1           0.446673  0.393452    11.442995    82.636303    25.659327
2           0.446673  0.393452    11.442995    82.636303    25.659327
3           0.446673  0.393452    11.442995    82.636303    25.659327
4           0.446673  0.393452    11.442995    82.636303    25.659327
...              ...       ...          ...          ...          ...
217177      0.447171 -0.245448    15.631580    25.712336    19.996500
217178      0.447171 -0.245448    15.631580    25.712336    19.996500
217179      0.447171 -0.245448    15.631580    25.712336    19.996500
217180      0.343195 -0.475807    18.353785    28.388537    21.724116
217181      0.343195 -0.475807    18.353785    28.388537    21.724116

[217182 rows x 146 columns]>
The Cleaned DataFrame size is 243.574 MB.
```

Exploratory Data Analysis (EDA)

Plan

The data is now ready for exploratory data analysis (EDA). The goal here is to identify a set of predictors that could have value as features in the machine learning models. First steps here is to investigate some common visualizations such as KDE plots to check for variable distributions and correlation meatmaps to get a bearing of predictor quality.

Discussion

Initial visualizatiosn and assessment of feature quality was done by examining correlation with the dependent variable NatWalkInd and checking for quality of the predictors. I set a correlation threshold of 0.5 in either positive or negative direction to qualify features. This filter left us with 13 columns which is a pretty big reduction from the 146 after data cleaning. Further visualization with the correlation matrix shows that a couple sets of predictors are perfectly correlated and would be redundant to include in a model.

The features D5DRI, D5DEI, D5DR, D5DE, and D4A all reflect highly similar measures of destination accessibility quantifications, so excluding all but D4A is a reasonable choice. D4D, D4C, and D4E also reflect highly similar active transporation metrics so I also exlude all but D4C. Since the columns I eliminate appear to be perfectly redundant, I didn't have a reason for keeping the columns that I did other than that they are alphabetically first. Removing redundant features leaves us with 7 remaining predictors that do not appear to have any codependence.

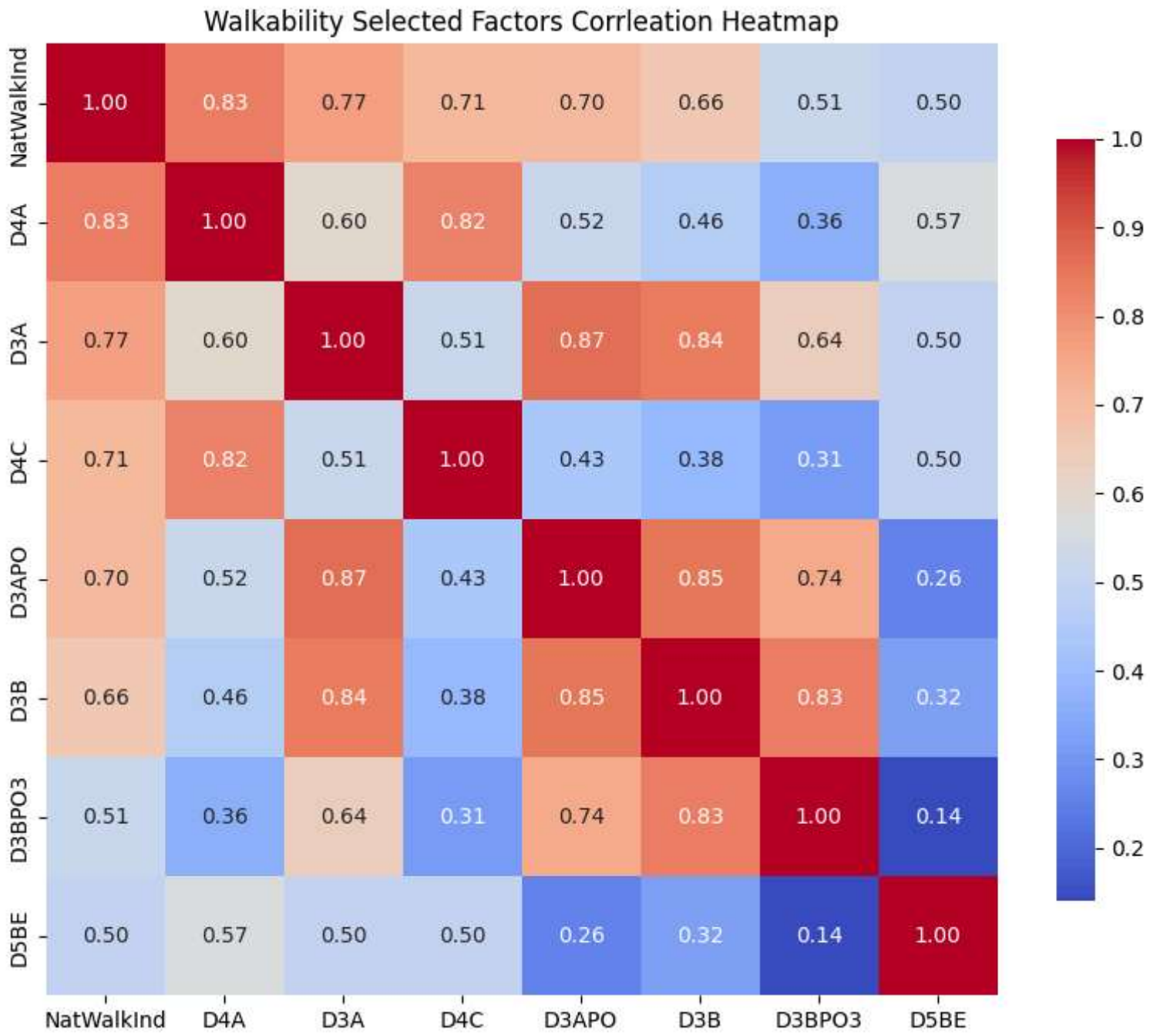
The KDE plot does show certain features to be highly bimodal and also shows some skewed distributions. This indicates that a non-linear model such as a Random Forest or Gradient Boosting will be able to better predict outcomes than a simple regression model.

After visualizations, I ran a check for multicollinearity and found that D3A and D3APO were highly collinear, so I decided to drop D3APO as D3A has a higher correlation with NatWalkInd. This leaves us with a final dataset for model building of 7 rows (6 predictors) and 217,182 rows.

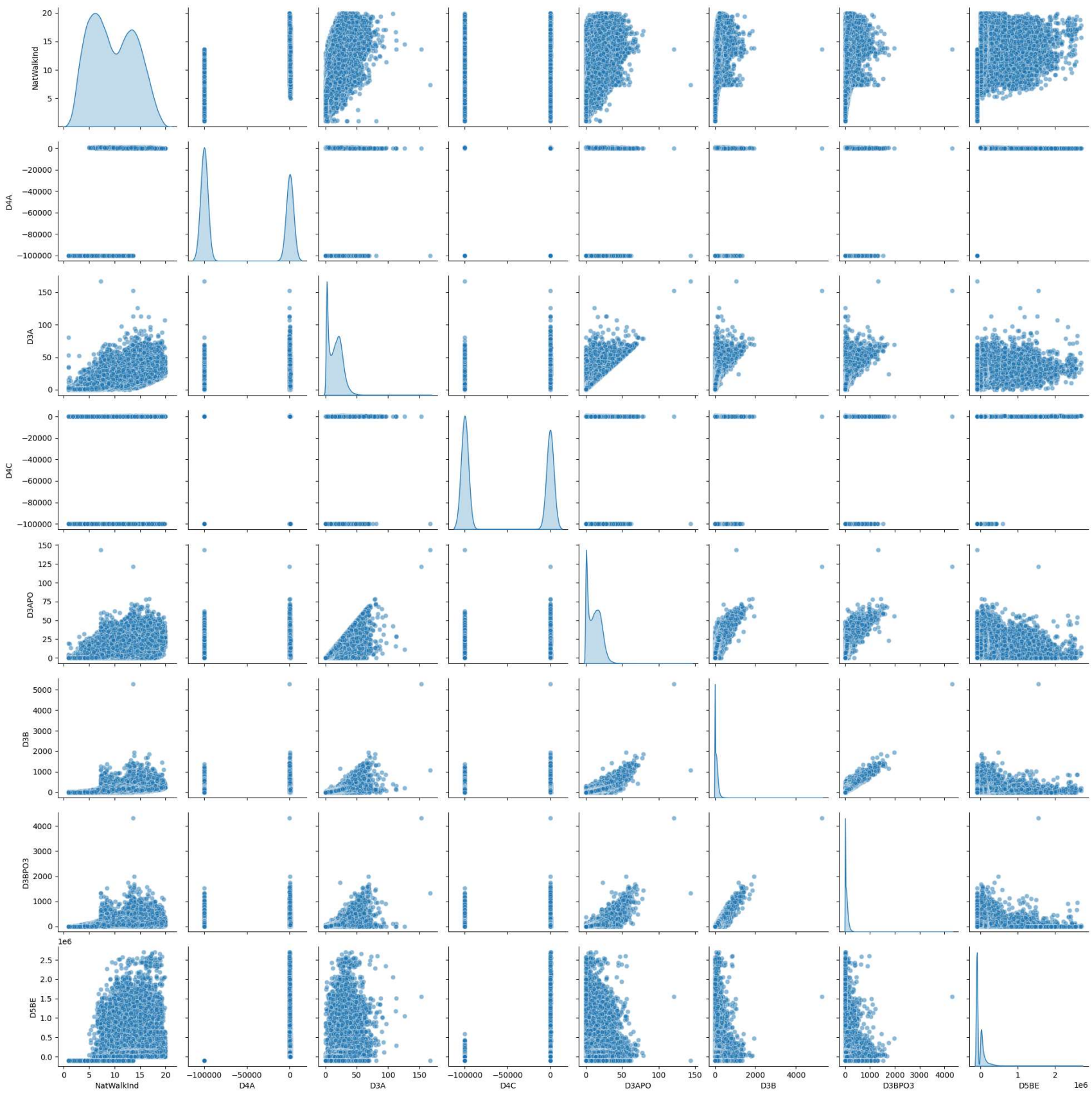
In [21]:
```
import seaborn as sns
from matplotlib import pyplot as plt
pd.set_option('display.max_columns', 999)  # Increase max number of columns/rows shown to increase readability of pri
pd.set_option('display.max_rows', 999)

# Specifically print correlations with our target dependent variable
corr_matrix = cleaned_walkability_data.corr()
natwalkind_corr = corr_matrix["NatWalkInd"].sort_values(ascending=False)

# Want to examine factors with a "strong" correlation, we'll use the threshold of 0.5 here in either the positive or
threshold = 0.5
strong_corr = natwalkind_corr[abs(natwalkind_corr) >= threshold]
print(strong_corr)

# Save strongly correlated features
strong_cols = strong_corr.index.to_list()

# Need to drop some features because of redundancy
features_to_drop = ["D5DRI", "D5DEI", "D5DR", "D5DE", "D4D", "D4E"]
strong_cols = [col for col in strong_cols if col not in features_to_drop]
features_df = cleaned_walkability_data[strong_cols]
features_corr_matrix = features_df.corr()

# Heatmap to visualize
plt.figure(figsize=(10, 8))
sns.heatmap(features_corr_matrix,
            annot=True,
            cmap='coolwarm',
            fmt='.2f',
            square=True,
            cbar_kws={'shrink': 0.8})
plt.title('Walkability Selected Factors Corrleation Heatmap')

# KDE plot to check selected variable distributions
plt.figure(figsize=(12, 12))
sns.pairplot(features_df,
             diag_kind='kde',
             plot_kws={'alpha': 0.5},
             diag_kws={'fill': True})
plt.suptitle("KDE Matrix Plot of Strongly Correlated Feeatures",
             y=1.02,
             fontsize=16)
```

```
NatWalkInd    1.000000
D5DRI         0.833210
D5DEI         0.833210
D5DR          0.833209
D5DE          0.833209
D4A           0.832624
D3A           0.768176
D4D           0.709511
D4C           0.709149
D4E           0.709107
D3APO         0.699990
D3B           0.662435
D3BPO3        0.512583
D5BE          0.502522
Name: NatWalkInd, dtype: float64
```

Out[21]: Text(0.5, 1.02, 'KDE Matrix Plot of Strongly Correlated Feeatures')



<Figure size 1200x1200 with 0 Axes>



In [27]:
```python
# Need to check for multicollinearity by getting VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

predictors = features_df.drop(columns=["NatWalkInd"])

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = predictors.columns
vif_data["VIF"] = [variance_inflation_factor(predictors.values, i) for i in range(predictors.shape[1])]

print(vif_data)

# Drop an additional column due to high VIF
predictors = predictors.drop(columns=["D3APO"])

# Recalculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = predictors.columns
vif_data["VIF"] = [variance_inflation_factor(predictors.values, i) for i in range(predictors.shape[1])]

print()  # print an extra line break for readability
print(vif_data)

final_features_df = features_df.drop(columns=["D3APO"])
```

	Feature	VIF
0	D4A	7.061964
1	D3A	17.950515
2	D4C	6.603426
3	D3APO	17.361835
4	D3B	12.314492
5	D3BPO3	6.095355
6	D5BE	1.925738

	Feature	VIF
0	D4A	7.014708
1	D3A	7.775423
2	D4C	6.595187
3	D3B	11.939816
4	D3BPO3	5.820051
5	D5BE	1.685860

Machine Learning Models

Model selection

I want to use both Gradient Boosting and Linear Regression to predict walkability of cities using my cleaned and filtered data.

- Gradient Boosting
  - Handles nonlinear relationships that are likely present in my data given the observation of bimodal predictor variables.
  - Can identify which features contribute most to prediction.
  - Can outperform simple models when relationships are more complex which is likely the case in my data.
- Linear Regression
  - Very simple model to train and understand outputs of. Highly interpretable.
  - Serves as an intuitive baseline to compare to more complex models like gradient boosting
  - Very efficient to build model as high compute power and long training time not required.

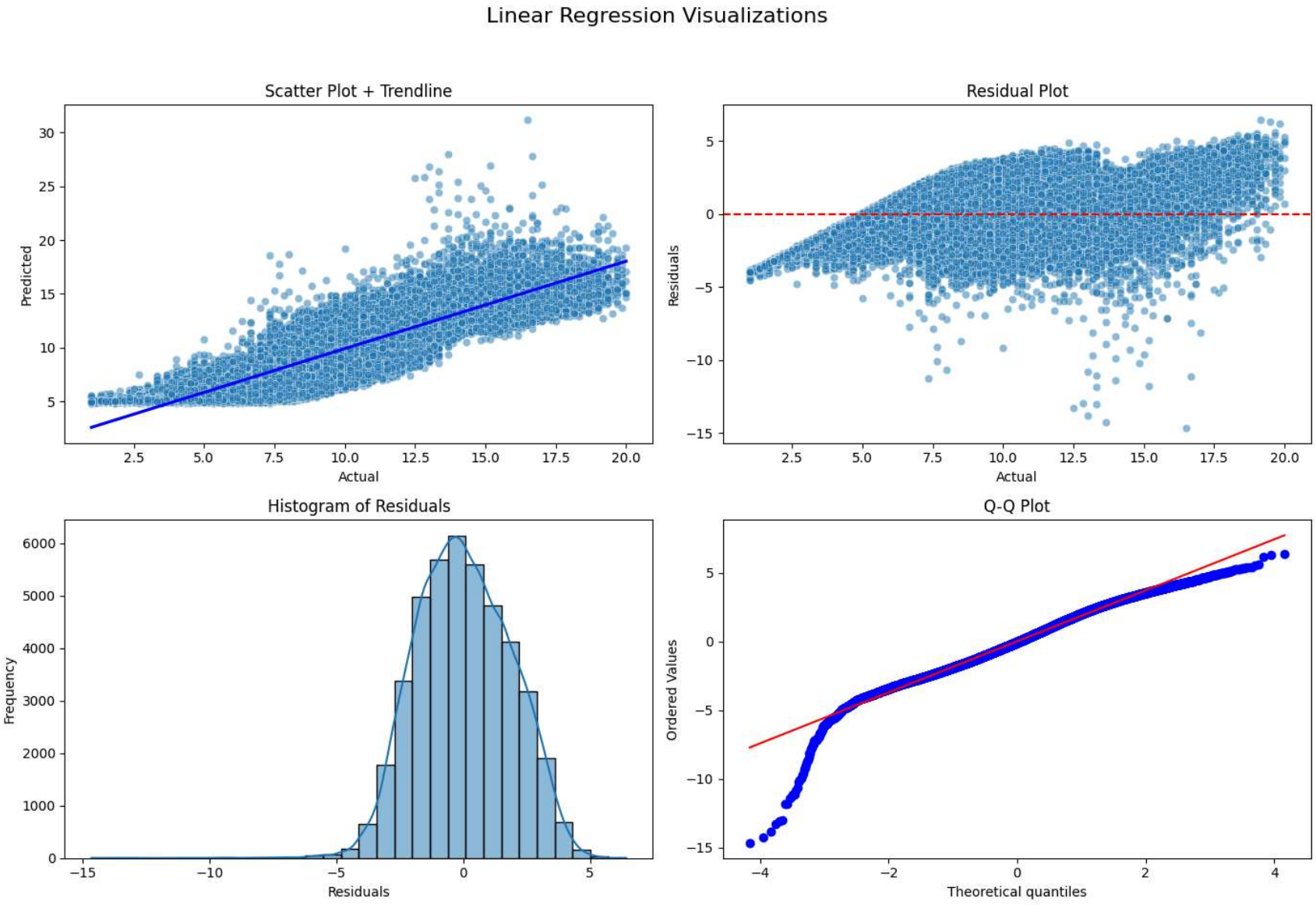
Plan for model building

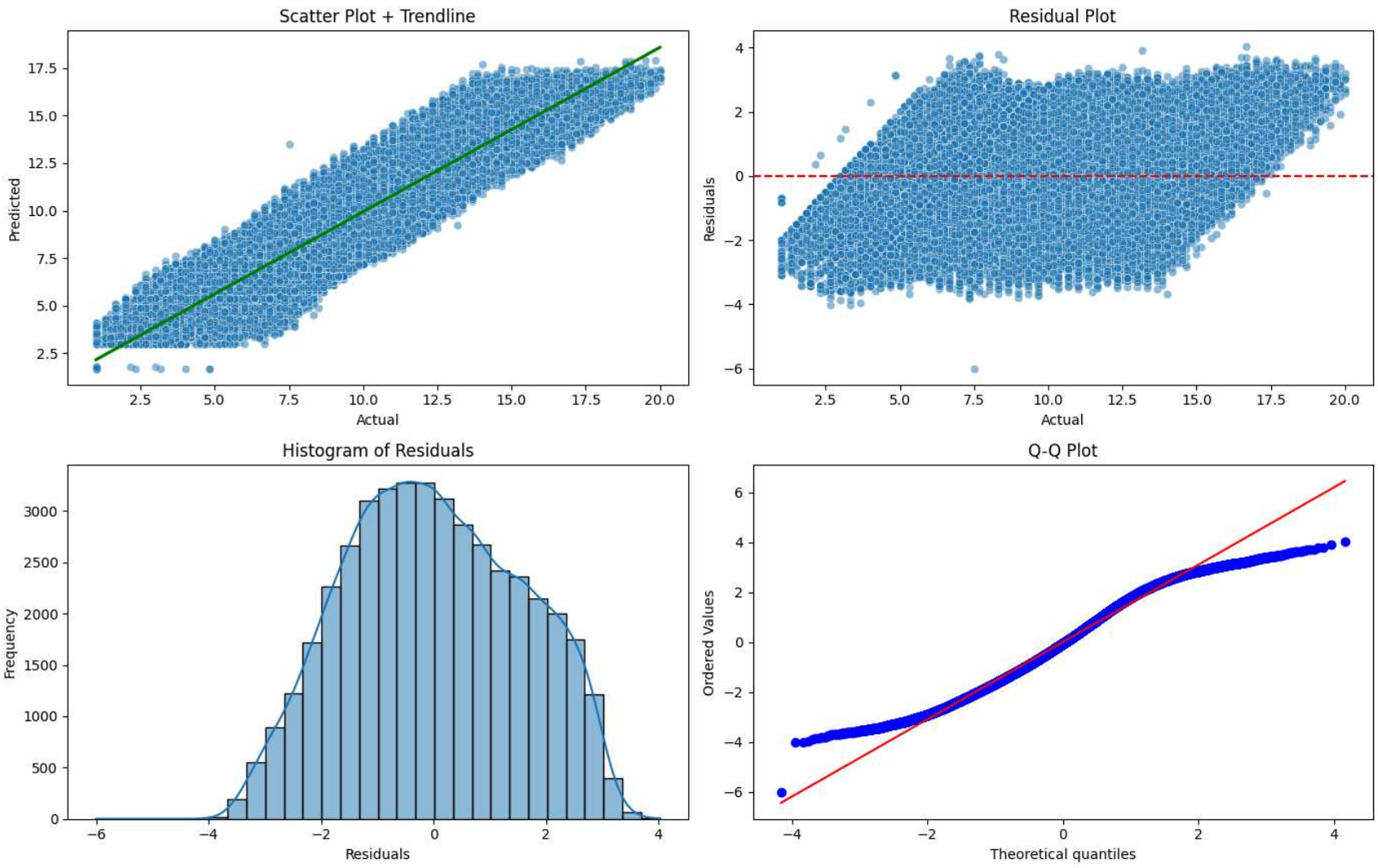
I plan to use an 80-20 train-test split across the ~200k rows of my data. I'll use this split to train both a simple linear regression model and a gradient boosting model and compare results. I will compare performance by using metrics such as R^2, RMSE, and MAE to evaluate the models.

In [31]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

random_state = 81798  # I like to use my birthday as random state

# Splitting dependent variable and predictors
X = final_features_df.drop(columns=["NatWalkInd"])
y = final_features_df["NatWalkInd"]

# Splitting train and test groups
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=random_state)

# Training the linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_linear = linear_model.predict(X_test)

# Training the gradient boosting model
gb_model = GradientBoostingRegressor(random_state=random_state)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)

# Model evaluation
metrics = {
    "Model": ["Linear Regression", "Gradient Boosting"],
    "R^2 Score": [
        r2_score(y_test, y_pred_linear),
        r2_score(y_test, y_pred_gb)
    ],
    "RMSE": [
        np.sqrt(mean_squared_error(y_test, y_pred_linear)),
        np.sqrt(mean_squared_error(y_test, y_pred_gb))
    ],
    "MAE": [
        mean_absolute_error(y_test, y_pred_linear),
        mean_absolute_error(y_test, y_pred_gb)
    ]
}

results_df = pd.DataFrame(metrics)

print(results_df.to_markdown(index=False, floatfmt=".4f"))
```

Model	R^2 Score	RMSE	MAE
Linear Regression	0.8162	1.8655	1.5184
Gradient Boosting	0.8717	1.5587	1.3009

In [34]:
```
from scipy import stats

# Visualzations of performance
# Residuals for both models
residuals_linear = y_test - y_pred_linear
residuals_gb = y_test - y_pred_gb

# Create a 2x2 grid for Linear Regression
fig, axs = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle("Linear Regression Visualizations", fontsize=16)

# Scatter Plot + Trendline
sns.scatterplot(x=y_test, y=y_pred_linear, alpha=0.5, ax=axs[0, 0])
sns.regplot(x=y_test, y=y_pred_linear, scatter=False, color="blue", ax=axs[0, 0])
axs[0, 0].set_title("Scatter Plot + Trendline")
axs[0, 0].set_xlabel("Actual")
axs[0, 0].set_ylabel("Predicted")

# Residual Plot
sns.scatterplot(x=y_test, y=residuals_linear, alpha=0.5, ax=axs[0, 1])
axs[0, 1].axhline(0, color='red', linestyle='--')
axs[0, 1].set_title("Residual Plot")
axs[0, 1].set_xlabel("Actual")
axs[0, 1].set_ylabel("Residuals")

# Histogram of Residuals
sns.histplot(residuals_linear, kde=True, bins=30, ax=axs[1, 0])
axs[1, 0].set_title("Histogram of Residuals")
axs[1, 0].set_xlabel("Residuals")
axs[1, 0].set_ylabel("Frequency")

# Q-Q Plot
stats.probplot(residuals_linear, dist="norm", plot=axs[1, 1])
axs[1, 1].set_title("Q-Q Plot")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

# Create a 2x2 grid for Gradient Boosting
fig, axs = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle("Gradient Boosting Visualizations", fontsize=16)

# Scatter Plot + Trendline
sns.scatterplot(x=y_test, y=y_pred_gb, alpha=0.5, ax=axs[0, 0])
sns.regplot(x=y_test, y=y_pred_gb, scatter=False, color="green", ax=axs[0, 0])
axs[0, 0].set_title("Scatter Plot + Trendline")
axs[0, 0].set_xlabel("Actual")
axs[0, 0].set_ylabel("Predicted")

# Residual Plot
sns.scatterplot(x=y_test, y=residuals_gb, alpha=0.5, ax=axs[0, 1])
axs[0, 1].axhline(0, color='red', linestyle='--')
axs[0, 1].set_title("Residual Plot")
axs[0, 1].set_xlabel("Actual")
axs[0, 1].set_ylabel("Residuals")

# Histogram of Residuals
sns.histplot(residuals_gb, kde=True, bins=30, ax=axs[1, 0])
axs[1, 0].set_title("Histogram of Residuals")
axs[1, 0].set_xlabel("Residuals")
axs[1, 0].set_ylabel("Frequency")

# Q-Q Plot
stats.probplot(residuals_gb, dist="norm", plot=axs[1, 1])
axs[1, 1].set_title("Q-Q Plot")

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```





Results and Analysis

Here is the table of results from model evaluation performed above

Model	R^2 Score	RMSE	MAE
Linear Regression	0.8162	1.8655	1.5184
Gradient Boosting	0.8717	1.5587	1.3009

R^2 Score

The R^2 score indicates the percentage of variance that is explained by the dependent variable NatWalkInd. Both models actually produce a good score, but gradient boosting seems to slightly outperform linear regression. The additional explained variance in gradient boosting may be due to capturing nonlinearity as predicted earlier during model selection. Additionally, the scatter plots comparing predicted and actual values show very few outliers meaning the models are farily consistent in predicting an accurate

score. As expected by the lower R^2, the linear regression model seems to predict slightly higher walkability than actual whereas the gradient boosting model is very evenly distributed across the trendline.

RMSE

The root mean squared error measures average magnitude of prediction error. Much like the R^2 score, the gradient boosting model outperorms the linear regression model with an RMSE reduction of about 16.5%. This indicates higher accuracy of the gradient boosting model.

MAE

Mean absolute error measures average prediction error. For both models, the similarity between MSE and RMSE indicates a low incidence of high-magnitude error. RMSE is a quadratic scoring that amplifies large error where MAE treats all error magnitudes equally. Because the scores are highly similar, it seems that both models do a good job of avoiding outliers.

Visualizations

The visualizations seem to confirm observations from the statistics. Clustering around the trendline is stronger for the gradient boosting model on the scatter plot. Residual plots show a slight degree of heteroscedasticity (variance increases as value increases). This indicates that both models struggle slightly with capturing variability for higher walkability values, though gradient boosting again performs better. The residual histogram is well distributed around 0 for both linear regression and gradient boosting models. The linear regression model actually looks a bit more normally distributed than the gradient boosting model. Overall, the range of the gradient boosting histogram is narrower which reflects the previously discussed higher precision. In the Q-Q plot, both perform well around the central region, but deviate towards the tails. Generally, both seem to be fairly well fit.

Summary of Findings

While both models perform well, linear regression is consistently outperformed by gradient boosting in capturing non-linear relationships and reducing error variance. Gradient boosting produces tighter residuals with a more normal distribution. Overall, gradient boosting is fairly clearly the superior model. The performance savings of linear regression aren't enough to make up for the loss in predictive value of the model.

Discussion and Potential Improvements

This project successfully built two machine learning models to predict walkability based on six strong features. The higher performance of the gradient boosting model indicates complex, non-linear relationships between predictors. This makes sense, as the predictors included (e.g. D4A, D3A, D3B, D5BE) are all complex measurements of destination accessibility, transportation diversity, and urban density. It makes sense that these are the factors that most strongly affect walkability. Despite good performance, there are still a few challenges. Residual plots showed good performance, but still showed some degree of variability in error across different actual values. This heteroscedasticity suggests taht certain aspects of walkability might not be fully captured by the selected predictors, or the dataset could include heterogeneity not fully addressed by the model. Metrics reflecting quality of pedestrian infrastructure (e.g. sidewalks, crossings), crime rates, and average weather/climate conditions might serve as strong additional predictors.

Model refinement could be approached a few ways. Feature engineering could be done with some of the additional metrics previously discussed. Exploring additional models or transformations could help to handle heteroscedasticity by handling variance in errors across the range of actual values. With additional time, I would experiment with log-transforms on individual predictors for example. Tuning hyperparameters of the gradient boosting model such as learning rate or max depth might also improve performance.