analysis

March 8, 2025

[1]:
```
import pandas as pd
from sqlalchemy import create_engine
import geopandas as gpd

import os

# Database connection details from zshrc environment variables
db_name = 'colorado_spills'
user = os.getenv('DB_USER')
password = os.getenv('DB_PASSWORD')
host = os.getenv('DB_HOST')
port = os.getenv('DB_PORT')


# Create an engine to connect to the PostgreSQL database
engine = create_engine(f'postgresql+psycopg2://{user}:{password}@{host}:{port}/
 ↪{db_name}')

# Function to load data from a table
def load_table(table_name):
    query = f'SELECT * FROM {table_name}'
    df = pd.read_sql(query, engine)
    return df

# Load the spills and demographics data
spills_with_demographics = load_table('spills_with_demographics')
```

[2]:
```
print(spills_with_demographics.columns.tolist())
```

```
['Document #', 'Report', 'Operator', 'Operator #', 'Tracking #', 'Initial Report
Date', 'Date of Discovery', 'Spill Type', 'Qtr Qtr', 'Section', 'Township',
'range', 'meridian', 'Latitude', 'Longitude', 'Municipality', 'county',
'Facility Type', 'Facility ID', 'API County Code', 'API Sequence Number',
'Spilled outside of berms', 'More than five barrels spilled', 'Oil Spill
Volume', 'Condensate Spill Volume', 'Flow Back Spill Volume', 'Produced Water
Spill Volume', 'E&P Waste Spill Volume', 'Other Waste', 'Drilling Fluid Spill
Volume', 'Current Land Use', 'Other Land Use', 'Weather Conditions', 'Surface
Owner', 'Surface Owner Other', 'Waters of the State', 'Residence / Occupied
```

```
Structure', 'livestock', 'Public Byway', 'Surface Water Supply Area', 'Spill
Description', 'Supplemental Report Date', 'Oil BBLs Spilled', 'Oil BBLs
Recovered', 'Oil Unknown', 'Condensate BBLs Spilled', 'Condensate BBLs
Recovered', 'Condensate Unknown', 'Produced Water BBLs Spilled', 'Produced Water
BBLs Recovered', 'Produced Water Unknown', 'Drilling Fluid BBLs Spilled',
'Drilling Fluid BBLs Recovered', 'Drilling Fluid Unknown', 'Flow Back Fluid BBLs
Spilled', 'Flow Back Fluid BBLs Recovered', 'Flow Back Fluid Unkown', 'Other E&P
Waste BBLS Spilled', 'Other E&P Waste BBLS Recovered', 'Other E&P Waste
Unknown', 'Other E&P Waste', 'Spill Contained within Berm', 'Emergency Pit
Constructed', 'soil', 'groundwater', 'Surface Water', 'Dry Drainage Feature',
'Surface Area Length', 'Surface Area Width', 'Depth of Impact in Feet', 'Depth
of Impact in Inches', 'Area Depth Determined', 'Geology Description', 'Depth to
Groundwater', 'Water wells in area', 'Water Wells', 'Water Wells None', 'Surface
Water Near', 'Surface Water None', 'Wetlands', 'Wetlands None', 'Springs',
'Springs None', 'Livestock Near', 'Livestock None', 'Occupied Buildings',
'Occupied Buildings None', 'Additional Spill Details', 'Supplemental Report Date
CA', 'Human Error', 'Equipment Failure', 'Historical Unkown', 'Other', 'Other
Description', 'Root Cause', 'Preventative Measures', 'Soil Excavated', 'Offsite
Disposal', 'Onsite Treatment', 'Other Disposition', 'Other Disposition
Description', 'Ground Water Removed', 'Surface Water Removed', 'Corrective
Actions Completed', 'Approved Form 27', 'Form 27 Project Number', 'GEOID',
'TRACT_NAME', 'total_population', 'white_population', 'hispanic_population',
'median_household_income', 'poverty_population', 'unemployed_population',
'percent_white', 'percent_hispanic', 'percent_poverty', 'unemployment_rate']
```

[3]:
```
import pandas as pd

date_columns = ['Initial Report Date', 'Date of Discovery', 'Supplemental␣
 ↪Report Date', 'Supplemental Report Date CA']

for col in date_columns:
    spills_with_demographics[col] = pd.
 ↪to_datetime(spills_with_demographics[col], errors='coerce')

# Check the result
print(spills_with_demographics[date_columns].dtypes)
```

```
Initial Report Date            datetime64[ns]
Date of Discovery              datetime64[ns]
Supplemental Report Date       datetime64[ns]
Supplemental Report Date CA    datetime64[ns]
dtype: object
```

[8]:
```
import numpy as np

def clean_and_convert(x):
    if isinstance(x, str):
        # Remove any non-numeric characters except decimal point
```

```
        x = ''.join(c for c in x if c.isdigit() or c == '.')
        return float(x) if x else np.nan
    return x

spills_with_demographics['Oil Spill Volume'] = spills_with_demographics['Oil␣
 ↪Spill Volume'].apply(clean_and_convert)
spills_with_demographics['Produced Water Spill Volume'] =␣
 ↪spills_with_demographics['Produced Water Spill Volume'].
 ↪apply(clean_and_convert)

# Convert to numeric type
spills_with_demographics['Oil Spill Volume'] = pd.
 ↪to_numeric(spills_with_demographics['Oil Spill Volume'], errors='coerce')
spills_with_demographics['Produced Water Spill Volume'] = pd.
 ↪to_numeric(spills_with_demographics['Produced Water Spill Volume'],␣
 ↪errors='coerce')

# Check the result
print(spills_with_demographics[['Oil Spill Volume', 'Produced Water Spill␣
 ↪Volume']].dtypes)
```

```
Oil Spill Volume              float64
Produced Water Spill Volume    float64
dtype: object
```

[10]:
```
# Count of spills per year
spills_per_year = spills_with_demographics['Year'].value_counts().sort_index()
print("Spills per year:")
print(spills_per_year)

# Sum of spill volumes by year
yearly_total_volumes = spills_with_demographics.groupby('Year').agg({
    'Oil Spill Volume': lambda x: x.astype(float).sum(skipna=True),
    'Produced Water Spill Volume': lambda x: x.astype(float).sum(skipna=True)
})
print("\nTotal spill volumes by year:")
print(yearly_total_volumes)

# Average spill volumes by year
yearly_avg_volumes = yearly_total_volumes.div(spills_per_year, axis=0)
print("\nAverage spill volumes by year:")
print(yearly_avg_volumes)

# Correlation between time to report and spill volume
correlation = spills_with_demographics[['Days to Report', 'Oil Spill Volume',␣
 ↪'Produced Water Spill Volume']].corr(method='spearman')
print("\nSpearman Correlation between reporting delay and spill volumes:")
```

```
print(correlation)
```

```
Spills per year:
Year
1994       1
2004       1
2009       3
2010       8
2011       5
2012       7
2013      21
2014    1099
2015    1580
2016    1373
2017    1599
2018    1581
2019    1557
2020    1166
2021    1790
2022    2155
2023    2080
2024     864
Name: count, dtype: int64

Total spill volumes by year:
      Oil Spill Volume  Produced Water Spill Volume
Year
1994               0.0                          0.0
2004               0.0                          1.0
2009               0.0                          0.0
2010               0.0                          0.0
2011               0.0                      10200.0
2012               3.0                      30700.0
2013               0.0                      46115.0
2014          404477.0                    1080000.0
2015          553498.0                    1977935.0
2016          590644.0                    1811414.0
2017          637575.0                    1514706.0
2018          571563.0                    1854170.0
2019          601360.0                    2094208.0
2020          447700.0                    1258209.0
2021          416769.0                    1273508.0
2022          512475.0                    1556782.0
2023          400198.0                    1195050.0
2024          134124.0                     383838.0

Average spill volumes by year:
      Oil Spill Volume  Produced Water Spill Volume
```

```
Year
1994          0.000000                    0.000000
2004          0.000000                    1.000000
2009          0.000000                    0.000000
2010          0.000000                    0.000000
2011          0.000000                 2040.000000
2012          0.428571                 4385.714286
2013          0.000000                 2195.952381
2014        368.040946                  982.711556
2015        350.315190                 1251.857595
2016        430.184996                 1319.310998
2017        398.733583                  947.283302
2018        361.519924                 1172.783049
2019        386.229929                 1345.027617
2020        383.962264                 1079.081475
2021        232.831844                  711.456983
2022        237.807425                  722.404640
2023        192.402885                  574.543269
2024        155.236111                  444.256944

Spearman Correlation between reporting delay and spill volumes:
                            Days to Report  Oil Spill Volume  \
Days to Report                    1.000000         -0.014038
Oil Spill Volume                 -0.014038          1.000000
Produced Water Spill Volume       0.006207         -0.254482

                            Produced Water Spill Volume
Days to Report                                 0.006207
Oil Spill Volume                              -0.254482
Produced Water Spill Volume                    1.000000
```

[6]:
```python
# Average demographic characteristics of spill locations by year
yearly_demographics = spills_with_demographics.
 ↪groupby('Year')[['percent_white', 'percent_hispanic',␣
 ↪'median_household_income', 'percent_poverty']].mean()
print("\nAverage demographic characteristics of spill locations by year:")
print(yearly_demographics)

# Correlation between spill frequency and demographic characteristics
spills_per_tract = spills_with_demographics.groupby('GEOID').size().
 ↪reset_index(name='spill_count')
tract_demographics = spills_with_demographics.
 ↪groupby('GEOID')[['percent_white', 'percent_hispanic',␣
 ↪'median_household_income', 'percent_poverty']].first()
tract_analysis = pd.merge(spills_per_tract, tract_demographics, on='GEOID')
correlation = tract_analysis[['spill_count', 'percent_white',␣
 ↪'percent_hispanic', 'median_household_income', 'percent_poverty']].corr()
```

```
print("\nCorrelation between spill frequency and demographic characteristics:")
print(correlation['spill_count'])
```

```
Average demographic characteristics of spill locations by year:
      percent_white  percent_hispanic  median_household_income  \
Year
1994      28.591783         23.557350             1.012500e+05
2004      79.166667         10.784314             1.436460e+05
2009      81.660008         20.347086             7.067033e+04
2010      77.983425         22.914365             6.457300e+04
2011      83.450494         28.346579             8.144260e+04
2012      84.676145         11.650869             5.971800e+04
2013      80.786704         26.118158             6.534219e+04
2014      82.974270         23.382431            -9.629444e+06
2015      83.703315         22.309731            -4.142396e+06
2016      83.478642         23.321560            -4.295124e+06
2017      83.914508         21.828941            -7.426691e+06
2018      84.082954         21.682858             7.731813e+04
2019      83.985918         21.211912            -7.630970e+06
2020      83.125641         21.580733            -1.478967e+07
2021      83.962007         22.303013            -5.504801e+06
2022      83.105479         23.252316            -6.414875e+06
2023      82.844350         23.542315            -6.971533e+06
2024      83.594628         23.552187            -3.005627e+06

      percent_poverty
Year
1994        10.757540
2004         4.166667
2009         8.601781
2010         9.571823
2011         7.043807
2012         9.834123
2013        10.637816
2014        10.344828
2015        10.787674
2016        10.976589
2017        10.238045
2018        11.052733
2019        11.045891
2020        10.627531
2021         9.849063
2022         9.581029
2023         9.860368
2024         9.919254

Correlation between spill frequency and demographic characteristics:
```

```
spill_count                 1.000000
percent_white              -0.008777
percent_hispanic            0.048403
median_household_income     0.012053
percent_poverty             0.133947
Name: spill_count, dtype: float64
```

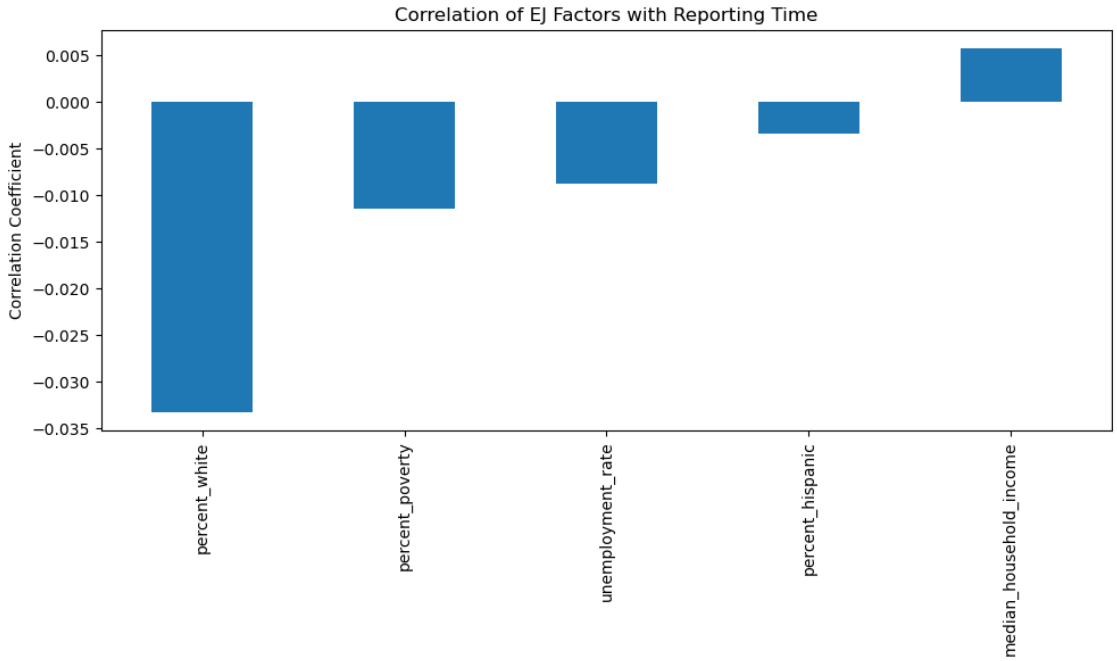
[11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Ensure 'Days to Report' is calculated correctly
spills_with_demographics['Days to Report'] = (spills_with_demographics['Initial␣
 ↪Report Date'] - spills_with_demographics['Date of Discovery']).dt.days

# List of EJ variables we'll examine
ej_variables = ['percent_white', 'percent_hispanic', 'median_household_income',␣
 ↪'percent_poverty', 'unemployment_rate']

# Remove any rows with missing values in these columns
analysis_df = spills_with_demographics.dropna(subset=['Days to Report'] +␣
 ↪ej_variables)

print(f"Number of spills for analysis: {len(analysis_df)}")
```

```
Number of spills for analysis: 16886
```

[12]:
```
# Calculate correlations
correlations = analysis_df[['Days to Report'] + ej_variables].corr()['Days to␣
 ↪Report'].sort_values()

print("Correlations with Days to Report:")
print(correlations)

# Visualize correlations
plt.figure(figsize=(10, 6))
correlations.drop('Days to Report').plot(kind='bar')
plt.title('Correlation of EJ Factors with Reporting Time')
plt.ylabel('Correlation Coefficient')
plt.tight_layout()
plt.show()
```

```
Correlations with Days to Report:
percent_white              -0.033242
percent_poverty            -0.011415
unemployment_rate          -0.008741
```

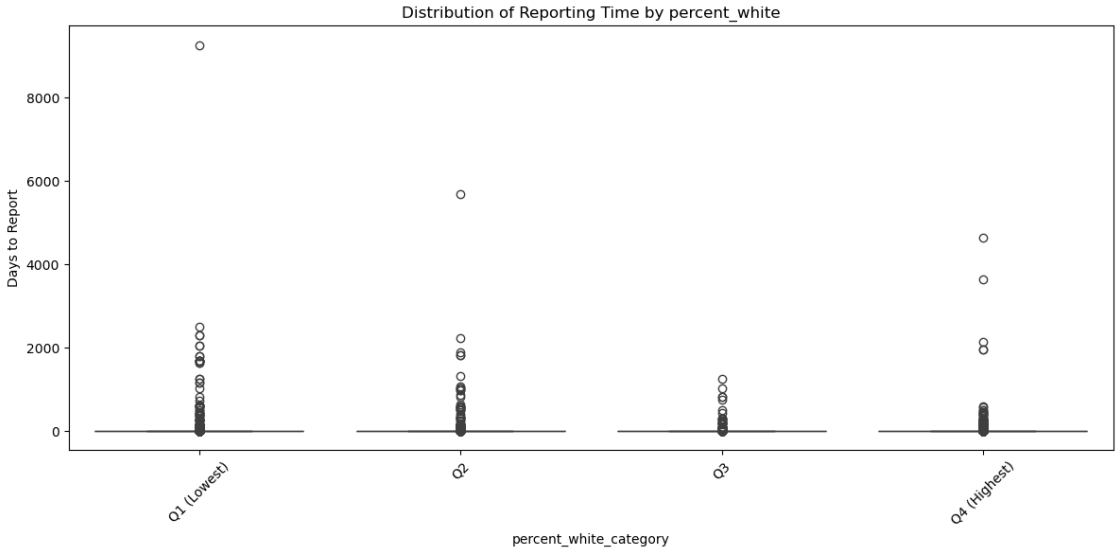
```
percent_hispanic          -0.003342
median_household_income    0.005737
Days to Report             1.000000
Name: Days to Report, dtype: float64
```

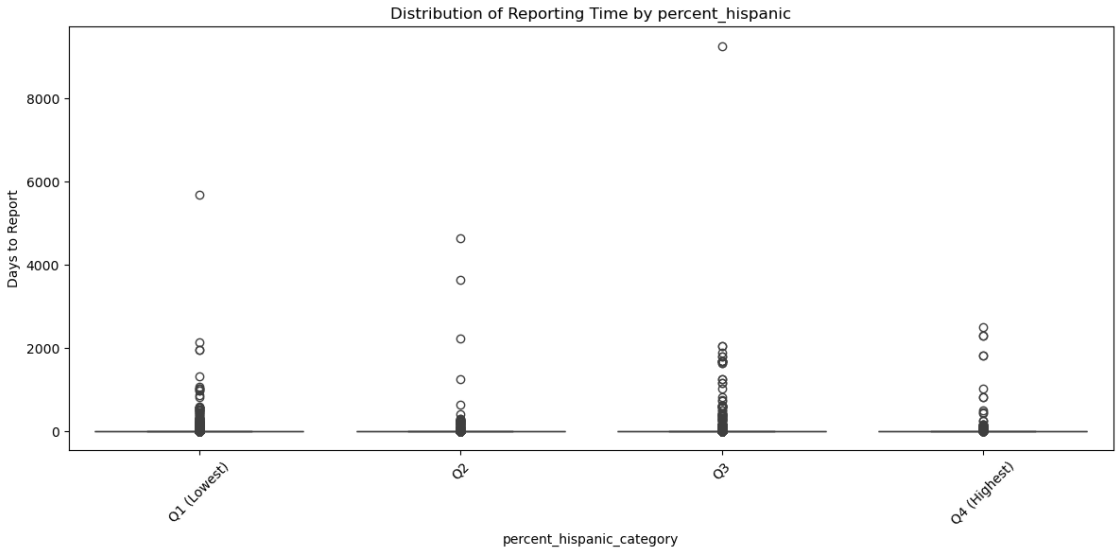


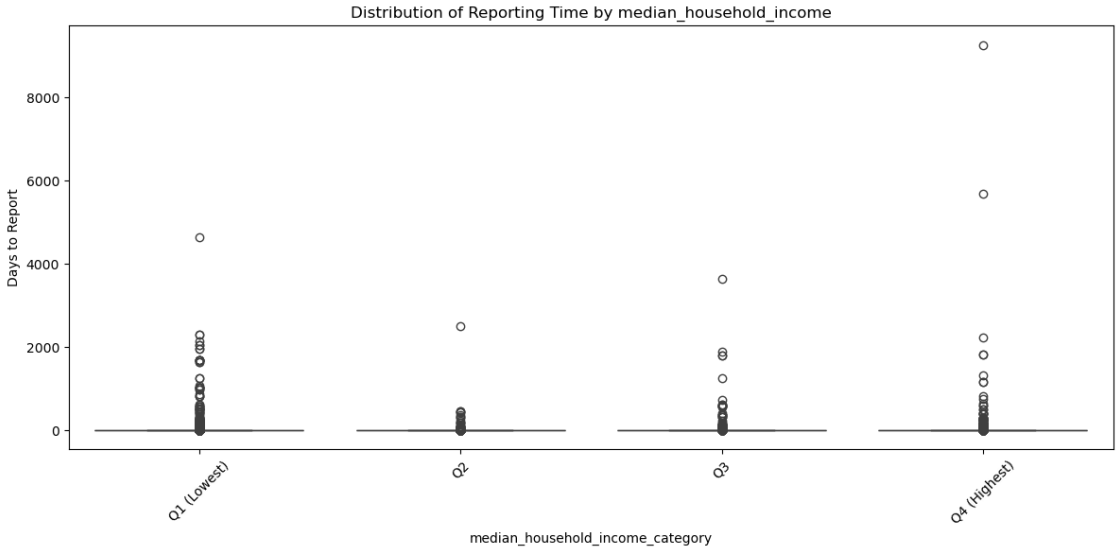
[16]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Create a copy of the DataFrame to work with
analysis_df = spills_with_demographics.copy()

# Ensure 'Days to Report' is calculated correctly
analysis_df['Days to Report'] = (analysis_df['Initial Report Date'] -␣
 ↪analysis_df['Date of Discovery']).dt.days

# List of EJ variables we'll examine
ej_variables = ['percent_white', 'percent_hispanic', 'median_household_income',␣
 ↪'percent_poverty', 'unemployment_rate']

# Remove any rows with missing values in these columns
analysis_df = analysis_df.dropna(subset=['Days to Report'] + ej_variables)
```

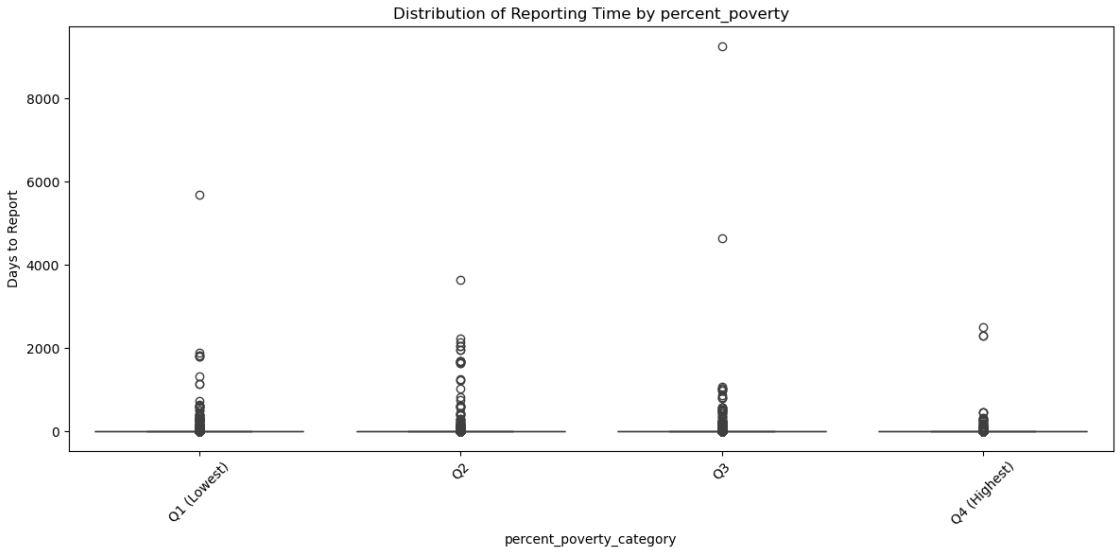
```
print(f"Number of spills for analysis: {len(analysis_df)}")

# Function to categorize tracts into quartiles
def categorize_quartiles(series):
    return pd.qcut(series, q=4, labels=['Q1 (Lowest)', 'Q2', 'Q3', 'Q4␣
 ↪(Highest)'])

# Create categorical variables for each EJ factor
for var in ej_variables:
    analysis_df[f'{var}_category'] = categorize_quartiles(analysis_df[var])

# Function to plot boxplots
def plot_boxplots(variable):
    plt.figure(figsize=(12, 6))
    sns.boxplot(x=f'{variable}_category', y='Days to Report', data=analysis_df)
    plt.title(f'Distribution of Reporting Time by {variable}')
    plt.ylabel('Days to Report')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Plot boxplots for each EJ variable
for var in ej_variables:
    plot_boxplots(var)
```

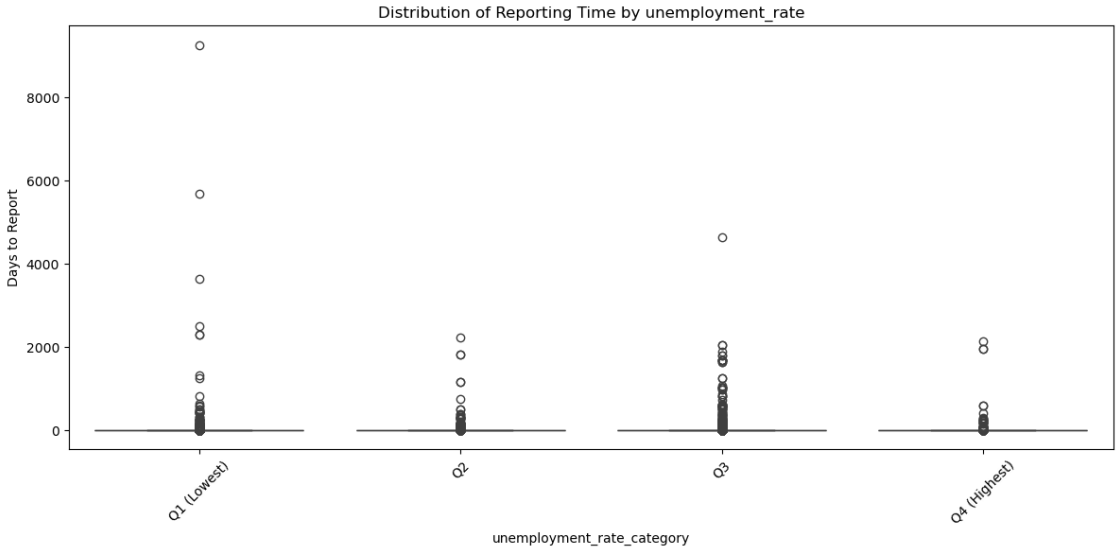
```
Number of spills for analysis: 16886
```




Distribution of Reporting Time by percent_hispanic
Days to Report
8000
6000
4000
2000
0
Q1 (Lowest)
Q2
Q3
Q4 (Highest)
percent_hispanic_category


Distribution of Reporting Time by median_household_income
Days to Report
8000
6000
4000
2000
0
Q1 (Lowest)
Q2
Q3
Q4 (Highest)
median_household_income_category





[18]:
```
# Function to perform ANOVA
def perform_anova(variable):
    categories = analysis_df[f'{variable}_category'].unique()
    samples = [group['Days to Report'].values for name, group in analysis_df.
 ↪groupby(f'{variable}_category', observed=True)]
    f_value, p_value = stats.f_oneway(*samples)
    return f_value, p_value

# Perform ANOVA for each EJ variable
for var in ej_variables:
```

```
    f_value, p_value = perform_anova(var)
    print(f"ANOVA results for {var}:")
    print(f"F-value: {f_value:.4f}")
    print(f"p-value: {p_value:.4f}")
    print()
```

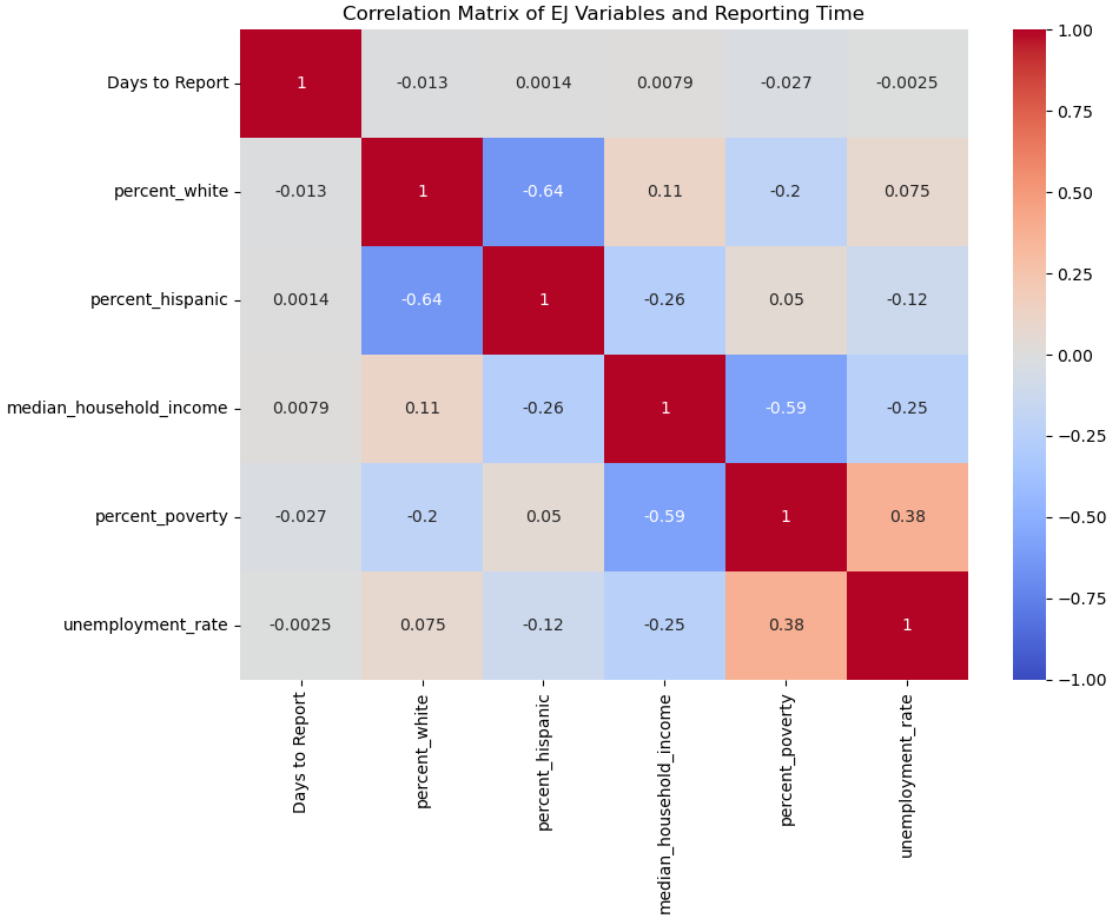
```
ANOVA results for percent_white:
F-value: 3.4856
p-value: 0.0151

ANOVA results for percent_hispanic:
F-value: 2.8655
p-value: 0.0352

ANOVA results for median_household_income:
F-value: 7.6346
p-value: 0.0000

ANOVA results for percent_poverty:
F-value: 3.3987
p-value: 0.0170

ANOVA results for unemployment_rate:
F-value: 5.1023
p-value: 0.0016
```

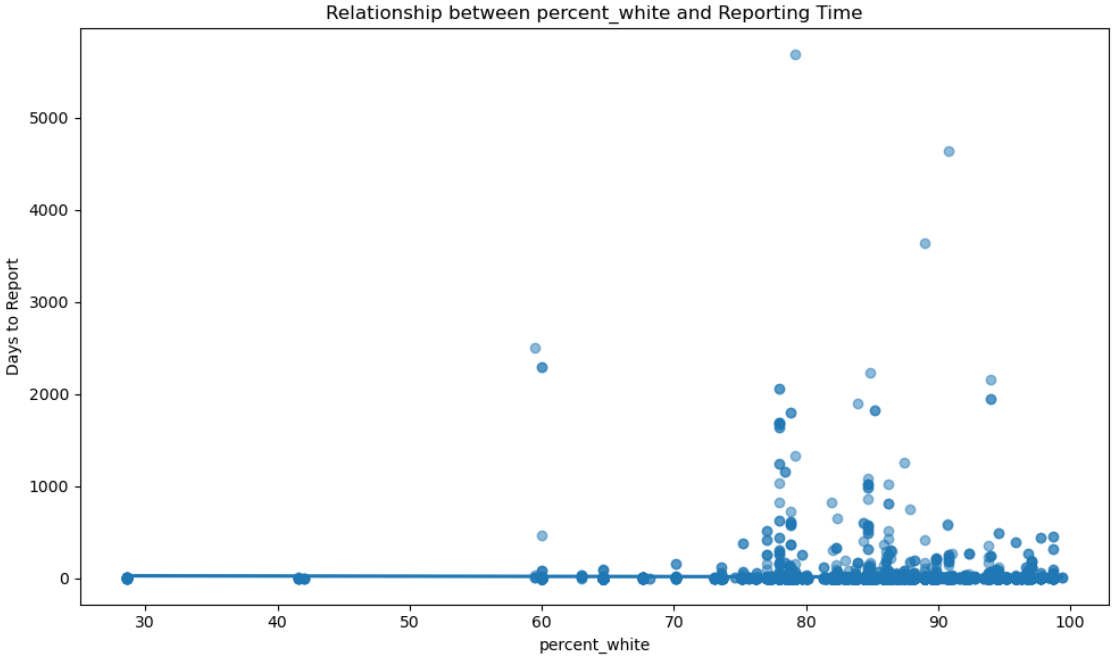
[24]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

# Remove extreme outliers (above 8000 days), ensure positive days, and remove␣
 ↪negative incomes
analysis_df_cleaned = analysis_df[
    (analysis_df['Days to Report'] <= 8000) &
    (analysis_df['Days to Report'] > 0) &
    (analysis_df['median_household_income'] > 0)
]

print(f"Original dataset size: {len(analysis_df)}")
print(f"Dataset size after cleaning: {len(analysis_df_cleaned)}")
print(f"Number of records removed: {len(analysis_df) -␣
 ↪len(analysis_df_cleaned)}")

# List of EJ variables
```

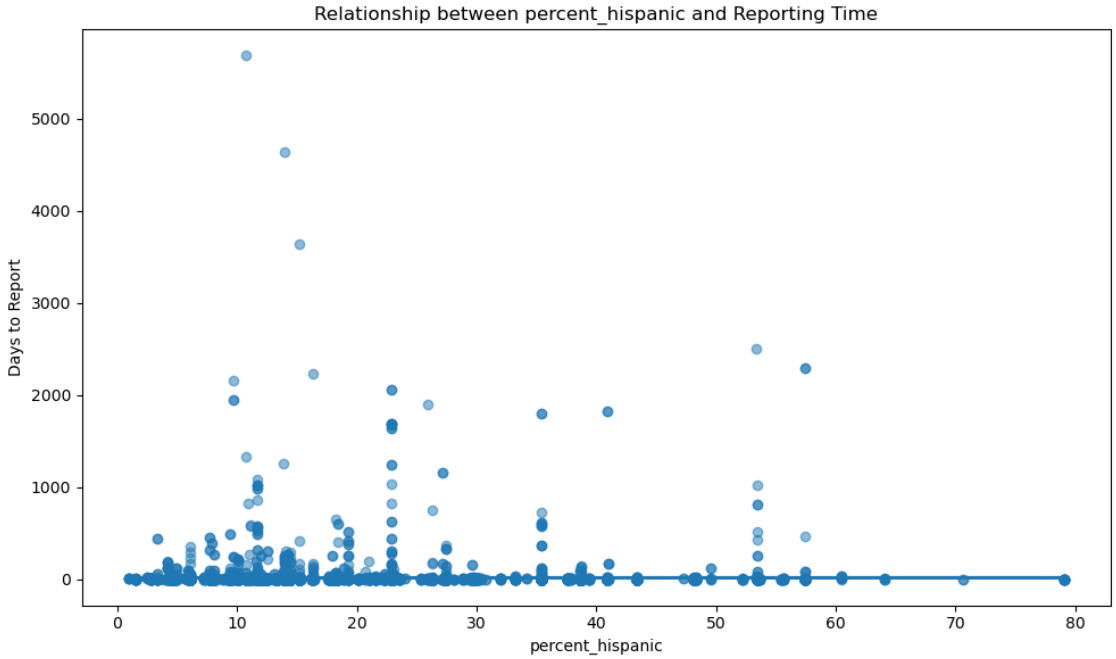
```
ej_variables = ['percent_white', 'percent_hispanic', 'median_household_income',␣
 ↪'percent_poverty', 'unemployment_rate']

# Correlation matrix
correlation_matrix = analysis_df_cleaned[['Days to Report'] + ej_variables].
 ↪corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)

# Visualize correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1,␣
 ↪center=0)
plt.title('Correlation Matrix of EJ Variables and Reporting Time')
plt.tight_layout()
plt.show()

# Scatter plots with regression lines
for var in ej_variables:
    plt.figure(figsize=(10, 6))
    sns.regplot(x=var, y='Days to Report', data=analysis_df_cleaned,␣
 ↪scatter_kws={'alpha':0.5})
    plt.title(f'Relationship between {var} and Reporting Time')
    plt.xlabel(var)
    plt.ylabel('Days to Report')
    plt.tight_layout()
    plt.show()

# Multiple Linear Regression
X = sm.add_constant(analysis_df_cleaned[ej_variables])
y = analysis_df_cleaned['Days to Report']
model = sm.OLS(y, X).fit()
print(model.summary())

# Residual Analysis
residuals = model.resid
fitted_values = model.fittedvalues

plt.figure(figsize=(10, 6))
sns.scatterplot(x=fitted_values, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.tight_layout()
plt.show()
```

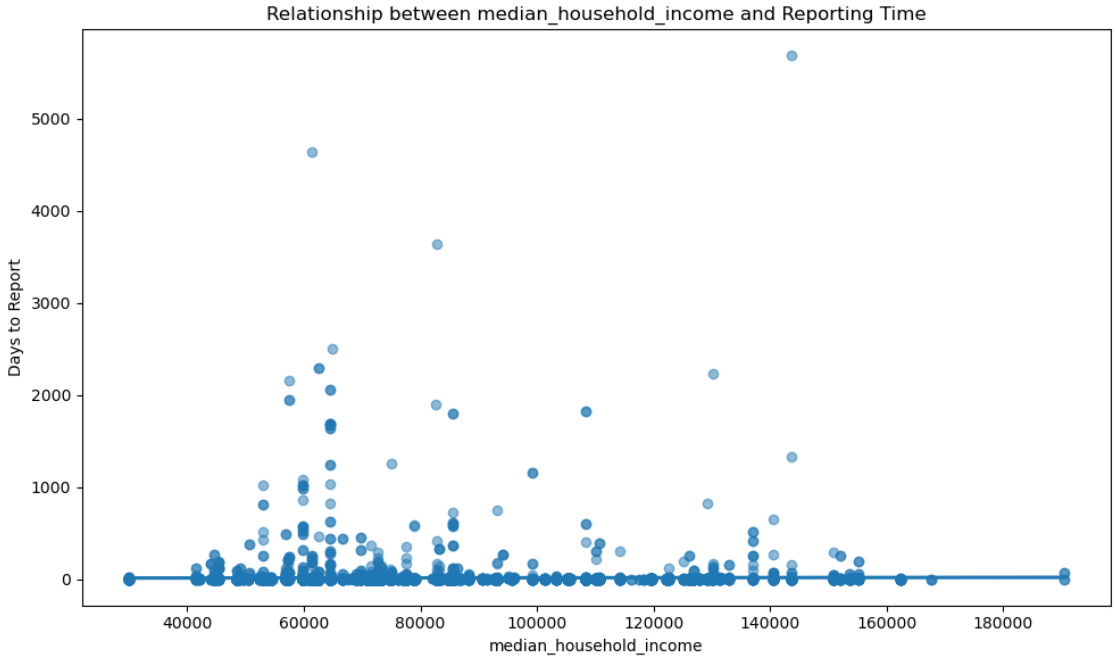
```python
# Q-Q plot for normality check
fig, ax = plt.subplots(figsize=(10, 6))
sm.qqplot(residuals, ax=ax)
ax.set_title('Q-Q Plot of Residuals')
plt.tight_layout()
plt.show()

# Variance Inflation Factor for multicollinearity check
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
  ↪shape[1])]
print("\nVariance Inflation Factors:")
print(vif_data)
```

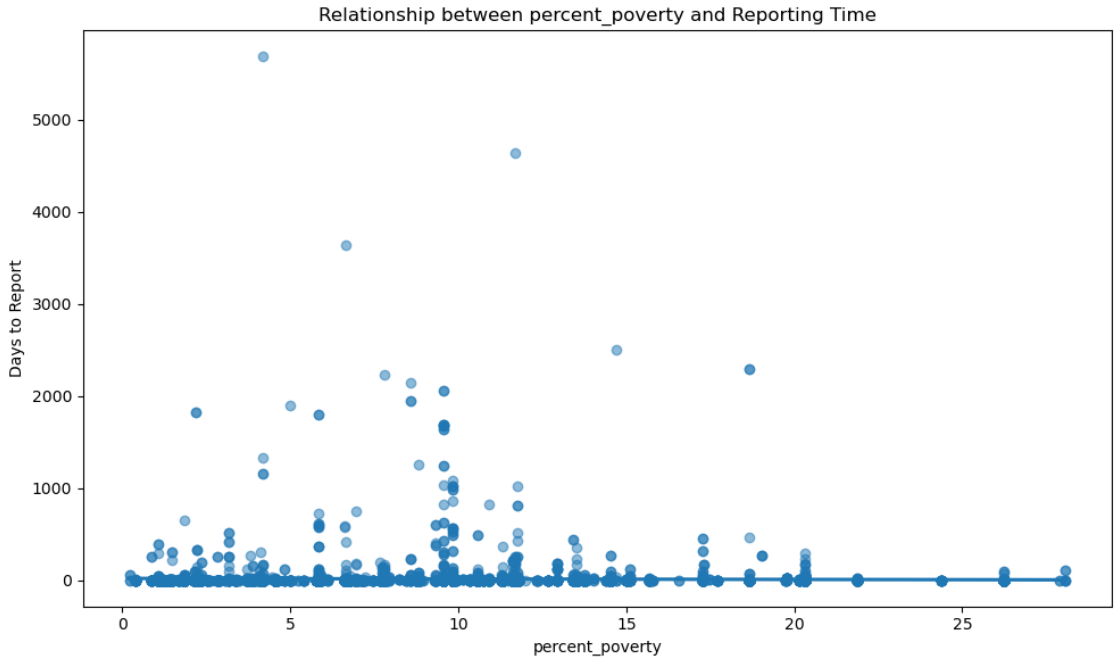
```
Original dataset size: 16886
Dataset size after cleaning: 10916
Number of records removed: 5970

Correlation Matrix:
                        Days to Report  percent_white  percent_hispanic  \
Days to Report                1.000000      -0.012665          0.001356
percent_white                -0.012665       1.000000         -0.641930
percent_hispanic              0.001356      -0.641930          1.000000
median_household_income       0.007872       0.113642         -0.259411
percent_poverty              -0.027037      -0.204533          0.049957
unemployment_rate            -0.002549       0.074651         -0.120400

                        median_household_income  percent_poverty  \
Days to Report                         0.007872        -0.027037
percent_white                          0.113642        -0.204533
percent_hispanic                      -0.259411         0.049957
median_household_income                1.000000        -0.585739
percent_poverty                       -0.585739         1.000000
unemployment_rate                     -0.246535         0.382319

                        unemployment_rate
Days to Report                  -0.002549
percent_white                    0.074651
percent_hispanic                -0.120400
median_household_income         -0.246535
percent_poverty                  0.382319
unemployment_rate                1.000000
```

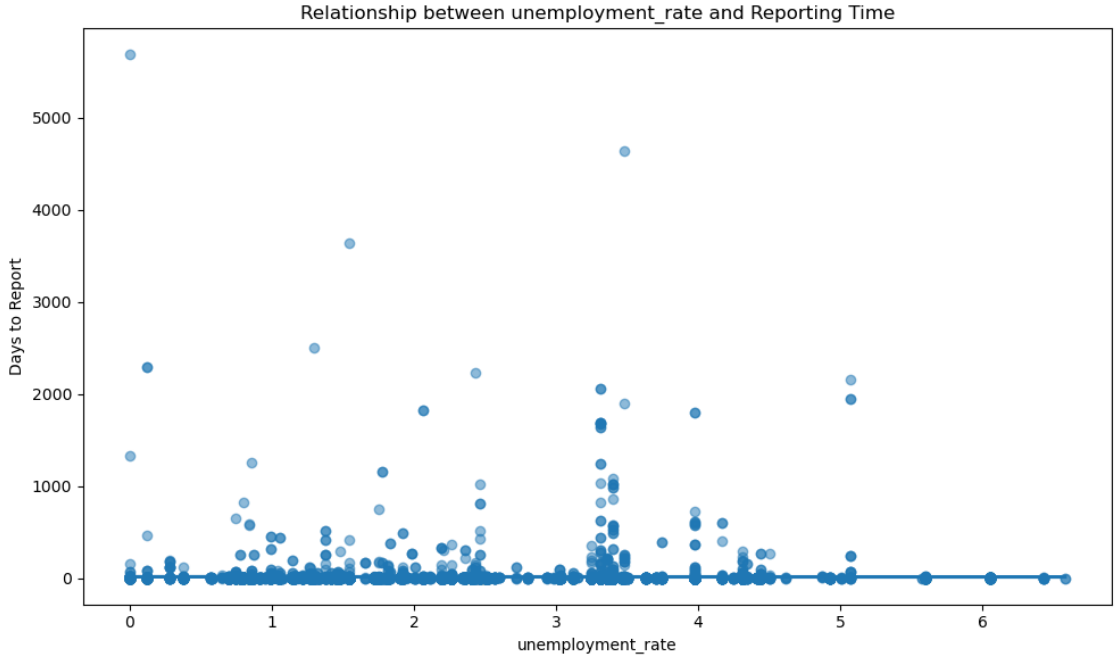

Correlation Matrix of EJ Variables and Reporting Time
Days to Report
percent_white
percent_hispanic
median_household_income
percent_poverty
unemployment_rate
1
-0.013
0.0014
0.0079
-0.027
-0.0025
-0.013
1
-0.64
0.11
-0.2
0.075
0.0014
-0.64
1
-0.26
0.05
-0.12
0.0079
0.11
-0.26
1
-0.59
-0.25
-0.027
-0.2
0.05
-0.59
1
0.38
-0.0025
0.075
-0.12
-0.25
0.38
1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Relationship between percent_white and Reporting Time
Days to Report
percent_white
5000
4000
3000
2000
1000
0
30
40
50
60
70
80
90
100


Relationship between percent_hispanic and Reporting Time
Days to Report
percent_hispanic
0
1000
2000
3000
4000
5000
0
10
20
30
40
50
60
70
80


Relationship between median_household_income and Reporting Time
Days to Report
median_household_income
0
1000
2000
3000
4000
5000
40000
60000
80000
100000
120000
140000
160000
180000

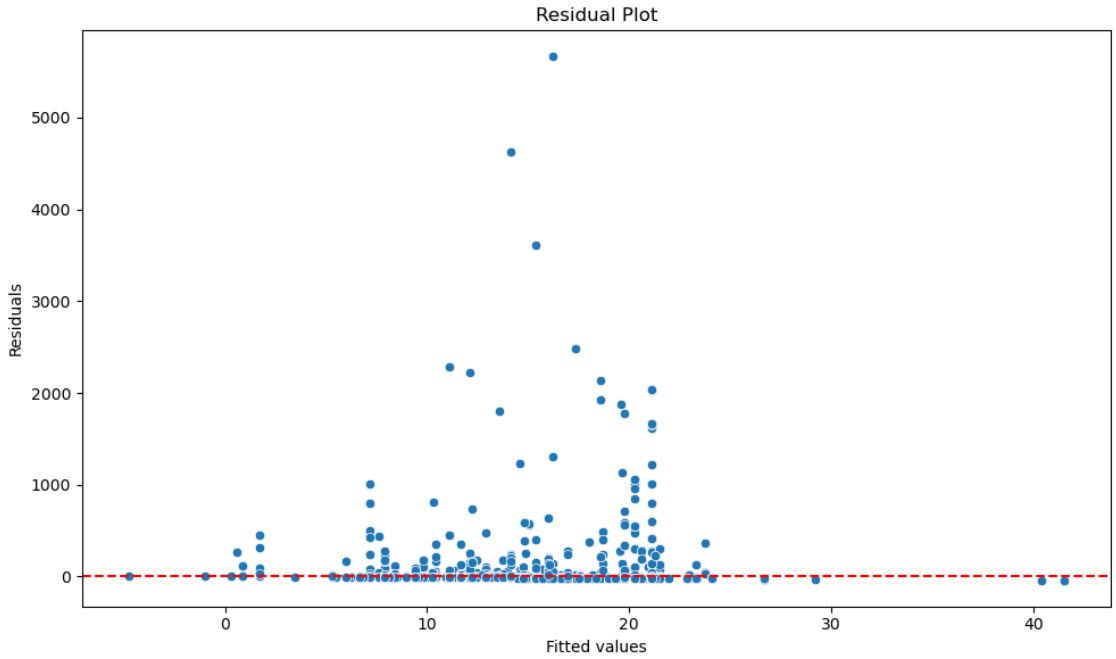


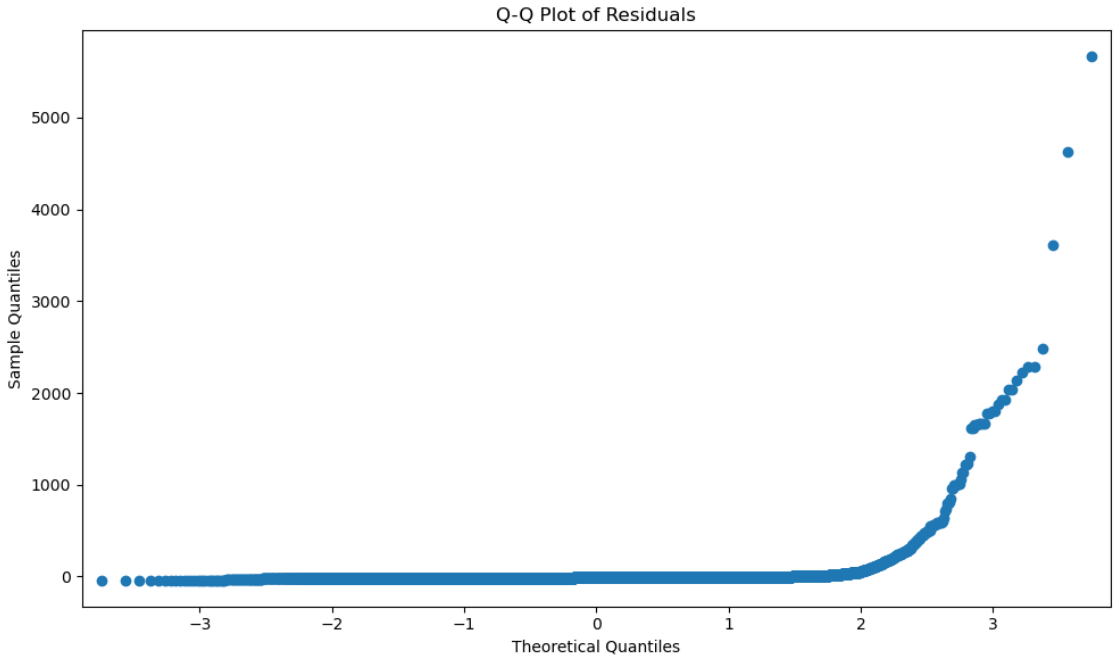


OLS Regression Results

Dep. Variable:	Days to Report	R-squared:	0.002
Model:	OLS	Adj. R-squared:	0.001

```
Method:                 Least Squares   F-statistic:                     3.450
Date:                Fri, 09 Aug 2024   Prob (F-statistic):            0.00407
Time:                        14:17:50   Log-Likelihood:                -68649.
No. Observations:               10916   AIC:                         1.373e+05
Df Residuals:                   10910   BIC:                         1.374e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         85.5864     23.853      3.588      0.000      38.830     132.343
percent_white -0.5888      0.216     -2.732      0.006      -1.011      -0.166
percent_hispanic -0.2400      0.139     -1.722      0.085      -0.513       0.033
median_household_income -0.0001   7.03e-05     -1.588      0.112      -0.000    2.62e-05
percent_poverty -0.9954      0.266     -3.747      0.000      -1.516      -0.475
unemployment_rate  1.1161      1.044      1.069      0.285      -0.930       3.162
==============================================================================
Omnibus:                    24581.151   Durbin-Watson:                   1.900
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        182543160.185
Skew:                          21.262   Prob(JB):                         0.00
Kurtosis:                     635.086   Cond. No.                     1.55e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.55e+06. This might indicate that there are strong multicollinearity or other numerical problems.
```





Variance Inflation Factors:

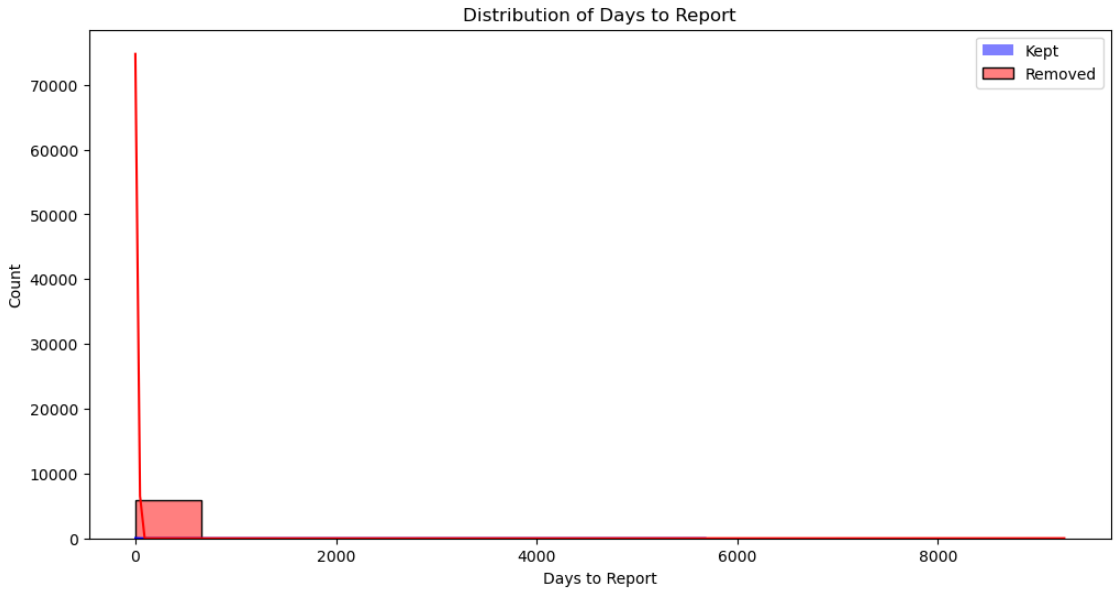
	Variable	VIF
0	const	365.555860

```
1            percent_white    1.935037
2          percent_hispanic    2.020569
3  median_household_income    1.783219
4           percent_poverty    1.906722
5         unemployment_rate    1.212996
```

[25]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create a DataFrame of removed rows
removed_df = analysis_df[
    (analysis_df['Days to Report'] > 8000) |
    (analysis_df['Days to Report'] <= 0) |
    (analysis_df['median_household_income'] <= 0)
]

# Create a DataFrame of kept rows
kept_df = analysis_df_cleaned

print(f"Total rows in original dataset: {len(analysis_df)}")
print(f"Rows kept: {len(kept_df)}")
print(f"Rows removed: {len(removed_df)}")

# Function to compare distributions
def compare_distributions(variable):
    plt.figure(figsize=(12, 6))
    sns.histplot(kept_df[variable], kde=True, color='blue', alpha=0.5,␣
 ↪label='Kept')
    sns.histplot(removed_df[variable], kde=True, color='red', alpha=0.5,␣
 ↪label='Removed')
    plt.title(f'Distribution of {variable}')
    plt.legend()
    plt.show()

# Compare distributions for key variables
for var in ['Days to Report', 'median_household_income'] + ej_variables:
    compare_distributions(var)

# Summary statistics for removed rows
print("\nSummary statistics for removed rows:")
print(removed_df.describe())

# Count of rows removed for each reason
reason_counts = {
    'Days > 8000': sum(removed_df['Days to Report'] > 8000),
    'Days <= 0': sum(removed_df['Days to Report'] <= 0),
```

```
    'Income <= 0': sum(removed_df['median_household_income'] <= 0)
}
print("\nReason for removal:")
for reason, count in reason_counts.items():
    print(f"{reason}: {count}")

# Check for any patterns in categorical variables
categorical_vars = ['Spill Type', 'county', 'Facility Type']  # Add any other␣
 ↪relevant categorical variables
for var in categorical_vars:
    if var in removed_df.columns:
        print(f"\nTop 10 {var} in removed data:")
        print(removed_df[var].value_counts().head(10))
```

```
Total rows in original dataset: 16886
Rows kept: 10916
Rows removed: 5970
```




```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[25], line 30
     28 # Compare distributions for key variables
     29 for var in ['Days to Report', 'median_household_income'] + ej_variables
---> 30     compare_distributions(var)
     32 # Summary statistics for removed rows
     33 print("\nSummary statistics for removed rows:")

Cell In[25], line 26, in compare_distributions(variable)
```

```
     24 plt.title(f'Distribution of {variable}')
     25 plt.legend()
---> 26 plt.show()

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/pyplot
 ↪py:527, in show(*args, **kwargs)
    483 """
    484 Display all open figures.
    485
   (…)
    524 explicitly there.
    525 """
    526 _warn_if_gui_out_of_main_thread()
--> 527 return _get_backend_mod().show(*args, **kwargs)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib_inline
 ↪backend_inline.py:90, in show(close, block)
     88 try:
     89     for figure_manager in Gcf.get_all_fig_managers():
---> 90         display(
     91             figure_manager.canvas.figure,
     92             metadata=_fetch_figure_metadata(figure_manager.canvas.figur )
     93         )
     94 finally:
     95     show._to_draw = []

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/IPython/core/
 ↪display_functions.py:298, in display(include, exclude, metadata, transient,␣
 ↪display_id, raw, clear, *objs, **kwargs)
    296     publish_display_data(data=obj, metadata=metadata, **kwargs)
    297 else:
--> 298     format_dict, md_dict = format(obj, include=include, exclude=exclude
    299     if not format_dict:
    300         # nothing to display (e.g. _ipython_display_ took over)
    301         continue

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/IPython/core/
 ↪formatters.py:182, in DisplayFormatter.format(self, obj, include, exclude)
    180 md = None
    181 try:
--> 182     data = formatter(obj)
    183 except:
    184     # FIXME: log the exception
    185     raise

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/decorator.py:232,
 ↪in decorate.<locals>.fun(*args, **kw)
    230 if not kwsyntax:
```

```
    231     args, kw = fix(args, kw, sig)
--> 232 return caller(func, *(extras + args), **kw)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/IPython/core/
 ↪formatters.py:226, in catch_format_error(method, self, *args, **kwargs)
    224 """show traceback on failed format call"""
    225 try:
--> 226     r = method(self, *args, **kwargs)
    227 except NotImplementedError:
    228     # don't warn on NotImplementedErrors
    229     return self._check_return(None, args[0])

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/IPython/core/
 ↪formatters.py:343, in BaseFormatter.__call__(self, obj)
    341     pass
    342 else:
--> 343     return printer(obj)
    344 # Finally look for special method names
    345 method = get_real_method(obj, self.print_method)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/IPython/core/
 ↪pylabtools.py:170, in print_figure(fig, fmt, bbox_inches, base64, **kwargs)
    167     from matplotlib.backend_bases import FigureCanvasBase
    168     FigureCanvasBase(fig)
--> 170 fig.canvas.print_figure(bytes_io, **kw)
    171 data = bytes_io.getvalue()
    172 if fmt == 'svg':

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/
 ↪backend_bases.py:2164, in FigureCanvasBase.print_figure(self, filename, dpi,␣
 ↪facecolor, edgecolor, orientation, format, bbox_inches, pad_inches,␣
 ↪bbox_extra_artists, backend, **kwargs)
   2161     # we do this instead of `self.figure.draw_without_rendering`
   2162     # so that we can inject the orientation
   2163     with getattr(renderer, "_draw_disabled", nullcontext)():
-> 2164         self.figure.draw(renderer)
   2165 if bbox_inches:
   2166     if bbox_inches == "tight":

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/artist
 ↪py:95, in _finalize_rasterization.<locals>.draw_wrapper(artist, renderer,␣
 ↪*args, **kwargs)
     93 @wraps(draw)
     94 def draw_wrapper(artist, renderer, *args, **kwargs):
---> 95     result = draw(artist, renderer, *args, **kwargs)
     96     if renderer._rasterizing:
     97         renderer.stop_rasterizing()
```

```
File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/artist
 ↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/figure
 ↪py:3154, in Figure.draw(self, renderer)
   3151         # ValueError can occur when resizing a window.
   3153 self.patch.draw(renderer)
-> 3154 mimage._draw_list_compositing_images(
   3155     renderer, self, artists, self.suppressComposite)
   3157 for sfig in self.subfigs:
   3158     sfig.draw(renderer)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/image.
 ↪py:132, in _draw_list_compositing_images(renderer, parent, artists,␣
 ↪suppress_composite)
    130 if not_composite or not has_images:
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/artist
 ↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/axes/
 ↪_base.py:3070, in _AxesBase.draw(self, renderer)
   3067 if artists_rasterized:
   3068     _draw_rasterized(self.figure, artists_rasterized, renderer)
-> 3070 mimage._draw_list_compositing_images(
   3071     renderer, self, artists, self.figure.suppressComposite)
   3073 renderer.close_group('axes')
   3074 self.stale = False

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/image.
 ↪py:132, in _draw_list_compositing_images(renderer, parent, artists,␣
 ↪suppress_composite)
    130 if not_composite or not has_images:
```

```
    131     for a in artists:
--> 132         a.draw(renderer)
    133 else:
    134     # Composite any adjacent images together
    135     image_group = []

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/artist
 ↪py:72, in allow_rasterization.<locals>.draw_wrapper(artist, renderer)
     69     if artist.get_agg_filter() is not None:
     70         renderer.start_filter()
---> 72     return draw(artist, renderer)
     73 finally:
     74     if artist.get_agg_filter() is not None:

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/legend
 ↪py:780, in Legend.draw(self, renderer)
    777     Shadow(self.legendPatch, **self._shadow_props).draw(renderer)
    779 self.legendPatch.draw(renderer)
--> 780 self._legend_box.draw(renderer)
    782 renderer.close_group('legend')
    783 self.stale = False

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/artist
 ↪py:39, in _prevent_rasterization.<locals>.draw_wrapper(artist, renderer,␣
 ↪*args, **kwargs)
     36     renderer.stop_rasterizing()
     37     renderer._rasterizing = False
---> 39 return draw(artist, renderer, *args, **kwargs)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/
 ↪offsetbox.py:412, in OffsetBox.draw(self, renderer)
    407 """
    408 Update the location of children if necessary and draw them
    409 to the given *renderer*.
    410 """
    411 bbox, offsets = self._get_bbox_and_child_offsets(renderer)
--> 412 px, py = self.get_offset(bbox, renderer)
    413 for c, (ox, oy) in zip(self.get_visible_children(), offsets):
    414     c.set_offset((px + ox, py + oy))

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/
 ↪offsetbox.py:60, in _compat_get_offset.<locals>.get_offset(self, *args,␣
 ↪**kwargs)
     56 params = _api.select_matching_signature(sigs, self, *args, **kwargs)
     57 bbox = (params["bbox"] if "bbox" in params else
     58         Bbox.from_bounds(-params["xdescent"], -params["ydescent"],
     59                          params["width"], params["height"]))
---> 60 return meth(params["self"], bbox, params["renderer"])
```

```
File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/
 ↪offsetbox.py:312, in OffsetBox.get_offset(self, bbox, renderer)
    297 @_compat_get_offset
    298 def get_offset(self, bbox, renderer):
    299     """
    300     Return the offset as a tuple (x, y).
    301
   (…)
    309     renderer : `.RendererBase` subclass
    310     """
    311     return (
--> 312         ␣
 ↪self._offset(bbox.width, bbox.height, -bbox.x0, -bbox.y0, renderer)
    313         if callable(self._offset)
    314         else self._offset)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/legend
 ↪py:738, in Legend._findoffset(self, width, height, xdescent, ydescent,␣
 ↪renderer)
    735 """Helper function to locate the legend."""
    737 if self._loc == 0:  # "best".
--> 738     x, y = self._find_best_position(width, height, renderer)
    739 elif self._loc in Legend.codes.values():  # Fixed location.
    740     bbox = Bbox.from_bounds(0, 0, width, height)

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/legend
 ↪py:1171, in Legend._find_best_position(self, width, height, renderer, conside
 ↪)
   1167 assert self.isaxes  # always holds, as this is only called internally
   1169 start_time = time.perf_counter()
-> 1171 bboxes, lines, offsets = self._auto_legend_data()
   1173 bbox = Bbox.from_bounds(0, 0, width, height)
   1174 if consider is None:

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/legend
 ↪py:977, in Legend._auto_legend_data(self)
    973     lines.append(
    974         artist.get_transform().transform_path(artist.get_path()))
    975 elif isinstance(artist, Rectangle):
    976     bboxes.append(
--> 977         artist.get_bbox().transformed(artist.get_data_transform()))
    978 elif isinstance(artist, Patch):
    979     lines.append(
    980         artist.get_transform().transform_path(artist.get_path()))

File ~/miniconda3/envs/funkyfunk/lib/python3.10/site-packages/matplotlib/
 ↪transforms.py:468, in BboxBase.transformed(self, transform)
    464 """
```

```
    465 Construct a `Bbox` by statically transforming this one by *transform*.
    466 """
    467 pts = self.get_points()
--> 468 ll, ul, lr = transform.transform(np.array(
    469     [pts[0], [pts[0, 0], pts[1, 1]], [pts[1, 0], pts[0, 1]]]))
    470 return Bbox([ll, [lr[0], ul[1]]])

KeyboardInterrupt:
```