Load in the CSV we collected during data collection.

```
import pandas as pd
import seaborn as sns


df = pd.read_csv("https://raw.githubusercontent.com/AlexanderCalafiura/DataScience112FinalProject/refs/heads/main/AC_ML_dataset_1000.csv")
df = df.rename(columns={'Zip Code': 'zipcode'})
df
```

	zipcode	Per Diem Daily Rate	Rate Zone	House Price	City	State	Population Size	Population Density	County FIPS	County Name	isStandard	Cluster	Price_to_PerDiem_Ratio
0	10002	256.666667	New York City	1.007951e+06	New York	NY	75517	35458.5	36061	New York	False	2	3927.083523
1	10003	256.666667	New York City	1.396697e+06	New York	NY	53825	36357.3	36061	New York	False	2	5441.678039
2	10009	256.666667	New York City	8.344787e+05	New York	NY	58341	36524.7	36061	New York	False	2	3251.215762
3	10016	256.666667	New York City	9.288574e+05	New York	NY	54297	38131.6	36061	New York	False	2	3618.924930
4	10023	256.666667	New York City	1.337869e+06	New York	NY	67468	26875.1	36061	New York	False	2	5212.475343
...	...	...	...	...	...	...	...	...	...	...	...	...	...
931	98661	164.500000	Vancouver	3.321296e+05	Vancouver	WA	52027	1842.8	53011	Clark	False	0	2019.024970
932	98682	164.500000	Vancouver	3.498594e+05	Vancouver	WA	67297	911.2	53011	Clark	False	0	2126.804651
933	99208	127.000000	Spokane	3.130112e+05	Spokane	WA	58834	466.1	53063	Spokane	False	0	2464.654825
934	99301	118.000000	Richland / Pasco	2.813411e+05	Pasco	WA	86467	68.7	53021	Franklin	False	1	2384.246692
935	99336	118.000000	Richland / Pasco	2.628567e+05	Kennewick	WA	51180	1475.2	53005	Benton	False	0	2227.599373

```
import pandas as pd
import seaborn as sns


df = pd.read_csv("https://raw.githubusercontent.com/AlexanderCalafiura/DataScience112FinalProject/refs/heads/main/AC_ML_dataset_26000.csv")
df = df.rename(columns={'Zip Code': 'zipcode'})
df
```

	zipcode	Per Diem Daily Rate	Rate Zone	House Price	City	State	Population Size	Population Density	County FIPS	County Name	isStandard
0	1001	122.0	Springfield	219508.964752	Agawam	MA	16136	551.7	25013	Hampden	False
1	1002	146.0	Northampton	339812.737345	Amherst	MA	24726	179.3	25015	Hampshire	False
2	1005	130.0	Worcester	244509.442046	Barre	MA	4786	42.8	25027	Worcester	False
3	1007	146.0	Northampton	295540.622491	Belchertown	MA	15406	108.4	25015	Hampshire	False
4	1008	122.0	Springfield	232702.865612	Blandford	MA	1324	8.4	25013	Hampden	False
...	...	...	...	...	...	...	...	...	...	...	...
26176	99360	107.0	Standard Rate	348803.942100	Touchet	WA	1299	3.1	53071	Walla Walla	True
26177	99361	107.0	Standard Rate	249922.063928	Waitsburg	WA	1800	3.1	53071	Walla Walla	True
26178	99362	107.0	Standard Rate	270400.509440	Walla Walla	WA	42794	54.5	53071	Walla Walla	True
26179	99402	107.0	Standard Rate	283674.534416	Asotin	WA	1628	1.9	53003	Asotin	True
26180	99403	107.0	Standard Rate	254236.175503	Clarkston	WA	20483	55.1	53003	Asotin	True

26181 rows × 11 columns

Download a bunch of stuff to make Geopandas work. Download shapefiles and the necessary packages for zip codes and states information.

```
# get the U.S. Census zip code and states shapefile
!wget https://www2.census.gov/geo/tiger/TIGER2020/ZCTA520/tl_2020_us_zcta520.zip
!wget https://www2.census.gov/geo/tiger/TIGER2022/STATE/tl_2022_us_state.zip

# create a directory and unzip the zip code shapefile
!mkdir -p zipcode_data
!unzip -o tl_2020_us_zcta520.zip -d zipcode_data

# Step 2: create a directory and unzip the states shapefile
!mkdir -p states_data
!unzip -o tl_2022_us_state.zip -d states_data

# install necessary packages
!pip install mapclassify
!pip install contextily
!pip install geodatasets
```

```
--2025-03-21 06:00:24--  https://www2.census.gov/geo/tiger/TIGER2020/ZCTA520/tl_2020_us_zcta520.zip
Resolving www2.census.gov (www2.census.gov)... 92.123.200.196, 2600:1409:9800:e86::208c, 2600:1409:9800:e8d::208c
Connecting to www2.census.gov (www2.census.gov)|92.123.200.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: ‘tl_2020_us_zcta520.zip’

tl_2020_us_zcta520.     [           <=>      ] 503.54M  15.5MB/s    in 31s

2025-03-21 06:00:56 (16.3 MB/s) - ‘tl_2020_us_zcta520.zip’ saved [527995578]
```

```
--2025-03-21 06:00:56--  https://www2.census.gov/geo/tiger/TIGER2022/STATE/tl_2022_us_state.zip
Resolving www2.census.gov (www2.census.gov)... 2.19.140.252, 2600:1409:9800:e86::208c, 2600:1409:9800:e8d::208c
Connecting to www2.census.gov (www2.census.gov)|2.19.140.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'tl_2022_us_state.zip'

tl_2022_us_state.zi     [        <=>         ]   9.50M  5.98MB/s    in 1.6s

2025-03-21 06:00:59 (5.98 MB/s) - 'tl_2022_us_state.zip' saved [9967184]

Archive:  tl_2020_us_zcta520.zip
 extracting: zipcode_data/tl_2020_us_zcta520.cpg
  inflating: zipcode_data/tl_2020_us_zcta520.dbf
  inflating: zipcode_data/tl_2020_us_zcta520.prj
  inflating: zipcode_data/tl_2020_us_zcta520.shp
  inflating: zipcode_data/tl_2020_us_zcta520.shp.ea.iso.xml
  inflating: zipcode_data/tl_2020_us_zcta520.shp.iso.xml
  inflating: zipcode_data/tl_2020_us_zcta520.shx
Archive:  tl_2022_us_state.zip
 extracting: states_data/tl_2022_us_state.cpg
  inflating: states_data/tl_2022_us_state.dbf
  inflating: states_data/tl_2022_us_state.prj
  inflating: states_data/tl_2022_us_state.shp
  inflating: states_data/tl_2022_us_state.shp.ea.iso.xml
  inflating: states_data/tl_2022_us_state.shp.iso.xml
  inflating: states_data/tl_2022_us_state.shx
Collecting mapclassify
  Downloading mapclassify-2.8.1-py3-none-any.whl.metadata (2.8 kB)
Requirement already satisfied: networkx>=2.7 in /usr/local/lib/python3.11/dist-packages (from mapclassify) (3.4.2)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from mapclassify) (2.0.2)
Requirement already satisfied: pandas!=1.5.0,>=1.4 in /usr/local/lib/python3.11/dist-packages (from mapclassify) (2.2.2)
Requirement already satisfied: scikit-learn>=1.0 in /usr/local/lib/python3.11/dist-packages (from mapclassify) (1.6.1)
Requirement already satisfied: scipy>=1.8 in /usr/local/lib/python3.11/dist-packages (from mapclassify) (1.14.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas!=1.5.0,>=1.4->mapclassify) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=1.5.0,>=1.4->mapclassify) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=1.5.0,>=1.4->mapclassify) (2025.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.0->mapclassify) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.0->mapclassify) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas!=1.5.0,>=1.4->mapclassify) (1.17.0)
Downloading mapclassify-2.8.1-py3-none-any.whl (59 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.1/59.1 kB 5.2 MB/s eta 0:00:00
Installing collected packages: mapclassify
Successfully installed mapclassify-2.8.1
Collecting contextily
  Downloading contextily-1.6.2-py3-none-any.whl.metadata (2.9 kB)
```

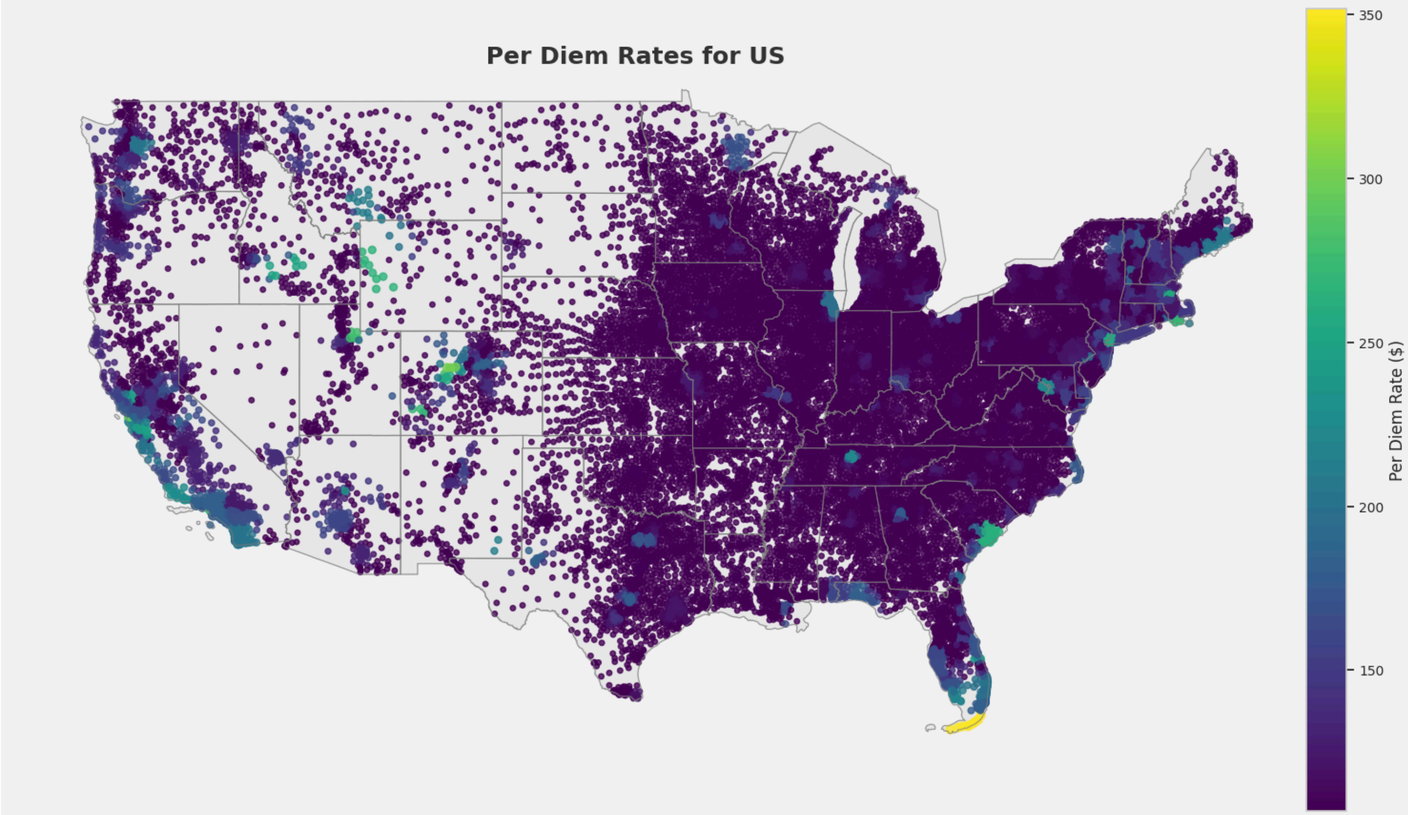
Preload the boundaries of all US zip codes to make loading the choropleth signficantly quicker. Otherwise it would need to render this every single iteration.

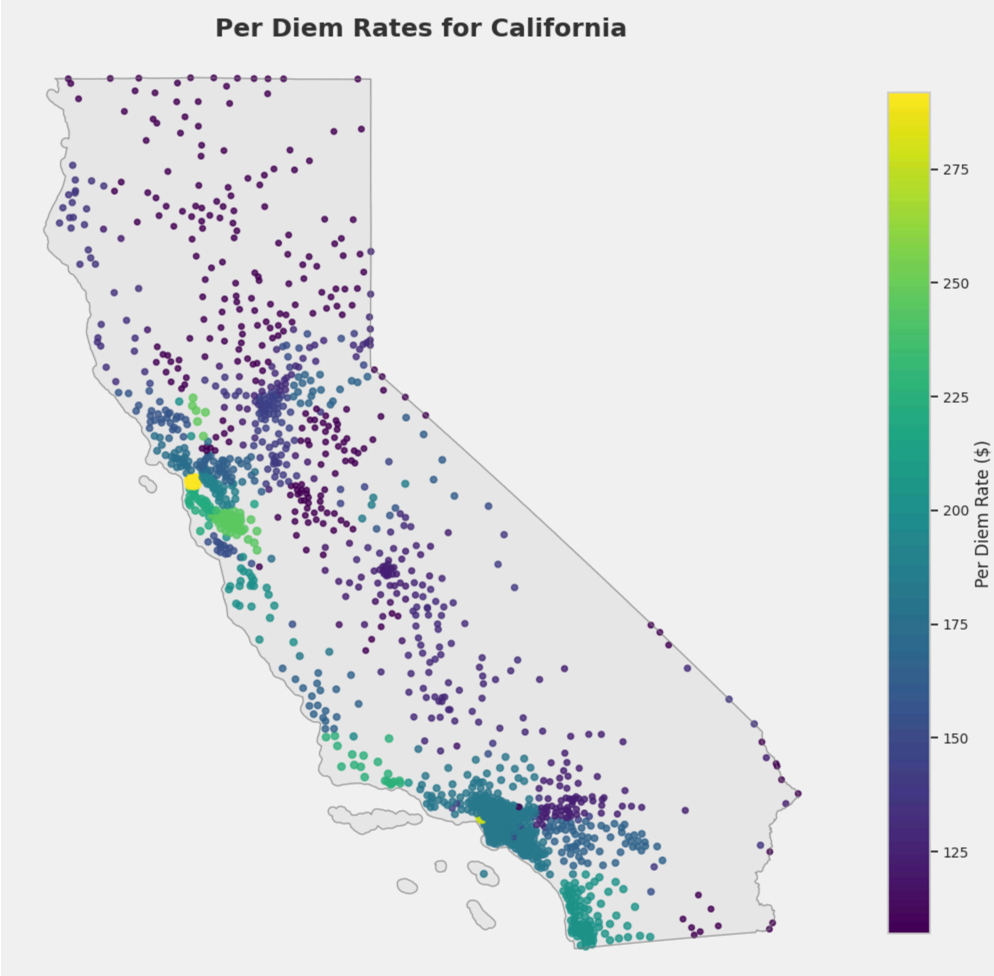
```python
import geopandas as gpd

zip_boundaries = gpd.read_file("zipcode_data/tl_2020_us_zcta520.shp")
zip_boundaries = zip_boundaries.rename(columns={'ZCTA5CE20': 'zipcode'})
# make sure there is a leading 0 for each zip code with less than 5 digits (strange government convention)
zip_boundaries['zipcode'] = zip_boundaries['zipcode'].astype(str).str.zfill(5)

# make the geometry work with Web Mercator
zip_boundaries = zip_boundaries.to_crs('EPSG:3857')
zip_boundaries['geometry'] = zip_boundaries.geometry.simplify(0.001)
```

Render a choropleth heatmap of the actual per diem rates of every zip code across the country with sufficient population data. We can also generate maps for individual states.

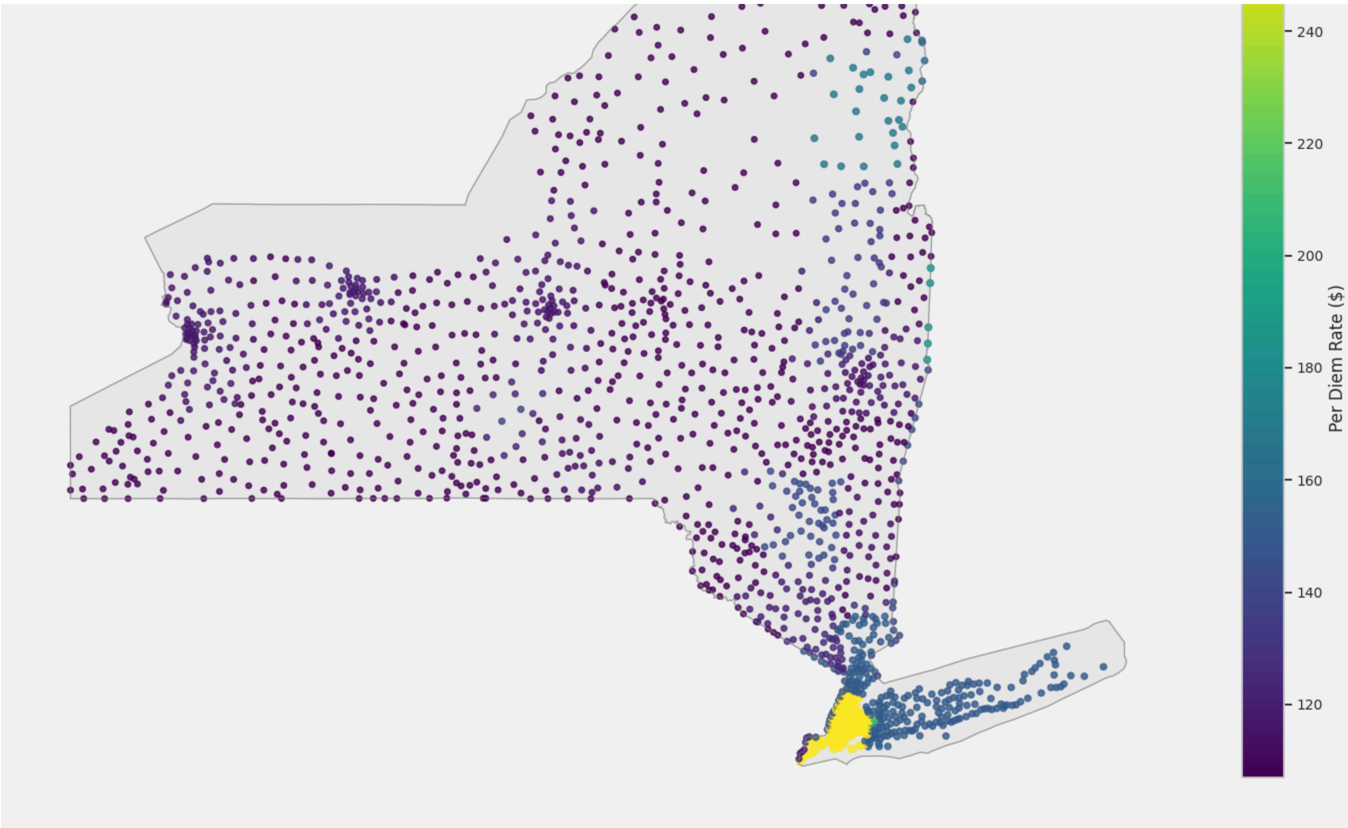
```python
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

def create_actual_diem_map(diem_df, zipcode_shapefile_path, states_shapefile_path, state_name=None, output_file=None):
    # make sure all zip codes are strings with 5 digits
    diem_df['zipcode'] = diem_df['zipcode'].astype(str).str.zfill(5)

    # merge data into one easy to access data frame
    choropleth_data = zip_boundaries.merge(
        diem_df[['zipcode', 'Per Diem Daily Rate']],
        on='zipcode',
        how='inner'
    )

    # load in the US states. Exclude Alaska and Hawaii, which have per diem rates not calculated by the GSA (and are frankly
    # unable to fit on the map anyways)
    us_states = gpd.read_file(states_shapefile_path).to_crs(choropleth_data.crs)
    us_states = us_states[~us_states['NAME'].isin(['Alaska', 'Hawaii'])]

    # if we want, allow us to filter for a specific US state only
    if state_name:
        us_states = us_states[us_states['NAME'] == state_name]
        choropleth_data = gpd.clip(choropleth_data, us_states)

    # convert to easy to access centroids
    choropleth_data['geometry'] = choropleth_data.geometry.centroid

    # make background of the visualization
    fig, ax = plt.subplots(figsize=(15, 10))
    fig.patch.set_facecolor('#f0f0f0')
    ax.set_facecolor('#ffffff')

    # plot base layers of the visualizations
    us_states.plot(ax=ax, color='#e6e6e6', edgecolor='#cccccc', linewidth=0.7)
    us_states.boundary.plot(ax=ax, linewidth=0.8, color='gray', alpha=0.7)
```

```
    # plot the per diem daily rates
    scatter = choropleth_data.plot(
        ax=ax,
        markersize=(choropleth_data['Per Diem Daily Rate'] / 10) + 5, # marker size is proportional to daily rate so we can more easily see non-standard rates
        column='Per Diem Daily Rate',
        cmap='viridis',
        alpha=0.8,
        legend=False
    )

    # color bar that accurately maps the spread of the data
    norm = Normalize(
        vmin=choropleth_data['Per Diem Daily Rate'].min(),
        vmax=choropleth_data['Per Diem Daily Rate'].max()
    )

    # make sure the plot has nice details
    sm = plt.cm.ScalarMappable(cmap='viridis', norm=norm)
    sm.set_array([])
    cbar = fig.colorbar(sm, ax=ax, fraction=0.03, pad=0.04)
    cbar.set_label('Per Diem Rate ($)', fontsize=12)
    cbar.ax.tick_params(labelsize=10)

    # stylize the plot, generate corresponding title
    title = f'Per Diem Rates for {state_name}' if state_name else ' Per Diem Rates for US'
    plt.title(title, fontsize=18, fontweight='bold', color='#333333')

    # set the proper bounds for the map (whether we are trying to get the map of a state or the etnire Continental U.S.)
    if state_name:
        minx, miny, maxx, maxy = us_states.total_bounds
        ax.set_xlim(minx - 50000, maxx + 50000)
        ax.set_ylim(miny - 50000, maxy + 50000)
    else:
        ax.set_xlim(-14284029, -7453304)
        ax.set_ylim(2717774, 6440332)

    ax.set_axis_off()
    plt.tight_layout()

    plt.savefig(output_file, dpi=300, bbox_inches='tight')

    return fig, ax

# create full US map
create_actual_diem_map(
    df,
    'zipcode_data/tl_2020_us_zcta520.shp',
    'states_data/tl_2022_us_state.shp',
    output_file='predicted_diem_map_us.png'
)

# create individual state maps at our own discretion
for state in ['California', 'New York', 'Florida', 'Michigan', "Kentucky"]:
    create_actual_diem_map(
        df,
        'zipcode_data/tl_2020_us_zcta520.shp',
        'states_data/tl_2022_us_state.shp',
        state_name=state,
        output_file=f'predicted_diem_map_{state.lower().replace(" ", "_")}.png'
    )

plt.show()
```

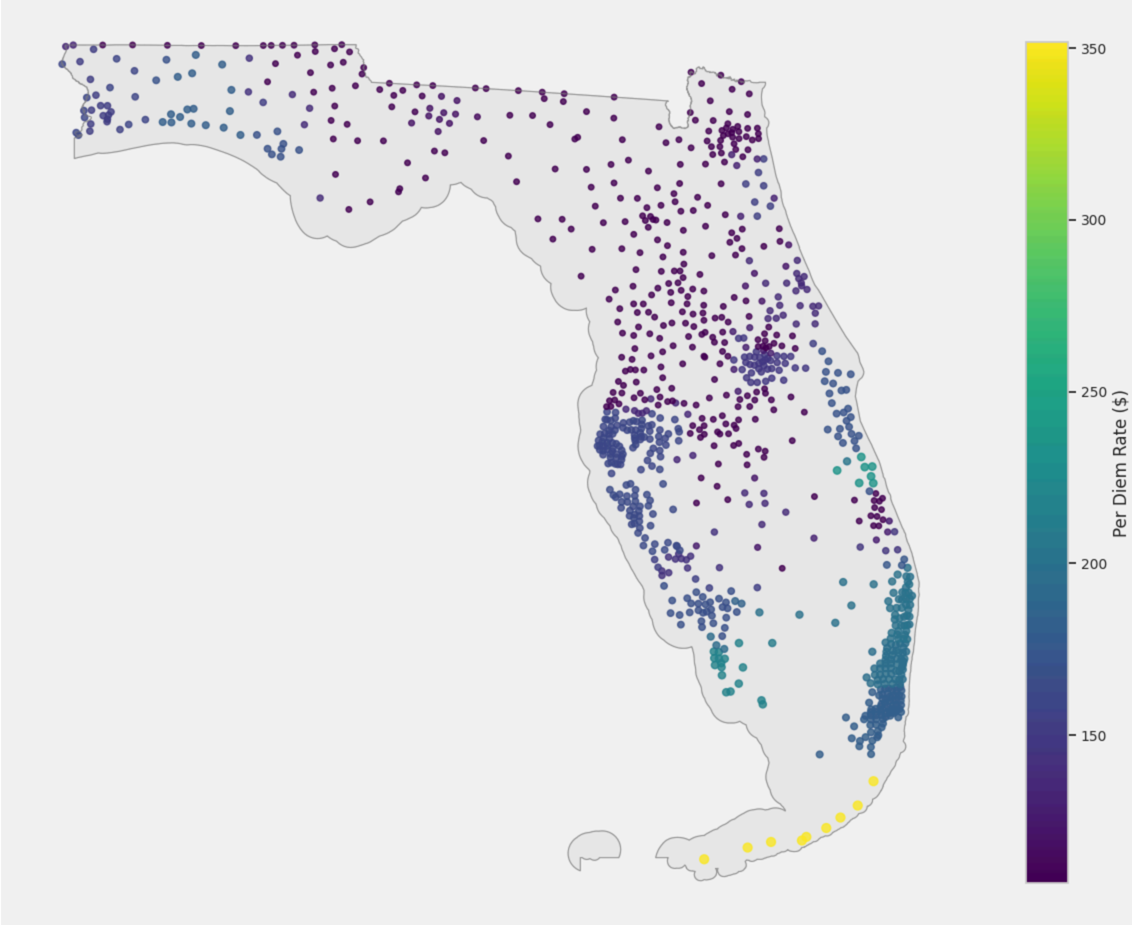

Per Diem Rates for US
Per Diem Rate ($)
350
300
250
200
150


Per Diem Rates for California
Per Diem Rate ($)
275
250
225
200
175
150
125

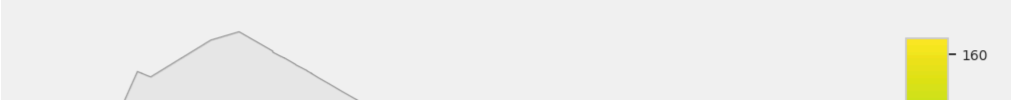

Per Diem Rates for New York

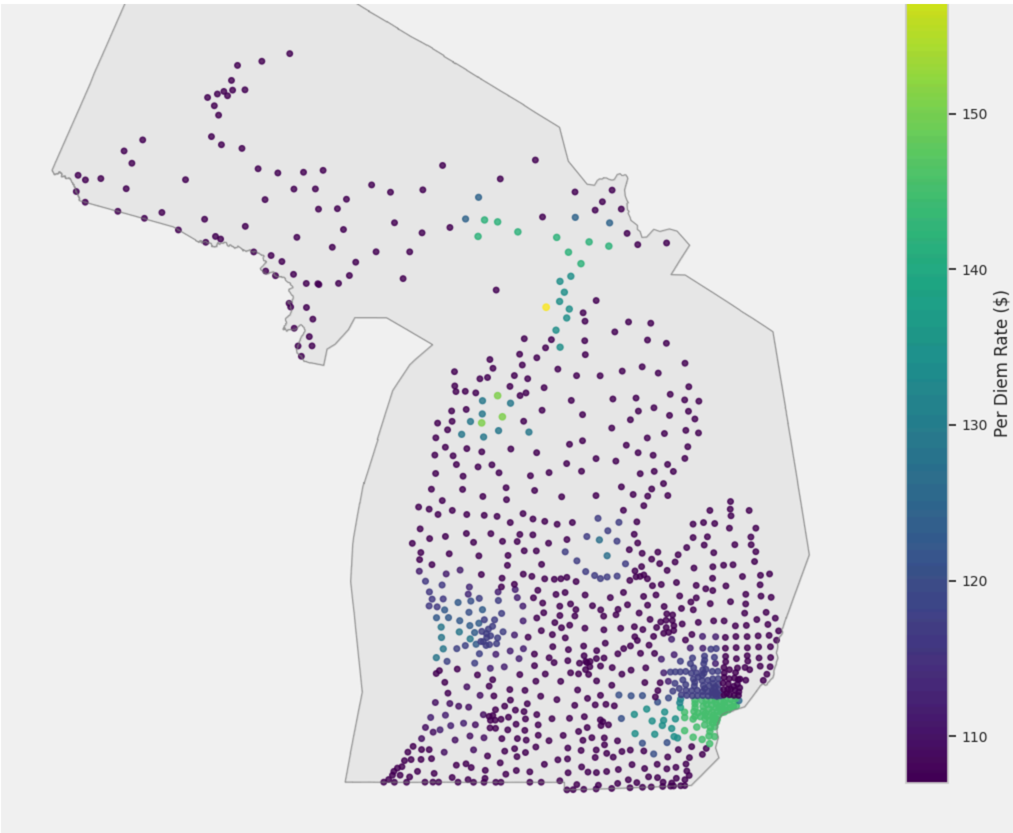


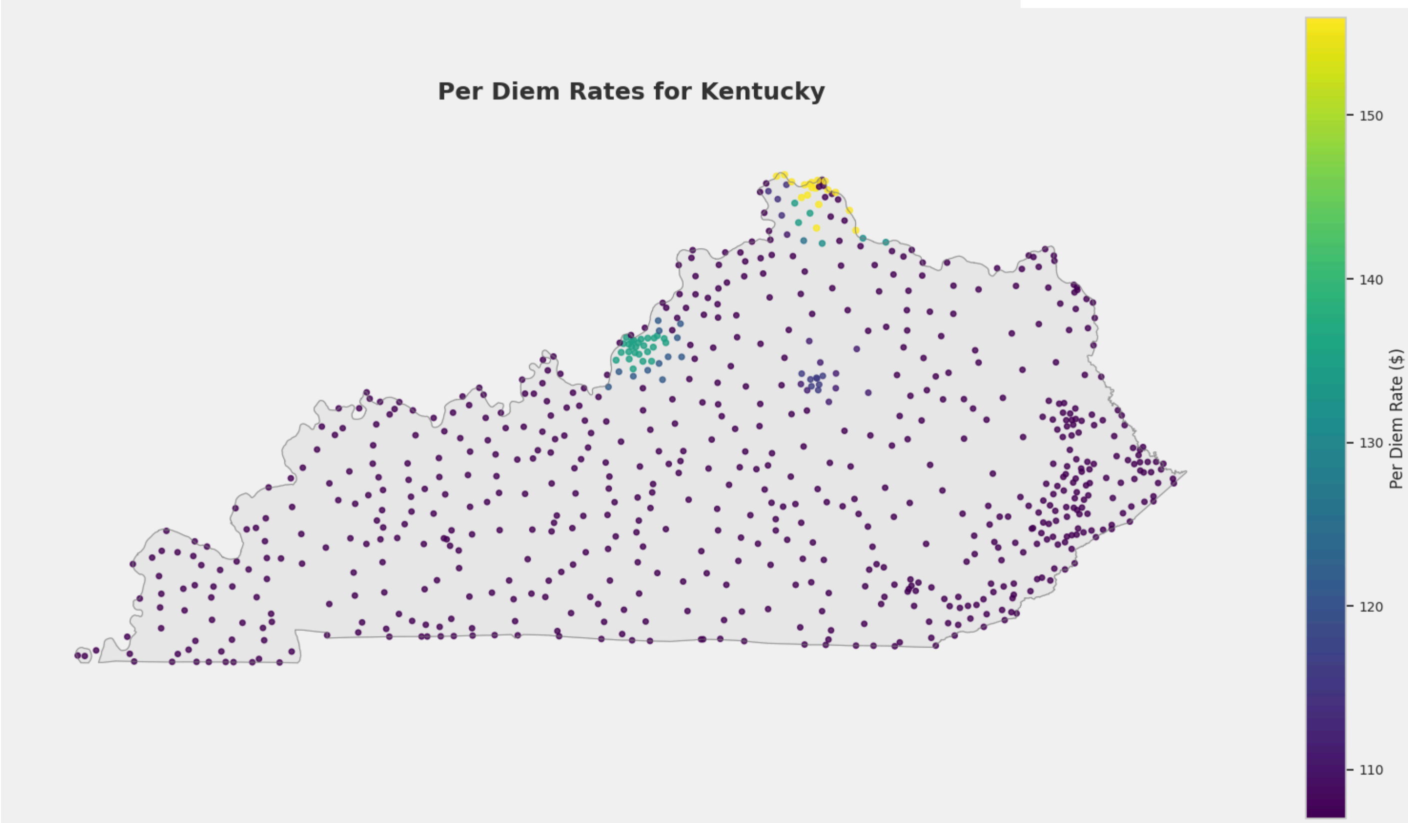
Per Diem Rates for Florida



Per Diem Rates for Michigan



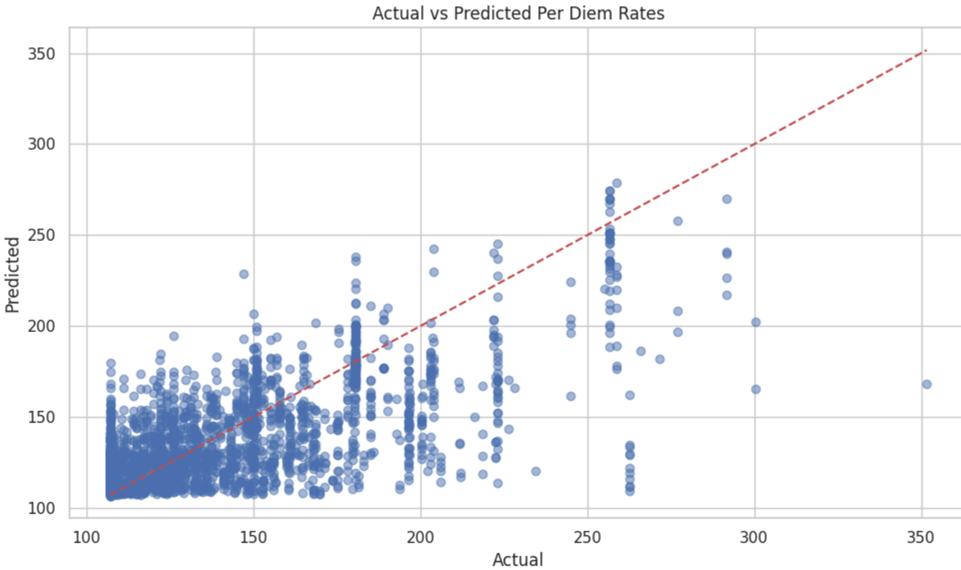

150
140
130
120
110
Per Diem Rate ($)


Per Diem Rates for Kentucky
150
140
130
120
110
Per Diem Rate ($)

Using Gradient Booster, predict what the actual per diem rates should be using a cost of living index based on population density, size, and house price.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

X = df[["House Price", "Population Size", "Population Density"]]
y = df["Per Diem Daily Rate"]

# split X and Y into training and test data. The test data is 20% of the training data and is locked to a constant state (so we get reproducible runs)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# create a gradient boosting pipeline with proper preprocessing. Use early stopping to save time if the model stops improving.
pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    RobustScaler(),
    HistGradientBoostingRegressor(
        random_state=42,
        early_stopping=True,
        scoring='loss',
        validation_fraction=0.2
    )
)

# search parameters for the best parameter of the model to minimize RMSE
param_grid = {
    'histgradientboostingregressor__learning_rate': [0.01, 0.05, 0.1],
    'histgradientboostingregressor__max_iter': [500, 1000],
    'histgradientboostingregressor__max_depth': [None, 5, 7],
    'histgradientboostingregressor__l2_regularization': [0, 0.1, 1],
    'histgradientboostingregressor__max_bins': [128, 255]
}

# use grid-search to find the best parameters
search = RandomizedSearchCV(
    estimator=pipeline,
    param_distributions=param_grid,
    n_iter=30,
    scoring="neg_root_mean_squared_error",
    cv=5,
    return_train_score=True,
    verbose=1,
    n_jobs=-1
)

# fit the data to the gridsearch
search.fit(X_train, y_train)

# obtain the best model and run our prediction on it
best_model = search.best_estimator_
y_pred_test = best_model.predict(X_test)

# get RMSE
test_rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
test_r2 = r2_score(y_test, y_pred_test)
test_mae = mean_absolute_error(y_test, y_pred_test)

print(f"Test RMSE: {test_rmse:.4f}")
print(f"Test R² Score: {test_r2:.4f}")
print(f"Test MAE: {test_mae:.4f}")
print("\nBest parameters:", search.best_params_)

# Plot actual vs predicted values
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Per Diem Rates')
plt.tight_layout()
plt.show()
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Test RMSE: 19.3252
Test R² Score: 0.5904
Test MAE: 11.0813

Best parameters: {'histgradientboostingregressor__max_iter': 1000, 'histgradientboostingregressor__max_depth': 5, 'histgradientboostingregressor__max_bins': 128, 'histg
```



Get predicted rates for each zip code in the US.

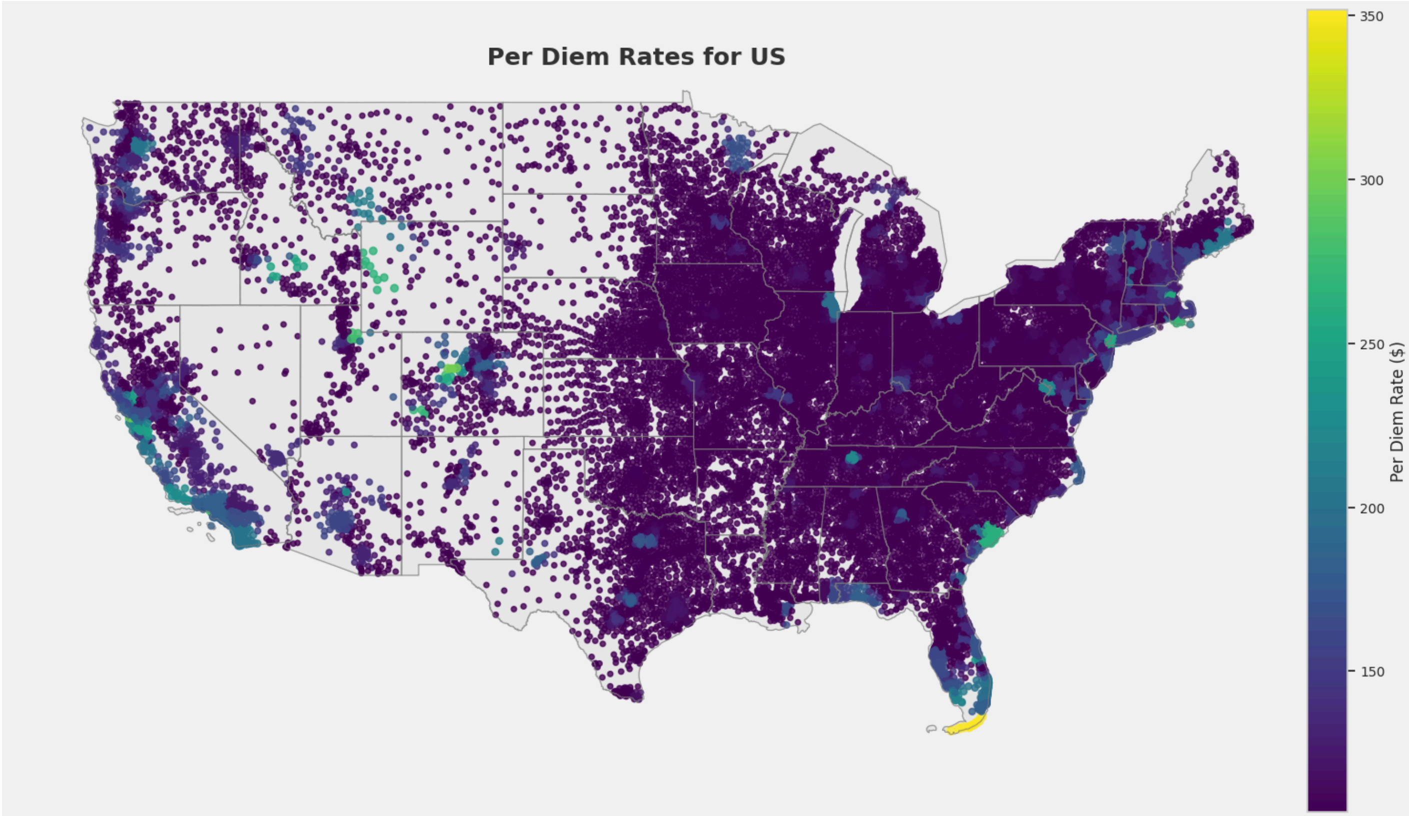
```
df["Predicted Rate"] = best_model.predict(X)
df
```

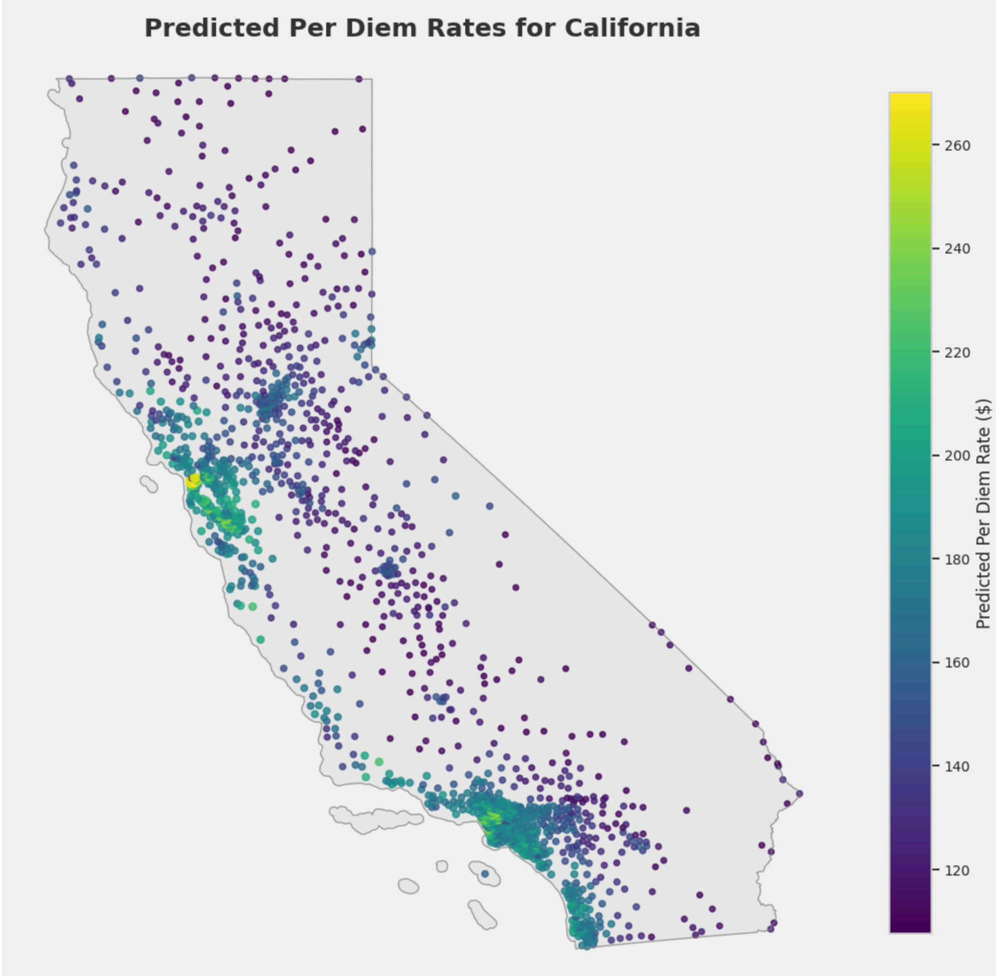
	zipcode	Per Diem Daily Rate	Rate Zone	House Price	City	State	Population Size	Population Density	County FIPS	County Name	isStandard	Predicted Rate
0	01001	122.0	Springfield	219508.964752	Agawam	MA	16136	551.7	25013	Hampden	False	125.052240
1	01002	146.0	Northampton	339812.737345	Amherst	MA	24726	179.3	25015	Hampshire	False	128.880528
2	01005	130.0	Worcester	244509.442046	Barre	MA	4786	42.8	25027	Worcester	False	113.923045
3	01007	146.0	Northampton	295540.622491	Belchertown	MA	15406	108.4	25015	Hampshire	False	122.846105
4	01008	122.0	Springfield	232702.865612	Blandford	MA	1324	8.4	25013	Hampden	False	113.709329
...	...	...	...	...	...	...	...	...	...	...	...	...
26176	99360	107.0	Standard Rate	348803.942100	Touchet	WA	1299	3.1	53071	Walla Walla	True	126.849263
26177	99361	107.0	Standard Rate	249922.063928	Waitsburg	WA	1800	3.1	53071	Walla Walla	True	114.619018
26178	99362	107.0	Standard Rate	270400.509440	Walla Walla	WA	42794	54.5	53071	Walla Walla	True	115.601047
26179	99402	107.0	Standard Rate	283674.534416	Asotin	WA	1628	1.9	53003	Asotin	True	121.882075

Render a choropleth heatmap of the predicted per diem rates of every zip code across the country with sufficient population data. We can also generate maps for individual states.

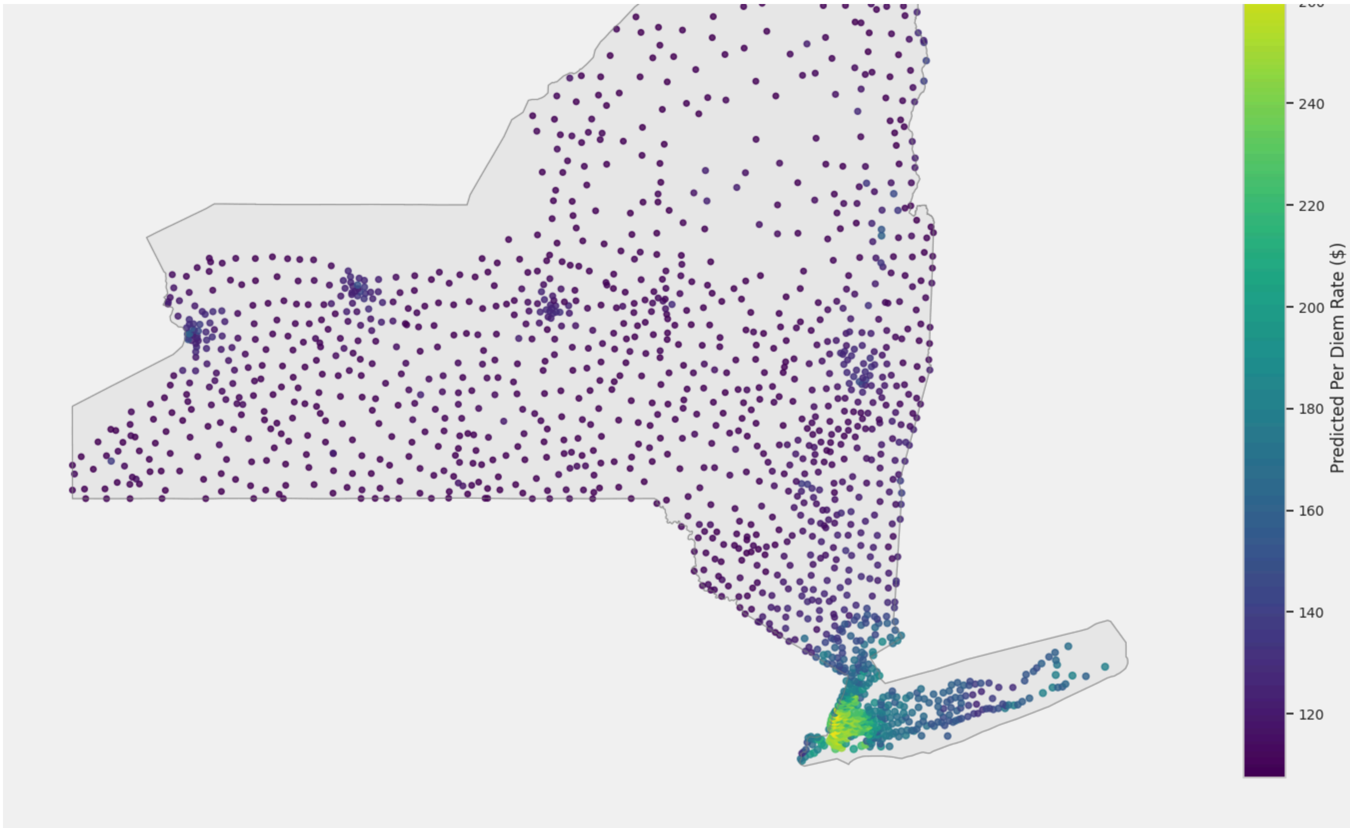
```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

def create_predicted_diem_map(diem_df, zipcode_shapefile_path, states_shapefile_path, state_name=None, output_file=None):
    # make sure all zip codes are strings with 5 digits
    diem_df['zipcode'] = diem_df['zipcode'].astype(str).str.zfill(5)

    # merge data into one easy to access data frame
    choropleth_data = zip_boundaries.merge(
        diem_df[['zipcode', 'Predicted Rate']],
        on='zipcode',
        how='inner'
    )

    # load in the US states. Exclude Alaska and Hawaii, which have per diem rates not calculated by the GSA (and are frankly
    # unable to fit on the map anyways)
    us_states = gpd.read_file(states_shapefile_path).to_crs(choropleth_data.crs)
    us_states = us_states[~us_states['NAME'].isin(['Alaska', 'Hawaii'])]

    # if we want, allow us to filter for a specific US state only
    if state_name:
        us_states = us_states[us_states['NAME'] == state_name]
        choropleth_data = gpd.clip(choropleth_data, us_states)

    # convert to easy to access centroids
    choropleth_data['geometry'] = choropleth_data.geometry.centroid

    # make background of the visualization
```

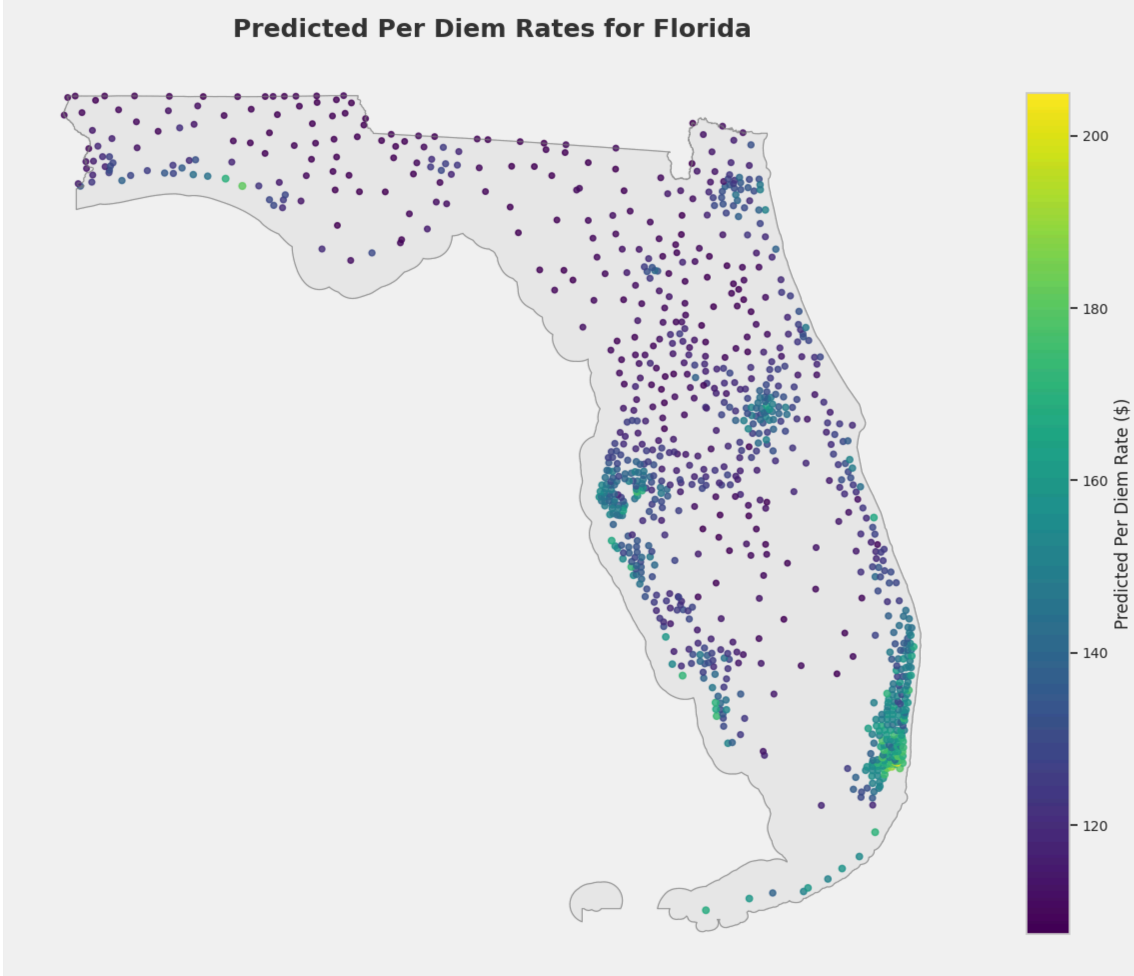
```
    fig, ax = plt.subplots(figsize=(15, 10))
    fig.patch.set_facecolor('#f0f0f0')
    ax.set_facecolor('#ffffff')

    # plot base layers of the visualizations
    us_states.plot(ax=ax, color='#e6e6e6', edgecolor='#cccccc', linewidth=0.7)
    us_states.boundary.plot(ax=ax, linewidth=0.8, color='gray', alpha=0.7)

    # plot the per diem daily rates
    scatter = choropleth_data.plot(
        ax=ax,
        markersize=(choropleth_data['Predicted Rate'] / 10) + 5, # marker size is proportional to predicted rate so we can more easily see non-standard rates
        column='Predicted Rate',
        cmap='viridis',
        alpha=0.8,
        legend=False
    )

    # color bar that accurately maps the spread of the data
    norm = Normalize(
        vmin=choropleth_data['Predicted Rate'].min(),
        vmax=choropleth_data['Predicted Rate'].max()
    )

    # make sure the plot has nice details
    sm = plt.cm.ScalarMappable(cmap='viridis', norm=norm)
    sm.set_array([])
    cbar = fig.colorbar(sm, ax=ax, fraction=0.03, pad=0.04)
    cbar.set_label('Predicted Per Diem Rate ($)', fontsize=12)
    cbar.ax.tick_params(labelsize=10)

    # stylize the plot
    title = f'Predicted Per Diem Rates for {state_name}' if state_name else 'Predicted Per Diem Rates for US'
    plt.title(title, fontsize=18, fontweight='bold', color='#333333')

    # set the proper bounds for the map (whether we are trying to get the map of a state or the etnire Continental U.S.)
    if state_name:
        minx, miny, maxx, maxy = us_states.total_bounds
        ax.set_xlim(minx - 50000, maxx + 50000)
        ax.set_ylim(miny - 50000, maxy + 50000)
    else:
        ax.set_xlim(-14284029, -7453304)
        ax.set_ylim(2717774, 6440332)

    ax.set_axis_off()
    plt.tight_layout()

    plt.savefig(output_file, dpi=300, bbox_inches='tight')

    return fig, ax

# create full US map
create_actual_diem_map(
    df,
    'zipcode_data/tl_2020_us_zcta520.shp',
    'states_data/tl_2022_us_state.shp',
    output_file='predicted_diem_map_us.png'
)

# create individual state maps at our own discretion
for state in ['California', 'New York', 'Florida', 'Michigan', "Kentucky"]:
    create_predicted_diem_map(
        df,
        'zipcode_data/tl_2020_us_zcta520.shp',
        'states_data/tl_2022_us_state.shp',
        state_name=state,
        output_file=f'predicted_diem_map_{state.lower().replace(" ", "_")}.png'
    )

plt.show()
```

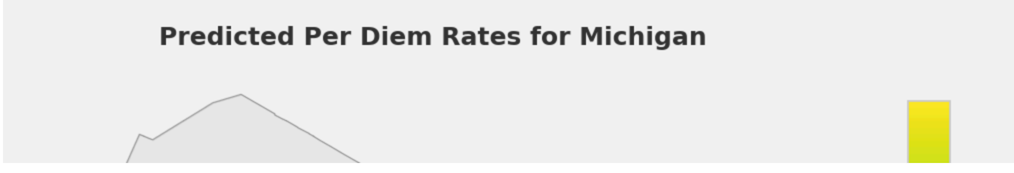

Per Diem Rates for US
350
300
250
200
150
Per Diem Rate ($)

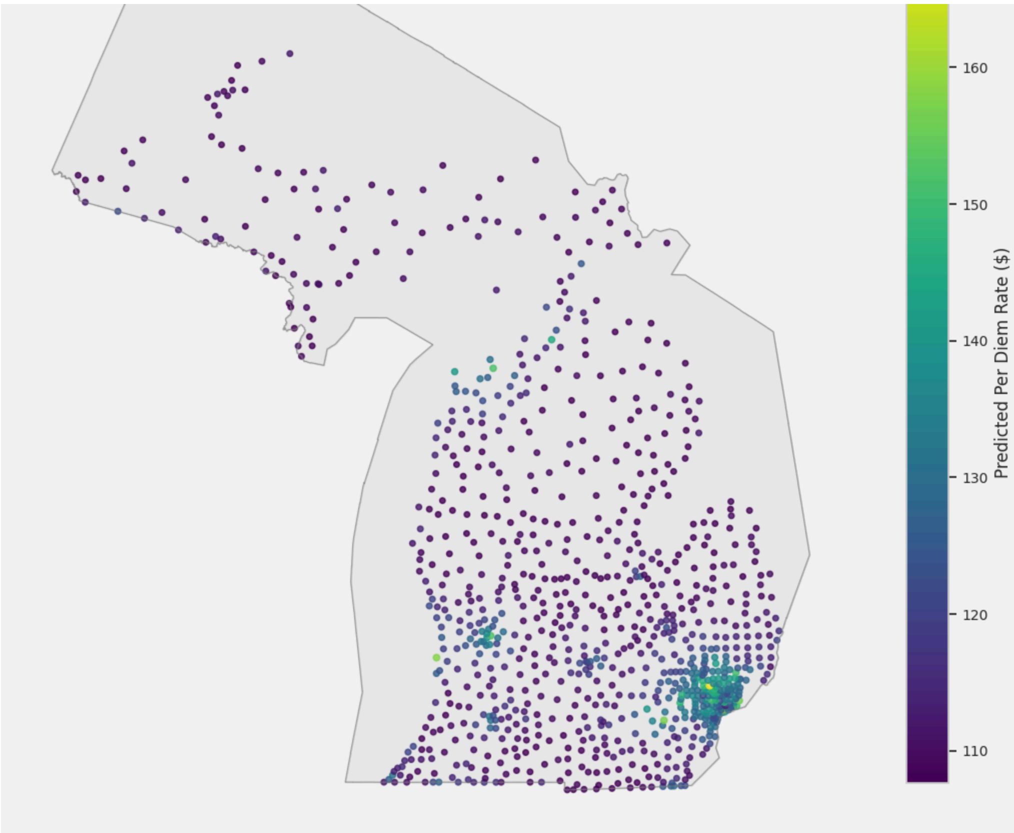

Predicted Per Diem Rates for California
260
240
220
200
180
160
140
120
Predicted Per Diem Rate ($)

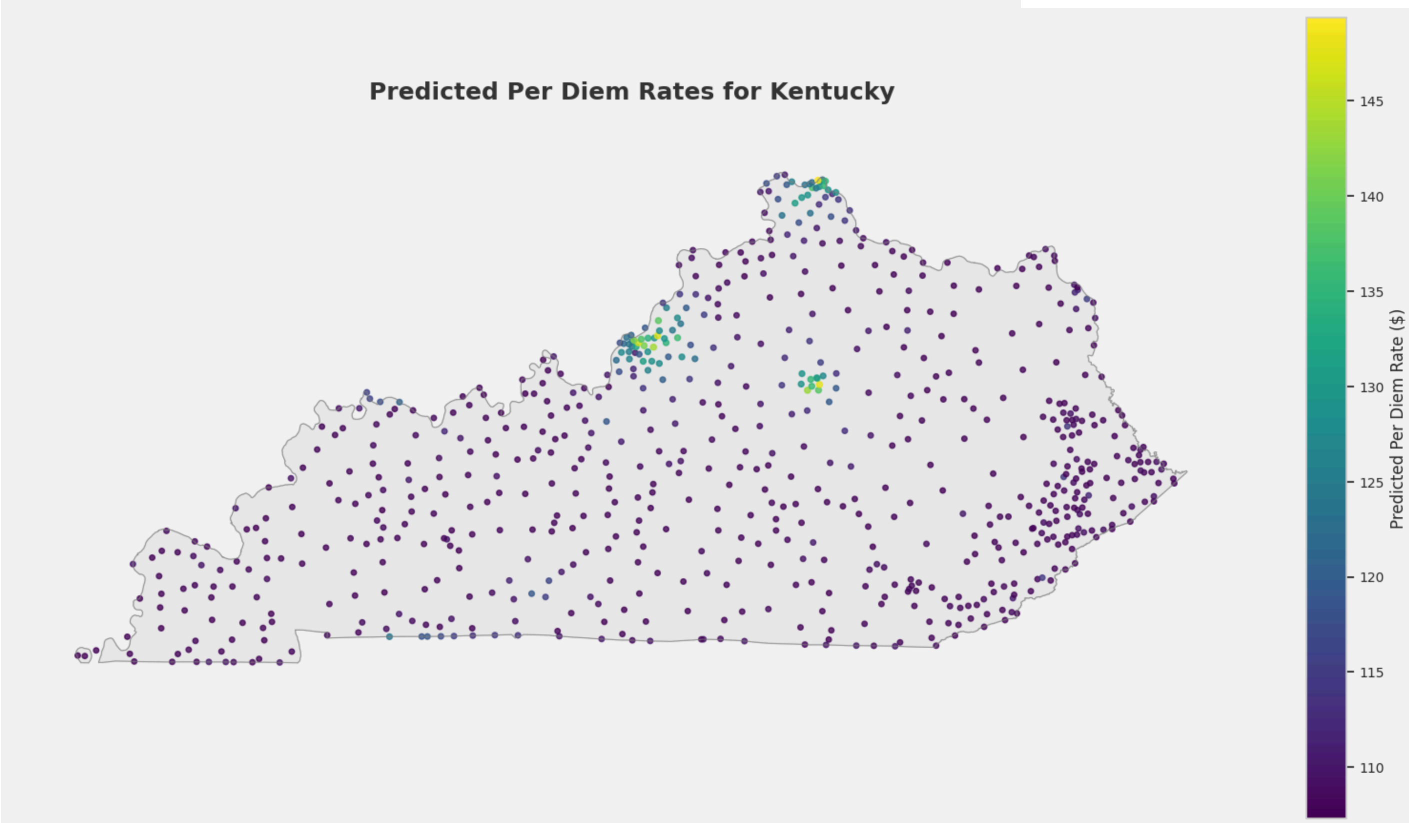

Predicted Per Diem Rates for New York
260


240
220
200
180
160
140
120
Predicted Per Diem Rate ($)


Predicted Per Diem Rates for Florida
200
180
160
140
120
Predicted Per Diem Rate ($)

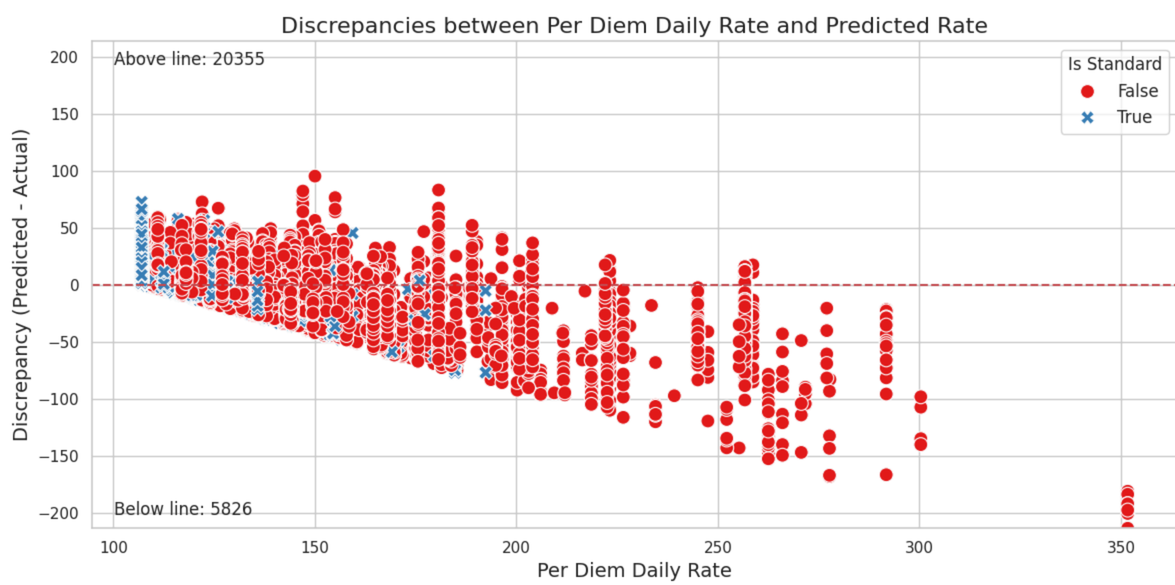

Predicted Per Diem Rates for Michigan


160
150
140
130
120
110
Predicted Per Diem Rate ($)


Predicted Per Diem Rates for Kentucky
145
140
135
130
125
120
115
110
Predicted Per Diem Rate ($)

Create a rendering of the discrepancies between predicted rates ancd actual rates.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# calculate the discrepancy between predicted rate and actual daily rate
df['Discrepancy'] = df['Predicted Rate'] - df['Per Diem Daily Rate']
sns.set(style="whitegrid")

plt.figure(figsize=(12, 6))
sns.scatterplot(
    x="Per Diem Daily Rate",
    y="Discrepancy",
    hue="isStandard",
    style="isStandard",
    palette="Set1",
    data=df,
    s=100
)

# add a baseline to show where there is no discrepancy
plt.axhline(y=0, color='r', linestyle='--')

# add labels and title
plt.title("Discrepancies between Per Diem Daily Rate and Predicted Rate", fontsize=16)
plt.xlabel("Per Diem Daily Rate", fontsize=14)
plt.ylabel("Discrepancy (Predicted - Actual)", fontsize=14)
plt.legend(title="Is Standard", fontsize=12)

# adjust y-axis to be symmetrical around 0
y_max = max(abs(df['Discrepancy'].min()), abs(df['Discrepancy'].max()))
plt.ylim(-y_max, y_max)

# count points above and below the line
above_count = sum(df['Discrepancy'] > 0)
below_count = sum(df['Discrepancy'] < 0)

# add text annotations for counts
plt.text(0.02, 0.98, f"Above line: {above_count}", transform=plt.gca().transAxes, verticalalignment='top')
plt.text(0.02, 0.02, f"Below line: {below_count}", transform=plt.gca().transAxes, verticalalignment='bottom')

plt.tight_layout()

# Show plot
plt.show()
```



Create graph showing the number of underpaid and overpaid employees of each type.

```
import plotly.express as px

# make columns in the dataframe corresponding to underpaid, correctly paid, and overpaid rates
df['Payment_Status'] = pd.cut(df['Discrepancy'],
                             bins=[-float('inf'), -1, 1, float('inf')],
                             labels=['Underpaid', 'Correctly Paid', 'Overpaid'])

# Create descriptive labels
df['Rate_Type'] = df['isStandard'].map({True: 'Standard Rate', False: 'Non-Standard Rate'})

# calculate proportions and averages
payment_proportions = df.groupby(['Rate_Type', 'Payment_Status']).agg(
    Count=('Discrepancy', 'size'),
    Mean_Discrepancy=('Discrepancy', 'mean')
).reset_index()
```