



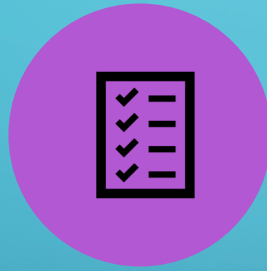
OPTIMIERUNG DER STANDORTWAHL FÜR ELEKTRO- LADESTATIONEN

FABIAN BEERLI & MORENO GALLO

INHALTSVERZEICHNIS



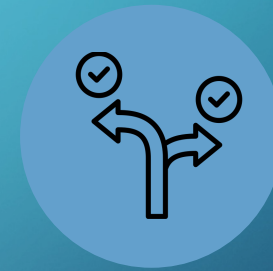
EINLEITUNG



METHODIK



ERGEBNISSE



AUSBLICK



EINLEITUNG

PROBLEMSTELLUNG

- Akku muss oft geladen werden
- Zeitverlust bei Anfahrt zu Ladestationen
- Ladestationen-Netz < Tankstellennetz



Zeitverlust minimieren




Effizienz der Ladeinfrastruktur steigern



EINLEITUNG

HYPOTHESEN

- 1) Eine Erhöhung der Sichtbarkeit und Zugänglichkeit von Ladestationen erhöht die Akzeptanz und Nutzung von Elektroautos in der Region.
- 2) Die Erhöhung der Anzahl von Schnellladestationen entlang der Hauptreiserouten verringert signifikant die Gesamtladezeit für Elektroautofahrer.



WIE BEEINFLUSSEN GEOGRAFISCHE UND DEMOGRAFISCHE FAKTOREN DIE VERTEILUNG UND NUTZUNG VON ELEKTROAUTO-LADESTATIONEN IN WINTERTHUR?

FORSCHUNGSFRAGE

METHODIK & ERGEBNISSE



METHODIK

- Beschreibung der Daten
- Angewandte Methoden und Tools
- Vorgehen erläutern
- Ergebnisse präsentieren & interpretieren



METHODIK

BESCHREIBUNG DER DATEN

- OSM Daten für die aktuell vorhandenen Ladestationen in Winterthur
- GWR Daten des Kanton Zürich für alle Wohn- und Gewerbsgebäude
- Winterthur mit QGIS ausgefiltert
- Ergebnis in QGIS dargestellt



METHODIK

ANGEWANDTE METHODEN UND TOOLS

- PG Admin
- Python und VS Code
- Clustering
- QGIS

VORGEHEN



METHODIK

```
# Verbindung zur PostgreSQL-Datenbank herstellen
engine = create_engine('postgresql://postgres:changeit@localhost:5432/osm')
```

```
# SQL-Abfrage für Ladestationen ausführen und in DataFrame laden
query_charging_station = "SELECT * FROM planet_osm_point_winterthur WHERE amenity = 'charging_station'"
df_charging_station = pd.read_sql_query(query_charging_station, engine)
df_charging_station.head()
```

```
# Verbindung schließen
engine.dispose()
```

```
# Verbindung zur PostgreSQL-Datenbank herstellen
engine = create_engine('postgresql://postgres:changeit@localhost:5432/gwr')
```

```
# SQL-Abfrage für Wohngebäude ausführen und in DataFrame laden
query_residential = "SELECT * FROM buildings_winterthur WHERE buildingcategory IN (1020, 1030)"
df_residential = pd.read_sql_query(query_residential, engine)
df_residential.head()
```



VORGEHEN

```
# SQL-Abfrage für gewerbliche Gebäude ausführen und in DataFrame laden
query_commercial = "SELECT * FROM buildings_winterthur WHERE buildingcategory IN (1060, 1080)"
df_commercial = pd.read_sql_query(query_commercial, engine)
df_commercial.head()
```

```
# Verbindung schließen
engine.dispose()
```

```
# Konvertiere das WKB-Format in ein Shapely-Geometry-Objekt für Wohngebäude, gewerbliche Gebäude und Ladestationen
df_residential['geometry'] = df_residential['wkb_geometry'].apply(wkb.loads)
df_commercial['geometry'] = df_commercial['wkb_geometry'].apply(wkb.loads)
df_charging_station['geometry'] = df_charging_station['wkb_geometry'].apply(wkb.loads)
```

```
# Konvertiere das DataFrame in ein GeoDataFrame
gdf_residential = gpd.GeoDataFrame(df_residential)
gdf_commercial = gpd.GeoDataFrame(df_commercial)
gdf_charging_station = gpd.GeoDataFrame(df_charging_station)
```

```
# Definiere das CRS jedes GeoDataFrame
gdf_charging_station.crs = 'EPSG:3857'
gdf_residential.crs = 'EPSG:2056'
gdf_commercial.crs = 'EPSG:2056'
```




VORGEHEN

```
# Konvertiere die Geometrien der Ladestationen in das CRS der Wohngebäude und gewerblichen Gebäude
gdf_charging_station['geometry'] = gdf_charging_station['geometry'].to_crs('EPSG:2056')
```

```
# Radius für den Puffer (in Metern) festlegen
buffer_radius = 500
```

```
# Funktion definieren, um einen Puffer um jede Ladestation zu erstellen
def create_buffer(geometry, radius):
    return geometry.buffer(radius)
```

```
# Einzelne Puffer für jede Ladestation erstellen
gdf_charging_station['buffer'] = gdf_charging_station['geometry'].apply(lambda x: create_buffer(x, buffer_radius))
```

```
# Einzelne Puffer für jedes gewerbliche Gebäude erstellen
gdf_commercial['buffer'] = gdf_commercial['geometry'].apply(lambda x: create_buffer(x, buffer_radius))
```

```
# Funktion definieren, um zu zählen, wie viele Wohngebäude sich innerhalb eines Puffers befinden
count = 0
def count_residential_buildings_within_buffer(buffer, residential_buildings):
    global count
    count += 1
    print(count)
    return len([building for building in residential_buildings if buffer.contains(building)])
```




METHODIK

VORGEHEN

```
# Funktion definieren, um zu zählen, wie viele Ladestationen sich innerhalb eines Puffers befinden
def count_charging_stations_within_buffer(buffer, charging_stations):
    return len([station for station in charging_stations if buffer.contains(station)])
```

```
# Funktion zur Berechnung des Verhältnisses definieren
def calculate_ratio(num_residential_buildings_nearby, num_charging_stations_nearby):
    if num_charging_stations_nearby == 0:
        return num_residential_buildings_nearby
    else:
        return num_residential_buildings_nearby / (num_charging_stations_nearby + 1)
```

```
# Anzahl der Wohngebäude und Ladestationen in der Nähe jeder Ladestation zählen
gdf_charging_station['num_residential_buildings_nearby'] = gdf_charging_station.apply(lambda row: count_residential_buildings_within_buffer(row['buffer'], gdf_residential['geometry']), axis=1)
gdf_charging_station['num_charging_stations_nearby'] = gdf_charging_station.apply(lambda row: count_charging_stations_within_buffer(row['buffer'], gdf_charging_station['geometry']), axis=1)
gdf_charging_station['ratio'] = gdf_charging_station.apply(lambda row: calculate_ratio(row['num_residential_buildings_nearby'], row['num_charging_stations_nearby']), axis=1)
```

```
# Anzahl der Wohngebäude und Ladestationen in der Nähe jedes gewerblichen Gebäudes zählen
gdf_commercial['num_residential_buildings_nearby'] = gdf_commercial.apply(lambda row: count_residential_buildings_within_buffer(row['buffer'], gdf_residential['geometry']), axis=1)
gdf_commercial['num_charging_stations_nearby'] = gdf_commercial.apply(lambda row: count_charging_stations_within_buffer(row['buffer'], gdf_charging_station['geometry']), axis=1)
gdf_commercial['ratio'] = gdf_commercial.apply(lambda row: calculate_ratio(row['num_residential_buildings_nearby'], row['num_charging_stations_nearby']), axis=1)
```

```
# Ergebnisse anzeigen und weiterverarbeiten
print("Ladestationen:")
print(gdf_charging_station)

print("\nWohngebäude:")
print(gdf_residential)

print("\nGewerbliche Gebäude:")
print(gdf_commercial)
```



VORGEHEN

```
# Vorbereiten der Daten für das Clustering
X = gdf_charging_station[['num_residential_buildings_nearby', 'num_charging_stations_nearby', 'ratio']]
```

```
# Initialisieren und Anpassen des K-Means-Modells
kmeans_model = KMeans(n_clusters=2)
kmeans_model.fit(X)
```

```
# Vorhersagen der Cluster-Zuweisungen für die gewerblichen Gebäude
gdf_commercial['cluster_label'] = kmeans_model.predict(gdf_commercial[['num_residential_buildings_nearby', 'num_charging_stations_nearby', 'ratio']])
```

```
# Filtere Einträge mit Cluster 0 aus dem gdf_commercial
gdf_commercial_filtered = gdf_commercial[gdf_commercial['cluster_label'] != 0]
```

```
# Sortiere das gefilterte GeoDataFrame nach dem Feld "ratio" absteigend
gdf_commercial_sorted = gdf_commercial_filtered.sort_values(by='ratio', ascending=False)
```

```
# Überprüfe, ob ein Punkt in einem der vorhandenen Buffer liegt
def check_point_in_buffers(point, buffer_list):
    for buffer in buffer_list:
        if buffer.contains(point):
            return True
    return False
```




VORGEHEN

```
# Erstelle eine leere Liste, um die eindeutigen Buffer zu speichern  
unique_buffers = []
```

```
# Filtere die top 10 gewerblichen Gebäude nach den eindeutigen Buffern  
unique_commercial_top_10 = []
```

```
for index, row in gdf_commercial_sorted.iterrows():  
    # Überprüfe, ob der Punkt im gleichen Buffer wie einer der bereits ausgewählten Punkte liegt  
    point_in_buffer = check_point_in_buffers(row['geometry'], unique_buffers)  
  
    # Wenn der Punkt nicht im gleichen Buffer wie einer der bereits ausgewählten Punkte liegt, füge ihn zur Liste hinzu  
    if not point_in_buffer:  
        unique_commercial_top_10.append(row)  
        unique_buffers.append(row['buffer'])
```

```
# Konvertiere die eindeutigen gewerblichen Gebäude in ein GeoDataFrame  
gdf_unique_commercial_top_10 = gpd.GeoDataFrame(unique_commercial_top_10)
```

```
# New potential charging stations  
# Ergebnisse anzeigen oder weiterverarbeiten  
print("Die eindeutigen gewerblichen Gebäude:")  
print(gdf_unique_commercial_top_10.head(10))
```




METHODIK

VORGEHEN

```
# Verbindung zur PostgreSQL-Datenbank herstellen
engine = create_engine('postgresql://postgres:changeit@localhost:5432/gwr')

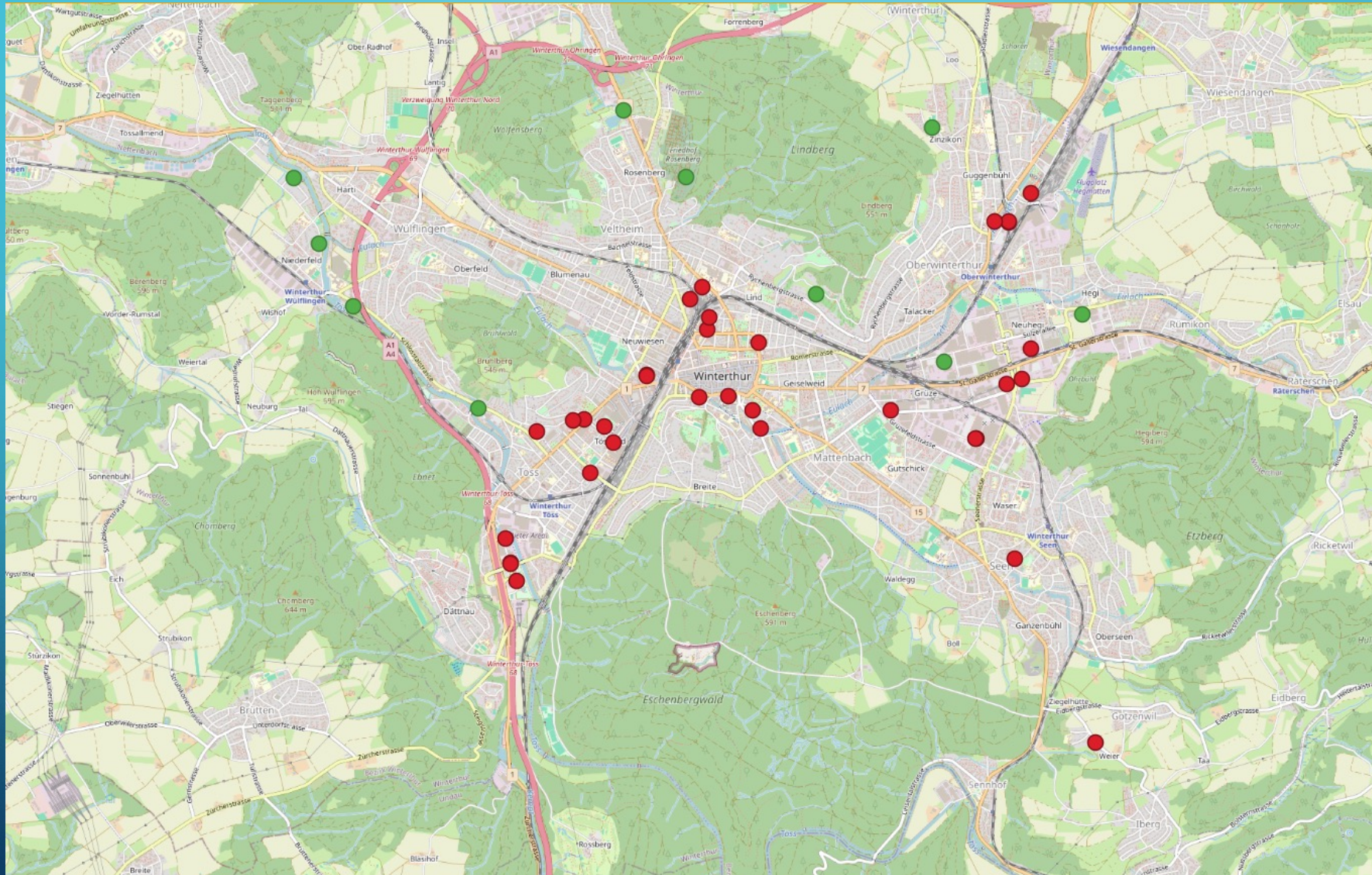
# Schreibe den GeoDataFrame in die PostgreSQL-Tabelle 'new_table' mit der Geometriespalte 'wkb_geometry'
gdf_unique_commercial_top_10.head(10).to_postgis(name='new_charging_station', con=engine, if_exists='replace', index=False)

# Verbindung schließen
engine.dispose()
```


ERGEBNIS



ERGEBNISSE





ERGEBNISSE

ERGEBNIS INTERPRETATION

- Die roten Punkte stellen die bereits vorhandenen Ladestationen dar
- Durch unsere Analyse stellen die grünen Punkte die möglichen Gebäude und Standorte dar, welche sich sehr gut für weitere Ladestationen eignen
- In der Region um Wülflingen hatte es bisher keine Ladestationen und unser Model konnte dies korrekt identifizieren und in dieser Region mögliche Standorte für Elektro-Ladestationen grün markieren



ERGEBNISSE

FORSCHUNGSFRAGE BEANTWORTEN

- Geografische und demografische Merkmale beeinflussen die Standortwahl
- Bevölkerungsdichte hat eine hohe Korrelation mit der Ladestation-Verfügbarkeit
- Gebäudeanzahl & -lage ist wichtig für die Standortwahl
- Planungsbedarf für strategische Standortoptimierung erforderlich



ERGEBNISSE

HYPOTHESE 1

EINE ERHÖHUNG DER SICHTBARKEIT UND ZUGÄNGLICHKEIT VON LADESTATIONEN ERHÖHT DIE AKZEPTANZ UND NUTZUNG VON ELEKTROAUTOS IN DER REGION

Beispiel Wülflingen:

Einwohner könnten sich den Umstieg überlegen, da sie miterleben, wie sich das Netz an Elektro-Stationen in ihrer Umgebung vergrößert.



ERGEBNISSE

HYPOTHESE 2

DIE ERHÖHUNG DER ANZAHL VON SCHNELLLADESTATIONEN ENTLANG DER FAHRSTRASSEN VERRINGERT SIGNIFIKANT DIE GESAMTLADEZEIT FÜR ELEKTROAUTOFAHRER

1. Die Erhöhung der Anzahl von Schnellladestationen entlang der Fahrstraßen ermöglicht Elektroautofahrern mehr Gelegenheiten zum Laden ihrer Fahrzeuge, was die Wartezeiten an den Ladestationen verringert und somit die Gesamtladezeit signifikant reduziert.
2. Durch den Ausbau der Schnellladeinfrastruktur entlang der Fahrstraßen wird die Ladezeit effizienter verteilt, da Elektroautofahrer kürzere Strecken zwischen den Ladestationen zurücklegen müssen und somit ihre Fahrten weniger häufig unterbrechen, was insgesamt zu einer signifikanten Reduktion der Gesamtladezeit führt.



AUSBLICK

SCHLUSSFOLGERUNG

PROJEKTLIMITATIONEN / HANDLUNGSEMPFEHLUNGEN / AUSBLICK

PROJEKTLIMITATIONEN



AUSBLICK

- Anzahl Ladestationen pro charging point
 - Wenn es an einer aktuell vorhandenen Standort nur 1 Ladestation gibt ist auch dieser Standort ausbaufähig auf mehrere Ladestationen, falls im Umkreis viel Fahrverkehr besteht
- Anzahl Mitarbeiter pro Gebäude

AUSBLICK AUF ZUKÜNFTIGE FORSCHUNG



AUSBLICK

- Integration von Ladestationen in bestehende Verkehrs- und Energieinfrastrukturprojekte
- Internationale Vergleiche zur Förderung der nachhaltigen Mobilität
- Untersuchung der Reaktionen von Verbrauchern auf verschiedene Anreize

A decorative graphic on the left side of the slide, consisting of a network of light blue lines and small circles, resembling a circuit board or a stylized tree structure, set against a blue gradient background.

VIELEN DANK FÜR EURE
AUFMERKSAMKEIT