*Используемый датасет: https://www.kaggle.com/datasets/sarfarazmulla/google-data-analytics-capstone-cyclistic*

# 1. Разведочный анализ данных

Разведочный анализ данных – анализ основных свойств данных, нахождение в них общих закономерностей, распределений и аномалий, построение начальных моделей, зачастую с использованием инструментов визуализации.

Проведём разведочный анализ датасета с использованием фреймворка PySpark. Apache Spark — это платформа параллельной обработки с открытым кодом, которая поддерживает обработку в памяти, чтобы повысить производительность приложений, анализирующих большие данные. Решения для работы с большими данными предназначены для обработки данных со слишком большим объемом или сложностью для традиционных баз данных. Spark обрабатывает большие объемы данных в памяти, что намного быстрее, чем альтернативная обработка с использованием диска.

PySpark – это API Apache Spark для Python, позволяющий использовать возможно по обработке данных Spark в удобной среде исполнения Python. При этом запускается сервер Spark, осуществляющий обработку данных, взаимодействие с которым происходит через SparkContext в Python-программе. При этом пользователю доступны все возможности Spark, включая чтение и запись данных на диск, SQL-подобные манипуляции над данными (Spark SQL) и использование различных алгоритмов анализа данных и машинного обучения (Spark ML).

Работа выполнялась в Jupyter Notebook, сервер которого запускался в виде Docker-контейнера на основе образа jupyter/pyspark-notebook.

## 1.1 Обзор датасета

Данный датасет содержит данные о поездках на велосипедах, предоставленные компанией Cyclistics, занимающейся арендой велосипедов. Датасет содержит такие данные как точки начала и конца поездки, дата и время, длительность поездки. Пример содержимого датасета представлен на рисунках 1 и 2.

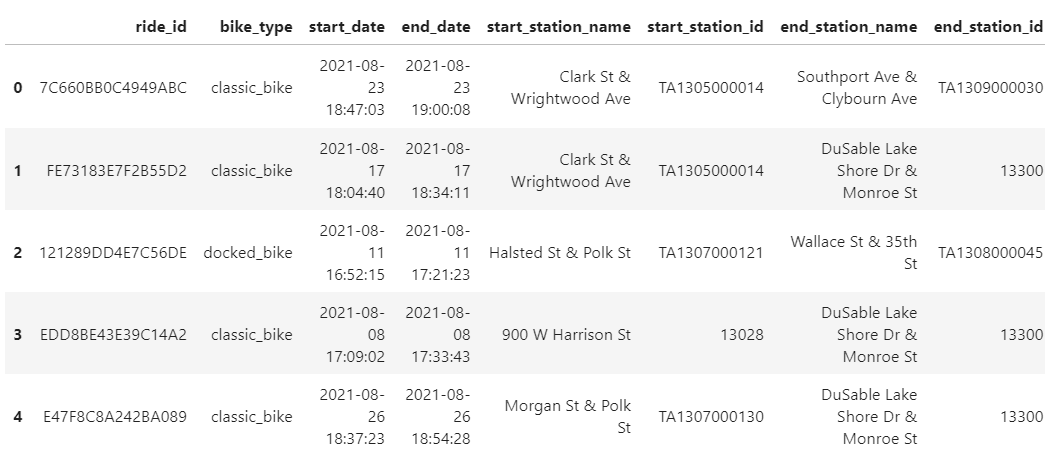


Рисунок 1. Начало датасета



Рисунок 2. Конец датасета

Рассмотрим подробнее каждую колонку:

* ride\_id - уникальный идентификатор поездки, *string*
* bike\_type - тип велосипеда, *string*
* start\_date и end\_date - временные метки начала и окончания поездки, *string*
* start\_station\_name и end\_station\_name - названия начальной и конечной станции, *string*
* start\_station\_id и end\_station\_id - идентификаторы начальной и конечной станции, *string*
* start\_lat, start\_lng, end\_lat и end\_lng - координаты точек начала и окончания поездки, *double*
* member\_casual - имеет ли пользователь членство компании Cyclistics, *string*
* duration - длительность поездки, *string*
* day\_of\_week, month - день недели и месяц, когда была совершена поездка, *string*
* distance - расстояние между начальной и конечной точками, *double*

## 1.2 Очистка и подготовка датасета

Очевидно, некоторые из колонок имеют неподходящий для них тип – даты записаны в виде строк и имеют два различных формата; длительность указана в виде строки в формате “h:mm:ss”.

Для преобразования дат к нужному формату воспользуемся функцией parse модуля dateutil, преобразовав её к udf. Длительность поездки приведём к секундам, реализовав собственную функцию.

parse\_date\_udf = udf(parse, returnType=TimestampType())

@udf(returnType=IntegerType())

def get\_total\_seconds(time):

h, m, s = time.split(':')

return int(h) \* 3600 + int(m) \* 60 + int(s)

Посмотрим на описательные характеристики датасета:

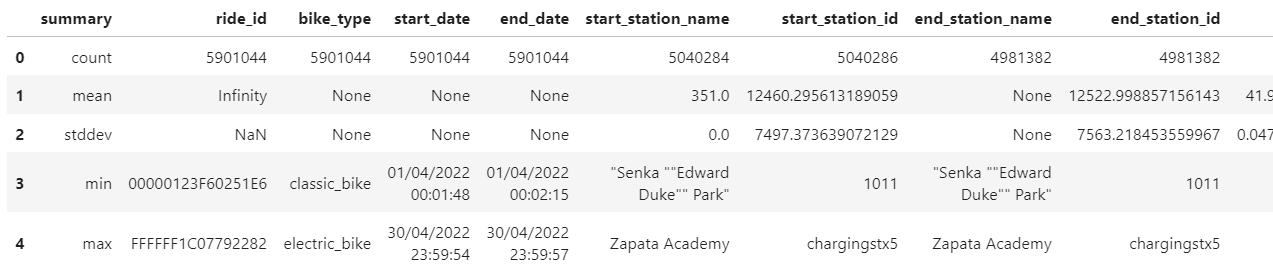


Рисунок 3. Описательные характеристики (начало)

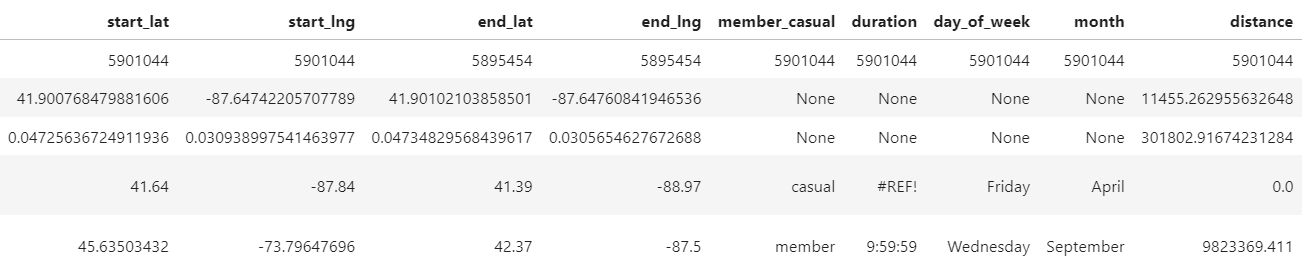


Рисунок 4. Описательные характеристики (продолжение)

Можно заметить нереалистичные значения некоторых характеристик – например, расстояние более чем 9 тыс. км. Как видно из следующего рисунка, у таких поездок не указана точка назначения.

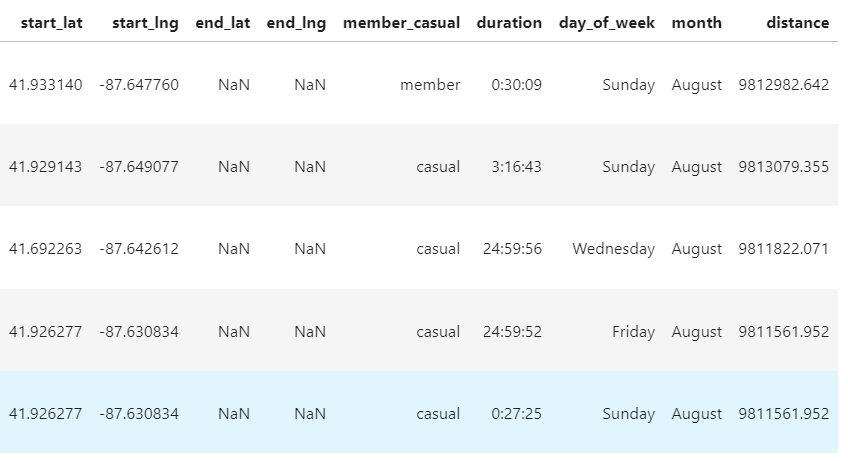


Рисунок 5. Поездки без точки назначения

Удалим такие строки при помощи dropna а также применим ранее объявленные функции для приведения времени и даты к правильным форматам.

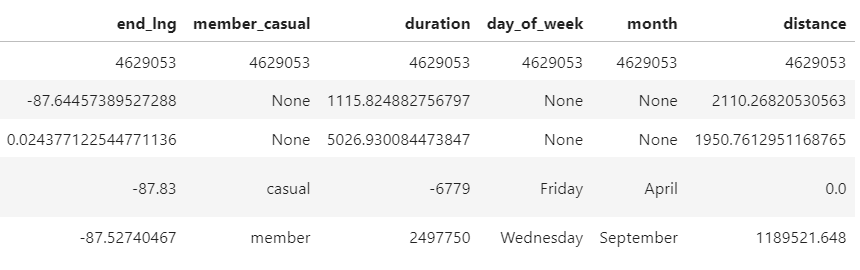


Рисунок 6. Новые храктеристики

Максимальное значение уменьшилось, но оно всё ещё превышает 1 тыс. км. Определим строки с таким значением.

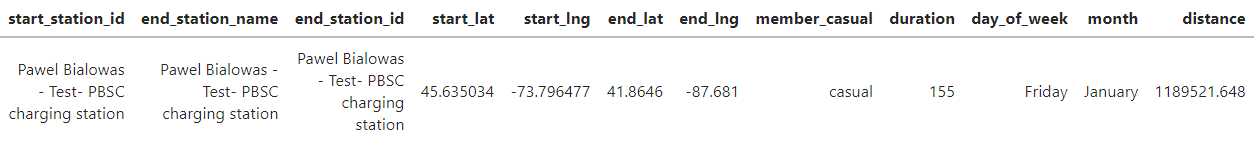


Рисунок 7. Выброс

Судя по координатам, этот человек доехал от Монреаля до Чикаго, причём сделал это всего за 2.5 минуты. Вероятно, это тестовые данные, которые остались в базе данных компании (об этом также говорит Test в названии станции). Удалим этот и подобные выбросы используя расстояние Махаланобиса.

columns = ['start\_lat', 'start\_lng', 'end\_lat', 'end\_lng']

means\_devs = {column: (float(desc[column][1]), float(desc[column][2])) for column in columns}

cols = []

for column in columns:

mean, dev = means\_devs[column]

cols.append( f.abs((col(column) - mean) / dev) )

df = df.withColumn('mah', f.greatest(\*cols)) # Mahalanobis distance

df = df.where(df.mah < 10).drop(df.mah) # 10 standard deviations away

Также можно заметить, что минимальное значение длительности поездки – отрицательное. Отфильтруем поездки с нулевой или отрицательной длительностью.

Некоторые поездки имеют нулевую дистанцию при адекватном времени поездки. Эти поездки, также, начинаются и заканчиваются на одной и той же станции. Пометим такие поездки как “круговые” в новой колонке.

df = df.withColumn('round\_trip', when(df.distance == 0.0, 1).otherwise(0))

Колонки bike\_type и member\_casual можно преобразовать к бинарным признакам:

df = (

df.withColumn('electric\_bike', when(df.bike\_type == 'electric\_bike', 1).otherwise(0))

.withColumn('docked\_bike', when(df.bike\_type == 'docked\_bike', 1).otherwise(0))

.withColumn('member', when(df.member\_casual == 'member', 1).otherwise(0))

.drop('bike\_type', 'member\_casual', 'ride\_id', 'start\_station\_name', 'start\_station\_id', 'end\_station\_name', 'end\_station\_id')

)

После преобразований и удаления ненужных для дальнейшего анализа колонок получаем следующий датасет.

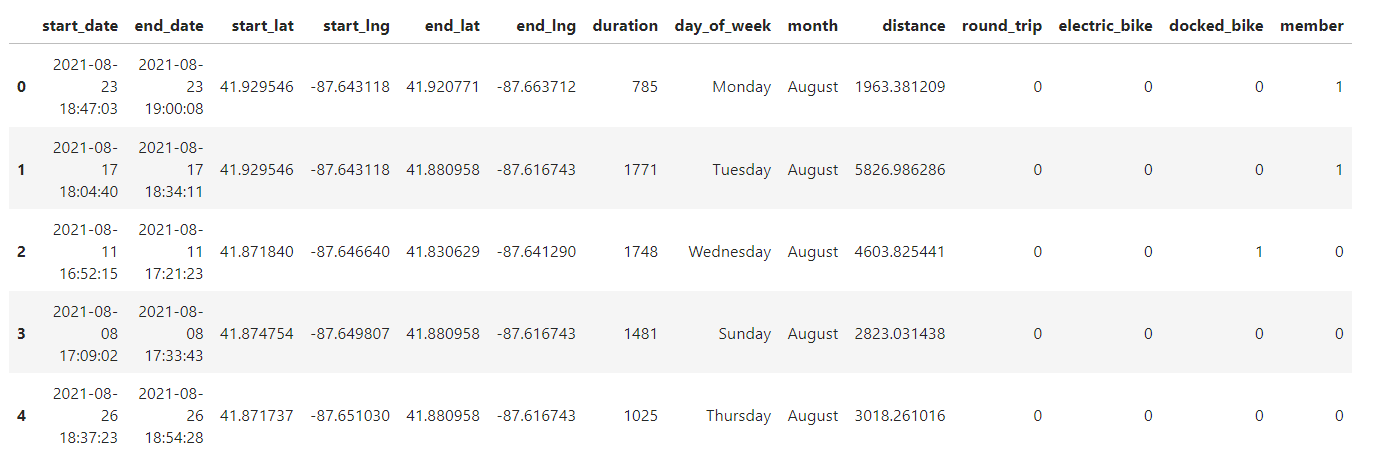


Рисунок 8. Результирующий датасет

## 1.3 Построение диаграмм

Построим круговые диаграммы для типа поездки, типа велосипеда и типа членства.

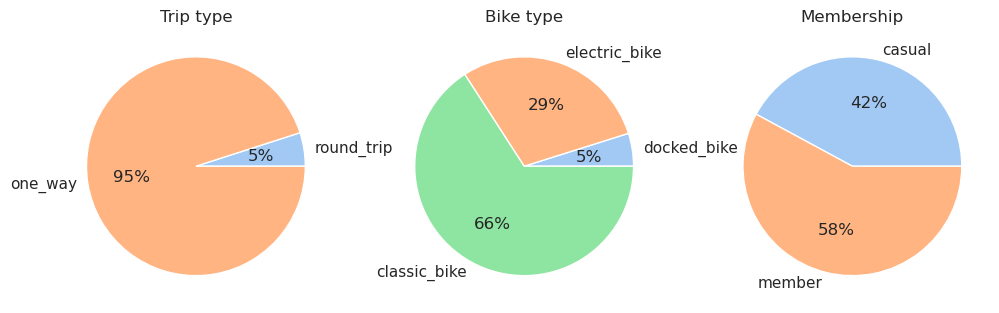


Рисунок 9. Тип поездки, тип велосипеда и тип членства

По диаграммам видно, что большинство совершаемых поездок – в одну сторону; наибольшей популярностью пользуются классические велосипеды, наименьшей – docked велосипеды; членство компании имеет чуть больше половины пользователей велосипедов.

Построим диаграммы распределения поездок по дням недели и месяцам.

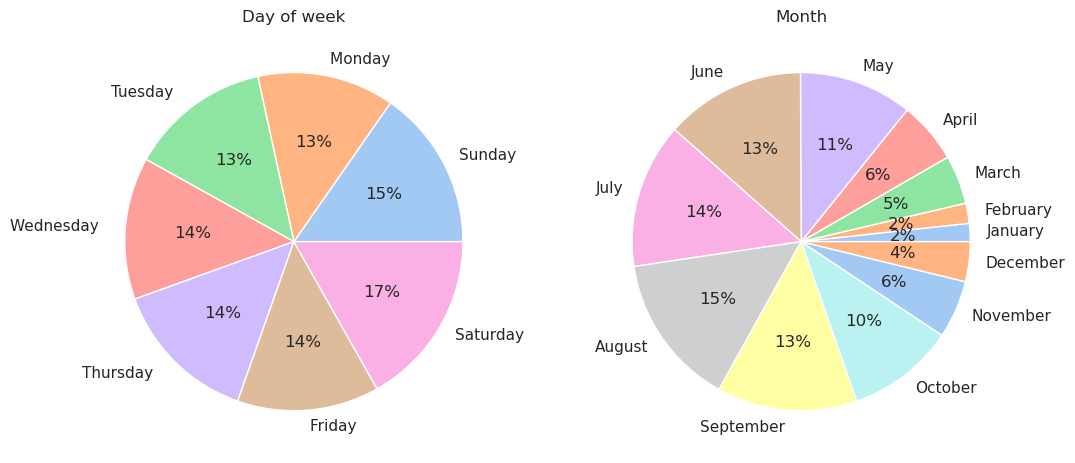


Рисунок 10. Распределение по дням недели и месяцам

Можно заметить, что день недели лишь незначительно влияет на количество поездок. С другой стороны, месяц оказывает сильное влияние на количество поездок – более тёплые месяца лучше способствуют поездкам на велосипеде. Построим то же распределение в виде гистограммы.

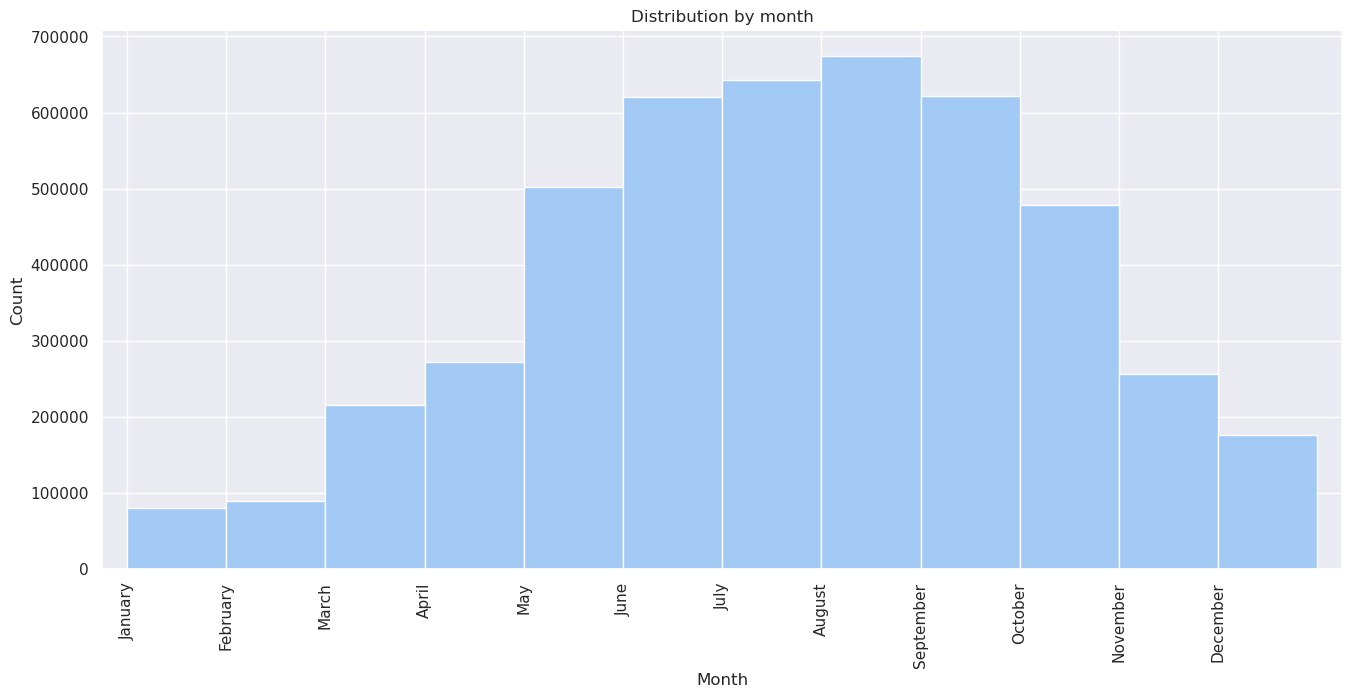


Рисунок . Распределение поездок по месяцам

Из такого графика сразу видно, что количество поездок распределено по нормальному закону распределения.

Так как датасет содержит географические данные, логичным шагом является отметить их на карте. Было определено, что все поездки итогового датасета происходили в Чикаго. Воспользуемся модулем geopandas для изображения очертаний города на графике. Отметим точки начала поездок синими знаками “+”, а точки окончания – оранжевыми знаками “x”.

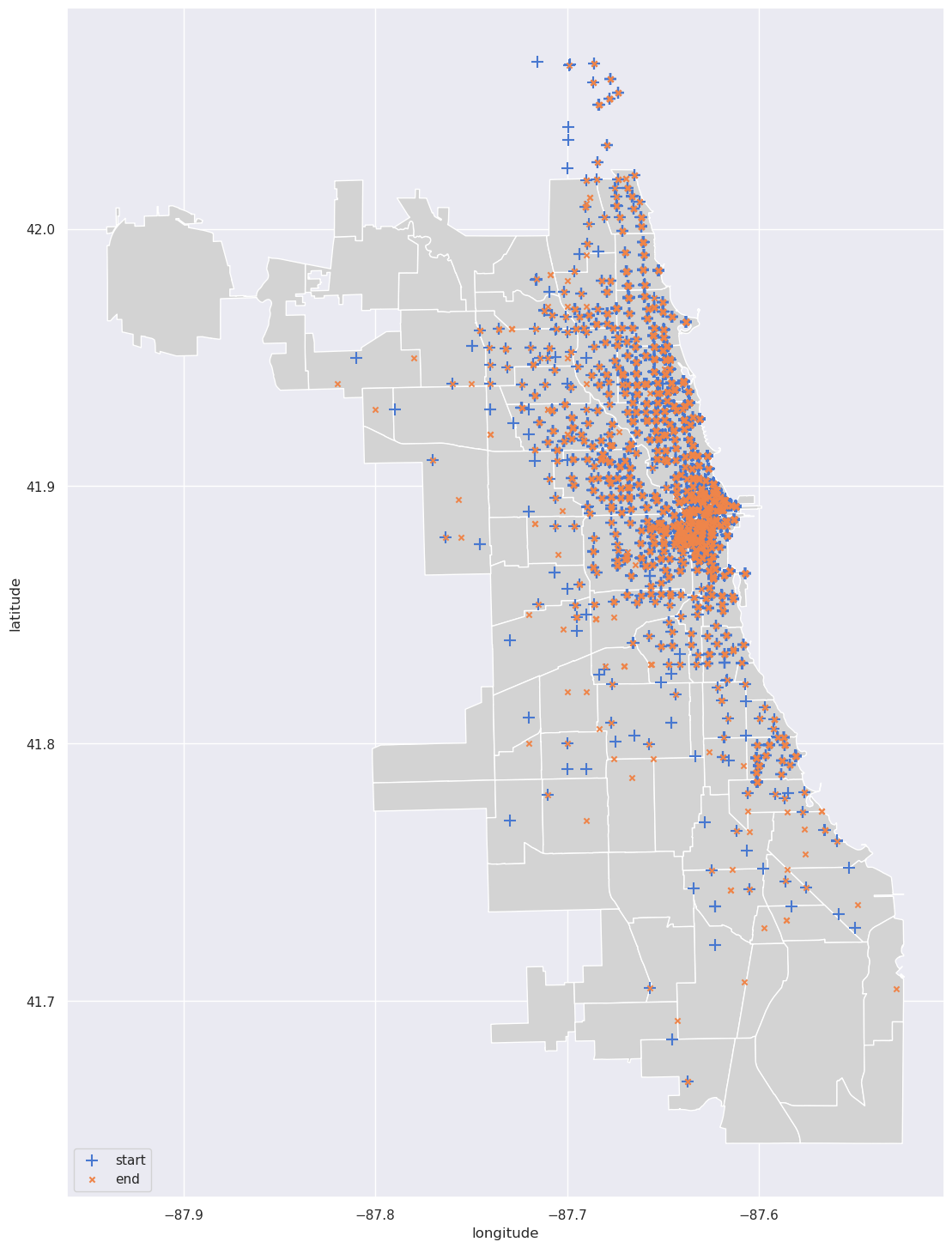


Рисунок . Точки на карте города Чикаго

Большая часть поездок сконцентрирована в северо-восточной – восточной части города, в то время как южный и западный концы города практически не имеют у себя отметок.

Таким образом, был проведён разведочный анализ датасета, а также датасет был очищен и подготовлен к дальнейшему исследованию.