# Capstone Project

Docks availability prediction (Bicing)


Sherezade Fuentes Julian

Maria Monserrat Martínez

Demian Moschin Pierucci

Núria Romero Herreros

Roger Salvador Guasch

Profesores:
Mariona Carós Roca
Pere Gilabert Roca

# Contenido

- **Limpieza y preprocesamiento de los datos.**
  *¿Realmente debemos usar los datos del año 2020?*
- **Variables extras añadidas al dataset base.**
  *¿Qué puede afectar a la predicción de huecos disponibles en una estación del Bicing?*
- **Selección de variables y modelos predictivos.**
  *Balance entre una precisión óptima y optimización de la complejidad.*
- **Insights de los datos.**
  *Identificar patrones de uso de las estaciones del Bicing.*
- **Web App**
  *Visualización de las predicciones*

# Dataset and additional features

## First dataset and extra variables

| Variable | Description |
|---|---|
| station_id | Bicing station id |
| year | year |
| month | month |
| day | day |
| hour | hour |
| ctx-1 | mean percentage of docks available 1 hour before |
| ctx-2 | mean percentage of docks available 2 hours before |
| ctx-3 | mean percentage of docks available 3 hours before |
| ctx-4 | mean percentage of docks available 4 hours before |

**Location related features**

**Time related features**

Open Data **BCN**

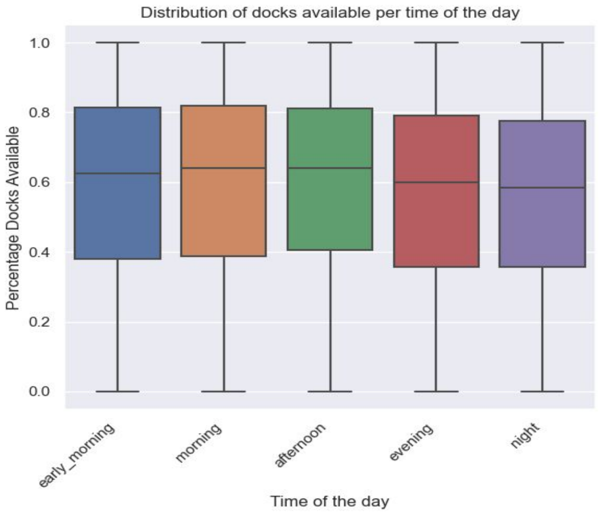
| Variable | Description |
|---|---|
| lat | latitude |
| lon | longitude |
| altitude | altitude |
| post_code | postal code |
| neighborhood | neighborhood |
| capacity | capacity |
| nearby_stations | number of Bicing nearby stations (<300m) |
| nearby_stations_list | list of Bicing nearby stations (<300m) |
| nearby_avg_ctx1 | mean dock availability of the nearby stations in the previous hour |
| near_transport | is near a public transport station (<200) |
| near_college | is near a college (<300) |
| nearby_colleges | number of nearby colleges (<300) |
| near_library | is near a library (<200) |
| near_museum | is near a museum (<200) |
| near_theater | is near a theater or cinema (<200) |
| near_bar | is near a bar or club (<200) |
| nearby_bars | number of nearby bars or clubs (<200) |
| day_info | day of the week |
| is_weekend | is weekend |
| is_holiday | is holiday |
| is_not_workday | is non working day |
| hour_info | moment of the day |
| season_info | season |

Issues with large data
Other features not included: weather, distance to beach and mountain.
2020 not included
Difficulties: calculate nearby_avg_ctx1, select distances, etc.
Improvements: all station info. play with distances

# Data Exploration and Feature Selection

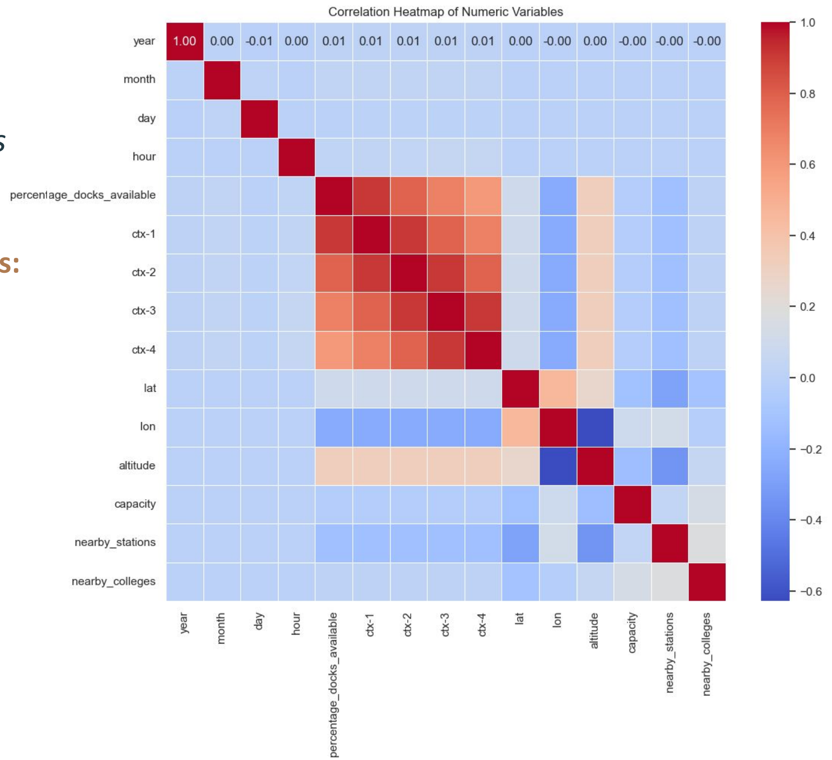
**Feature selection based on:**

- **Correlation Heatmap** → *Numerical variables*
- **Boxplots** → *Exploring categorical variables*
- **Feature importance** → *Applying some predictive models*




**Important features:**

*Context variables*
*Time of the day*
*Altitude*
*Post code*
*Workday*
*Nearby Stations*
*…*



# Split, Transformation and Modelization

### Train / Validation / Test

- **Train/Validation** → *80/20 from 2021, 2022 and 2023*
- **Test** → *Kaggle* → *First months of 2024*

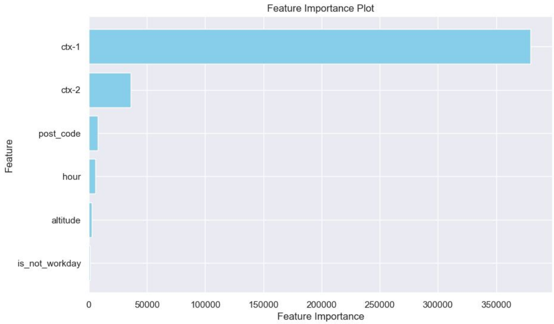
### Data Transformation

- **OneHotEncoder** → *Binary: 0 or 1*
- **MinMaxScaler** → *Range from 0 to 1*

### Predictive models

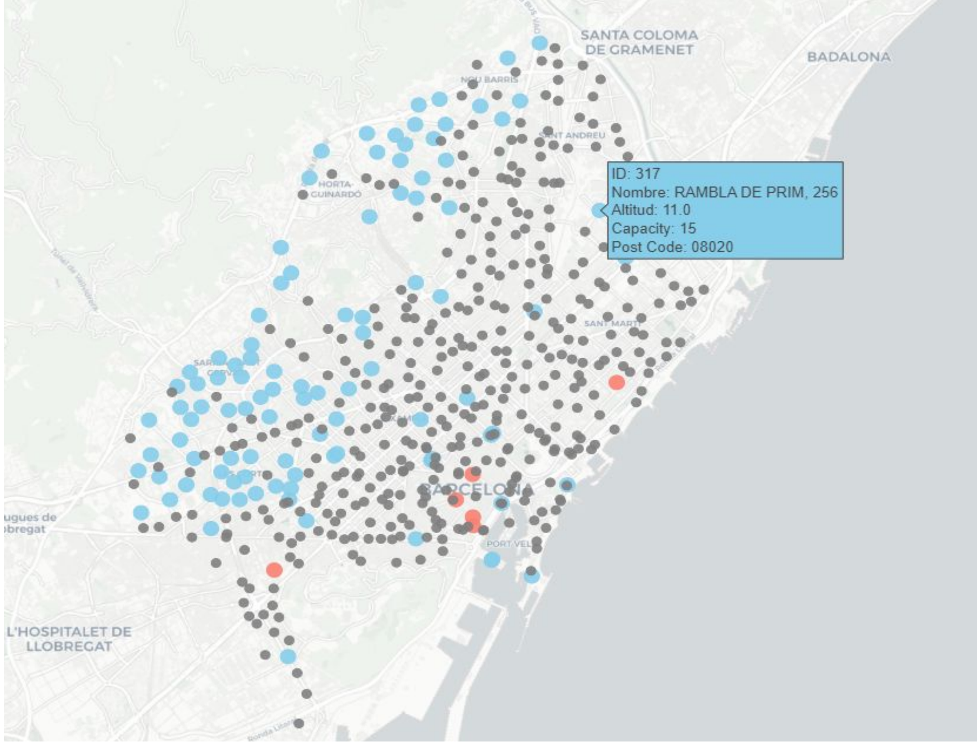
- **Linear Regression**→ Simple but less accurate
- **XGBoost** → Better approach than Linear Regression, specially with categorical features
- **Light GBM** → Best performance with **rmse: 0.0957699** and **R2 Score: 0.8691991**
- **Neural Network** → Accuracy very similar to Light GBM but adds complexity and computation time

```
percentage_docks_available      float64
ctx-1                           float64
ctx-2                           float64
post_code                      category
hour                           category
altitude                          int64
is_not_workday                    int64
dtype: object
```




| | Feature | Importance |
|---|---|---|
| 0 | ctx-1 | 379500.767302 |
| 1 | ctx-2 | 35996.876162 |
| 2 | post_code | 7553.171606 |
| 3 | hour | 5734.648390 |
| 4 | altitude | 2858.569710 |
| 5 | is_not_workday | 878.583539 |

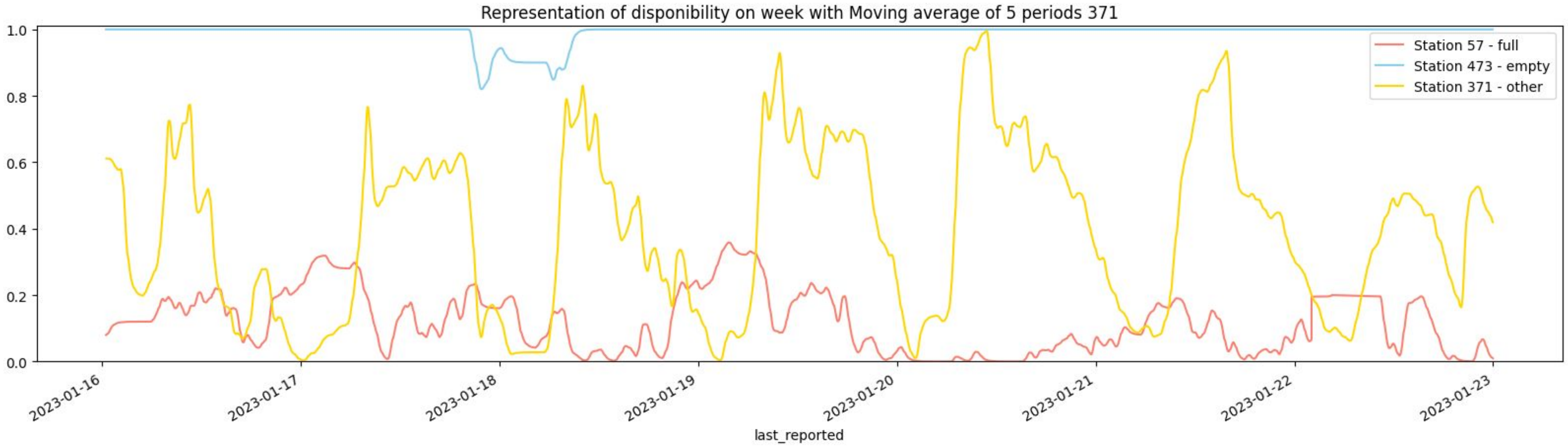
# Visualization of Stations in Barcelona Map



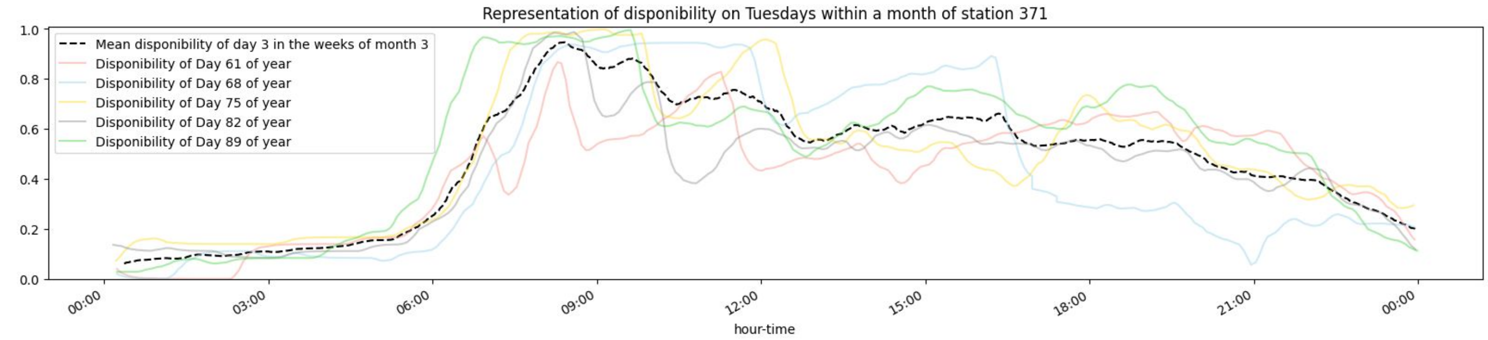

- **Objetivo Principal**: buscar rutas o movimientos entre estaciones más eficientes

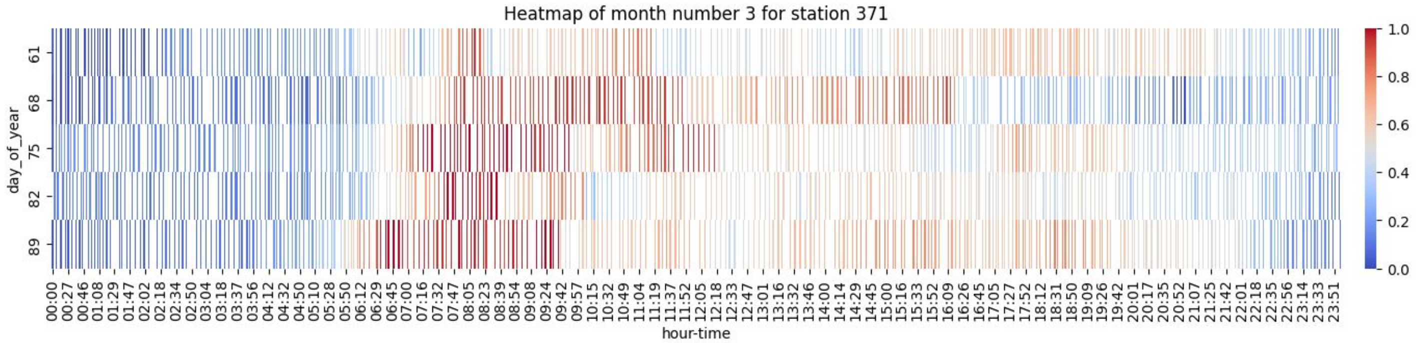
- **Objetivo**: clasificar las estaciones por su disponibilidad usual y localizarlas para observar los diferentes comportamientos.
- ¿Su capacidad máxima depende de la localización?
- ¿La localización tiene algo que ver con su estado usual?

# Visualization of Stations 371, 57 and 473. Week number 11



# Visualization of March of availability on 371




Franjas destacables:

- 00:00 - 06:00
- 06:00 - 12:00
- 12:00 - 17:00
- 17:00 - 24:00

# Wishlist and Future steps

- **¿Qué necesitaríamos para seguir trabajando?**
- Registrar de las entradas y salidas de bicis de las estaciones.
- Actualización de la Base de Datos y limpieza.

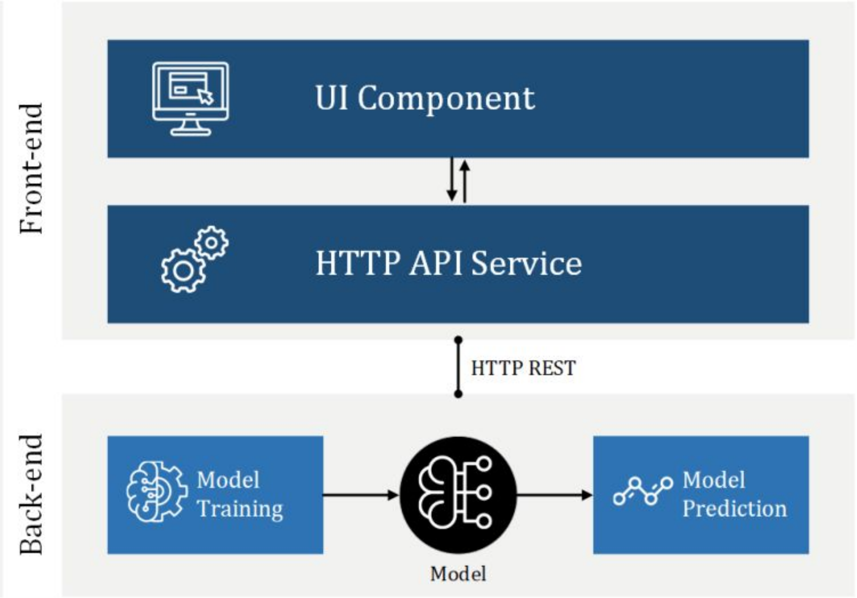
- **Próximos pasos en la investigación**
- Estudiar el comportamiento de estaciones próximas
- Estudio de patrones en días especiales: de frío, de calor, festivos…
- Estudio especial a los casos en que las estaciones están o siempre vacías o siempre llenas.

# Product proposal: Web App predictor

Meta:

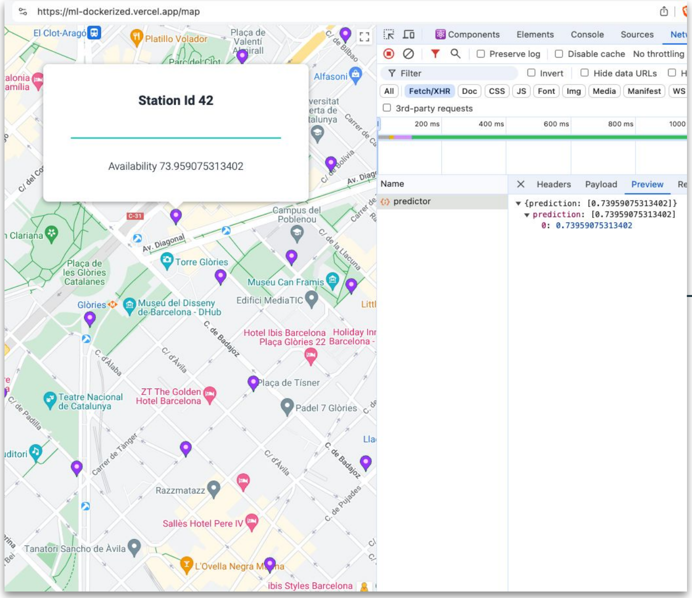
Ofrecer una manera amigable de elegir una bicicleta para alquiler, tomando decisiones más inteligentes utilizando nuestro modelo predictivo

Pero, cómo podemos productizar esto y hacerlo de acceso público?



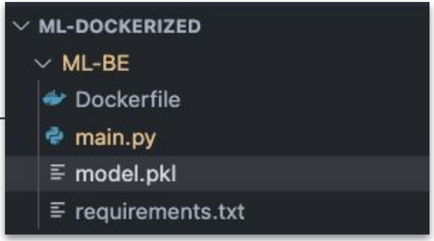
# Demo

## User Interface




## Backend Prediction API

```python
import os
import uvicorn
import joblib
from pydantic import BaseModel
from fastapi import FastAPI, HTTPException


app = FastAPI()
model = None


class PredictionRequest(BaseModel):
    input: list


class PredictionResponse(BaseModel):
    prediction: list


def load_model():
    global model
    if model is None:
        model = joblib.load('model.pkl')


@app.get("/")
async def root():
    return {"message": "Welcome to our ML prediction algorithm! "}


@app.post("/predict", response_model=PredictionResponse)
def predict(request: PredictionRequest):
    load_model()

    prediction = model.predict(request.input)
    return PredictionResponse(prediction=prediction.tolist())


@app.get("/healthcheck/")
def healthcheck():
    return 'Health - OK'


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=os.environ.get('PORT', 8000))
```




- Por razones de limitaciones en nuestro hosting service, vamos a utilizar un modelo sin entrenamiento dinámico

# Demo

# MUCHAS GRACIAS POR SU ATENCIÓN

## ¿PREGUNTAS?

UNIVERSITAT DE BARCELONA