# K-Nearest Neighbours

Ondřej Duba Filip Molhanec

2025-01-10

## What is KNN

Although K-Nearest Neigbours (KNN) is often regarded as very simple machine learning algorithm, its utility and power are undeniable. It is one of the core algorithms for supervised learning. Simply put, supervised learning is proccess of creating model that can predict value of target variable based on input data, using knowledge from dataset where we know the actual values of the target variables. KNN can be effectively used for both classification (target variable can take a limited number of values) and regression (target variable can take on continuous range of values) tasks. The simplest explanation of KNN for classification tasks is that an object is classifed by plurality vote of its $k$ nearest neighbours. For regression tasks, KNN can be generalized so that an object is assigned value that is the average of the values of its $k$ nearest neigbours. However, this is just a basic overview and there are several factors to consider when using KNN to develop an effective machine learning model.

“‘roaqn latex

## What is KNN?

Although K-Nearest Neigbours (KNN) is often regarded as very simple machine learning algorithm, its utility and power are undeniable. It is one of the core algorithms for supervised learning. Simply put, supervised learning is proccess of creating model that can predict value of target variable based on input data, using knowledge from dataset where we know the actual values of the target variables. KNN can be effectively used for both classification (target variable can take a limited number of values) and regression (target variable can take on continuous range of values) tasks. The simplest explanation of KNN for classification tasks is that an object is classifed by plurality vote of its k nearest neighbours. For regression tasks, KNN can be generalized so that an object is assigned value that is the average of the values of its k nearest neigbours. However, this is just a basic overview and there are several factors to consider when using KNN to develop an effective machine learning model.

\end{document}"'

## Dataset description

We have chosen *Air Quality and Pollution Assesment* dataset to show how K-Nearest Neighbours algorithm works. This Dataset is derived from World Health Organization and World Bank Group This Dataset contains several features, in other words, columns, lets go through each one of them and explain what they mean.

- **Temperature(°C)**: Average temperature of the region
- **Humidity (%)**: Relative humidity recorded in the region
- **PM2.5 Concentration (µg/m³)**: Fine particulate matter level
- **PM10 Concentration (µg/m³)**: Coarse particulate matter levels
- **NO2 Concentration (ppb)**: Nitrogen dioxide levels
- **SO2 Concentration (ppb)**: Sulfur dioxide levels
- **CO Concentration (ppm)**: Carbon monoxide levels
- **Proximity to Industrial Areas (km)**: Distance to the nearest industrial zone
- **Population Density (people/km²)**: Number of people per square kilometer in the region

Then there is so called Target Variable, that's the variable that we are trying to predict, in our dataset, it is called **Air Quality** and it can have 4 possible values depending on Air Quality, these values are the following:

- **Good**: Clean air with low pollution levels.
- **Moderate**: Acceptable air quality but with some pollutants present.
- **Poor**: Noticeable pollution that may cause health issues for sensitive groups.
- **Hazardous**: Highly polluted air posing serious health risks to the population.n.

## Importing all libraries that will be used

## Preprocessing data

As part of data preprocessing we read data from csv file and then we preprocess them. Considering that there are no missing values, there is no need to fill them. All features are already numerical, so there is no need to convert them any further. Target variable can take on 4 values, and there is order between those values (it is ordinal categorical datatype), so we convert it into categorical type with order between them. We split data into 3 parts, train, validate and test with train size being 60% of the original dataset and validate and test both being 20% of the original dataset

```python
def preprocess_data(df:pd.DataFrame)->pd.DataFrame:
    """ Function, for preprocessing data
    """
    qual_category = pd.api.types.CategoricalDtype(categories=['Hazardous', 'Poor', 'Moderate
    df['Air Quality'] = df['Air Quality'].astype(qual_category)
    return df
```

```python
def read_data(path:str='data/data.csv', y:str='Air Quality',**kwargs)->tuple:
    """ Function thats read data, and splits them into Train, Validation and Test datasets
    also separates, target value from others values.
    ---
    Attributes:
    path: [str], path to csv data file
    y: [str], name of Target value
    kwargs: options, use seed for random_seed
    ---
    Returns:
    tuple with Train,Validation,Test parametrs set, Target values: Train, Test, Validation
    """
    df = pd.read_csv(path)

    display(df.info())
    display(df.describe())

    df = preprocess_data(df)
    # Split the training dataset into train and rest (default 60% : 40%)
    Xtrain, Xrest, ytrain, yrest = train_test_split(
        df.drop(columns=[y]), df[y], test_size=0.4, random_state=kwargs.get('seed',42))
    # Split the rest of the data into validation dataset and test dataset (default: 24% : 16
    Xtest, Xval, ytest, yval = train_test_split(
        Xrest, yrest, test_size=0.5, random_state=kwargs.get('seed',42))
    print(f'Dataset: {path} | Target value: {y} | Seed: {kwargs.get('seed',42)}')
    return Xtrain, Xtest, Xval, ytrain, ytest, yval
```

```python
Xtrain, Xtest, Xval, ytrain, ytest, yval = read_data(seed=42)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 10 columns):
 #   Column                         Non-Null Count  Dtype
```

```
---  ------                         --------------  -----
 0   Temperature                    5000 non-null   float64
 1   Humidity                       5000 non-null   float64
 2   PM2.5                          5000 non-null   float64
 3   PM10                           5000 non-null   float64
 4   NO2                            5000 non-null   float64
 5   SO2                            5000 non-null   float64
 6   CO                             5000 non-null   float64
 7   Proximity_to_Industrial_Areas  5000 non-null   float64
 8   Population_Density             5000 non-null   int64
 9   Air Quality                    5000 non-null   object
dtypes: float64(8), int64(1), object(1)
memory usage: 390.8+ KB
```

```
None
```

| | Temperature | Humidity | PM2.5 | PM10 | NO2 | SO2 | CO | Proximity_to_Industrial_Areas | Populati |
|---|---|---|---|---|---|---|---|---|---|
| count | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | | |
| mean | 30.029020 | 70.056120 | 20.142140 | 30.218360 | 26.412100 | 10.014820 | 1.500354 | | |
| std | 6.720661 | 15.863577 | 24.554546 | 27.349199 | 8.895356 | 6.750303 | 0.546027 | | |
| min | 13.400000 | 36.000000 | 0.000000 | -0.200000 | 7.400000 | -6.200000 | 0.650000 | | |
| 25% | 25.100000 | 58.300000 | 4.600000 | 12.300000 | 20.100000 | 5.100000 | 1.030000 | | |
| 50% | 29.000000 | 69.800000 | 12.000000 | 21.700000 | 25.300000 | 8.000000 | 1.410000 | | |
| 75% | 34.000000 | 80.300000 | 26.100000 | 38.100000 | 31.900000 | 13.725000 | 1.840000 | | |
| max | 58.600000 | 128.100000 | 295.000000 | 315.800000 | 64.900000 | 44.900000 | 3.720000 | | |

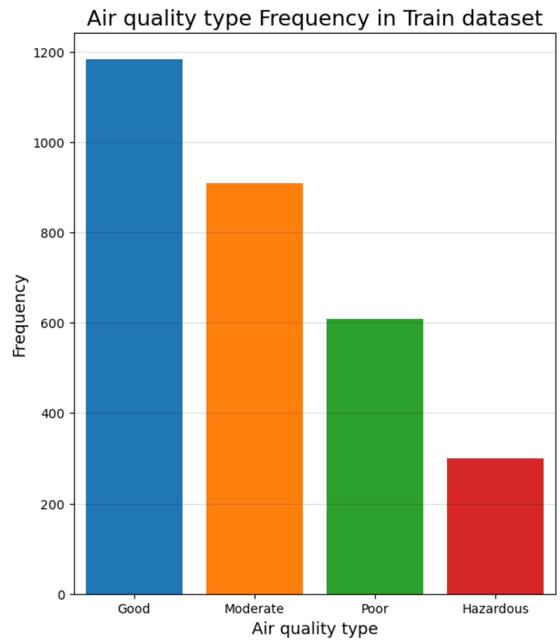
```
Dataset: data/data.csv | Target value: Air Quality | Seed: 42
```

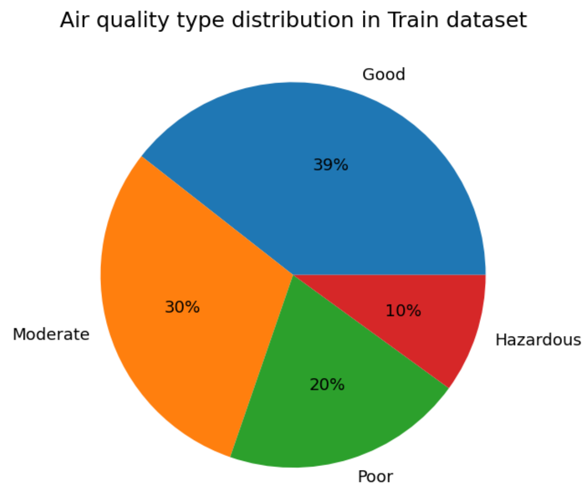
## Train data analyzation

- Before training the model on the train part of the dataset, we can try to look at the values to learn a little bit more about the dataset. The reason we only explore the train part of the dataset is that we don't look at values from validating and test datasets because we want to treat them as "new" data that the model has not seen so that they can be used for accuracy estimates.

```
df_original = Xtrain

df_tmp = ytrain.copy()
df_tmp = df_tmp.astype("category")
df_tmp_counts = df_tmp.value_counts()


custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

fig = plt.figure(figsize=(15,8))
ax1 = fig.add_subplot(1,2,1)
sns.barplot(x = df_tmp_counts.index, y = df_tmp_counts.values, ax = ax1, order=df_tmp_counts
for i, bar in enumerate(ax1.patches):
    bar.set_facecolor(custom_colors[i])

ax1.set_xlabel("Air quality type", fontsize = 13)
ax1.set_ylabel("Frequency",fontsize = 13)
ax1.set_title("Air quality type Frequency in Train dataset",fontsize = 17)
ax1.grid(axis='y', color='black', alpha=.2, linewidth=.5)

animal_count = df_tmp.value_counts()
ax2 = fig.add_subplot(1,2,2)
ax2.pie(animal_count, labels=animal_count.index,autopct='%.0f%%', textprops={"fontsize": 13}
ax2.set_title("Air quality type distribution in Train dataset", fontsize = 17)
```

```
Text(0.5, 1.0, 'Air quality type distribution in Train dataset')
```



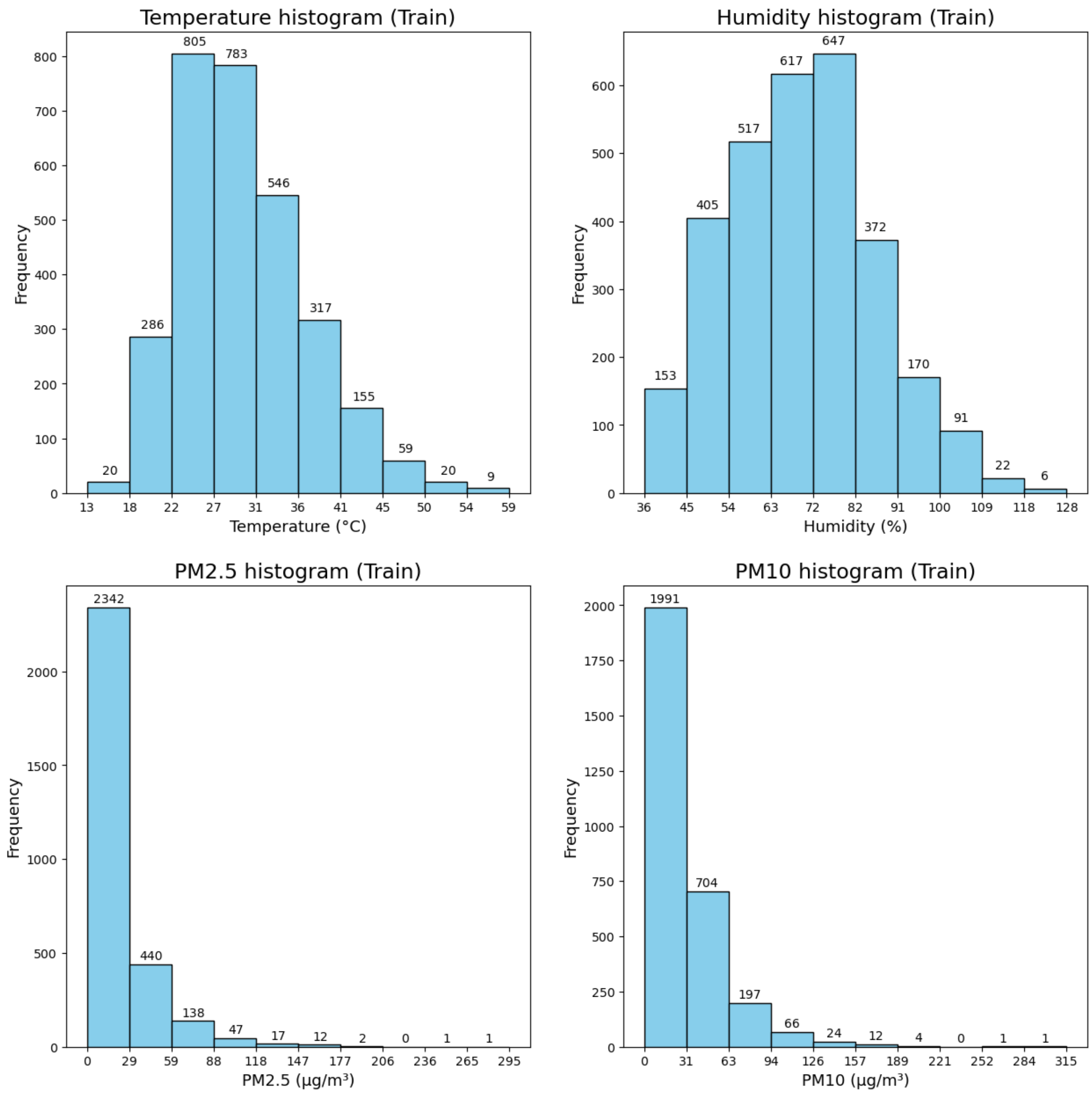

```
df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3000 entries, 4576 to 860
Data columns (total 9 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Temperature                    3000 non-null   float64
 1   Humidity                       3000 non-null   float64
 2   PM2.5                          3000 non-null   float64
 3   PM10                           3000 non-null   float64
 4   NO2                            3000 non-null   float64
 5   SO2                            3000 non-null   float64
 6   CO                             3000 non-null   float64
 7   Proximity_to_Industrial_Areas  3000 non-null   float64
 8   Population_Density             3000 non-null   int64
dtypes: float64(8), int64(1)
memory usage: 234.4 KB
```

```
fig, axes = plt.subplots(2, 2, figsize=(15, 15))

ax = axes[0,0]
```

```
# Plot the histogram
counts, bins, patches = ax.hist(df_original["Temperature"], edgecolor="black", color="skyblue

ax.set_xticks(bins)
ax.set_xticklabels([round(x) for x in bins])

# Set titles and labels
ax.set_title("Temperature histogram (Train)", fontsize = 17)
ax.set_xlabel("Temperature (°C)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+2.5, count + 10, int(count), ha="center", va="bottom")

ax = axes[0,1]

# Plot the histogram
counts, bins, patches = ax.hist(df_original["Humidity"], edgecolor="black", color="skyblue")

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])

# Set titles and labels
ax.set_title("Humidity histogram (Train)", fontsize = 17)
ax.set_xlabel("Humidity (%)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+4.5, count + 10, int(count), ha="center", va="bottom")


ax = axes[1,0]

# Plot the histogram
counts, bins, patches = ax.hist(df_original["PM2.5"], edgecolor="black", color="skyblue")

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])

# Set titles and labels
ax.set_title("PM2.5 histogram (Train)", fontsize = 17)
ax.set_xlabel("PM2.5 (µg/m³)", fontsize = 13)
```

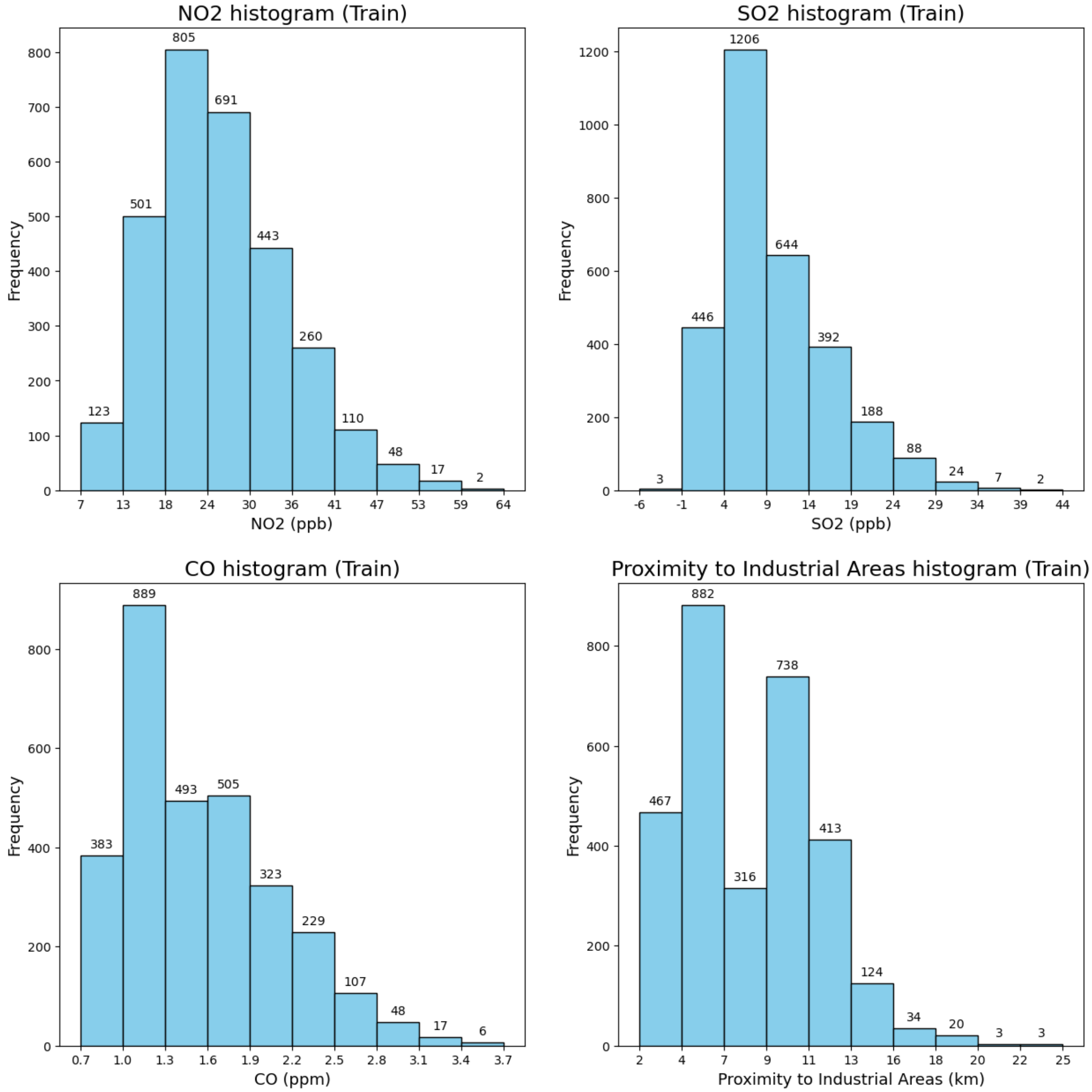
```python
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+15.5, count + 10, int(count), ha="center", va="bottom")

ax = axes[1,1]

# Plot the histogram
counts, bins, patches = ax.hist(df_original["PM10"], edgecolor="black", color="skyblue")

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])

# Set titles and labels
ax.set_title("PM10 histogram (Train)", fontsize = 17)
ax.set_xlabel("PM10 (µg/m³)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+15.5, count + 10, int(count), ha="center", va="bottom")
```




```
fig, axes = plt.subplots(2, 2, figsize=(15, 15))

ax = axes[0,0]

# Plot the histogram
counts, bins, patches = ax.hist(df_original["NO2"], edgecolor="black", color="skyblue")

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])
```

```python
# Set titles and labels
ax.set_title("NO2 histogram (Train)", fontsize = 17)
ax.set_xlabel("NO2 (ppb)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+2.5, count + 10, int(count), ha="center", va="bottom")

ax = axes[0,1]

# Plot the histogram
counts, bins, patches = ax.hist(df_original["SO2"], edgecolor="black", color="skyblue")

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])

# Set titles and labels
ax.set_title("SO2 histogram (Train)", fontsize = 17)
ax.set_xlabel("SO2 (ppb)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+2.5, count + 10, int(count), ha="center", va="bottom")


ax = axes[1,0]

# Plot the histogram
counts, bins, patches = ax.hist(df_original["CO"], edgecolor="black", color="skyblue")

ax.set_xticks(bins)
ax.set_xticklabels([f'{x:.1f}' for x in bins])

# Set titles and labels
ax.set_title("CO histogram (Train)", fontsize = 17)
ax.set_xlabel("CO (ppm)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i] + 0.15, count + 10, int(count), ha="center", va="bottom")

ax = axes[1,1]
```

```
# Plot the histogram
counts, bins, patches = ax.hist(df_original["Proximity_to_Industrial_Areas"], edgecolor="bla

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])



# Set titles and labels
ax.set_title("Proximity to Industrial Areas histogram (Train)", fontsize = 17)
ax.set_xlabel("Proximity to Industrial Areas (km)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i] + (bins[i+1] - bins[i])/2, count + 10, int(count),
            ha="center", va="bottom")
```




```
fig, axes = plt.subplots(2, 2, figsize=(13, 13))

ax = axes[0,0]
# Plot the histogram
counts, bins, patches = ax.hist(df_original["Population_Density"], edgecolor="black", color='

ax.set_xticks(bins)
ax.set_xticklabels([int(x) for x in bins])
```

```
# Set titles and labels
ax.set_title("Population Density histogram (Train)", fontsize = 17)
ax.set_xlabel("Population Density (people/km²)", fontsize = 13)
ax.set_ylabel("Frequency", fontsize = 13)

for i, count in enumerate(counts):
    ax.text(bins[i]+28.5, count + 5, int(count), ha="center", va="bottom")

axes[0,1].set_visible(False)
axes[1,1].set_visible(False)
axes[1,0].set_visible(False)
```