STK-IN9300- Mandatory assignment 1 of 2

Author: Anna Lina Sjur

Presentation of dataset

Short introduction

I have chosen to work with the dataset **Seoul bike sharing demand**, which I found in the UCI machine learning repository. As it sais on the repository's web page: "The dataset contains counts of public bicycles rented per hour in the Seoul Bike Sharing System, with corresponding wheater data and holiday information." It is licensed under a CC BY 4.0 license, which allows for using the dataset for any purpose, as long as credit is given.

Note: I changed the variable name containing degrees C from ° C to degC in the file, because I had trouble reading in the data with the original fomatting.

Possible use of dataset

To have a functioning bike sharing servise it is important to have enough bikes available, espesially at peak hours. It is also possible to cut costs by reducing the number of available bikes when demand is lower. We expect the time of year and time of day to influence the demand for bikes. It is also reasonable to expect weather conditions to influnece the number of bikes rented. Thus, this dataset can be used to construct a model for expected bike demand, which can be used to organize the bike sharing service. We can also use this dataset to investigate how different weather parameters influence peoples biking habits.

I will treate the number of rented bikes as the target, and a subgroup of the remaining variables as features.

Summary table

First, I read in the dataset as a Pandas DataFrame and have a look at what the data look like.

In [1]:
```
# import libraries
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# use seaborn to set the overall theme of plots
sns.set_theme("notebook")
```

In [2]:
```
# read in data
df = pd.read_csv("SeoulBikeData.csv", delimiter=",", parse_dates=["Date"], c
df.T
```

Out[2]:

	0	1	2	3	4	5	6	
Date	2017-12-01 00:00:00	2017-12-01 00:00:00	2017-12-01 00:00:00	2017-12-01 00:00:00	2017-12-01 00:00:00	2017-12-01 00:00:00	2017-12-01 00:00:00	00
Rented Bike Count	254	204	173	107	78	100	181	
Hour	0	1	2	3	4	5	6	
Temperature(degC)	-5.2	-5.5	-6.0	-6.2	-6.0	-6.4	-6.6	
Humidity(%)	37	38	39	40	36	37	35	
Wind speed (m/s)	2.2	0.8	1.0	0.9	2.3	1.5	1.3	
Visibility (10m)	2000	2000	2000	2000	2000	2000	2000	
Dew point temperature(degC)	-17.6	-17.6	-17.7	-17.6	-18.6	-18.7	-19.5	
Solar Radiation (MJ/m2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Rainfall(mm)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Snowfall (cm)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Seasons	Winter	Winter	Winter	Winter	Winter	Winter	Winter	W
Holiday	No Holiday	No Holiday	No Holiday	No Holiday	No Holiday	No Holiday	No Holiday	H
Functioning Day	Yes	Yes	Yes	Yes	Yes	Yes	Yes	

14 rows × 8760 columns

From the above DataFrame, we can see the different variables from the column names, and that there are a total of 8760 instances by the number of rows. We also get and idea about the data types and values. To get a better overview of the different variable values, I construct a new DataFrame with summary statistics.

In [3]:
```
# Initalize a sictionary for summary
summary = {
    "Variable name" : [],
    "Units" : [],
    "Type" : [],
    "Value range" : [],
    "Mean value" : [],
    "Median" : [],
    "Standard deviation" : []
}
```

```
# loop over variables
for keyword in df.columns:
    # check if column name contains the unit. Store unit if relevant
    if len(keyword.split("(")) > 1:
        unit = keyword.split("(")[1][:-1]
        name = keyword.split("(")[0]
        # remove whitespace at end
        if name[-1] == " ":
            name = name[:-1]
        summary["Variable name"].append(name)
        summary["Units"].append(unit)
        df.rename({keyword:name}, inplace=True, axis='columns')
        keyword = name
    else:
        summary["Variable name"].append(keyword)
        summary["Units"].append("-")

    # extract variable values
    values = df[keyword].values

    # check if variable is numeric or not
    if np.issubdtype(values.dtype, np.number):
        summary["Type"].append("Numerical")
        summary["Value range"].append(f"{np.min(values)} – {np.max(values)}"
        summary["Mean value"].append(np.mean(values))
        summary["Median"].append(np.median(values))
        summary["Standard deviation"].append(np.std(values))
    else:
        if keyword == "Date":
            summary["Type"].append("Numerical/Categorical")
            summary["Value range"].append(f"{df["Date"].min().strftime("%d.%
        else:
            summary["Type"].append("Categorical")
            summary["Value range"].append(', '.join(np.unique(df[keyword].va
        summary["Mean value"].append("-")
        summary["Median"].append("-")
        summary["Standard deviation"].append("-")
```

In [4]:

```
summary_df = pd.DataFrame.from_dict(summary)
summary_df.set_index("Variable name", inplace=True)

summary_df
```

Out[4]:

Variable name	Units	Type	Value range	Mean value	Median	Standard deviation
Date	-	Numerical/Categorical	01.12.2017 to 30.11.2018	-	-	-
Rented Bike Count	-	Numerical	0 – 3556	704.602055	504.5	644.960652
Hour	-	Numerical	0 – 23	11.5	11.5	6.922187
Temperature	degC	Numerical	-17.8 – 39.4	12.882922	13.7	11.944143
Humidity	%	Numerical	0 – 98	58.226256	57.0	20.361251
Wind speed	m/s	Numerical	0.0 – 7.4	1.724909	1.5	1.036241
Visibility	10m	Numerical	27 – 2000	1436.825799	1698.0	608.263991
Dew point temperature	degC	Numerical	-30.6 – 27.2	4.073813	5.1	13.059624
Solar Radiation	MJ/m2	Numerical	0.0 – 3.52	0.569111	0.01	0.868697
Rainfall	mm	Numerical	0.0 – 35.0	0.148687	0.0	1.128129
Snowfall	cm	Numerical	0.0 – 8.8	0.075068	0.0	0.436721
Seasons	-	Categorical	Autumn, Spring, Summer, Winter	-	-	-
Holiday	-	Categorical	Holiday, No Holiday	-	-	-
Functioning Day	-	Categorical	No, Yes	-	-	-

Note on table: I have classified the date as both a numerical and a categorical variable. The reasoning goes like this: The date can be viewd as a numerical value, for example number of days, sinse some epoch. But the date can also be viewd as a category, for example as a working day or a weekend day. So the type depends on how you treat the date.

Bad Data Visualization

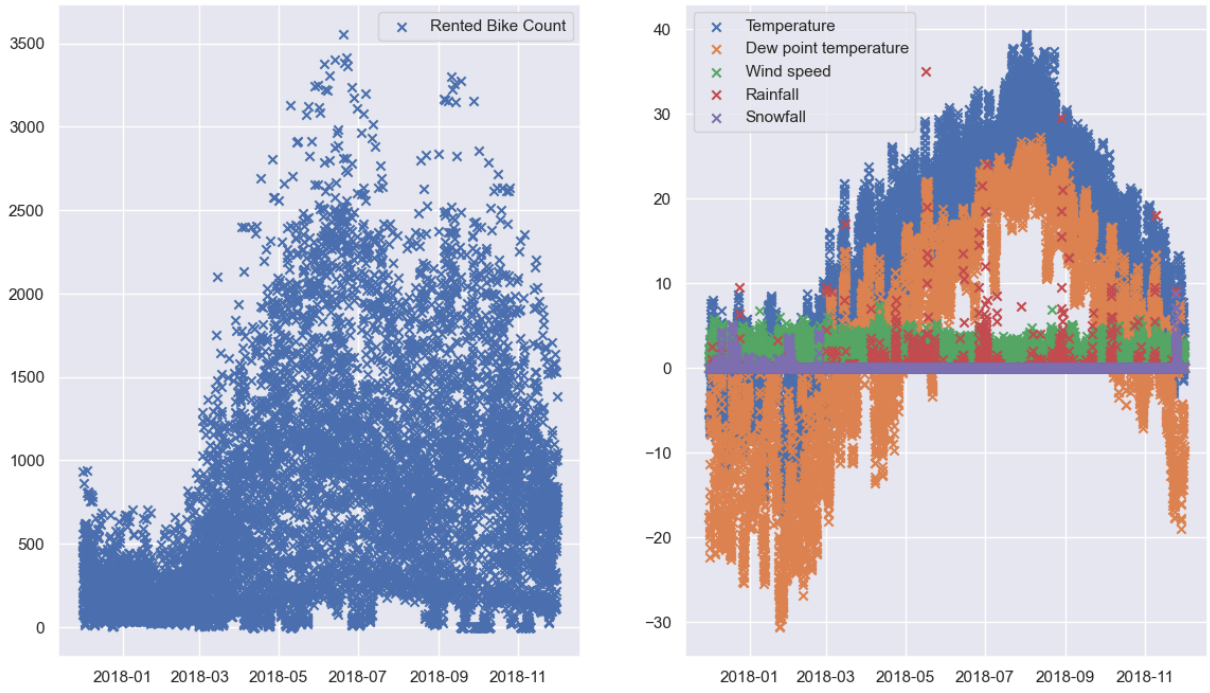
Since the variables in the dataset can be considered as timeseries, an intuitive way to visualize them is to plot them as a function of time. However, there are good and less good ways to do this. I will start with a less good one.

The goal is to plot the variable *Rented Bike Count*, together with a selsection of the meteorological variables. First, I tried to plot them all in one frame, but this turned out to

be an extremely bad visualization, since *Rented Bike Count* is several orders of magnitude larger than the other variables, and totally dominated the plot. I chose to make a slightly better plot, plotted and shown below:

In [5]:
```
fig, [ax1, ax2] = plt.subplots(figsize=(14,8), ncols=2)

# Combine date and hour into a single time array
t = df["Date"]+pd.to_timedelta(df["Hour"], unit='h')

# plot rented bike count in separate frame
ax1.scatter(t, df["Rented Bike Count"], label="Rented Bike Count", marker="x

# loop over variables to plot
for keyword in [ "Temperature", "Dew point temperature","Wind speed","Rainfa
    var = df[keyword]
    ax2.scatter(t, var, label=keyword, marker="x")

# create legends
ax1.legend()
ax2.legend()
```

Out[5]: `<matplotlib.legend.Legend at 0x7fc6f844bce0>`



Why is this a bad plot?

Although this plot does give us some idea of the variables time dependense, and also hint at some dependence between the variables, it is a bad plot for several reasons:

- **Data clutter**: The plots contains a lot of overlapping opaque data points, making it impossible to make out the distribution of the data, and even hiding one variable under another in the right subplot. Thus, we miss a lot of information.

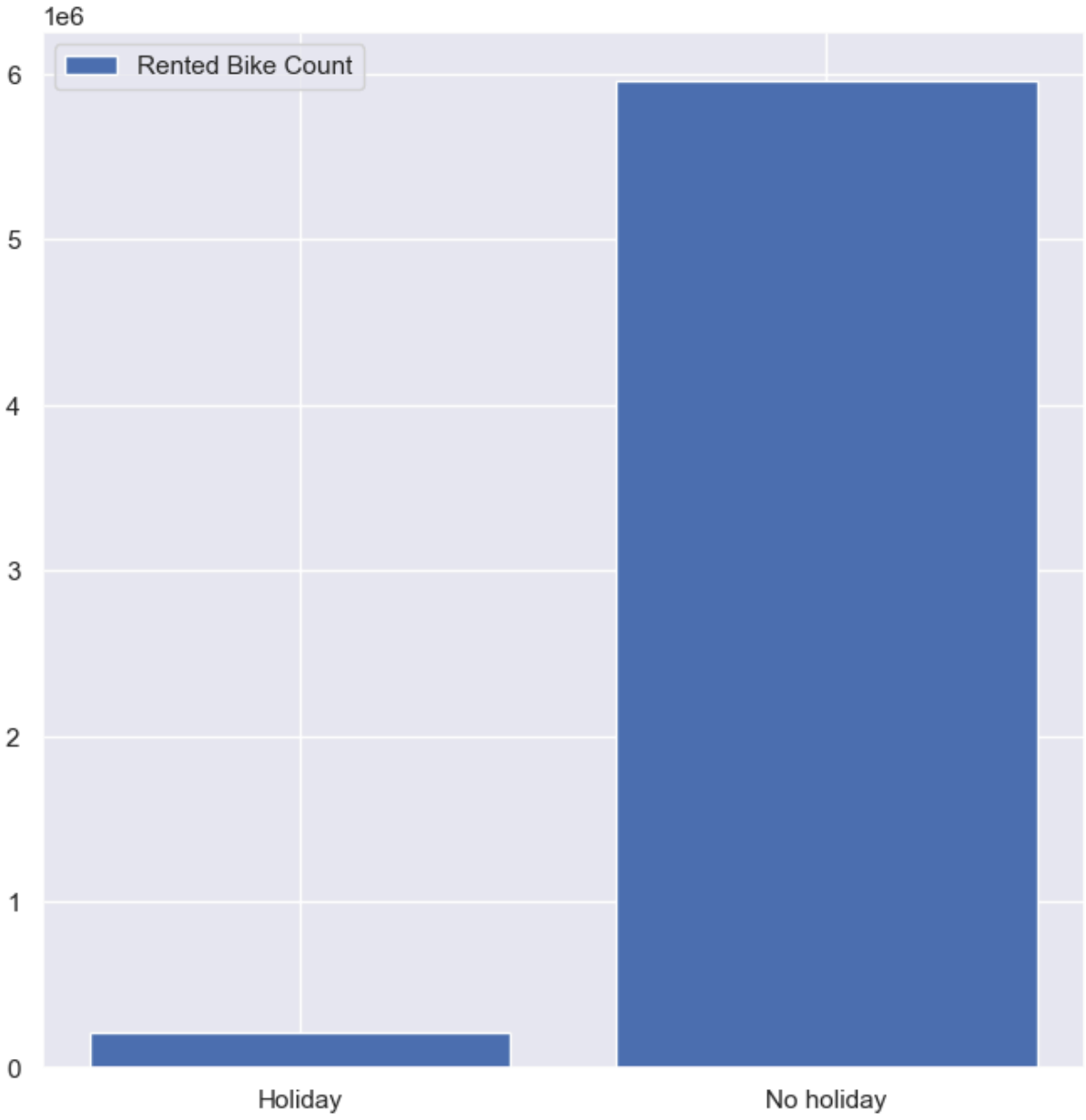
- **Multiple time scales**: This point relates to the point above. The plots contain variables varying both on a yearly time scale, and a daily time scale. This often has to be accounted for to make a sensible plot.
- **Multiple units on one axis**: The right subplot contains variables with different units and magnitudes plotted on the same axis. This leads to data being unnecesarrily compressed. This can also make it seem like some variables contain a lot more variability than others, even if this might not be the case.
- **Subplot positions**: Since the two subplots have the same x-axis, it would be better to plot one under the other.
- **Labels**: The axes are missing labels.
- **Units**: The plots does not contain any information about units. This could be very misleading, as the unit is important for intepreting data values.
- **Colorblind unfriendly**: The color palette is not suitable for colorblind people. For example, one should avoid using green and red together. Also, the colors are not well suited for black and white prints.

Next, we want to visualize a chategorical variable. It would be interesting to see how the demand for bikes are related to whether it is a holiday or not.

My bad solution is to simply plot the total number of rented bikes on holidays and no holidays as a bar plot:

In [6]:
```
fig, ax = plt.subplots(figsize=(8,8))

# split the DataFrame into a dataframe containing data for holidays and no h
holiday = df[df["Holiday"] == "Holiday"]
noholiday = df[df["Holiday"] == "No Holiday"]

# sum number of rented bikes for each DataFrame
nholiday = holiday["Rented Bike Count"].sum()
nnoholiday = noholiday["Rented Bike Count"].sum()

# make a bar plot
ax.bar(["Holiday", "No holiday"], [nholiday, nnoholiday], label="Rented Bike

ax.legend()
```

Out[6]:
```
<matplotlib.legend.Legend at 0x7fc6f48fd5e0>
```



Why is this a bad plot?

Although this plot clearly conveys that in total, fewer bikes are rented on holidays, it is bad for several reasons:

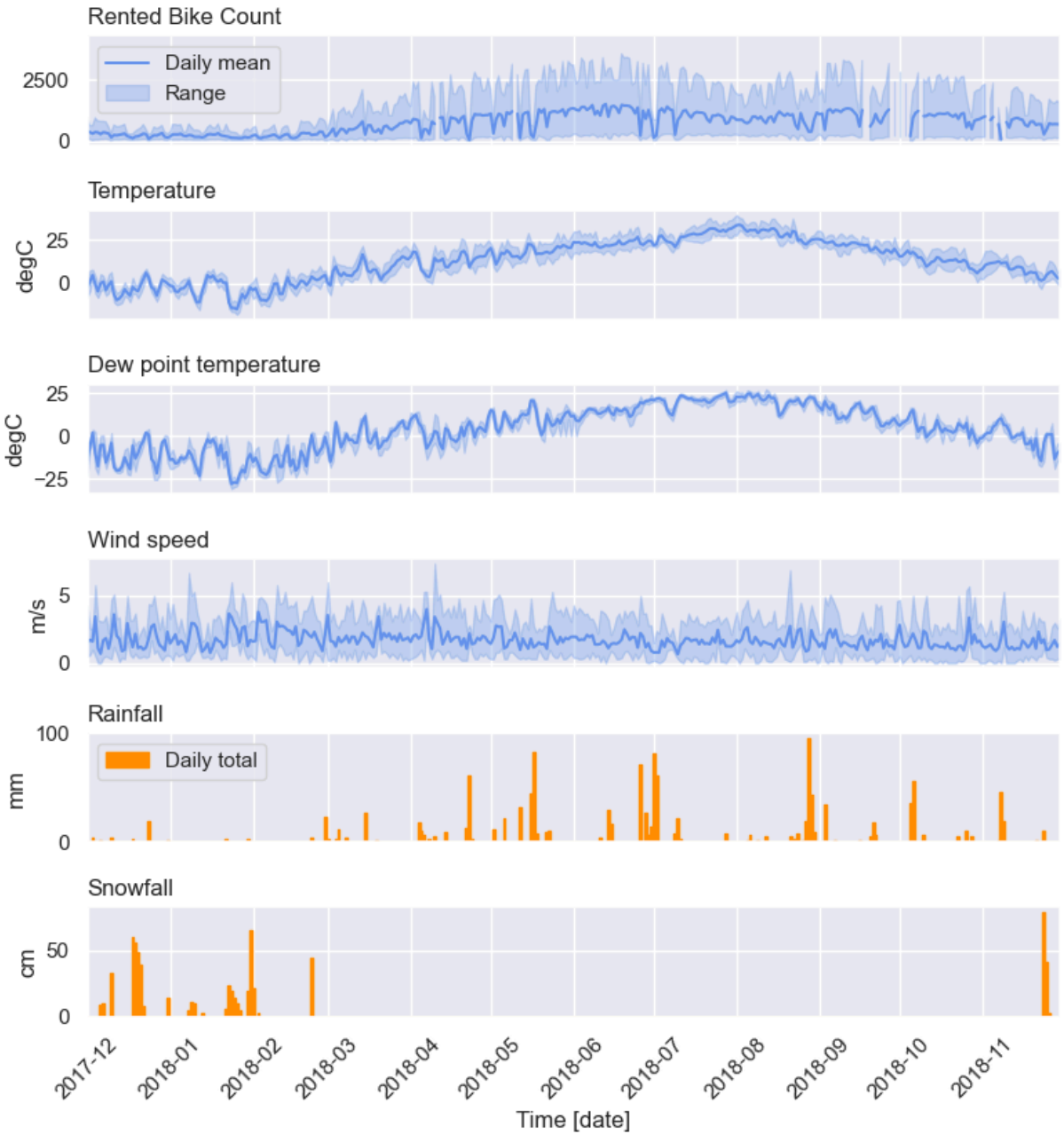
- **Overkill**: The same information could as easily, and more efficiently, have been given in the text or in a table.
- **Little information**: This plot contains very little information (only two numbers). It does not tell us the story behind these two numbers. Are there so few rented bikes on holidays because people don't rent bikes then, or is it because there are very few holidays?

Improved Data Visualization

Let's see if we can improve the first plot:

In [7]:
```
# list of variables to plot
keywords = ["Rented Bike Count", "Temperature", "Dew point temperature","Wir
#keywords = ["Rented Bike Count", "Temperature", "Dew point temperature", "H
mosaic = [[keyword] for keyword in keywords]

# spesify colors. I have a weak spot for the combination blue and orange. It
color_line = "cornflowerblue"
color_bar = "darkorange"

# create figure and axis mosaic. One subplot for each variable
fig, axd = plt.subplot_mosaic(
    mosaic,
    sharex = True,
    figsize=(8,8.5)
                                )

# set the date as the index for easily calculation daily values
df_dated = df.set_index("Date")

# exclude non-functioning days from the rented bike count variable
df_dated.loc[df_dated["Functioning Day"] == "No", "Rented Bike Count"] = np.

# loop over variables to plot
for keyword in keywords:

    # extract variable information
    var = df_dated[keyword]
    unit = summary_df.transpose()[keyword]["Units"]

    # extract corresponding axis object
    ax = axd[keyword]

    # check if variable should be plotted as bar plot or not
    if keyword in ["Rainfall", "Snowfall"]:
        # calculate total for each day and plot
        total = var.groupby(pd.Grouper(freq='1D')).sum()

        ax.bar(total.index, total, width=1, color=color_bar, edgecolor=color
    else:
        # calculate statistics
        means = var.groupby(pd.Grouper(freq='1D')).mean()
        mins = var.groupby(pd.Grouper(freq='1D')).min()
        maxs = var.groupby(pd.Grouper(freq='1D')).max()

        ax.plot(means.index, means, color=color_line)
        ax.fill_between(means.index, mins, maxs, alpha=0.3, color=color_line

    # include unit on y-axis
    if unit == "-":
        ax.set_ylabel("")
    else:
        ax.set_ylabel(unit)
```

```
    # Set variable name as subplot title
    ax.set_title(keyword, loc="Left")

# Add legends
axd["Rented Bike Count"].legend(["Daily mean", "Range"])
axd["Rainfall"].legend(["Daily total"], loc="upper left")

# Adjust y-limit to make room for the legend
axd["Rented Bike Count"].set_ylim(None, 4300)

# Adjust x-axis limit and apperance
ax.set_xlim(df["Date"].values[0], df["Date"].values[-1])
ax.tick_params(axis='x', labelrotation=45)
ax.set_xlabel("Time [date]")

fig.tight_layout()
```

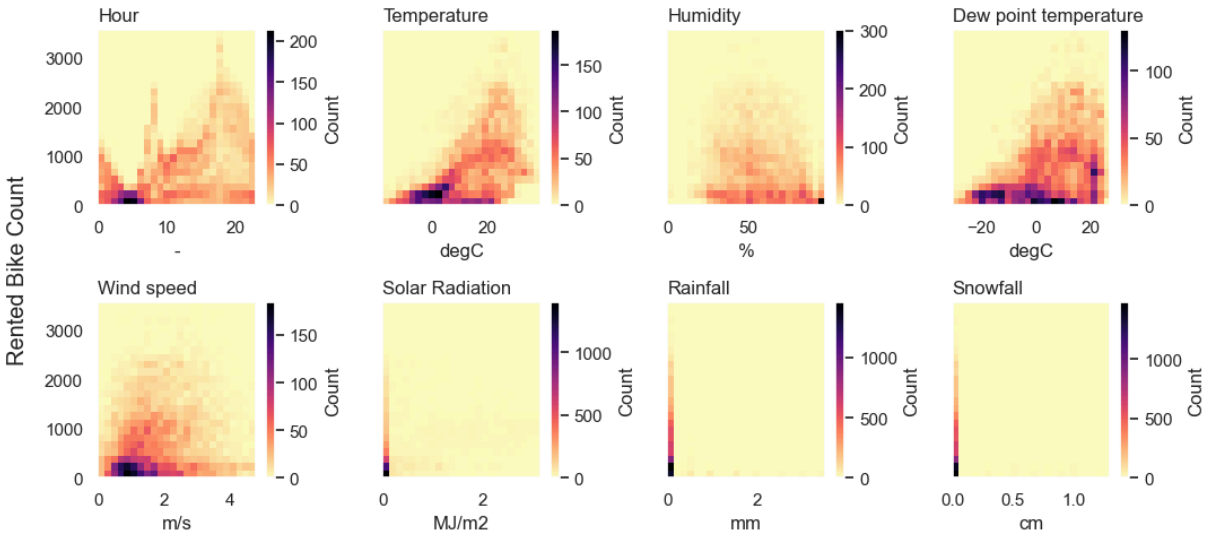


Why is this plot an improvement?

By splitting up the plot in several subplots, it's easier to see how the individual time series vary. Also, aligning the x-axes vertically makes it easier to compare variables. Futher, plotting the daily mean and spread instead of individual data points makes the plot less cluttered. Finaly, plotting daily rainfall and snowfall as bar plots align better with how these variables are usually plotted, and fit their physical interpertation better.

One might argue that creatign separate plots for each variable makes it harder to see whether variables correlate. This is a valid point. However, I prefere to use 2D histograms if I want to investigate possible correlations. So let's do that:

In [8]:
```
from scipy import stats

keywords = ["Hour", "Temperature", "Humidity", "Dew point temperature", "Wir
target = "Rented Bike Count"

fig, axd = plt.subplot_mosaic(
    [keywords[:4], keywords[4:]],#+["blank"]],
    empty_sentinel="blank",
    sharey = True,
    figsize=(11,5)
                              )

for keyword in keywords:
    # removing outliers outside 3 standard deviations
    df_nooutlier = df[np.abs(stats.zscore(df[keyword])) < 3]
    ax = axd[keyword]
    x = df_nooutlier[keyword]
    y = df_nooutlier[target]


    cb = ax.hist2d(x, y, bins=24, cmap="magma_r",
                   )
    fig.colorbar(cb[-1], ax=ax, label="Count")
    ax.set_title(keyword, loc="Left")
    ax.set_xlabel(summary_df["Units"][keyword])

fig.supylabel(target)
fig.tight_layout()
```



From these plots, we see signs of positive correlation between Rented Bike Count and some of the variables, like temperature, dew point temperature and wind speed. Also, there are indications of a negative correlation between humidity and Rented Bike COunt. Further, we see indications of increased bike use during rush hours from the first subplot. Futher, we see that there is mostly no snow or rainfall, so data tends to cluster around x equals 0 for the two last subplots.

Let's move on and try to improve out plot looking into the differnce between holiday and no holiday:

In [9]:
```
# Initiate figure
fig, axd = plt.subplot_mosaic(
    [["Holiday"], ["No Holiday"]],
    sharex = True,
    sharey=True,
    figsize=(8,8)
                                  )
# prepare a list fir colors ax.set_aspect("")
colors = sns.color_palette("crest_r", 6)

# Group the DataFrames for holiday and no holiday into the time of day
holiday_grouped = holiday.groupby(["Hour"])["Rented Bike Count"]
noholiday_grouped = noholiday.groupby(["Hour"])["Rented Bike Count"]

# calculate the median for each hour
holiday_median = holiday_grouped.median()
noholiday_median = noholiday_grouped.median()

# plot median
axd["Holiday"].plot(holiday_median.index, holiday_median, label="Median", zo
axd["No Holiday"].plot(noholiday_median.index, noholiday_median, zorder=10,


# plot quantiles
zorder = 9
for quantile, color in zip([0.2, 0.4, 0.6, 0.8, 0.99], colors[1:]):
```

```
    holiday_min = holiday_grouped.quantile((1-quantile)/2)
    holiday_max = holiday_grouped.quantile(quantile+(1-quantile)/2)

    noholiday_min = noholiday_grouped.quantile((1-quantile)/2)
    noholiday_max = noholiday_grouped.quantile(quantile+(1-quantile)/2)

    axd["Holiday"].fill_between(holiday_median.index, holiday_min, holiday_m
    axd["No Holiday"].fill_between(noholiday_median.index, noholiday_min, no
    zorder -= 1

# Set tiltle to holiday/no holiday, and the number of datapoints
axd["Holiday"].set_title(f"Rented bike count on holidays, {len(holiday)//24}
axd["No Holiday"].set_title(f"Rented bike count on no holiday, {len(noholida

# Spesify labels and legends
axd["Holiday"].legend()
axd["No Holiday"].set_xlabel("Hour of day")
axd["No Holiday"].set_xlim(0, 23)
axd["No Holiday"].set_xticks(np.arange(0,24));
```

Why is this plot an improvement?

By plotting the distibution of rented bike count as a function of hour of the day, we get a lot more information on how holidays and time of day influence the demand for bikes. By plotting the data this way, we see that there is generally a lower demand for bikes on holidays, even though the maximum value is comparable to no holidays. Futher, we see that there is a clear peak in demand during rush hours on no holidays - a feature that dissapears on holiday (I think it's quite cool that we see this so clearly). Finaly, the plot is a lot more pleasing to look at.

The next step could be to make similar plots to the plot above, but divide the data into season insted of holiday/no holiday. But I'll leave that to the future.

Simple analysis

It's time to do a simple analysis on the data. I will use methods form sklearn for this.

In [10]:
```
from sklearn.linear_model import LassoCV, LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import KFold, train_test_split
from sklearn.feature_selection import RFECV
```

First, I spesify parameters for the analysis and spesify the model, target and features.

In [11]:
```
# Fraction used for training and validation.
training_fraction = 0.75
validation_fraction = 1 - training_fraction

# Number of splits for the k-fold cross validation
n_splits = 5

# I will train a linear regression model
model = LinearRegression()

# Variable name of target and features for the analysis
target = "Rented Bike Count"
features = ["Day of Year", "Hour", "Temperature", "Humidity", "Dew point tem
            "Wind speed", "Solar Radiation", "Rainfall", "Snowfall", "Visibi

# Assign numerical value to categorical feature "Holiday"
holiday_map = {"No Holiday":0, "Holiday":1}
```

Then, I'll prepare the dataset. I have choosen to remove days where the bike rental is closed, and map the variable "Holidat" to have a binary value. Futher, I translate the date to day of year.

In [12]:
```
# Prepare dataset
df_prep = df.copy()

# Remove non-unctioning days
df_prep.loc[df_prep["Functioning Day"] == "No", "Rented Bike Count"] = np.na
df_prep.dropna(inplace=True)

# map holiday value to numeric value
df_prep["Numeric Holiday"] = df_prep["Holiday"].map(holiday_map)

# map date to day of year
df_prep["Day of Year"] = df_prep["Date"].dt.day_of_year
```

I split the dataset into a training dataset and validation dataset.

In [13]:
```
df_training, df_validation = train_test_split(df_prep, test_size=validation_

X_training = df_training[features]
y_training = df_training[target]

X_validation = df_validation[features]
y_validation = df_validation[target]
```

I use recursive feature elimination with cross-validation to select features. The cross-validation is done with the k-folds method.

In [14]:
```
# initialise the cross-validation
kfold = KFold(n_splits=n_splits, shuffle=True, random_state=42)

# preform feature selection
rfecv = RFECV(estimator=model, step=1, cv=kfold, scoring='r2')
rfecv.fit(X_training, y_training)
selected_features = X_training.columns[rfecv.support_]

print(f"Selected features: {", ".join(selected_features)}")
```

```
Selected features: Day of Year, Hour, Temperature, Humidity, Dew point tempe
rature, Wind speed, Solar Radiation, Rainfall, Snowfall, Numeric Holiday
```

Finally, I train the model on the training set using the selected features only. I use both rmse and r2 as metrics for preformance. The model preformance must be tested on the validation set. For curiosity, I have also included how the model preforms on the training set.

In [15]:
```
# prepare datasets containing only selected features
X_training_selected = X_training[selected_features]
X_validation_selected = X_validation[selected_features]

# fit model
model.fit(X_training_selected, y_training)

# predictions for validation set and training set
y_predicted_validation = model.predict(X_validation_selected)
y_predicted_training = model.predict(X_training_selected)

# calculate preformance metrics
rmse_training = np.sqrt(mean_squared_error(y_training, y_predicted_training)
r2_training = r2_score(y_training, y_predicted_training)

rmse = np.sqrt(mean_squared_error(y_validation, y_predicted_validation))
r2 = r2_score(y_validation, y_predicted_validation)

print(f"Training Data   - Root Mean Squared Error: {rmse_training:.0f}, R-sc
print(f"Validation Data - Root Mean Squared Error: {rmse:.0f}, R-squared: {r
```

```
Training Data   - Root Mean Squared Error: 450, R-squared: 0.517
Validation Data - Root Mean Squared Error: 432, R-squared: 0.528
```

Interestingly, the model preforms slightly better on the validation set than on the training set.

It can also be interesting to see what the final linear coeficients are for the selected features. Let's make a table:

In [16]:
```
coefs = model.coef_
print("Feature            Coefficient")
print("-------------------------------")
for feature, coef in zip(selected_features, coefs):
    print(f"{feature:<22s}{coef:>8.3f}")
```

```
Feature            Coefficient
-------------------------------
Day of Year              0.447
Hour                    28.279
Temperature             21.758
Humidity               -10.934
Dew point temperature   10.799
Wind speed              11.889
Solar Radiation        -78.693
Rainfall               -62.106
Snowfall                 2.349
Numeric Holiday       -154.946
```

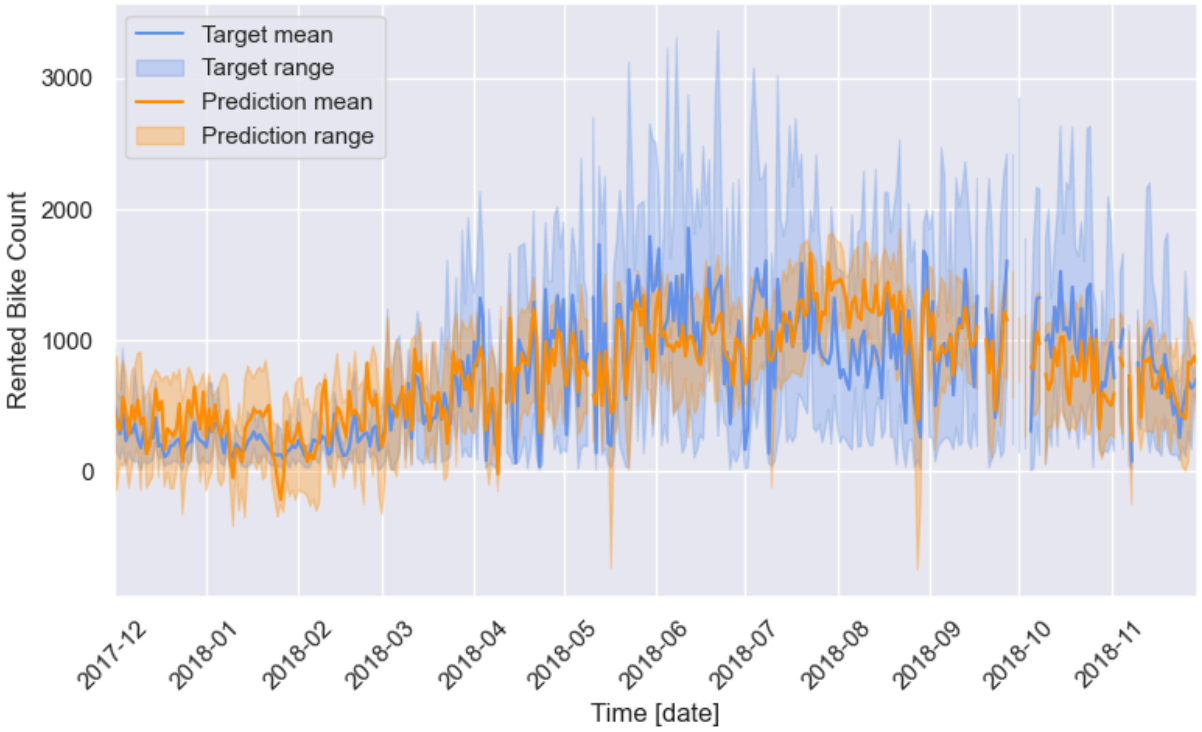
An interesting observations: The number of rented bikes seems to increase as the snowfall increases. Maybe public transportation breaks down under snowy conditions, so people have to bike to get home?

Let's also make a plot of the model prediction on the validation set:

In [17]:
```
df_validation[f"{target} Prediction"] = y_predicted_validation
df_training[f"{target} Prediction"] = y_predicted_training

df_validation.sort_index(inplace=True)
df_training.sort_index(inplace=True)
```

In [18]:
```
keywords = [target, f"{target} Prediction"]
colors = ["cornflowerblue", "darkorange"]

fig, ax = plt.subplots(figsize=(8,5))

df_dated = df_validation.set_index("Date")
df_dated.loc[df_dated["Functioning Day"] == "No", "Rented Bike Count"] = np.

for keyword, color in zip(keywords, colors):
    var = df_dated[keyword]

    means = var.groupby(pd.Grouper(freq='1D')).mean()
    mins = var.groupby(pd.Grouper(freq='1D')).min()
    maxs = var.groupby(pd.Grouper(freq='1D')).max()

    ax.plot(means.index, means, color=color)
```

```
    ax.fill_between(means.index, mins, maxs, alpha=0.3, color=color)


ax.set_xlim(df["Date"].values[0], df["Date"].values[-1])
ax.tick_params(axis='x', labelrotation=45)
ax.legend(["Target mean", "Target range", "Prediction mean", "Prediction rar
ax.set_ylabel("Rented Bike Count")
ax.set_xlabel("Time [date]")

fig.tight_layout()
```



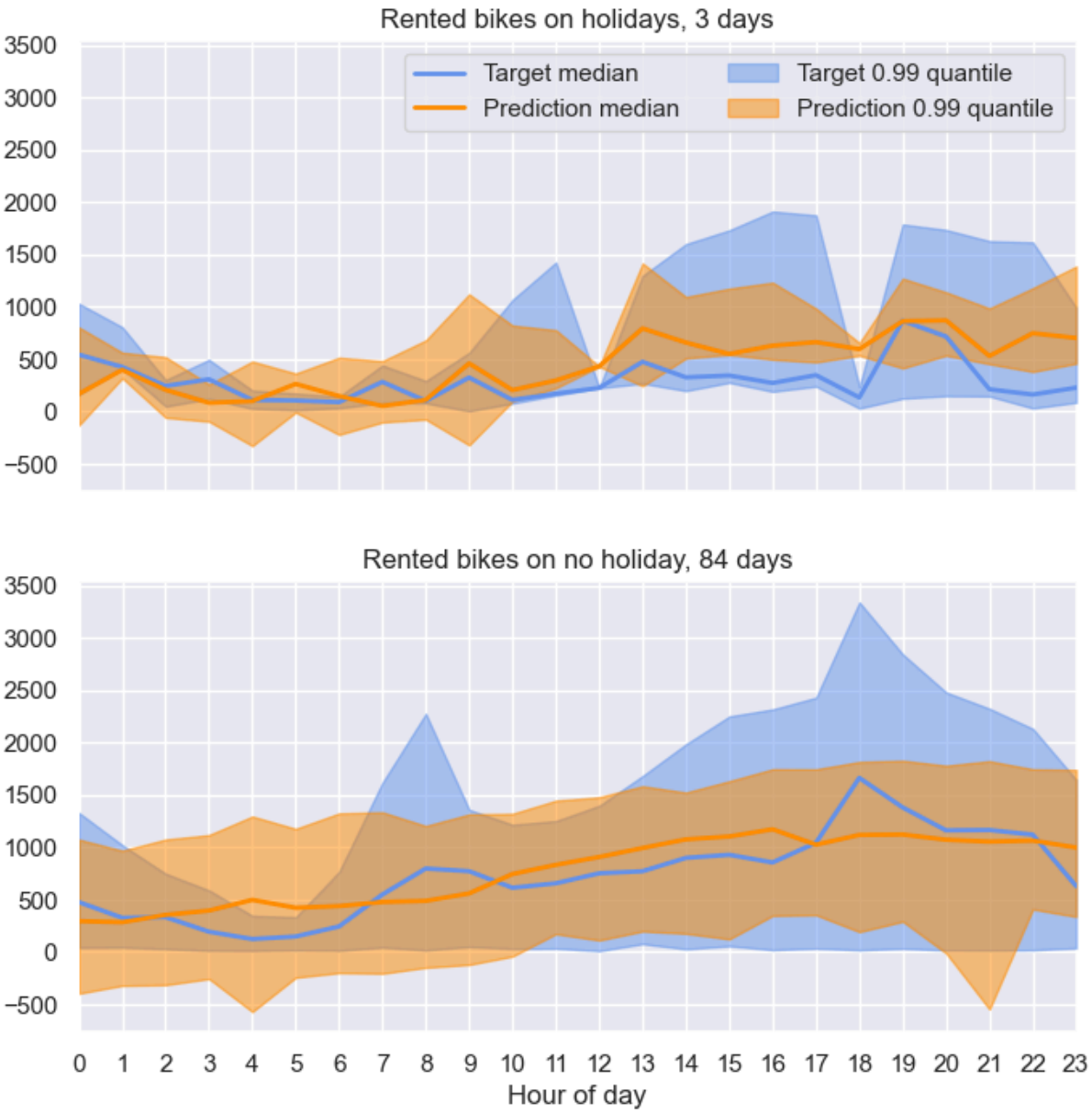
From this time series plot, we see that the model is able to capture some of the changes that we see in the target over time. Spesifically, we see variations on the same time scale, and an increas in the number of rented bikes in the summer months. However, the model also predicts a negative count at times, which is not very physical. Also, the range of the prediction is quite constant over time, while we see clear variation in the range of the target over the year.

Finally, let's see if our model is able to capture the peak in rush hours during no holidays:

In [19]:
```
fig, axd = plt.subplot_mosaic(
    [["Holiday"], ["No Holiday"]],
    sharex = True,
    sharey=True,
    figsize=(8,8)
                              )

holiday = df_validation[df_validation["Holiday"] == "Holiday"]
noholiday = df_validation[df_validation["Holiday"] == "No Holiday"]

holiday_grouped = holiday.groupby(["Hour"])[[target, f"{target} Prediction"]
```

```
noholiday_grouped = noholiday.groupby(["Hour"])[[target, f"{target} Predicti

holiday_median = holiday_grouped.median()
noholiday_median = noholiday_grouped.median()

color_target = "cornflowerblue"
color_prediction = "darkorange"
axd["Holiday"].plot(holiday_median.index, holiday_median[target], label="Mec
axd["Holiday"].plot(holiday_median.index, holiday_median[f"{target} Predicti

axd["No Holiday"].plot(noholiday_median.index, noholiday_median[target], zor
axd["No Holiday"].plot(noholiday_median.index, noholiday_median[f"{target} P

quantile = 0.99
zorder = 9

holiday_min = holiday_grouped.quantile((1-quantile)/2)
holiday_max = holiday_grouped.quantile(quantile+(1-quantile)/2)

noholiday_min = noholiday_grouped.quantile((1-quantile)/2)
noholiday_max = noholiday_grouped.quantile(quantile+(1-quantile)/2)

axd["Holiday"].fill_between(holiday_median.index, holiday_min[target], holic
                            label=f"{quantile} quantile",
                            zorder=zorder,
                            color=color_target,
                            alpha=0.5
                            )
axd["Holiday"].fill_between(holiday_median.index, holiday_min[f"{target} Pre
                            label=f"{quantile} quantile",
                            zorder=zorder,
                            color=color_prediction,
                            alpha=0.5
                            )


axd["No Holiday"].fill_between(noholiday_median.index, noholiday_min[target]
                            zorder=zorder,
                            color=color_target,
                            alpha=0.5
                            )
axd["No Holiday"].fill_between(noholiday_median.index, noholiday_min[f"{targ
                            zorder=zorder,
                            color=color_prediction,
                            alpha=0.5
                            )


axd["Holiday"].set_title(f"Rented bikes on holidays, {len(holiday)//24} days
#axd["Holiday"].text(0.1, 0.8, f"{len(holiday)//24} days" ,transform=axd["Ho


axd["No Holiday"].set_title(f"Rented bikes on no holiday, {len(noholiday)//2
#axd["No Holiday"].text(0.1, 0.8, f"{len(noholiday)//24} days" ,transform=ax

axd["Holiday"].legend(["Target median", "Prediction median", f"Target {quant
```

```
axd["No Holiday"].set_xlabel("Hour of day")
axd["No Holiday"].set_xlim(0, 23)
axd["No Holiday"].set_xticks(np.arange(0,24));
```



The median of our predition is not too far off the target median in both the holiday and no holiday case. But we do not see the rush hour signal, and the value range does not depend on the hour, as it does in the target.

Final Thoughts

Our linear model captures some key patterns in the dataset, such as increased demand during summer and evenings and decreased demand during holidays. However, it surely has limitations. The model fails to capture the full range of values and even predicts negative bike counts in some cases. The root mean squared error of 432 indicates the

typical magnitude of error in the predictions. Additionally, the R-squared value of 0.528 shows that our model explains about half of the variance in the target variable.

From our time-series plots and 2D histograms, it appears that the relationship between the rented bike count and the selected features is more complex than a simple linear relationship, although some correlation is evident. Whether our linear model is good enough depends on its intended use. If we only need a rough estimate of the rented bike count, the model may be sufficient. However, for more accurate predictions, a more advanced model would be necessary.

Here are some suggestions for improving the linear model in the future:

- **Remove the daily cycle from the target variable** to better isolate the effect of meteorological features. This could be done by calculating the mean bike count for each hour across all days.
- **Remove the yearly cycle** to account for seasonal patterns. This is more challenging with only one year of data, but one approach could be to smooth the data over time.
- **Map the "Hour" feature to a categorical feature like "Rush Hour/Non-Rush Hour"** to better capture the effect of rush hours on bike demand.

BONUS!

In [20]:
```
# Initiate figure
fig, axd = plt.subplot_mosaic(
    [[ "Summer", "Spring"], ["Autumn","Winter"]],
    sharex = True,
    sharey=True,
    figsize=(8,8)
                              )
# prepare a list fir colors
colorss = [sns.color_palette(cp, 6) for cp in ["RdPu_r", "Greens_r", "Orange

for season, colors in zip(["Spring", "Summer", "Autumn", "Winter"], colorss)
    ax = axd[season]

    df_season = df_dated[df_dated["Seasons"] == season]

    # Group the DataFrames into the time of day
    df_grouped = df_season.groupby(["Hour"])["Rented Bike Count"]

    # calculate the median for each hour
    median = df_grouped.median()

    # plot median
    ax.plot(median.index, median, label="Median", zorder=10, color=colors[0]

    # plot quantiles
    zorder = 9
    for quantile, color in zip([0.2, 0.4, 0.6, 0.8, 0.99], colors[1:]):
```

```
        dfmin = df_grouped.quantile((1-quantile)/2)
        dfmax = df_grouped.quantile(quantile+(1-quantile)/2)

        ax.fill_between(median.index, dfmin, dfmax, label=f"{quantile} quant
        zorder -= 1

    ax.set_xlim(0, 23)
    ax.set_xticks(np.arange(0,24, 2))
    ax.set_ylim(0, None)
    ax.set_title(season, loc="left")

# # Spesify labels and legends
axd["Winter"].legend()
fig.supxlabel("Hour of day")
fig.supylabel("Rented Bike Count")
```

Out[20]: Text(0.02, 0.5, 'Rented Bike Count')