Introduction

In this lab, we get some initial experience with using some of the main python tools for this course, including Numpy, Matplotlib and Pandas. We also load some datasets, compute some basic statistics on them and plot them.

The lab assumes that you are familiar with Python. Please complete `Week 01 lab: Introduction to python` before attempting this lab.

The lab can be executed on either your own machine (with anaconda installation) or lab computer.

- Please refer canvas for instructions on installing anaconda python and Jupyter Notebook.

Objective

- Continue to familiarise with Python and Jupyter Notebook
- Load dataset and examine the dataset
- Learn to compute basic statistics to understand the dataset more
- Plot the datasets to visually investigate the dataset

Dataset

We examine two regression based datasets in this lab. The first one is to do with house prices, some factors associated with the prices and trying to predict house prices. The second dataset is predicting the amount of share bikes hired every day in Washington D.C., USA, based on time of the year, day of the week and weather factors. These datasets are available in 'housing.data.csv' and 'bikeShareDay.csv' in the code repository.

First, ensure the two data files are located within the Jupyter workspace.

- If you are on the local machine copy the two data data directories ('BostonHousingPrice','Bike-Sharing-Dataset') to your current folder.

Load dataset to Python Notebook

Next we examine how to load these into Python and Jupyter notebooks. We will first analyse the House prices dataset, then you'll repeat the process to analyse the bike hire dataset.

First we need to import a few packages that will be used for our data loading and analysis. In python notebook you can load packages just before it is called (no need to load them at the start of the program).

Pandas is a great Python package for loading data. We will use Matplotlib to visualise some of the distributions. Numpy is a numeric library that has many useful matrices and mathematical functionality.

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
housedata = pd.read_csv(r"Lab\housing.data.csv", delimiter="\s+")
```

Replace the filename with the relative or absolute path to your files. We strongly encourage you to look up the documentation of the functions we use in the lab, in this case examine Pandas read_csv documentation.

The `read_csv()` command loads the input file, which is a csv formatted file delimited by tabs, into a **Pandas dataframe** (which can be thought of as a table). A dataframe can store the column names as well as the data. Examine what has been loaded into the dataframe `housedata`.

```
print(housedata)
```

If you are interested in checking only the first few rows of the dataframe to see if you have read the data in correctly, you can use the head method in dataframe.

```
housedata.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Now we have loaded the data into a data frame and printed it out, next we will compute some very basic statistics. The abbreviated column names:

- CRIM: per capita crime rate by town
- ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS: proportion of non-retail business acres per town
- CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX: nitric oxides concentration (parts per 10 million)
- RM: average number of rooms per dwelling
- AGE: proportion of owner-occupied units built prior to 1940
- DIS: weighted distances to five Boston employment centres
- RAD: index of accessibility to radial highways
- TAX: full-value property-tax rate per USD10,000
- PTRATIO: pupil-teacher ratio by town
- B: 1000 (Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT: lower status of the population
- MEDV: Median value of owner-occupied homes in USD1000's

The target column is **MEDV** and all the other columns are attributes.

Study the variables carefully and understand what they represent before moving to the next section.

Exploratory Data Analysis (EDA)

Often the first step in developing a machine learning solution for a given dataset is the EDA. EDA refers to the critical process of performing initial investigations on data so as to:

- Maximize insight into a data set;
- Uncover underlying structure;
- Extract important variables;
- Detect outliers and anomalies;
- Test underlying assumptions;
- Develop parsimonious models; and
- Determine optimal factor settings.

with the help of summary statistics and graphical representations. The particular graphical techniques employed in EDA are often quite simple, consisting of various techniques of:

- Plotting the raw data (such as data traces, histograms, bi-histograms, probability plots, lag plots, block plots, and Youden plots.
- Plotting simple statistics such as mean plots, standard deviation plots, box plots, and main effects plots of the raw data.
- Positioning such plots so as to maximize our natural pattern-recognition abilities, such as using multiple plots per page.

> ⚠ **Warning: EDA is a subjective process and will depend on the task & the data you have. There is no globally correct way of doing this.** Usually you need to have a good understanding of the task before deciding what EDA techniques to use and continuously refine them based on the observations you make in the initial steps. Since we are still at the beginning of the course, let's explore some commonly used techniques. You will understand the significance of these methods and observations in terms of ML in the next couple of weeks.

Let's first see the shape of the dataframe.

```
housedata.shape
```

```
(506, 14)
```

� What does the above output tell you?

It is also a good practice to know the columns and their corresponding data types, along with finding whether they contain null values or not.

```
housedata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In pandas, any missing values in the data (your input CSV file) is represented as NaN.

� Are there any missing values in the dataset?

Next let's compute some summary statistics of the data we have read.

```
pd.DataFrame.min(housedata)
```

```
CRIM         0.00632
ZN           0.00000
INDUS        0.46000
CHAS         0.00000
NOX          0.38500
RM           3.56100
AGE          2.90000
DIS          1.12960
RAD          1.00000
TAX        187.00000
PTRATIO     12.60000
B            0.32000
LSTAT        1.73000
MEDV         5.00000
dtype: float64
```

```
pd.DataFrame.min(housedata)
```

```
CRIM        88.9762
ZN         100.0000
INDUS       27.7400
CHAS         1.0000
NOX          0.8710
RM           8.7800
AGE        100.0000
DIS         12.1265
RAD         24.0000
TAX        711.0000
PTRATIO     22.0000
B          396.9000
LSTAT       37.9700
MEDV        50.0000
dtype: float64
```

```
pd.DataFrame.mean(housedata)
```

```
CRIM         3.613524
ZN          11.363636
INDUS       11.136779
CHAS         0.069170
NOX          0.554695
RM           6.284634
```

```
AGE         68.574901
DIS          3.795043
RAD          9.549407
TAX        408.237154
PTRATIO     18.455534
B          356.674032
LSTAT       12.653063
MEDV        22.532806
dtype: float64
```

```python
pd.DataFrame.median(housedata)
```

```
CRIM         0.25651
ZN           0.00000
INDUS        9.69000
CHAS         0.00000
NOX          0.53800
RM           6.20850
AGE         77.50000
DIS          3.20745
RAD          5.00000
TAX        330.00000
PTRATIO     19.05000
B          391.44000
LSTAT       11.36000
MEDV        21.20000
dtype: float64
```

The describe() function in pandas is very handy in getting various summary statistics. This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.

```python
housedata.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTR
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.4
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.1
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.6
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.4
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.0
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.2
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.0

Data comes in two principle types in statistics, and it is crucial that we recognize the differences between these two types of data.

1. Categorical Variables: These are data points that take on a finite number of values, AND whose values do not have a numerical interpretation.
   - Ordinal categorical variables take on values which can be logically ordered. For example, the reviews for a product which are given as 0-5 stars.
   - Nominal categorical variables cannot be put in any logical order. Examples of this would be the gender, race, etc.
2. Numerical Variables: These are variables which are numerical in nature
   - Continuous Variables: Take on a continuous values (no breaks). For example, height, weight.
   - Discrete numerical variables take on a set of values which can be counted. For example, the number of rooms in a house.

�Try to identify what type of data is in the `datahouse` dataframe.
�What is the type of data for CHAS and RAD?

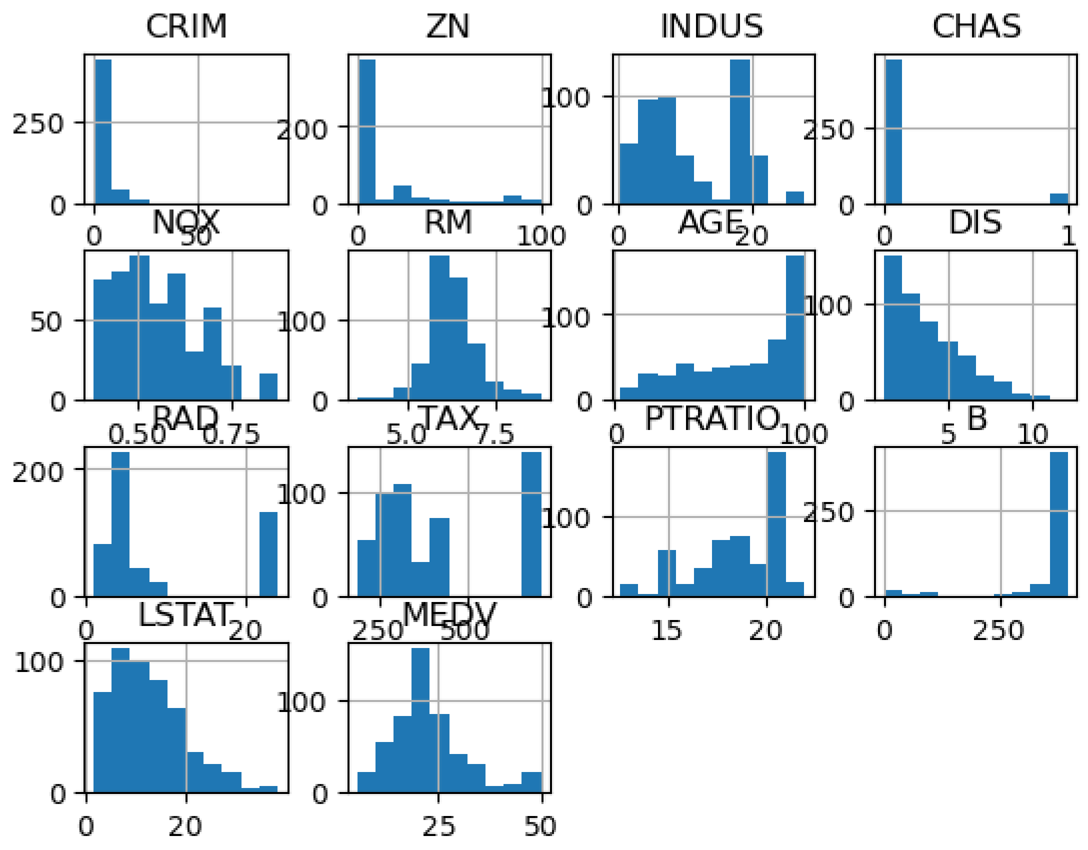
� What insights did you get from the above output? Look closely at attributes `ZN`, and `CHAS`, do you see a difference in those two compared to the others.

Data Distribution

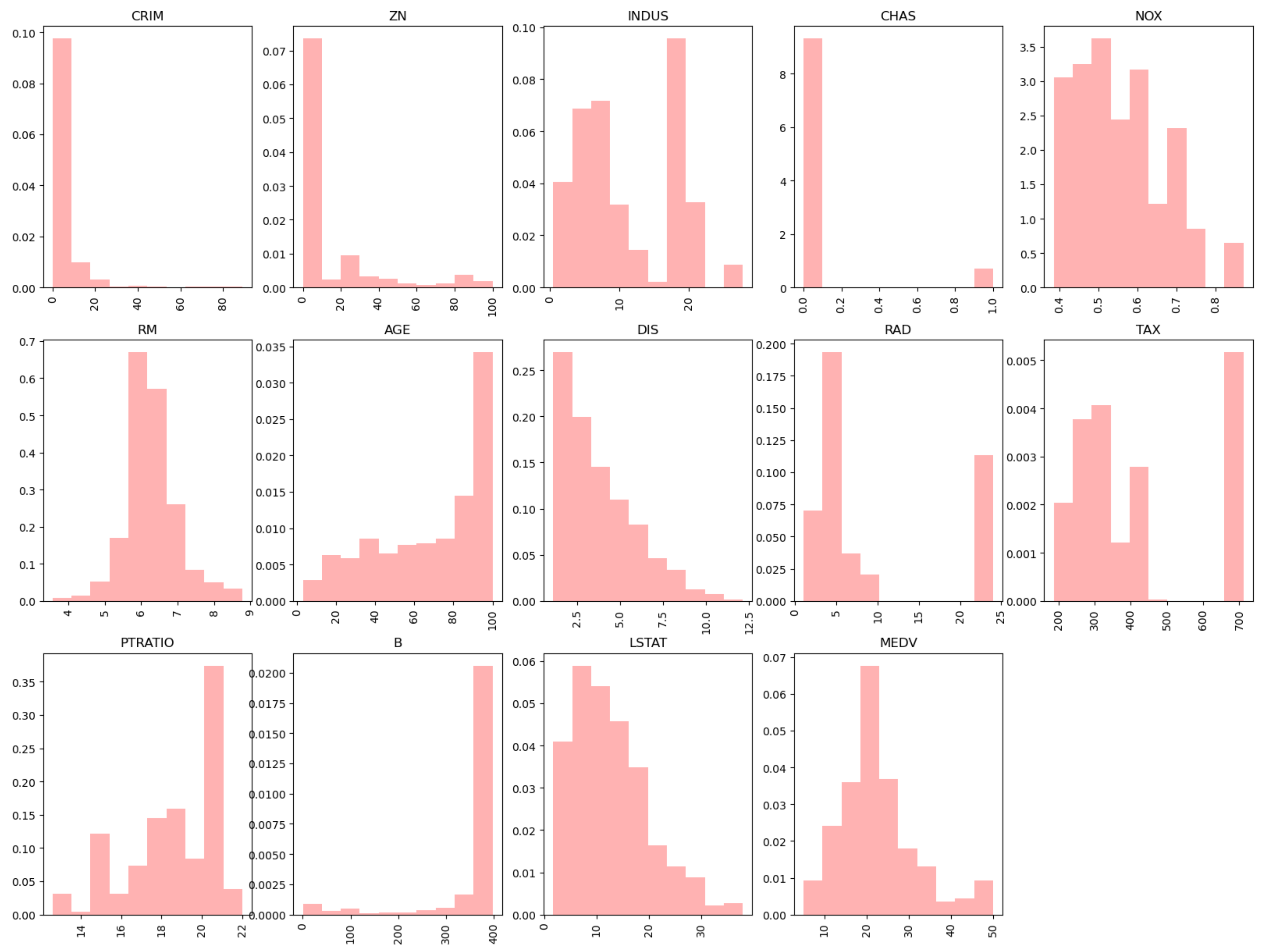
One of the most important step in EDA is estimating the distribution of a variable. Lets begin with histogram plot.

```
plt.figure(figsize=(20, 20), dpi=80)
housedata.hist()
plt.show()
```

```
<Figure size 1600x1600 with 0 Axes>
```



```
plt.figure(figsize=(20,20))
for i, col in enumerate(housedata.columns):
    plt.subplot(4,5,i+1)
    plt.hist(housedata[col], alpha=0.3, color='r', density=True)
    plt.title(col)
    plt.xticks(rotation='vertical')
```



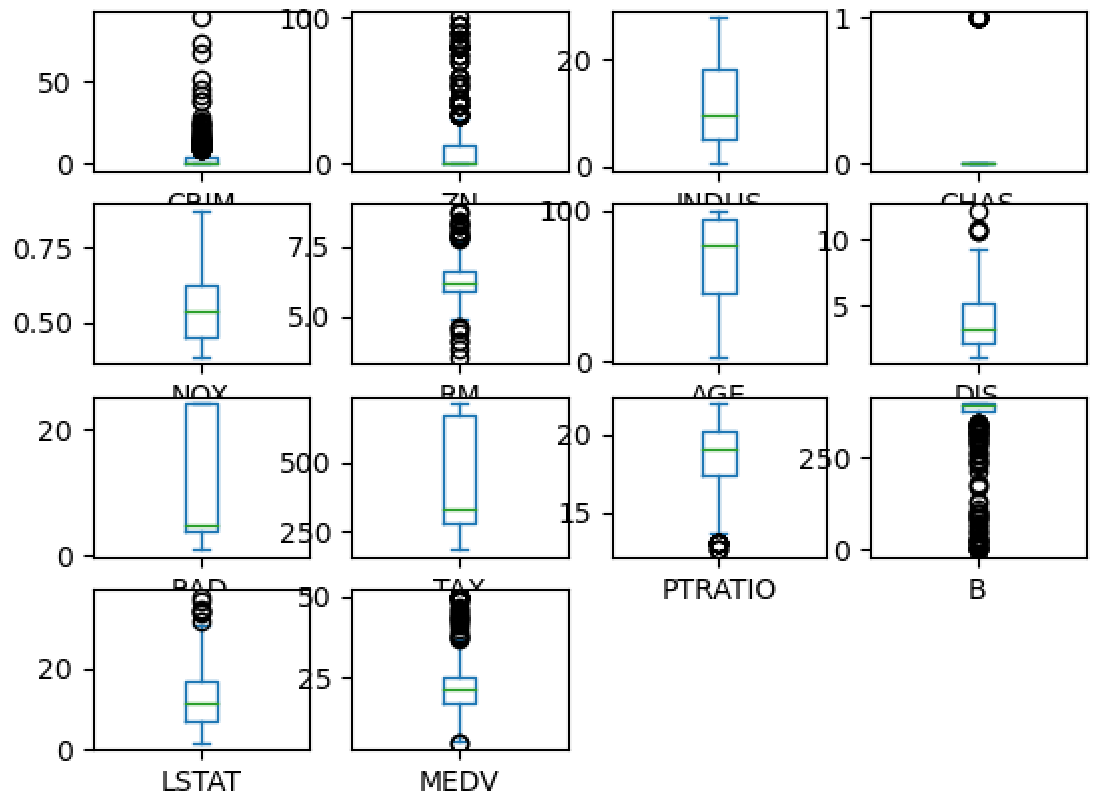
> ⚠ **Warning: Always question the bin sizes** of a histogram to see whether they are appropriate for the plot being presented. If you see a histogram with illogically large or small bin sizes and/or uneven bin sizes beware of the results being presented!

� What observations did you make?

> ✔ **Observations:**
>
> - Attribute CHAS is a categorical variable. Most data instances are from class 0 and only a few instances are from class 1.
> - Many attributes are heavily skewed. e.g. CRIM, ZN, DIS, AGE, B ...
> - Attributes RAD and TAX has values that are far from the majority values. Further investigations are needed.
> - Target variable MEDV is distributed around 22 with some extreme values around 50.
> - ...

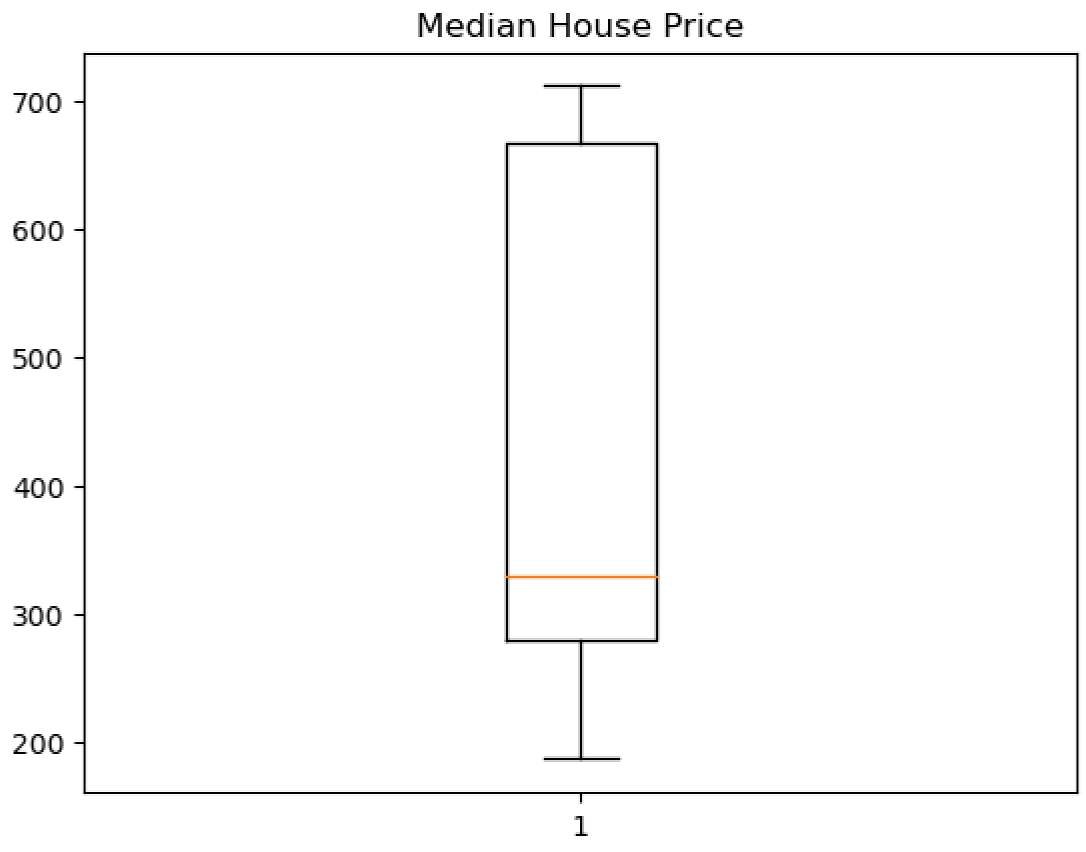
Box plot is another useful tool in examining the data. Lets use a box plot to observe our data.

```
housedata.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False)
plt.show()
```



You can focus on only one variable. Lets use a box plot to observe our target variable `MEDV`.

```python
plt.boxplot(housedata['TAX'])
plt.title('Median House Price')
plt.show()
```



How to read Box Plots:

- The thick line in the middle of the box gives the median value.
- The top of the box shows Quantile-.75
- The bottom of the box shows Quantile-.25
- So the height of the box in the Inter Quantile Range (IQR)
- The top whisker —| shows Q0.75+1.5∗IQR, the upper cutoff for outliers using Tukey's rule
- The bottom whisker —| shows Q0.25−1.5∗IQR, the lower cutoff for outliers using Tukey's rule
- Any data points (circles) show outlier values

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. In a sense, this definition leaves it up to the analyst (or a consensus process) to decide what will be considered abnormal. Before abnormal observations can be singled out, it is necessary to characterize normal observations.

Are there any outliers in datahouse???

Relationship between variable

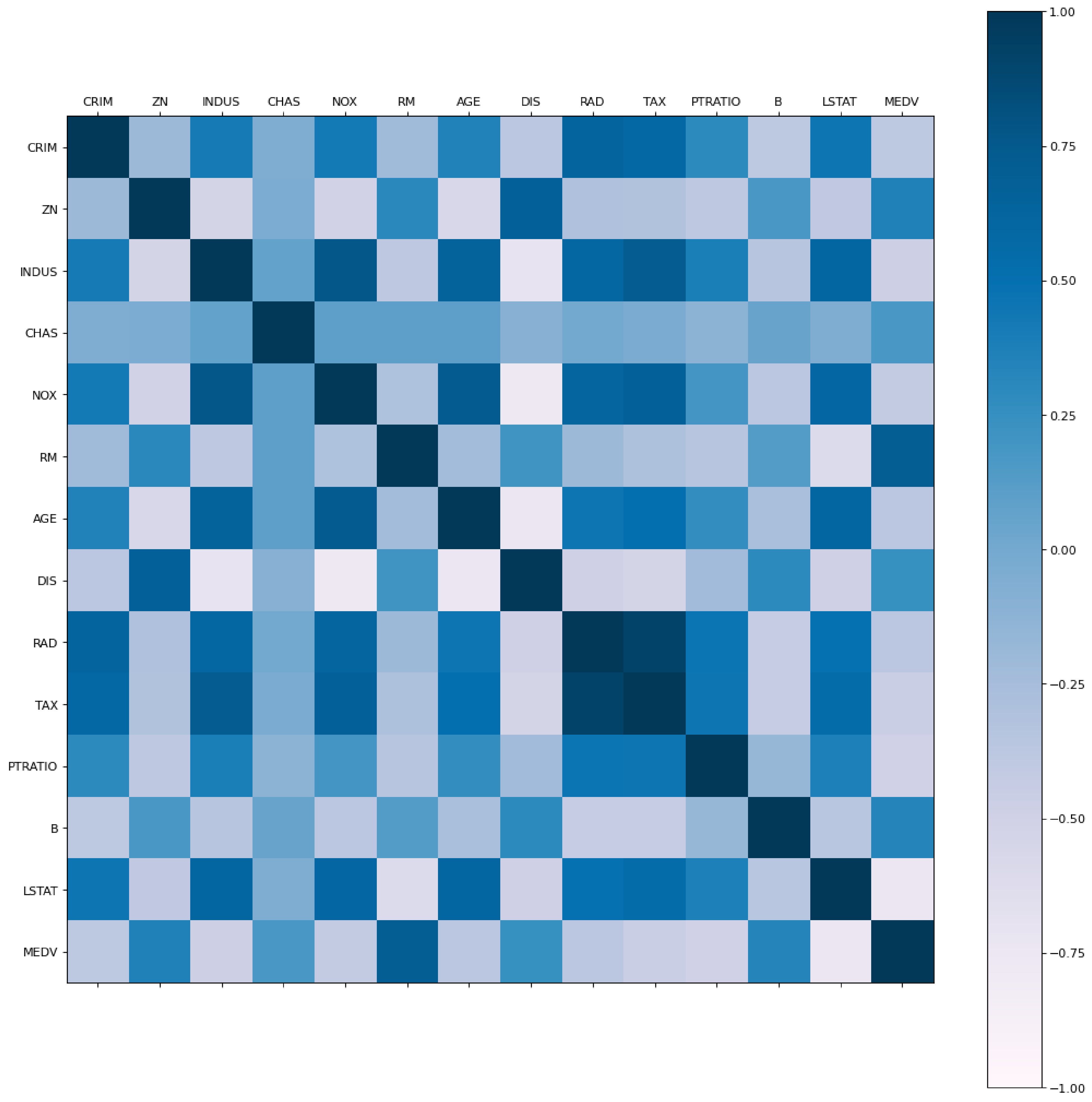
In the previous section we observed each attribute (data column) independently. Sometimes it is also useful to observe the relationship between two variables. There are several techniques that we can use for this purpose. One of the key techniques is a scatter plot.

Since our task is to predict MEDV (target variable) using all other attributes, let's plot the relationship between MEDV and other columns.

For this we can use matplotlib. However there is another python package called seaborn that plots nice looking figures. Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. You can learn more about seaborn at seaborn: statistical data visualization

Correlation is another important statistic when developing ML models. Lets plot the correlation matrix for the numerical data we have:

```
correlation = housedata.corr()
fig = plt.figure(figsize=(16, 16), dpi=80)
ax = fig.add_subplot(111)
cax = ax.matshow(correlation, vmin=-1, vmax=1, cmap=plt.cm.PuBu)
fig.colorbar(cax)
ticks = np.arange(0,14,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(housedata.columns)
ax.set_yticklabels(housedata.columns)
plt.show()
```



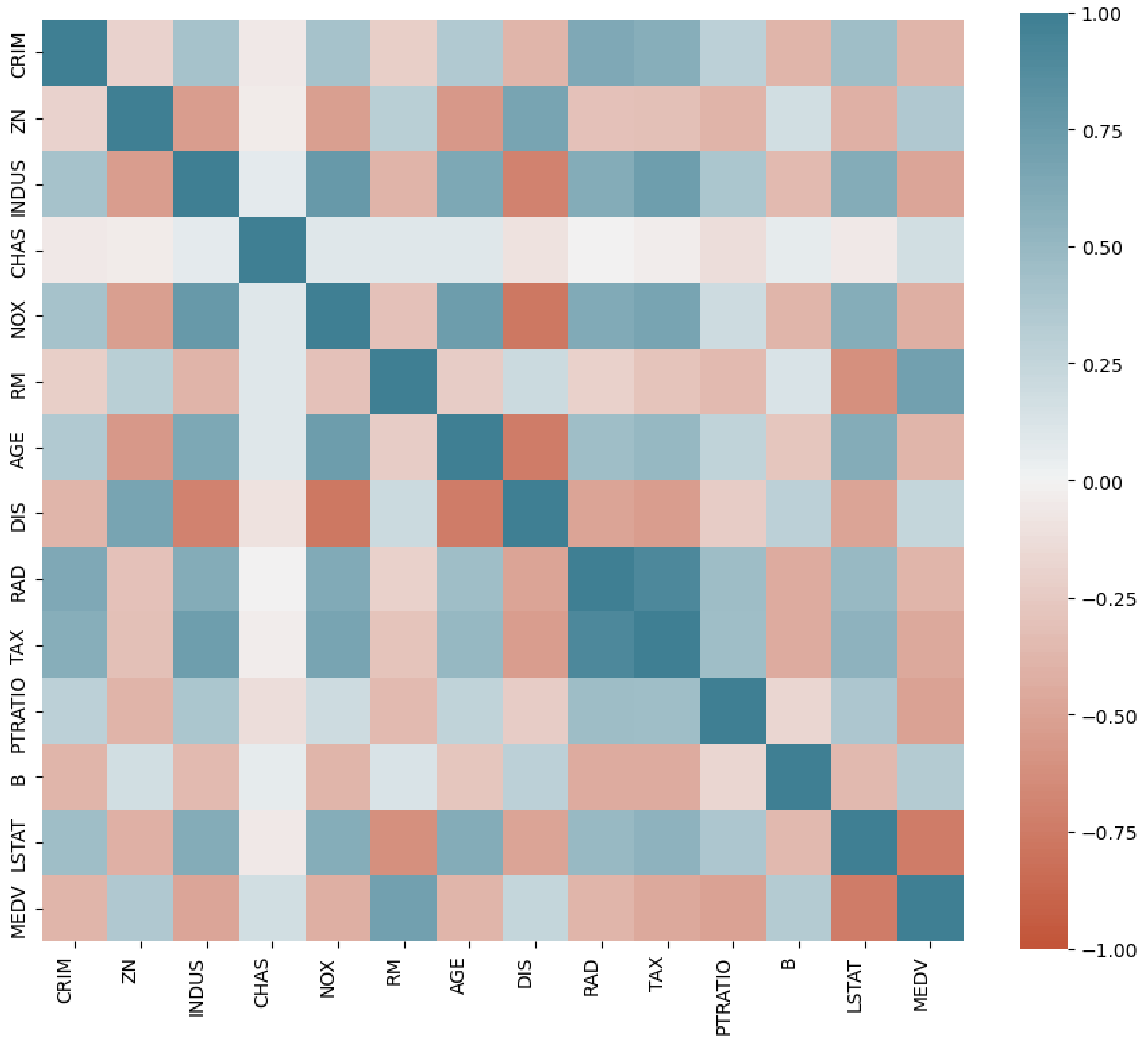
� What observations did you make?

> ✔ **Observations:**
>
> - There seems to be a good linear relationship between MEDV and RM.
> - The relationship between MEDV and some variables appears to be nonlinier (e.g. LSAT).
> - ...

If you want to use seaborn, you can try

```
import seaborn as sns

f, ax = plt.subplots(figsize=(11, 9))
corr = housedata.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=90,
    horizontalalignment='right'
);
```



Exercise: Analyse the Bike Share Data

✍ Task: Analyse the Bike Share Data.
Now you seen how to do this task for the Bike Sharing (rentals) dataset. Repeat the same process for the Daily Bike Share rental data.

Answer the following questions and discuss this with your lab demonstrator. Please do attempt this, and don't wait to see if solutions are released (they will not be!)

- What is the range of some of the attributes?
- Which of the features have a very different average to the others?
- Which feature is skewed (hint examine the histogram)?
- Which features are highly correlated?

> Relate above questions back to the domain of the dataset (bike sharing) and see if you can come up with explanations for the observed data.

```
# Exercise: Analyze the Bike Share Data

# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

"""
This function reads the Bike Share data from a CSV file,
displays the shape of the dataframe, and returns the dataframe.
"""
# Read the Bike Share data
bike_data = pd.read_csv(r'Lab\bikeShareDay-1.csv')

print(bike_data)

# Display the first few rows of the dataframe
bike_data.head()

# Display the shape of the dataframe
print(bike_data.shape)



```

```
     instant      dteday  season  yr  mnth  holiday  weekday  workingday  \
0          1  2011-01-01       1   0     1        0        6           0
1          2  2011-01-02       1   0     1        0        0           0
2          3  2011-01-03       1   0     1        0        1           1
3          4  2011-01-04       1   0     1        0        2           1
4          5  2011-01-05       1   0     1        0        3           1
..       ...         ...     ...  ..   ...      ...      ...         ...
726      727  2012-12-27       1   1    12        0        4           1
727      728  2012-12-28       1   1    12        0        5           1
728      729  2012-12-29       1   1    12        0        6           0
729      730  2012-12-30       1   1    12        0        0           0
730      731  2012-12-31       1   1    12        0        1           1

     weathersit      temp     atemp       hum  windspeed  casual  registered  \
0             2  0.344167  0.363625  0.805833   0.160446     331         654
1             2  0.363478  0.353739  0.696087   0.248539     131         670
2             1  0.196364  0.189405  0.437273   0.248309     120        1229
3             1  0.200000  0.212122  0.590435   0.160296     108        1454
4             1  0.226957  0.229270  0.436957   0.186900      82        1518
..          ...       ...       ...       ...        ...     ...         ...
726           2  0.254167  0.226642  0.652917   0.350133     247        1867
727           2  0.253333  0.255046  0.590000   0.155471     644        2451
728           2  0.253333  0.242400  0.752917   0.124383     159        1182
729           1  0.255833  0.231700  0.483333   0.350754     364        1432
730           2  0.215833  0.223487  0.577500   0.154846     439        2290

      cnt
0     985
1     801
2    1349
3    1562
4    1600
..    ...
726  2114
727  3095
728  1341
729  1796
730  2729

[731 rows x 16 columns]
(731, 16)
```

First, the necessary libraries are imported. Pandas is used for data manipulation and analysis, matplotlib is used for creating static, animated, and interactive visualizations in Python, and seaborn is a Python data visualization library based on matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics.

Next, a CSV file containing the bike share data is read into a pandas DataFrame using the `pd.read_csv()` function. The file path to the CSV file is provided as a raw string to this function. The resulting DataFrame, `bike_data`, holds the bike share data that will be analyzed.

After loading the data, the code prints the entire DataFrame to the console using the `print()` function. This is useful for getting a quick overview of the data, but could be problematic if the dataset is very large.

The `head()` method of the DataFrame is then called, which returns the first five rows of the DataFrame. This is a common way to get a quick look at the structure of the data and the types of values in each column.

Finally, the shape of the DataFrame is printed to the console. The `shape` attribute of a DataFrame returns a tuple representing the dimensions of the DataFrame. The first element of the tuple is the number of rows and the second element is the number of columns. This gives a quick sense of the size of the dataset.

```
bike_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   instant     731 non-null    int64  
 1   dteday      731 non-null    object 
 2   season      731 non-null    int64  
 3   yr          731 non-null    int64  
 4   mnth        731 non-null    int64  
 5   holiday     731 non-null    int64  
 6   weekday     731 non-null    int64  
 7   workingday  731 non-null    int64  
 8   weathersit  731 non-null    int64  
 9   temp        731 non-null    float64
 10  atemp       731 non-null    float64
 11  hum         731 non-null    float64
 12  windspeed   731 non-null    float64
 13  casual      731 non-null    int64  
 14  registered  731 non-null    int64  
 15  cnt         731 non-null    int64  
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

```
bike_data.describe()
```

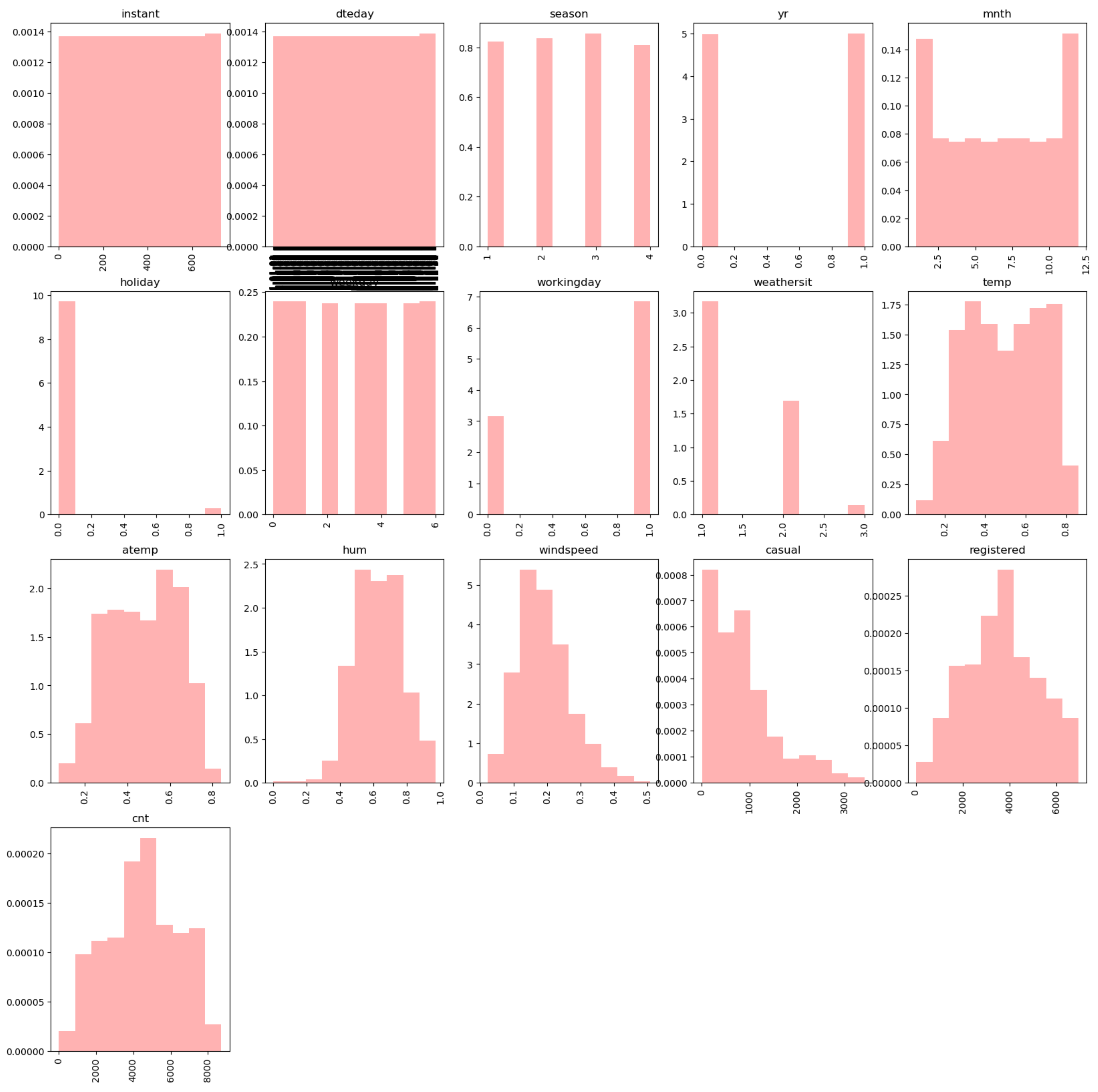
	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	73
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349	0.495385	0.474354	
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894	0.183051	0.162961	
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.059130	0.079070	
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	0.337083	0.337842	
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	1.000000	0.498333	0.486733	
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	0.655417	0.608602	
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	0.861667	0.840896	

```
"""
This code block creates a figure with multiple subplots, each displaying a histogram of a column in th

Parameters:
- bike_data: DataFrame containing the data to be plotted

Returns:
- None
"""

plt.figure(figsize=(20,20))
for i, col in enumerate(bike_data.columns):
    plt.subplot(4,5,i+1)
    plt.hist(bike_data[col], alpha=0.3, color='r', density=True)
    plt.title(col)
    plt.xticks(rotation='vertical')
```



The provided Python code is used to create a series of histograms for each column in the `bike_data` DataFrame. This is a common task in exploratory data analysis, as it allows you to quickly visualize the distribution of values in each column.

The `plt.figure(figsize=(20,20))` line is used to create a new figure for plotting, with a specified size of 20x20 units. This size is chosen to ensure that the individual subplots do not overlap and are large enough to be clearly visible.

Next, the `enumerate()` function is used in a for loop to get both the index (`i`) and the value (`col`) for each column in the DataFrame. The `enumerate()` function adds a counter to an iterable and returns it as an enumerate object. This can be useful when you need to have access to the index of the items in the loop.

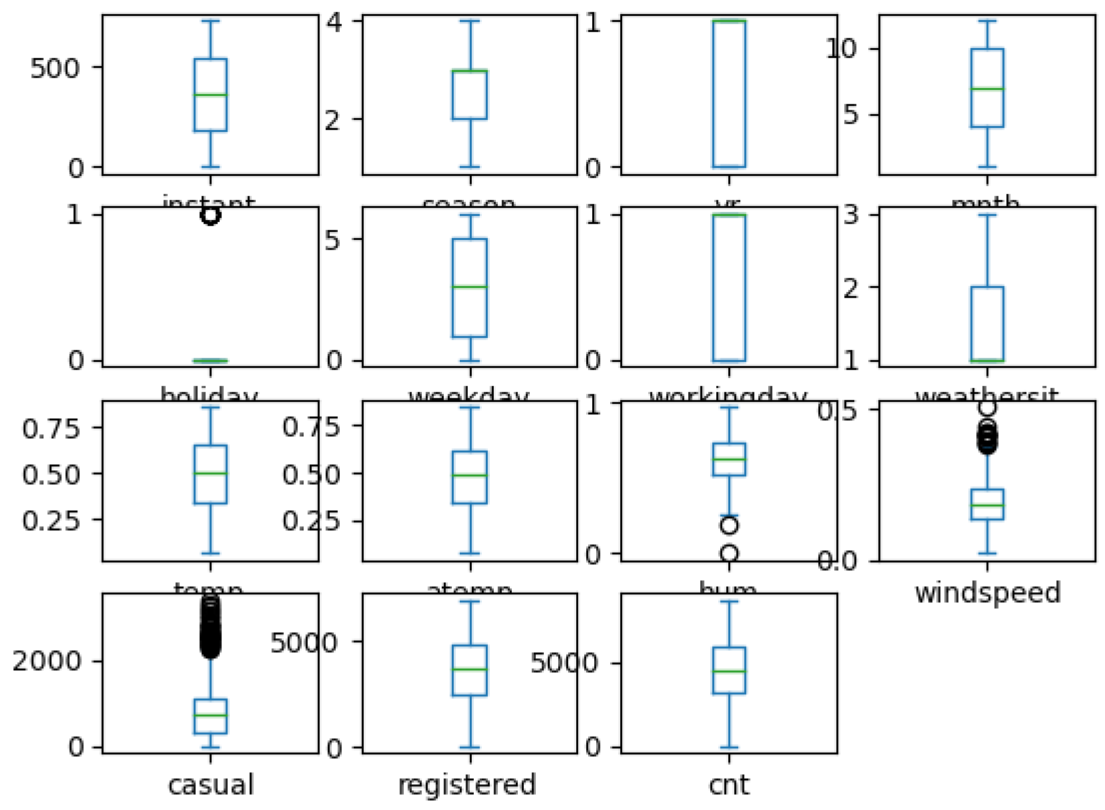
Inside the loop, `plt.subplot(4,5,i+1)` is used to create a subplot in a 4x5 grid for each column. The `i+1` is used to specify the position of the subplot in the grid, starting from 1.

The `plt.hist()` function is then used to create a histogram for the current column. The `alpha` parameter is used to specify the transparency of the histogram, the `color` parameter is used to specify the color of the histogram, and the `density` parameter is used to normalize the histogram.

The `plt.title(col)` line is used to set the title of the subplot to the name of the current column.

Finally, `plt.xticks(rotation='vertical')` is used to rotate the x-axis labels vertically. This can be useful when the labels are long or numerous, as it prevents them from overlapping.

```
bike_data.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False)
plt.show()
```



```
# Range of attributes
numeric_data = bike_data.select_dtypes(include=[np.number])  # Select only numerical columns
attribute_range = numeric_data.max() - numeric_data.min()
print("Range of attributes:")
print(attribute_range)

```

```
Range of attributes:
instant        730.000000
season           3.000000
yr               1.000000
mnth            11.000000
holiday          1.000000
weekday          6.000000
workingday       1.000000
weathersit       2.000000
temp             0.802537
atemp            0.761826
hum              0.972500
windspeed        0.485071
casual        3408.000000
registered    6926.000000
cnt           8692.000000
dtype: float64
```

The provided Python code is used to calculate the range of each numerical attribute in a DataFrame named `bike_data`. The range of an attribute is the difference between its maximum and minimum values, and it gives an idea of how spread out the values are.

First, the `select_dtypes()` method of the DataFrame is used to select only the columns with numerical data. The `include` parameter is set to `[np.number]`, which means that only columns with a data type that is a subtype of numpy's number (which includes integer and float types) will be selected. The result is a new DataFrame, `numeric_data`, that includes only the numerical columns from `bike_data`.

Next, the `max()` and `min()` methods of the DataFrame are used to calculate the maximum and minimum values of each column, respectively. The difference between these two Series (the maximum and minimum values) is calculated using the `-` operator, which performs element-wise subtraction when used with pandas Series. The result is a Series where the value for each attribute is the range of that attribute.

Finally, the range of each attribute is printed to the console using the `print()` function. The string "Range of attributes:" is printed first to provide a label for the output, and then the Series containing the range of each attribute is printed. This output provides a quick overview of the range of each numerical attribute in the DataFrame.

```
# Features with a different average
average_diff = bike_data.mean(numeric_only=True) - bike_data.mean(numeric_only=True).mean()
print("Features with a different average:")
print(average_diff)
```

```
Features with a different average:
instant       -260.073882
season        -623.577302
yr            -625.573198
mnth          -619.554046
holiday       -626.045154
weekday       -623.076618
workingday    -625.389887
weathersit    -624.678533
temp          -625.578497
atemp         -625.599528
hum           -625.445988
windspeed     -625.883396
casual         222.102589
registered    3030.098485
cnt           3878.274955
dtype: float64
```

The provided Python code is used to identify the features in the `bike_data` DataFrame that have a different average compared to the others.

First, the `mean()` method of the DataFrame is called with the `numeric_only=True` parameter. This calculates the mean (average) of each numerical column in the DataFrame. The `numeric_only=True` parameter ensures that the mean is only calculated for columns with numerical data. The result is a pandas Series where the index is the column names and the values are the mean values of those columns.

Next, the mean of these mean values is calculated by calling the `mean()` method again on the result. This gives the average of the average values of all numerical columns.

The difference between the mean of each column and the overall mean is then calculated using the `-` operator. This performs element-wise subtraction, resulting in a Series where the value for each attribute is the difference between its mean and the overall mean.

Finally, this Series is printed to the console using the `print()` function. The string "Features with a different average:" is printed first to provide a label for the output. The output provides a quick overview of which features have a mean that is different from the mean of the means of all features.

```
# Skewed feature
skewed_feature = bike_data.skew(numeric_only=True).idxmax()
print("Skewed feature:")
print(skewed_feature)
```

```
Skewed feature:
holiday
```

The provided Python code is used to identify the most skewed numerical feature in the `bike_data` DataFrame.

First, the `skew()` method of the DataFrame is called with the `numeric_only=True` parameter. This calculates the skewness of each numerical column in the DataFrame. Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The `numeric_only=True` parameter ensures that skewness is only calculated for columns with numerical data. The result is a pandas Series where the index is the column names and the values are the skewness values of those columns.

Next, the `idxmax()` method is called on the result. This returns the index (in this case, the column name) of the maximum value in the Series. In other words, it identifies the column with the highest skewness value.

Finally, the name of the most skewed feature is printed to the console using the `print()` function. The string "Skewed feature:" is printed first to provide a label for the output, and then the name of the most skewed feature is printed. This output provides a quick way to identify which feature in the DataFrame is the most skewed.

```
# Highly correlated features
```