In the name of Friend

Introducing to data Analysis with python without further explanation

MAHDI MOMENI

Contents

Author's words

I am Mahdi Momeni, a student of computer engineering. I don't have any high claims in the field of data analysis, and I'm just trying to teach others what I know. While writing this small book, I reviewed various sources and made every effort to summarize the most important and most practical topics in a very simple language in this book. During the time of writing this book, I am someone who is learning and I am not a master in these topics, so you may see various mistakes in this book, for which I apologize to you very much.

The assumption of this book is that you have some knowledge of programming in Python, and some knowledge of statistics and probability.
I have made every effort to explain the topics without additional explanation so that you are ready to start your project and analyze your data by reading these topics.
It's important to remember that everyone starts somewhere, and it's through making mistakes and continuous learning that we improve. I hope this book could be a valuable resource for beginners in data analysis, especially those familiar with Python and have some understanding of statistics and probability.

Data analysis is a broad field with many sub-disciplines, including data cleaning, exploratory data analysis, inferential statistics, predictive analytics, and data visualization, among others. Each of these areas has its own set of techniques and methodologies. For example, Python libraries like pandas, numpy, and matplotlib are often used for data manipulation and visualization, while libraries like scikit-learn, Pytorch and TensorFlow are used for machine learning.

Remember, the goal of data analysis is to discover useful information, suggest conclusions, and support decision-making. This can be applied across a wide range of fields, including business, science, social science, and more. Keep learning and exploring, and don't be afraid to make mistakes along the way. They're just stepping stones on your path to mastery.

In this book, we will first examine the required libraries and their commonly used functions, then we will explain important statistical concepts. *I have good news for you, and that is at the end of the book I am preparing a roadmap for you, which you can go through its steps for analyzing any data and reach the best analysis.*

be successful and victorious.

Types of data

1. **Text Data:** Text data, also known as unstructured data, is non-numerical and often language-based. It includes emails, social media posts, customer reviews, etc. Text data is analyzed using techniques like Natural Language Processing (NLP) to extract meaningful insights.

2. **Excel Data:** Excel is a powerful tool for data analysis. It provides features like pivot tables, conditional formatting, and various statistical functions to analyze data. Excel also has a feature called “Analyze Data” that can provide high-level visual summaries, trends, and patterns.

3. **CSV Data:** CSV (Comma-Separated Values) files are widely used for storing and exchanging tabular data. They are compatible with various data analysis tools and languages, such as Python, R, and SQL. Analysts can easily manipulate and analyze data stored in CSV files.

In addition to these, data in data analysis can be broadly categorized into two types: Discrete and Continuous.

- **Discrete Data:** Discrete data can only assume specific values that you cannot subdivide. Typically, you count them, and the results are integers. For example, the number of students in a class, the number of chocolates in a bag, the number of strings on a guitar, the number of fishes in the aquarium.

- **Continuous Data:** Continuous data can assume any numeric value and can be meaningfully split into smaller parts. They have valid fractional and decimal values. Examples of continuous data include weight, height, length, time, and temperature.

11 Introduction to NumPy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing of array. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

Why Numpy
NumPy arrays are densely packed arrays of homogeneous type. Python lists, by contrast, are arrays of pointers to objects, even when all of them are of the same type. So, you get much more speed and efficiency when performing operations with NumPy.

Most Useful Functions

1. **Array Creation**
   - **`np.array()`:** *Creates a new array.*
   - **`np.zeros()`:** *Creates an array filled with zeros.*
   - **`np.ones()`:** *Creates an array filled with ones.*
   - **`np.empty()`:** *Creates an array filled with ones.*
   - **`np.full()`:** *Creates an array filled with a specified value.*
   - **`np.arange()`:** *Creates an array with a range of elements.*
   - **`np.linspace()`:** *Creates an array with evenly spaced elements between specified start and end values.*
2. **Array Mainpulation**
   - **`np.reshape()`:** *Changes the shape of an array without changing its data.*
   - **`np.revel()`:** *Returns a flattened array.*
   - **`np.transpose()`***: Permute the dimensions of an array.*
   - **`np.hstack()`:** *Stacks arrays in sequence horizontally (column wise).*
   - **`np.vstack()`***: Stacks arrays in sequence vertically (row wise).*
3. **Mathematical Functions**
   - **`np.add()`:** *Element-wise addition of arrays.*

- **`np.subtract()`:** *Element-wise subtraction of arrays.*
- **`np.multiply()`:** *Element-wise multiplication of arrays.*
- **`np.divide()`***: Element-wise division of arrays.*
- **`np.sqrt()`:** *Square root of each element in the array.*
- **`np.sin()`:** *Trigonometric sine of each element in the array.*
- **`np.cos()`:** *Trigonometric cosine of each element in the array.*
- **`np.tan()`:** *Trigonometric tangent of each element in the array.*
- **`np.log()`:** *Natural logarithm of each element in the array.*
- **`np.dot()`:** *Dot product of two arrays.*

4. **Statistical Functions**
   - **`np.min()`:** *Returns the minimum value of an array.*
   - **`np.max()`:** *Returns the maximum value of an array.*
   - **`np.sum()`:** *Returns the sum of array elements.*
   - **`np.mean()`:** *Returns the arithmetic mean of array elements.*
   - **`np.median()`:** *Returns the median of array elements.*
   - **`np.std()`***: Returns the standard deviation of array elements.*
5. **Linear Algebra**
   - **`np.linalg.inv()`:** *Compute the (multiplicative) inverse of a matrix.*
   - **`np.linalg.det()`:** *Compute the determinant of an array.*
   - **`np.linalg.eig()`:** *Compute the eigenvalues and right eigenvectors of a square array.*

Examples

```
import numpy as np

1# . Array Creation
a = np.array([1, 2, 3])
print(“a:”, a) # Output: a: [1 2 3]

b = np.zeros((3,3))
print("b:", b) # Output: b: [[0. 0. 0.] [0. 0. 0.] [0. 0. 0.]]

c = np.ones((3,3))
print("c:", c) # Output: c: [[1. 1. 1.] [1. 1. 1.] [1. 1. 1.]]

d = np.empty((3,3))
```

```
print("d:", d) # Output: d: [[1 1 1] [1 1 1] [1 1 1]]

e = np.full((3,3), 7)
print("e:", e) # Output: e: [[7 7 7] [7 7 7] [7 7 7]]

f = np.arange(10)
print("f:", f) # Output: f: [0 1 2 3 4 5 6 7 8 9]

g = np.linspace(0, 1, 5)
print("g:", g) # Output: g: [0. 0.25 0.5 0.75 1. ]

# 2. Array Manipulation
h = np.reshape(a, (1, 3))
print("a:", a)
print("h:", h) # Output: a: [1 2 3] h: [[1 2 3]]

i = np.ravel(b)
print("i:", i) # Output: i: [0. 0. 0. 0. 0. 0. 0. 0. 0.]

array = [[1,2,3],[4,5,6]]
j = np.transpose(array)
print("j:", j) # Output: j: [[1. 1. 1.] [1. 1. 1.] [1. 1. 1.]]

# 3 . Mathematical Functions
k = np.add(a, a)
print("k:", k) # Output: k: [2 4 6]
l = np.subtract(a, a)
print("l:", l) # Output: l: [0 0 0]

m = np.multiply(a, a)
print("m:", m) # Output: m: [1 4 9]

n = np.divide(a, a)
print("n:", n) # Output: n: [1. 1. 1.]

o = np.sqrt(a)
print("o:", o) # Output: o: [1. 1.41421356 1.73205081]

p = np.min(a)
print("p:", p) # Output: p: 1

q = np.max(a)
print("q:", q) # Output: q: 3

r = np.sum(a)
print("r:", r) # Output: r: 6

s = np.mean(a)
print("s:", s) # Output: s: 2.0
```

```
t = np.median(a)
print("t:", t) # Output: t: 2.0

u = np.std(a)
print("u:", u) # Output: u: 0.816496580927726
```

12 Introduction to Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. It provides data structures and functions needed to manipulate structured data, including functions for reading and writing diverse files, data cleaning and reshaping, merging datasets, and data summarization, among other features.

Most Useful Functions

1. **Data Structure Creation**
   - **`pd.Series()`:** *Creates a one-dimensional labeled array capable of holding any data type.*
   - **`pd.DataFrame()`:** *Creates a one-dimensional labeled array capable of holding any data type.*
2. **Data Import/Export**
   - **`pd.read_csv()`:** *Reads a comma-separated values (csv) file.*
   - **`pd.to_csv()`:** *Writes DataFrame to a comma-separated values (csv) file.*
   - **`pd.read_excel()`***: Reads an Excel file into DataFrame.*
   - **`pd.to_sxcel()`:** *Writes DataFrame to an Excel file.*
3. **Data Selection**
   - **`df.loc[]`:** *Access a group of rows and columns by label(s) or a boolean array.*
   - **`df.iloc[]`:** *Purely integer-location based indexing for selection by position.*
4. **Data Cleaning**
   - **`df.dropna()`:** *Removes missing values.*
   - **`df.fillna()`:** *Fills missing values.*
   - **`df.replace()`***: Replaces a specific value with another value.*
5. **Data Manipulation**
   - **`df.set_index()`:** *Sets the DataFrame index using existing columns.*

- **`df.reset_index()`:** *Resets the index of the DataFrame.*
- **`df.sort_values()`:** *Sorts a DataFrame along either axis.*

6. **Data Aggregation**
   - **`df.groupby()`:** *Groups DataFrame using a mapper or by a Series of columns.*
   - **`df.pivot_table()`:** *Creates a spreadsheet-style pivot table as a DataFrame.*
7. **Statistical Functions**
   - **`df.mean()`:** *Returns the mean of all columns.*
   - **`df.median()`:** *Returns the median of all columns.*
   - **`df.std()`:** *Returns the standard deviation of all columns.*
8. **String Methods**
   - **`df['column'].str.lower()`:** *Converts strings in the Series/Index to lower case.*
   - **`df['column'].str.upper()`:** *Converts strings in the Series/Index to upper case.*
   - **`df['column'].str.strip()`:** *Helps strip whitespace(including newline) from each string in the Series/index from both the sides.*

Examples

```
# 1. Data Structure Creation
s = pd.Series([1, 2, 3])
print(s.head())
# Output:
# 0 1
# 1 2
# 2 3
# dtype: int64

df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]), columns=['a', 'b', 'c'])
print(df.head())
# Output:
# a b c
# 0 1 2 3
# 1 4 5 6

# 2. Data Import/Export
# df.to_csv('data.csv') # Writes DataFrame to a csv file
# df = pd.read_csv('data.csv') # Reads a csv file into DataFrame

# 3. Data Selection
print("df.loc[0]:\n", df.loc[0])
# Output: a 1
```

```
# b 2
# c 3
# Name: 0, dtype: int64

print("df.iloc[0]:\n", df.iloc[0])
# Output: a 1
# b 2
# c 3
# Name: 0, dtype: int64

# 4. Data Cleaning
df2 = df.replace(1, 10)
print(df2)
# Output:
# a b c
# 0 10 2 3
# 1 4 5 6

# 5. Statistical Functions
print("df.mean():\n", df.mean())
# df.mean():
# a 2.5
# b 3.5
# c 4.5
# dtype: float64

# 6. String Methods
df_str = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog',
'cat'])
df_str_lower = df_str.str.lower()
print(df_str_lower)

# 0 a
# 1 b
# 2 c
# 3 aaba
# 4 baca
# 5 NaN
# 6 caba
# 7 dog
# 8 cat
# dtype: object
```

13 Introduction to MatPlotLib

Matplotlib is a plotting library for the Python programming language. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. Matplotlib is also a popular library for creating static, animated, and interactive visualizations in Python.

Why Matplotlib
Matplotlib is one of the most popular Python packages used for data visualization. It provides a quick way to visualize data from Python and publication-quality figures in many formats. It is also highly customizable, allowing you to create rich visualizations of complex data.

Most Useful Functions

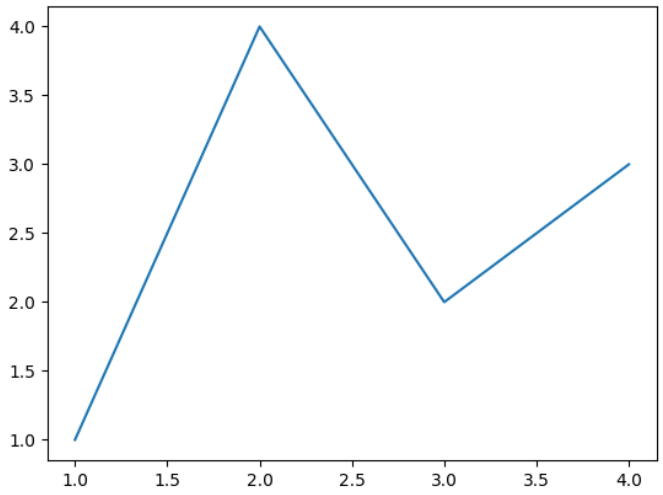
1. **Figure Creation**
   - **`plt.figure()`:** *Creates a new figure.*
   - **`plt.subplots()`:** *Creates a figure and a set of subplots.*
2. **Figure Creation**
   - **`plt.plot()`:** *Plots y versus x as lines and/or markers.*
   - **`plt.scatter()`:** *Makes a scatter plot of x vs y.*
   - **`plt.bar()`:** *Makes a bar plot.*
   - **`plt.hist()`:** *Plots a histogram.*
   - **`plt.boxplot()`:** *Makes a box and whisker plot.*
3. **Customizing Plots**
   - **`plt.title()`:** *Sets a title for the axes.*
   - **`plt.xlabel()`:** *Sets the label for the x-axis.*
   - **`plt.ylabel()`:** *Sets the label for the y-axis.*
   - **`plt.grid()`:** *Configures the grid lines.*
   - **`plt.legend()`:** *Places a legend on the axes.*
   - **`plt.xticks()`:** *Gets or sets the current tick locations and labels of the x-axis.*

- **`plt.yticks()`:** *Gets or sets the current tick locations and labels of the y-axis.*
- **`plt.axis()`:** *customize Chart range. Sample: plt.axis([-20,20, -20, 20])*

4. **Showing & Saving Plots**
   - **`plt.show()`**: *Displays a figure.*
   - **`plt.savefig()`:** *Saves the current figure.*

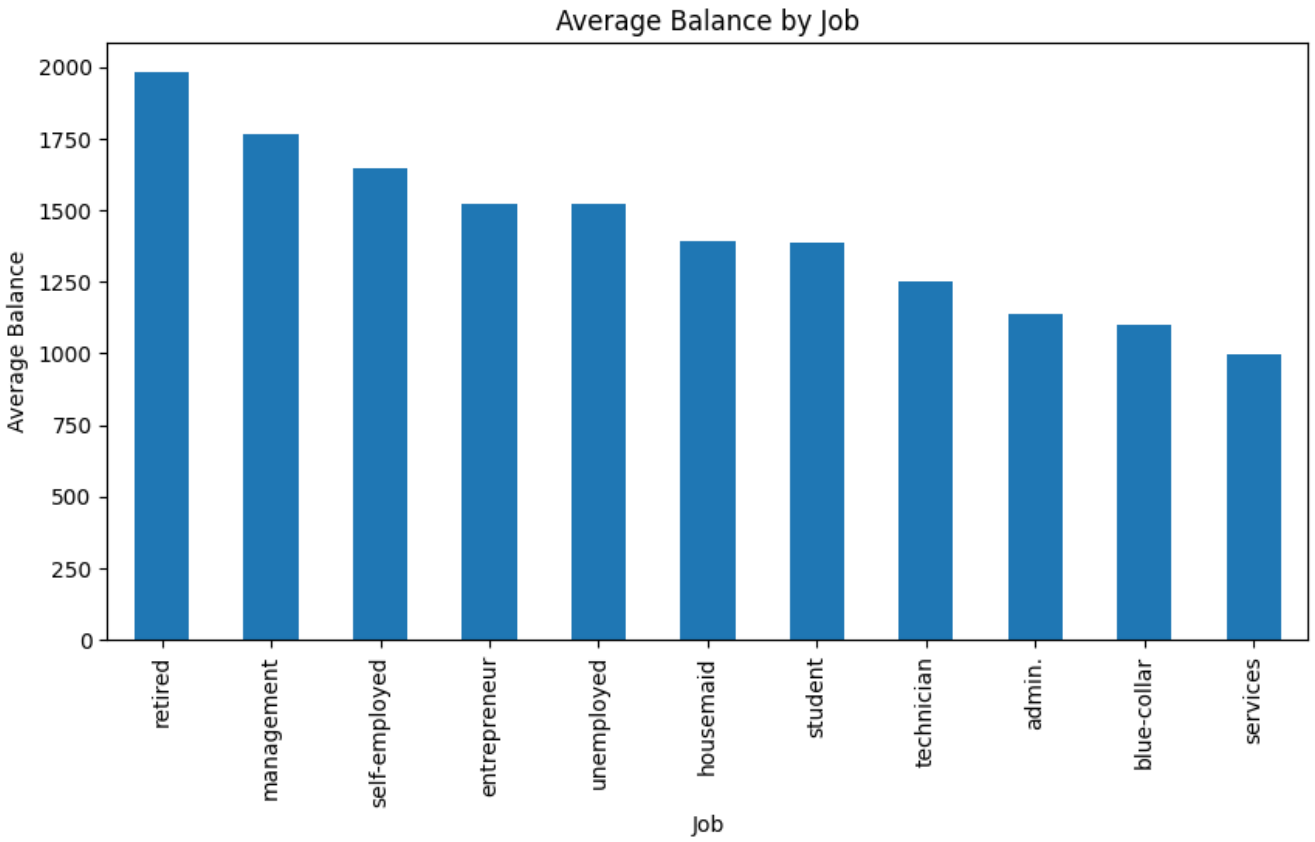
Examples

```
# Data
x = [1, 2, 3, 4]
y = [1, 4, 2, 3]

# Create a figure and a set of subplots
fig, ax = plt.subplots()

# Plot y versus x as lines and/or markers
ax.plot(x, y)

# Display the figure
plt.show()
```



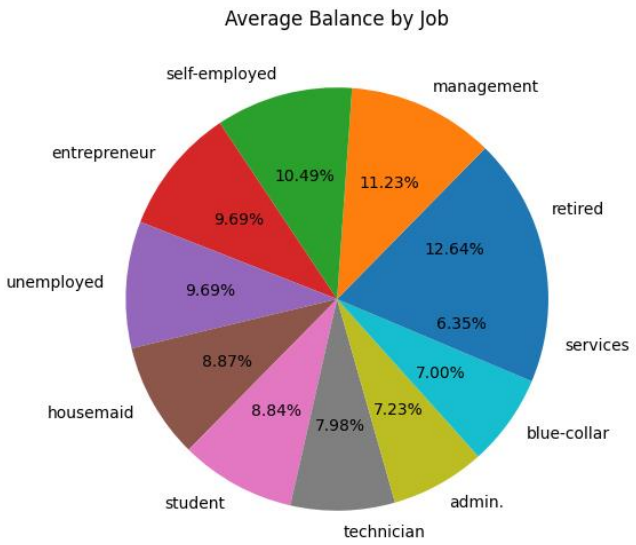
Display a column chart of balance based on job.

```
df = bankdf

job_balance = df.groupby('job')['balance'].mean()
sorted_job_balance = job_balance.sort_values(ascending=False)
# Now we can create a bar plot

sorted_job_balance.plot(kind='bar', figsize=(10,5))
plt.title('Average Balance by Job')
plt.xlabel('Job')
plt.ylabel('Average Balance')
plt.show()
```



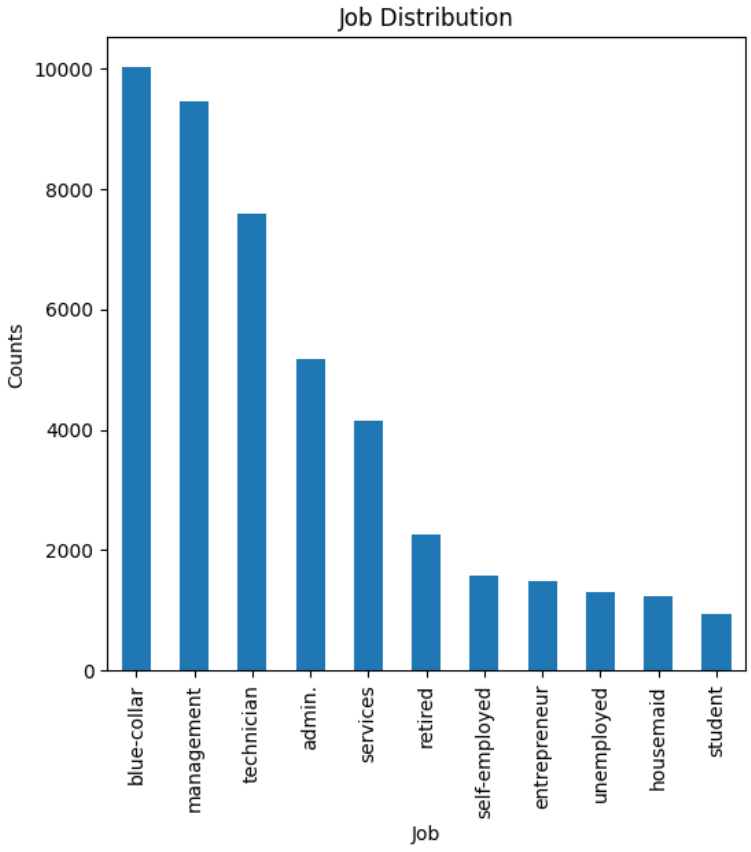
Draw a pie chart of the same data.

```
job_balance = df.groupby('job')['balance'].mean()
sorted_job_balance = job_balance.sort_values(ascending=False)

sorted_job_balance.plot(kind='pie', figsize=(7,6), autopct='%1.2f%%')
plt.title('Average Balance by Job')
plt.ylabel('') # We remove the y-label for pie charts as it's not necessary
plt.show()
```



Displaying the distribution of jobs.

```
# Bar Chart
df['job'].value_counts().plot(kind='bar',figsize=(6,6))
plt.title('Job Distribution')
plt.xlabel('Job')
plt.ylabel('Counts')
plt.show()
```



2 Data Preprocessing

Data cleaning, also known as data cleansing or data scrubbing, is a crucial step in the data analysis process. It involves identifying and correcting or removing errors, inaccuracies, and inconsistencies in datasets to improve their quality and reliability.

some key aspects of data cleaning:

1. **Handling Missing Values:** Data can have missing values due to various reasons such as errors in data collection or data entry. These missing values need to be handled carefully because they can lead to incorrect or biased analysis results. Techniques for handling missing data include imputation, where missing values are replaced with statistical estimates, and deletion, where rows or columns with missing data are removed.
2. **Removing Duplicates:** Duplicate data can occur due to various reasons such as data entry errors or merging of datasets. Duplicate data can distort analysis results and lead to incorrect conclusions. Therefore, it's important to identify and remove duplicate data.
3. **Data Transformation:** This involves converting data from one format or structure into another. For example, categorical data might be converted into numerical data for certain types of analysis.
4. **Outlier Detection:** Outliers are data points that significantly differ from other observations. They can be caused by variability in the data or experimental errors. Outliers can skew the analysis and lead to misleading results. Therefore, it's important to detect and handle outliers appropriately.
5. **Data Normalization:** This is the process of scaling numeric data from different variables down to a similar scale to prevent any single variable from dominating the others.
6. **Data Verification:** This ensures that the data is accurate, consistent and used appropriately. It involves processes like cross-checking and validation.

Now lets do that.
We have this dataset witch is about a bank customers.

Read Data

```
bankdf = pd.read_csv("Assignment-2_Data.csv")
bankdf.head(5)
```

	Id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	1001	NaN	management	married	tertiary	no	2143.0	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	1002	NaN	NaN	single	secondary	no	29.0	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	1003	NaN	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	1004	47.0	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	1005	33.0	unknown	single	unknown	no	1.0	no	no	unknown	5	may	198	1	-1	0	unknown	no

```
# 1. Handling Missing Values:
# Remove the contact column, Because that column has no value
bankdf = bankdf.drop('contact', axis=1)

# Remove the default column.
bankdf = bankdf.drop('default', axis=1)

# Remove the day column.
bankdf = bankdf.drop('day', axis=1)

# Remove the pdays column.
bankdf = bankdf.drop('pdays', axis=1)

# Remove the previous column.
bankdf = bankdf.drop('previous', axis=1)

# Remove the month column.
bankdf = bankdf.drop('month', axis=1)

# Remove the campaign column.
bankdf = bankdf.drop('campaign', axis=1)

# Remove the poutcome column.
bankdf = bankdf.drop('poutcome', axis=1)

# Replce Unknown wiith Nan
bankdf = bankdf.replace('unknown', np.nan)
bankdf = bankdf.replace('Unknown', np.nan)

# For simplicity, let's fill the missing values with appropriate values.
# For numerical columns, we'll use the mean of the column, and for categorical
columns, we'll use the mode.

num_cols = bankdf.select_dtypes(include=np.number).columns # getting all
numerical columns
cat_cols = bankdf.select_dtypes(exclude=np.number).columns # getting all
categorical columns
```

```
bankdf[num_cols] = bankdf[num_cols].fillna(bankdf[num_cols].mean())
bankdf[cat_cols] = bankdf[cat_cols].fillna(bankdf[cat_cols].mode().iloc[0])

bankdf.head(11)
```

	Id	age	job	marital	education	balance	housing	loan	duration	y
0	1001	40.933627	management	married	tertiary	2143.00000	yes	no	261	no
1	1002	40.933627	blue-collar	single	secondary	29.00000	yes	no	151	no
2	1003	40.933627	entrepreneur	married	secondary	2.00000	yes	yes	76	no
3	1004	47.000000	blue-collar	married	secondary	1506.00000	yes	no	92	no
4	1005	33.000000	blue-collar	single	secondary	1.00000	no	no	198	no
5	1006	35.000000	management	married	tertiary	231.00000	yes	no	139	no
6	1007	28.000000	management	single	tertiary	447.00000	yes	yes	217	no
7	1008	40.933627	entrepreneur	divorced	tertiary	1362.34662	yes	no	380	no
8	1009	58.000000	retired	married	primary	1362.34662	yes	no	50	no
9	1010	43.000000	technician	single	secondary	1362.34662	yes	no	55	no
10	1011	41.000000	admin.	divorced	secondary	270.00000	yes	no	222	no

1. **bankdf = bankdf.drop('contact', axis=1):** This line is dropping the 'contact' column from the DataFrame **bankdf**. The **drop** function is used to remove rows or columns. The **axis=1** parameter indicates that a column should be dropped.
2. **bankdf = bankdf.replace('unknown', np.nan):** This line is replacing all instances of the string 'unknown' in the DataFrame with **NaN** (Not a Number), which is a special marker used to denote missing values in pandas.
3. **bankdf = bankdf.replace('Unknown', np.nan):** This line is doing the same as the previous line, but for the string 'Unknown'. This is necessary because string comparisons in Python are case-sensitive.
4. **num_cols = bankdf.select_dtypes(include=np.number).columns:** This line is getting a list of all columns in **bankdf** that have a numeric data type. The **select_dtypes** function is used to select columns based on their data type.
5. **cat_cols = bankdf.select_dtypes(exclude=np.number).columns:** This line is getting a list of all columns in **bankdf** that have a non-numeric data type.
6. **bankdf[num_cols] = bankdf[num_cols].fillna(bankdf[num_cols].mean()):** This line is filling all missing values in numeric columns with the mean value of each column. The **fillna** function is used to fill missing values, and the **mean** function is used to calculate the mean value of each column.
7. **bankdf[cat_cols] = bankdf[cat_cols].fillna(bankdf[cat_cols].mode().iloc[0]):** This line is filling all missing values in non-numeric columns with the most frequent value (mode) of each column. The **mode** function is used to calculate the mode, and **iloc[0]** is used to select the first mode in case there are multiple modes.

8. **bankdf.head(5):** This line is displaying the first 5 rows of the cleaned DataFrame. The **head** function is used to select the first **n** rows.

This code is a basic example of data cleaning in pandas, where missing values are filled with appropriate values and unnecessary columns are dropped. The specific steps can vary depending on the dataset and the requirements of the analysis.

```
# 2. Removing Duplicates:
bankdf = bankdf.drop_duplicates()
```

Now we have to transform the string values to integer.
First we should to find out that what the unique values of each columns are.

```
unique_values_education = bankdf['education'].unique()
print(unique_values_education)

print("-----------------------------------")

unique_values_marital = bankdf['marital'].unique()
print("marital:",unique_values_marital)

print("-----------------------------------")

unique_values_job = bankdf['job'].unique()
print("job:",unique_values_job)

print("-----------------------------------")

unique_values_housing = bankdf['housing'].unique()
print("housing:",unique_values_housing)

print("-----------------------------------")

unique_values_loan = bankdf['loan'].unique()
print("loan:",unique_values_loan)
```

```
['tertiary' 'secondary' 'primary' 'primar']
-----------------------------------
marital: ['married' 'single' 'divorced']
-----------------------------------
job: ['management' 'blue-collar' 'entrepreneur' 'retired' 'technician' 'admin.'
 'services' 'self-employed' 'unemployed' 'housemaid' 'student']
-----------------------------------
housing: ['yes' 'no']
-----------------------------------
loan: ['no' 'yes']
```

Now, its time to transforming

```
# 3. Data Transformation:
# This highly depends on the specific requirements of your analysis.
# As an example, let's transform the 'education' column into numerical values.

edu_mapping = {'primary': 1, 'secondary': 2, 'tertiary': 3}
bankdf['education'] = bankdf['education'].map(edu_mapping)

mar_mapping = {'married': 0, 'single': 1, 'divorced': 2}
bankdf['marital'] = bankdf['marital'].map(mar_mapping)

job_mapping = {'management': 0, 'blue-collar':1, 'entrepreneur':2,
'retired':3, 'technician':4, 'admin.':5, 'services':6,
'self-employed':7, 'unemployed':8, 'housemaid':9, 'student':10}
bankdf['job'] = bankdf['job'].map(job_mapping)

hou_mapping = {'no': 0, 'yes': 1,}
bankdf['housing'] = bankdf['housing'].map(hou_mapping)

loa_mapping = {'no': 0, 'yes': 1,}
bankdf['loan'] = bankdf['loan'].map(loa_mapping)

bankdf.head(5)
```

	Id	age	job	marital	education	balance	housing	loan	duration	y
0	1001	41.186636	0	0	3.0	2143.0	1	0	261.0	no
1	1002	41.186636	1	1	2.0	29.0	1	0	151.0	no
2	1003	41.186636	2	0	2.0	2.0	1	1	76.0	no
3	1004	47.000000	1	0	2.0	1506.0	1	0	92.0	no
4	1005	33.000000	1	1	2.0	1.0	0	0	198.0	no

we also can transforming like this :

```
# Apply factorize to the 'education' column
bankdf['education'] = pd.factorize(bankdf['education'])[0]
```

```
# 4. Outlier Detection:
# This also depends on the specific requirements of your analysis.
# As an example, let's consider values in the 'balance' column that are 3
standard deviations away from the mean to be outliers.

upper_limit = bankdf['balance'].mean() + 3*bankdf['balance'].std()
lower_limit = bankdf['balance'].mean() - 3*bankdf['balance'].std()
bankdf = bankdf[(bankdf['balance'] < upper_limit) & (bankdf['balance'] >
lower_limit)]
```

upper_limit = bankdf['balance'].mean() + 3*bankdf['balance'].std():This line calculates the upper limit for what is considered a normal value in the 'balance' column. It does this by adding three times the standard deviation of 'balance' to the mean of 'balance'. Any value in 'balance' greater than this upper_limit is considered an outlier.
lower_limit = bankdf['balance'].mean() - 3*bankdf['balance'].std():This line calculates the lower limit for what is considered a normal value in the 'balance' column. It does this by subtracting three times the standard deviation of 'balance' from the mean of 'balance'. Any value in 'balance' less than this lower_limit is considered an outlier.
bankdf = bankdf[(bankdf['balance'] < upper_limit) & (bankdf['balance'] > lower_limit)]: This line filters the bankdf DataFrame to only include rows where the 'balance' is less than the upper_limit and greater than the lower_limit. In other words, it removes the outliers from bankdf.
So, in summary, this code identifies and removes outliers in the 'balance' column of the bankdf DataFrame. The outliers are defined as any values that are more than 3 standard deviations away from the mean.
we will examine this issue in the **statistical inference** section.

And now its time to normalization custom columns.
You can use the MinMaxScaler.

```
# 5. Data Normalization:

from sklearn.preprocessing import MinMaxScaler

# Assuming df is your DataFrame and 'column_to_normalize' is the column you
want to normalize
scaler = MinMaxScaler()
df['column_to_normalize'] = scaler.fit_transform(df[['column_to_normalize']])
```

Or you can use the StandardScaler

```
from sklearn.preprocessing import StandardScaler

# Assuming df is your DataFrame and 'column_to_normalize' is the column you
want to normalize
scaler = StandardScaler()
df['column_to_normalize'] = scaler.fit_transform(df[['column_to_normalize']])
```

3 Exploratory Data Analysis

Descriptive statistics refer to the set of methods used for organizing, summarizing, and describing information.

Before analyzing data, certain preliminary steps must be taken. When a researcher is faced with a volume of collected information for research, it is necessary to organize and summarize them in a meaningful and understandable way to reveal the hidden points of the data. Before going directly to statistical tests, they first explore the data. The subject of descriptive statistics is the arrangement and classification of data, graphical representation, and the calculation of values such as mode, mean, median, etc., which indicate the characteristics of each member of the society under discussion. So, descriptive statistical methods are used for this purpose. In general, three methods are used in descriptive statistics to summarize data: using tables, using graphs, and calculating specific values that represent important characteristics of the data.

1. Calculation of central indicators

In statistical calculations, it is necessary to determine the characteristics and overall position of the data. For this purpose, central indices are calculated. Central indices are in three types: Mode, Median, and Mean, each of which has its own specific application. In research where the measurement scale of data is at least interval, the mean is the best index. But in research where the measurement scale of data is ordinal or nominal, the median or mode is used.

2. Calculation of dispersion indices

Dispersion indices are contrary to central indices. They show the amount of dispersion or changes that exist among the data of a distribution (research results). The range of changes, quartile deviation, variance, and standard deviation are indices that are used for this purpose in research.

3. Descriptive statistics methods

- **The formation of a frequency distribution table**

  Frequency distribution is the organization of data or observations in classes along with the frequency of each class. To form a frequency distribution table, the range of changes, the number of classes, and the volume of classes must be calculated by the relevant

formulas, and then proceed to write the distribution table in two columns X (class column) and F (class frequency).

- **Drawing a chart**

  One of the weaknesses of displaying data in a frequency table is the lack of quick understanding of table information. Charts are a suitable tool for visual display of information. There are different types of charts, including histogram, column chart, cumulative polygon, pie chart, time series chart, and so on.

- **Calculating Correlation**

  In statistics, correlation or dependence is any statistical relationship, whether causal or not, between two random variables or bivariate data. Although in the broadest sense, "correlation" may indicate any type of association, in statistics it usually refers to the degree to which a pair of variables are linearly related. Familiar examples of dependent phenomena include the correlation between the height of parents and their offspring, and the correlation between the price of a good and the quantity the consumers are willing to purchase, as it is depicted in the so-called demand curve.

- **Analysis of covariance matrix data**

  Among the correlation analyses, there is the analysis of the covariance matrix or correlation matrix. Two of the most famous of these analyses are: the factor analysis model for discovering the underlying variables of a phenomenon in two exploratory and confirmatory categories, and the structural equation model for examining the causal relationships between variables.

Let's take a closer look

Mode: The mode is the value that appears most frequently in a data set. A set of data may have one mode, more than one mode, or no mode at all.

Median: The median is the middle value when a data set is ordered from least to greatest. If the data set has an odd number of observations, the number in the middle is the median. If the data set has an even number of observations, the median is the average of the two middle numbers.

Mean: The mean, often referred to as the average, is calculated by adding up all the numbers in the data set and then dividing by the count of numbers in the data set. The mean is sensitive to extremely large or small values in the data set, and it may not accurately represent the center of the values if the data set is skewed.

let's consider a simple data set: 1, 2, 2, 3, 4.

Mode: The mode is the value that appears most frequently in a data set. In our example, the mode is 2 because it appears twice, more than any other number. There's no specific formula for the mode, it's simply the most frequently occurring number.

Median: the median is the middle number, which is 2.

Mean: The mean, often referred to as the average, is calculated by adding up all the numbers in the data set and then dividing by the count of numbers in the data set. The formula for the mean is:

$$Mean = \frac{Sum\ of\ all\ observations}{Number\ of\ observations}$$

In our example, the mean would be calculated as follows:

$$Mean = \frac{1 + 2 + 2 + 3 + 4}{5} = 2.4$$

So, the mean of our data set is 2.4.

Quantiles

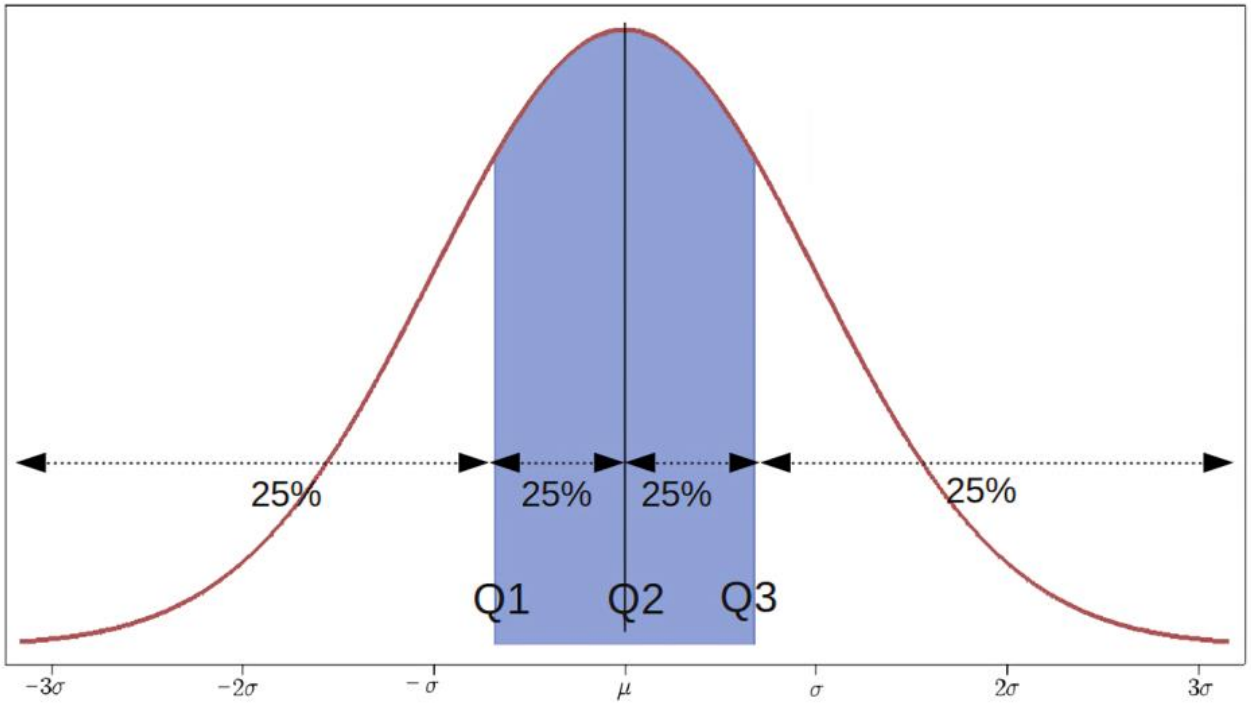


Image source: wikipedia.org

Quantiles are a fundamental concept in data analysis and statistics. They are points in a data distribution that relate to the rank order of values in that data. Quantiles divide a probability distribution into

regions of equal probability, or, in the case of empirical distributions, divide the data set into equal-sized consecutive subsets.

There are several types of quantiles:

- **Quartiles**: These divide the data into four equal parts. The first quartile (Q1) is the 25th percentile of the data, the second quartile (Q2) is the 50th percentile (also known as the median), and the third quartile (Q3) is the 75th percentile.
- **Deciles**: These divide the data into ten equal parts.
- **Percentiles**: These divide the data into 100 equal parts.

Quantiles are useful in statistics and data analysis for understanding the distribution of data points. They can help identify outliers, understand the spread and skewness of the data, and assist in the creation of box plots, among other things.

For example, if you have a dataset of exam scores, the 90th percentile would represent the score below which 90% of the scores fall. This could be used to understand the top 10% of scores in this dataset.

In mathematical notation, the quantile can be expressed as follows:

Quartile Deviation (QD): Also known as the semi-interquartile range, it is a measure of statistical dispersion. It is half the difference between the upper quartile (Q3) and the lower quartile (Q1). The formula for Quartile Deviation is:

$$QD = \frac{Q3 - Q1}{2}$$

For example, consider the data set: 1, 2, 5, 6, 7, 9, 12. The lower quartile (Q1) is 2, and the upper quartile (Q3) is 9. So, the Quartile Deviation would be (9-2)/2 = 3.5.

Variance

Variance is a statistical measurement that describes the spread of numbers in a data set. It measures how far each number in the set is from the mean (average) and thus from every other number in the set. Variance is often denoted by the symbol

$$\sigma^2$$

- **Statistical Inference**: Variance is a key parameter in many statistical procedures, such as hypothesis tests and confidence intervals. Knowing the variance allows you to judge the reliability of any conclusions drawn from the data.

Remember, while variance gives you a measure of dispersion, it does not tell you about the direction of the dispersion because it squares the deviations around the mean. This is where standard deviation comes into play, which is the square root of variance and provides information on the average distance of data points from the mean.

The formula for Variance is:

$$Variance = \frac{\sum(X_i - Mean)^2}{n}$$

For example, consider the data set: 1, 2, 3, 4, 5. The mean is 3. The squared differences from the mean are: 4, 1, 0, 1, 4. The average of these squared differences is 2, so the variance is 2.

Standard Deviation (STD)

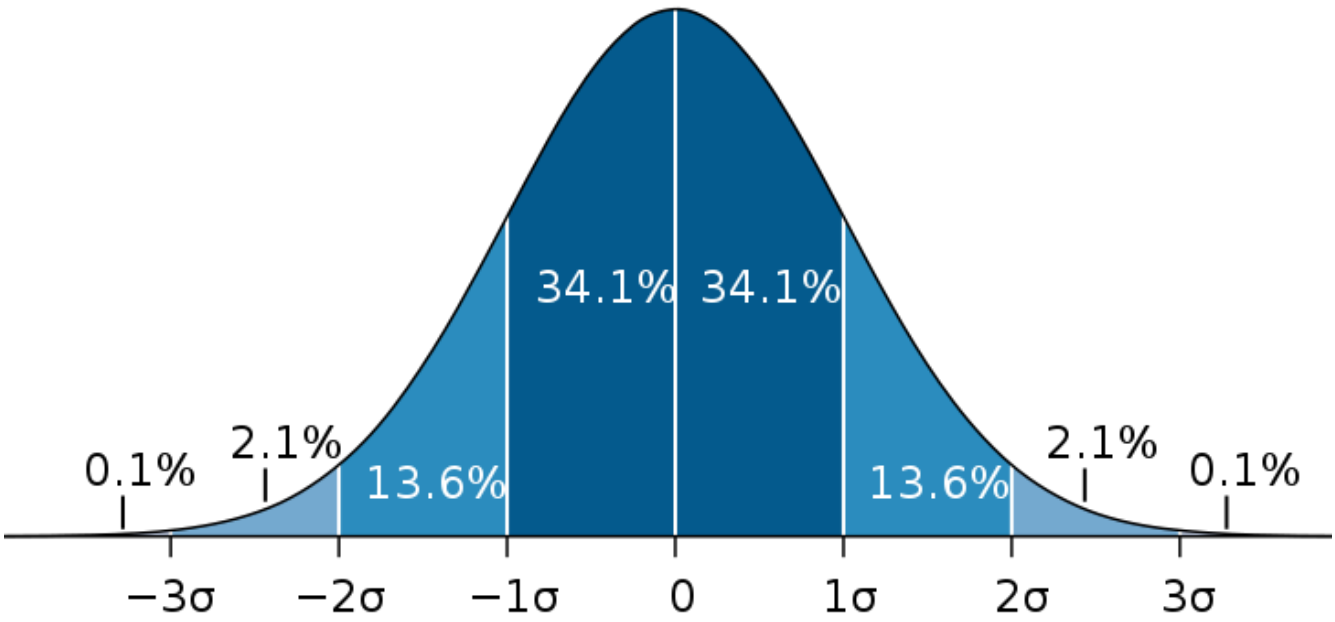


Image source: wikipedia.org

The Standard Deviation is a measure of how spread out numbers are from the mean. It is the square root of the Variance. The formula for Standard Deviation is:

$$STD = \sqrt{Variance}$$

Continuing with the previous example, the standard deviation would be the square root of the variance, so SD = √2 ≈ 1.41.

When the standard deviation of the age column in a dataset is 10, it means that most of the data (about 68% if it's a normal distribution) falls within a range of the mean minus 10 years and the mean plus 10 years. In other words, if the average age is 30, most of the data (about 68%) will be between 20 and 40 years old.

In summary, while both variance and standard deviation provide information about the dispersion of data, standard deviation is often preferred because it is in the same units as the data and thus easier to interpret. However, variance is used in many areas of statistics and probability theory because of its mathematical properties.

Let's write the codes

I have a the dataset that I cleaned that.

```
bankdf = pd.read_csv("Assignment-2_Data.csv")
bankdf.head(11)
```

	Id	age	job	marital	education	balance	housing	loan	duration	y
0	1001	40.933627	management	married	tertiary	2143.00000	yes	no	261	no
1	1002	40.933627	blue-collar	single	secondary	29.00000	yes	no	151	no
2	1003	40.933627	entrepreneur	married	secondary	2.00000	yes	yes	76	no
3	1004	47.000000	blue-collar	married	secondary	1506.00000	yes	no	92	no
4	1005	33.000000	blue-collar	single	secondary	1.00000	no	no	198	no
5	1006	35.000000	management	married	tertiary	231.00000	yes	no	139	no
6	1007	28.000000	management	single	tertiary	447.00000	yes	yes	217	no
7	1008	40.933627	entrepreneur	divorced	tertiary	1362.34662	yes	no	380	no
8	1009	58.000000	retired	married	primary	1362.34662	yes	no	50	no
9	1010	43.000000	technician	single	secondary	1362.34662	yes	no	55	no
10	1011	41.000000	admin.	divorced	secondary	270.00000	yes	no	222	no

First we should to find the mode, median, mean and the standard deviation balance column. So let's do that.

```
std_dev = bankdf['balance'].std()


mode = bankdf['balance'].mode()[0]
median = bankdf['balance'].median()
mean = bankdf['balance'].mean()

print(f"Mode: {mode}, Median: {median}, Mean: {mean}, standard Deviation:
{std_dev}")
```

```
Mode: 0.0, Median: 378.0, Mean: 946.7719265647424, standard Deviation: 1549.4566808315094
```

Now its time to find out the correlation between balance and all columns.

```
df = bankdf

# Calculate correlations
correlations = {
    'duration': df['balance'].corr(df['duration']),
    'age': df['balance'].corr(df['age']),
    'job': df['balance'].corr(df['job']),
    'education': df['balance'].corr(df['education']),
    'marital': df['balance'].corr(df['marital']),
    'loan': df['balance'].corr(df['loan']),
    'housing': df['balance'].corr(df['housing'])
}

# Create a DataFrame from the correlations
corr_df = pd.DataFrame(list(correlations.items()), columns=['Column', 'Corre-
lation'])

# Sort the DataFrame by correlation in descending order
sorted_corr_df = corr_df.sort_values('Correlation', ascending=False)

print(sorted_corr_df)
```

```
      Column  Correlation
1        age     0.103584
0   duration     0.024308
3  education     0.023607
2        job     0.001273
4    marital    -0.044893
5       loan    -0.074487
6    housing    -0.088893
```

As you can see, the highest correlation is between age and balance. And this means that in this dataset, age has a direct relationship with balance.

And now, we are going to use another way to find the correlation between the balance and other columns.

```
from sklearn.ensemble import RandomForestRegressor

X = bankdf.drop(['balance', 'y', 'Id'], axis=1)  # Features (all columns except 'balance', 'y', and 'Id')
y = bankdf['balance']  # Target (the 'balance' column)

# Create a random forest regressor
model = RandomForestRegressor(random_state=1)

# Train the model
model.fit(X, y)

# Get feature importances
importances = model.feature_importances_

# Create a DataFrame for visualization
feature_importances = pd.DataFrame({'feature': X.columns, 'importance': importances})

# Sort the DataFrame by importance
feature_importances = feature_importances.sort_values('importance', ascending=False)

print(feature_importances)
```

```
     feature  importance
6   duration    0.568175
0        age    0.210968
1        job    0.100061
2    marital    0.054991
3  education    0.032694
4    housing    0.025196
5       loan    0.007916
```

This script is implementing a **Random Forest Regressor** model using the sklearn.ensemble library in Python. Here's a step-by-step explanation:

1. **Importing Necessary Library:** The script starts by importing the RandomForestRegressor class from the sklearn.ensemble module. This class implements a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
2. **Defining Features and Target:** The script then defines the features X and the target y for the model. The features X are all columns in the bankdf DataFrame except 'balance', 'y', and 'Id'. The target y is the 'balance' column in the bankdf DataFrame.

3. **Creating the Model:** A RandomForestRegressor model is created with a specified random_state of 1 to ensure the results are reproducible.
4. **Training the Model:** The model is trained using the fit() method, which fits the random forest regressor to the data.
5. **Getting Feature Importances:** The feature_importances_ attribute of the trained model is accessed to get the importance of each feature. This gives an array where each element corresponds to the importance of a feature.
6. **Creating a DataFrame for Visualization:** A DataFrame is created to visualize the feature importances. This DataFrame has two columns: 'feature' (the name of the feature) and 'importance' (the importance of the feature).
7. **Sorting the DataFrame:** The DataFrame is sorted by the 'importance' column in descending order. This means the features with the highest importance are at the top of the DataFrame.

This script is a common way to train a Random Forest Regressor model and visualize the importance of each feature in the model. It can be useful for understanding which features are most influential in predicting the target variable.

Frequency distribution table

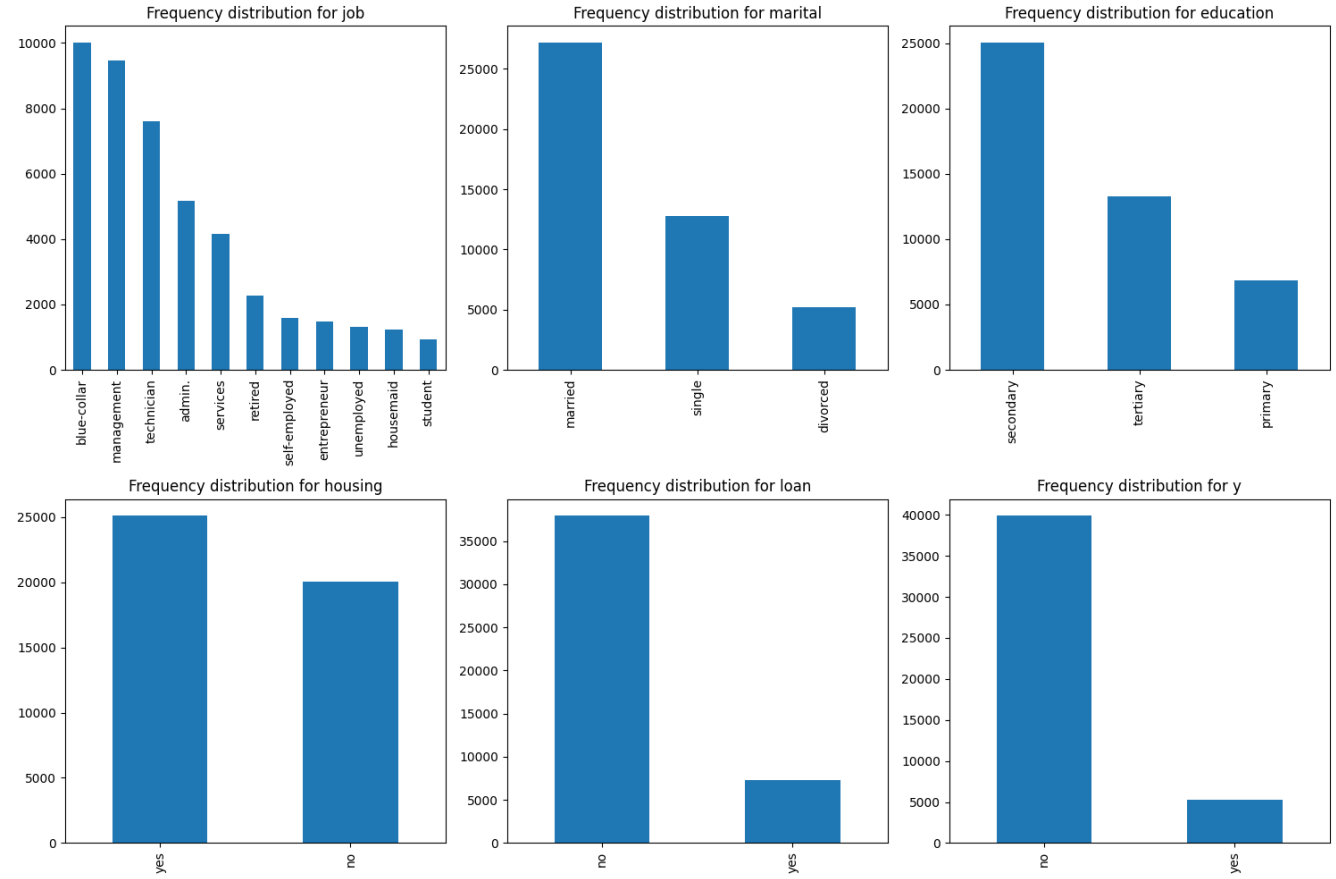
```
# For each categorical column, we can generate a frequency distribution table
categorical_columns = ['job', 'marital', 'education', 'housing', 'loan', 'y']

for col in categorical_columns:
    print(f"Frequency distribution for {col}:")
    print(bankdf[col].value_counts())
    print("\n")
```

```
Frequency distribution for job:
blue-collar      10021
management        9458
technician        7596
admin.            5171
services          4154
retired           2264
self-employed     1579
entrepreneur      1487
unemployed        1303
housemaid         1240
student            938
Name: job, dtype: int64


Frequency distribution for marital:
married     27214
single      12790
divorced     5207
Name: marital, dtype: int64


Frequency distribution for education:
secondary    25059
tertiary     13301
primary       6851
Name: education, dtype: int64
```

```
Frequency distribution for housing:
yes    25130
no     20081
Name: housing, dtype: int64


Frequency distribution for loan:
no     37967
yes     7244
Name: loan, dtype: int64


Frequency distribution for y:
no     39922
yes     5289
Name: y, dtype: int64
```

Frequency distribution Bar plot

```
# For each categorical column, we can generate a frequency distribution table
categorical_columns = ['job', 'marital', 'education', 'housing', 'loan', 'y']

# Define the number of rows and columns for the subplot grid
n = len(categorical_columns)
ncols = 3
nrows = math.ceil(n / ncols)

# Define the size of the overall figure
fig, axs = plt.subplots(nrows, ncols, figsize=(15,5*nrows))

# Flatten the axes array, if there's more than one row
if nrows > 1:
    axs = axs.flatten()

# For each categorical column, generate a bar plot
for i, col in enumerate(categorical_columns):
    bankdf[col].value_counts().plot(kind='bar', ax=axs[i])
    axs[i].set_title(f"Frequency distribution for {col}")

# Remove any unused subplots
if n < nrows*ncols:
    for i in range(n, nrows*ncols):
        fig.delaxes(axs[i])

# Show the plot
plt.tight_layout()
plt.show()
```



Correlation Analysis

Correlation analysis is a statistical method used to evaluate the strength and direction of the relationship between two or more variables. The correlation coefficient, which ranges from -1 to 1, is used to quantify this relationship.

- A correlation coefficient of **1** indicates a perfect positive correlation, meaning that as one variable increases, the other variable also increases.
- A correlation coefficient of **-1** indicates a perfect negative correlation, meaning that as one variable increases, the other variable decreases.
- A correlation coefficient of **0** means that there's no linear relationship between the two variables.

The process of conducting a correlation analysis involves several steps:

1. **Define the Problem:** Identify the variables that you think might be related. The variables must be measurable on an interval or ratio scale.
2. **Data Collection:** Collect data on the variables of interest. The data could be collected through various means such as surveys, observations, or experiments.
3. **Data Inspection:** Check the data for any errors or anomalies such as outliers or missing values.
4. **Choose the Appropriate Correlation Method:** Select the correlation method that's most appropriate for your data.
5. **Compute the Correlation Coefficient:** Once you've selected the appropriate method, compute the correlation coefficient.
6. **Interpret the Results:** Interpret the correlation coefficient you obtained.
7. **Check the Significance:** It's also important to test the statistical significance of the correlation.

There are different types of correlation coefficients, such as Pearson's r, Spearman's rho, and Kendall's tau, each with its own assumptions and use cases.

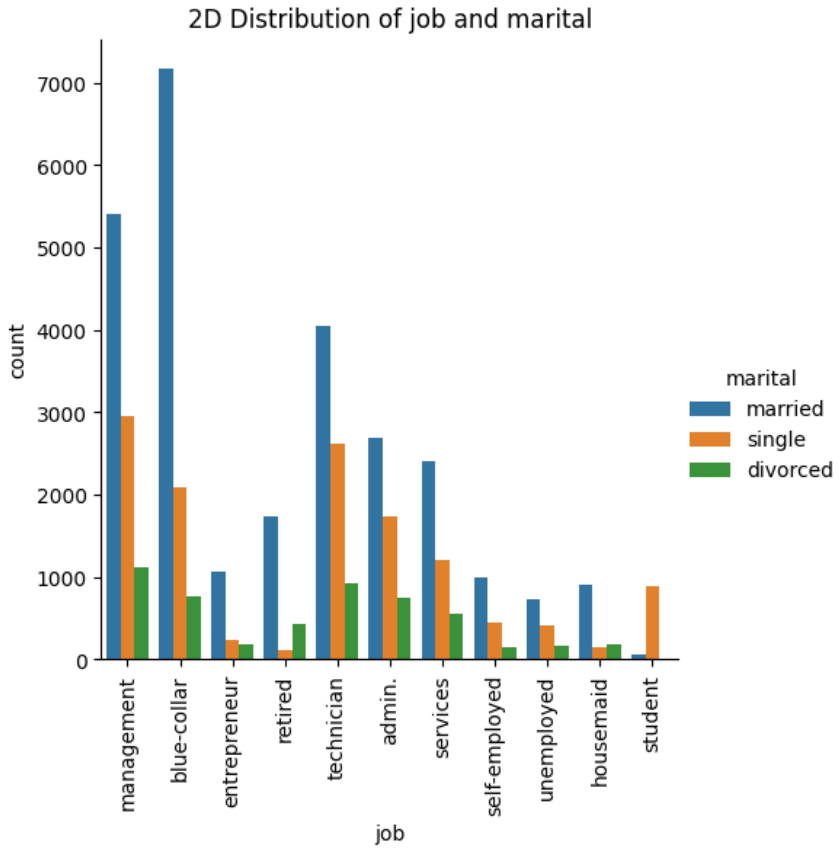
In the previous pages, we examined how to find the correlation between two variables. What I want to tell you is that if your data is continuous, you can display your desired columns in the dataset as a scatter plot or a line chart to better understand this topic.

When your data is continuous, visualizing it using scatter plots or line charts can be very helpful. A scatter plot displays the relationship between two numerical variables, one along the x-axis and the other along the y-axis. Each point on the plot corresponds to a single observation in the dataset. If there's a correlation between the variables, it will be evident in the pattern of the plotted points.

4 2-d distributions

A 2-D distribution plot visualizes the relationship between two variables. It shows where values are concentrated, revealing patterns, trends, and outliers. Examples include scatter plots, hexbin plots, and contour plots.

```
import seaborn as sns
col_1 = 'job'
col_2 = 'marital'
plt.figure(figsize=(10,4))
cat_plot = sns.catplot(x=col_1, hue=col_2, kind="count", data=bankdf)
cat_plot.set_xticklabels(rotation=90)  # Rotate labels
plt.title(f'2D Distribution of {col_1} and {col_2}')
plt.show()
```



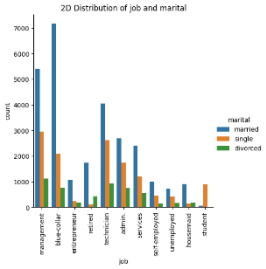
This code generates a 2D categorical plot using seaborn, a Python data visualization library.

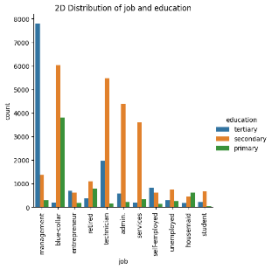
1. **`col_1 = 'job' and col_2 = 'marital':`** *These lines set the two categorical columns to be plotted.*
2. **`plt.figure(figsize=(10,4)):`** *This line sets the size of the figure to be plotted.*
3. **`sns.catplot(x=col_1, hue=col_2, kind="count", data=bankdf):`** *This line creates a categorical plot (catplot) with 'job' on the x-axis and 'marital' as the hue (color). The kind="count" argument means it's a count plot, which shows the counts of observations in each categorical bin.*
4. **`cat_plot.set_xticklabels(rotation=90):`** *This line rotates the x-axis labels by 90 degrees for better readability.*
5. **`plt.title(f'2D Distribution of {col_1} and {col_2}'):`** *This line sets the title of the plot.*
6. **`plt.show():`** *This line displays the plot.*

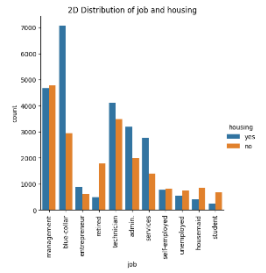
In summary, this code visualizes the distribution of 'job' and 'marital' categories in the bankdf DataFrame.

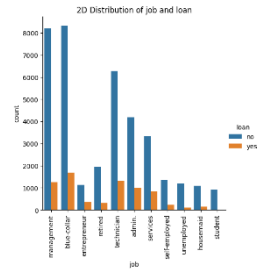
The difference between following code and the previous one is that this code generates a 2D categorical plot for **each pair** of categorical columns in our DataFrame, while the previous code generates a 2D categorical plot for **only one pair** of categorical columns ('job' and 'marital').

```
import seaborn as sns
categorical_columns = ['job', 'marital', 'education', 'housing', 'loan', 'y']

for i in range(len(categorical_columns)):
    for j in range(i+1, len(categorical_columns)):
        plt.figure(figsize=(10,4))
        cat_plot = sns.catplot(x=categorical_columns[i], hue=categorical_columns[j], kind="count", data = bankdf)
        cat_plot.set_xticklabels(rotation=90)  # Rotate labels
        plt.title(f'2D Distribution of {categorical_columns[i]} and {categorical_columns[j]}')
        plt.show()
```



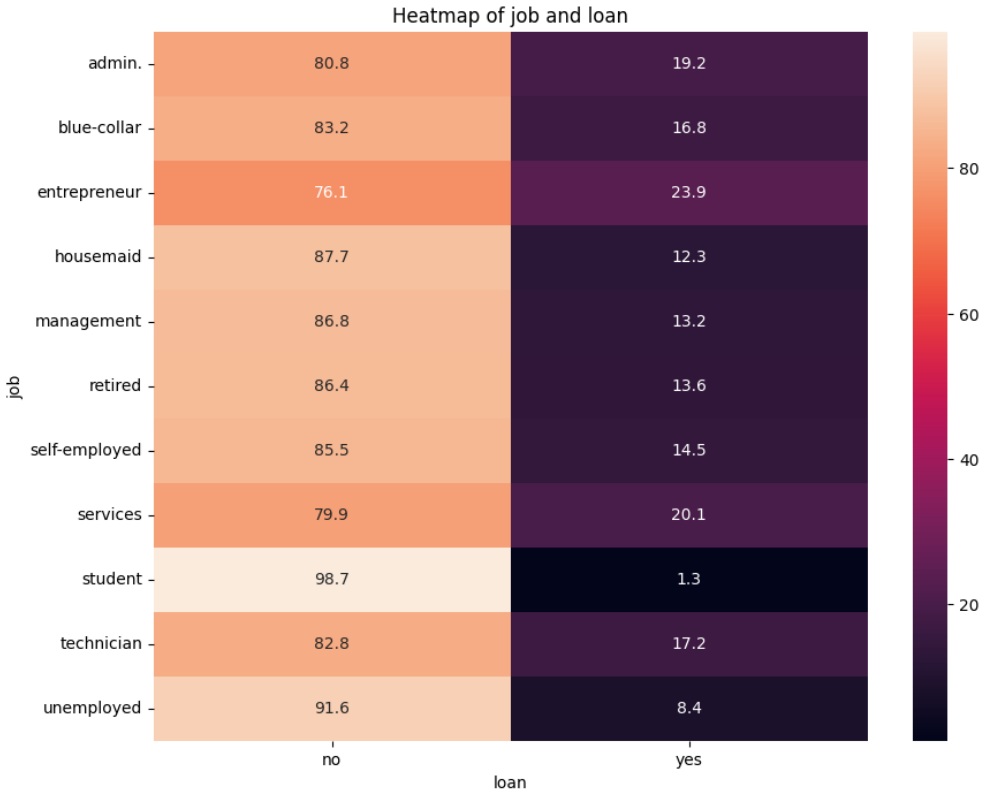






2-d distributions (heatmap)

```
import seaborn as sns
col_1 = 'job'
col_2 = 'loan'
grouped = bankdf.groupby([col_1, col_2]).size()
percentage = grouped.groupby(level=0).apply(lambda x: 100 * x /
float(x.sum()))

percentage_df = pd.DataFrame(percentage).reset_index()
percentage_df.columns = [col_1, col_2, 'percentage']

heatmap_data = percentage_df.pivot(col_1, col_2, 'percentage')
plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data, annot=True, fmt=".1f")
plt.title(f'Heatmap of {col_1} and {col_2}')
plt.show()
```



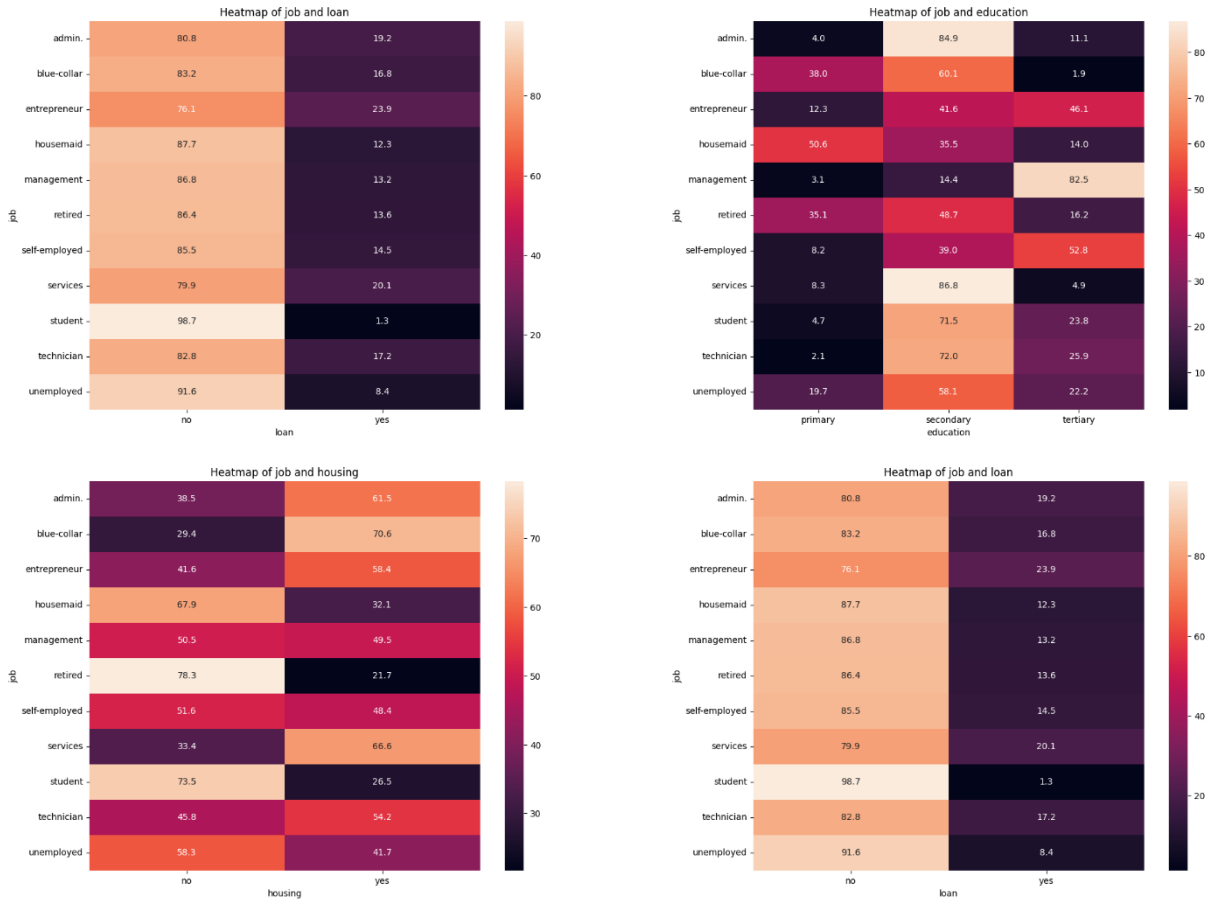
This code generates a heatmap to visualize the relationship between two categorical variables in our DataFrame, ‘job’ and ‘loan’. Here’s a breakdown:

1. **`grouped = bankdf.groupby([col_1, col_2]).size`***(): This line groups the DataFrame by 'job' and 'loan', then calculates the size of each group.*
2. **`percentage = grouped.groupby(level=0).apply(lambda x: 100 * x / float(x.sum())):`** *This line calculates the percentage of each group relative to the total for each 'job'.*
3. **`percentage_df = pd.DataFrame(percentage).reset_index`***(): This line converts the series to a DataFrame and resets the index.*
4. **`percentage_df.columns = [col_1, col_2, 'percentage']:`** *This line names the columns of the DataFrame.*
5. **`heatmap_data = percentage_df.pivot(col_1, col_2, 'percentage'`***): This line reshapes the DataFrame suitable for a heatmap.*
6. **`plt.figure(figsize=(10, 8)):`** *This line sets the size of the figure to be plotted.*
7. **`sns.heatmap(heatmap_data, annot=True, fmt=".1f"):`** *This line creates a heatmap where the x-axis represents 'job', the y-axis represents 'loan', and the color represents the percentage of each combination.*
8. **`plt.title(f'Heatmap of {col_1} and {col_2}'):`** *This line sets the title of the plot.*
9. *plt.show(): This line displays the plot.*

In summary, this code visualizes the distribution of 'job' and 'loan' categories in the bankdf DataFrame as a heatmap.

This code generates a heatmap for **each pair** of categorical columns in our DataFrame, similar to the previous code. The difference is that this code iterates **over all pairs** of categorical columns in our DataFrame, while the previous code only generates a heatmap for **one specific pair** of columns **('job' and 'loan')**.

```
import seaborn as sns
categorical_columns = ['job', 'marital', 'education', 'housing', 'loan', 'y']

for i in range(len(categorical_columns)):
    for j in range(i+1, len(categorical_columns)):
        grouped = bankdf.groupby([categorical_columns[i], categorical_columns[j]]).size()
        percentage = grouped.groupby(level=0).apply(lambda x: 100 * x / float(x.sum()))

        percentage_df = pd.DataFrame(percentage).reset_index()
        percentage_df.columns = [categorical_columns[i], categorical_columns[j], 'percentage']

        heatmap_data = percentage_df.pivot(categorical_columns[i], categorical_columns[j], 'percentage')

        plt.figure(figsize=(10, 8))
        sns.heatmap(heatmap_data, annot=True, fmt=".1f")
        plt.title(f'Heatmap of {categorical_columns[i]} and {categorical_columns[j]}')
        plt.show()
```



correlation

To do this, you need to transform the values of the dataset to numbers. Go to the chapter 2 data transformations part.

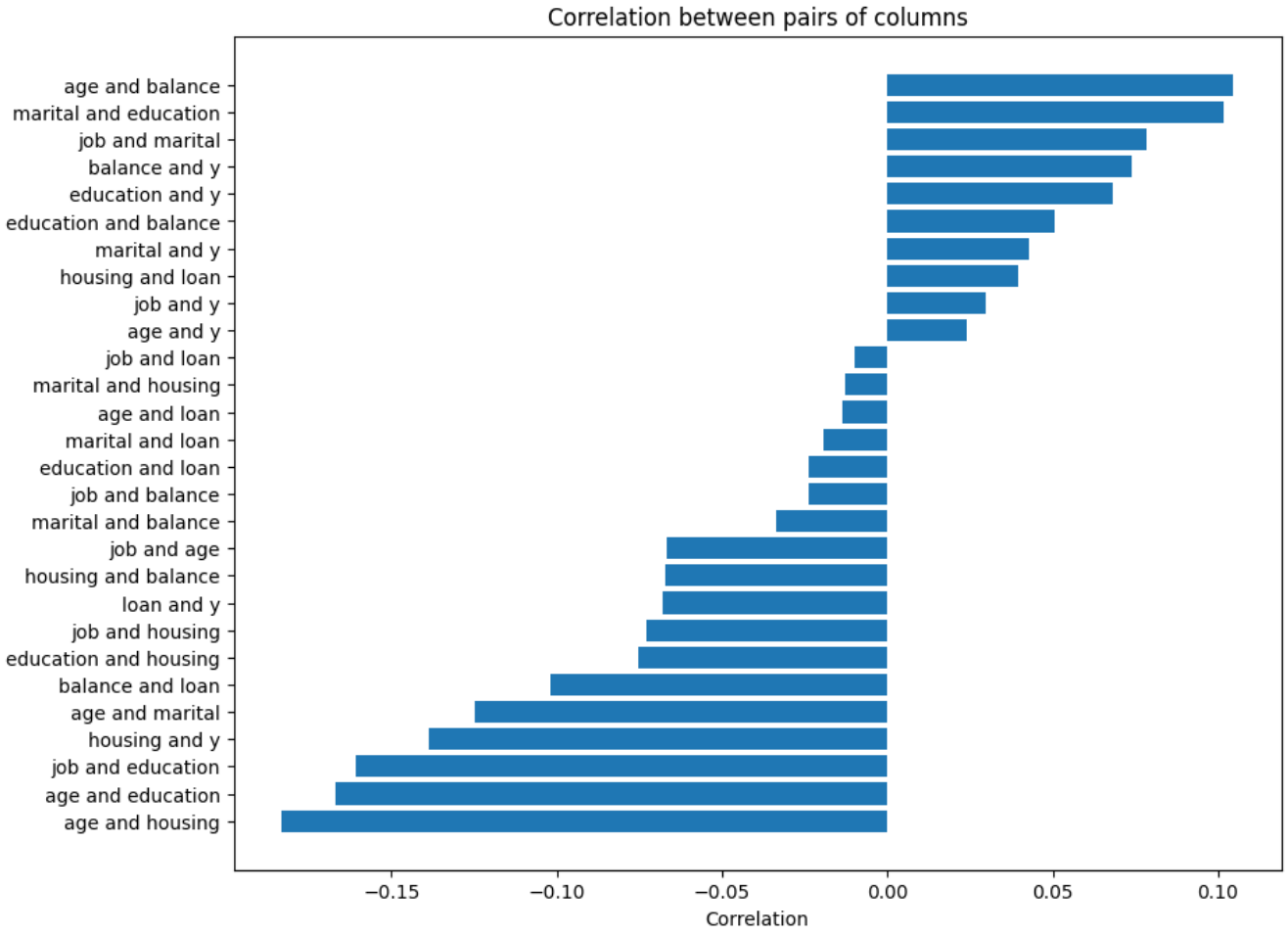
Then you can use the following code:

```
categorical_columns = ['job', 'age','marital', 'education', 'housing','balance', 'loan', 'y']
correlations = []

for i in range(len(categorical_columns)):
    for j in range(i+1, len(categorical_columns)):
        correlation = bankdf[categorical_columns[i]].corr(bankdf[categorical_columns[j]])
        correlations.append((f"{categorical_columns[i]} and {categorical_columns[j]}", correlation))

# Sort the correlations in descending order
correlations.sort(key=lambda x: x[1], reverse=True)

# Print the sorted correlations
for pair, corr in correlations:
    print(f"Correlation between {pair}: {corr:.2f}")
```

```
Correlation between age and balance: 0.10
Correlation between marital and education: 0.10
Correlation between job and marital: 0.08
Correlation between balance and y: 0.07
Correlation between education and y: 0.07
Correlation between education and balance: 0.05
Correlation between marital and y: 0.04
Correlation between housing and loan: 0.04
Correlation between job and y: 0.03
Correlation between age and y: 0.02
Correlation between job and loan: -0.01
Correlation between marital and housing: -0.01
Correlation between age and loan: -0.01
Correlation between marital and loan: -0.02
Correlation between education and loan: -0.02
Correlation between job and balance: -0.02
Correlation between marital and balance: -0.03
Correlation between job and age: -0.07
Correlation between housing and balance: -0.07
Correlation between loan and y: -0.07
Correlation between job and housing: -0.07
Correlation between education and housing: -0.08
Correlation between balance and loan: -0.10
Correlation between age and marital: -0.12
Correlation between housing and y: -0.14
Correlation between job and education: -0.16
Correlation between age and education: -0.17
Correlation between age and housing: -0.18
```

And you can plot it.

```
# Extract the correlation values and pair names
correlation_values = [corr for pair, corr in correlations]
pair_names = [pair for pair, corr in correlations]

# Create the bar plot
plt.figure(figsize=(10, 8))
plt.barh(pair_names, correlation_values)
plt.xlabel('Correlation')
plt.title('Correlation between pairs of columns')
plt.gca().invert_yaxis()  # Invert the y-axis to have the highest correlation at
the top
plt.show()
```



Note that during data transformation, you must set the values correctly, for example, in binary columns, consider positive values as 1 and negative values as 0 and -1 For columns with three value, For example, in the marital column, I set -1 for divorced, 1 for married, and 0 for single, although I could set them as numbers 0 to 2. and in columns with many classes, set values based on a certain principle not random numbers. For example, I set jobs based on their balance from 0 to 11. So it is better to draw and observe the frequency distribution bar before data transformation.

5 Statistical Inference

Probability Distributions

Probability distributions are mathematical functions that describe the likelihood of different outcomes or events in a random experiment or process. They are fundamental in statistics and are used both on a theoretical and practical level.
A probability distribution describes the likelihood of obtaining all possible values that a random variable can take. The values of the variable vary based on the underlying probability distribution. They are often depicted using graphs or probability tables.
Probability distributions can be divided into two types:

1. **Discrete probability distributions** for discrete variables
2. **Probability density functions** for continuous variables

The mean of a probability distribution is referred to as its expected value. Probability distributions describe the dispersion of the values of a random variable.
For example, if you draw a random sample and measure the heights of the subjects, you create a distribution of heights. This type of distribution is useful when you need to know which outcomes are most likely, the spread of potential values, and the likelihood of different results.
In data analysis, probability distributions provide important insights into the behavior of random variables and are used to calculate confidence intervals for parameters and to calculate critical regions for hypothesis tests.

Discrete probability distributions

1. **Bernoulli and uniform distribution**
   Bernoulli distribution: If an experiment has two outcomes (success and failure), and their probabilities are P and 1-P, respectively. Then the random variable x as the number of wins is equal to 1 or 0, then there is a Bernoulli distribution. and its probability function is as follows:

$$f(x) = p^x\ (1-p)^{1-x} \qquad ,x = 0,1$$

2. **Binomial and hypergeometric distribution**
   If the Bernoulli experiment is repeated n times so that the number of trials is a fixed number n, and the parameter p is the same probability of success for all trials, and the trials are all independent, then the random variable x is equal to the number of wins in n trials, Random variable is called binomial and its probability function is as follows:

$$f(x) = \binom{n}{x}\ p^x\ (1-p)^x \quad ,x = 0,1,2,\dots,n$$

Example: find the probability of recovery of 7 out of 10 patients if the assumption of independence is maintained and the probability of recovery of each of them is equal to 0.8:

Each patient recovers (win) or does not (fail): probability of win = P = 0.8. We have 10 people, so n = 10. And 7 people recover (7 victories) So x = 7

$$f(x) = \binom{10}{7} (0.8)^7 (1-0.8)^{10-7} = \frac{10!}{7!\,3!} \times (0.8)^7 \times (0.2)^3$$

3. Poisson distribution

It is a discrete probability distribution that describes the probability that an event will occur a certain number of times in a fixed time interval or place; Provided that these events occur at a certain average rate and independent of the time of the last event.

Suppose that in a binomial experiment, the probability of success depends on the number of experiments, i.e. distribution in the form B(n,pn) be Also consider the case that as n increases, the value pn shrink Then we can say that the number of successes is from the Poisson distribution with parameter npn=λ. And If we call the random variable related to the number of successes X, we have

$$p(X = x) = \frac{(e^{-\lambda} \times \lambda^x)}{x!}$$

Note: If the probability of success is constant, the number of successes has a binomial distribution.

4. Geometric and negative binomial distribution

Suppose we have a sequence of random Bernoulli trials with probability p of success. In this case, the result of each experiment with probability p is equal to 1 and with probability 1−p is also equal to 0. We know that the value 1 indicates success and 0 indicates failure. This test continues until r failures occur.

If X is a random variable with support S={0,1,2,...}, we consider its probability distribution to be negative binomial if its probability function is as follows.

$$f(k; r, p) \equiv \Pr(X = k) = \binom{r+k-1}{k} p^k (1-p)^r$$

It is clear that here r is the number of failures and k is the number of successes. Also, p is the probability of success each time the Bernoulli test is performed.

Hypothesis Testing

Hypothesis testing is a formal procedure used in statistics to test specific predictions or hypotheses that arise from theories. It involves the following main steps:

1. **State your null and alternate hypotheses (H0 and Ha):** After developing your initial research hypothesis, it is important to restate it as a null (H0) and alternate (Ha) hypothesis so that you can test it mathematically. The null hypothesis assumes that there is no difference between certain characteristics of data. The alternate hypothesis shows that the observations of an experiment are due to some real effect.
2. **Collect data:** For a statistical test to be valid, it is important to perform sampling and collect data in a way that is designed to test your hypothesis. If your data are not representative, then you cannot make statistical inferences about the population you are interested in.
3. **Perform a statistical test:** There are a variety of statistical tests available, but they are all based on the comparison of within-group variance (how spread out the data is within a category) versus between-group variance (how different the categories are from one another).
4. **Decide whether to reject or fail to reject your null hypothesis:** The hypothesis testing results in either rejecting or not rejecting the null hypothesis.
5. **Present your findings:** The findings are presented in your results and discussion section.

Hypothesis testing is a tool for making statistical inferences about the population data. It tests an assumption and determines how likely something is within a given standard of accuracy. It provides a way to verify whether the results of an experiment are valid.

```
from scipy import stats

# Let's say we want to test if the average balance is different for clients with a housing
loan and those without.
# This is a two-sample t-test.

# Formulate the hypotheses
# H0: The average balance is the same for clients with a house and those without.
# Ha: The average balance is different for clients with a house and those without.

# Choose a significance level
alpha = 0.05

# Separate the data into two groups
group1 = bankdf[bankdf['housing'] == 'yes']['balance']
group2 = bankdf[bankdf['housing'] == 'no']['balance']

# Perform the t-test
t_stat, p_value = stats.ttest_ind(group1, group2)

# Make a decision
if p_value < alpha:
    print("We reject the null hypothesis and conclude that the average balance is different
for clients with a house and those without.")
else:
    print("We do not reject the null hypothesis and conclude that the average balance is the
same for clients with a house and those without.")
```

This code is performing a two-sample t-test to compare the average balance of clients with a housing and those without.

1. **Hypotheses:** The null hypothesis (H0) is that the average balance is the same for clients with a house and those without. The alternative hypothesis (Ha) is that the average balance is different for these two groups.

2. **Significance Level:** A significance level of 0.05 is chosen. This is the probability of rejecting the null hypothesis when it is true.

3. **Data Separation:** The data is separated into two groups - group1 contains the balances of clients with a house , and group2 contains the balances of clients without a house.

4. **T-Test:** A two-sample t-test is performed on these two groups using the ttest_ind function from the scipy.stats module. This function returns the t-statistic and the p-value.

5. **Decision Making:** If the p-value is less than the significance level (0.05), the null hypothesis is rejected, and it is concluded that the average balance is different for clients with a house and those without. If the p-value is not less than the significance level, the null hypothesis is not rejected, and it is concluded that the average balance is the same for clients with a house and those without.

```
from scipy.stats import chi2_contingency


# Formulate the hypotheses
# H0: There is no significant association between age and marital status.
# Ha: There is a significant association between age and marital status.

# Create a contingency table
contingency_table = pd.crosstab(bankdf['age'], bankdf['marital'])

# Perform the Chi-Square Test of Independence
chi2, p_val, dof, expected = chi2_contingency(contingency_table)

# Print the results
print(f"Chi-Square Statistic: {chi2}")
print(f"P-value: {p_val}")
print(f"Degrees of Freedom: {dof}")

# Make a decision based on the p-value
alpha = 0.05  # significance level
if p_val < alpha:
    print("We reject the null hypothesis and conclude that there is a significant asso-
ciation between age and marital status.")
else:
    print("We do not reject the null hypothesis and conclude that there is no signifi-
cant association between age and marital status.")
```

This code is performing a Chi-Square Test of Independence to examine the relationship between age and marital status.

1. **Hypotheses:** The null hypothesis (H0) is that there is no significant association between age and marital status. The alternative hypothesis (Ha) is that there is a significant association between these two variables.

2. **Contingency Table:** A contingency table is created using the crosstab function from pandas. This table shows the frequency distribution of the variables 'age' and 'marital'.

3. **Chi-Square Test:** A Chi-Square Test of Independence is performed on the contingency table using the chi2_contingency function from the scipy.stats module. This function returns the Chi-Square statistic, the p-value, the degrees of freedom, and the expected frequencies.

4. **Decision Making:** If the p-value is less than the significance level (0.05), the null hypothesis is rejected, and it is concluded that there is a significant association between age and marital status. If the p-value is not less than the significance level, the null hypothesis is not rejected, and it is concluded that there is no significant association between age and marital status.

Helping codes

Changing a column value

```
# Assuming data is your DataFrame and 'column' is the column with the values
tertiary_rows = data[data['education'] == 'tertiary']

# Randomly select 80% of the tertiary rows
group1 = tertiary_rows.sample(frac=0.8, random_state=1)

# Select the rest of the tertiary rows
group2 = tertiary_rows.drop(group1.index)

# If you want to include the primary and secondary rows in group2
group2 = pd.concat([group2, data[data['education'].isin(['primary', 'secondary'])]])

data.loc[data['Id'].isin(group1['Id']), 'housing'] = 'yes'
```

This code segregates a DataFrame data into two groups based on the 'education' column: 80% of 'tertiary' rows are randomly selected into group1, and the rest, along with 'primary' and 'secondary' rows, are put into group2. Then, it marks the 'housing' column as 'yes' for all rows in group1.

Data Analysis RoadMap

1. Load the data
2. Clean the data
3. Claculate the mode, median and mean for all columns
4. Claculate and plot the std
5. Claculate and plot the quantiles
6. Plot or scatter plot the continuous columns
7. Plot the 2-d distributions
8. Plot the 2-d heatmap distributions
9. Perform the Hypothesis Testing
10. Arbitrary analysis
11. Transform the data and find the correlation between the columns