10-11-2024 # Training Day – 35

Topic:* Styling Matplotlib Visualizations

- Experimented with styles like seaborn-dark and ggplot.
- Example: Customized a bar chart with labels, gridlines, and colors.

visualization easier and identifying outliers easily.

Matplotlib comes with a variety of built-in styles that can be applied to your plots with a single line of code. These styles can dramatically change the look and feel of your plots, making them more suitable for different purposes like presentations, reports, or technical papers

```
from matplotlib import style
```

1. IQR: It stand for "inter quartile range", which define as the difference of "third quartile(q3) and first quartile (q0)".
2. Outliers are those value which comes after the last quartile to affect our mean, as well as below the first quartile.
3. Our whole data is divided in four part i.e. 25%, 50%, 75%, 100%, and these percentile values refers to our quartile(q1,q2,q3,q4).

```
from matplotlib import style
print(plt.style.available)
```

Output:

['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot','grayscale','seaborn','seaborn-bright','seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster','seaborn-talk','seaborn-ticks','seaborn-white','seaborn-whitegrid','tableau-colorblind10']

11-11-2024 **Training Day – 36**

Topic:* Data Cleaning

- Handled missing values and duplicates in a dataset.
- Example: Used fillna() to replace missing values with the column mean. visualization easier and identifying outliers easily.

What is data cleaning? Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled

```python
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('titanic.csv')
df.head()
```

Output:

```
PassengerId    Survived    Pclass    Name    Sex    Age    SibSp    Parch    Ticket    Fare
Cabin    Embarked
0    1    0    3    Braund, Mr. Owen Harris    male    22.0    1    0    A/5 21171
7.2500    NaN    S
1    2    1    1    Cumings, Mrs. John Bradley (Florence Briggs Th...    female    38.0
1    0    PC 17599    71.2833    C85    C
2    3    1    3    Heikkinen, Miss. Laina    female    26.0    0    0    STON/O2. 3101282
7.9250    NaN    S
3    4    1    1    Futrelle, Mrs. Jacques Heath (Lily May Peel)    female    35.0    1
0    113803    53.1000    C123    S
4    5    0    3    Allen, Mr. William Henry    male    35.0    0    0    373450    8.0500
NaN    S
```

jupyter project-of-DA Last Checkpoint: 29 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[8]:
```
df.shape
```

[8]:
```
(36, 6)
```

[9]:
```
(117869/6122893)*100   #find active ration by dived by total
```

[9]:
```
1.9250540553297273
```

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   State/UTs         36 non-null     object
 1   Total Cases       36 non-null     int64
 2   Active            36 non-null     int64
 3   Deaths            36 non-null     int64
 4   Active Ratio (%)  36 non-null     float64
 5   Death Ratio (%)   36 non-null     float64
dtypes: float64(2), int64(3), object(1)
memory usage: 1.8+ KB
```

[11]:
```
df.isnull().sum()
```

[11]:
```
State/UTs           0
Total Cases         0
Active              0
Deaths              0
Active Ratio (%)    0
Death Ratio (%)     0
dtype: int64
```

[12]:
```
df.columns
```

12-11-2024 # Training Day – 37

Topic:* Standardizing Data

- Applied transformations to ensure consistency in data formatting.
- Example: Converted all text columns to uppercase using .str.upper**().**

Data standardization is an important technique that is mostly performed as a pre-processing step before inputting data into many machine learning models, to standardize the range of features of an input data set.

Standardizing Data in Excel

Excel STANDARDIZE is available under Excel Statistical Functions. It returns a normalized value, which is also called Z-score.

The mean and standard deviation are the basis of the z-score. The z-score (or standard score) is a method to standardize scores across the same scale. It divides a score's deviation by the standard deviation in a data set. The resulting score is the standard deviation of a data point from the mean.

Zero is the average of all z-scores for a dataset. A negative z score indicates that the value is lower than the mean. A positive z score indicates that the value is higher than the mean.

Z-Score Formula = STANDARDIZE(x, mean, standard_dev)

Here: X= data value that you need to normalize.

Mean= Distribution arithmetic mean

Standard_dev= Distribution standard deviation.

13-11-2024 **Training Day – 38**

Topic:* Combining Datasets with Pandas

- Learned to concatenate and merge datasets.
- Example: Merged two datasets on a common key using pd.merge().set.

Combining Multiple Datasets:

- **Concatenation:**
  Combine along rows or columns.

  python
  Copy code
  ```
  df1 = pd.DataFrame({'A': [1, 2]})
  df2 = pd.DataFrame({'A': [3, 4]})
  combined = pd.concat([df1, df2])
  ```

- **Merging (Join):**
  Combine based on a common key.

  python
  Copy code
  ```
  df1 = pd.DataFrame({'ID': [1, 2], 'Name': ['Alice', 'Bob']})
  df2 = pd.DataFrame({'ID': [1, 2], 'Score': [85, 90]})
  merged = pd.merge(df1, df2, on='ID')
  ```

- **Cleaning After Merging:**
  Ensure no duplicate or irrelevant columns remain.

  Copy code
  ```
  merged.dropna(inplace=True)
  ```

Topic: Combining Datasets with Pandas

- Learned to concatenate and merge datasets.
- **Example:**

```
import pandas as pd
df1 = pd.DataFrame({"ID": [1, 2], "Value": [10, 20]})
df2 = pd.DataFrame({"ID": [1, 2], "Description": ["A", "B"]}
merged_df = pd.merge(df1, df2, on="ID")
print(merged_df)
```

Output:

```
   ID  Value Description
0   1     10           A
1   2     20           B
```

16-11-2024 **Training Day – 40**

November 12, Tuesday*

Topic: Advanced Groupby Operations

- Applied multiple aggregation functions to grouped data.
- Example: Calculated mean and max for grouped columns.

Advanced Groupby Operations

Import Libraries

python
Copy code
```
import pandas as pd
import numpy as np
```

Create the Dataset

python
Copy code
```
data = {
    "Department": ["HR", "HR", "IT", "IT", "Finance", "Finance", "HR", "IT"],
    "Employee": ["Alice", "Bob", "Charlie", "David", "Eve", "Frank", "Grace", "Hank"],
    "Salary": [50000, 60000, 80000, 90000, 70000, 75000, 62000, 88000],
    "Bonus": [5000, 7000, 10000, 12000, 8000, 8500, 6000, 11000],
    "Years": [2, 3, 5, 6, 4, 4, 3, 7]
}
df = pd.DataFrame(data)
df
```

Output of Dataset

Department	Employee	Salary	Bonus	Years
HR	Alice	50000	5000	2
HR	Bob	60000	7000	3
IT	Charlie	80000	10000	5
IT	David	90000	12000	6
Finance	Eve	70000	8000	4
Finance	Frank	75000	8500	4
HR	Grace	62000	6000	3
IT	Hank	88000	11000	7

Applying Advanced Groupby Operations

1. Multiple Aggregations

python
Copy code
```
grouped = df.groupby("Department").agg({
    "Salary": ["mean", "sum", "max"],
    "Bonus": ["sum", "max"],
    "Years": ["mean"]
})
```

```
print(grouped)
```

Output

	Salary			Bonus		Years
Department	mean	sum	max	sum	max	mean
HR	57333.33	172000	62000	18000	7000	2.67
IT	86000.00	258000	90000	33000	12000	6.00
Finance	72500.00	145000	75000	16500	8500	4.00

---

2. Custom Aggregation Function

python
Copy code

```
def custom_salary_range(series):
    return series.max() - series.min()

grouped_custom = df.groupby("Department").agg({
    "Salary": ["mean", custom_salary_range],
    "Bonus": "sum"
})
print(grouped_custom)
```

Output

	Salary		Bonus
Department	mean	custom_salary_range	sum
HR	57333.33	12000	18000
IT	86000.00	10000	33000
Finance	72500.00	5000	16500

---

3. Broadcasting Aggregation Results

python
Copy code

```
df["Total Salary by Dept"] = df.groupby("Department")["Salary"].transform("sum")
df["Max Bonus by Dept"] = df.groupby("Department")["Bonus"].transform("max")
print(df)
```

Output

Department	Employee	Salary	Bonus	Years	Total Salary by Dept	Max Bonus by Dept
HR	Alice	50000	5000	2	172000	7000
HR	Bob	60000	7000	3	172000	7000
IT	Charlie	80000	10000	5	258000	12000
IT	David	90000	12000	6	258000	12000
Finance	Eve	70000	8000	4	145000	8500
Finance	Frank	75000	8500	4	145000	8500
HR	Grace	62000	6000	3	172000	7000
IT	Hank	88000	11000	7	258000	12000

Discussion

Advanced `groupby` operations are crucial for deriving insights from grouped data. These techniques include:

- Applying multiple aggregation functions.
- Using custom functions to extract specific insights.
- Broadcasting results back to the original dataset for further analysis.