From Excel to Pandas: 10 Essential Functions You Must Know

Alexander Joel Molinar Sr

October 14, 2024

1 From Excel to Pandas: 10 Essential Functions You Must Know

Excel functions are powerful tools that many professionals use daily for data analysis. However, when dealing with larger datasets or requiring more advanced operations, transitioning to Python's pandas library can be beneficial. In this guide, we'll recreate common Excel functions using pandas with a randomly generated employee salary dataset.

1.1 Generating a Random Employee Dataset

First, we'll create a random dataset to simulate employee data. We'll use the `faker` library to generate random names and other relevant information.

[1]:
```
# # Install faker if not already installed
# !pip install faker
# !pip install pandas
# !pip install numpy
# !pip install tabulate

import pandas as pd
import numpy as np
from faker import Faker
import random
from tabulate import tabulate

pd.set_option("display.notebook_repr_html", False)

from IPython.core.display import HTML
```

[2]:
```
# Initialize Faker
fake = Faker()

# Set the random seed for reproducibility
Faker.seed(0)
np.random.seed(0)
random.seed(0)

# Generate random data
num_employees = 100
```

```
data = {
    "Employee ID": np.arange(1001, 1001 + num_employees),
    "Name": [fake.name() for _ in range(num_employees)],
    "Department": np.random.choice(["HR", "IT", "Marketing", "Sales",␣
 ↪"Finance"], num_employees),
    "Position": [fake.job() for _ in range(num_employees)],
    "Salary": np.random.randint(40000, 120000, num_employees),
    "Date of Hire": [fake.date_between(start_date='-10y', end_date='today') for␣
 ↪_ in range(num_employees)],
    "Performance Score": np.round(np.random.uniform(1.0, 5.0, num_employees),␣
 ↪1),
    "Bonus": np.random.randint(1000, 10000, num_employees),
    "Leave Days Taken": np.random.randint(0, 15, num_employees),
    "Status": np.random.choice(["Active", "Inactive"], num_employees, p=[0.9, 0.
 ↪1])
}

# Create a DataFrame
df = pd.DataFrame(data)

# Preview the DataFrame
display(df.head())
```

```
   Employee ID              Name Department                       Position  \
0         1001      Norma Fisher    Finance       Chartered loss adjuster
1         1002    Jorge Sullivan         HR          Brewing technologist
2         1003   Elizabeth Woods      Sales          Chartered accountant
3         1004      Susan Wagner      Sales  Engineer, civil (consulting)
4         1005  Peter Montgomery      Sales      Environmental consultant

   Salary Date of Hire  Performance Score  Bonus  Leave Days Taken    Status
0  118778   2024-07-18                2.5   9393                14    Active
1   76223   2016-04-29                2.7   8468                 9  Inactive
2  101570   2023-08-06                1.2   2805                 3  Inactive
3   46521   2016-01-31                2.5   2862                 7    Active
4   96894   2022-01-14                1.1   9742                 5    Active
```

1.2 Comparing Excel Functions with Pandas Equivalents

Now, let's demonstrate how common Excel functions can be replicated in pandas, including formatting the salary figures for better readability.

1.2.1 1. SUM: Total Salary of All Employees

Excel Function: `=SUM(E2:E101)`

[3]:
```
# Pandas Equivalent with Formatting
total_salary = df['Salary'].sum()
```

```
print("Total Salary of All Employees: ${:,.2f}".format(total_salary))
```

```
Total Salary of All Employees: $8,115,608.00
```

1.2.2 2. AVERAGE: Average Salary of Employees

Excel Function: `=AVERAGE(E2:E101)`

[4]:
```
# Pandas Equivalent with Formatting
average_salary = df['Salary'].mean()
print("Average Salary of Employees: ${:,.2f}".format(average_salary))
```

```
Average Salary of Employees: $81,156.08
```

1.2.3 3. MAX: Highest Salary Among Employees

Excel Function: `=MAX(E2:E101)`

[5]:
```
# Pandas Equivalent with Formatting
highest_salary = df['Salary'].max()
print("Highest Salary Among Employees: ${:,.2f}".format(highest_salary))
```

```
Highest Salary Among Employees: $119,983.00
```

1.2.4 4. MIN: Lowest Salary Among Employees

Excel Function: `=MIN(E2:E101)`

[6]:
```
# Pandas Equivalent with Formatting
lowest_salary = df['Salary'].min()
print("Lowest Salary Among Employees: ${:,.2f}".format(lowest_salary))
```

```
Lowest Salary Among Employees: $40,469.00
```

1.2.5 5. COUNT: Number of Employees with a Salary

Excel Function: `=COUNT(E2:E101)`

[7]:
```
# Pandas Equivalent with Formatting
number_with_salary = df['Salary'].count()
print("Number of Employees with a Salary: {:,}".format(number_with_salary))
```

```
Number of Employees with a Salary: 100
```

1.2.6 6. COUNTA: Total Number of Employees Listed

Excel Function: `=COUNTA(B2:B101)`

[8]:
```
# Pandas Equivalent with Formatting
total_employees = df['Name'].count()
print("Total Number of Employees Listed: {:,}".format(total_employees))
```

```
Total Number of Employees Listed: 100
```

1.2.7 7. COUNTIF: Employees with Bonuses Greater Than 5000

Excel Function: `=COUNTIF(H2:H101, ">5000")`

[9]:
```
# Pandas Equivalent with Formatting
employees_with_bonus_gt_5000 = (df['Bonus'] > 5000).sum()
print("Employees with Bonuses Greater Than 5000: {:,}".
 ↪format(employees_with_bonus_gt_5000))
```

```
Employees with Bonuses Greater Than 5000: 50
```

1.2.8 8. SUMIF: Sum of Salaries for Employees with Bonuses Greater Than 5000

Excel Function: `=SUMIF(H2:H101, ">5000", E2:E101)`

[10]:
```
# Pandas Equivalent with Formatting
sum_salaries_bonus_gt_5000 = df.loc[df['Bonus'] > 5000, 'Salary'].sum()
print("Sum of Salaries for Bonuses > 5000: ${:,.2f}".
 ↪format(sum_salaries_bonus_gt_5000))
```

```
Sum of Salaries for Bonuses > 5000: $4,092,457.00
```

1.2.9 9. IF: Classify Employee Performance Based on Score

Excel Function: `=IF(G2>4, "Excellent", "Needs Improvement")`

[12]:
```
# Pandas Equivalent
df['Performance Classification'] = np.where(df['Performance Score'] > 4,␣
 ↪"Excellent", "Needs Improvement")


# Display the first few rows using tabulate
table_data = df[['Name', 'Performance Score', 'Performance Classification']].
 ↪head()
table_data.head()
```

[12]:

	Name	Performance Score	Performance Classification
0	Norma Fisher	2.5	Needs Improvement
1	Jorge Sullivan	2.7	Needs Improvement
2	Elizabeth Woods	1.2	Needs Improvement
3	Susan Wagner	2.5	Needs Improvement
4	Peter Montgomery	1.1	Needs Improvement

1.2.10 10. VLOOKUP: Find the Position of an Employee with a Specific ID

Excel Function: `=VLOOKUP(1025, A2:J101, 4, FALSE)`

[13]:
```python
# Pandas Equivalent
employee_id = 1025
employee_position = df.loc[df['Employee ID'] == employee_id, 'Position'].values
if len(employee_position) > 0:
    print(f"Position of Employee with ID {employee_id}: {employee_position[0]}")
else:
    print(f"Employee with ID {employee_id} not found.")
```

```
Position of Employee with ID 1025: Plant breeder/geneticist
```

1.3 Conclusion

Transitioning from Excel to pandas can greatly enhance your data analysis capabilities. Pandas offers efficient and scalable operations, especially for large datasets. By understanding the equivalents of common Excel functions in pandas and applying proper formatting, you can leverage Python's power to perform complex data manipulations with ease.

Feel free to use this notebook as a starting point for your data analysis tasks. The combination of pandas and Python provides a robust platform for handling and analyzing data, making it a valuable skill for any data professional.