ACC 424 Accounting Information System

Benford's Law Analysis
Python notes for 2.1-2.3

Section 001 MW 9:30 AM – 10:45 AM at Rm. 257 over Jan 13 – May 07
Section 002 MW 11:00 AM – 12:15 AM at Rm. 127 over Jan 13 – May 07

2.1 ***What is Benford's Law Analysis?***

Benford's Law is a statistical principle that describes how the **first digits** of numbers in naturally occurring datasets are distributed. According to this law, **lower** digits appear as the **first digit** more **frequently** than **higher** digits.

For example, the number 1 is expected to appear as the first digit about **30.1%** of the time, the number 2 is expected to appear as the first digit about **17.6%** of the time, the number 3 is expected to appear as the first digit about **12.5%** of the time, and so on, the number 9 appears only about **4.6%** of the time. We can observe that each **subsequent** digit appears **less frequently**. This distribution holds across many real-world datasets, including financial transactions, stock prices, and population figures.

2.2. Application of Benford's Law Analysis

In **accounting fraud detection**, Benford's Law is a powerful tool for identifying **anomalies** in financial records. When a company reports its revenues, expenses, or other financial figures, the numbers should typically **follow** Benford's expected distribution. If the actual distribution deviates significantly from this pattern, it may suggest possible manipulation of financial results.

Auditors use Benford's Law to examine financial statements and transactional data. If an organization is engaged in fraudulent activities, managers manipulating the numbers may not be aware of this natural distribution and might introduce **distortions** that make their falsifications detectable. For example, if certain digits, such as 6 or 7, appear far more often than expected while digits like 1 and 2 appear less frequently, it raises a **red flag** for further investigation.

Regulators, tax authorities, and internal auditors apply this method as a **preliminary screening tool**. Benford's Law serves as an indicator that warrants deeper analysis. If **discrepancies** are found in reported revenue or expense accounts, auditors may then conduct more detailed investigations, such as **reviewing source documents** or **conducting interviews**, to determine whether **intentional misstatements** have occurred.

2.3. *Python coding of Benford's Law Analysis*

https://anaconda.cloud/share/notebooks/ed1b63e8-c936-4fa7-b83a-a08400b2f986/overview

Our class 8 coding script can be downloaded from the above link.

```
def analyze_benfords_law(numbers):
```

This defines a function called **analyze_benfords_law** that takes one argument, numbers, which is expected to be a **list** of numerical values.

Extract first digits

```
first_digits = [int(str(abs(num))[0]) for num in numbers if num != 0]
```

This line creates a list of **first digits** extracted from each number in numbers.

abs(num): Takes the absolute value to handle negative numbers.

str(abs(num)): Converts the number into a string so that we can access the first character.

[0]: Selects the **first** character (i.e., the first digit).

int(...): Converts the first character back into an **integer**.

if num != 0: Ensures that zero values (which don't have a meaningful first digit) are **excluded**.

Example:

- Input list: [1234, -2345, 3456, 0, 7890]
- First digits extracted: [1, 2, 3, 7]

Calculate Actual Frequencies

```
digit_counts = pd.Series(first_digits).value_counts().sort_index()
```

pd.Series(first_digits): Creates a Pandas Series (a column of values). first_digits is a list containing the first digits extracted from the numbers in the dataset. pd.Series(first_digits) converts this list into a Pandas Series, which allows for easier data manipulation and analysis.

value_counts(): Counts how often each digit appears.

.sort_index(): Sorts the counts by digit (from 1 to 9).

Example:

- First digits: [1, 2, 3, 7, 1, 3, 1, 2]
- Count result: {1: 3, 2: 2, 3: 2, 7: 1}

Normalize the Counts to Get Frequencies

```
actual_frequencies = digit_counts / len(first_digits)
```

Divides the count of each digit by the total number of digits to get the proportion.

Example:

- If digit 1 appears **3 times** in **8 numbers**, then 3 / 8 = 0.375.

Define Benford's Expected Frequencies

```
expected_frequencies = pd.Series({
    1: 0.301, 2: 0.176, 3: 0.125, 4: 0.097,
    5: 0.079, 6: 0.067, 7: 0.058, 8: 0.051, 9: 0.046
})
```

This creates a Pandas Series with the expected probabilities for each digit based on Benford's Law.

Each digit is assigned a probability that represents how often it should appear as the first digit in **naturally occurring datasets**.

Example:

- 1 should appear **30.1%** of the time
- 2 should appear **17.6%** of the time, and so on.

Compare Actual and Expected Frequencies

```
comparison = pd.DataFrame({
    'Actual': actual_frequencies,
    'Expected': expected_frequencies,
    'Difference': abs(actual_frequencies - expected_frequencies)
})
```

Creates a Pandas DataFrame (**a table**) with:

- 'Actual': The actual frequency of each first digit.
- 'Expected': The expected frequency from Benford's Law.
- 'Difference': The absolute difference between the two

Return the Dataframe

```
return comparison
```

The function returns the DataFrame so it can be used later.

Code Execution: Running the Function

```
if __name__ == "__main__":
```

This ensures the script runs only when executed directly.

Example Data

```
sample_data = [
    1234, 2345, 3456, 1478, 1599, 1788,
    2876, 3981, 1002, 1234, 5438, 1123
]
```

A list of sample financial numbers to analyze.

Run the Function and Print Results

```
results = analyze_benfords_law(sample_data)
print("Benford's Law Analysis Results:")
print(results)
```

Calls the function analyze_benfords_law(sample_data).

Prints the comparison table.

Check for Significant Deviations

```
significant_deviations = results[results['Difference'] > 0.05]
```

Filters the DataFrame to keep only rows where the difference between actual and expected frequencies is greater than **0.05** .

Print Any Significant Deviations

```
if not significant_deviations.empty:
    print("\nSignificant deviations found for digits:")
    for index, row in significant_deviations.iterrows():
        print(f"Digit {index}: Expected {row['Expected']:.3f}, Actual {row['Actual']:.3f}")
```

If significant deviations exist, prints details about those digits.