PANDAS LIBRARY DEMONSTRATION(CCP)
Code Explaination:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
import io
```

These lines import the necessary libraries:

- pandas (as pd): For data manipulation and analysis.
- numpy (as np): For numerical operations.
- matplotlib.pyplot (as plt): For creating visualizations.
- google.colab.files: To handle file uploads in Google Colab.
- io: To work with in-memory text streams.

library_addcontent_copy

```
print("=== PANDAS LIBRARY DEMONSTRATION ===")
print("This program demonstrates CSV file upload and data analysis using Pandas")
print("-" * 60)
```

These lines print a header and description for the program.
library_addcontent_copy

```
# Step 1: Upload CSV file
print("Step 1: Uploading CSV file...")
print("Please select a CSV file to upload:")

uploaded = files.upload()
```

This section prompts the user to upload a CSV file using Google Colab's file upload functionality. The uploaded file(s) are stored in the uploaded dictionary.
library_addcontent_copy

```
# Step 2: Process the uploaded file
print("\nStep 2: Processing uploaded file...")

filename = list(uploaded.keys())[0]
print(f"Uploaded file: {filename}")
```

This part gets the name of the first uploaded file from the uploaded dictionary and prints it.
library_addcontent_copy

```
# Step 3: Read CSV file into a Pandas DataFrame
print("\nStep 3: Reading CSV file into Pandas DataFrame...")

file_content = uploaded[filename]
```

```
df = pd.read_csv(io.StringIO(file_content.decode('utf-8')))

print("✓ CSV file successfully loaded into DataFrame!")
print(f"DataFrame shape: {df.shape}")
```

Here, the content of the uploaded file is read. `io.StringIO` is used to treat the file content as a text stream, and `pd.read_csv()` reads this stream into a pandas DataFrame named `df`. It then prints a success message and the shape of the DataFrame.
library_addcontent_copy

```
# Step 4: Display basic information about the dataset
print("\n" + "="*50)
print("DATASET OVERVIEW")
print("="*50)
```

This prints a header for the dataset overview section.
library_addcontent_copy

```
# Display first few rows
print("\n1. First 5 rows of the dataset:")
print(df.head())
```

This displays the first 5 rows of the DataFrame using the `head()` method.
library_addcontent_copy

```
# Display basic information
print("\n2. Dataset Information:")
print(f"   - Number of rows: {len(df)}")
print(f"   - Number of columns: {len(df.columns)}")
print(f"   - Column names: {list(df.columns)}")
```

This prints the number of rows, columns, and the names of the columns in the DataFrame.
library_addcontent_copy

```
# Display data types
print("\n3. Data Types:")
print(df.dtypes)
```

This displays the data types of each column in the DataFrame using the `dtypes` attribute.
library_addcontent_copy

```
# Display summary statistics for numeric columns
print("\n4. Summary Statistics (Numeric Columns):")
print(df.describe())
```

This calculates and displays descriptive statistics (like mean, standard deviation, min, max, etc.) for the numeric columns in the DataFrame using the `describe()` method.
library_addcontent_copy

```
# Check for missing values
print("\n5. Missing Values Check:")
missing_values = df.isnull().sum()
print(missing_values)

if missing_values.sum() > 0:
    print(f"    Total missing values: {missing_values.sum()}")
else:
    print("    No missing values found!")
```

This checks for missing values in each column using `isnull().sum()` and prints the count of missing values per column. It also prints the total number of missing values if any are found.
library_addcontent_copy

```
# Step 5: Advanced Data Analysis (if numeric columns exist)
numeric_columns = df.select_dtypes(include=[np.number]).columns

if len(numeric_columns) > 0:
    print("\n" + "="*50)
    print("ADVANCED ANALYSIS")
    print("="*50)

    print(f"\nNumeric columns found: {list(numeric_columns)}")

    # Calculate correlation matrix
    print("\n6. Correlation Matrix:")
    correlation_matrix = df[numeric_columns].corr()
    print(correlation_matrix)

    # Find columns with highest correlation
    if len(numeric_columns) > 1:
        # Get absolute values and remove diagonal
        corr_abs = correlation_matrix.abs()
        np.fill_diagonal(corr_abs.values, 0)

        # Find maximum correlation
        max_corr = corr_abs.max().max()
        max_corr_pair = corr_abs.stack().idxmax()

        print(f"\n7. Highest Correlation:")
        print(f"   Columns: {max_corr_pair[0]} & {max_corr_pair[1]}")
        print(f"   Correlation coefficient: {max_corr:.3f}")

else:
    print("\n  No numeric columns found for advanced analysis.")
```

This section performs advanced analysis if there are numeric columns. It calculates the correlation matrix for numeric columns, finds the pair of columns with the highest absolute correlation, and prints the correlation matrix and the highest correlation pair.
library_addcontent_copy

```
# Step 6: Data Visualization (if possible)
print("\n" + "="*50)
print("DATA VISUALIZATION")
print("="*50)

if len(numeric_columns) > 0:
    # Create a simple histogram for the first numeric column
    plt.figure(figsize=(10, 6))

    first_numeric_col = numeric_columns[0]
    plt.hist(df[first_numeric_col].dropna(), bins=20, alpha=0.7, color='skyblue')
    plt.title(f'Distribution of {first_numeric_col}')
    plt.xlabel(first_numeric_col)
    plt.ylabel('Frequency')
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    print(f"✓ Histogram created for column: {first_numeric_col}")
else:
    print("  No numeric columns available for visualization.")
```

This part performs data visualization if numeric columns exist. It creates a histogram for the first numeric column to show its distribution.
library_addcontent_copy

```
# Step 7: Data Export Example
print("\n" + "="*50)
print("DATA EXPORT EXAMPLE")
print("="*50)

# Create a sample processed dataset
processed_df = df.copy()

# Add a sample calculated column if numeric data exists
if len(numeric_columns) > 0:
    first_col = numeric_columns[0]
    processed_df[f'{first_col}_normalized'] = (df[first_col] - df[first_col].mean()) / df[first_col].std()
    print(f"✓ Added normalized column for {first_col}")

# Show export options
print("\n8. Export Options Available:")
print("   - CSV: processed_df.to_csv('output.csv', index=False)")
print("   - Excel: processed_df.to_excel('output.xlsx', index=False)")
print("   - JSON: processed_df.to_json('output.json')")
```

This section demonstrates data export options. It creates a copy of the DataFrame, adds a normalized column for the first numeric column if it exists, and then shows examples of how to export the processed data to CSV, Excel, and JSON formats.
library_addcontent_copy

```
# Step 8: Summary and Conclusion
print("\n" + "="*60)
print("ANALYSIS COMPLETE - SUMMARY")
print("="*60)

print(f"""
 Successfully processed CSV file: {filename}
 Dataset contains {len(df)} rows and {len(df.columns)} columns
 Numeric columns: {len(numeric_columns)}
 Missing values: {df.isnull().sum().sum()}
 Data types: {df.dtypes.value_counts().to_dict()}
""")

print(" Pandas demonstration completed successfully!")
print("\nThis program demonstrated:")
print("• File upload capabilities")
print("• Data loading and inspection")
print("• Statistical analysis")
print("• Data visualization")
print("• Export options")
```

Finally, this section prints a summary of the analysis performed, including the file name, dataset dimensions, number of numeric columns, total missing values, and data type counts.

Output Explaination:

1. **Header and Introduction:**

library_addcontent_copy

```
=== PANDAS LIBRARY DEMONSTRATION ===
This program demonstrates CSV file upload and data analysis using Pandas
------------------------------------------------------------
Step 1: Uploading CSV file...
Please select a CSV file to upload:
```

This is simply the program printing the initial header and instructions, as defined in the first `print` statements of the code. It's guiding you to upload a file.

1. **File Upload Interaction:**

library_addcontent_copy

```
Saving employees.csv to employees.csv
```

This line confirms that you successfully uploaded a file named `employees.csv` and that Colab has saved it in the environment.

1. **File Processing and DataFrame Loading:**

library_addcontent_copy

```
Step 2: Processing uploaded file...
Uploaded file: employees.csv
```

```
Step 3: Reading CSV file into Pandas DataFrame...
✓ CSV file successfully loaded into DataFrame!
DataFrame shape: (1000, 8)
```

This shows the program moving to the next steps. It confirms the name of the uploaded file it will process (`employees.csv`). Then, it indicates that it has successfully read the CSV data into a pandas DataFrame. The `DataFrame shape: (1000, 8)` tells you that the DataFrame has 1000 rows and 8 columns.

1. **Dataset Overview Header:**

library_addcontent_copy

```
==================================================
DATASET OVERVIEW
==================================================
```

This is a header indicating the start of the section providing basic information about the loaded dataset.

1. **First 5 Rows:**

library_addcontent_copy

```
1. First 5 rows of the dataset:
  First Name  Gender Start Date Last Login Time  Salary  Bonus %  \
0    Douglas    Male   8/6/1993       12:42 PM   97308    6.945
1     Thomas    Male  3/31/1996        6:53 AM   61933    4.170
2      Maria  Female  4/23/1993       11:17 AM  130590   11.858
3      Jerry    Male   3/4/2005        1:00 PM  138705    9.340
4      Larry    Male  1/24/1998        4:47 PM  101004    1.389

  Senior Management            Team
0              True       Marketing
1              True             NaN
2             False         Finance
3              True         Finance
4              True  Client Services
```

This is the output of `df.head()`. It shows the first 5 rows of your `employees.csv` file. This is useful for getting a quick look at the structure and content of your data. The output includes the column names and the values for the first five entries. Notice that some values are missing (`NaN` in the 'Team' column for row 1).

1. **Dataset Information:**

library_addcontent_copy

```
2. Dataset Information:
  - Number of rows: 1000
  - Number of columns: 8
  - Column names: ['First Name', 'Gender', 'Start Date', 'Last Login Time', 'Salary', 'Bonus %',
'Senior Management', 'Team']
```

This provides a summary of the dataset's dimensions and lists all the column names. This confirms the shape we saw earlier and gives you the exact names of each column.

1. **Data Types:**

library_addcontent_copy

```
3. Data Types:
First Name          object
Gender              object
Start Date          object
Last Login Time     object
Salary               int64
Bonus %            float64
Senior Management   object
Team                object
dtype: object
```

This is the output of `df.dtypes`. It shows the data type that pandas inferred for each column.

- `object`: This typically means the column contains strings or mixed data types. 'First Name', 'Gender', 'Start Date', 'Last Login Time', 'Senior Management', and 'Team' are interpreted as objects (likely strings).
- `int64`: This indicates integer numbers. 'Salary' is an integer column.
- `float64`: This indicates floating-point numbers (numbers with decimal points). 'Bonus %' is a float column.

1. **Summary Statistics (Numeric Columns):**

library_addcontent_copy

```
4. Summary Statistics (Numeric Columns):
             Salary      Bonus %
count    1000.000000  1000.000000
mean    90662.181000    10.207555
std     32923.693342     5.528481
min     35013.000000     1.015000
25%     62613.000000     5.401750
50%     90428.000000     9.838500
75%    118740.250000    14.838000
max    149908.000000    19.944000
```

This is the output of `df.describe()` for the numeric columns ('Salary' and 'Bonus %'). It provides key statistical measures:

- `count`: The number of non-null values in the column (1000 for both, meaning no missing values in these columns).
- `mean`: The average value.
- `std`: The standard deviation, a measure of the spread of the data.
- `min`: The minimum value.
- `25%`: The first quartile (25% of the data is below this value).
- `50%`: The median (the middle value).
- `75%`: The third quartile (75% of the data is below this value).

- `max`: The maximum value.

1. **Missing Values Check:**

library_addcontent_copy

```
5. Missing Values Check:
First Name          67
Gender             145
Start Date           0
Last Login Time      0
Salary               0
Bonus %              0
Senior Management    67
Team                43
dtype: int64
   Total missing values: 322
```

This is the output of `df.isnull().sum()`. It shows the count of missing (null) values for each column. As we saw in `df.head()`, 'Gender', 'First Name', 'Senior Management', and 'Team' have missing values. 'Salary' and 'Bonus %' do not. The `Total missing values: 322` is the sum of all missing values across all columns.

1. **Advanced Analysis Header:**

library_addcontent_copy

```
==================================================
ADVANCED ANALYSIS
==================================================
```

This header indicates the start of the section for more advanced analysis, which in this code focu ses on numeric column relationships.

1. **Numeric Columns Found:**

library_addcontent_copy

```
Numeric columns found: ['Salary', 'Bonus %']
```

This line confirms which columns were identified as numeric and used for the advanced analysis.

1. **Correlation Matrix:**

library_addcontent_copy

```
6. Correlation Matrix:
          Salary   Bonus %
Salary   1.000000 -0.036381
Bonus % -0.036381  1.000000
```

This is the output of `df[numeric_columns].corr()`. The correlation matrix shows the Pearson corr elation coefficient between pairs of numeric columns.
* The diagonal values are always 1.0 because a column is perfectly correlated with itself.
* The value `-

0.036381` (and its symmetric value) represents the correlation between 'Salary' and 'Bonus %'. A value close to 0 indicates a very weak linear relationship between the two variables.

1. **Highest Correlation:**

library_addcontent_copy

```
7. Highest Correlation:
  Columns: Salary & Bonus %
  Correlation coefficient: 0.036
```

This part identifies the pair of numeric columns with the highest absolute correlation (excluding t he correlation of a column with itself). In this case, it's 'Salary' and 'Bonus %', with an absolute co rrelation coefficient of approximately 0.036. This confirms the weak linear relationship seen in th e correlation matrix.

1. **Data Visualization Header:**

library_addcontent_copy

```
==================================================
DATA VISUALIZATION
==================================================
```

This header indicates the start of the visualization section.

1. **Histogram Output:**

library_addcontent_copy

```
<Figure size 1000x600 with 1 Axes>
```

This is the output indicating that a matplotlib figure object was created for the histogram. Below this line in Google Colab, you would typically see the generated histogram plot itself. The `✓ Hist ogram created for column: Salary` line confirms that the histogram was successfully generated fo r the 'Salary' column.

1. **Data Export Example Header:**

library_addcontent_copy

```
==================================================
DATA EXPORT EXAMPLE
==================================================
```

This header indicates the start of the data export example section.

1. **Normalized Column Added:**

library_addcontent_copy

```
✓ Added normalized column for Salary
```

This confirms that the code successfully created a new column called `Salary_normalized` in the ` processed_df` DataFrame. This column contains the normalized values of the 'Salary' column (ea ch value is transformed so the column has a mean of 0 and a standard deviation of 1).

1. **Export Options Available:**

library_addcontent_copy

```
8. Export Options Available:
  - CSV: processed_df.to_csv('output.csv', index=False)
  - Excel: processed_df.to_excel('output.xlsx', index=False)
  - JSON: processed_df.to_json('output.json')
```

This section shows example code snippets you could use to export the `processed_df` DataFrame to different file formats (CSV, Excel, and JSON). It doesn't actually perform the export, but demonstrates how you would do it. `index=False` prevents pandas from writing the DataFrame index as a column in the output file.

1. **Summary and Conclusion Header:**

library_addcontent_copy

```
============================================================
ANALYSIS COMPLETE - SUMMARY
============================================================
```

This header indicates the end of the analysis and the start of the summary.

1. **Analysis Summary:**

library_addcontent_copy

```
Successfully processed CSV file: employees.csv
Dataset contains 1000 rows and 8 columns
Numeric columns: 2
Missing values: 322
Data types: {dtype('O'): 6, dtype('int64'): 1, dtype('float64'): 1}

Pandas demonstration completed successfully!

This program demonstrated:
• File upload capabilities
• Data loading and inspection
• Statistical analysis
• Data visualization
• Export options
```

This final section provides a concise summary of the key findings and actions performed by the script:

* Confirms the file processed.
* Restates the dataset's dimensions.
* States the number of numeric columns found.
* Reports the total count of missing values across the entire dataset.
* Summarizes the count of each data type found in the DataFrame.
* Lists the capabilities that the program demonstrated.

In essence, the output walks you through the steps the code took: loading the data, inspecting its contents and structure, performing basic statistics and a correlation analysis, creating a simple visualization, and showing examples of how to export the processed data.