```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
from fpdf import FPDF
from jinja2 import Template

class DataReport:
    def __init__(self, df):
        self.df = df
        self.output_folder = "output"
        os.makedirs(self.output_folder, exist_ok=True)

    def missing_values(self):
        missing = self.df.isnull().sum()
        missing = missing[missing > 0]
        if missing.empty:
            print("No missing values found.")
        else:
            print("Columns with missing values:")
            print(missing)

    def categorize_columns(self):
        numeric_cols =
self.df.select_dtypes(include=[np.number]).columns.tolist()
        categorical_cols =
self.df.select_dtypes(include=['object']).columns.tolist()
        print("Numeric columns:", numeric_cols)
        print("Categorical columns:", categorical_cols)

    def handle_duplicates(self):
        before = len(self.df)
        self.df = self.df.drop_duplicates()
        after = len(self.df)
        print(f"Duplicates removed: {before - after}")

    def handle_constants(self):
        before = self.df.shape[1]
        constant_cols = [col for col in self.df.columns if
self.df[col].nunique() == 1]
        self.df.drop(columns=constant_cols, inplace=True)
        after = self.df.shape[1]
        print(f"Constant columns removed: {before - after}")
        print("Removed columns:", constant_cols)

    def create_box_plots(self):
        numeric_cols =
self.df.select_dtypes(include=[np.number]).columns.tolist()
        for col in numeric_cols:
```

```
            plt.figure(figsize=(8, 6))
            sns.boxplot(x=self.df[col])
            plt.title(f'Box Plot for {col}')
            plt.savefig(f'{self.output_folder}/{col}_boxplot.png')
            plt.close()

    def create_distributions(self):
        cols_to_plot = self.df.columns[:6]  # Choose first 6 columns
for visualization
        for col in cols_to_plot:
            plt.figure(figsize=(8, 6))
            if self.df[col].dtype in [np.number]:
                sns.histplot(self.df[col].dropna(), kde=True)
            else:
                sns.countplot(y=self.df[col],
order=self.df[col].value_counts().index)
            plt.title(f'Distribution for {col}')

plt.savefig(f'{self.output_folder}/{col}_distribution.png')
            plt.close()

    def generate_report(self):
        # Gather data for the report
        summary = {
            "missing_values": self.df.isnull().sum().to_dict(),
            "categorical_columns":
self.df.select_dtypes(include=['object']).columns.tolist(),
            "numeric_columns":
self.df.select_dtypes(include=[np.number]).columns.tolist()
        }

        # Create HTML report
        html_template = """
        <!DOCTYPE html>
        <html>
        <head>
            <title>Data Report</title>
        </head>
        <body>
            <h1>Data Report</h1>
            <h2>Missing Values</h2>
            <pre>{{ missing_values }}</pre>
            <h2>Categorical Columns</h2>
            <pre>{{ categorical_columns }}</pre>
            <h2>Numeric Columns</h2>
            <pre>{{ numeric_columns }}</pre>
        </body>
        </html>
        """
        template = Template(html_template)
```

```
        html_content = template.render(**summary)

        with open(f"{self.output_folder}/report.html", "w") as file:
            file.write(html_content)

    def to_pdf(self):
        pdf = FPDF()
        pdf.set_auto_page_break(auto=True, margin=15)
        pdf.add_page()
        pdf.set_font("Arial", size=12)

        # Add summary
        pdf.multi_cell(0, 10, txt="Data Report Summary")

        # Add images
        for img_file in os.listdir(self.output_folder):
            if img_file.endswith(".png"):
                pdf.add_page()
                pdf.image(f"{self.output_folder}/{img_file}", x=10,
y=30, w=190)

        pdf.output(f"{self.output_folder}/report.pdf")

# Example Usage
if __name__ == "__main__":
    # Example DataFrame
    df = pd.DataFrame({
        'A': [1, 2, np.nan, 4],
        'B': [np.nan, np.nan, 2, 3],
        'C': [1, 2, 3, 4],
        'D': ['a', 'b', 'a', 'a'],
        'E': ['x', 'x', 'x', 'x']  # Constant column
    })

    report = DataReport(df)
    report.missing_values()
    report.categorize_columns()
    report.handle_duplicates()
    report.handle_constants()
    report.create_box_plots()
    report.create_distributions()
    report.generate_report()
    report.to_pdf()

Columns with missing values:
A    1
B    2
dtype: int64
Numeric columns: ['A', 'B', 'C']
Categorical columns: ['D', 'E']
```

```
Duplicates removed: 0
Constant columns removed: 1
Removed columns: ['E']

C:\Users\Saimo\AppData\Local\Temp\ipykernel_848\1854021632.py:57:
DeprecationWarning: Converting `np.inexact` or `np.floating` to a
dtype is deprecated. The current result is `float64` which is not
strictly correct.
  if self.df[col].dtype in [np.number]:
C:\Users\Saimo\AppData\Local\Temp\ipykernel_848\1854021632.py:57:
DeprecationWarning: Converting `np.inexact` or `np.floating` to a
dtype is deprecated. The current result is `float64` which is not
strictly correct.
  if self.df[col].dtype in [np.number]:
C:\Users\Saimo\AppData\Local\Temp\ipykernel_848\1854021632.py:57:
DeprecationWarning: Converting `np.inexact` or `np.floating` to a
dtype is deprecated. The current result is `float64` which is not
strictly correct.
  if self.df[col].dtype in [np.number]:
C:\Users\Saimo\AppData\Local\Temp\ipykernel_848\1854021632.py:57:
DeprecationWarning: Converting `np.inexact` or `np.floating` to a
dtype is deprecated. The current result is `float64` which is not
strictly correct.
  if self.df[col].dtype in [np.number]:
```