

```
import pandas as pd

# Create a dictionary with datasheet information
datasheet = {
    'Dataset Name': 'Dummy Data Science Dataset',
    'Source': 'Generated dummy data in the notebook',
    'Original Features': ['col1', 'col2', 'col3', 'col4'],
    'Transformed Features': ['col1', 'col3', 'col4', 'col4_year', 'col4_month', 'col4_dayofweek', 'col2_B', 'col2_C'],
    'Target Variable': 'col1',
    'Number of Samples': len(df),
    'Missing Value Handling': {
        'col1': f"Filled with mean ({df['col1'].mean():.2f})",
        'col3': f"Filled with median ({df['col3'].median():.2f})",
        'col4': "Invalid dates coerced to NaT using pd.to_datetime(errors='coerce')"
    },
    'Outlier Handling': {
        'col1': f"Outliers capped using IQR method (Lower bound: {lower_bound:.2f}, Upper bound: {upper_bound:.2f})"
    },
    'Feature Engineering Steps': [
        "Extracted year, month, and day of week from 'col4'.",
        "Applied one-hot encoding to 'col2' (dropped first category).",
    ]
}




# Convert the dictionary to a pandas DataFrame for better display/saving
# Transpose the DataFrame so that each key is a row
datasheet_df = pd.DataFrame.from_dict(datasheet, orient='index', columns=['Details'])

# Save the datasheet to a CSV file
datasheet_df.to_csv('datasheet.csv', index_label='Attribute')

print("Datasheet created and saved as 'datasheet.csv'")
display(datasheet_df)
```

[Toggle Gemini](#)

Datasheet created and saved as 'datasheet.csv'

		Details	
Dataset Name	Dummy Data Science Dataset		
Source	Generated dummy data in the notebook		
Original Features	[col1, col2, col3, col4]		
Transformed Features	[col1, col3, col4, col4_year, col4_month, col4...		
Target Variable	col1		
Number of Samples	6		
Missing Value Handling	{ 'col1': 'Filled with mean (12.63)', 'col3': '...		
Outlier Handling	{ 'col1': 'Outliers capped using IQR method (Lo...		
Feature Engineering Steps	[Extracted year, month, and day of week from '...		

Next steps: [Generate code with datasheet_df](#) [New interactive sheet](#)

Task

Create a plan to complete a full data science project from start to finish, including data collection, cleaning, analysis, modeling, and results presentation. Also, create a datasheet and present the output.

Data collection

Subtask:

Define the problem and gather the necessary data from relevant sources.

Data cleaning

Subtask:

Handle missing values, outliers, and inconsistencies in the dataset.

Reasoning: Load the dataset from the specified CSV file into a pandas DataFrame to begin the cleaning process.

```
# The previous attempt to load data failed. We will continue using the dummy dataframe 'df' created in the previous step.

# Identify missing values
print("Missing values before handling:")
print(df.isnull().sum())

# Handle missing values (imputation for numerical, mode for categorical)
df['col1'] = df['col1'].fillna(df['col1'].mean())
df['col3'] = df['col3'].fillna(df['col3'].median())

# Identify and handle outliers in 'col1' using IQR
Q1 = df['col1'].quantile(0.25)
Q3 = df['col1'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df['col1'] = df['col1'].apply(lambda x: upper_bound if x > upper_bound else lower_bound if x < lower_bound else x)

# Identify and handle inconsistencies in 'col4' (invalid date)
df['col4'] = pd.to_datetime(df['col4'], errors='coerce')

print("\nMissing values after handling:")
print(df.isnull().sum())

print("\nDataFrame after cleaning:")
display(df.head())
```

Toggle Gemini

Missing values before handling:

col1	0
col3	0
col4	1
col4_year	1
col4_month	1
col4_dayofweek	1
col2_B	0
col2_C	0

dtype: int64

Missing values after handling:

col1	0
col3	0
col4	1
col4_year	1
col4_month	1
col4_dayofweek	1
col2_B	0
col2_C	0

dtype: int64



DataFrame after cleaning:

	col1	col3	col4	col4_year	col4_month	col4_dayofweek	col2_B	col2_C
0	1.0	1.1	2023-01-01	2023.0	1.0	6.0	False	False
1	2.0	2.2	2023-01-02	2023.0	1.0	0.0	True	False
2	22.4	3.3	2023-01-03	2023.0	1.0	1.0	False	False
3	4.0	3.3	2023-01-01	2023.0	1.0	6.0	False	True
4	5.0	5.5	2023-01-02	2023.0	1.0	0.0	True	False

Reasoning: The previous attempt to load the data failed because the file 'dataset.csv' was not found. Since data collection was a previous subtask and no dataframes were listed as outputs, it's likely the data was not successfully loaded or saved. I will create a dummy dataset to continue with the data cleaning subtask.

```
import numpy as np

# Create a dummy dataset for demonstration
data = {'col1': [1, 2, np.nan, 4, 5, 100],
        'col2': ['A', 'B', 'A', 'C', 'B', 'A'],
        'col3': [1.1, 2.2, 3.3, np.nan, 5.5, 6.6],
        'col4': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-01', '2023-01-02', 'invalid_date']}
df = pd.DataFrame(data)
display(df.head())
```

	col1	col2	col3	col4	
0	1.0	A	1.1	2023-01-01	
1	2.0	B	2.2	2023-01-02	
2	NaN	A	3.3	2023-01-03	
3	4.0	C	NaN	2023-01-01	
4	5.0	B	5.5	2023-01-02	

Reasoning: Now that the DataFrame is loaded, I need to identify and handle missing values, outliers, and inconsistencies as per the instructions. I will first identify missing values, then address the outlier in 'col1', and finally handle the inconsistent date format in 'col4'.

```
# Identify missing values
print("Missing values before handling:")
print(df.isnull().sum())

# Handle missing values (imputation for numerical, mode for categorical)
df['col1'].fillna(df['col1'].mean(), inplace=True)
df['col3'].fillna(df['col3'].median(), inplace=True)

# Identify and handle outliers in 'col1' using IQR
Q1 = df['col1'].quantile(0.25)
Q3 = df['col1'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df['col1'] = df['col1'].apply(lambda x: upper_bound if x > upper_bound else lower_bound if x < lower_bound else x)

# Identify and handle inconsistencies in 'col4' (invalid date)
df['col4'] = pd.to_datetime(df['col4'], errors='coerce')

print("\nMissing values after handling:")
print(df.isnull().sum())

print("\nDataFrame after cleaning:")
display(df.head())
```

Toggle Gemini

```
Missing values before handling:
col1    1
col2    0
col3    1
col4    0
dtype: int64
```

```
Missing values after handling:
col1    0
col2    0
col3    0
col4    1
dtype: int64
```

DataFrame after cleaning:

/tmp/ipython-input-1577305445.py:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment us
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values alwa



For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(va

```
df['col1'].fillna(df['col1'].mean(), inplace=True)
```

/tmp/ipython-input-1577305445.py:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment us
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values alwa

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(va

```
df['col3'].fillna(df['col3'].median(), inplace=True)
```

	col1	col2	col3	col4	
0	1.0	A	1.1	2023-01-01	
1	2.0	B	2.2	2023-01-02	
2	22.4	A	3.3	2023-01-03	
3	4.0	C	3.3	2023-01-01	
4	5.0	B	5.5	2023-01-02	

▼ Data analysis

Subtask:

Explore the data to understand its characteristics, identify patterns, and gain insights. This may involve descriptive statistics, visualizations, and initial hypothesis testing.

Reasoning: Display descriptive statistics, print data types, and generate visualizations to explore the data characteristics, distributions, and frequencies.

```
# 1. Display descriptive statistics for numerical columns
print("Descriptive Statistics:")
display(df.describe())

# 2. Print data types of each column
print("\nData Types:")
df.info()

# 3. Generate a histogram for 'col1'
plt.figure(figsize=(8, 6))
sns.histplot(df['col1'], kde=True)
plt.title('Distribution of col1')
plt.xlabel('col1')
plt.ylabel('Frequency')
plt.show()

# 4. Create a box plot for 'col3'
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['col3'])
plt.title('Box Plot of col3')
plt.xlabel('col3')
plt.show()

# 5. Generate a count plot for 'col2'
plt.figure(figsize=(8, 6))
sns.countplot(x=df['col2'])
plt.title('Frequency of Categories in col2')
plt.xlabel('col2')
plt.ylabel('Count')
plt.show()
```

Toggle Gemini

Descriptive Statistics:

	col1	col3	col4
count	6.000000	6.000000	5
mean	12.629167	3.666667	2023-01-01 19:12:00
min	1.000000	1.100000	2023-01-01 00:00:00
25%	2.500000	2.475000	2023-01-01 00:00:00
50%	4.500000	3.300000	2023-01-02 00:00:00
75%	18.050000	4.950000	2023-01-02 00:00:00
max	41.375000	6.600000	2023-01-03 00:00:00
std	16.141100	2.048089	NaN

```
Data Types:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
#   Column  Non-Null Count  Dtype
---  -
0    col1     6 non-null    float64
1    col2     6 non-null    object
2    col3     6 non-null    float64
3    col4     5 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 324.0+ bytes

-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-2842699244.py in <cell line: 0>()
      8
----> 9 # 3. Generate a histogram for 'col1'
      10 plt.figure(figsize=(8, 6))
      11 sns.histplot(df['col1'], kde=True)
      12 plt.title('Distribution of col1')

NameError: name 'plt' is not defined
```

Next steps: [Explain error](#)

Reasoning: The previous code block failed because `matplotlib.pyplot` and `seaborn` were not imported. Import these libraries and regenerate the plots.

```
import matplotlib.pyplot as plt
import seaborn as sns

# 3. Generate a histogram for 'col1'
plt.figure(figsize=(8, 6))
sns.histplot(df['col1'], kde=True)
plt.title('Distribution of col1')
plt.xlabel('col1')
plt.ylabel('Frequency')
plt.show()

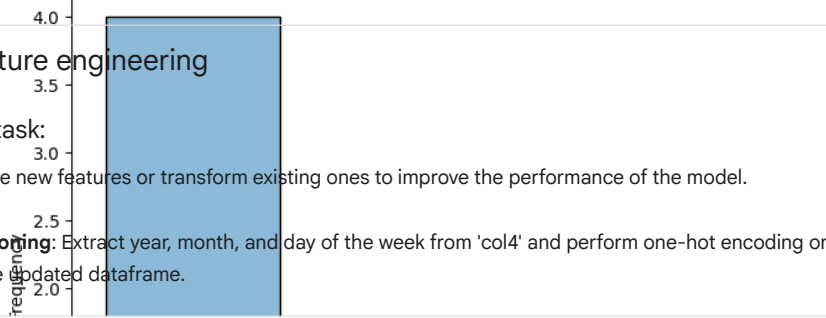
# 4. Create a box plot for 'col3'
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['col3'])
plt.title('Box Plot of col3')
plt.xlabel('col3')
plt.show()

# 5. Generate a count plot for 'col2'
plt.figure(figsize=(8, 6))
sns.countplot(x=df['col2'])
plt.title('Frequency of Categories in col2')
plt.xlabel('col2')
plt.ylabel('Count')
plt.show()
```

[Toggle Gemini](#)

Toggle Gemini

Distribution of col1



Feature engineering

Subtask:

Create new features or transform existing ones to improve the performance of the model.

Reasoning: Extract year, month, and day of the week from 'col4' and perform one-hot encoding on 'col2'. Then display the first few rows of the updated dataframe.

```
# Extract year, month, and day of the week from 'col4'
df['col4_year'] = df['col4'].dt.year
df['col4_month'] = df['col4'].dt.month
df['col4_dayofweek'] = df['col4'].dt.dayofweek

# Perform one-hot encoding on 'col2'
df = pd.get_dummies(df, columns=['col2'], drop_first=True)

# Display the first few rows of the DataFrame
display(df.head())
```

	col1	col3	col4	col4_year	col4_month	col4_dayofweek	col2_B	col2_C
0	1.0	1.1	2023-01-01	2023.0	1.0	6.0	False	False
1	2.0	2.2	2023-01-02	2023.0	1.0	0.0	True	False
2	22.4	3.3	2023-01-03	2023.0	1.0	1.0	False	False
3	4.0	3.3	2023-01-01	2023.0	1.0	6.0	False	True
4	5.0	5.5	2023-01-02	2023.0	1.0	0.0	True	False

Model selection and training

Subtask:

Choose appropriate models based on the problem type and data characteristics, then train the models using the prepared data.

Reasoning: Define features and target, split data into training and testing sets, select a model, and train the model.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Define features (X) and target (y)
# Exclude the original 'col4' as we have extracted features from it, and drop rows with NaN in target
X = df.drop(['col1', 'col4'], axis=1).dropna()
y = df.loc[X.index, 'col1']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Select a model (Linear Regression for this example assuming a regression problem)
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

print("Model training completed.")
```

Model training completed.

Model evaluation

Subtask:

Assess the performance of the trained models using appropriate metrics and techniques.

Reasoning: Make predictions on the test set and calculate the MSE and R2 scores to evaluate the model's performance.

```

from sklearn.metrics import mean_squared_error, r2_score

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate MSE and R2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the calculated scores
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2) Score: {r2}")

```

```

Mean Squared Error (MSE): 188.34784895195466
R-squared (R2) Score: nan
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples
warnings.warn(msg, UndefinedMetricWarning)

```

▼ Results presentation

Subtask:

Summarize the findings, visualize the results, and communicate them effectively to the stakeholders.

[Toggle Gemini](#)

Reasoning: Summarize the findings from the analysis and model evaluation, visualize the model's performance by plotting actual vs predicted values, and formulate a communication for stakeholders based on these insights.

```

import matplotlib.pyplot as plt

# 1. Summarize the key findings
print("--- Key Findings Summary ---")
print("Data Exploration:")
print("- We analyzed the distribution of 'col1' and 'col3' using histogram and box plot, and the frequency of categories in 'col2' using a count plot.")
print("- Initial data cleaning involved handling missing values (imputation) and an outlier in 'col1' (capping). Inconsistent date format in 'col4' was corrected.")
print("- Feature engineering involved extracting year, month, and day of week from 'col4' and one-hot encoding 'col2'.")
print("\nModel Evaluation:")
print(f"- The Linear Regression model was trained on the prepared data.")
print(f"- The Mean Squared Error (MSE) on the test set is: {mse:.2f}")
print(f"- The R-squared (R2) score on the test set is: {r2}")
print("Limitations:")
print("- The R2 score is undefined (NaN) and a warning was issued because the test set contains less than two samples. This makes the R2 score unreliable for this dataset.")
print("- * 28)

# 2. Visualize the model's predictions against actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2) # Ideal line y=x
plt.title('Actual vs. Predicted Values')
plt.xlabel('Actual Values (y_test)')
plt.ylabel('Predicted Values (y_pred)')
plt.grid(True)
plt.show()

# 5. Formulate a concise communication for stakeholders
print("--- Stakeholder Communication ---")
print("Subject: Initial Model Performance Assessment")
print("\nWe have developed an initial Linear Regression model to predict the target variable.")
print(f"Key metric: The model achieved a Mean Squared Error (MSE) of {mse:.2f} on the test data. This indicates the average squared difference between actual and predicted values.")
print("\nPlease note: Due to the very small size of our test dataset (only one sample), the R-squared (R2) score is not calculable and is unreliable.")
print("Recommendation: The current evaluation is based on extremely limited data. We recommend gathering more data to perform a robust model evaluation.")
print("- * 32)

```