Practical NO. 1:

Apply data cleaning techniques on any dataset (e.g. Chronic Kidney Disease dataset from UCI repository). Techniques may include handling missing values, outliers and inconsistent values. Also, a set of validation rules may be specified for the particular dataset and validation checks performed.

Steps:

1. Loading the dataset
2. Inspect/Know the data (EDA)
3. Handling missing values
4. Handling outliers
5. Handling inconsistent values
6. Set/Define validation rules/check
7. Apply validation checks

1) Loading the dataset

```
import pandas as pd

# Load the dataset
df = pd.read_csv('kidney_disease.csv')

# Try uploading dataset using different ways (explore...)
```

2) Inspect/Know the data (EDA)

```
#dimensions of the dataframe
df.shape
```

1) Loading the dataset

```
import pandas as pd

# Load the dataset
df = pd.read_csv('kidney_disease.csv')

# Try uploading dataset using different ways (explore...)
```

2) Inspect/Know the data (EDA)

```
#dimensions of the dataframe
df.shape

#df.shape[0]
#df.shape[1]
```

```
#show the data types of all attributes(columns)
df.dtypes
```

```
#show the first X rows in the dataframe, no value defaults to 5
df.head()

#df.head(1)
```

```
#show the last X rows in the dataframe, no value defaults to 5
df.tail()

#df.tail(1)
```

```
#get basic stats on any numeric features
df.describe()
```

```
#get basic stats on any numeric features
df.describe()

#df.describe(include='object')
#df.describe(include='all')
```

```
#Summary of the dataframe
df.info()
```

3) Handling missing values

```
# Find/check missing values
print(df.isnull().sum())

#missingValueCount=df.isnull().sum()
#missingValueCount[0:10]
```

```
# Drop rows with missing values
df_droppedRows = df.dropna()

#df_droppedRows
#df_droppedRows.shape[0]
```

```
# Drop columns with missing values
df_droppedColumns = df.dropna(axis=1)

#df_droppedColumns
#df_droppedColumns.shape[1]
```

```
# Drop columns with more than 50% missing values
df_dropped50Percent=df.dropna(axis=1, thresh=int(0.5 * len(df)), inplace=True)

#df_dropped50Percent
```

```
# Drop columns with more than 50% missing values
df_dropped50Percent=df.dropna(axis=1, thresh=int(0.5 * len(df)), inplace=True)

#df_dropped50Percent


# Other methods like Propogation(Backward/Forward filling)


# Filling in missing values
df_filled=df.fillna(0)

#df_filled
#print(df_filled.isnull().sum())


# Fill missing values with the mean(for Numeric cols.) or mode(for Categorical cols.) of each column
for column in df.columns:
    if df[column].dtype == 'object':
        df[column].fillna(df[column].mode()[0], inplace=True)
    else:
        df[column].fillna(df[column].mean(), inplace=True)

#df

#verify if missing values are handled
#print(df.isnull().sum())
```

4) Handling outliers

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

#Seaborn is a library for making statistical graphics in Python.
#It builds on top of matplotlib and integrates closely with pandas data structures.

# Visualize outliers using boxplots for a numerical column
```

4) Handling outliers

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

#Seaborn is a library for making statistical graphics in Python.
#It builds on top of matplotlib and integrates closely with pandas data structures.

# Visualize outliers using boxplots for a numerical column
sns.boxplot(x=df['age'])
plt.show()


#1) Central Box: represents the interquartile range (IQR), i.e., the range between the
#   first quartile (Q1) and the third quartile (Q3).

#2) Line Inside the Box: represents the median (Q2) of the data.

#3) Whiskers: The lines (whiskers) extending from the box represent the range of data
#   within 1.5 times the IQR from the lower and upper quartiles.

#4) Outliers: Data points that are plotted as individual dots outside the whiskers are considered outliers.


#Remove Outliers Using IQR:

# Calculate the Interquartile Range (IQR)
Q1 = df['age'].quantile(0.25)
Q3 = df['age'].quantile(0.75)
IQR = Q3 - Q1

# Filter out rows with outliers
df = df[~((df['age'] < (Q1 - 1.5 * IQR)) | (df['age'] > (Q3 + 1.5 * IQR)))]

#df
```

5) Handling inconsistent values

5) Handling inconsistent values

Inconsistent values, especially in categorical columns, need to be cleaned up. For example, entries like 'Normal' and 'normal' should be standardized.

```
# Convert all categorical values to lowercase for standardization
df['rbc'] = df['rbc'].str.lower()

# Replace inconsistent values
df['rbc'].replace({'n': 'normal', 'ab': 'abnormal'}, inplace=True)

#df
```

6) Validation Checks Specify validation rules to ensure the data is clean and consistent.

```
# Validation check: Age should be between 0 and 100
invalid_age = df[(df['age'] < 0) | (df['age'] > 100)]
if not invalid_age.empty:
    print("Invalid age entries found:\n", invalid_age)


# Validation check for blood pressure values
invalid_bp = df[(df['bp'] < 0) | (df['bp'] > 300)]
if not invalid_bp.empty:
    print("Invalid blood pressure entries found:\n", invalid_bp)
```

7) Final Review and Save the cleaned Dataset

```
# Review the cleaned dataset
print(df.info())

# Save the cleaned dataset
df.to_csv("cleaned_chronic_kidney_disease.csv", index=False)
```