Ex. No 2a: Implement various missing handling mechanisms

Aim:
To write a Python program for handling missing data using mean, median, and most frequent imputation.

Algorithm:

1. **Start**
2. **Import Libraries**
   - `pandas` as `pd`
   - `numpy` as `np`
   - `SimpleImputer` from `sklearn.impute`
3. **Create or Load Dataset**
   - Input data manually or read using `pd.read_csv()`
   - Ensure it contains some missing values (`NaN`)
4. **Display Original Data**
5. **Check and Print Missing Values (Before Imputation)**
   - Use `DataFrame.isnull().sum()` to count missing values column-wise
6. **Apply Mean Imputation (for numeric columns)**
   - Create `SimpleImputer(strategy="mean")`
   - Apply to numeric columns
   - Store results in new columns (e.g., `Age_mean`, `Salary_mean`)
7. **Apply Median Imputation (for numeric columns)**

   - Create `SimpleImputer(strategy="median")`
   - Apply to numeric columns
   - Store results in new columns
8. **Apply Most Frequent Imputation (for any column)**
   - Create `SimpleImputer(strategy="most_frequent")`
   - Apply to all columns
   - Store in new columns (e.g., `Department_mode`)
9. **Check and Print Missing Values (After Imputation)**
10. **Export Final Data to CSV**
    - Use `DataFrame.to_csv("imputed_data.csv")`
11. **End**

Program:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

# Step 1: Create a sample dataset with missing values
data = {
    'Age': [25, np.nan, 28, 35, np.nan, 40],
    'Salary': [50000, 54000, np.nan, 58000, 60000, np.nan],
    'Department': ['HR', 'HR', 'IT', np.nan, 'Finance', 'Finance']
}

df = pd.DataFrame(data)
print("Original Data:\n", df)

# Step 2: Show missing values before imputation
print("\nMissing Values (Before Imputation):\n", df.isnull().sum())

# Step 3: Mean imputation for numeric columns
```

```
mean_imputer = SimpleImputer(strategy='mean')
df['Age_mean'] = mean_imputer.fit_transform(df[['Age']])
df['Salary_mean'] = mean_imputer.fit_transform(df[['Salary']])

# Step 4: Median imputation for numeric columns
median_imputer = SimpleImputer(strategy='median')
df['Age_median'] = median_imputer.fit_transform(df[['Age']])
df['Salary_median'] = median_imputer.fit_transform(df[['Salary']])

# Step 5: Most frequent imputation for categorical columns
freq_imputer = SimpleImputer(strategy='most_frequent')
df['Department_mode'] = freq_imputer.fit_transform(df[['Department']])

# Step 6: Show missing values after imputation (in new columns, original still has
NaNs)
print("\n Missing Values (After Imputation on New Columns):")
print(df[['Age_mean', 'Salary_mean', 'Age_median', 'Salary_median',
'Department_mode']].isnull().sum())

# Step 7: Export final data to CSV
df.to_csv("imputed_data.csv", index=False)
print("\n  Final Data exported to 'imputed_data.csv'")

#  Show the updated DataFrame
print("\n Final DataFrame:\n", df)
```

Sample Output:

```
    Age   Salary Department
0  25.0  50000.0         HR
1   NaN  54000.0         HR
2  28.0      NaN         IT
3  35.0  58000.0        NaN
4   NaN  60000.0    Finance
5  40.0      NaN    Finance


Missing Values (Before Imputation):
Age          2
Salary       2
Department   1
dtype: int64

Missing Values (After Imputation on New Columns):
```

```
Age_mean          0
Salary_mean       0
Age_median        0
Salary_median     0
Department_mode   0
dtype: int64
```

Final DataFrame:

	Age	Salary	Department	Age_mean	Salary_mean	Age_median	Salary_median	Department_mode
0	25.0	50000.0	HR	25.0	50000.0	25.0	50000.0	HR
1	NaN	54000.0	HR	32.0	54000.0	31.5	54000.0	HR
2	28.0	NaN	IT	28.0	55500.0	28.0	56000.0	IT
3	35.0	58000.0	NaN	35.0	58000.0	35.0	58000.0	HR
4	NaN	60000.0	Finance	32.0	60000.0	31.5	60000.0	Finance
5	40.0	NaN	Finance	40.0	55500.0	40.0	56000.0	Finance

Exercise No: 2b Implement various noisy handling mechanisms

Aim:
To write python program for noisy mechanism

Algorithm:

1. Import necessary libraries

- numpy for numerical operations
- `pandas` for moving average
- `scipy.signal.medfilt` for median filtering
- `sklearn.linear_model` for robust regression
- `matplotlib.pyplot` for visualization

2. Generate noisy data

- Create a sequence of x-values (e.g., using `linspace`)

- Compute corresponding y-values using a linear function (e.g., `y = 3x + 5`)
- Add random noise (normally distributed) to y-values
- Inject artificial outliers (e.g., every 10th point increased)

3. Define function: remove_outliers(data, threshold)

- Calculate mean and standard deviation of data
- Mark values as NaN if they deviate more than `threshold × std` from the mean

4. Define function: apply_median_filter(data, filter_size)

- Apply median filtering using a sliding window of size `filter_size`

5. Define function: apply_moving_average(data, window_size)

- Use rolling window to compute moving average of the data

6. Define function: perform_robust_regression(x, y)

- Use `RANSACRegressor` with a base `LinearRegression` model
- Fit model to `(x, y)` data
- Separate inliers (fit well) from outliers (don't fit well)

7. Define function: apply_kalman_filter(data, measurement_noise, process_noise)

- Initialize estimated v**alue and error**
- **For each time step:**
    - **Predict next value using previous state**
    - **Compute Kalman gain**
    - **Update estimate with current measurement**

- **Update error covariance**

8. Apply all filters to noisy data

- **Call `remove_outliers(y, threshold)`**
- **Call `apply_median_filter(y, filter_size)`**
- **Call `apply_moving_average(y, window_size)`**
- **Call `perform_robust_regression(x, y)`**
- **Call `apply_kalman_filter(y, measurement_noise, process_noise)`**

9. Display results

- Print or plot the filtered data for comparison

Program:

```
import numpy as np
import pandas as pd
from scipy.signal import medfilt
from sklearn.linear_model import RANSACRegressor, LinearRegression
import matplotlib.pyplot as plt

# 1. Define Functions

def remove_outliers(data, threshold=2.0):
    mean = np.mean(data)
    std = np.std(data)
    filtered = [x if abs(x - mean) <= threshold * std else np.nan for x in data]
    return np.array(filtered)

def apply_median_filter(data, filter_size=3):
    return medfilt(data, kernel_size=filter_size)

def apply_moving_average(data, window_size=3):
    return pd.Series(data).rolling(window=window_size, min_periods=1,
center=True).mean().to_numpy()
```

```
def perform_robust_regression(x, y):
    x = x.reshape(-1, 1)
    model = RANSACRegressor(base_estimator=LinearRegression(), residual_threshold=10)
    model.fit(x, y)
    inlier_mask = model.inlier_mask_
    outlier_mask = ~inlier_mask
    return x[inlier_mask], y[inlier_mask], x[outlier_mask], y[outlier_mask]

def apply_kalman_filter(data, measurement_noise=1.0, process_noise=0.01):
    n = len(data)
    x_est = np.zeros(n)
    p = np.zeros(n)
    x_est[0] = data[0]
    p[0] = 1.0
    for k in range(1, n):
        x_pred = x_est[k-1]
        p_pred = p[k-1] + process_noise
        k_gain = p_pred / (p_pred + measurement_noise)
        x_est[k] = x_pred + k_gain * (data[k] - x_pred)
        p[k] = (1 - k_gain) * p_pred
    return x_est

# 2. Define Noisy Input Data (Simulated)
np.random.seed(42)
x = np.linspace(0, 10, 100)
true_y = 3 * x + 5
noise = np.random.normal(0, 5, size=x.shape)
y = true_y + noise
# Introduce outliers
y[::10] += 30

# 3a. Outlier Removal
filtered_outliers = remove_outliers(y, threshold=2.0)

# 3b. Median Filter
filtered_median = apply_median_filter(y, filter_size=5)

# 3c. Moving Average
smoothed_avg = apply_moving_average(y, window_size=5)

# 3d. Robust Regression
inlier_x, inlier_y, outlier_x, outlier_y = perform_robust_regression(x, y)

# 3e. Kalman Filter
```

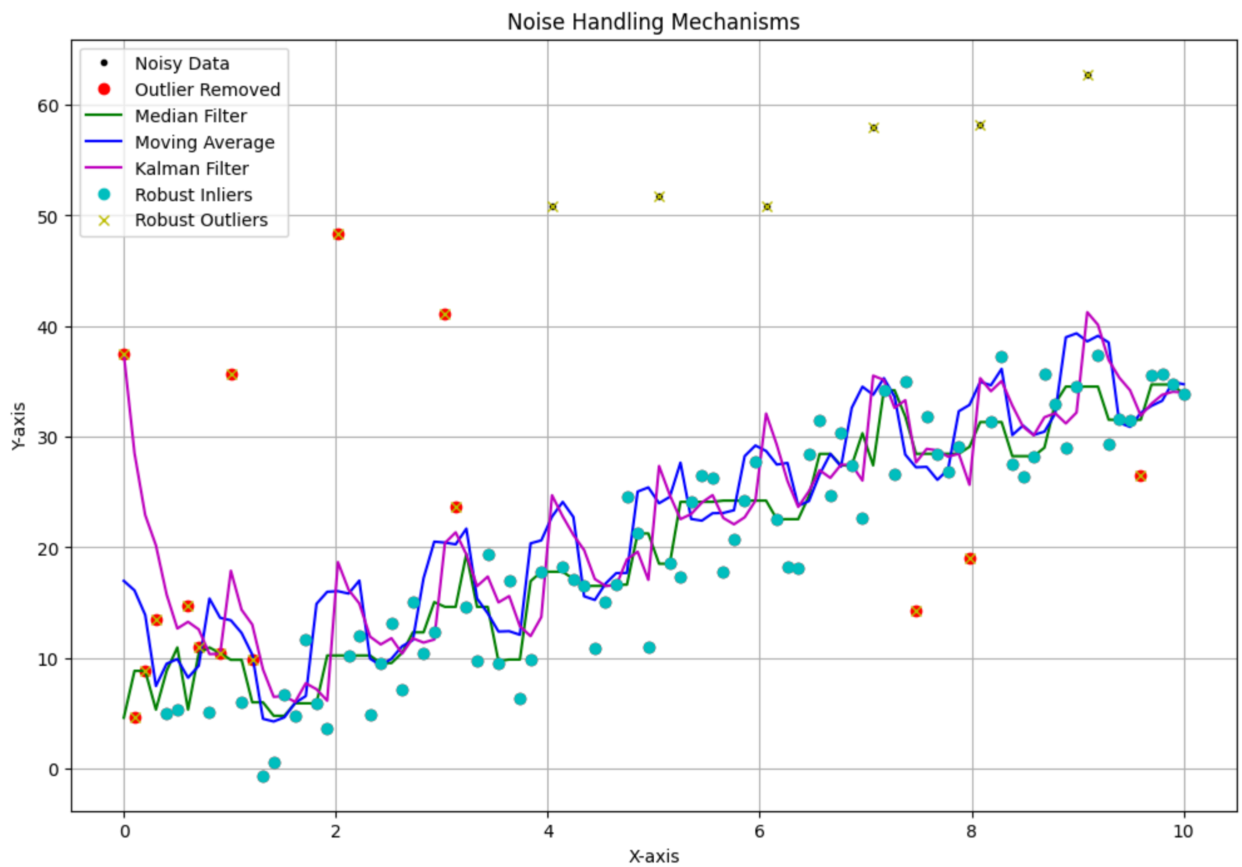
```
filtered_kalman = apply_kalman_filter(y, measurement_noise=4, process_noise=0.5)

# 4. Print summaries
print("Original Data with Noise (first 10):", y[:10])
print("Outlier Removed Data (first 10):", filtered_outliers[:10])
print("Median Filtered Data (first 10):", filtered_median[:10])
print("Moving Average Smoothed Data (first 10):", smoothed_avg[:10])
print("Kalman Filtered Data (first 10):", filtered_kalman[:10])
print(f"Robust Regression: {len(inlier_x)} inliers, {len(outlier_x)} outliers")

# I Plotting (for visualization)
plt.figure(figsize=(12, 8))
plt.plot(x, y, 'k.', label='Noisy Data')
plt.plot(x, filtered_outliers, 'ro', label='Outlier Removed')
plt.plot(x, filtered_median, 'g-', label='Median Filter')
plt.plot(x, smoothed_avg, 'b-', label='Moving Average')
plt.plot(x, filtered_kalman, 'm-', label='Kalman Filter')
plt.plot(inlier_x, inlier_y, 'co', label='Robust Inliers')
plt.plot(outlier_x, outlier_y, 'yx', label='Robust Outliers')
plt.legend()
plt.title("Noise Handling Mechanisms")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.show()
```

Sample Output:

```
Original Data with Noise (first 10): [37.48357077  4.6117088   8.8445033
13.52424019  5.04135434  5.34446673
 14.7142459  10.95838577  5.07687049 10.44007295]
Outlier Removed Data (first 10): [37.48357077  4.6117088   8.8445033
13.52424019  5.04135434  5.34446673
 14.7142459  10.95838577  5.07687049 10.44007295]
Median Filtered Data (first 10): [ 4.6117088   8.8445033   8.8445033
5.34446673  8.8445033  10.95838577
  5.34446673 10.44007295 10.95838577 10.44007295]
Moving Average Smoothed Data (first 10): [16.97992762 16.11600576
13.90107548  7.47325467  9.49376209  9.91653858
  8.22706465  9.30680837 15.38055793 13.63864567]
Kalman Filtered Data (first 10): [37.48357077 28.5185175  22.92022078
20.19017018 15.74303261 12.6748166
 13.27809024 12.59105476 10.36417929 10.38667771]
Robust Regression: 79 inliers, 21 outliers
```


Noise Handling Mechanisms
Noisy Data
Outlier Removed
Median Filter
Moving Average
Kalman Filter
Robust Inliers
Robust Outliers
Y-axis
X-axis
0
10
20
30
40
50
60
0
2
4
6
8
10