SCREEN SENSE – KID'S SCREENTIME VISUALIZATION

REPORT BY

PRABAKARAN N

1.OBJECTIVE

In This week 02, my main focus was on preprocessing the dataset and creating new features to make the data ready for analysis. I handled missing values and standardized inconsistent categories to ensure the data is clean and reliable. I also derived new columns such as age groups, screen time levels, device categories, and educational vs recreational ratios to help uncover meaningful patterns. Additionally, I formatted any date or time fields and saved the pre-processed dataset for future analysis. Overall, my goal was to prepare a well-structured dataset that is ready for deeper analysis and visualization in the upcoming weeks.

2.PREPROCESSING AND FEATURE ENGINEERING

2.1 Identifying and Handling Missing Data

Step 01:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df= pd.read_csv("Indian_Kids_Screen_Time.csv")
df.head()
```

	Age	Gender	Avg_Daily_Screen_Time_hr	Primary_Device	Exceeded_Recommended_Limit	Educational_to_Recreational_Ratio	Health_Impacts	Urban_or_Rural
0	14	Male	3.99	Smartphone	True	0.42	Poor Sleep, Eye Strain	Urban
1	11	Female	4.61	Laptop	True	0.30	Poor Sleep	Urban
2	18	Female	3.73	TV	True	0.32	Poor Sleep	Urban
3	15	Female	1.21	Laptop	False	0.39	NaN	Urban
4	12	Female	5.89	Smartphone	True	0.49	Poor Sleep, Anxiety	Urban

Output:

- To load the dataset and preview the first few rows to understand its structure and contents.

Step 02:

```
#missing values
print(df.isnull().sum())
```

Output:

- Most columns are complete with no missing values, making them ready for analysis.
- The Health Impacts column has 3218 missing values, indicating that many entries do not have reported health effects.

```
Age                                0
Gender                             0
Avg_Daily_Screen_Time_hr           0
Primary_Device                     0
Exceeded_Recommended_Limit         0
Educational_to_Recreational_Ratio  0
Health_Impacts                  3218
Urban_or_Rural                     0
dtype: int64
```

Step 03:

```
#unique values in Health Impacts
unique_values = df['Health_Impacts'].unique()
print(unique_values)
```

Output:

- To identify all distinct health issues reported in the Health Impacts column.
- The output lists all unique entries, like below

```
['Poor Sleep, Eye Strain' 'Poor Sleep' nan 'Poor Sleep, Anxiety'
 'Poor Sleep, Obesity Risk' 'Eye Strain' 'Obesity Risk' 'Anxiety'
 'Poor Sleep, Anxiety, Obesity Risk' 'Eye Strain, Obesity Risk'
 'Eye Strain, Anxiety, Obesity Risk' 'Anxiety, Obesity Risk'
 'Poor Sleep, Eye Strain, Obesity Risk' 'Poor Sleep, Eye Strain, Anxiety'
 'Poor Sleep, Eye Strain, Anxiety, Obesity Risk' 'Eye Strain, Anxiety']
```

Step 04:

```
# Step 1: Mapping from screen time : most common health impact
impact_mapping = df.groupby('Avg_Daily_Screen_Time_hr')['Health_Impacts'].agg(
    lambda x: x.mode().iloc[0] if not x.mode().empty else np.nan
)

# Step 2: Find most common overall health impact (as fallback)
most_common_impact = df['Health_Impacts'].mode().iloc[0]

# Step 3: Function to fill missing values
def fill_health_impact(row):
    if pd.isnull(row['Health_Impacts']):
        mapped_value = impact_mapping.get(row['Avg_Daily_Screen_Time_hr'], np.nan)
        if pd.isnull(mapped_value):
            return most_common_impact  # fallback
        else:
            return mapped_value
    else:
        return row['Health_Impacts']

# Step 4: Apply to DataFrame
df['Health_Impacts'] = df.apply(fill_health_impact, axis=1)

df.head()
```

Output:

- To fill missing values in the Health Impacts column by mapping each child's screen time to the most common health impact for that average screen time column.
- If no specific mapping exists, a fallback to the most common overall health impact is applied.

	Age	Gender	Avg_Daily_Screen_Time_hr	Primary_Device	Exceeded_Recommended_Limit	Educational_to_Recreational_Ratio	Health_Impacts	Urban_or_Rural
0	14	Male	3.99	Smartphone	True	0.42	Poor Sleep, Eye Strain	Urban
1	11	Female	4.61	Laptop	True	0.30	Poor Sleep	Urban
2	18	Female	3.73	TV	True	0.32	Poor Sleep	Urban
3	15	Female	1.21	Laptop	False	0.39	Poor Sleep	Urban
4	12	Female	5.89	Smartphone	True	0.49	Poor Sleep, Anxiety	Urban

Step 05:

```
print(df.isnull().sum())
```

```
Age                                0
Gender                             0
Avg_Daily_Screen_Time_hr           0
Primary_Device                     0
Exceeded_Recommended_Limit         0
Educational_to_Recreational_Ratio  0
Health_Impacts                     0
Urban_or_Rural                     0
dtype: int64
```

o

Output:

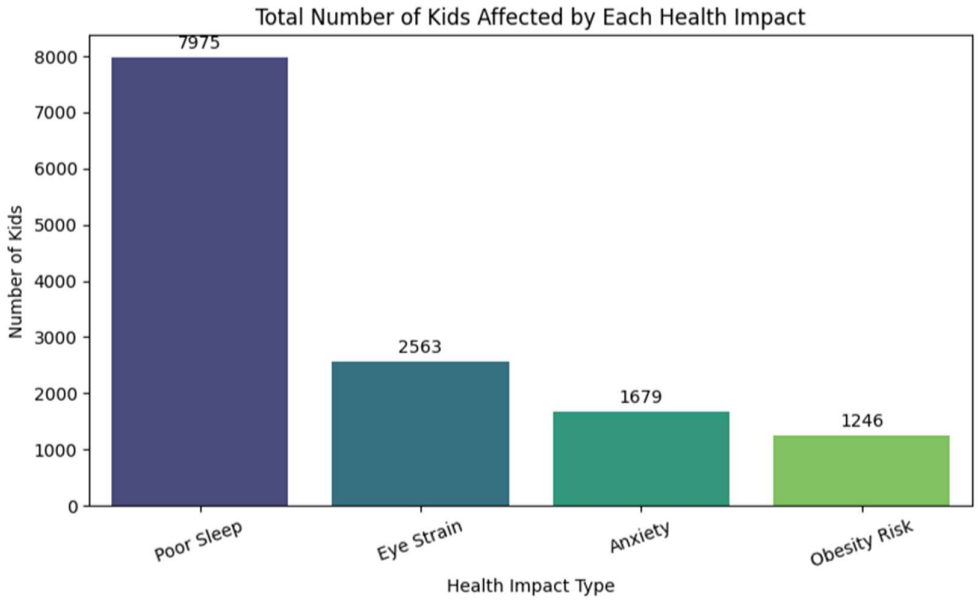
- To verify that all missing values in the Health Impacts column have been successfully filled.

Step 06:

```
# 1. Split the 'Health_Impacts' column by comma and explode
df_exploded = df.copy()
df_exploded['Health_Impacts'] = df_exploded['Health_Impacts'].fillna("")  # Handle NaN
df_exploded = df_exploded.assign(Health_Impacts=df_exploded['Health_Impacts'].str.split(',')).explode('Health_Impacts')

# 2. Remove extra spaces from impact names
df_exploded['Health_Impacts'] = df_exploded['Health_Impacts'].str.strip()

# 3. Count occurrences
impact_counts = df_exploded['Health_Impacts'][df_exploded['Health_Impacts'] != ""].value_counts()

impact_counts_df = impact_counts.reset_index()
impact_counts_df.columns = ['Health_Impact', 'Count']

# 4. Plot
plt.figure(figsize=(8,5))
ax = sns.barplot(data=impact_counts_df, x='Health_Impact', y='Count', palette='viridis')

# Add labels on each bar
for container in ax.containers:
    ax.bar_label(container, fmt='%d', label_type='edge', padding=3, fontsize=10)

plt.title("Total Number of Kids Affected by Each Health Impact")
plt.xlabel("Health Impact Type")
plt.ylabel("Number of Kids")
plt.xticks(rotation=20)
plt.tight_layout()
plt.show()
```

Output:

- To analyze the frequency of each health impact by splitting multiple impacts listed in one row and counting them individually.
- This helps understand which health issues are most common among children based on their screen time.



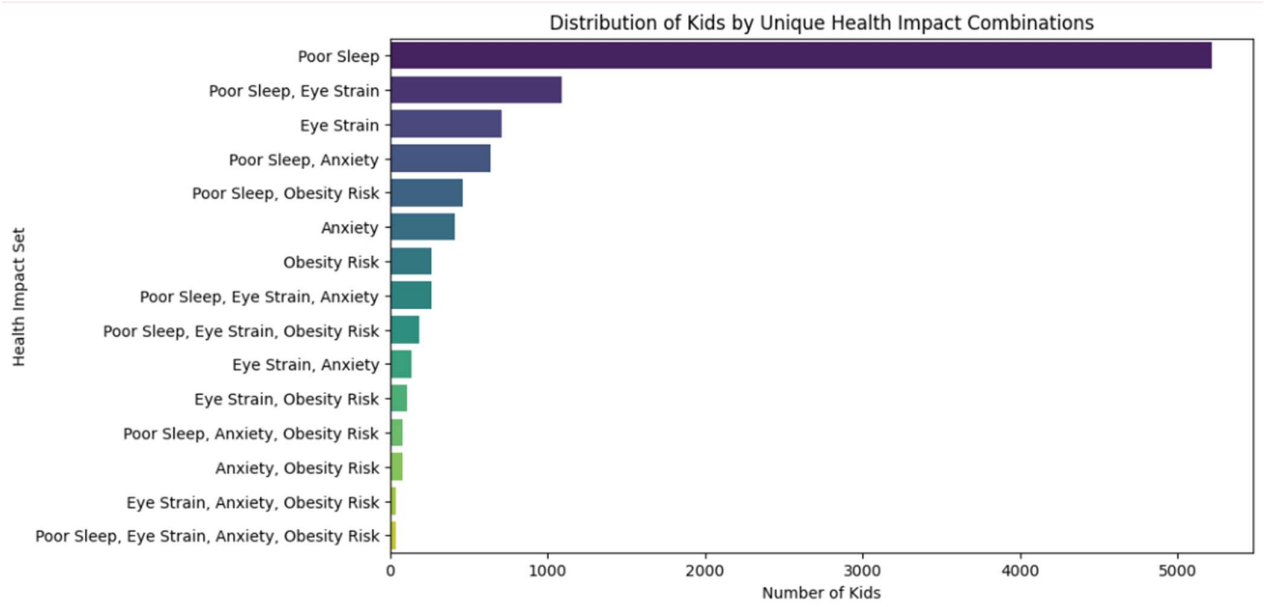
Step 07:

```
# Count each unique combination of Health_Impacts
combo_counts = df['Health_Impacts'].value_counts().reset_index()
combo_counts.columns = ['Health_Impact_Set', 'Count']
```

```
plt.figure(figsize=(10,6))
sns.barplot(data=combo_counts, y='Health_Impact_Set', x='Count', palette="viridis")
plt.xlabel("Number of Kids")
plt.ylabel("Health Impact Set")
plt.title("Distribution of Kids by Unique Health Impact Combinations")
plt.show()
```

Output:

- To analyze the frequency of unique combinations of health impacts reported by children, instead of individual impacts.
- This helps identify which specific sets of health issues occur together most often.



2.2 Harmonizing Categorical Values

Step 01:

```
#Category Standardization
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical Columns:", list(categorical_cols))
```

Output:

- To identify all categorical columns in the dataset that may require standardization
- The categorical columns identified are: Gender, Primary Device, Health Impacts, and Urban or Rural.

```
Categorical Columns: ['Gender', 'Primary_Device', 'Health_Impacts', 'Urban_or_Rural', 'Age_Group', 'Device_Category', 'Screen_Time_Level']
```

Step 02:

```
for col in categorical_cols:
    df[col] = df[col].astype(str).str.strip().str.title()
# Standardize Gender
df['Gender'] = df['Gender'].replace({
    'M': 'Male',
    'F': 'Female',
    'Boy': 'Male',
    'Girl': 'Female'
})

# Standardize Location
df['Urban_or_Rural'] = df['Urban_or_Rural'].replace({
    'City': 'Urban',
    'Town': 'Urban',
    'Village': 'Rural',
    'Countryside': 'Rural'
})

# Standardize Device names
df['Primary_Device'] = df['Primary_Device'].replace({
    'Mobile': 'Smartphone',
    'Phone': 'Smartphone',
    'Tab': 'Tablet',
    'Computer': 'Laptop'
})

# Standardize Exceeded_Recommended_Limit
df['Exceeded_Recommended_Limit'] = df['Exceeded_Recommended_Limit'].replace({
    'True': True,
    'False': False,
    'Yes': True,
    'No': False
})
```

Output:

- To standardize categorical columns and ensure consistency across the dataset, which is essential for accurate analysis and visualization.

Step 03

```
# Specify the columns you want to check
columns_to_check = ['Age','Gender', 'Primary_Device', 'Health_Impacts', 'Urban_or_Rural']

# Loop through and print unique values
for col in columns_to_check:
    print(f"\nColumn: {col}")
    print(df[col].unique())
```

Output:

- To examine the unique values in selected columns and verify that categorical standardization has been applied correctly.
- The code displays unique values for Age, Gender, Primary Device, Health Impacts, and Urban or Rural.

```
Column: Age
[14 11 18 15 12 17 10 13  9  8 16]

Column: Gender
['Male' 'Female']

Column: Primary_Device
['Smartphone' 'Laptop' 'Tv' 'Tablet']

Column: Health_Impacts
['Poor Sleep, Eye Strain' 'Poor Sleep' 'Poor Sleep, Anxiety'
 'Poor Sleep, Obesity Risk' 'Eye Strain' 'Obesity Risk' 'Anxiety'
 'Poor Sleep, Anxiety, Obesity Risk' 'Eye Strain, Obesity Risk'
 'Eye Strain, Anxiety, Obesity Risk' 'Anxiety, Obesity Risk'
 'Poor Sleep, Eye Strain, Obesity Risk' 'Poor Sleep, Eye Strain, Anxiety'
 'Poor Sleep, Eye Strain, Anxiety, Obesity Risk' 'Eye Strain, Anxiety']

Column: Urban_or_Rural
['Urban' 'Rural']
```

2.3 Feature Engineering: Derived Columns for Insights

Step 1:

```
# 1. Age Group
def age_group(age):
    if age <= 5:
        return "Toddler"
    elif age <= 10:
        return "Child"
    elif age <= 14:
        return "Early Teen"
    else:
        return "Teen"

df['Age_Group'] = df['Age'].apply(age_group)

# 2. Device Category
def device_category(device):
    if device in ['Smartphone', 'Tablet']:
        return "Mobile"
    elif device == 'Laptop':
        return "Laptop"
    else:
        return "TV"

df['Device_Category'] = df['Primary_Device'].apply(device_category)
```

```
# 3. Screen Time Level
def screen_time_level(hours):
    if hours < 2:
        return "Low"
    elif hours < 4:
        return "Medium"
    elif hours < 6:
        return "High"
    else:
        return "Very High"

df['Screen_Time_Level'] = df['Avg_Daily_Screen_Time_hr'].apply(screen_time_level)

# 4. Health Impact Count
df['Health_Impact_Count'] = df['Health_Impacts'].fillna("").apply(lambda x: len(x.split(',')) if x else 0)

# 5. Educational vs Recreational Dominance
def edu_vs_rec(ratio):
    if ratio > 0.5:
        return "Education Dominant"
    elif ratio >= 0.5:
        return "Balanced"
    else:
        return "Recreational Dominant"
df['Edu_vs_Rec'] = df['Educational_to_Recreational_Ratio'].apply(edu_vs_rec)

df.head(10)
```

Output:

- To create new derived columns that provide more meaningful insights and enable easier analysis.
- These features include age groups, device categories, screen time levels, health impact counts, and educational vs recreational dominance.
- Age Group: Categorizes children into Toddler, Child, Early Teen, and Teen based on age.
- Device Category: Groups devices into Mobile, Laptop, or TV.
- Screen Time Level: Classifies average daily screen time as Low, Medium, High, or Very High.
- Health Impact Count: Counts the number of health impacts reported per child.
- Edu vs Rec: Indicates whether a child's screen time is Education Dominant, Balanced, or Recreational Dominant.
- The first 10 rows confirm that all derived features are correctly calculated and added to the dataset, ready for further analysis or visualization.

Age_Group	Device_Category	Screen_Time_Level	Health_Impact_Count	Edu_vs_Rec
Early Teen	Mobile	Medium	2	Recreational Dominant
Early Teen	Laptop	High	1	Recreational Dominant
Teen	TV	Medium	1	Recreational Dominant
Teen	Laptop	Low	1	Recreational Dominant
Early Teen	Mobile	High	2	Recreational Dominant
Early Teen	Mobile	High	1	Recreational Dominant
Teen	TV	Medium	1	Recreational Dominant
Child	TV	Medium	1	Education Dominant
Early Teen	Laptop	High	2	Recreational Dominant
Teen	Mobile	Medium	2	Recreational Dominant

2.4 Exporting Preprocessed Data:

```
import os
os.getcwd()
os.makedirs('C:\\Users\\Prabakaran\\DEMO',exist_ok=True)
os.makedirs('C:/MyDatasets',exist_ok=True)
df.to_csv('C:/MyDatasets/Preprocessed_Indian_Kids_Screen_Time.csv', index=False)
```

Output:

- To save the pre-processed dataset so it can be reused for analysis and visualization in future steps without repeating preprocessing.

3. CONCLUSION

In Week 2, I focused on preprocessing the dataset and performing feature engineering to make the data clean, consistent, and analysis-ready. I successfully handled missing values, standardized categorical fields, and created new derived columns such as age groups, device categories, screen time levels, health impact counts, and educational vs recreational ratios. These steps ensure that the dataset is structured, meaningful, and ready for further exploration and visualization.

By saving the pre-processed dataset and documenting the logic behind each transformation, I have created a reusable and reliable dataset that will support the upcoming analysis in future weeks. Overall, this week laid a strong foundation for deriving actionable insights from the data.