business-case-aerofit-treadmill

May 26, 2025

Business Case: Aerofit - Descriptive Statistics & Probability - SUKANYA DEVI B

Importing Python Libraries:

numpy (np):

Purpose: Numerical operations, arrays, math functions.

Used for fast calculations and working with numeric data.

pandas (pd):

Purpose: Data analysis and manipulation.

Makes it easy to read, filter, group, and clean datasets (like CSV files).

matplotlib.pyplot (plt):

Purpose: Data visualization.

Used to create plots and graphs like line charts, bar charts, etc.

seaborn (sns):

Purpose: Statistical data visualization (built on top of matplotlib).

Used to make attractive and informative plots like heatmaps, pair plots, etc., with less code.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Downloading the csv file:

[2]:
```
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/125/
↪original/aerofit_treadmill.csv?1639992749
```

```
Downloading…
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/125/ori
ginal/aerofit_treadmill.csv?1639992749
To: /content/aerofit_treadmill.csv?1639992749
100% 7.28k/7.28k [00:00<00:00, 37.4MB/s]
```

Reading a CSV file into pandas DataFrame:

[3]:
```python
df= pd.read_csv('aerofit_treadmill.csv?1639992749')
```

This line loads the aerofit_treadmill dataset into memory as a table (df) so we can explore, analyze, and visualize it using pandas.

Checking the data types of each column:

[4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Product        180 non-null    object
 1   Age            180 non-null    int64
 2   Gender         180 non-null    object
 3   Education      180 non-null    int64
 4   MaritalStatus  180 non-null    object
 5   Usage          180 non-null    int64
 6   Fitness        180 non-null    int64
 7   Income         180 non-null    int64
 8   Miles          180 non-null    int64
dtypes: int64(6), object(3)
memory usage: 12.8+ KB
```

This method provides a summary of the dataset, including:

1. Total number of rows
2. Column names
3. Count of non-null values (helpful for checking missing data)
4. Data types of each column
5. Memory usage

Statistical Summary of Numerical Features:

[5]:
```python
df.describe()
```

[5]:
```
              Age   Education       Usage     Fitness         Income  \
count  180.000000  180.000000  180.000000  180.000000     180.000000
mean    28.788889   15.572222    3.455556    3.311111   53719.577778
std      6.943498    1.617055    1.084797    0.958869   16506.684226
min     18.000000   12.000000    2.000000    1.000000   29562.000000
25%     24.000000   14.000000    3.000000    3.000000   44058.750000
50%     26.000000   16.000000    3.000000    3.000000   50596.500000
75%     33.000000   16.000000    4.000000    4.000000   58668.000000
max     50.000000   21.000000    7.000000    5.000000  104581.000000
```

```
            Miles
count  180.000000
mean   103.194444
std     51.863605
min     21.000000
25%     66.000000
50%     94.000000
75%    114.750000
max    360.000000
```

'describe' is used to generate summary statistics for the numerical columns in a DataFrame.

Checking column names in the dataset:

[6]:
```
df.columns
```

[6]:
```
Index(['Product', 'Age', 'Gender', 'Education', 'MaritalStatus', 'Usage',
       'Fitness', 'Income', 'Miles'],
      dtype='object')
```

'columns' is used to see what variables (columns) are available in our dataset.

Useful when we are exploring the data or writing code to reference specific columns.

Checking the shape of dataset:

[7]:
```
df.shape
```

[7]:
```
(180, 9)
```

'shape' is used to get the dimensions of the DataFrame df.

It returns a tuple: (number_of_rows, number_of_columns)

Non-Graphical Analysis:

Checking unique values:

[8]:
```
df.nunique()
```

[8]:
```
Product           3
Age              32
Gender            2
Education         8
MaritalStatus     2
Usage             6
Fitness           5
Income           62
Miles            37
dtype: int64
```

'nunique' is used to count the number of unique (distinct) values in each column of the DataFrame df.

Purpose:

To understand data variability.

To identify categorical columns (few unique values).

To catch unexpected values or duplicates.

Checking duplicates:

[9]:
```
duplicate=df.duplicated().value_counts()
print(duplicate)
```

```
False    180
Name: count, dtype: int64
```

df.duplicated()

Returns a Boolean Series where each row is marked as:

True if it is a duplicate of a previous row

False if it is unique (the first occurrence)

.value_counts()

Counts how many True and False values are there, i.e., how many duplicates and how many unique rows.

print(duplicate)

Displays the counts of duplicate vs. unique rows.

Checking missing values:

[10]:
```
df.isnull().sum()
```

[10]:
```
Product          0
Age              0
Gender           0
Education        0
MaritalStatus    0
Usage            0
Fitness          0
Income           0
Miles            0
dtype: int64
```

df.isnull()

Returns a DataFrame of the same shape as df with True where values are missing (NaN) and False elsewhere.

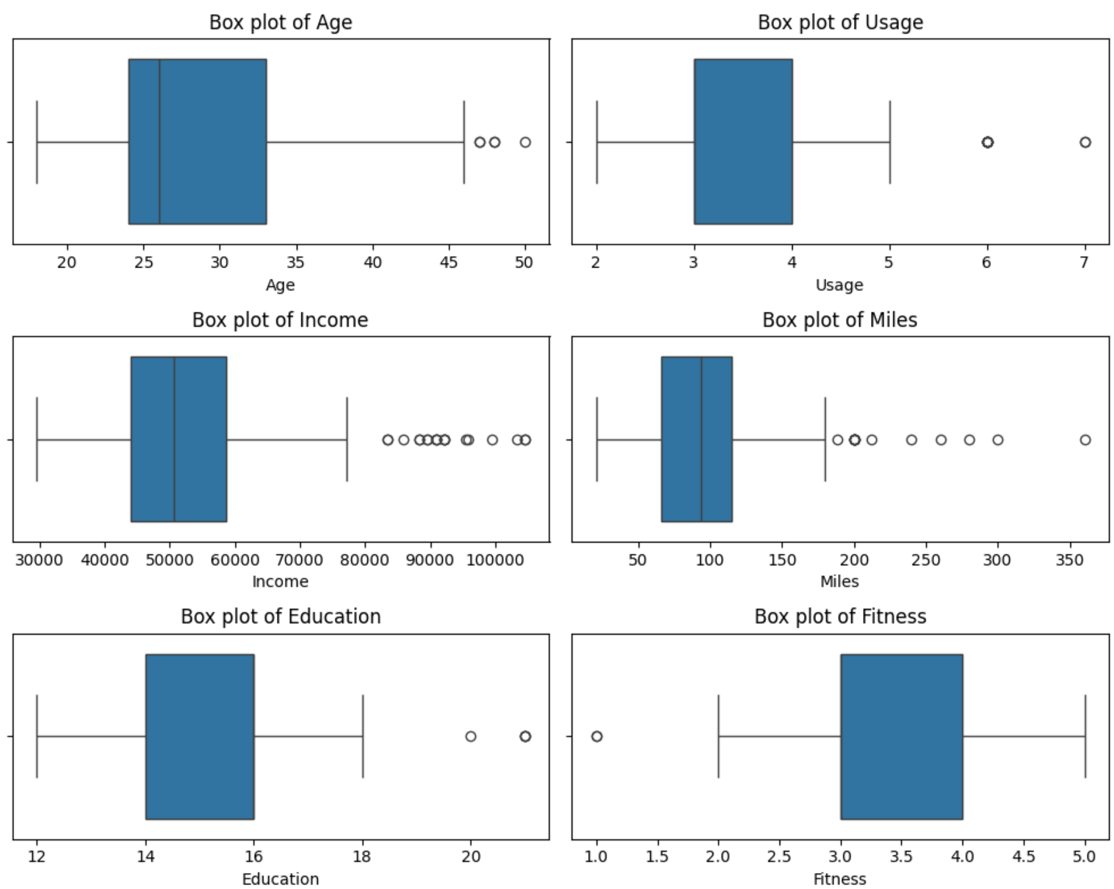
.sum()

Adds up the True values column-wise (treating True as 1), giving the count of missing values per column

Visual Analysis(Univariate & Bivariate):

Visualizing the distribution and detecting outliers in the continuous variables of our dataset using box plots:

[11]:
```
# Identify continuous variables (adjust if necessary based on data␣
↪characteristics)
continuous_vars = ['Age', 'Usage', 'Income', 'Miles', 'Education', 'Fitness']

# Plot box plots for continuous variables
plt.figure(figsize=(10, 8)) # Adjusted figure size for more plots
for i, var in enumerate(continuous_vars):
    plt.subplot(3, 2, i + 1) # Adjusted subplot grid for more plots
    sns.boxplot(x=df[var])
    plt.title(f'Box plot of {var}')
plt.tight_layout()
plt.show()
```



continuous_vars = ['Age', 'Usage', 'Income', 'Miles', 'Education', 'Fitness']

Defines which columns are continuous (numeric) variables you want to analyze.

For each variable, a box plot is drawn:

Box shows the interquartile range (IQR) — middle 50% of data.

Line inside the box is the median.

Whiskers show the range (excluding outliers).

Dots are outliers (values far from the rest).

Why Box Plots Are Useful Here:

Detect outliers (unusually high/low values).

Understand data distribution and skewness.

Compare spread across different variables.

Helps in data cleaning (e.g., handling extreme values).

Detecting and Handling outliers in numerical columns of your dataset using the Interquartile Range (IQR) method:

[12]:
```
#IQR
Q1 = df['Age'].quantile(0.25)
Q3 = df['Age'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

temp = df[(df['Age'] < lower) | (df['Age'] > upper)]
print('Percentage of Outliers are:' , len(temp)/180*100, '\n\n')
temp

df['Age'] = np.clip(df['Age'], lower, upper)
df['Age']
```

```
Percentage of Outliers are: 2.7777777777777777
```

[12]:
```
0      18.0
1      19.0
2      19.0
3      19.0
4      20.0
```

```
         …
175    40.0
176    42.0
177    45.0
178    46.5
179    46.5
Name: Age, Length: 180, dtype: float64
```

[13]:
```
Q1 = df['Usage'].quantile(0.25)
Q3 = df['Usage'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

temp = df[(df['Usage'] < lower) | (df['Usage'] > upper)]
print('Percentage of Outliers are:' , len(temp)/180*100, '\n\n')
temp

df['Usage'] = np.clip(df['Usage'], lower, upper)
df['Usage']
```

```
Percentage of Outliers are: 5.0
```

[13]:
```
0      3.0
1      2.0
2      4.0
3      3.0
4      4.0
      …
175    5.5
176    5.0
177    5.0
178    4.0
179    4.0
Name: Usage, Length: 180, dtype: float64
```

[14]:
```
Q1 = df['Income'].quantile(0.25)
Q3 = df['Income'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR
```

```python
temp = df[(df['Income'] < lower) | (df['Income'] > upper)]
print('Percentage of Outliers are:' , len(temp)/180*100, '\n\n')
temp

df['Income'] = np.clip(df['Income'], lower, upper)
df['Income']
```

```
Percentage of Outliers are: 10.555555555555555
```

```
[14]: 0      29562.000
1      31836.000
2      30699.000
3      32973.000
4      35247.000
          …
175    80581.875
176    80581.875
177    80581.875
178    80581.875
179    80581.875
Name: Income, Length: 180, dtype: float64
```

```python
[15]: Q1 = df['Miles'].quantile(0.25)
Q3 = df['Miles'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

temp = df[(df['Miles'] < lower) | (df['Miles'] > upper)]
print('Percentage of Outliers are:' , len(temp)/180*100, '\n\n')
temp

df['Miles'] = np.clip(df['Miles'], lower, upper)
df['Miles']
```

```
Percentage of Outliers are: 7.222222222222221
```

```
[15]: 0      112.000
1       75.000
2       66.000
```

```
3       85.000
4       47.000
         …
175    187.875
176    187.875
177    160.000
178    120.000
179    180.000
Name: Miles, Length: 180, dtype: float64
```

[16]:
```
Q1 = df['Education'].quantile(0.25)
Q3 = df['Education'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

temp = df[(df['Education'] < lower) | (df['Education'] > upper)]
print('Percentage of Outliers are:' , len(temp)/180*100, '\n\n')
temp

df['Education'] = np.clip(df['Education'], lower, upper)
df['Education']
```

```
Percentage of Outliers are: 2.2222222222222223
```

[16]:
```
0      14
1      15
2      14
3      12
4      13
       ..
175    19
176    18
177    16
178    18
179    18
Name: Education, Length: 180, dtype: int64
```

[17]:
```
Q1 = df['Fitness'].quantile(0.25)
Q3 = df['Fitness'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
```

```
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

temp = df[(df['Fitness'] < lower) | (df['Fitness'] > upper)]
print('Percentage of Outliers are:' , len(temp)/180*100, '\n\n')
temp

df['Fitness'] = np.clip(df['Fitness'], lower, upper)
df['Fitness']
```

```
Percentage of Outliers are: 1.1111111111111112
```

[17]:
```
0      4.0
1      3.0
2      3.0
3      3.0
4      2.0
      …
175    5.0
176    4.0
177    5.0
178    5.0
179    5.0
Name: Fitness, Length: 180, dtype: float64
```

Why use IQR method and clipping for outliers?

Robust to skewed data:

Unlike methods based on mean and standard deviation, IQR focuses on the middle 50% of the data, making it less sensitive to extreme values or skewed distributions.

Simple and effective rule:

The 1.5 * IQR rule is a well-established, easy-to-understand standard to flag unusually high or low values.

Prevents influence of extreme values:

Outliers can disproportionately affect statistical summaries, correlations, and model training. Clipping keeps these values within a realistic range.

Keeps dataset size intact:

Instead of dropping rows with outliers (which might reduce data or bias the sample), clipping replaces outliers with boundary values, preserving all data points.

Improves model stability and performance:

Many machine learning algorithms (especially distance-based or regression models) are sensitive to outliers and can perform better when outliers are controlled.

Easier interpretation:

By limiting the range, summary statistics (mean, variance) better reflect the typical values in the dataset.

Prepares data for visualization:

Outliers can stretch axes in plots like boxplots or scatterplots, hiding the real data pattern. Handling them improves plot readability.

Detects data errors or anomalies:

Some outliers might be input errors or rare events worth investigating separately.

Checking the difference between Mean and Median:

[18]:
```python
mean_age = df['Age'].mean()
median_age = df['Age'].median()
diff_age = mean_age - median_age

print("Age:")
print(f"Mean: {mean_age:.2f}, Median: {median_age:.2f}")
print(f"Difference: {diff_age:.2f}\n")
```

```
Age:
Mean: 28.75, Median: 26.00
Difference: 2.75
```

[19]:
```python
mean_usage = df['Usage'].mean()
median_usage = df['Usage'].median()
diff_usage = mean_usage - median_usage

print("Usage:")
print(f"Mean: {mean_usage:.2f}, Median: {median_usage:.2f}")
print(f"Difference: {diff_usage:.2f}\n")
```

```
Usage:
Mean: 3.42, Median: 3.00
Difference: 0.42
```

[20]:
```python
mean_fitness = df['Fitness'].mean()
median_fitness = df['Fitness'].median()
diff_fitness = mean_fitness - median_fitness

print("Fitness:")
print(f"Mean: {mean_fitness:.2f}, Median: {median_fitness:.2f}")
print(f"Difference: {diff_fitness:.2f}\n")
```

```
Fitness:
```

```
Mean: 3.32, Median: 3.00
Difference: 0.32
```

[21]:
```python
mean_income = df['Income'].mean()
median_income = df['Income'].median()
diff_income = mean_income - median_income

print("Income:")
print(f"Mean: {mean_income:.2f}, Median: {median_income:.2f}")
print(f"Difference: {diff_income:.2f}\n")
```

```
Income:
Mean: 52440.24, Median: 50596.50
Difference: 1843.74
```

[22]:
```python
mean_miles = df['Miles'].mean()
median_miles = df['Miles'].median()
diff_miles = mean_miles - median_miles

print("Miles:")
print(f"Mean: {mean_miles:.2f}, Median: {median_miles:.2f}")
print(f"Difference: {diff_miles:.2f}\n")
```

```
Miles:
Mean: 99.87, Median: 94.00
Difference: 5.87
```

[23]:
```python
mean_edu = df['Education'].mean()
median_edu = df['Education'].median()
diff_edu = mean_edu - median_edu

print("Education:")
print(f"Mean: {mean_edu:.2f}, Median: {median_edu:.2f}")
print(f"Difference: {diff_edu:.2f}\n")
```

```
Education:
Mean: 15.53, Median: 16.00
Difference: -0.47
```

Comparing the mean and median of numerical columns is a basic but powerful exploratory data analysis (EDA) technique.

Why to compare mean and median?

1. Detecting Skewness in Data

Mean is sensitive to extreme values (outliers).

Median is the middle value and is more robust to outliers.

If mean   median, your data is skewed.

Mean > Median → Right-skewed (positive skew) Example: Income (few people earning very high).

Mean < Median → Left-skewed (negative skew) Example: Age in a retirement home (more older people).

2. Understanding Data Distribution

Helps decide whether to use mean or median in summary statistics or modeling.

If data is skewed, median gives a better sense of "typical" value.

3. Feature Engineering Knowing skewness helps in:

Deciding transformations (e.g., log scale).

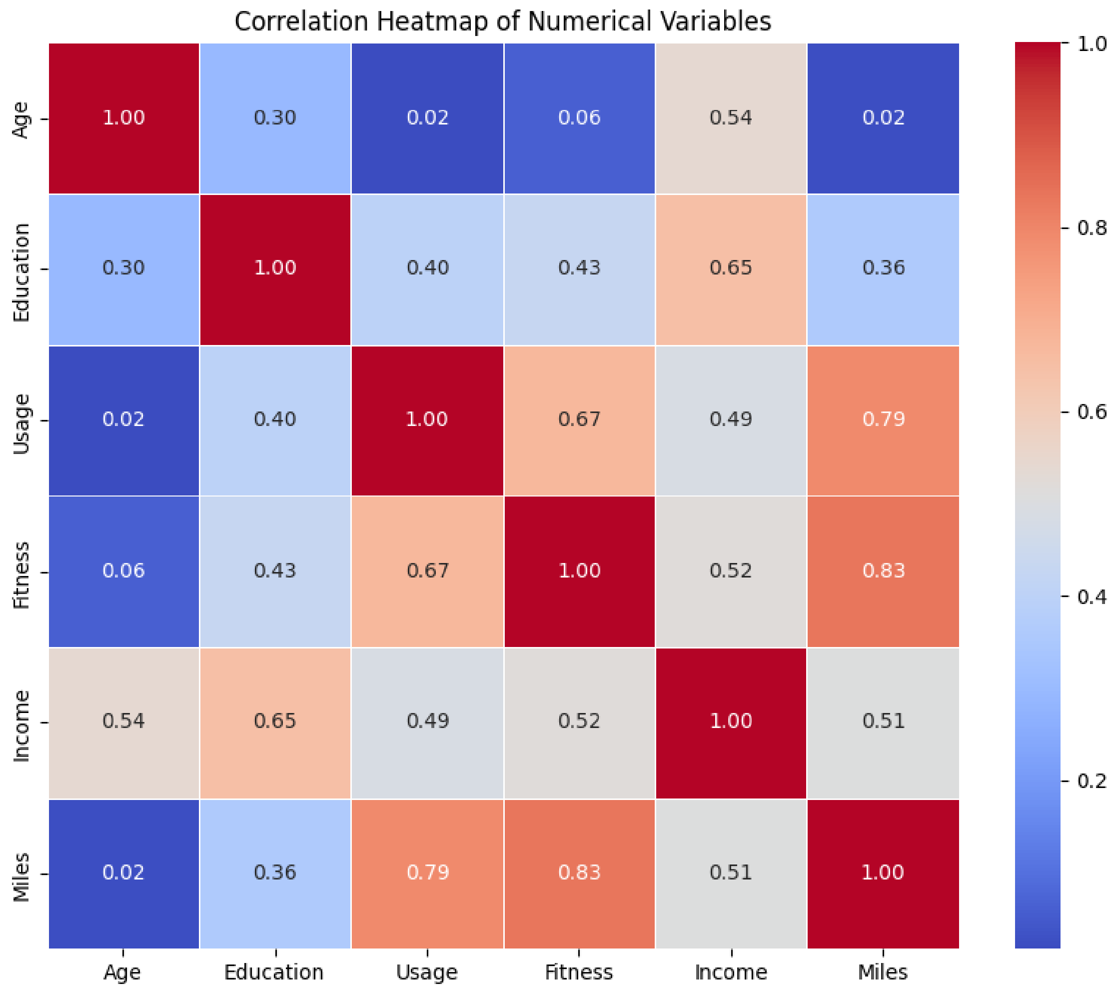
Choosing the right models (some assume normality).

4. Outlier Detection

A large difference between mean and median may indicate outliers affecting your data.

Computing and visualizing the correlation matrix of the numerical variables in our dataset using a heatmap:

[24]:
```
# Select only numerical columns for correlation
numerical_cols = df.select_dtypes(include=np.number).columns

# Calculate the correlation matrix
correlation_matrix = df[numerical_cols].corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
 ↪linewidths=.5)
plt.title('Correlation Heatmap of Numerical Variables')
plt.show()
```



This filters out only the numeric columns (e.g., Age, Income, Miles, etc.) from your DataFrame df.

Calculates the Pearson correlation coefficient between each pair of numerical variables.

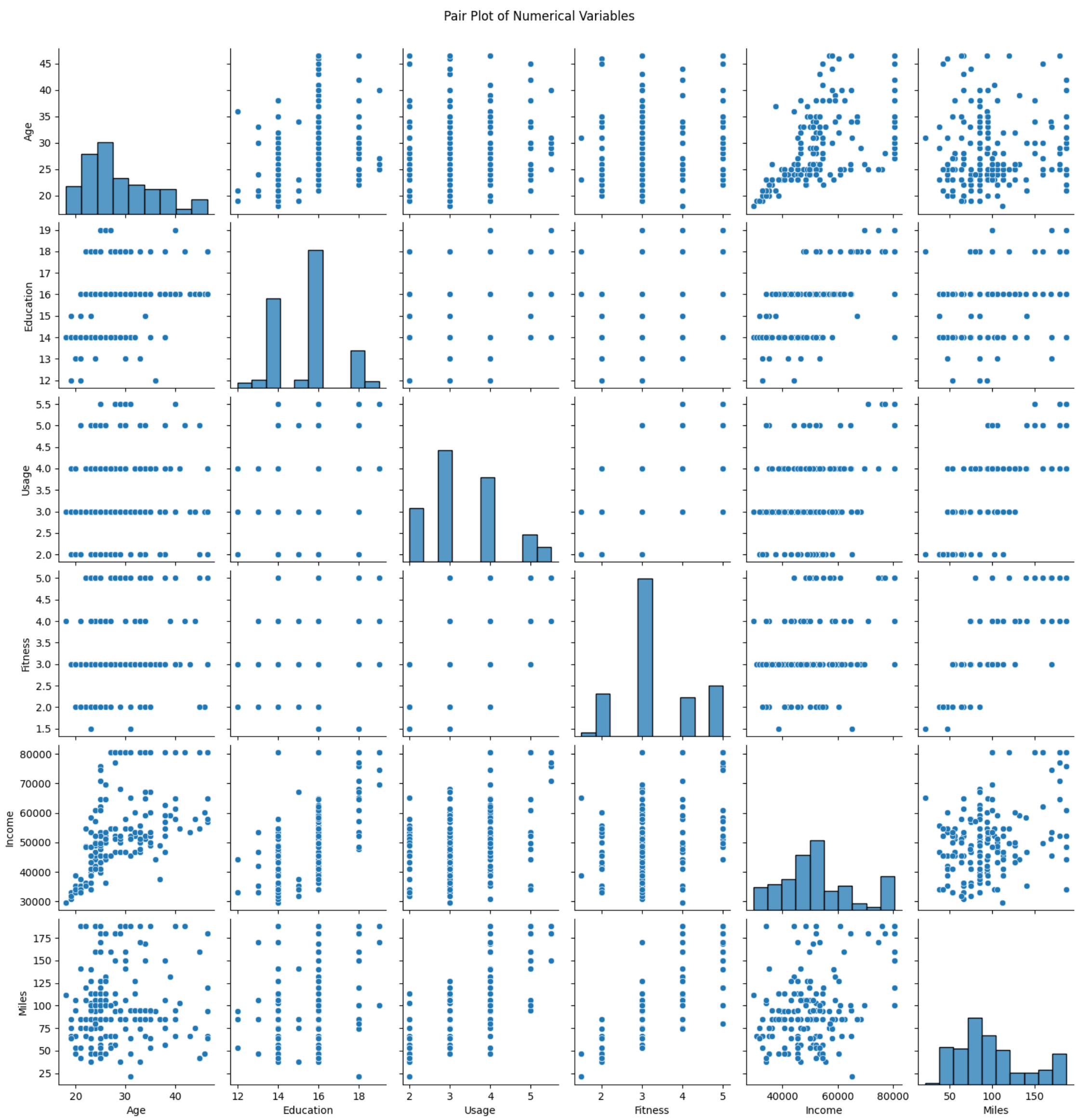
Creates a heatmap to visualize the correlation matrix.

Purpose of using Heatmap:

1. To identify relationships and dependencies between numerical features.
2. Helps spot highly correlated variables (which may indicate redundancy).
3. Guides feature selection or engineering decisions.
4. Detects potential multicollinearity problems for modeling.
5. Provides insights into which variables might influence each other.

Visualizing relationships (linear or nonlinear) between pairs of numerical variables using Pairplot:

[25]:
```
# Select only numerical columns for the pair plot
numerical_cols = df.select_dtypes(include=np.number).columns

# Create a pair plot
sns.pairplot(df[numerical_cols])
plt.suptitle('Pair Plot of Numerical Variables', y=1.02) # Add a title to the␣
  ↪overall plot
plt.show()
```



A pair plot displays scatter plots for each pair of numerical variables, along with histograms or KDE plots on the diagonal.

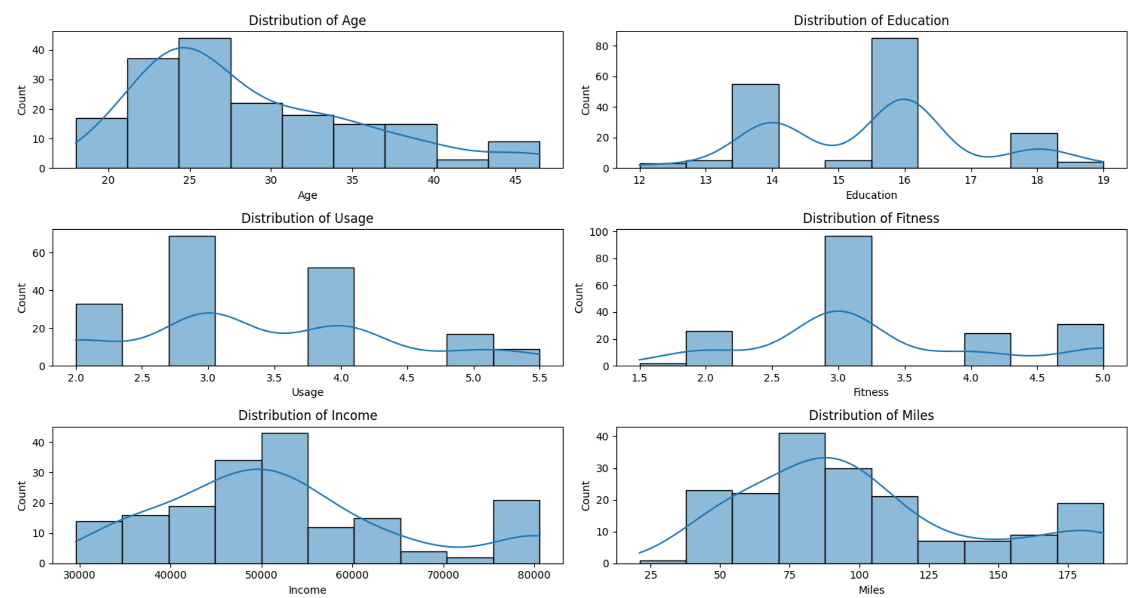
It helps identify relationships, patterns, trends, and outliers among numerical features.

Purpose of Pairplots:

1. To spot correlations, clusters, or patterns.
2. To identify outliers or unusual data points.
3. To understand the distribution and spread of each variable.
4. Helpful for exploratory data analysis (EDA) before modeling.
5. To visually inspect correlation between variables.
6. To identify linear or non-linear relationships.
7. To detect outliers.
8. It's especially helpful before applying statistical or machine learning models.

Visualizing the distribution of each numerical variable in our dataset using histograms combined with Kernel Density Estimates (KDE):

[26]:
```
numerical_cols = df.select_dtypes(include=np.number).columns

# Plot distribution plots for numerical columns
plt.figure(figsize=(15, 8)) # Adjust figure size as needed
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols)//2 + len(numerical_cols)%2, 2, i + 1) # Adjust subplot grid
    sns.histplot(data=df, x=col, kde=True) # Use histplot with KDE
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```



This identifies only the columns in the DataFrame df that have numerical data types (e.g., int, float).

These columns might include variables like Age, Income, Miles, Fitness, etc.

This loops through each numerical column and plots them in a grid layout using plt.subplot.

sns.histplot(…): Draws a histogram of the column.

kde=True adds a smooth curve showing the estimated distribution of the data.

Purpose of Histogram & KDE plot:

1. To understand how data points are distributed for each numeric variable (e.g., normal, skewed, bimodal).
2. KDE helps visualize the underlying probability density more smoothly than histograms alone.
3. Helps identify outliers, gaps, or multiple modes in data.
4. Informs decisions about transformations or feature engineering.
5. Crucial for exploratory data analysis (EDA).

Converting Numerical columns into categorical:

[27]:
```
df['Age_cat'] = pd.cut(df['Age'], bins=4, labels=['Low', 'Medium', 'Medium␣
↪high', 'High'])
df['Age_cat']
```

[27]:
```
0        Low
1        Low
2        Low
3        Low
4        Low
       …
175    High
176    High
177    High
178    High
179    High
Name: Age_cat, Length: 180, dtype: category
Categories (4, object): ['Low' < 'Medium' < 'Medium high' < 'High']
```

[28]:
```
df['Usage_cat'] = pd.cut(df['Usage'], bins=4, labels=['Low', 'Medium', 'Medium␣
↪high', 'High'])
df['Usage_cat']
```

[28]:
```
0           Medium
1              Low
2      Medium high
3           Medium
4      Medium high
```

```
          …
175           High
176           High
177           High
178    Medium high
179    Medium high
Name: Usage_cat, Length: 180, dtype: category
Categories (4, object): ['Low' < 'Medium' < 'Medium high' < 'High']
```

[29]:
```
df['Fitness_cat'] = pd.cut(df['Fitness'], bins=4, labels=['Low', 'Medium', 'Medium high', 'High'])
df['Fitness_cat']
```

[29]:
```
0      Medium high
1           Medium
2           Medium
3           Medium
4              Low
          …
175           High
176    Medium high
177           High
178           High
179           High
Name: Fitness_cat, Length: 180, dtype: category
Categories (4, object): ['Low' < 'Medium' < 'Medium high' < 'High']
```

[30]:
```
df['Income_cat'] = pd.cut(df['Income'], bins=4, labels=['Low', 'Medium', 'MediumHigh', 'High'])
df['Income_cat']
```

[30]:
```
0       Low
1       Low
2       Low
3       Low
4       Low
       …
175    High
176    High
177    High
178    High
179    High
Name: Income_cat, Length: 180, dtype: category
Categories (4, object): ['Low' < 'Medium' < 'MediumHigh' < 'High']
```

[31]:
```
df['Miles_cat'] = pd.cut(df['Miles'], bins=4, labels=['Low', 'Medium', 'Medium high', 'High'])
```

```
df['Miles_cat']
```

[31]:
```
0      Medium high
1           Medium
2           Medium
3           Medium
4              Low
          …
175           High
176           High
177           High
178    Medium high
179           High
Name: Miles_cat, Length: 180, dtype: category
Categories (4, object): ['Low' < 'Medium' < 'Medium high' < 'High']
```

[32]:
```
df['Education_cat'] = pd.cut(df['Education'], bins=4, labels=['Low', 'Medium',␣
↪'Medium high', 'High'])
df['Education_cat']
```

[32]:
```
0           Medium
1           Medium
2           Medium
3              Low
4              Low
          …
175           High
176           High
177    Medium high
178           High
179           High
Name: Education_cat, Length: 180, dtype: category
Categories (4, object): ['Low' < 'Medium' < 'Medium high' < 'High']
```

Converting numerical columns to categorical is done to:

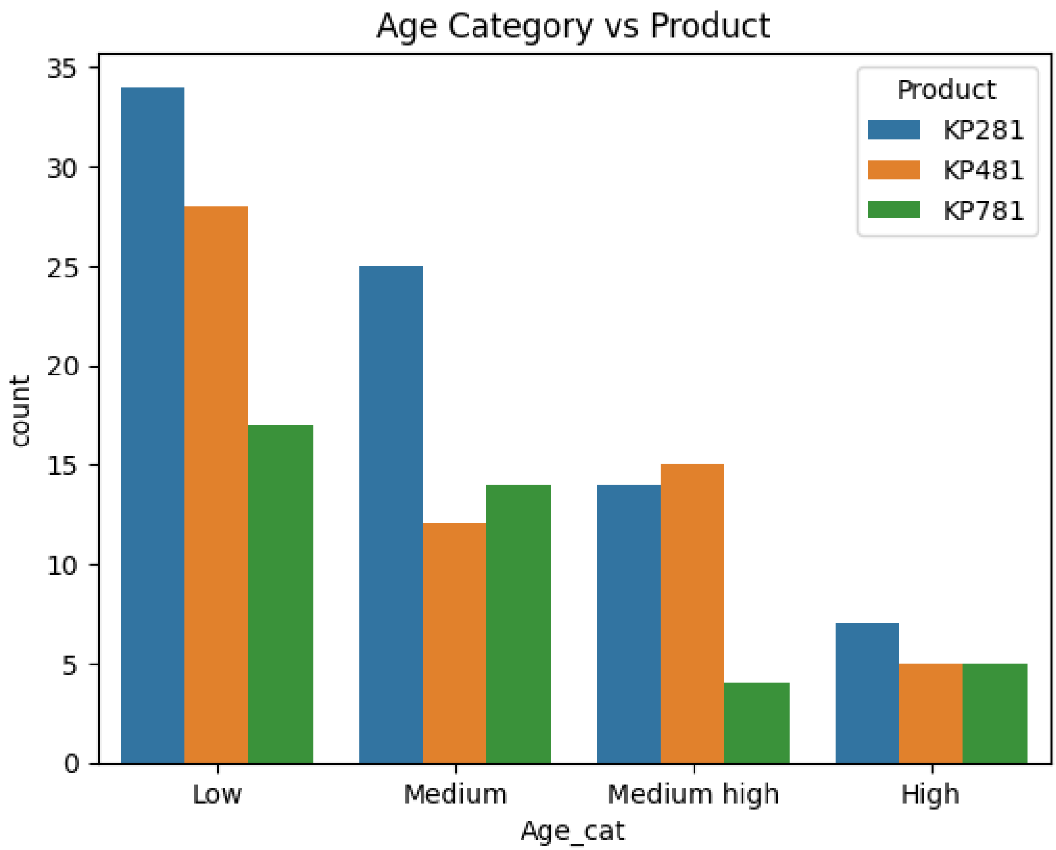
1. Group continuous values into meaningful categories or bins (e.g., age groups like "young", "middle-aged", "senior").
2. Simplify analysis and interpretation by reducing many unique values to a few categories.
3. Handle non-linear relationships where numeric values themselves don't directly relate to the target but categories do.
4. Improve model performance when models benefit from categorical inputs or when categories capture important thresholds.
5. Create features for segmentation or stratification, making it easier to compare groups.
6. Prepare data for visualization where categories are easier to plot and interpret than raw

continuous data.

Creating a Count plot to visualize how the distribution of customers across different categories varies by the product they utilized (Product):

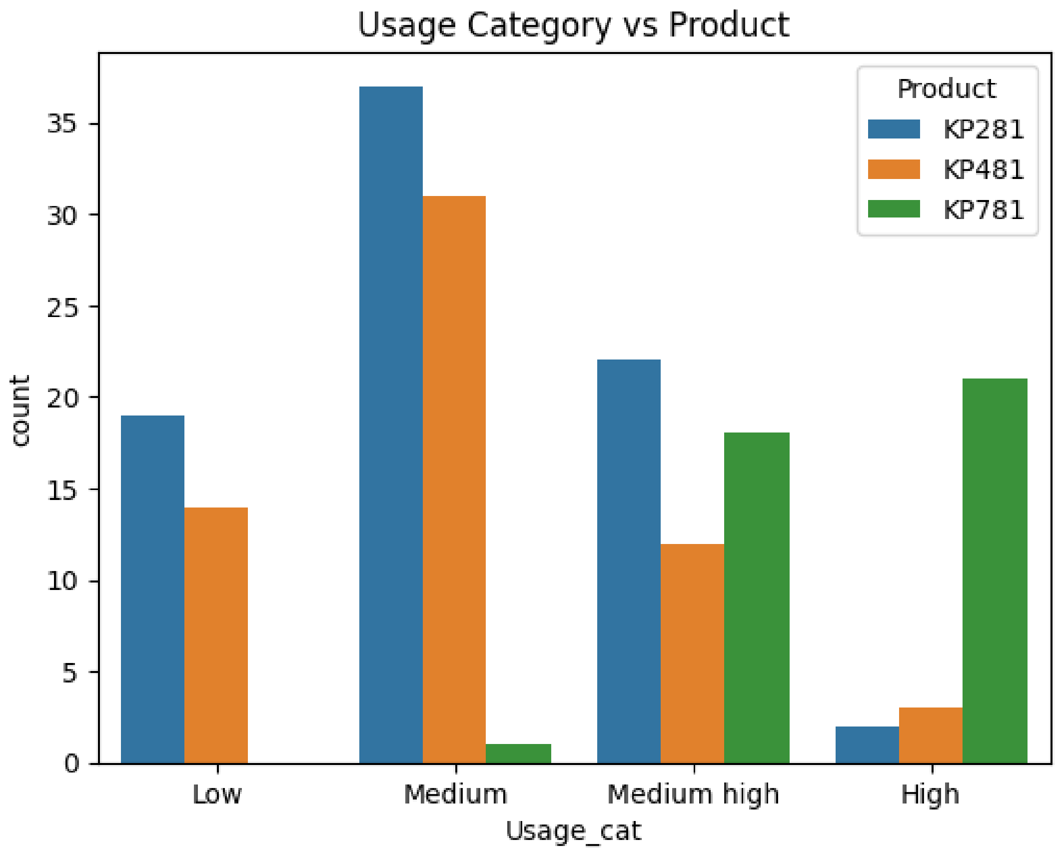
Age_cat vs Product:

[33]:
```
sns.countplot(x='Age_cat', hue='Product', data=df)
plt.title('Age Category vs Product')
plt.show()
```



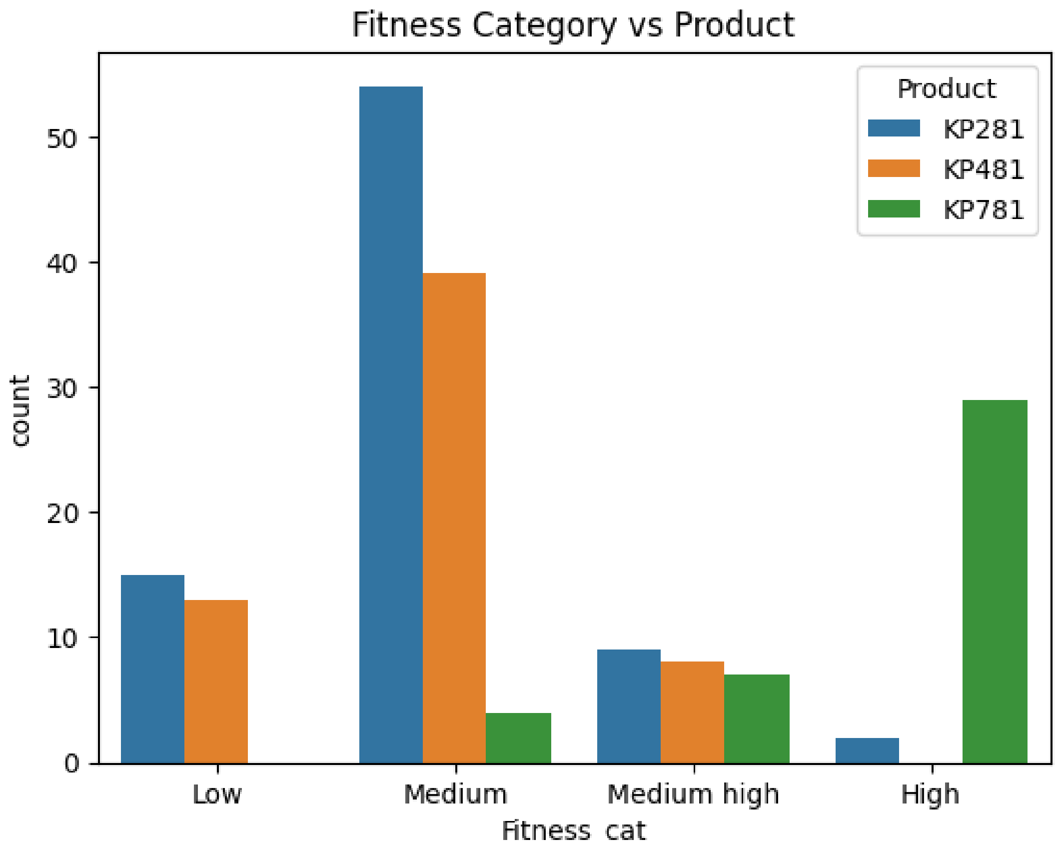
Usage_cat vs Product:

[34]:
```
sns.countplot(x='Usage_cat', hue='Product', data=df)
plt.title('Usage Category vs Product')
plt.show()
```



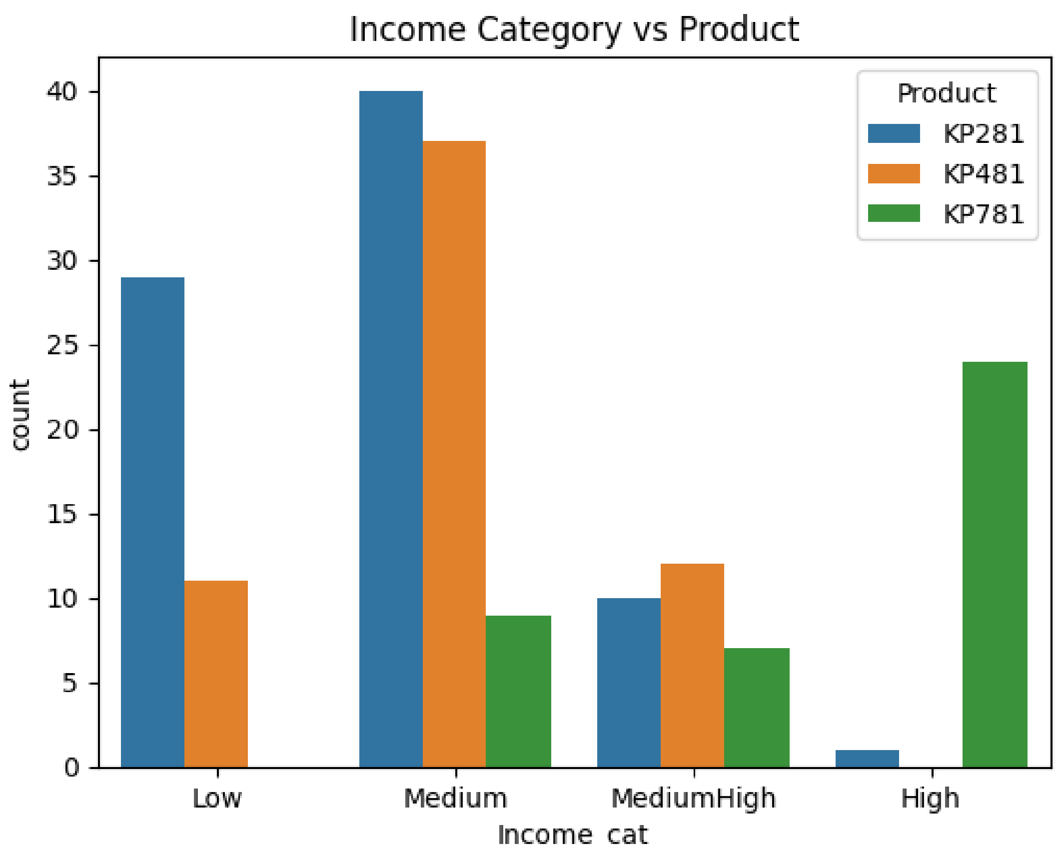
Fitness_cat vs Product:

[35]:
```
sns.countplot(x='Fitness_cat', hue='Product', data=df)
plt.title('Fitness Category vs Product')
plt.show()
```



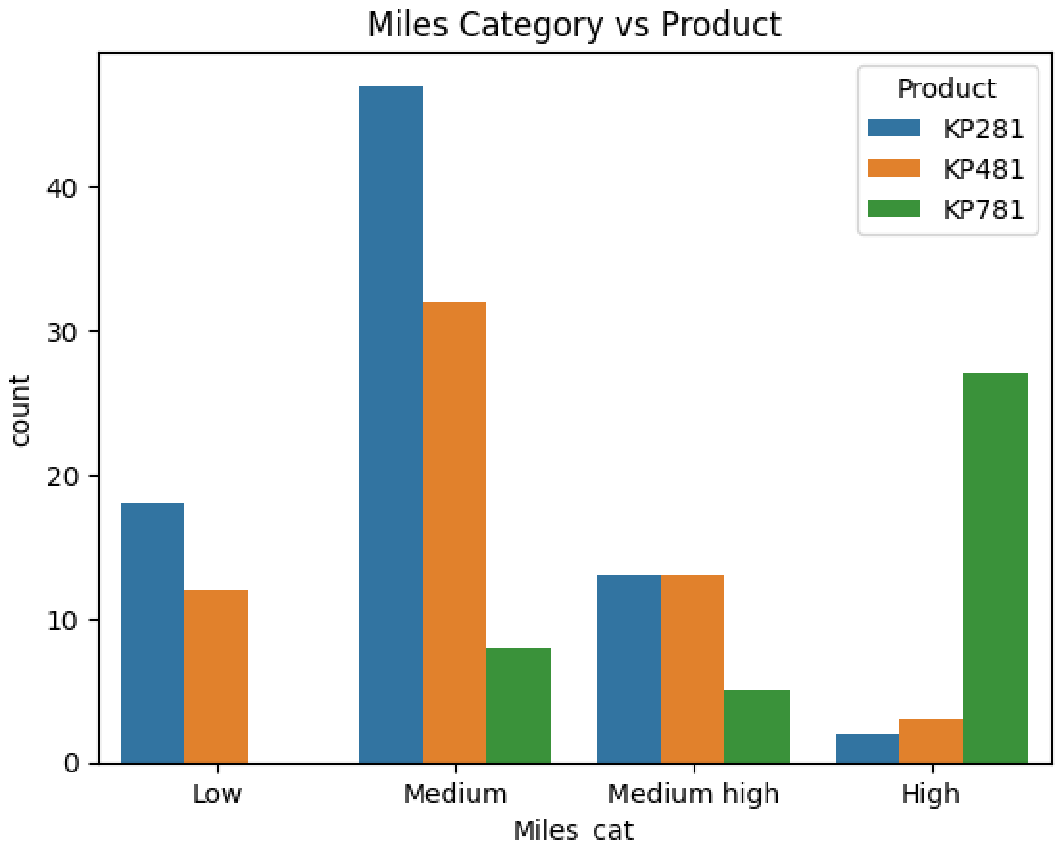
Income_cat vs Product:

[36]:
```
sns.countplot(x='Income_cat', hue='Product', data=df)
plt.title('Income Category vs Product')
plt.show()
```



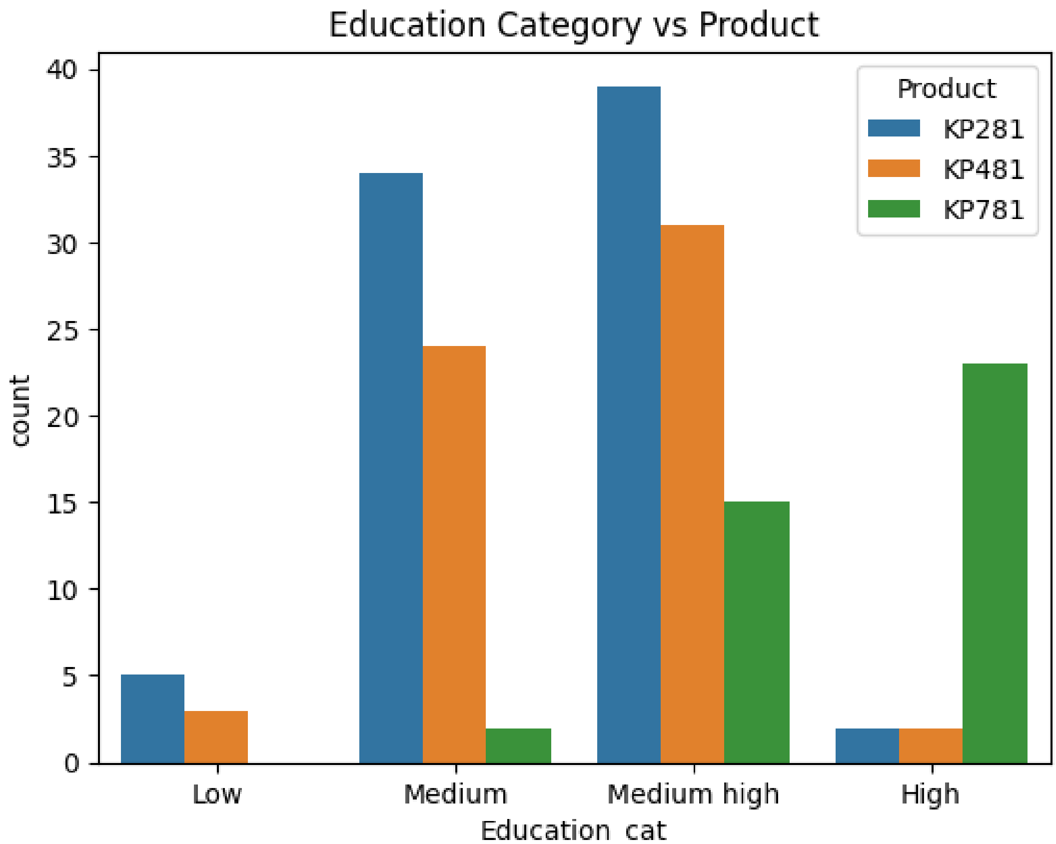
Miles_cat vs Product:

[37]:
```
sns.countplot(x='Miles_cat', hue='Product', data=df)
plt.title('Miles Category vs Product')
plt.show()
```



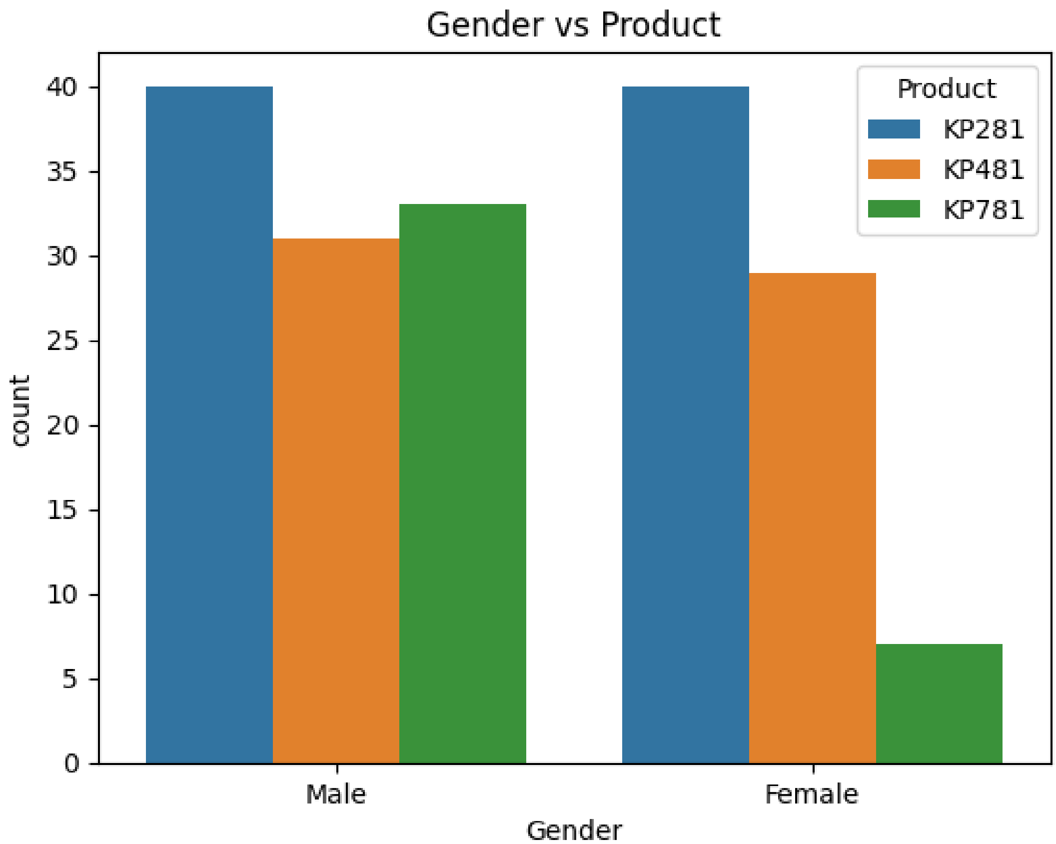
Education_cat vs Product:

[38]:
```
sns.countplot(x='Education_cat', hue='Product', data=df)
plt.title('Education Category vs Product')
plt.show()
```



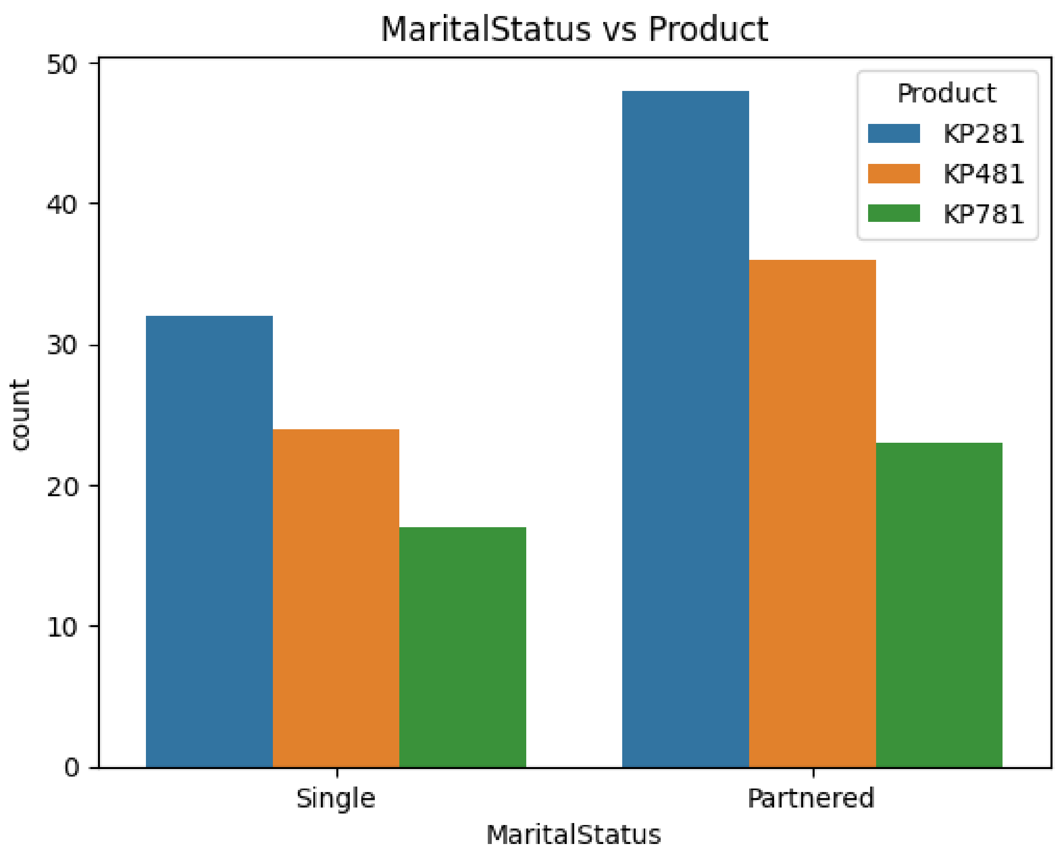
Gender vs Product:

[39]:
```
sns.countplot(x='Gender', hue='Product', data=df)
plt.title('Gender vs Product')
plt.show()
```



MaritalStatus vs Product:

[40]:
```
sns.countplot(x='MaritalStatus', hue='Product', data=df)
plt.title('MaritalStatus vs Product')
plt.show()
```



Column-wise Explanation:

1. Gender vs Product:

Shows how many males vs. females purchased each product.

Helps identify gender preferences for specific treadmill models.

2. MaritalStatus vs Product:

Compares purchases between married vs. single customers.

Can help target marketing strategies based on marital status.

3. Age_cat vs Product:

Shows product preference across age groups (Low, Medium, Medium High, High).

Useful for age-based segmentation.

4. Usage_cat vs Product:

Compares product choice based on usage frequency (e.g., low, medium, high).

Indicates if frequent users prefer higher-end models.

5. Income_cat vs Product:

Examines how income level affects product selection.

Useful for pricing and positioning decisions.

6. Fitness_cat vs Product:

Shows product preference by self-reported fitness level.

Could reflect whether more fit people buy advanced models

7. Miles_cat vs Product:

Compares how far customers run or walk based on product bought.

High-mileage users may lean toward durable models.

8. Education_cat vs Product:

Shows how education level affects treadmill preference.

Could suggest awareness or affordability patterns.

Purpose of Count plot used here for visualizing distribution:

1. To compare product popularity across age groups visually.
2. To identify which products are preferred by different categories.
3. Helps in market segmentation and targeted marketing strategies.
4. Makes it easy to spot trends or imbalances in customer distribution.

Conditional Probability:

Cross-tabulation (crosstab) table to summarize the relationship between two categorical variables:

Relationship between Gender & Product:

[41]:
```
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Gender'],margins= True)
product_distribution
```

[41]:
```
Gender   Female  Male  All
Product
KP281        40    40   80
KP481        29    31   60
KP781         7    33   40
All          76   104  180
```

Relationship between MaritalStatus & Product:

[42]:
```
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['MaritalStatus'],margins= True)
product_distribution
```

[42]:

MaritalStatus	Partnered	Single	All
Product			
KP281	48	32	80
KP481	36	24	60
KP781	23	17	40
All	107	73	180

Relationship between Age_cat & Product:

[43]:
```python
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Age_cat'],margins= True)
product_distribution
```

[43]:

Age_cat	Low	Medium	Medium high	High	All
Product					
KP281	34	25	14	7	80
KP481	28	12	15	5	60
KP781	17	14	4	5	40
All	79	51	33	17	180

Relationship between Usage_cat & Product:

[44]:
```python
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Usage_cat'],margins= True)
product_distribution
```

[44]:

Usage_cat	Low	Medium	Medium high	High	All
Product					
KP281	19	37	22	2	80
KP481	14	31	12	3	60
KP781	0	1	18	21	40
All	33	69	52	26	180

Relationship between Fitness_cat & Product:

[45]:
```python
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Fitness_cat'],margins= True)
product_distribution
```

[45]:

Fitness_cat	Low	Medium	Medium high	High	All
Product					
KP281	15	54	9	2	80
KP481	13	39	8	0	60
KP781	0	4	7	29	40
All	28	97	24	31	180

Relationship between Income_cat & Product:

[46]:
```
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Income_cat'],margins= True)
product_distribution
```

[46]:

Income_cat	Low	Medium	MediumHigh	High	All
Product					
KP281	29	40	10	1	80
KP481	11	37	12	0	60
KP781	0	9	7	24	40
All	40	86	29	25	180

Relationship between Miles_cat & Product:

[47]:
```
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Miles_cat'],margins= True)
product_distribution
```

[47]:

Miles_cat	Low	Medium	Medium high	High	All
Product					
KP281	18	47	13	2	80
KP481	12	32	13	3	60
KP781	0	8	5	27	40
All	30	87	31	32	180

Relationship between Education_cat & Product:

[48]:
```
# Create crosstab (frequency table) and normalize to get percentage
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Education_cat'],margins= True)
product_distribution
```

[48]:

Education_cat	Low	Medium	Medium high	High	All
Product					
KP281	5	34	39	2	80
KP481	3	24	31	2	60
KP781	0	2	15	23	40
All	8	60	85	27	180

Purpose of this crosstab:

1. Understand the distribution of products by various aspects like Gender, Marital status, Age_cat, Usage_cat, Fitness_cat, Income_cat, Miles_cat, Education_cat.
2. Compare preference for each product.
3. Check for imbalances or trends (e.g., Are males more likely to buy KP781?).

4. Provide summary statistics for reports or dashboards.
5. Useful in segmentation and marketing analysis.
6. Quick frequency count between two categorical features.
7. Helps in conditional probability calculations.

Calculating and printing the Conditional probabilities (in percentage form):

Conditional probability of Gender & Product:

[49]:
```
print('The probabilty of female choosing KP281:',40/76*100)
print('The probabilty of male choosing KP281:',40/104*100)

print('The probabilty of female choosing KP481:',29/76*100)
print('The probabilty of male choosing KP481:',31/104*100)

print('The probabilty of female choosing KP781:',7/76*100)
print('The probabilty of male choosing KP781:',33/104*100)
```

```
The probabilty of female choosing KP281: 52.63157894736842
The probabilty of male choosing KP281: 38.46153846153847
The probabilty of female choosing KP481: 38.15789473684211
The probabilty of male choosing KP481: 29.807692307692307
The probabilty of female choosing KP781: 9.210526315789473
The probabilty of male choosing KP781: 31.73076923076923
```

Conditional probability of MaritalStatus & Product:

[50]:
```
print('The probabilty of Partnered choosing KP281:',48/107*100)
print('The probabilty of single choosing KP281:',32/73*100)

print('The probabilty of Partnered choosing KP481:',36/107*100)
print('The probabilty of single choosing KP481:',24/73*100)

print('The probabilty of Partnered choosing KP781:',23/107*100)
print('The probabilty of single choosing KP781:',17/73*100)
```

```
The probabilty of Partnered choosing KP281: 44.85981308411215
The probabilty of single choosing KP281: 43.83561643835616
The probabilty of Partnered choosing KP481: 33.64485981308411
The probabilty of single choosing KP481: 32.87671232876712
The probabilty of Partnered choosing KP781: 21.49532710280374
The probabilty of single choosing KP781: 23.28767123287671
```

Conditional probability of Age_cat & Product:

[51]:
```
print('The probabilty of Low aged choosing KP281:',34/79*100)
print('The probabilty of Medium aged choosing KP281:',25/51*100)
print('The probabilty of Medium high aged choosing KP281:',14/33*100)
print('The probabilty of High aged choosing KP281:',7/17*100)
```

```
print('The probabilty of Low aged choosing KP481:',28/79*100)
print('The probabilty of Medium aged choosing KP481:',12/51*100)
print('The probabilty of Medium high aged choosing KP481:',15/33*100)
print('The probabilty of High aged choosing KP481:',5/17*100)

print('The probabilty of Low aged choosing KP781:',17/79*100)
print('The probabilty of Medium aged choosing KP781:',14/51*100)
print('The probabilty of Medium high aged choosing KP781:',4/33*100)
print('The probabilty of High aged choosing KP781:',5/17*100)
```

```
The probabilty of Low aged choosing KP281: 43.037974683544306
The probabilty of Medium aged choosing KP281: 49.01960784313725
The probabilty of Medium high aged choosing KP281: 42.42424242424242
The probabilty of High aged choosing KP281: 41.17647058823529
The probabilty of Low aged choosing KP481: 35.44303797468354
The probabilty of Medium aged choosing KP481: 23.52941176470588
The probabilty of Medium high aged choosing KP481: 45.45454545454545
The probabilty of High aged choosing KP481: 29.411764705882355
The probabilty of Low aged choosing KP781: 21.518987341772153
The probabilty of Medium aged choosing KP781: 27.450980392156865
The probabilty of Medium high aged choosing KP781: 12.121212121212121
The probabilty of High aged choosing KP781: 29.411764705882355
```

Conditional probability of Usage_cat & Product:

[52]:
```
print('The probabilty of low covered people using KP281:', 19/33*100)
print('The probabilty of Medium covered people using KP281:', 37/69*100)
print('The probabilty of Medium high covered people using KP281:', 22/52*100)
print('The probabilty of High covered people using KP281:', 2/26*100)

print('The probabilty of low covered people using KP481:', 14/33*100)
print('The probabilty of Medium covered people using KP481:', 31/69*100)
print('The probabilty of Medium high covered people using KP481:', 12/52*100)
print('The probabilty of High covered people using KP481:', 3/26*100)

print('The probabilty of low covered people using KP781:', 0/33*100)
print('The probabilty of Medium covered people using KP781:', 1/69*100)
print('The probabilty of Medium high covered people using KP781:', 18/52*100)
print('The probabilty of High covered people using KP781:', 21/26*100)
```

```
The probabilty of low covered people using KP281: 57.57575757575758
The probabilty of Medium covered people using KP281: 53.62318840579711
The probabilty of Medium high covered people using KP281: 42.30769230769231
The probabilty of High covered people using KP281: 7.6923076923076925
The probabilty of low covered people using KP481: 42.42424242424242
The probabilty of Medium covered people using KP481: 44.927536231884055
The probabilty of Medium high covered people using KP481: 23.076923076923077
```

```
The probabilty of High covered people using KP481: 11.538461538461538
The probabilty of low covered people using KP781: 0.0
The probabilty of Medium covered people using KP781: 1.4492753623188406
The probabilty of Medium high covered people using KP781: 34.61538461538461
The probabilty of High covered people using KP781: 80.76923076923077
```

Conditional probability of Fitness_cat & Product:

[53]:
```
print('The probabilty of Low fit people choosing KP281:',15/28*100)
print('The probabilty of Medium fit people choosing KP281:',54/97*100)
print('The probabilty of Medium high fit people choosing KP281:',9/24*100)
print('The probabilty of High fit people choosing KP281:',2/31*100)

print('The probabilty of Low fit people choosing KP481:',13/28*100)
print('The probabilty of Medium fit people choosing KP481:',39/97*100)
print('The probabilty of Medium high fit people choosing KP481:',8/24*100)
print('The probabilty of High fit people choosing KP481:',0/31*100)

print('The probabilty of Low fit people choosing KP781:',0/28*100)
print('The probabilty of Medium fit people choosing KP781:',4/97*100)
print('The probabilty of Medium high fit people choosing KP781:',7/24*100)
print('The probabilty of High fit people choosing KP781:',29/31*100)
```

```
The probabilty of Low fit people choosing KP281: 53.57142857142857
The probabilty of Medium fit people choosing KP281: 55.670103092783506
The probabilty of Medium high fit people choosing KP281: 37.5
The probabilty of High fit people choosing KP281: 6.451612903225806
The probabilty of Low fit people choosing KP481: 46.42857142857143
The probabilty of Medium fit people choosing KP481: 40.20618556701031
The probabilty of Medium high fit people choosing KP481: 33.33333333333333
The probabilty of High fit people choosing KP481: 0.0
The probabilty of Low fit people choosing KP781: 0.0
The probabilty of Medium fit people choosing KP781: 4.123711340206185
The probabilty of Medium high fit people choosing KP781: 29.166666666666668
The probabilty of High fit people choosing KP781: 93.54838709677419
```

Conditional probability of Income_cat & Product:

[54]:
```
print('The probabilty of Low income people choosing KP281:',29/40*100)
print('The probabilty of Medium income people choosing KP281:',40/86*100)
print('The probabilty of Medium high income people choosing KP281:',10/29*100)
print('The probabilty of High income choosing KP281:',1/25*100)

print('The probabilty of Low income people choosing KP481:',11/40*100)
print('The probabilty of Medium income people choosing KP481:',37/86*100)
print('The probabilty of Medium high income people choosing KP481:',12/29*100)
print('The probabilty of High income choosing KP481:',0/25*100)

print('The probabilty of Low income people choosing KP781:',0/40*100)
```

```python
print('The probabilty of Medium income people choosing KP781:',9/86*100)
print('The probabilty of Medium high income people choosing KP781:',7/29*100)
print('The probabilty of High income choosing KP781:',24/25*100)
```

```
The probabilty of Low income people choosing KP281: 72.5
The probabilty of Medium income people choosing KP281: 46.51162790697674
The probabilty of Medium high income people choosing KP281: 34.48275862068966
The probabilty of High income choosing KP281: 4.0
The probabilty of Low income people choosing KP481: 27.500000000000004
The probabilty of Medium income people choosing KP481: 43.02325581395349
The probabilty of Medium high income people choosing KP481: 41.37931034482759
The probabilty of High income choosing KP481: 0.0
The probabilty of Low income people choosing KP781: 0.0
The probabilty of Medium income people choosing KP781: 10.465116279069768
The probabilty of Medium high income people choosing KP781: 24.137931034482758
The probabilty of High income choosing KP781: 96.0
```

Conditional probability of Miles_cat & Product:

[55]:
```python
print('The probabilty of people covering low miles choosing KP281:',18/30*100)
print('The probabilty of people covering Medium miles choosing KP281:',47/
 ↪87*100)
print('The probabilty of people covering Medium high miles choosing KP281:',13/
 ↪31*100)
print('The probabilty of people covering High miles choosing KP281:',2/32*100)

print('The probabilty of people covering low miles choosing KP481:',12/30*100)
print('The probabilty of people covering Medium miles choosing KP481:',32/
 ↪87*100)
print('The probabilty of people covering Medium high miles choosing KP481:',13/
 ↪31*100)
print('The probabilty of people covering High miles choosing KP481:',3/32*100)

print('The probabilty of people covering low miles choosing KP781:',0/30*100)
print('The probabilty of people covering Medium miles choosing KP781:',8/87*100)
print('The probabilty of people covering Medium high miles choosing KP781:',5/
 ↪31*100)
print('The probabilty of people covering High miles choosing KP781:',27/32*100)
```

```
The probabilty of people covering low miles choosing KP281: 60.0
The probabilty of people covering Medium miles choosing KP281: 54.02298850574713
The probabilty of people covering Medium high miles choosing KP281:
41.935483870967744
The probabilty of people covering High miles choosing KP281: 6.25
The probabilty of people covering low miles choosing KP481: 40.0
The probabilty of people covering Medium miles choosing KP481: 36.7816091954023
The probabilty of people covering Medium high miles choosing KP481:
41.935483870967744
```

```
The probabilty of people covering High miles choosing KP481: 9.375
The probabilty of people covering low miles choosing KP781: 0.0
The probabilty of people covering Medium miles choosing KP781: 9.195402298850574
The probabilty of people covering Medium high miles choosing KP781:
16.129032258064516
The probabilty of people covering High miles choosing KP781: 84.375
```

Conditional probability of Education_cat & Product:

[56]:
```
print('The probabilty of Low educated people choosing KP281:',5/8*100)
print('The probabilty of Medium educated people choosing KP281:',34/60*100)
print('The probabilty of Medium high educated people choosing KP281:',39/85*100)
print('The probabilty of High educated people choosing KP281:',2/27*100)

print('The probabilty of Low educated people choosing KP481:',3/8*100)
print('The probabilty of Medium educated people choosing KP481:',24/60*100)
print('The probabilty of Medium high educated people choosing KP481:',31/85*100)
print('The probabilty of High educated people choosing KP481:',2/27*100)

print('The probabilty of Low educated people choosing KP781:',0/8*100)
print('The probabilty of Medium educated people choosing KP781:',2/60*100)
print('The probabilty of Medium high educated people choosing KP781:',15/85*100)
print('The probabilty of High educated people choosing KP781:',23/27*100)
```

```
The probabilty of Low educated people choosing KP281: 62.5
The probabilty of Medium educated people choosing KP281: 56.666666666666664
The probabilty of Medium high educated people choosing KP281: 45.88235294117647
The probabilty of High educated people choosing KP281: 7.4074074074074066
The probabilty of Low educated people choosing KP481: 37.5
The probabilty of Medium educated people choosing KP481: 40.0
The probabilty of Medium high educated people choosing KP481: 36.470588235294116
The probabilty of High educated people choosing KP481: 7.4074074074074066
The probabilty of Low educated people choosing KP781: 0.0
The probabilty of Medium educated people choosing KP781: 3.3333333333333335
The probabilty of Medium high educated people choosing KP781: 17.647058823529413
The probabilty of High educated people choosing KP781: 85.18518518518519
```

Purpose of Conditional Probabilty that is used here:

1. Gender:

Understand how product choices differ between males and females.

Useful for targeted marketing and gender-specific campaigns.

Reveals if a product appeals more to one gender.

2. Marital Status:

Examine if being married or single influences product preference.

Insightful for lifestyle-based targeting — married users may have different usage goals than singles.

3. Age Category (Age_cat):

Discover how different age groups (low, medium, medium high, high) choose products.

Helps age-segment marketing: e.g., younger users may prefer affordable or tech-rich models.

Supports design decisions (e.g., interface, comfort, speed range).

4. Usage Category (Usage_cat):

Shows which product is more popular based on how frequently customers use treadmills.

Assists in recommending models based on intensity of use.

Important for durability-focused marketing.

5. Income Category (Income_cat):

Reveals whether product preference varies by income level.

Useful for price sensitivity analysis.

Supports tiered product offerings (budget vs premium models).

6. Fitness Category (Fitness_cat):

Identifies product preference based on a customer's fitness level.

Helps target beginners vs advanced users.

Can guide training features or app integrations.

7. Miles Category (Miles_cat):

Correlates distance run per week to product choice.

Helps in understanding which products are preferred by long-distance vs casual runners.

Can be used to highlight performance features.

8. Education Category (Education_cat):

Analyzes whether education level influences product decisions.

May reflect on brand awareness, research behavior, or trust in features.

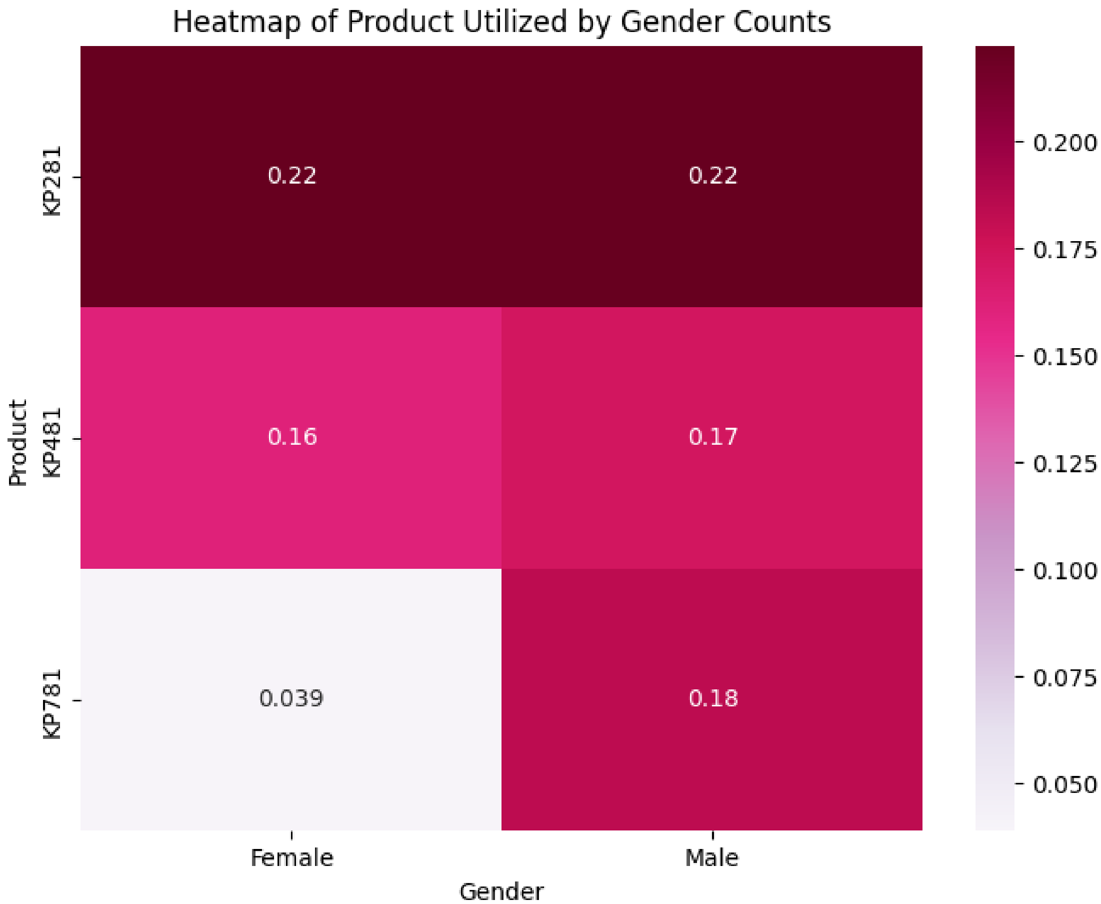
Useful for content strategy and communication style.

Customer Behavior Visualization Using Heatmaps:

Gender vs Product:

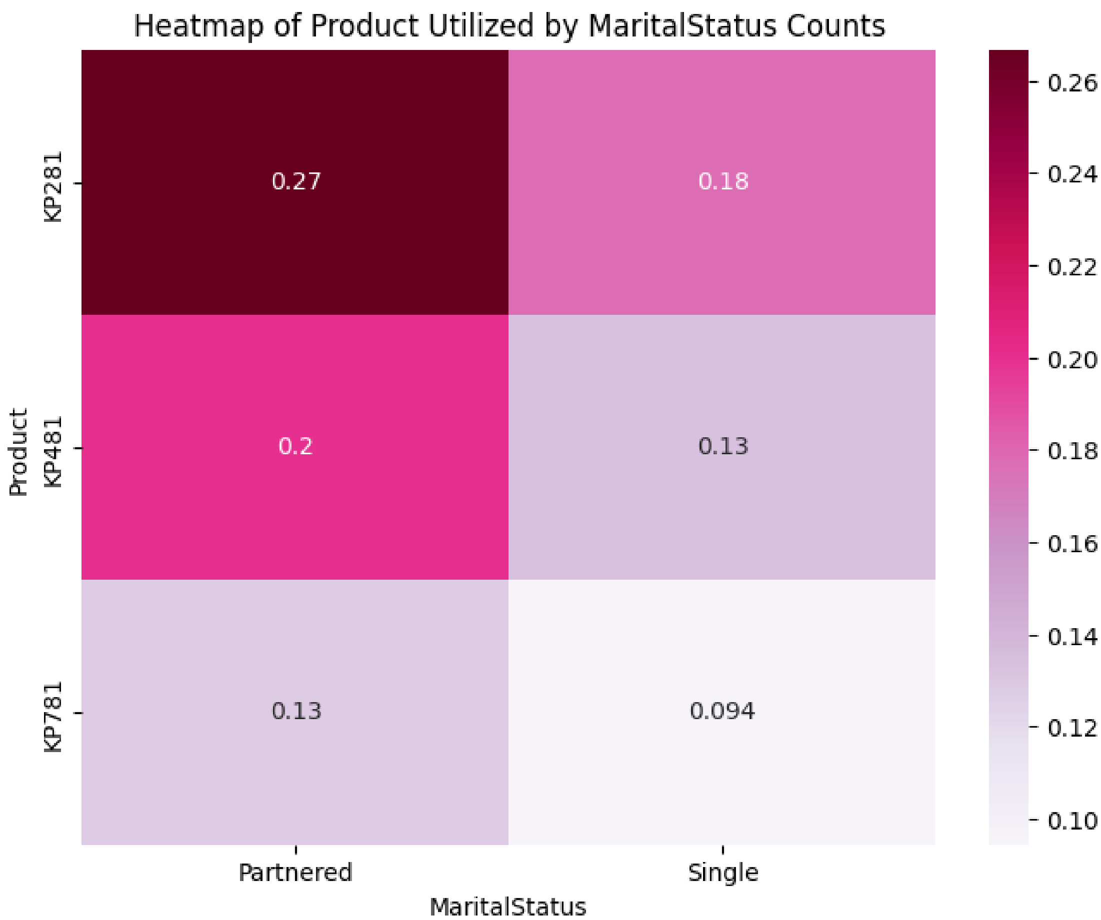
[57]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Gender'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Gender Counts')
plt.xlabel('Gender')
```

```
plt.ylabel('Product')
plt.show()
```



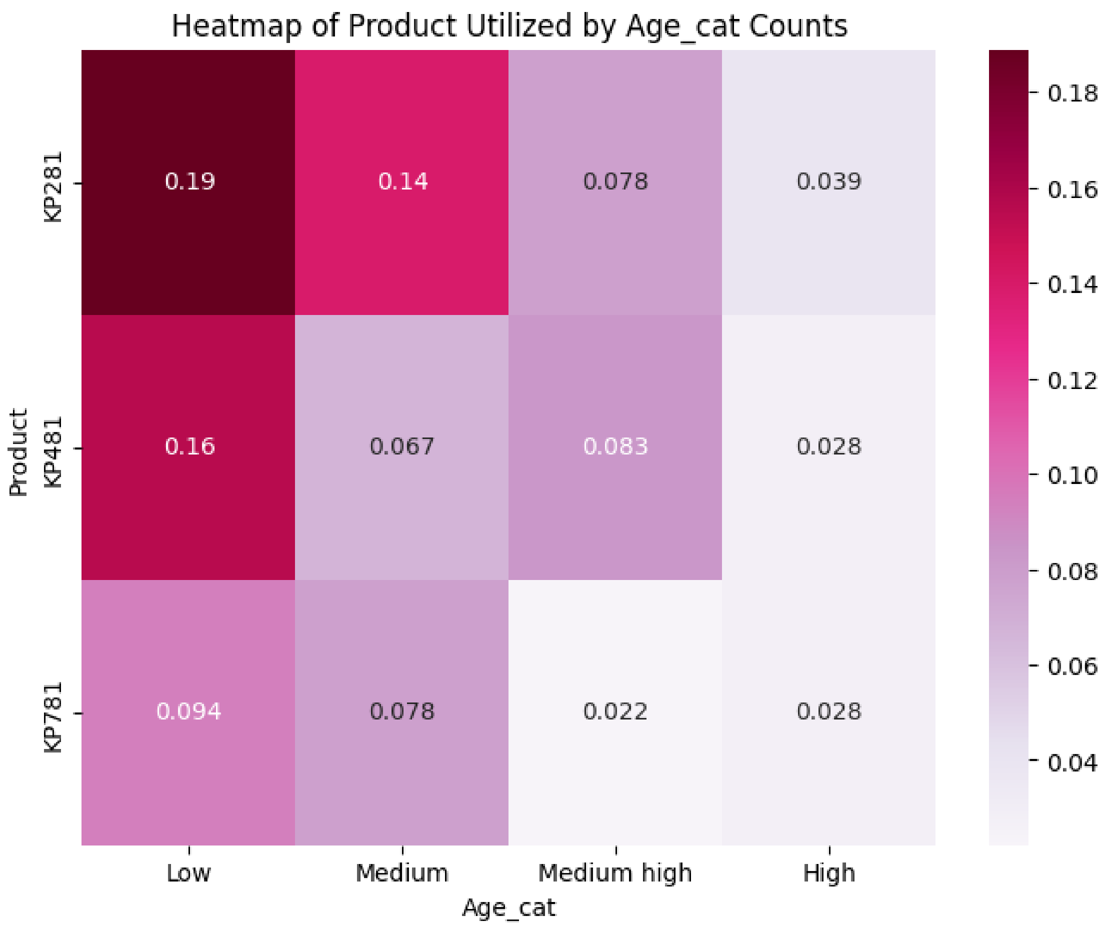
MaritalStatus vs Product:

[58]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['MaritalStatus'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by MaritalStatus Counts')
plt.xlabel('MaritalStatus')
plt.ylabel('Product')
plt.show()
```



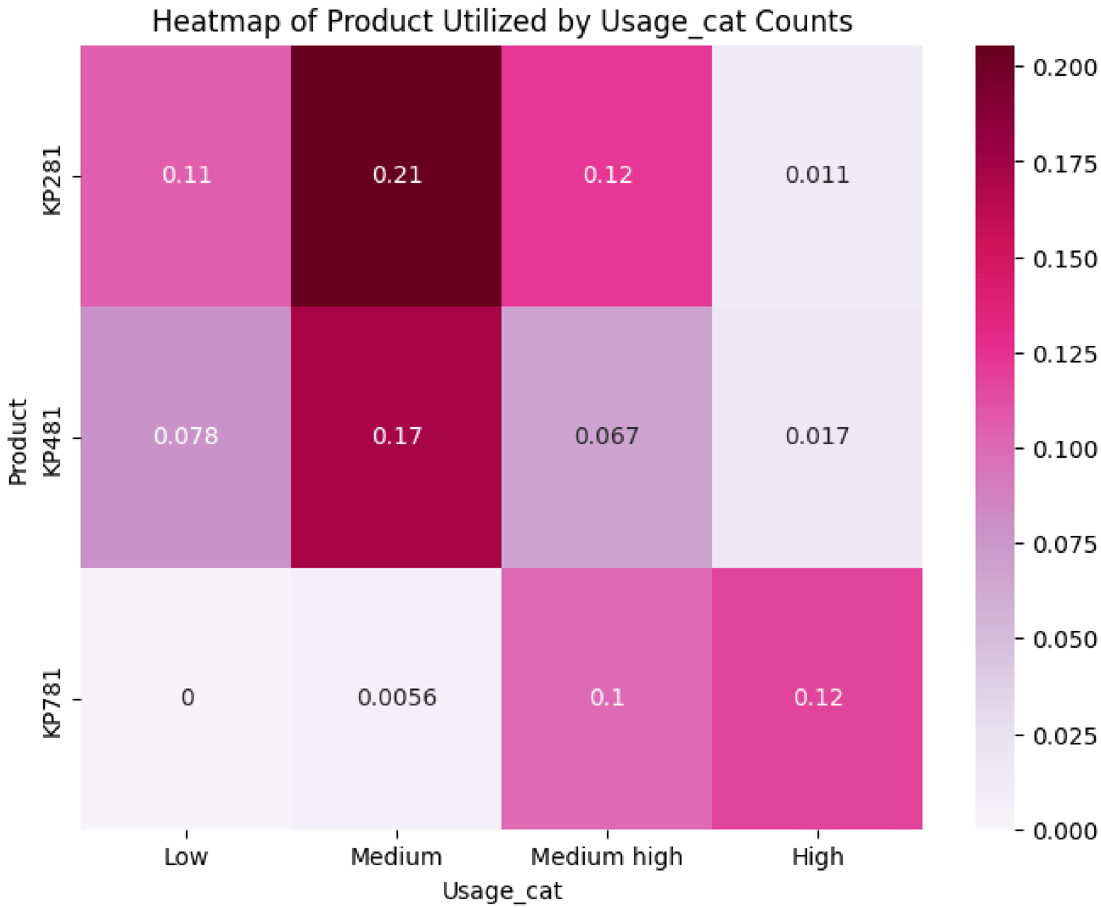
Age_cat vs Product:

[59]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Age_cat'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Age_cat Counts')
plt.xlabel('Age_cat')
plt.ylabel('Product')
plt.show()
```



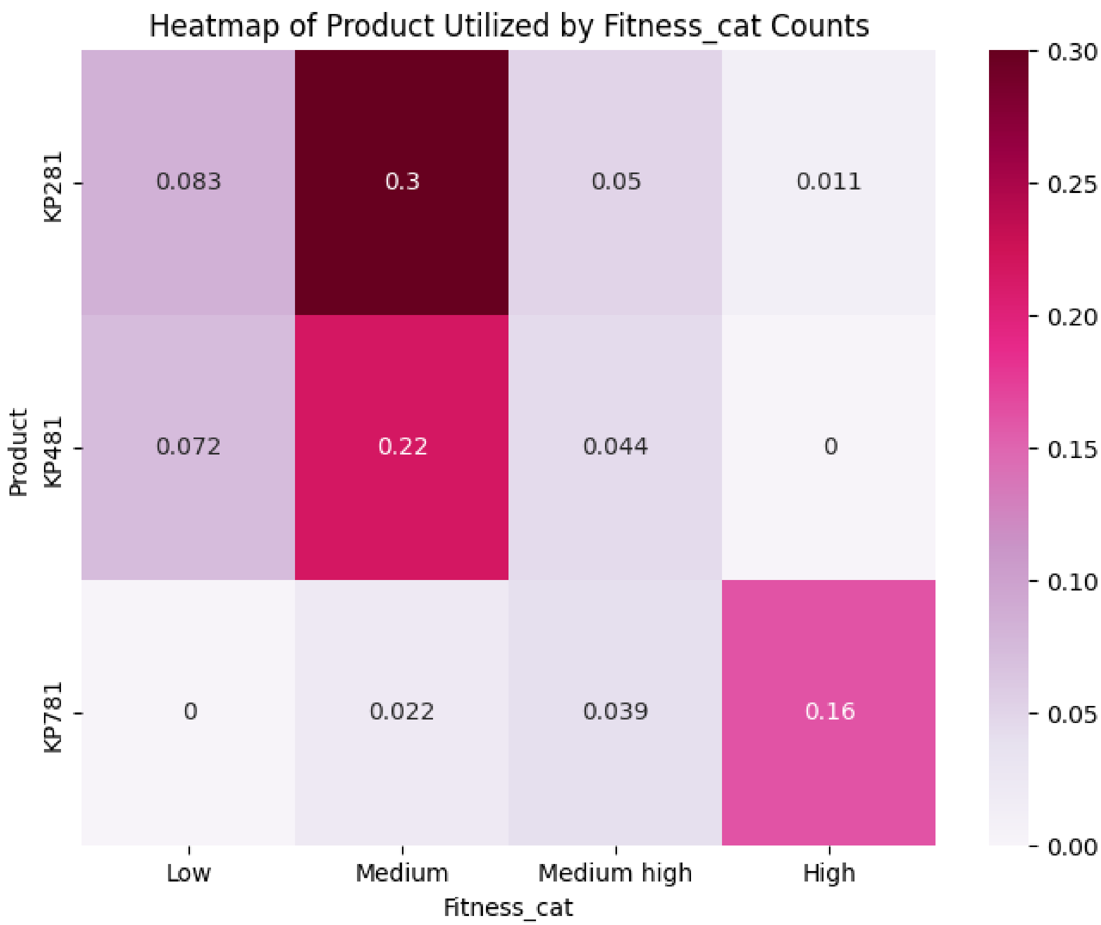
Usage_cat vs Product:

[60]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Usage_cat'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Usage_cat Counts')
plt.xlabel('Usage_cat')
plt.ylabel('Product')
plt.show()
```



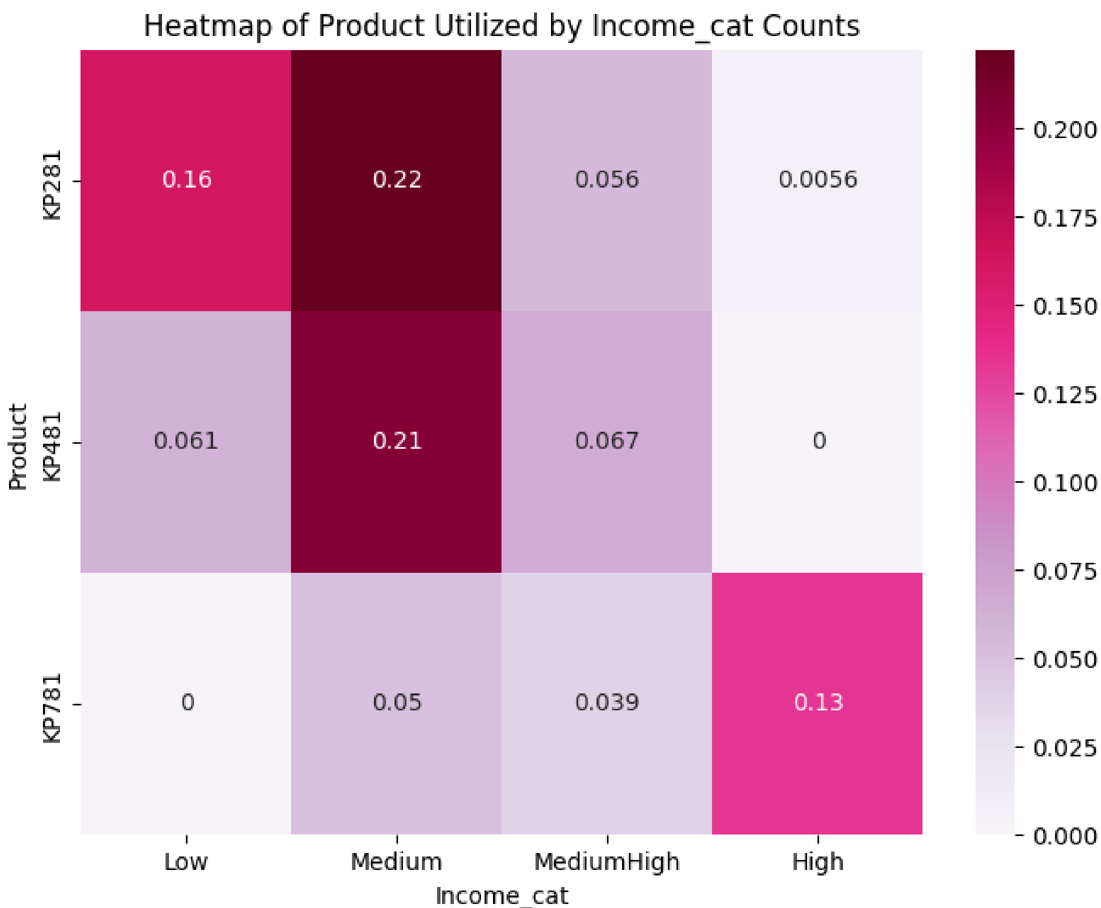
Fitness_cat vs Product:

[61]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
  ↪columns=df['Fitness_cat'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Fitness_cat Counts')
plt.xlabel('Fitness_cat')
plt.ylabel('Product')
plt.show()
```



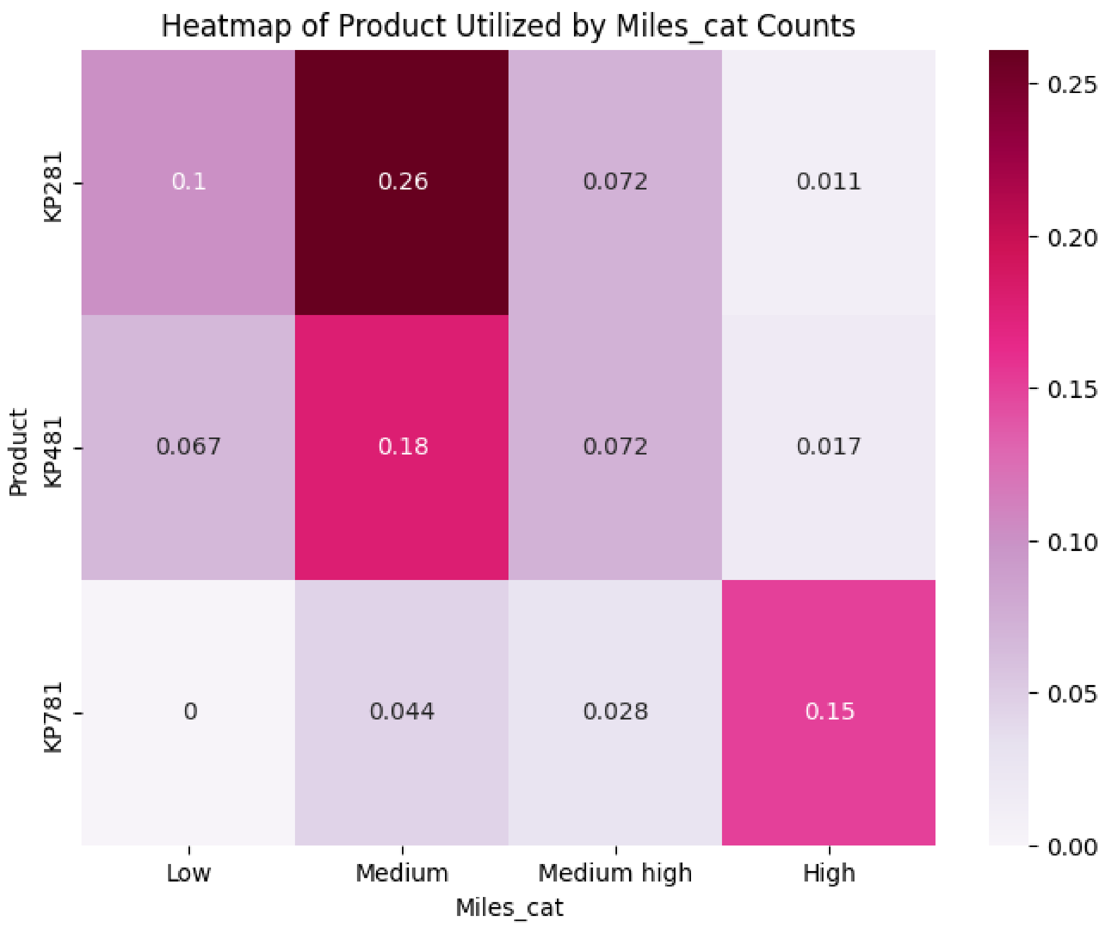
Income_cat vs Product:

[62]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Income_cat'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Income_cat Counts')
plt.xlabel('Income_cat')
plt.ylabel('Product')
plt.show()
```



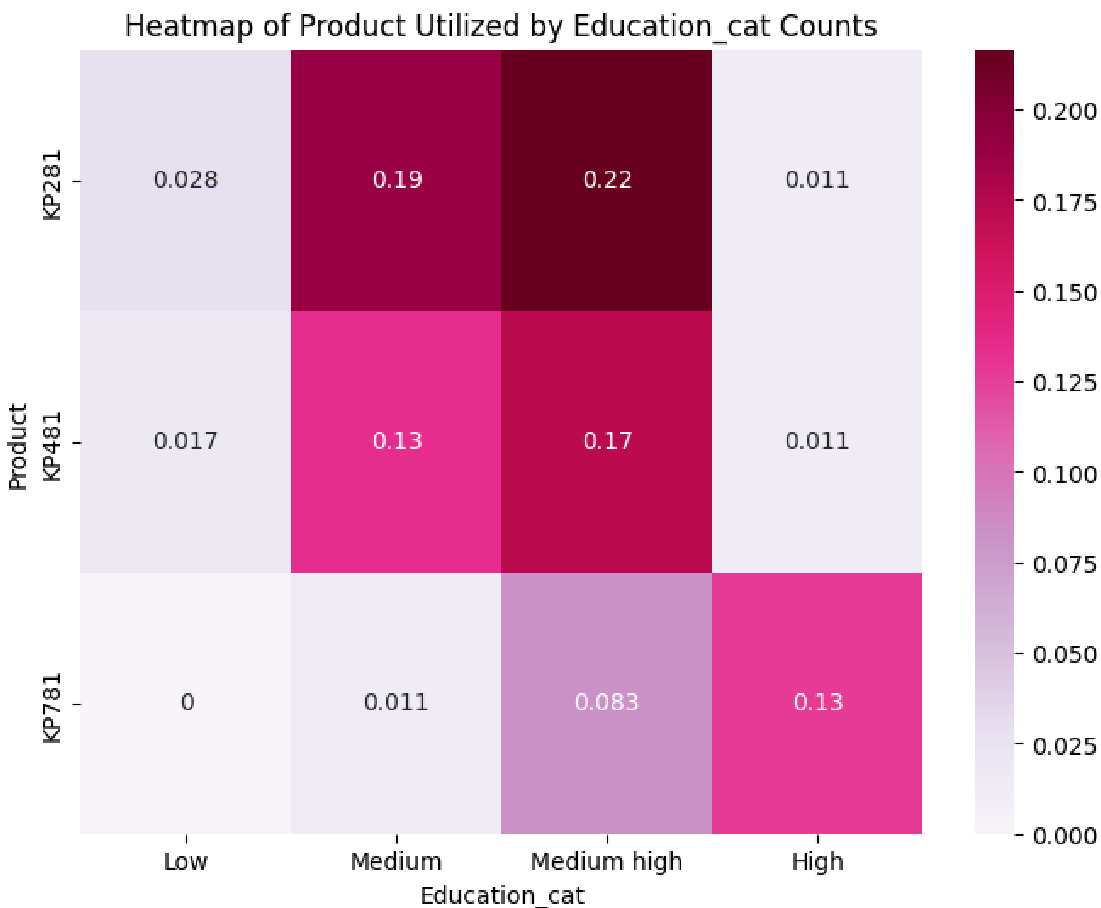
Miles_cat vs Product:

[63]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Miles_cat'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Miles_cat Counts')
plt.xlabel('Miles_cat')
plt.ylabel('Product')
plt.show()
```



Education_cat vs Product:

[64]:
```
product_distribution = pd.crosstab(index=df['Product'],␣
↪columns=df['Education_cat'],normalize = True)

# Plot the heatmap of the product_distribution crosstab
plt.figure(figsize=(8, 6))
sns.heatmap(product_distribution, annot=True, cmap='PuRd')
plt.title('Heatmap of Product Utilized by Education_cat Counts')
plt.xlabel('Education_cat')
plt.ylabel('Product')
plt.show()
```



Heatmap usage Explanation category-wise:

1. Gender vs Product:

Shows how males and females differ in product choice.

Identifies if a product is gender-skewed.

Useful for gender-specific marketing strategies.

2. Marital Status vs Product:

Visualizes how married vs single customers choose products.

May reveal lifestyle-based preferences (e.g., single users might choose more high-performance models).

Useful for segmentation and messaging tone (e.g., family-oriented vs solo fitness).

3. Age Category (Age_cat) vs Product:

Shows which age groups prefer which product.

Useful to know if KP281 is popular among young adults or KP781 among older adults.

Supports age-targeted promotions and UI/UX design adjustments.

4. Usage Category (Usage_cat) vs Product:

Reveals how usage frequency relates to product selection.

Heavier users may lean toward more durable models like KP781.

Helps in product positioning (casual vs heavy usage).

5. Income Category (Income_cat) vs Product:

Indicates whether higher or lower-income groups prefer certain models.

Helpful in understanding price sensitivity.

Drives tiered pricing strategies or EMI-based offers.

6. Fitness Category (Fitness_cat) vs Product:

Assesses which products are chosen by customers with varied fitness levels.

Beginners might prefer basic models; advanced users might prefer feature-rich models.

Helps tailor training programs or fitness plans per product.

7. Miles Category (Miles_cat) vs Product:

Indicates if people running longer distances prefer a specific product (likely KP781).

Supports product optimization for endurance vs casual workouts.

Informs performance marketing and feature design.

8. Education Category (Education_cat) vs Product:

Explores if education level affects product choice.

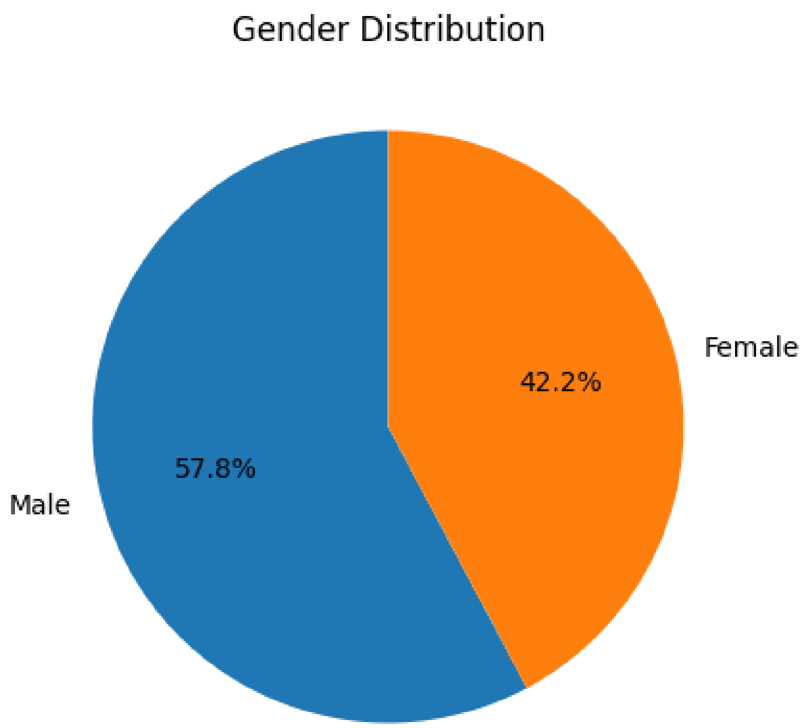
May reflect decision-making style (e.g., more educated customers researching specs more deeply).

Informs communication strategy (simpler vs detailed product pages).

Marginal Probability using Pie chart:

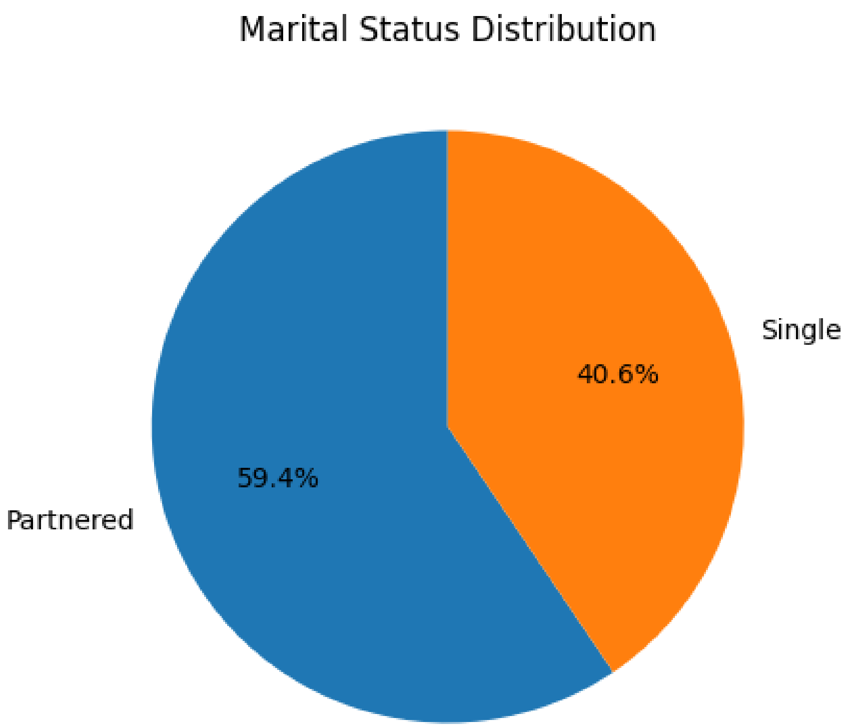
Gender Distribution:

[65]:
```
df['Gender'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,␣
↪title='Gender Distribution', figsize=(5, 5))
plt.ylabel('')
plt.show()
```



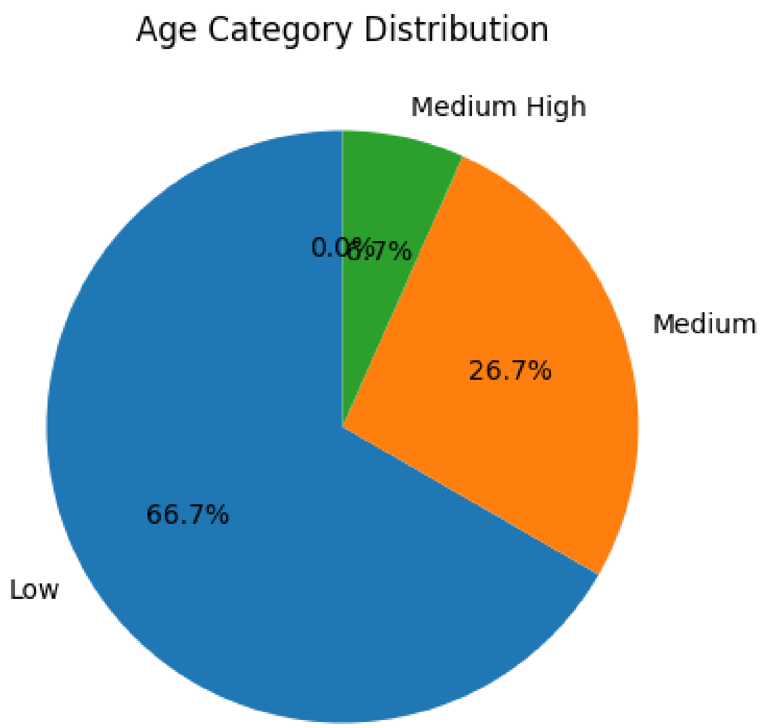
MaritalStatus Distribution:

[66]:
```
df['MaritalStatus'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,␣
 ↪title='Marital Status Distribution', figsize=(5, 5))
plt.ylabel('')
plt.show()
```



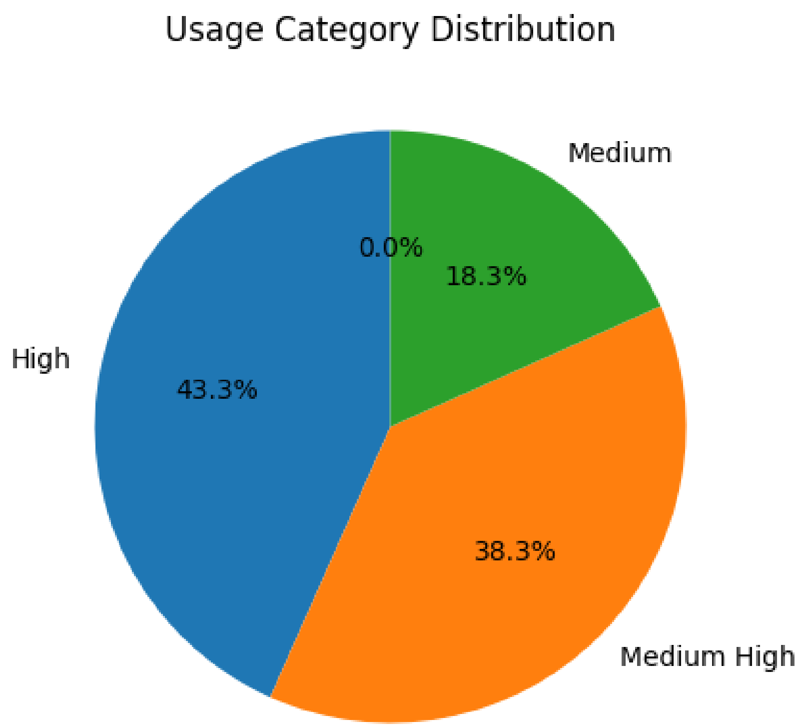
Age_cat Distribution:

[67]:
```
df['age_cat'] = pd.cut(df['Age'], bins=[0, 30, 40, 50, 100], labels=['Low',␣
↪'Medium', 'Medium High', 'High'])
df['age_cat'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,␣
↪title='Age Category Distribution', figsize=(5, 5))
plt.ylabel('')
plt.show()
```



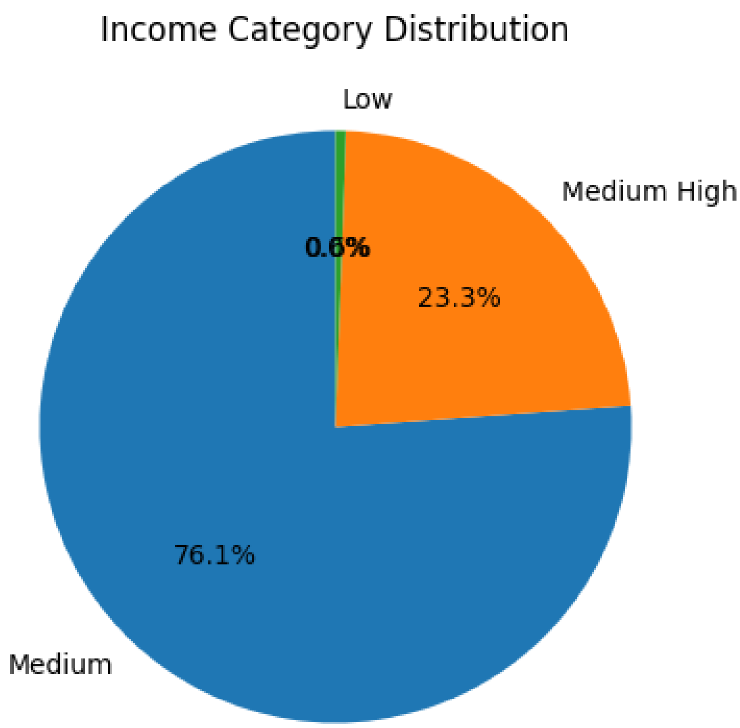
Usage_cat Distribution:

```
[68]: df['usage_cat'] = pd.cut(df['Usage'], bins=[0, 1, 2, 3, 7], labels=['Low',␣
       ↪'Medium', 'Medium High', 'High'])
      df['usage_cat'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,␣
       ↪title='Usage Category Distribution', figsize=(5, 5))
      plt.ylabel('')
      plt.show()
```



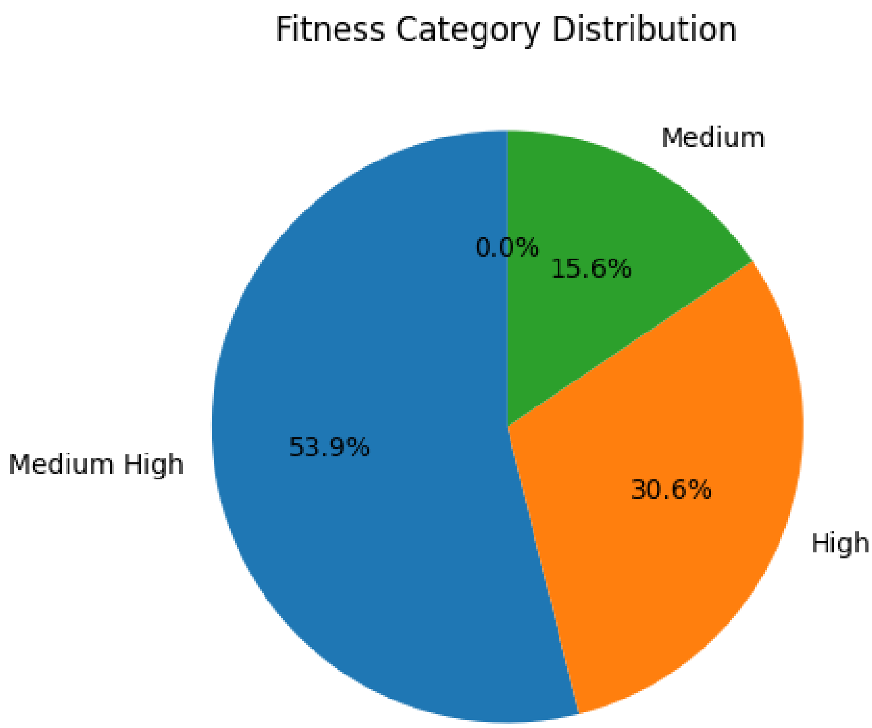
Income_cat Distribution:

[69]:
```
df['income_cat'] = pd.cut(df['Income'], bins=[0, 30000, 60000, 90000, 200000],␣
↪labels=['Low', 'Medium', 'Medium High', 'High'])
df['income_cat'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,␣
↪title='Income Category Distribution', figsize=(5, 5))
plt.ylabel('')
plt.show()
```



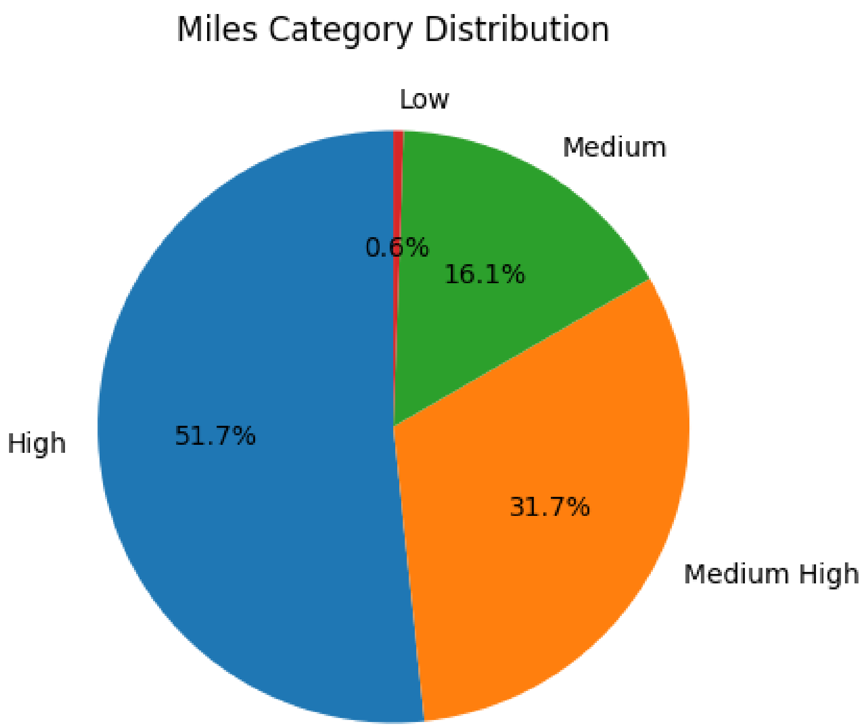
Fitness_cat Distribution:

[70]:
```
df['fitness_cat'] = pd.cut(df['Fitness'], bins=[0, 1, 2, 3, 5], labels=['Low',
 ↪'Medium', 'Medium High', 'High'])
df['fitness_cat'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,
 ↪title='Fitness Category Distribution', figsize=(5, 5))
plt.ylabel('')
plt.show()
```



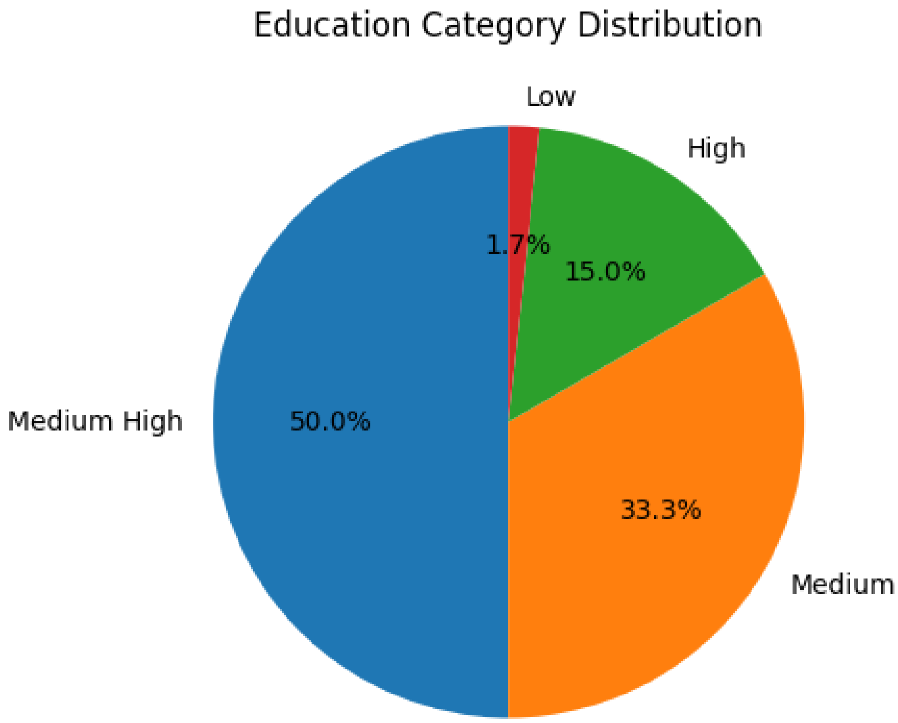
Miles_cat Distribution:

```
[71]: df['miles_cat'] = pd.cut(df['Miles'], bins=[0, 30, 60, 90, 200], labels=['Low',
      ↪'Medium', 'Medium High', 'High'])
      df['miles_cat'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,
      ↪title='Miles Category Distribution', figsize=(5, 5))
      plt.ylabel('')
      plt.show()
```



Education_cat Distribution:

[72]:
```
df['education_cat'] = pd.cut(df['Education'], bins=[0, 12, 14, 16, 20],␣
↪labels=['Low', 'Medium', 'Medium High', 'High'])
df['education_cat'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90,␣
↪title='Education Category Distribution', figsize=(5, 5))
plt.ylabel('')
plt.show()
```



Category-wise Explanation:

1. Gender:

What it shows: Proportion of male vs female customers.

Why it is useful: Helps you see which gender dominates the dataset and analyze if product preference is gender-biased.

2. Marital Status:

What it shows: Split between married and single customers.

Why it is useful: Reveals how many customers are married/single — this can influence fitness equipment preferences or budget.

3. Age Category (Age_cat):

What it shows: Distribution of customers across age groups (Low, Medium, Medium High, High).

Why it is useful: Identifies which age group is more active in purchasing. Important for targeted marketing and product design.

4. Usage Category (Usage_cat):

What it shows: Frequency of treadmill usage categorized (e.g., Low to High).

Why it is useful: Indicates how often customers use the treadmill. This can correlate with fitness goals and product type (e.g., basic vs. advanced).

5. Income Category (Income_cat):

What it shows: Income distribution of customers (Low to High).

Why it is useful: Helps in understanding purchasing power. Higher-income groups might prefer more premium models like KP781.

6. Fitness Category (Fitness_cat):

What it shows: Self-reported fitness level of customers.

Why it is useful: Allows you to see if more fit individuals buy higher-end models or how fitness level relates to usage/miles.

7. Miles Category (Miles_cat):

What it shows: Distance covered per week, categorized (Low to High).

Why it is useful: Indicates how actively a treadmill is used — might affect the wear and tear or product preference.

8. Education Category (Education_cat):

What it shows: Years of education grouped into categories.

Why it is useful: Sometimes education correlates with awareness about fitness and income, which in turn affects product choice.

Purpose of Pie Charts for Each Category:

Pie charts help us visualize the proportion of categories within a feature. They're especially useful when:

We want to understand which group is dominant.

We need to compare class balance across different categorical features.

We are doing exploratory data analysis (EDA) to assess distributions before modeling.

Business Insights from Non-Graphical and Visual Analysis:

Here's a detailed breakdown of the business insights derived from our analysis:

1. Comments on the range of attributes:

Product: There are three distinct products (KP281, KP481, KP781) with varying popularity, indicating a tiered product offering. KP281 is the most frequent product in the dataset.

Age: Ages range from 18 to 50, showing a diverse customer base spanning young adults to middle-aged individuals. The IQR analysis and clipping showed a small percentage of outliers on the higher end, which were handled.

Gender: The dataset includes both Male and Female customers, with a slightly higher number of males.

Education: Education levels range from 12 to 21 years. The IQR analysis and clipping identified a small percentage of outliers on the higher end.

MaritalStatus: Customers are categorized as either Partnered or Single, with a higher count of partnered individuals.

Usage: The planned usage per week ranges from 2 to 7 days. The IQR analysis and clipping showed a percentage of outliers on the higher end, suggesting some customers plan for very high usage.

Fitness: Self-rated fitness levels range from 1 to 5. The IQR analysis and clipping indicated a small percentage of outliers on the higher end.

Income: Income levels range from approximately 29,562 to 104,581. The IQR analysis and clipping revealed a notable percentage of outliers on the higher end, indicating a segment of high-income customers.

Miles: The planned miles per week range from 21 to 360. The IQR analysis and clipping showed a percentage of outliers on the higher end, suggesting some customers plan to cover very long distances.

2. Comments on the distribution of the variables and relationship between them:

Distribution of Variables:

Age: The distribution of Age appears to be somewhat skewed to the right (mean > median), indicating a tail of older customers, although the clipping of outliers has mitigated this somewhat.

Education: The distribution of Education is slightly skewed to the left (mean < median), suggesting a tail of customers with fewer years of education.

Usage, Fitness, Income, Miles: These variables all show varying degrees of right skewness (mean > median), which is expected as higher values are less frequent. The presence of outliers, particularly in Income and Miles, contributes to this skew.

Gender and MaritalStatus: These are categorical and their distributions are shown in the pie charts, indicating the proportion of each category in the dataset.

Relationship between Variables (from Correlation Heatmap and Pair Plot):

Strong Positive Correlations: There are strong positive correlations between Usage, Fitness, and Miles. This is intuitive: people who plan to use the treadmill more often, rate their fitness higher, and plan to cover more miles.

Moderate Positive Correlations: Income and Education show moderate positive correlations with each other, and also with Usage, Fitness, and Miles. This suggests that higher income and education levels are associated with higher planned usage, fitness levels, and mileage.

Income and Age: There is a moderate positive correlation between Income and Age, indicating that older customers tend to have higher incomes. Weak/No Correlations: Age shows weak correlations with Usage, Fitness, and Miles. Gender and MaritalStatus are not included in the numerical correlation matrix but their relationships with other variables are explored through count plots and heatmaps.

3. Comments for each univariate and bivariate plot:

Univariate Plots (Histograms with KDE and Pie Charts):

Histograms with KDE: These plots visually confirm the distributions observed from the mean/median comparison. They show the frequency of values for each numerical variable and the estimated probability density. The skewness in variables like Income and Miles is clearly visible.

Pie Charts:

The pie charts provide a clear visual representation of the proportion of customers in each category for Gender, Marital Status, and the categorical versions of Age, Usage, Income, Fitness, Miles, and Education. They highlight the dominant categories within each attribute. Bivariate Plots (Count Plots and Heatmaps):

Count Plots (Categorical vs Product):

These plots show the absolute counts of customers for each product within each category of the other variables. They are useful for seeing the raw numbers and comparing the popularity of products across categories. For example, the count plot for Gender vs Product clearly shows that KP281 and KP481 have a more balanced gender distribution compared to KP781, which is dominated by males.

Heatmaps (Categorical vs Product - Normalized):

These heatmaps visualize the conditional probabilities (or proportions) of product choice within each category. The intensity of the color indicates the likelihood of a customer in a given category choosing a particular product. These are excellent for quickly identifying strong associations. For instance, the heatmap for Income_cat vs Product clearly shows that customers in the 'High' income category are overwhelmingly likely to choose the KP781, while those in 'Low' income are more likely to choose KP281. Similarly, the heatmaps for Usage_cat, Fitness_cat, and Miles_cat vs Product strongly indicate that higher usage, fitness, and mileage are highly associated with the KP781.

Insights Summary:

1. KP281 is the entry-level product: Popular with younger, lower-to-medium income/education, and less active users.
2. KP481 is the mid-range option: Appeals to a similar demographic as KP281 but with slightly higher usage and mileage.
3. KP781 is the premium product: Strongly preferred by males, higher income/education, and highly active/fit users who cover significant mileage.
4. Income and Education are key drivers for KP781: Higher levels in these attributes strongly correlate with the purchase of the premium model.
5. Usage and Fitness levels predict product choice: High usage and fitness are strong indicators for KP781 purchase.
6. Gender influences KP781 purchase: Males are significantly more likely to buy the KP781 than females.
7. Marital Status is not a primary factor: The distribution of product choice is similar between partnered and single individuals.
8. Outliers exist and were handled: While clipping was used, investigating the nature of these outliers in a real-world scenario could provide further insights into niche customer segments.
9. Data distributions are mostly skewed: Variables like Income, Usage, Fitness, and Miles show right skewness, which is important to consider for modeling.

10. Strong correlations exist between activity-related variables: Usage, Fitness, and Miles are highly correlated, as expected.

Simple Action Items for the Business (Recommendations):

Here are straightforward steps your business can take based on the treadmill data analysis:

Know Your Treadmills and Who Buys Them:

KP281 (Entry-Level):

This one is popular with younger folks and those with average income and education. They don't use it super often or run very far. Action: Advertise this as an easy, affordable treadmill for beginners or casual use. Show young people enjoying it.

KP481 (Mid-Range):

Similar buyers to the KP281, but they might use it a bit more. Action: Market this as the next step up from the basic model, great for people getting a bit more serious about fitness but still want value.

KP781 (High-End):

Mostly bought by men who are fitter, earn more, are more educated, and use the treadmill a lot for long distances. Action: Promote this as a top-tier, high-performance machine for serious athletes. Focus ads on features for intense workouts and target wealthier, more educated men. Maybe partner with fitness experts.

Money and Education Matter for the Best Treadmill:

People with higher incomes and more education tend to buy the most expensive KP781. Action: For the KP781, talk about its quality and long-term value in ads aimed at wealthier people. For the KP281 and KP481, focus on the good price and what you get for your money for everyone else.

Match Treadmills to How People Want to Use Them:

Heavy users and very fit people love the KP781. Casual users stick to the KP281 and KP481. Action: When someone is thinking of buying, ask them how often they plan to use it and what their fitness goals are. Recommend the KP781 for serious runners and the other models for more casual use. Show different types of workouts for each treadmill in marketing.

Figure Out Why Fewer Women Buy the Top Treadmill:

More men than women buy the high-end KP781. Action: Try to understand what women might want in a high-end treadmill. Maybe the ads or features aren't appealing to them? Create marketing that speaks directly to women who are serious about fitness and show them using the KP781.

Don't Worry Too Much About If Someone is Married or Single:

Marital status doesn't really predict which treadmill someone buys. Action: You don't need to create separate ads or change features just because someone is married or single.

Look Into the Unusual Customers:

We saw a few customers who were very different from the rest (outliers). Action: Find out more about these unusual customers. Are they just odd cases, or is there a small group of customers with unique needs we don't know about yet?

In Simple Terms:

Advertise Smart: Tailor your ads for each treadmill to the people most likely to buy it.

Explain Clearly: Make it obvious what each treadmill is best for (beginner vs. pro).

Keep Improving: Listen to what customers say to make better treadmills in the future.

Sell Where They Shop: Figure out the best places to sell each treadmill based on who buys it.

Learn More: Talk to customers to understand what they really want and why they choose certain treadmills.