

- (By default, here,
axis = 0)

②

pd

- pd.isnull(df): • To check the null value in DF.
• prints the whole table but instead of the actual data, it prints 'True' for the cells/values having null values & 'False' for cells/values having non-null values.

NOT OPTIMAL

- pd.isnull(df).sum(): • prints the no. of null values ~~the~~ the column, after checking.
• print in both ways:
 table with 'T'/'F' no. of null values per column.
- pd.isnull(df).sum(): • no. of null values per column

- df.dropna(): • Drops the values in the column unlike the whole row/column by using df.drop().

OPTIMAL → df.dropna(inplace=True) • saves or implements the changes in the real dataset

both x same thing

- OPTIMAL** → df-test.dropna(inplace=True)
→ df-test = df-test.dropna()

because, df-test = ---- will create two dataframes thus take double memo. for the same thing.

- df['Amount'] = df['Amount'].astype('int')
- Changes the datatype
 - gets saved as 'Amount' जोरि उसका पहेले से नाम था।

then to check for the changes:

- df['Amount'].dtype : • checks the ~~X~~ datatype for the certain / respective column.

• df.dtypes : • checks datatypes for all the columns.

- df.columns : • gets the names of all the columns name with its dtype as object.
- dtype = 'object'

• It doesn't mean the DataFrame are of object type; it just means the column names themselves are stored as an object array.

• In Pandas, when an Index holds 'str', its dtype is 'obj' (the general purpose dtype in Numpy for text & mixed data.)

- df.rename(columns = {'Marital-Status': 'Shaadi'})

rename in
columns

to be change
or rename

Renamed
as that.

In a ~~list~~ dictionary

- to rename the column.
- NE save X एडिट without : inplace() & '=' operators

4

→ df.describe() : returns the description of the data in the DataFrame, including - count, mean, std, min, 25%, 50%, 75%, max.

Percentile.

• to observe for the useful insights.

→ df[['Age', 'Orders', 'Amount']].describe() :
• to describe specific columns.

df['Age'] ← '[]' (single square brackets represents - Series (a single column))

df[['Age', 'B']] ← '[[]] - (double square brackets represents - Dataframe.
(Subset with multiple columns)

→ • In 'count' that comes as the result by using df.describe() ⇒ counts the no. of values present under the particular Index = no. of rows.

• if count ≠ no. of rows.

• then ⇒ missing value that needs to be handled.

EDA - Exploratory Data Analysis.

Data cleaning + Summary + Visualization + Insights.

Exploratory Data Analysis.

→ (1) on the basis of: Gender

→ df.columns

(to use this so we do not error in writing the correct column names).

Seaborn shorthand.

→ ax = sns.countplot(x='Gender', data=df)

Variable
की संख्या है
अगर और

a plot
in
Seaborn.

axis x for value

gender - F/M

data to
be used

frequency value
will be plotted
of y-axis.

of the
var - 'df'.

उत्तर modify
करना है तो

otherwise skip

basic plot के

for skip using it.

∴ column to be in ' ' & var as norm

→ ax = sns.countplot(x='Gender', data=df)

for bars in ax.containers:

ax.bar_label(bars)

adds labels on top of
each bar inside the container,
showing their height (value)

6

→ groupby on the basis of 'Gender'.

df.groupby(['Gender'], as_index=False)['Amount']
sum().sort_values(by='Amount', ascending=False)
the amt. for each gender.

• means, 'Gender' column remains a regular column in the o/p instead of becoming the index.

• after grouping, ['Amount'] we are using this to perform calculations.

groupby → Select → Sum → Sort

→ **OPTIMAL** (If have to visualize)


• Sales = df.groupby(['Gender'], as_index=False)
['Amount'].sum().sort_values(by='Amount',
ascending=False)

this code creates a table showing the total sales (Amount) per gender, sorted for highest to lowest.

• Sns.barplot(x='Gender', y='Amount', data=Sales)

To remove: < Axes: xlabel='Gender', ylabel='Amount'

assign a
Variable

used  at the end of
the code line.

Put the Insights

∴ Hue must be a categorical column.

→ (2) on the basis of 'Age'

→ df.columns

→ Sns.countplot(data=df, x='Age Group', hue='Gender')
OPTIMAL

∴ Age + Gender^(Vs)

मैन कौन है और
फेमेल कौन है

(hue = Gender)

• using this, we get a legend thing on our plot.

• age group bar gets divided into the no. of genders.

• Sns.countplot(data=df, x='Age Group')

→ ax = Sns.countplot(data=df, x='Age Group', hue='Gender')

for bars in ax.containers:

ax.bar_label(bars)

• Same as above but with these numeric value on each bar.

Insights

∴ Age + Amount^(Vs)

both numeric:

group by func.

→ Sales age = df.groupby(['Age Group'], as_index=False)

['Amount'].sum().sort_values(by='Amount', ascending=False)

Sns.barplot(x='Age Group', y='Amount', data=Sales age)

Insights

⑧

→ (3) on the basis of State

∴ Total no. of orders State-wise (कितना है?)

→ ax = sns.countplot(data = df, y = 'State', hue = 'Orders')
for bars in ax.containers:
ax.bar_label(bars)

or,

ax = sns.countplot(data = df, y = 'State', hue = 'Orders')
for bars in ax.containers:
ax.bar_label(bars)

width, height

→ sns.set(rc = {'figure.figsize': (15, 9)})

a sns
func. to set the
default styling

runtime
configuration

key size
inside the

parameters & plots
like colors, sizes, font,
etc.

dict: controls the
overall size of the figure

(Total amount spent state wise)

→ ax = **OPTIMAL** →

→ Sales - Sales = df.groupby(['State'], as_index = False)
['Orders'].sum().sort_values(by = 'Orders',
ascending = False).head(6)

∴ groupby → sum → sort → head → now, barplot.
sns.set(rc = {'figure.figsize': (15, 5)})
sns.barplot(data = df, x = 'State', y = 'Orders')
Sales - States

9

∴ Total amount spend per state wise

→ salesa-state = df.groupby(['State'], as-index=False)
['Amount'].sum().sort_values(by='Amount',
ascending=False).head(6).

sns.set(rc = {'figure.figsize': (15, 5)})

sns.barplot(data=salesa-state, x='State', y=
'Amount')

insights !!

deg. Purchasing power किस state की ज्यादा है और
orders कहां के ज्यादा है।

*

→ (4) on the basis of Marital-status

no. of married & unmarried

∴ (MS & Orders)

→ ax = sns.countplot(data=df, x='Marital-status')
sns.set(rc={'figure.figsize': (7, 5)})
for bars in ax.containers:
ax.bar_label(bars).

and,

→ mar = df.groupby(['Marital-status'], as_index=False)
['Orders'].sum().sort_values(by='Orders'
ascending=False)]

sns.barplot(data=mar, x='Marital-status', y='Orders')

bars
of size
if previously
of size
use
of size
depend
hoga!

sns.countplot doesn't take values for 'x' & 'y'
like barplot.

∴ (MS & Amount)

→ mar-amt = df.groupby(['Marital-status'],
as_index=False)['Amount'].sum().sort_values
(by='Amount', ascending=False)]
sns.barplot(data=mar-amt, x='Marital-status',
y='Amount')

∴ Add ⇒ sns.set(rc={'figure.figsize': (7, 5)})
if to change the bars width & height.

∴ (MS + Gender & Amount)

→ mar-gen-amount = df.groupby(['Marital-Status', 'Gender'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)
sns.barplot(data=mar-gen-amount, x='Marital-Status', y='Amount', hue='Gender')



→ (5) on the basis of Occupation

∴ normal counterplot to count peeps in occupation

sns.set(rc={'figure.figsize': (20, 5)})
ax = sns.countplot(data=df, x='Occupation')
for bars in ax.containers:
ax.bar_label(bars)

∴ Occupation & Amt.

occ-amt = df.groupby(['Occupation'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)
sns.barplot(data=occ-amt, x='Occupation', y='Amount')



→ (6) on the basis of 'Product-Category'

```
sns.set(rc = {'figure.figsize': (20, 5)})  
ax = sns.countplot(data = df, x = 'Product-Category')  
for bars in ax.containers:  
ax.bar_label(bars)
```

∴ Product-Category to Amt
pro-amount = df.groupby(['Product-Category'], as_index =
False)['Amount'].sum().sort_values(by = 'Amount'
ascending = False)

```
sns.set(rc = {'figure.figsize': (40, 15)})  
sns.barplot(data = pro-amount, x = 'Product-Category', y =  
'Amount')
```

∴ Product-ID & Orders
Sales-state = df.groupby(['Product-ID'], as_index =
False)['Orders'].sum().sort_values(by = 'Orders'
ascending = False).head(9)

```
sns.set(rc = {'figure.figsize': (20, 5)})  
sns.barplot(data = Sales-state, x = 'Product-ID',  
y = 'Orders')
```

top 10 most sold products (same thing + but
लिखा अलग-अलग है)

```
fig1, ax1 = plt.subplots(figsize = (12, 7))  
df.groupby('Product-ID')['Orders'].sum().nlargest(10)  
sort_values(ascending = False).plot(kind = 'bar')
```

#

Conclusion

DOUBTS & ANSWERS

of PROJECT-01!

optional (depends on the location)

```
df = pd.read_csv(r"C:\Users\uswip\One drive\Desktop\PythonPractice\Diwali Sales Data.csv")
```

ERROR!

- Why is `r` being used here?
 - the '`r`' before the string makes it raw string.
 - '`r`' tells Python "treat this str literally & don't process backslashes as escape sequences."

Why ERROR?

It's UnicodeDecodeError - it happens because `pd.read_csv()` by default tries to read files using the '`utf-8`' encoding, & the file we used is probably saved with a diff. encoding.

'ISO-8859-1'

→ TO-DO:

- Use → `encoding="unicode_escape"` 'cp1252'
- `encoding='ISO-8859-1'`
- `encoding='cp1252'`
- `encoding_errors='ignore'`