```python
# === binning.ipynb ===
# just a convenience way to categorize your data w/o using if then else stmt
# a good example is converting numerical grades to letter grades

#%%
# Import Dependencies
import pandas as pd

#%%
raw_data = {
    'Class': ['Oct', 'Oct', 'Jan', 'Jan', 'Oct', 'Jan', 'Nov'],
    'Name': ["Cyndy", "Logan", "Laci", "Elmer", "Crystle", "Emmie", "test"],
    'Test Score': [90, 56, 72, 88, 98, 67, 59]}
df = pd.DataFrame(raw_data)
df

#%%
# Create the bins in which Data will be held
# Bins are 0, 60, 70, 80, 90, 100
bins = [0, 60, 70, 80, 90, 100]

# Create the names for the four bins
group_names = ["F", "D", "C", "B", "A"]

#%%
# ---------------------------------------------------------------------------
# When using the `pd.cut()` method, three parameters must be passed in:
#    1. the Series that is going to be cut
#    2. a list of the bins that the Series will be sliced into
#    3. a list of the names/values that will be given to the bins.
#
# It is important to note that, when creating the list for bins, Pandas will
# automatically determine the range between values. This means that, when
# given the list [0, 60, 70, 80, 90, 100], Pandas will create bins with
# ranges between those values in the list.
#
# The labels in the pd.cut() method must be one fewer element than those in an
# the bin (as the lowest possible value, otherwise an error will be returned:
#     `Bin labels must be one fewer than the number of bin edges`
# ---------------------------------------------------------------------------

df["Test Score Summary"] = pd.cut(df["Test Score"], bins, labels=group_names)
df


#%%
# What makes binning so powerful is that, after creating and applying these
# bins, the DataFrame can be grouped according to those values and thus a
# higher-level analysis can be conducted.

# Creating a group based off of the bins
df = df.groupby("Test Score Summary")
df.max()

#%%
```