```
import pandas as pd

df = pd.read_csv("C:/Users/KJCOEMR/Desktop/DC/Mall_Customers (3).csv")

df
```

```
     CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-
100)
0             1    Male   19                  15
39
1             2    Male   21                  15
81
2             3  Female   20                  16
6
3             4  Female   23                  16
77
4             5  Female   31                  17
40
..          ...     ...  ...                 ...                    .
..
195         196  Female   35                 120
79
196         197  Female   45                 126
28
197         198    Male   32                 126
74
198         199    Male   32                 137
18
199         200    Male   30                 137
83

[200 rows x 5 columns]
```

```
df['Spending Score (1-100)'].mean()
```

```
50.2
```

```
df['Age'].mean()
```

```
38.85
```

```
df['CustomerID'].mean()
```

```
100.5
```

```
df['Annual Income (k$)'].mean()
```

```
60.56
```

```
df.mean(axis=1)[0:4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[95], line 1
----> 1 df.mean(axis=1)[0:4]

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:11693, in DataFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11685 @doc(make_doc("mean", ndim=2))
  11686 def mean(
  11687     self,
   (...)
  11691     **kwargs,
  11692 ):
> 11693     result = super().mean(axis, skipna, numeric_only, **kwargs)
  11694     if isinstance(result, Series):
  11695         result = result.__finalize__(self, method="mean")

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (...)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs)
  12373 nv.validate_func(name, (), kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
  12379 )

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:11562, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwds)
  11558     df = df.T
  11560 # After possibly _get_data and transposing, we are now in the
  11561 #  simple case where we can use BlockManager.reduce
> 11562 res = df._mgr.reduce(blk_func)
```

```
  11563 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11564 if out_dtype is not None and out.dtype != "boolean":

File ~\anaconda3\Lib\site-packages\pandas\core\internals\
managers.py:1500, in BlockManager.reduce(self, func)
   1498 res_blocks: list[Block] = []
   1499 for blk in self.blocks:
-> 1500     nbs = blk.reduce(func)
   1501     res_blocks.extend(nbs)
   1503 index = Index([None])  # placeholder

File ~\anaconda3\Lib\site-packages\pandas\core\internals\
blocks.py:404, in Block.reduce(self, func)
    398 @final
    399 def reduce(self, func) -> list[Block]:
    400     # We will apply the function and reshape the result into a
single-row
    401     #  Block with the same mgr_locs; squeezing will be done at
a higher level
    402     assert self.ndim == 2
--> 404     result = func(self.values)
    406     if self.values.ndim == 1:
    407         res_values = result

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:11481, in
DataFrame._reduce.<locals>.blk_func(values, axis)
  11479         return np.array([result])
  11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File ~\anaconda3\Lib\site-packages\pandas\core\nanops.py:147, in
bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File ~\anaconda3\Lib\site-packages\pandas\core\nanops.py:404, in
_datetimelike_compat.<locals>.new_func(values, axis, skipna, mask,
**kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask,
**kwargs)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype,
fill_value=iNaT)

File ~\anaconda3\Lib\site-packages\pandas\core\nanops.py:719, in
nanmean(values, axis, skipna, mask)
```

```
    716     dtype_count = dtype
    718 count = _get_counts(values.shape, mask, axis,
dtype=dtype_count)
--> 719 the_sum = values.sum(axis, dtype=dtype_sum)
    720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):

File ~\anaconda3\Lib\site-packages\numpy\core\_methods.py:49, in
_sum(a, axis, dtype, out, keepdims, initial, where)
     47 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
     48          initial=_NoValue, where=True):
---> 49     return umr_sum(a, axis, dtype, out, keepdims, initial,
where)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
df['Spending Score (1-100)'].median()
```

```
50.0
```

```
df['Age'].median()
```

```
36.0
```

```
df['CustomerID'].median()
```

```
100.5
```

```
df['Annual Income (k$)'].median()
```

```
61.5
```

```
df.mode()
```

```
     CustomerID   Genre   Age  Annual Income (k$)  Spending Score (1-
100)
0             1  Female  32.0                54.0
42.0
1             2     NaN   NaN                78.0
NaN
2             3     NaN   NaN                 NaN
NaN
3             4     NaN   NaN                 NaN
NaN
4             5     NaN   NaN                 NaN
NaN
..          ...     ...   ...                 ...
...
195         196     NaN   NaN                 NaN
NaN
196         197     NaN   NaN                 NaN
NaN
```

```
197         198     NaN   NaN                  NaN
NaN
198         199     NaN   NaN                  NaN
NaN
199         200     NaN   NaN                  NaN
NaN

[200 rows x 5 columns]
```

```
df['Spending Score (1-100)'].mode()
```

```
0    42
Name: Spending Score (1-100), dtype: int64
```

```
df['Age'].mode()
```

```
0    32
Name: Age, dtype: int64
```

```
df['CustomerID'].mode()
```

```
0        1
1        2
2        3
3        4
4        5
      ...
195    196
196    197
197    198
198    199
199    200
Name: CustomerID, Length: 200, dtype: int64
```

```
df['Annual Income (k$)'].mode()
```

```
0    54
1    78
Name: Annual Income (k$), dtype: int64
```

```
df.min()
```

```
CustomerID                    1
Genre                    Female
Age                          18
Annual Income (k$)           15
Spending Score (1-100)        1
dtype: object
```

```
df['Spending Score (1-100)'].min()
```

```
1
```

```
df['Age'].min()
```

```
18
```

```
df['CustomerID'].min()
```

```
1
```

```
df['Annual Income (k$)'].min()
```

```
15
```

```
df.max()
```

```
CustomerID                 200
Genre                     Male
Age                         70
Annual Income (k$)         137
Spending Score (1-100)      99
dtype: object
```

```
df['Spending Score (1-100)'].max()
```

```
99
```

```
df['Age'].max()
```

```
70
```

```
df['CustomerID'].max()
```

```
200
```

```
df['Annual Income (k$)'].max()
```

```
137
```

```
df['Spending Score (1-100)'].std()
```

```
25.823521668370162
```

```
df['Age'].std()
```

```
13.969007331558883
```

```
df['CustomerID'].std()
```

```
57.879184513951124
```

```
df['Annual Income (k$)'].std()
```

```
26.264721165271254
```

```
df.groupby(['Genre'])['Age'].mean()
```

```
Genre
Female    38.098214
Male      39.806818
Name: Age, dtype: float64
```