Load the Data

```python
import pandas as pd

# Load with default settings
df = pd.read_csv('customer_data.csv')

# Check the shape
print(f"Shape: {df.shape}")
print(f"\nData types:\n{df.dtypes}")

# Measure memory usage
memory_before = df.memory_usage(deep=True).sum() / 1024**2  # Convert to MB
print(f"\nMemory usage (before optimization): {memory_before:.2f} MB")
```

```
Shape: (10000, 10)

Data types:
customer_id              int64
age                      int64
region                  object
customer_type           object
total_purchases          int64
total_spent            float64
avg_purchase_value     float64
satisfaction_score     float64
account_status          object
referral_source         object
dtype: object

Memory usage (before optimization): 2.90 MB
```

```python
# Select only needed columns
needed_columns = ['customer_id', 'age', 'region', 'customer_type', 'total_spent', 'satisfaction_score']

# Define optimal data types
dtype_map = {
    'customer_id': 'int32',
    'age': 'int8',
    'region': 'category',
    'customer_type': 'category',
    'total_spent': 'float32',
    'satisfaction_score': 'float32'
}

# Load with optimizations
df_optimized = pd.read_csv(
    'customer_data.csv',
    usecols=needed_columns,
    dtype=dtype_map
)

# Measure memory usage after optimization
memory_after = df_optimized.memory_usage(deep=True).sum() / 1024**2
print(f"\nMemory usage (after optimization): {memory_after:.2f} MB")

# Calculate improvement
improvement_pct = ((memory_before - memory_after) / memory_before) *
100
print(f"Memory improvement: {improvement_pct:.2f}%")

# Check the optimized DataFrame
print(f"\nOptimized shape: {df_optimized.shape}")
print(f"\nOptimized data types:\n{df_optimized.dtypes}")
print(f"\nFirst few rows:\n{df_optimized.head()}")
```

Optimize

```
# Select only needed columns
needed_columns = ['customer_id', 'age', 'region', 'customer_type',
'total_spent', 'satisfaction_score']

# Define optimal data types
dtype_map = {
    'customer_id': 'int32',
    'age': 'int8',
    'region': 'category',
    'customer_type': 'category',
    'total_spent': 'float32',
    'satisfaction_score': 'float32'
}

# Load with optimizations
df_optimized = pd.read_csv(
    'customer_data.csv',
    usecols=needed_columns,
    dtype=dtype_map
)

# Measure memory usage after optimization
memory_after = df_optimized.memory_usage(deep=True).sum() / 1024**2
print(f"\nMemory usage (after optimization): {memory_after:.2f} MB")

# Calculate improvement
improvement_pct = ((memory_before - memory_after) / memory_before) *
100
print(f"Memory improvement: {improvement_pct:.2f}%")
```

```python
# Check the optimized DataFrame
print(f"\nOptimized shape: {df_optimized.shape}")
print(f"\nOptimized data types:\n{df_optimized.dtypes}")
print(f"\nFirst few rows:\n{df_optimized.head()}")
```

```
Memory usage (after optimization): 0.14 MB
Memory improvement: 95.03%

Optimized shape: (10000, 6)

Optimized data types:
customer_id                int32
age                         int8
region                  category
customer_type           category
total_spent              float32
satisfaction_score       float32
dtype: object

First few rows:
   customer_id  age     region customer_type  total_spent
satisfaction_score
0            1   56  Northeast          Gold   246.130005
1.1
1            2   69  Northeast        Silver  7928.109863
3.5
2            3   46    Midwest        Bronze    20.570000
3.8
3            4   32  Southeast        Bronze  3439.129883
2.6
4            5   60       West      Platinum  4945.830078
1.7
```

Questions

Info to Submit Metric Your Result Memory usage before optimization 2.90 MB Memory usage after optimization 0.14 MB Memory reduction percentage 95.03 % Number of columns (before) shape: (10000, 10) Number of columns (after) shape:(10000, 6)