Noah Anderson

Module 3

6/21/2024

Question 1

In [3]:

```
import pandas as pd
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df1
```

Out[3]:

	Even	Odd
0	2	1
1	4	3
2	6	5

a.

In [5]:

```
# Indexing with iloc
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = df1.iloc[[0, 1], [0, 1]]
df2.iloc[0, 0] = 0
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

In this case indexing with `iloc` creates a copy of `df1` which explains why `df1` is unchanged when printed.

In [7]:

```
# Slicing
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = df1[0:2]
df2.iloc[0, 0] = 0
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     0    1
1     4    3
2     6    5
```

```
/var/folders/gw/q9_8w8jn0js6k5m5d98vhhcdzkx593/T/ipykernel_32314/2128375444.
py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df2.iloc[0, 0] = 0
```

In this case slicing creates a view of `df1` which explains why `df1` is changed when printed.

In [9]:

```
# Boolean Expression
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = df1[df1.Even < 6]
df2.iloc[0, 0] = 0
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

```
/var/folders/gw/q9_8w8jn0js6k5m5d98vhhcdzkx593/T/ipykernel_32314/1652574861.
py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df2.iloc[0, 0] = 0
```

In this case boolean indexing creates a copy when modified as seen by `df1` remaining unchanged when printed.

In [11]:

```
# Pandas Methods
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = (df1
  .query('Even < 6')
  .assign(Even = [0, 4])
)
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

In this case, indexing through query copies on modify as seen by `df1` being unchanged when printed.

For these use cases, `iloc`, boolean indexing, and querying all copy when indexed wheras slicing modifies in place altering the original data frame.

b.

In [14]:

```
# Indexing with iloc using .copy
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = df1.iloc[[0, 1], [0, 1]].copy()
df2.iloc[0, 0] = 0
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

In [15]:

```
# Slicing with .copy
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = df1[0:2].copy()
df2.iloc[0, 0] = 0
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

In [16]:

```
# Boolean Expression .copy
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = df1[df1.Even < 6].copy()
df2.iloc[0, 0] = 0
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

In [17]:
```
# Pandas Methods .copy
df1 = pd.DataFrame({'Even': [2, 4, 6], 'Odd': [1, 3, 5]})
df2 = (df1
  .query('Even < 6')
  .assign(Even = [0, 4])
).copy()
print(df2)
print(df1)
```

```
   Even  Odd
0     0    1
1     4    3
   Even  Odd
0     2    1
1     4    3
2     6    5
```

All the code chunks have consistent results when using `.copy` now copying on modification. All warnings have disappeared now.

Question 2

a.

In [21]:
```
churn = pd.read_csv("churn_modeling.csv")
```

b.

In [23]:
```
churn = churn.drop(columns = [ "RowNumber", "Surname", "Gender", "Age"])
```

c.

In [25]:
```
churn.isna().any()
```

Out[25]:
```
CustomerId         False
CreditScore        False
Geography          False
Tenure             False
Balance            False
NumOfProducts      False
HasCrCard          False
IsActiveMember     False
EstimatedSalary    False
Exited             False
dtype: bool
```

With all column values returning `FALSE` for `is.na`, we can conclude that there is no missing data.

d.

In [28]:
```
(churn
 .groupby(['Geography', 'Exited']) # Group by Geography and Exited
 .agg(average_estimate = ('Balance', 'mean')) # Calculate mean Balance
 .round(2) # Round to the nearest cent
 )
```

Out[28]:

		average_estimate
Geography	**Exited**	
France	**0**	60339.28
	1	71192.80
Germany	**0**	119427.11
	1	120361.08
Spain	**0**	59678.07
	1	72513.35

It does appear that there is a difference between customers who exited and those who did not and this does appear to also very by country. In Spain and France, those who exited had on average more than $\$10,000$ those who did not. In Germany the two categories are aproximately equal differing only by around $\$1,000$.

e.

In [58]:
```
# Read in new customer data

new_cust = pd.read_csv("new_customers.csv")

# Merge new customer data with the churn data nad filter out data where Exit
cust_merge = (pd.merge(new_cust, churn, 'left', 'CustomerId')
              .query('Exited != 0'))

exited_perc = cust_merge['Exited'].sum() / cust_merge.shape[0] * 100

exited_perc_round = round(exited_perc, 2)
print("The percent of customers who left and returned to the bank is",
      exited_perc_round, "%")
```

```
The percent of customers who left and returned to the bank is 0.86 %
```

In [ ]: