Exercise Set 3

Hina Arora

- **pandas** is a python library used for data manipulation and analysis; it has two key data structures - series objects and dataframes

Copy

Create a pd dataframe as below:

```
np.random.seed(10)
df = pd.DataFrame(
        np.random.randint(18, 100, size=(7,5)),
        columns=['age', 'conversations', 'friends', 'unk1', 'unk2'])
df
```

	age	conversations	friends	unk1	unk2
0	27	33	82	46	47
1	26	91	18	58	54
2	34	29	72	80	51
3	90	96	67	69	72
4	95	87	31	43	31
5	48	48	30	83	49
6	75	54	45	36	95

Question

(1) print the conversations column

(2) print the age and conversations column

(3) print row 2

(4) print rows 0 and 2

(5) print element at row 2 and conversations column - using chained indexing

(6) print element at row 2 and conversations column - using multi-dimensional indexing

(7) print elements at row 0 and row 2 and age column and conversation column - using chained indexing

(8) print elements at row 0 and row 2 and age column and conversation column - using multi-dimensional indexing

Answer

df['conversations']
df[['age', 'conversations’]]

df.loc[2]
df.loc[[0,2]]

df['conversations'][2]
df.loc[2, 'conversations’]

df[['age', 'conversations']].loc[[0,2]]
df.loc[[0,2], ['age', 'conversations']]

Question

Drop columns **friends**, **unk1** and **unk2** (so df no longer has these columns)

- how would you do this with using inplace?
- how would you do this without using inplace?

	age	conversations
0	27	33
1	26	91
2	34	29
3	90	96
4	95	87
5	48	48
6	75	54

Answer

```
# with inplace
df.drop(['friends', 'unk1','unk2'], axis=1, inplace=True)

# without inplace
df = df.drop(["friends', unk1','unk2'], axis=1)
```

Question

Create a **new column** called '**age-group**' and set it to age//10

- how would you do this without using apply and lambda?
- how would you do this with using apply and lambda?
- how would you do this with using apply and a custom function?

	age	conversations	age-group
0	27	33	2
1	26	91	2
2	34	29	3
3	90	96	9
4	95	87	9
5	48	48	4
6	75	54	7

Answer

```
# without apply and lambda
df['age-group'] = df['age']//10

# with apply and lambda
# df['age-group'] = df['age'].apply(lambda x: x//10)

# with apply and custom function
# def foo(x):
#     return x//10
# df['age-group'] = df['age'].apply(foo)

df
```

Question

Create a **new column** called '**gender**' in df with the following values:

- ['male', 'male', 'female', 'female', 'female', 'male', 'male']

	age	conversations	age-group	gender
0	27	33	2	male
1	26	91	2	male
2	34	29	3	female
3	90	96	9	female
4	95	87	9	female
5	48	48	4	male
6	75	54	7	male

Answer

```
df['gender'] =
        ['male', 'male', 'female', 'female', 'female', 'male', 'male']
df
```

Question

Return all samples in df that are
male and have > 50 conversations

	age	conversations	age-group	gender
1	26	91	2	male
6	75	54	7	male

Answer

```
df[
    (df['conversations'] > 50) &
    (df['gender']=='male’)
  ]
```

Question

Return all samples in df that are female or have age > 40

	age	conversations	age-group	gender
2	34	29	3	female
3	90	96	9	female
4	95	87	9	female
5	48	48	4	male
6	75	54	7	male

Answer

```
df[
        (df['gender']=='female') |
        (df[‘age'] > 40)
    ]
```

Question

Create **new dataframe** called **dfnew** which has the same data as df.
Then set the element at index 3 and column gender to MALE.

- while ensuring df doesn't change
- and while ensuring you don't get the SettingWithCopyWarning

dfnew

	age	conversations	age-group	gender
0	27	33	2	male
1	26	91	2	male
2	34	29	3	female
3	90	96	9	MALE
4	95	87	9	female
5	48	48	4	male
6	75	54	7	male

Answer

```
# using copy to ensure dfnew is not a view of df
# dfnew = df # this would give a view
dfnew = df.copy()

# using multi-dimensional indexing to avoid the SettingWithCopyWarning
# dfnew['gender'].loc[3] = 'MALE' # this would give SettingWithCopyWarning
dfnew.loc[3,'gender'] = 'MALE'

# df will not be updated with dfnew.loc[3,'gender'] = 'MALE'
df

# dfnew will be updated dfnew.loc[3,'gender'] = 'MALE'
dfnew
```

Copy

Create a pd dataframe as below:

```
np.random.seed(0)
ser1 = np.random.randint(-100, 100, 5)
ser2 = ser1 + np.random.random(5)
df = pd.DataFrame({'true':ser1, 'pred':ser2})
df
```

	true	pred
0	72	72.857946
1	-53	-52.152748
2	17	17.623564
3	92	92.384382
4	-33	-32.702465

Question

Create a new column with the squared error (true - pred)^2.
Then find mean squared error between true and predicted values.

	true	pred	sqerror
0	72	72.857946	0.736071
1	-53	-52.152748	0.717836
2	17	17.623564	0.388832
3	92	92.384382	0.147749
4	-33	-32.702465	0.088527

0.415802802922607 6

Answer

```
df['sqerror'] = (df['true'] - df['pred'])**2

mse = df['sqerror'].mean()
```

Question

Read kaggle train.**csv** data into dataframe called titanic

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
538	539	0	3	Risien, Mr. Samuel Beard	male	NaN	0	0	364498	14.5000	NaN	S
335	336	0	3	Denkoff, Mr. Mitto	male	NaN	0	0	349225	7.8958	NaN	S
626	627	0	2	Kirkland, Rev. Charles Leonard	male	57.0	0	0	219533	12.3500	NaN	Q
263	264	0	1	Harrison, Mr. William	male	40.0	0	0	112059	0.0000	B94	S
344	345	0	2	Fox, Mr. Stanley Hubert	male	36.0	0	0	229236	13.0000	NaN	S

Answer

```
titanic = pd.read_csv('train.csv')
titanic.sample(n=5)
```

Question

Create a **copy** of titanic and put it in a dataframe called df

Answer

```
df = titanic.copy()
```

Question

Get the column names of df (**columns**)

Get the #rows, #cols of df (**shape**)

Get info on df (**info()**)

Get basic stats on df (**describe()**)

Answer

df.columns

df.shape

df.info()

df.describe()

Question

The Cabin column on it's own isn't useful…
but the Deck (**1st character in Cabin**) might be.

Use **apply** and a **custom function** to create a Deck column.

- Your custom function will need to handle NaN values in Cabin (**pd.notna(x)**)
- Once done, use conditional selection with masking to look at a sample of rows where Cabin is not NaN to check transformation

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Deck
430	431	1	1	Bjornstrom-Steffansson, Mr. Mauritz Hakan	male	28.0	0	0	110564	26.5500	C52	S	C
345	346	1	2	Brown, Miss. Amelia "Mildred"	female	24.0	0	0	248733	13.0000	F33	S	F
195	196	1	1	Lurette, Miss. Elise	female	58.0	0	0	PC 17569	146.5208	B80	C	B
369	370	1	1	Aubart, Mme. Leontine Pauline	female	24.0	0	0	PC 17477	69.3000	B35	C	B
716	717	1	1	Endres, Miss. Caroline Louise	female	38.0	0	0	PC 17757	227.5250	C45	C	C

Answer

```
def getDeck(cabin):
    if pd.notna(cabin):
        return cabin[0]
    else:
        return np.nan

df['Deck'] = df['Cabin'].apply(getDeck)

df[df['Deck'].notna()].sample(n=5)
```

Question

The Name column on its own isn't useful…
but the Title (**2nd word in Name**) might be.

Use **apply** and a **custom function** to create a **Title** column.

- Your custom function will need to handle NaN values in Name
- Once done, use conditional selection with masking to look at a sample of rows where Name is not NaN to check transformation

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Deck	Title
433	434	0	3	Kallio, Mr. Nikolai Erland	male	17.0	0	0	STON/O 2. 3101274	7.1250	NaN	S	NaN	Mr.
228	229	0	2	Fahlstrom, Mr. Arne Jonas	male	18.0	0	0	236171	13.0000	NaN	S	NaN	Mr.
221	222	0	2	Bracken, Mr. James H	male	27.0	0	0	220367	13.0000	NaN	S	NaN	Mr.
150	151	0	2	Bateman, Rev. Robert James	male	51.0	0	0	S.O.P. 1166	12.5250	NaN	S	NaN	Rev.
852	853	0	3	Boulos, Miss. Nourelain	female	9.0	1	1	2678	15.2458	NaN	C	NaN	Miss.

Answer

```
def getTitle(name):
    if pd.notna(name):
        title = name.split()[1]
        return title
    else:
        return np.nan

df['Title'] = df['Name'].apply(getTitle)

df[df['Title'].notna()].sample(n=5)
```

Question

Get the frequency distribution of the various Titles.

Mr.	502
Miss.	179
Mrs.	121
Master.	40
Dr.	7
Rev.	6
y	4
Impe,	3
Planke,	3
Gordon,	2
Col.	2
Major.	2
Mlle.	2
Pelsmaeker,	1
Carlo,	1
der	1
Mulder,	1
Walle	1

Answer

```
df['Title'].value_counts()
```

Question

Drop the following columns from df
['Name', 'Ticket', 'Cabin’]

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck	Title
803	804	1	3	male	0.42	0	1	8.5167	C	NaN	Master.
34	35	0	1	male	28.00	1	0	82.1708	C	NaN	Mr.
546	547	1	2	female	19.00	1	0	26.0000	S	NaN	Mrs.
430	431	1	1	male	28.00	0	0	26.5500	S	C	Mr.
439	440	0	2	male	31.00	0	0	10.5000	S	NaN	Mr.

Answer

```
df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

df.sample(n=5)
```

Question

Get the amount of missing data for each column (count)

```
PassengerId      0
Survived         0
Pclass           0
Sex              0
Age            177
SibSp            0
Parch            0
Fare             0
Embarked         2
Deck           687
Title            0
dtype: int64
```

Answer

```
df.isna().sum()
```

Question

Get the amount of missing data for each column (percentage)

```
PassengerId    0.000000
Survived       0.000000
Pclass         0.000000
Sex            0.000000
Age            0.198653
SibSp          0.000000
Parch          0.000000
Fare           0.000000
Embarked       0.002245
Deck           0.771044
Title          0.000000
dtype: float64
```

Answer

```
df.isna().mean()
```

Question

Let's take care of missing Age data.
Let's create a new imputed column called impAge, which replaces missing Age values with mean of Age.

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck	Title	impAge
705	706	0	2	male	39.0	0	0	26.0000	S	NaN	Mr.	39.000000
611	612	0	3	male	NaN	0	0	7.0500	S	NaN	Mr.	29.699118
70	71	0	2	male	32.0	0	0	10.5000	S	NaN	Mr.	32.000000
194	195	1	1	female	44.0	0	0	27.7208	C	B	Mrs.	44.000000
762	763	1	3	male	20.0	0	0	7.2292	C	NaN	Mr.	20.000000

Answer

```
df[‘impAge'] = df[‘Age'].fillna(value=df['Age'].mean())

df.sample(5)
# df[df[‘Age'].isna()].head()
```

Question

Let's take care of missing Embarked data

- Get the frequency distribution of Embarked

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

Answer

```
df['Embarked'].value_counts()
```

Question

Use masking with conditional selection to examine the rows in df which have Embarked=Nan

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck	Title	impAge
61	62	1	1	female	38.0	0	0	80.0	NaN	B	Miss.	38.0
829	830	1	1	female	62.0	0	0	80.0	NaN	B	Mrs.	62.0

Answer

```
df[df['Embarked'].isna()]
```

Question

We could just drop the rows with missing Embarked data (how would you do this?)…
but instead, let's create a new imputed column called impEmbarked, which replaces missing Embarked values with 'X’.

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck	Title	impAge	impEmbarked
61	62	1	1	female	38.0	0	0	80.0	NaN	B	Miss.	38.0	X
829	830	1	1	female	62.0	0	0	80.0	NaN	B	Mrs.	62.0	X

Answer

```
# df = df.dropna(subset=['Embarked'])

df['impEmbarked'] = df['Embarked'].fillna(value='X’)

df[df['Embarked'].isna()].head()
```

Question

Similarly, let's create a new imputed column called impDeck, which replaces missing Deck values with 'X'

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck	Title	impAge	impEmbarked	impDeck
370	371	1	1	male	25.0	1	0	55.4417	C	E	Mr.	25.0	C	E
495	496	0	3	male	NaN	0	0	14.4583	C	NaN	Mr.	24.0	C	X
275	276	1	1	female	63.0	1	0	77.9583	S	D	Miss.	63.0	S	D
143	144	0	3	male	19.0	0	0	6.7500	Q	NaN	Mr.	19.0	Q	X
708	709	1	1	female	22.0	0	0	151.5500	S	NaN	Miss.	22.0	S	X

Answer

```
df['impDeck'] = df['Deck'].fillna(value='X’)

df.sample(5)
# df[df['Deck'].isna()].head()
```