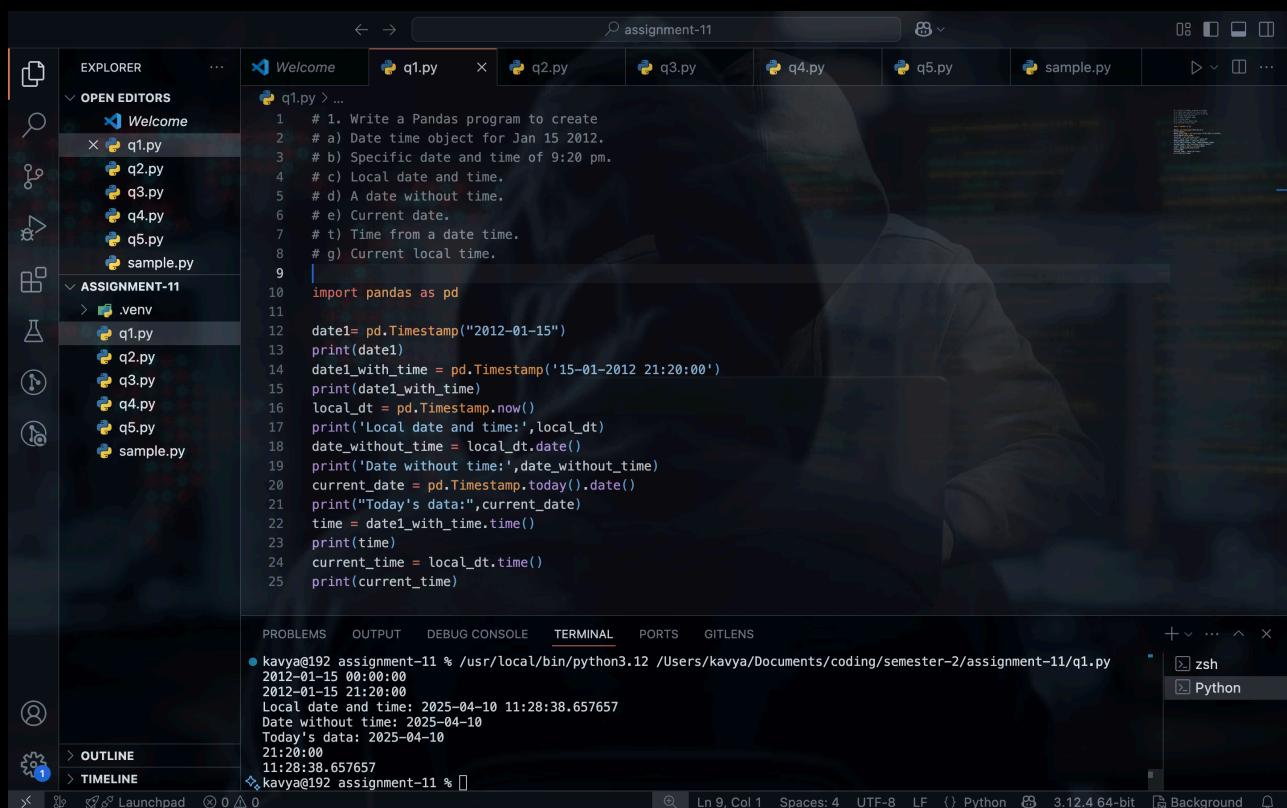


Assignment-11: U24AI038

```
# 1. Write a Pandas program to create  
# a) Date time object for Jan 15 2012.  
# b) Specific date and time of 9:20 pm.  
# c) Local date and time.  
# d) A date without time.  
# e) Current date.  
# t) Time from a date time.  
# g) Current local time.
```

```
import pandas as pd  
  
date1= pd.Timestamp("2012-01-15")  
print(date1)  
date1_with_time = pd.Timestamp('15-01-2012 21:20:00')  
print(date1_with_time)  
local_dt = pd.Timestamp.now()  
print('Local date and time:',local_dt)  
date_without_time = local_dt.date()  
print('Date without time:',date_without_time)  
current_date = pd.Timestamp.today().date()  
print("Today's data:",current_date)  
time = date1_with_time.time()  
print(time)  
current_time = local_dt.time()  
print(current_time)
```

Output:



The screenshot shows a dark-themed code editor interface with the following details:

- EXPLORER:** Shows the project structure under "OPEN EDITORS" and "ASSIGNMENT-11". The files q1.py, q2.py, q3.py, q4.py, q5.py, and sample.py are listed.
- EDITOR:** The code for q1.py is displayed. It contains the provided Python script to demonstrate Pandas timestamp operations.
- TERMINAL:** The terminal tab shows the execution of the script and its output:

```
kavya@192 assignment-11 % /usr/local/bin/python3.12 /Users/kavya/Documents/coding/semester-2/assignment-11/q1.py  
2012-01-15 00:00:00  
2012-01-15 21:20:00  
Local date and time: 2025-04-10 11:28:38.657657  
Date without time: 2025-04-10  
Today's data: 2025-04-10  
21:20:00  
11:28:38.657657  
kavya@192 assignment-11 %
```
- STATUS BAR:** Shows the current file is q1.py, and other details like Ln 9, Col 1, Spaces: 4, etc.

```

# Write a Pandas program to convert all the string values to
upper, lower cases in a given
# pandas series. Also find the length of the string values.
# s = pd.Series(['X', 'Y', 'T', 'Aaba', 'Baca', 'CABA', None,
'bird', 'horse', 'dog'])

import pandas as pd

def converterU(s):
    new_series = pd.Series([(str(ele).upper(), len(str(ele))) for
ele in s])
    return new_series

def converterL(s):
    new_series = pd.Series([(str(ele).lower(), len(str(ele))) for
ele in s])
    return new_series

s = pd.Series(['X', 'Y', 'T', 'Aaba', 'Baca', 'CABA', None, 'Bird',
'horse', 'dog'])
print('Lower cased ')
print(converterL(s))
print('\nUpper cased')
print(converterU(s))

```

Output:

The screenshot shows a dark-themed code editor interface with the following details:

- Explorer View:** Shows files in the current workspace, including `Welcome`, `q1.py`, `q2.py` (the active file), `q3.py`, `q4.py`, `q5.py`, and `sample.py`. It also lists a folder named `ASSIGNMENT-11` containing `.venv` and several other Python files.
- Code Editor:** The main area displays the Python code provided in the question. The code defines two functions, `converterU` and `converterL`, which take a pandas Series as input and return new Series with converted string values and their lengths. It also prints the original Series and the results of the conversion functions.
- Terminal:** The bottom right corner shows a terminal window with the following output:

```

1      (Y, 1)
2      (T, 1)
3      (AABA, 4)
4      (BACA, 4)
5      (CABA, 4)
6      (NONE, 4)
7      (BIRD, 4)
8      (HORSE, 5)
9      (DOG, 3)

```

- Status Bar:** The bottom status bar indicates the current line (Ln 20), column (Col 1), and file (Python 3.12.4 64-bit).

```
# After accidentally leaving an ice chest of fish and shrimp in
your car for a week while you
# were on vacation, you're now in the market for a new vehicle.
Your insurance didn't cover
# the loss, so you want to make sure you get a good deal on your
new car.
# Given a Series of car asking_prices and another Series of car
fair_prices, determine which
# cars for sale are a good deal. In other words, identify cars
whose asking price is less than
# their fair price.
# The result should be a list of integer indices corresponding to
the good deals
# in asking_prices.
import pandas as pd
def good_deals(asking_price,fair_price):
    ans = pd.Series([i for i in range(len(fair_price)) if
asking_price[i] < fair_price[i]])
    return ans
```

```
# Asking prices of cars (seller's price)
asking_prices = pd.Series([18000, 22000, 25000, 30000, 27000,
32000, 15000, 21000, 28000, 35000])

# Fair market prices of cars (estimated real value)
fair_prices = pd.Series([19000, 21000, 26000, 29000, 28000, 33000,
16000, 22000, 29000, 34000])
deals =good_deals(asking_prices,fair_prices)

data =pd.DataFrame({'asking price':asking_prices,'fair
price':fair_prices })
print(data[data['asking price'] <data['fair price']])
```

Output:

The screenshot shows a dark-themed instance of Visual Studio Code. In the Explorer sidebar, there are several files listed under 'OPEN EDITORS' and 'ASSIGNMENT-11'. The file 'q3.py' is currently selected and open in the main editor area. The code in 'q3.py' is as follows:

```
1 # After accidentally leaving an ice chest of fish and shrimp in your car for a week while you
2 # were on vacation, you're now in the market for a new vehicle. Your insurance didn't cover
3 # the loss, so you want to make sure you get a good deal on your new car.
4 # Given a Series of car asking_prices and another Series of car fair_prices, determine which
5 # cars for sale are a good deal. In other words, identify cars whose asking price is less than
6 # their fair price.
7 # The result should be a list of integer indices corresponding to the good deals
8 # in asking_prices.
9 import pandas as pd
10 def good_deals(asking_price,fair_price):
11     ans = pd.Series([i for i in range(len(fair_price)) if asking_price[i] < fair_price[i]])
12     return ans
13
14
15
16 # Asking prices of cars (seller's price)
17 asking_prices = pd.Series([18000, 22000, 25000, 30000, 27000, 32000, 15000, 21000, 28000, 35000])
18
19 # Fair market prices of cars (estimated real value)
20 fair_prices = pd.Series([19000, 21000, 26000, 29000, 28000, 33000, 16000, 22000, 29000, 34000])
21 deals = good_deals(asking_prices,fair_prices)
22
23 data =pd.DataFrame({'asking price':asking_prices,'fair price':fair_prices })
24 print(data[data['asking price'] < data['fair price']])
25
26
```

Below the editor, the 'TERMINAL' tab is active, showing the following output:

```
asking price    fair price
0      18000      19000
2      25000      26000
4      27000      28000
5      32000      33000
6      15000      16000
7      21000      22000
8      28000      29000
```

The status bar at the bottom indicates the terminal is running on 'kavya@192 assignment-11 %'.

```
# Whenever your friends John and Judy visit you together, y'all have a party. Given a
```

```
# DataFrame with 10 rows representing the next 10 days of your schedule and whether John
```

```
# and Judy are scheduled to make an appearance, insert a new column
```

```
# called days_til_party that indicates how many days until the next party.
```

```
# days_til_party should be 0 on days when a party occurs, 1 on days when a party doesn't
```

```
# occur but will occur the next day, etc.
```

```
import pandas as pd
```

```
# Sample data for 10 days
```

```
data = {
    'day': pd.date_range(start='2025-04-05', periods=10,
freq='D'),
    'John': [True, False, True, False, True, False, False,
True, False, True],
    'Judy': [True, True, False, False, True, False, True, True,
False, False]
}
```

```

party_days = []
for i in range(10):
    if data['John'][i] == True and data['Judy'][i] == True:
        party_days.append(True)
    else:
        party_days.append(False)

print(party_days)

days_til_party =[x*0 for x in range(10)]
# 

for i in range(10):
    if party_days[i] == True:
        continue
    else:
        j =0
        k=i
        no_party = 1

        if party_days[k] == False:
            while (k <10):
                if party_days[k] == True:
                    no_party=0
                k+=1

            k=i
            if no_party == 0:
                while(k <10 and party_days[k] == False):
                    j+=1
                    k+=1
            else:
                j =-1
        days_til_party[i] = j

print(days_til_party)

data.update({"days_til_Party":days_til_party})
df = pd.DataFrame(data)
print(df)

```

Output:

A screenshot of the Visual Studio Code (VS Code) interface. The title bar says "assignment-11". The left sidebar shows the "EXPLORER" view with files like "Welcome", "q1.py", "q2.py", "q3.py", "q4.py" (selected), "q5.py", and "sample.py". Below that is the "ASSIGNMENT-11" folder containing ".venv", "q1.py", "q2.py", "q3.py", "q4.py" (selected), "q5.py", and "sample.py". The main editor area displays the content of "q4.py". The code uses pandas to generate a DataFrame for 10 days, checking for both John and Judy's presence each day. The terminal tab shows the output of running the script, which is a table of dates, John's availability, Judy's availability, and the count of parties. The status bar at the bottom indicates the file is "kavyaa@192 assignment-11" and shows the current line and column as "Ln 14, Col 81".

```
1 # Whenever your friends John and Judy visit you together, y'all have a party. Given a
2 # DataFrame with 10 rows representing the next 10 days of your schedule and whether John
3 # and Judy are scheduled to make an appearance, insert a new column
4 # called days_til_party that indicates how many days until the next party.
5 # days_til_party should be 0 on days when a party occurs, 1 on days when a party doesn't
6 # occur but will occur the next day, etc.
7
8 import pandas as pd
9
10 # Sample data for 10 days
11 data = {
12     'day': pd.date_range(start='2025-04-05', periods=10, freq='D'),
13     'John': [True, False, True, False, True, False, False, True, False, True],
14     'Judy': [True, True, False, False, True, False, True, True, False, False]
15 }
16
17 party_days = []
18 for i in range(10):
19     if data['John'][i] == True and data['Judy'][i] == True:
20         party_days.append(True)
21     else:
22         party_days.append(False)
23
24 print(party_days)
25
```

day	John	Judy	party_days
2025-04-07	True	False	2
2025-04-08	False	False	1
2025-04-09	True	True	0
2025-04-10	False	False	2
2025-04-11	False	True	1
2025-04-12	True	True	0
2025-04-13	False	False	-1
2025-04-14	True	False	-1

```
# Given a dataset of concerts, count the number of concerts per
# (artist, venue), per year
# month. Make the resulting table be a wide table – one row per
# year month with a column
# for each unique (artist, venue) pair. Use the cross product of
# the artists and venues Series
# to determine which (artist, venue) pairs to include in the
# result.
```

```
import pandas as pd

# Sample concert data
concerts_data = {
    'date': [
        '2024-01-05', '2024-01-10', '2024-01-15',
        '2024-02-03', '2024-02-10', '2024-02-20',
        '2024-03-01'
    ],
    'artist': [
        'Taylor Swift', 'Ed Sheeran', 'Taylor Swift',
        'Taylor Swift', 'Ed Sheeran', 'BTS',
        'BTS'
    ],
    'venue': [
```

```

        'MSG', 'MSG', 'MSG',
        'Staples Center', 'MSG', 'Staples Center',
        'MSG'
    ]
}

concerts_df = pd.DataFrame(concerts_data)

# Convert date column to datetime
concerts_df['date'] = pd.to_datetime(concerts_df['date'])

# Sample artists and venues as Series
artists = pd.Series(['Taylor Swift', 'Ed Sheeran', 'BTS'],
name='artist')
venues = pd.Series(['MSG', 'Staples Center'], name='venue')

# print(concerts_df)

year_month = []
for date in concerts_df['date']:
    month = date.month
    year = date.year
    ym = str(year) + '-' + str(month)
    year_month.append(ym)

year_month1 = pd.to_datetime(year_month)
# print(year_month1)

artist_venue_pairs = pd.MultiIndex.from_product(
    [artists, venues], names=['artist', 'venue']
).to_frame(index=False)

# print(artist_venue_pairs)

# Add year_month column to concerts_df
concerts_df['year_month'] = year_month

# Group by year_month, artist, and venue, and count concerts
grouped = concerts_df.groupby(['year_month', 'artist',
'venue']).size().reset_index(name='concert_count')

# Pivot the grouped data into a wide format
wide_table = grouped.pivot(index='year_month', columns=['artist',
'venue'], values='concert_count')

# Ensure all (artist, venue) pairs are included
wide_table =
wide_table.reindex(columns=pd.MultiIndex.from_frame(artist_venue_p
airs), fill_value=0)

# Fill missing values with zeros
wide_table = wide_table.fillna(0)

```

```
print(wide_table)
```

Output:

The screenshot shows a dark-themed code editor interface with multiple tabs open. The active tab is 'q5.py'. The code in 'q5.py' is as follows:

```
1 # Given a dataset of concerts, count the number of concerts per (artist, venue), per year
2 # month. Make the resulting table be a wide table – one row per year month with a column
3 # for each unique (artist, venue) pair. Use the cross product of the artists and venues Series
4 # to determine which (artist, venue) pairs to include in the result.
5
6 import pandas as pd
7
8 # Sample concert data
9 concerts_data = {
10     'date': [
11         '2024-01-05', '2024-01-10', '2024-01-15',
12         '2024-02-03', '2024-02-10', '2024-02-20',
13         '2024-03-01'
14     ],
15     'artist': [
16         'Taylor Swift', 'Ed Sheeran', 'Taylor Swift',
17         'Taylor Swift', 'Ed Sheeran', 'BTS',
18         'BTS'
19     ],
20     'venue': [
21         'MSG', 'MSG', 'MSG',
22         'Staples Center', 'MSG', 'Staples Center',
23         'MSG'
24     ]
25 }
```

The terminal below the code shows the command run and the resulting wide table:

```
9 2025-04-14 True False -1
kavya@192 assignment-11 % /usr/local/bin/python3.12 /Users/kavya/Documents/coding/semester-2/assignment-11/q5.py
artist    Taylor Swift      Ed Sheeran      BTS
venue    MSG Staples Center      MSG Staples Center      MSG Staples Center
year_month
2024-1        2.0       0.0       1.0       0  0.0       0.0
2024-2        0.0       1.0       1.0       0  0.0       1.0
2024-3        0.0       0.0       0.0       0  1.0       0.0
```