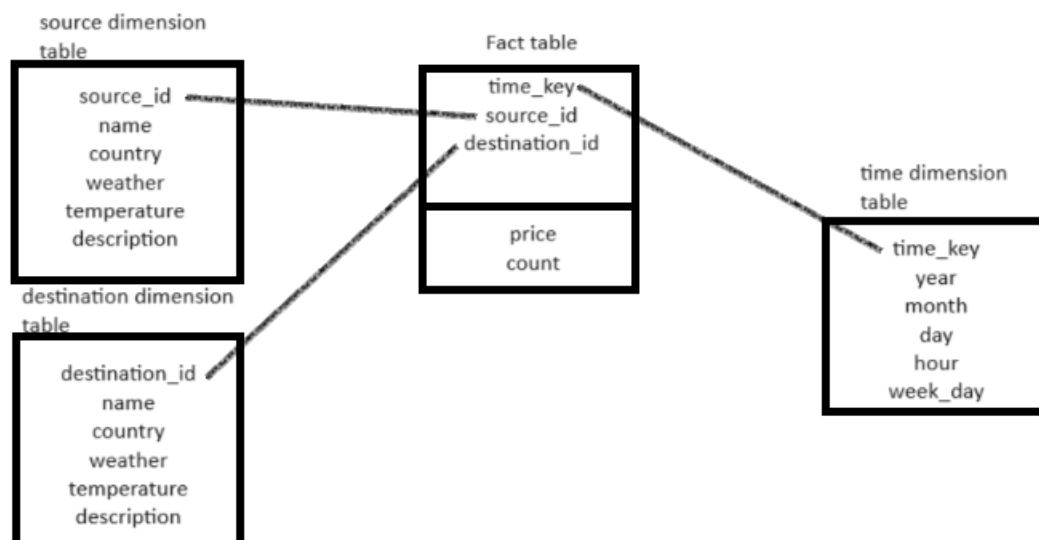


بخش تشریحی

سوال اول)

الف) *منظور از name همان city است.



ب) ابتدا تهران به میلان در خرداد ۱۴۰۲ :

- roll-up بر روی مقصد تا لول name و دایس بر روی میلان
- Roll-up بر روی مبدا تا لول name و دایس بر روی تهران
- Slice بر روی تایم یکی برای خرداد ۱۴۰۲ و یکی هم برای فروردین ۱۴۰۱
- Drill-down تا ماه (در بعد تایم)
- Aggregate بر روی price

سوال ۲)

الف) چون هدف مینیمايز کردن سايز اوليه است پس ابتدا B و C که کوچکترین ابعاد هستند، سپس B و A و در نهایت C و A.

ب)

BC شامل ۱۰۰ * ۱۰۰۰ سلول

BA شامل $100 * 1000000$ سلول

CA شامل $1000 * 1000000$ سلول

که مجموع آن ها را در ۴ بایت ضرب میکنیم (طبق گفته سوال سائز هر سلول ۴ بایت است)

که اگر این جمع و ضرب را انجام بدهیم به $4,400,400,000$ بایت میرسیم که حدود ۴۲۰۰ مگابایت است.

ج) اگر یگیت بر روی بعدی که نمیخواهیم بهترین روش خواهد بود، منظور جمع میانگین یا ... است برای مثال از کیوبید BA اگر یگیت میکنیم بر روی B (مثلا جمع یا میانگین بر روی کل B ها به ازای هر A) که در همین روش هم بهترین کار، کوچک کردن تعداد عملیات ها تا حد ممکن است مثلا برای A میتوانیم از BA یا CA استفاده کنیم که B در اینجا با توجه به سائز کوچکتر ارجحیت دارد(همین راه حل را میتوان برای B و C هم اعمال کرد)

سوال ۳

(الف)

در اینجا ۲ به توان n خواهد بود که با توجه به ۹ بودن میشود 2^{89}

(ب)

برای هر سلول این تعداد برابر ۲ به توان n منهای یک میشود (بدون خودش) از اونجایی که دو تا بیس داریم میشود $(2^9 - 1) * 2$ ولی باید اورلپ ها را هم کم کنیم که میشود ۳ سلول که حالت های مختلف آن میشود 2^3 پس در کل میشود : $2^3 - (2^9 - 1) * 2$

(ج)

دو تا بیس و یکی هم این سلول بسته اگر یگیت شده : $(_ , a2, _ , a4, a5, _ , a7, a8, _)$

(د)

اگر سه بعد مشترک را در نظر بگیریم، که بتوانیم شرط را هم ارضا کنیم 2^{83} داریم که کلا ۸ تا میشود.

سوال چهارم)

الف) برای محاسبه ترتیب بهینه لازم است کاردینالیتی هر بعد را حساب کنیم (که تعداد مقادیر منحصر به فرد هر بعد میشود)

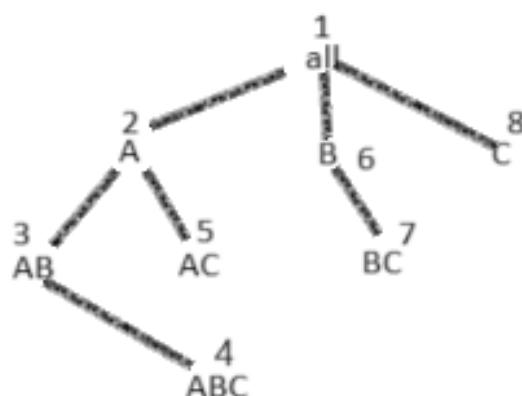
- Gender : ۲
- Education : ۳
- Occupation : ۴

حال بر مبنای این مقادیر ترتیب اپتیمال به صورت ۱. شغل ۲. تحصیلات ۳. جنسیت میشود.

دلایل هم برای این روش بدین صورت است که باعث بهبود پرون کردن میشود به این صورت که فضای جستجو زودتر پرون شود چون ترکیب آن ها مثلا با بعدهای دیگر شرط مینیمم ساپورت را رعایت نمیکند زودتر حذف میشوند .

ب)

با فرض اینکه $A=occupation$ ، $B=education$ و $C=gender$ به این درخت میرسیم :



حال برای هر کدام :

A: استاد: ۳ ، برنامه نویس : ۴

AB: برنامه نویس و دانشجو : ۳

B: دانشجو ۵ ، دبیرستان : ۳

BC: دانشجو و زن : ۳

C: زن : ۵ ، مرد : ۵

بخش عملی

(بر روی colab ران میشود و تنها لازم است فایل زیپ دیتاست در کنار فایل آپلود شود)

سوال یک)

در این بخش ابتدا پیش پردازش را کامل طبق ۱۲ مرحله تمرین اول انجام میدهیم، در ادامه داک و کد این بخش ها موجود است.

۱. ابتدا در رابطه با مشکلات ادغام کردن فایل ها و سپس راه حل های آن میپردازیم، همانطور که در pdf هم اشاره شد اصلی ترین مشکل تفاوت های موجود در نام ستون ها بود، مشکل بعدی هم وجود یک سری پارامترها در یک سری فایل ها بود که در دیگر ایستگاه ها حضور نداشتند.

با نگاه دقیق به فایل ها دریافتیم که این تفاوت ها معمولاً آنچنان قوی نیستند که نام پارامتر را به طور کلی تغییر دهند مثلاً صرفاً _ اضافه شده است یا حروف اول کلمات بزرگ هستند یا بین بخش های مختلف . گذاشته است، که این موارد به راحتی به کمک پایتون قابل حل است (اگر تفاوت در نام ها هم موجود بود دو روش میتوانستیم استفاده کنیم ۱. یک فانکشن استخراجگر بنویسیم که مثلاً به دنبال تکه temp و ۳ برود که هر ترکیبی از آن میتواند بیانگر دما در ساعت ۳ باشد ۲. یک فانکشن تعریف کنیم برای تشخیص نزدیکی کلمات بدین معنی که نزدیک ترین کلمه در فایل های مختلف یک ستون را تشکیل دهند و برای فاصله بین کلمات هم میتوانیم مثلاً از cosine similarity استفاده کنیم.) ولی الان با استفاده از فانکشن های زیر که ابتدا همه را lower case کنیم سپس _ و . را حذف کنیم مشکلمان حل میشود :

```

import pandas as pd
import os

def standardize_column_names(df):
    df.columns = df.columns.str.lower()
    df.columns = df.columns.str.replace(' ', '')
    df.columns = df.columns.str.replace('_', '')
    df.columns = df.columns.str.replace('.', '')
    return df

```

در گام بعدی برای هندل کردن ستون های متفاوت در ایستگاه های متفاوت تنها موقع مرج کردن دیتافریم ignore_index را True میگذاریم که دیتافریم جدید را دوباره index گذاری میکند و ستون های اضافی هم هندل میشود.

```

def standardize_column_names(df):
    df.columns = df.columns.str.lower()
    df.columns = df.columns.str.replace(' ', '')
    df.columns = df.columns.str.replace('_', '')
    df.columns = df.columns.str.replace('.', '')
    return df

def merge_excel_files(directory):
    merged_df = pd.DataFrame()

    for filename in os.listdir(directory):
        if filename.endswith(".csv"):
            df = pd.read_csv(os.path.join(directory, filename))

            df = standardize_column_names(df)

            merged_df = pd.concat([merged_df, df], ignore_index=True)

    return merged_df

merged_df = merge_excel_files('weatherAUS/')
merged_df.to_excel("combined_data.xlsx", index=False)

```

کد را هم از این جا به بعد روی google colab ران میکنیم چون حجم محاسبات زیاد شده است و میتوان از GPU TPU استفاده کرد.

در مورد توضیح کد هم ابتدا تمامی فایل های csv را میخواند نام ستون ها را نرمال میکند و سپس با کمک concat آنها را با هم ترکیب میکند و در نهایت در یک فایل excel ذخیره میکند خروجی هم ساختاری مانند شکل زیر دارد:

colab.research.google.com/drive/1wSypDbPmCIVw86H0EWq4cLcDaN6RODN#scrollTo=Vli9MxpwC9ju

DM_HW1.ipynb

File Edit View Insert Runtime Tools Help

Files

- sample_data
- weatherAUS
- combined_data.xlsx
- weatherAUS.zip

```
[6] <ipython-input-6-d0a4871e8ca3>:8: FutureWarning: The default value of regex will change from True to False in a future version. In addition, s
df.columns = df.columns.str.replace('.', '')

merged_df.head(5)
```

	windgustdir	temp9am	windspeed9am	raintomorrow	winddir9am	date	temp3pm	windspeed3pm	location	winddir3pm	humidity3pm	rainfall
0	ESE	21.7	30.0	No	E	2013-03-01	28.4	24.0	Uluru	E ...	54.0	0.8
1	E	24.6	22.0	No	E	2013-03-02	31.3	11.0	Uluru	N ...	33.0	0.0
2	E	27.6	24.0	No	ENE	2013-03-03	34.5	13.0	Uluru	SSE ...	27.0	0.0
3	ENE	28.7	28.0	No	E	2013-03-04	35.4	13.0	Uluru	SSE ...	22.0	0.0
4	S	29.9	20.0	No	E	2013-03-05	37.3	19.0	Uluru	S ...	21.0	0.0

5 rows x 27 columns

0s completed at 12:32 PM

۳. به کمک dtypes این کار را انجام می‌دهیم :

colab.research.google.com/drive/1wSypDbPmCIVw86H0EWq4cLcDaN6RODN#scrollTo=1wHptU0HsWW

DM_HW1.ipynb

File Edit View Insert Runtime Tools Help

Files

- sample_data
- weatherAUS
- combined_data.xlsx
- weatherAUS.zip

```
print(merged_df.dtypes)
```

```
windgustdir      object
temp9am          float64
windspeed9am     float64
raintomorrow     object
winddir9am       object
date             object
temp3pm          float64
windspeed3pm     float64
location         object
winddir3pm       object
pressure9am      float64
cloud3pm         float64
mintemp          float64
maxtemp          float64
pressure3pm      float64
windgustspeed    float64
raintoday        object
humidity3pm      float64
rainfall         float64
humidity9am      float64
cloud9am         float64
evaporation      float64
mintempf         float64
temp3pmf         float64
temp9amf         float64
maxtempf         float64
sunshine         float64
dtype: object
```

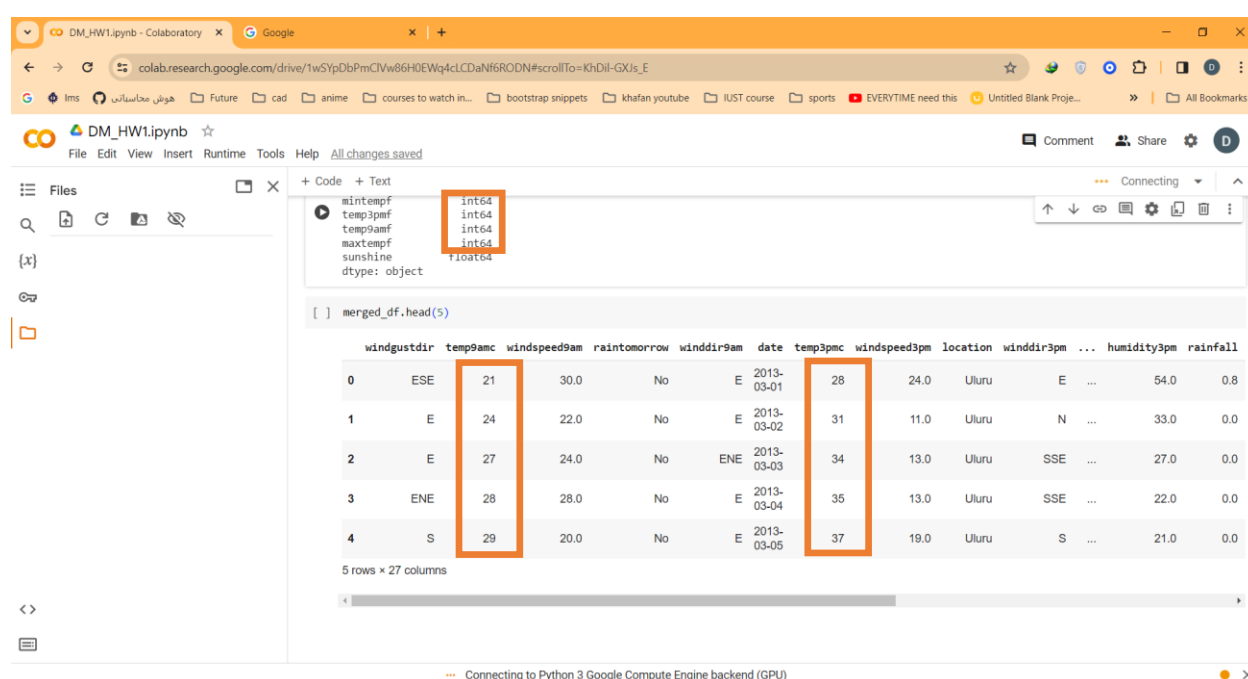
0s completed at 12:57 PM

۴. برای اینکار ابتدا لازم است که تمامی ستون هایی که temp دارند را شناسایی کنیم سپس تایپ آنها را به int تغییر دهیم، همچنین در اینجا یک چالش داشتیم آن هم اینکه Nan را نمیتوان به Int تغییر داد برای همین این مقادیر را در این ستون ها به مقدار ۹۹- تغییر دادیم که در داده های خودمان غیرممکن است (اگر با صفر پر

می‌کردیم ممکن بود داده واقعا در آن لحظه صفر باشد و در ادامه به مشکل می‌خوردیم). خروجی و کد را در ادامه می‌بینیم :

```
temp_columns = [col for col in merged_df.columns if 'temp' in col]

for col in temp_columns:
    merged_df[col] = merged_df[col].fillna(-99).astype(int)
print(merged_df.dtypes)
```



The screenshot shows a Google Colab notebook interface. The code cell contains the following code:

```
temp_columns = [col for col in merged_df.columns if 'temp' in col]

for col in temp_columns:
    merged_df[col] = merged_df[col].fillna(-99).astype(int)
print(merged_df.dtypes)
```

The output shows the dtypes of the merged DataFrame:

```
mintempf    int64
temp3pmf    int64
temp9amf    int64
maxtempf    int64
sunshine    float64
dtype: object
```

The output also shows the first 5 rows of the DataFrame:

```
[ ] merged_df.head(5)
```

	windgustdir	temp9amc	windspeed9am	rain tomorrow	winddir9am	date	temp3pmc	windspeed3pm	location	winddir3pm	...	humidity3pm	rainfall
0	ESE	21	30.0	No	E	2013-03-01	28	24.0	Uluru	E	...	54.0	0.8
1	E	24	22.0	No	E	2013-03-02	31	11.0	Uluru	N	...	33.0	0.0
2	E	27	24.0	No	ENE	2013-03-03	34	13.0	Uluru	SSE	...	27.0	0.0
3	ENE	28	28.0	No	E	2013-03-04	35	13.0	Uluru	SSE	...	22.0	0.0
4	S	29	20.0	No	E	2013-03-05	37	19.0	Uluru	S	...	21.0	0.0

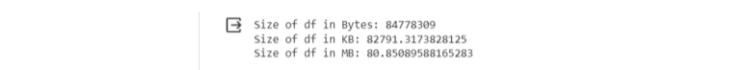
5 rows x 27 columns

۵. برای بدست آوردن سایز دیتافریم در RAM از کتابخانه sys استفاده می‌کنیم :

```
import sys

size_in_bytes = sys.getsizeof(merged_df)
size_in_kb = size_in_bytes / 1024
size_in_mb = size_in_kb / 1024

print(f"Size of df in Bytes: {size_in_bytes}")
print(f"Size of df in KB: {size_in_kb}")
print(f"Size of df in MB: {size_in_mb}")
```



The screenshot shows a Google Colab notebook interface. The code cell contains the following code:

```
import sys

size_in_bytes = sys.getsizeof(merged_df)
size_in_kb = size_in_bytes / 1024
size_in_mb = size_in_kb / 1024

print(f"Size of df in Bytes: {size_in_bytes}")
print(f"Size of df in KB: {size_in_kb}")
print(f"Size of df in MB: {size_in_mb}")
```

The output shows the size of the DataFrame:

```
Size of df in Bytes: 84778309
Size of df in KB: 82791.3173828125
Size of df in MB: 80.85089588165283
```

۶. بدین صورت تمامی متغیرهای باینری و اسمی را به category تغییر می‌دهیم، ابتدا مقادیر را از NULL به Unknown تغییر می‌دهیم سپس به کمک astype به category تبدیل می‌کنیم :


```

merged_df['windgustdir'].fillna('unknown', inplace=True)
merged_df['raintomorrow'].fillna('unknown', inplace=True)
merged_df['date'].fillna('unknown', inplace=True)
merged_df['location'].fillna('unknown', inplace=True)
merged_df['winddir3pm'].fillna('unknown', inplace=True)
merged_df['raintoday'].fillna('unknown', inplace=True)
merged_df['winddir9am'].fillna('unknown', inplace=True)

merged_df['windgustdir'] = merged_df['windgustdir'].astype('category')
merged_df['raintomorrow'] = merged_df['raintomorrow'].astype('category')
merged_df['date'] = merged_df['date'].astype('category')
merged_df['location'] = merged_df['location'].astype('category')
merged_df['winddir3pm'] = merged_df['winddir3pm'].astype('category')
merged_df['raintoday'] = merged_df['raintoday'].astype('category')
merged_df['winddir9am'] = merged_df['winddir9am'].astype('category')

print(merged_df.dtypes)

```

windgustdir	category
temp9am	int64
windspeed9am	float64
raintomorrow	category
winddir9am	category
date	category
temp3pm	int64
windspeed3pm	float64
location	category
winddir3pm	category

۷. با توجه به تغییرات داده شده و تغییرات که در ادامه میبینیم حجم به ۲۳ مگابایت کاهش یافت:

```

print(merged_df.dtypes)

```

winddir9am	category
humidity3pm	float64
raintoday	category
location	category
raintomorrow	category
humidity9am	float64
cloud3pm	float64
windgustdir	category
temp9am	int64
date	category
temp3pm	int64
mintemp	int64
pressure9am	float64
windspeed3pm	float64
evaporation	float64
windgustspeed	float64
rainfall	float64
pressure3pm	float64
sunshine	float64
windspeed9am	float64
winddir3pm	category
maxtemp	int64
cloud9am	float64
mintemp	int64
maxtempf	int64
temp9amf	int64
temp3pmf	int64
dtype:	object

```

import sys

size_in_bytes = sys.getsizeof(merged_df)
size_in_kb = size_in_bytes / 1024
size_in_mb = size_in_kb / 1024

print(f"Size of df in Bytes: {size_in_bytes}")
print(f"Size of df in KB: {size_in_kb}")
print(f"Size of df in MB: {size_in_mb}")

```

Size of df in Bytes: 24809347
Size of df in KB: 24227.8779296875
Size of df in MB: 23.66003704071045

که اگر بخواهیم تغییرات و درصد آن‌ها را نمایش دهیم به صورت مقابل میشود: ۲۳ - ۸۰ که ۵۷ مگ کاهش یافته است که درصد آن هم به صورت حدودا ۷۱٪ کاهش داشتیم.

۸. برای هر کدام از ستون‌ها چون یک سری مقادیر را برای محاسبات تغییر دادیم لازم است تا اندکی تابع محاسبه گر تعداد را تغییر دهیم:

```

import numpy as np

def count_missing_values(column):
    if column.dtype == np.number or column.dtype == np.int64 or column.dtype == np.float64:
        return sum(column == -99) + column.isnull().sum()
    else:
        return sum(column == 'Unknown') + column.isnull().sum()

missing_values_count = merged_df.apply(count_missing_values)
print(missing_values_count)

```

که نتایج آن به صورت زیر است :

```
<ipython-input-18-0c4daf7e9541>:4:
if column.dtype == np.number or c
winddir9am      10566
humidity3pm      4507
raintoday       3261
location        0
raintomorrow    3267
humidity9am     2654
cloud3pm        59358
windgustdir     10326
temp9amc        87251
date            0
temp3pmc        87886
mintempc        87232
pressure9am     15065
windspeed3pm    3062
evaporation     62790
windgustspeed   10263
rainfall        3261
pressure3pm     15028
sunshine        69835
windspeed9am    1767
winddir3pm      4228
maxtempc        87194
cloud9am        55888
mintempf        59713
maxtempf        59527
temp9amf        59976
temp3pmf        61183
dtype: int64
```

۹. برای ستون هایی که خودمان تغییر دادیم مثلا temp9amc که ممکن است در یک ایستگاه کلا بر مبنای فارنهایت ارائه شده باشد، حذف کردن سطر کاملاً کار اشتباهی است، چون دیتا کامل موجود است ولی در سطرهایی که مثلاً جهت باد وجود ندارد و ما هم نمیتوانیم پیش بینی داشته باشیم حذف سطر منطقی است همچنین در سطرهایی که مثلاً داده قبلی و بعدی را داریم، با یک تقریبی میتوان گفت که داده میانی هم مثل قبلی و بعدی بوده است، یا مثلاً بین دو روز ابری، احتمال زیاد روز بین هم ابری بوده است، همچنین در مورد داده های عددی مانند دما میتوان با یک احتمال خوب میانگین چند داده حول آن را در نظر گرفت، همچنین این مورد پر کردن داده های گم شده کاملاً به کاربرد مورد استفاده ما بستگی دارد شاید مثلاً تنها برای ما مفید باشد و از آن استفاده کنیم پس اگر رطوبت یا ابری بودن را نداشتیم، مشکلی بوجود نیاید. همچنین یک روش دیگری که میتوان استفاده کرد و بنده هم سعی میکنم این روش را در نظر بگیرم این است که چندین ستون با هم چک شوند اگر حجم زیادی از دیتا (تعداد زیادی ستون) از دست رفته بود آن سطر را حذف میکنیم اینطوری هم مطمئن میشیم اشتباه حذف نمیکنیم (مثال توضیح داده شده در سطر دوم همین صفحه) هم اگر دیتای باقی مانده مفید نبود حذف میشود.

۱۰. با توجه به توضیحات بالا کد را پیاده سازی میکنیم ، بدین صورت که در هر row مقدار missing value ها را پیدا میکنیم و بر مبنای آن عمل میکنیم :

```

[34] print(len(merged_df))

145460

def count_missing_values(row):
    if row.dtype == np.number or row.dtype == np.int64 or row.dtype == np.float64:
        return sum(row == -99) + row.isnull().sum()
    else:
        return sum(row == 'Unknown') + row.isnull().sum()

missing_values_count_row = merged_df.apply(count_missing_values, axis=1)

filled_df = merged_df[missing_values_count_row <= 10]

<ipython-input-36-8ba0236c03b1>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype
if row.dtype == np.number or row.dtype == np.int64 or row.dtype == np.float64:
<ipython-input-37-8ba0236c03b1>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype
if row.dtype == np.number or row.dtype == np.int64 or row.dtype == np.float64:

print(len(filled_df))

143776

```

۱۱. برای اینکار ابتدا ستون های را پیدا میکنیم که در اسم ستون **temp** وجود داشته باشد و با **f** تمام شود سپس با کمک فرمول تبدیل فارنهایت به سلیسیوس این تبدیل را انجام میدهیم :

```

def fahrenheit_to_celsius(value):
    if value != -99:
        return (value - 32) * 5.0/9.0
    else:
        return value

temp_columns = filled_df.filter(regex='temp.*f$').columns

for column in temp_columns:
    filled_df[column] = filled_df[column].apply(fahrenheit_to_celsius)

<ipython-input-39-4f0a866c6687>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

```

۱۲. از روش چارک برای این کار استفاده کردیم، برای ستون های عددی چارک اول و سوم را حساب کردیم سپس به کمک این دو مقدار **IQR** را حساب میکنیم، سپس داده هایی که بیش از ۱.۵ برابر **IQR**، کوچکتر از چارک اول هستند یا همین مقدار بزرگ از چارک سوم هستند را به عنوان **outlier** تشخیص میدهد که برای درک این مقادیر تعدادی را نیز چاپ کردیم:

```

def find_outliers(column):
    if column.dtype == np.number or column.dtype == np.int64 or column.dtype == np.float64:
        column = column.replace(-99, np.nan)

        Q1 = column.quantile(0.25)
        Q3 = column.quantile(0.75)
        IQR = Q3 - Q1

        outliers = (column < (Q1 - 1.5 * IQR)) | (column > (Q3 + 1.5 * IQR))

        return outliers
    else:
        column = column.replace('Unknown', np.nan)

        return pd.Series([np.nan]*len(column), index=column.index)

outliers = filled_df.apply(find_outliers)

```

```

outlier_rows = outliers.any(axis=1)

outlier_df = filled_df[outlier_rows]

outlier_df = outlier_df.dropna()

print(outlier_df.head(10))

```

	winddir9am	humidity3pm	raintoday	location	raintomorrow	humidity9am
1004	S	77.0	Yes	Sydney	Yes	78.0
1008	SE	53.0	Yes	Sydney	Yes	67.0
1011	E	52.0	No	Sydney	Yes	64.0
1012	SSE	54.0	Yes	Sydney	No	80.0
1014	ESE	43.0	Yes	Sydney	No	77.0
1019	S	65.0	Yes	Sydney	No	82.0
1033	S	91.0	Yes	Sydney	Yes	94.0
1034	ENE	89.0	Yes	Sydney	Yes	94.0
1035	ENE	78.0	Yes	Sydney	No	91.0
1050	SSW	71.0	Yes	Sydney	No	75.0

	cloud3pm	windgustdir	temp9amc	date	...	pressure3pm	sunshine
1004	8.0	SSE	18	2010-11-01	...	1012.1	0.1
1008	7.0	SE	15	2010-11-05	...	1022.1	2.7
1011	7.0	SW	23	2010-11-08	...	1015.1	7.9
1012	8.0	SSE	19	2010-11-09	...	1024.2	7.0
1014	3.0	E	20	2010-11-11	...	1012.0	9.3
1019	3.0	ESE	18	2010-11-16	...	1011.3	7.9
1033	8.0	SSW	17	2010-11-30	...	1014.5	0.9
1034	8.0	E	18	2010-12-01	...	1015.3	0.0
1035	7.0	ENE	20	2010-12-02	...	1015.9	5.4

0s completed at 2:40 PM

سوال دوم)

۱) در قسمت اول خواسته شده میانگین بارش در استرالیا را بدست بیاوریم، با توجه به پیش پردازش مناسب تنها لازم است که از فانکشن **built-in** میانگین استفاده کنیم کد یک خطی این قسمت به صورت زیر است :

```

df[['rainfall']].mean()

```

2.3562617405419966

۲) در این قسمت ابتدا برای تغییر نکردن دیتافریم اصلی، یک کپی از آن میگیریم در مرحله بعد نوع ستون زمان و را به **datetime** تغییر میدهم که کار کردن و فیلتر کردن سال راحتتر باشد، در ادامه ابتدا لوکیشن را به **watsonia** فیلتر میکنیم و همزمان میگوییم سطرهایی را انتخاب کن که در **datetime** آن ها بخش مربوط به سال برابر ۲۰۱۵ باشد در ادامه در خط بعد تنها تعداد روزهایی که بارانی بودند را به کمک **raintoday** بدست میآوریم و تعداد **Yes** ها را پرینت میکنیم. (که در اینجا ۷۸ جواب شد در صورتی که خودم که در اکسل فیلتر کردم دیدم این مقدار برابر ۷۹ شد با جست و جو و بررسی مورد فهمیدم که یک سطر آن تقریباً خالی بوده است و دیتای خاصی نداشته (طبق شرطی که گذاشته بودم در پیش پردازش اگر از یک حدی کمتر دیتا داشت کلا حذف میکند))

در ادامه کد این قسمت را مشاهده میکنیم :



```
DM_HW2.ipynb
File Edit View Insert Runtime Tools Help All changes saved

Files
[x] ..
  sample_data
  weatherAUS
  combined_data.xlsx
  weatherAUS.zip

[28] df["rainfall"].mean()

2.3562617405419966

df_copy = df.copy()
df_copy['date'] = pd.to_datetime(df_copy['date'])

watsonia_2015 = df_copy[(df_copy['location'] == 'Watsonia') & (df_copy['date'].dt.year == 2015)]

rainy_days_watsonia_2015 = watsonia_2015[watsonia_2015['raintoday'] == 'Yes'].shape[0]

print(f"Number of rainy days in Watsonia in 2015: {rainy_days_watsonia_2015}")

Number of rainy days in Watsonia in 2015: 78
```

۳) برای این قسمت ابتدا لوکیشن را فیلتر میکنیم به طوری که تنها سطریایی داشته باشیم که مربوط به شهر Townsville باشند، سپس با توجه به اینکه در روز دو بار رطوبت سنجیده میشود یک ستون جدید تعریف میکنیم که به ازای هر سطر این مقدار برابر با مقدار بزرگتر بین این دو رطوبت است، سپس برای بیان ماکسیمم کل از این ستون max میگیریم، همچنین برای بهتر شدن مفهوم آن روز و تاریخ متناسب با آن را هم پرینت میکنیم که در ادامه مشاهده میکنیم برابر ۱۰۰ است و در تاریخ ۱۳-۰۴-۲۰۱۴ رخ داده است.



```
DM_HW2.ipynb
File Edit View Insert Runtime Tools Help All changes saved

Files
[x] ..
  sample_data
  weatherAUS
  combined_data.xlsx
  weatherAUS.zip

[29]

Number of rainy days in Watsonia in 2015: 78

townsville_df = df[df['location'] == 'Townsville']

townsville_df['max_daily_humidity'] = townsville_df[['humidity9am', 'humidity3pm']].max(axis=1)

max_humidity = townsville_df['max_daily_humidity'].max()
max_humidity_dates = townsville_df[townsville_df['max_daily_humidity'] == max_humidity]['date']

print(f"Maximum daily humidity in Townsville: {max_humidity}%")
print("Date(s) with this humidity level:")
print(max_humidity_dates.to_string(index=False))

Maximum daily humidity in Townsville: 100.0%
Date(s) with this humidity level:
2014-04-13
```

۴) در این قسمت هدف محاسبه حداکثر اختلاف دما در ماه اول میلادی است، چیزی که مقداری این مسئله را سخت میکند بحث فارنهایت و سانتیگراد است که با توجه به پیش پردازش کامل این مشکل حل شده است و تمامی دماها به سانتیگراد تبدیل شده است پس دیگر کاری ندارد، ابتدا دوباره برای راحتی کار با تاریخ آن را به datetime تبدیل میکنیم و فقط آنهایی انتخاب میکنیم که ماه آنها برابر ۱ باشد، در ادامه یک ستون جدید از اختلافها تشکیل میدهیم البته چون در دو ستون هستند لازم است در هر کدام مقدار بزرگتر یا کوچکتر را انتخاب کنیم (دقت کنیم که جفت آن ها به سانتیگراد است)، در نهایت در ستون جدید مقدار ماکسیمم این تفاوت را پرینت میکنیم. در ادامه کد این قسمت را مشاهده میکنیم:

```

import pandas as pd

df['date'] = pd.to_datetime(df['date'])

january_df = df[df['date'].dt.month == 1]

january_df['daily_max_diff'] = january_df[['maxtemp', 'maxtemp']].max(axis=1) - january_df[['mintemp', 'mintemp']].min(axis=1)

max_temp_diff_january = january_df['daily_max_diff'].max()

print(f"The maximum temperature difference in January (across all years) is: {max_temp_diff_january}°C")

```

The maximum temperature difference in January (across all years) is: 30.0°C

۵) در این قسمت از این قابلیت که در مرحله قبل `date` را تبدیل به آبجکت تایپ `datetime` کردیم استفاده میکنیم، بدین صورت که ابتدا شهر را فیلتر میکنیم و همزمان چک میکنیم آیا ماه در ۳۰ و ۳۱ هست یا خیر که این یعنی سه ماهه اول هر سال.

حال که دیتایی که میخواستیم را بدست آوردیم به سراغ ۵ روز سرد میرویم، که در این رابطه ابتدا داده ها را سورت میکنیم (بر مبنای ستون جدیدی که تعریف کردیم و مینیمم هر روز را دارد) سپس ۵ تای اول آن که کمترین ها هستند را برمیداریم (به همراه تاریخ) کد و خروجی در ادامه دیده میشود :

```

import pandas as pd

mount_ginini_first_quarter = df[(df['location'] == 'MountGinini') & (df['date'].dt.month.isin([1, 2, 3]))]

mount_ginini_first_quarter['daily_min_temp'] = mount_ginini_first_quarter[['mintemp', 'mintemp']].min(axis=1)

coldest_5_days = mount_ginini_first_quarter.sort_values(by='daily_min_temp').head(5)

coldest_5_days = coldest_5_days[['date', 'daily_min_temp']]

print("The coldest 5 days in Mount Ginini during the first three months of the year:")
print(coldest_5_days.to_string(index=False))

```

date	daily_min_temp
2012-01-12	-1.666667
2017-02-20	-1.666667
2017-03-31	-1.111111
2012-03-24	-1.111111
2015-03-27	-1.111111

۳) در این بخش ابتدا لازم بود یک مقدار تحقیقات انجام دهیم، با توجه به نتایج یافت شده چندین حالت داریم،
۱. به صورت مرتب دیتا اضافه شود و لازم باشد بعد هر آپدیت این متریک ها هم آپدیت شوند

۲. هر شب آپدیت کنیم

۳. هر وقت فایل ها دیتاست تغییر کردند آپدیت کنیم

که با توجه به مثال ما گزینه سوم مناسب تر بود، در این زمینه از لایبرری `watchdog` میتوان استفاده کرد و ران کردن آن هم بدین صورت است که یک `Observer` تعریف میکنیم و روی دایرکتوری دیتاست قرار میدهم سپس به ازای `on_modified` لازم است تا چک کنیم اگر در دایرکتوری بود و مربوط به دیتاست ما بود در ادامه فانکشنهایی که جدا کردیم را فراخوانی مشکلی که در این قسمت داریم با توجه به ران کردن فایلمان در کولب

امکان تست کردن این ویژگی وجود ندارد چون کولب این قابلیت را ساپورت نمیکند ولی در ادامه کد این قسمت را مشاهده میکنیم که یک سری جاها خالی هستند و میتوانند با آدرس دیتاست یا اسم فایل آن کامل شوند و فایل آن هم در کنار نوتبوک قرار داده شده است :

```
*update_run.py - C:/Users/bamir/Desktop/update_run.py (3.6.0)*
File Edit Format Run Options Window Help
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import os

class MyHandler(FileSystemEventHandler):
    def on_modified(self, event):
        if not event.is_directory and os.path.basename(event.src_path) == "data_file.csv":
            print("Data file has been updated. Recalculating metrics...")

event_handler = MyHandler()
observer = Observer()
observer.schedule(event_handler, path='dataset/directory', recursive=False)
observer.start()
|
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    observer.stop()

observer.join()
```

در ادامه پرینت هم فانکشن هایی که در زیر تعریف میکنیم قرار داده میشوند :

```
DM_HW2.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
Comment Share ⚙️ D

Files
(x)
sample_data
weatherAUS
combined_data.xlsx
weatherAUS.zip

def calculate_average_rainfall(dataframe):
    return dataframe['rainfall'].mean()

def calculate_rainy_days_watsonia_2015(dataframe):
    filtered = dataframe[(dataframe['location'] == 'Watsonia') & (dataframe['date'].dt.year == 2015)]
    return filtered[filtered['raintoday'] == 'Yes'].shape[0]

def calculate_highest_humidity_townsville(dataframe):
    filtered = dataframe[dataframe['location'] == 'Townsville']
    filtered['max_daily_humidity'] = filtered[['humidity3pm', 'humidity9am']].max(axis=1)
    return filtered['max_daily_humidity'].max()

def calculate_max_temp_diff_january(dataframe):
    january_df = dataframe[dataframe['date'].dt.month == 1]
    january_df['temp_diff'] = january_df[['maxtemp', 'maxtempf']].max(axis=1) - january_df[['mintemp', 'mintempf']].min(axis=1)
    return january_df['temp_diff'].max()

def calculate_coldest_days_mount_ginini(dataframe):
    filtered = dataframe[(dataframe['location'] == 'MountGinini') & (dataframe['date'].dt.month <= 3)]
    filtered['min_temp'] = filtered[['mintemp', 'mintempf']].min(axis=1)
    return filtered.sort_values(by='min_temp').head(5)[['date', 'min_temp']]
```