Importing libraries

In [37]:
```
import pandas as pd
import numpy as np
import os
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing data, transforming json format and concatinating all files

In [2]:
```
#reading the json file and set orient to 'index'
df= pd.read_json(r'Running data\2016-06-09_04-16-11-UTC_5758f467da311371b647c6

#transpose columns and rows to data friendly format
df=df.T
df

#getting a list of all files from the directory
files =[file for file in os.listdir( './Running data')]

all_data = pd.DataFrame()

for file in files:
    df = pd.read_json('./Running data/' + file,orient='index')
    df=df.T
    #df['Filename'] = os.path.split(file)[-1]
    all_data = pd.concat([all_data,df])

#protect privacy and delete id column
del all_data['id']


all_data.shape
```

Out[2]:
```
(143, 37)
```

In [3]:
```
#let's review the first 5 rows
all_data.head()
```

Out[3]:

	start_time	end_time	created_at	updated_at	start_time_timezone_offset	en
0	1465445771000	1465447524000	1465447527000	1642911372000	43200000	
0	1465775264000	1465776933000	1465776936000	1642911372000	43200000	
0	1466378351000	1466380088000	1466380092000	1642911372000	43200000	
0	1468449087000	1468450897000	1468450900000	1642911372000	43200000	
0	1468538657000	1468541410000	1468541414000	1642911372000	43200000	

5 rows × 37 columns

In [4]:
```
#dropping rows that are not needed
del all_data['subjective_intensity']
del all_data['workout_data']
del all_data['location_context']
del all_data['notes']
del all_data['surface_id']
del all_data['indoor']
del all_data['live_tracking_active']
del all_data['live_tracking_enabled']
del all_data['cheering_enabled']
del all_data['manual']
del all_data['edited']
del all_data['pulse_max']
del all_data['avg_cadence']
del all_data['max_cadence']
del all_data['start_time_timezone_offset']
del all_data['end_time_timezone_offset']
del all_data['weather_condition_id']
del all_data['pulse_avg']
del all_data['completed']
del all_data['max_speed']
del all_data['pause_duration']
```

In [5]:
```
#inspecting null values
#note: Not surprising that subjective feeling has multiple null values as I di
all_data.isna().sum().sort_values(ascending=False)
```

Out[5]:
```
subjective_feeling_id    28
temperature               8
longitude                 5
latitude                  5
elevation_gain            2
elevation_loss            2
average_speed             2
duration_per_km           2
start_time                0
end_time                  0
created_at                0
updated_at                0
distance                  0
duration                  0
calories                  0
sport_type_id             0
dtype: int64
```

In [6]:
```
#getting info about columns
print(all_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 143 entries, 0 to 0
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   start_time            143 non-null    object
 1   end_time              143 non-null    object
 2   created_at            143 non-null    object
 3   updated_at            143 non-null    object
 4   distance              143 non-null    object
 5   duration              143 non-null    object
 6   elevation_gain        141 non-null    object
 7   elevation_loss        141 non-null    object
 8   average_speed         141 non-null    object
 9   calories              143 non-null    object
 10  longitude             138 non-null    object
 11  latitude              138 non-null    object
 12  duration_per_km       141 non-null    object
 13  sport_type_id         143 non-null    object
 14  subjective_feeling_id  115 non-null    object
 15  temperature           135 non-null    object
dtypes: object(16)
memory usage: 19.0+ KB
None
```

Converting data types

In [7]:
```
#convert data into correct datatypes
all_data = all_data.apply(pd.to_numeric, errors='ignore')

#converting milliseconds into datetime objects

all_data['start_time']=pd.to_datetime(all_data['start_time'], unit='ms')
all_data['end_time']=pd.to_datetime(all_data['end_time'], unit='ms')
all_data['created_at']=pd.to_datetime(all_data['created_at'], unit='ms')
all_data['updated_at']=pd.to_datetime(all_data['updated_at'], unit='ms')

all_data
```

Out[7]:

	start_time	end_time	created_at	updated_at	distance	duration	elevation_gain	elevation_l
0	2016-06-09 04:16:11	2016-06-09 04:45:24	2016-06-09 04:45:27	2022-01-23 04:16:12	6544	1752065	44.0	
0	2016-06-12 23:47:44	2016-06-13 00:15:33	2016-06-13 00:15:36	2022-01-23 04:16:12	6381	1667722	45.0	
0	2016-06-19 23:19:11	2016-06-19 23:48:08	2016-06-19 23:48:12	2022-01-23 04:16:12	6393	1736110	41.0	
0	2016-07-13 22:31:27	2016-07-13 23:01:37	2016-07-13 23:01:40	2022-01-23 04:16:12	5941	1734109	84.0	
0	2016-07-14 23:24:17	2016-07-15 00:10:10	2016-07-15 00:10:14	2022-01-23 04:16:12	8600	2597823	145.0	1
...	...	...	...	...	...	...	...	
0	2022-08-01 08:28:04	2022-08-01 09:04:52	2022-08-01 09:04:54	2022-08-01 09:04:57	7011	2206931	63.0	
0	2022-08-04 09:56:59	2022-08-04 10:26:35	2022-08-04 10:26:38	2022-08-04 10:26:44	5884	1775331	46.0	
0	2022-08-10 04:11:23	2022-08-10 04:41:47	2022-08-10 04:41:49	2022-08-10 04:41:50	6024	1823209	52.0	
0	2022-08-14 09:55:21	2022-08-14 10:24:31	2022-08-14 10:24:32	2022-08-14 10:24:35	6001	1748961	52.0	
0	2022-08-16 09:54:03	2022-08-16 10:23:34	2022-08-16 10:23:34	2022-08-16 10:31:57	6002	1769446	53.0	

143 rows × 16 columns

Adding new columns

In [8]:
```
#  Duration is currently shown in milliseconds, we will convert it into minute
all_data['duration'] = (all_data['duration']/60)/1000

#adding a count column
all_data['count']=1

#adding a column for pace 'minutes/km'
all_data['pace'] = all_data['duration_per_km']/60/1000

#adding time intelligence columns

all_data['year'] =all_data['start_time'].dt.year
all_data['month'] =all_data['start_time'].dt.month
all_data['month_name'] =all_data['start_time'].dt.month_name()



#all_data.reset_index()
all_data.head()
```

Out[8]:

	start_time	end_time	created_at	updated_at	distance	duration	elevation_gain	elevation_
0	2016-06-09 04:16:11	2016-06-09 04:45:24	2016-06-09 04:45:27	2022-01-23 04:16:12	6544	29.201083	44.0	
0	2016-06-12 23:47:44	2016-06-13 00:15:33	2016-06-13 00:15:36	2022-01-23 04:16:12	6381	27.795367	45.0	
0	2016-06-19 23:19:11	2016-06-19 23:48:08	2016-06-19 23:48:12	2022-01-23 04:16:12	6393	28.935167	41.0	
0	2016-07-13 22:31:27	2016-07-13 23:01:37	2016-07-13 23:01:40	2022-01-23 04:16:12	5941	28.901817	84.0	
0	2016-07-14 23:24:17	2016-07-15 00:10:10	2016-07-15 00:10:14	2022-01-23 04:16:12	8600	43.297050	145.0	

5 rows × 21 columns

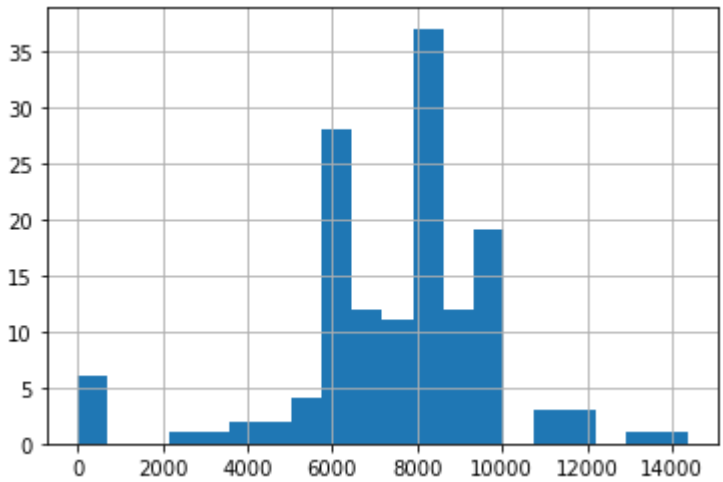
Statistical and visual Exploration of the dataset

In [9]:
```
# reviewing summary statistics for numeric fields to look for uncommon or weir
#note:max pace is above 22km/minute, that is too high --> investigate
#note:min distance is 0. This might indicate that the GPS did not work --> dro
all_data.describe()
```

Out[9]:

	distance	duration	elevation_gain	elevation_loss	average_speed	calories	l
count	143.000000	143.000000	141.000000	141.000000	141.000000	143.000000	13
mean	7549.013986	37.666641	57.510638	60.262411	11.994157	628.447552	3
std	2432.432174	11.052150	36.975014	36.638123	2.199242	208.515804	5
min	0.000000	0.054117	0.000000	0.000000	0.000000	0.000000	
25%	6211.000000	30.126458	41.000000	43.000000	11.834678	517.500000	
50%	8010.000000	37.721467	48.000000	51.000000	12.337791	652.000000	
75%	8783.500000	44.667675	70.000000	75.000000	12.960000	732.000000	
max	14349.000000	72.805033	216.000000	224.000000	14.432045	1286.000000	17

In [10]:
```
#Exploring the dataset by creating a histogram for distance
all_data['distance'].hist(bins=20)

plt.show()
```



In [11]:
```
#reviewing dataset as some values show a very small distance
all_data= all_data.sort_values('distance',ascending=True)
all_data.head(10)
```

Out[11]:

	start_time	end_time	created_at	updated_at	distance	duration	elevation_gain	elevation_
0	2021-06-11 17:23:49	2021-06-11 17:23:53	2021-06-11 17:23:53	2022-01-23 04:16:26	0	0.054117	0.0	
0	2020-04-09 12:20:17	2020-04-09 12:43:50	2020-04-09 12:44:29	2022-01-23 04:16:01	0	17.833333	NaN	
0	2021-02-02 16:13:14	2021-02-02 16:43:54	2021-02-02 16:44:21	2022-01-23 04:16:41	0	25.133333	NaN	
0	2021-03-25 09:20:25	2021-03-25 09:20:33	2021-03-25 09:20:33	2022-01-23 04:16:41	0	0.108533	0.0	
0	2022-07-12 23:41:37	2022-07-13 00:02:02	2022-07-13 00:02:01	2022-07-13 00:02:03	0	20.402533	0.0	
0	2020-07-10 14:23:18	2020-07-10 14:24:33	2020-07-10 14:24:33	2022-01-23 04:16:26	265	1.230583	0.0	
0	2020-04-13 13:41:53	2020-04-13 14:35:41	2020-04-13 14:35:41	2022-01-23 04:16:01	2353	53.788433	26.0	
0	2016-08-03 00:48:01	2016-08-03 01:23:18	2016-08-03 01:23:32	2022-01-23 04:16:38	3149	34.424317	57.0	
0	2020-03-14 14:41:15	2020-03-14 15:10:41	2020-03-14 15:11:22	2022-01-23 04:16:26	4035	20.353500	30.0	
0	2021-01-20 19:03:01	2021-01-20 20:15:42	2021-01-20 20:15:44	2022-01-23 04:16:26	4038	19.169483	28.0	

10 rows × 21 columns

Dropping rows with 0 km distance or activities which are not "running".

In [12]:
```
#some data show 0 distance or have a sport type id of '19' (which is walking I
#For further analysis I will exclude all exercises below 1000m distance or wit

all_data['sport_type_id'].value_counts()

all_data= all_data[all_data['distance']>1000]
all_data= all_data[all_data['sport_type_id'] != 19]

all_data.sort_values('start_time',ascending=False).head()
```

Out[12]:

	start_time	end_time	created_at	updated_at	distance	duration	elevation_gain	elevation_
0	2022-08-16 09:54:03	2022-08-16 10:23:34	2022-08-16 10:23:34	2022-08-16 10:31:57	6002	29.490767	53.0	
0	2022-08-14 09:55:21	2022-08-14 10:24:31	2022-08-14 10:24:32	2022-08-14 10:24:35	6001	29.149350	52.0	
0	2022-08-10 04:11:23	2022-08-10 04:41:47	2022-08-10 04:41:49	2022-08-10 04:41:50	6024	30.386817	52.0	
0	2022-08-04 09:56:59	2022-08-04 10:26:35	2022-08-04 10:26:38	2022-08-04 10:26:44	5884	29.588850	46.0	
0	2022-08-01 08:28:04	2022-08-01 09:04:52	2022-08-01 09:04:54	2022-08-01 09:04:57	7011	36.782183	63.0	

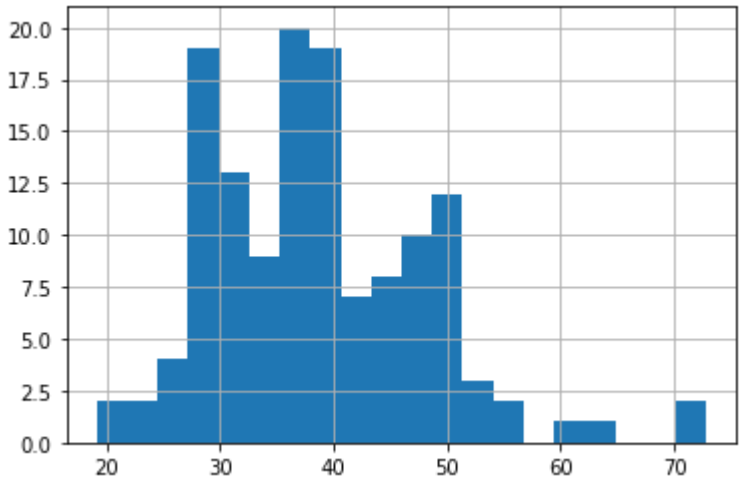
5 rows × 21 columns

In [13]:
```
#describe again to see changes
all_data.describe()
```

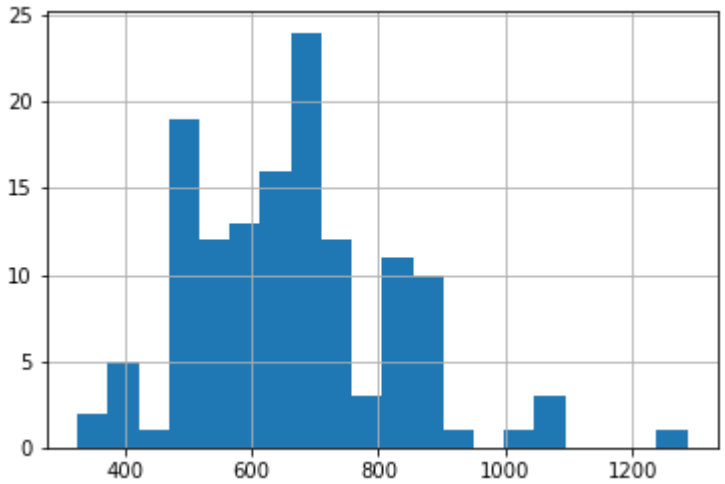
Out[13]:

	distance	duration	elevation_gain	elevation_loss	average_speed	calories	l
count	134.000000	134.000000	134.000000	134.000000	134.000000	134.000000	13
mean	7944.679104	38.739899	58.873134	61.708955	12.366602	664.037313	3
std	1802.805911	9.458460	35.816034	35.369760	0.844964	160.148353	5
min	4035.000000	19.169483	0.000000	0.000000	9.248755	326.000000	
25%	6397.750000	30.701058	41.000000	44.250000	11.863615	532.500000	
50%	8020.500000	37.804850	48.000000	51.500000	12.359500	658.000000	
75%	8966.000000	45.099717	70.750000	75.750000	12.962000	740.000000	
max	14349.000000	72.805033	216.000000	224.000000	14.432045	1286.000000	17

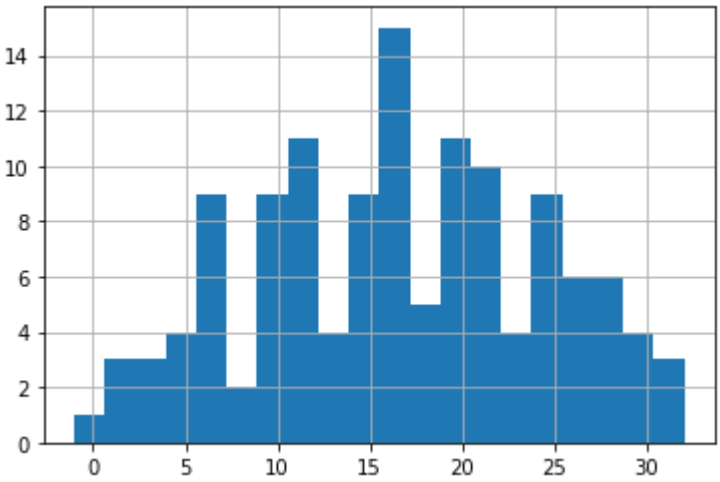
In [14]:
```
#Exploring the dataset by creating a histogram for duration
all_data['duration'].hist(bins=20)

plt.show()
```



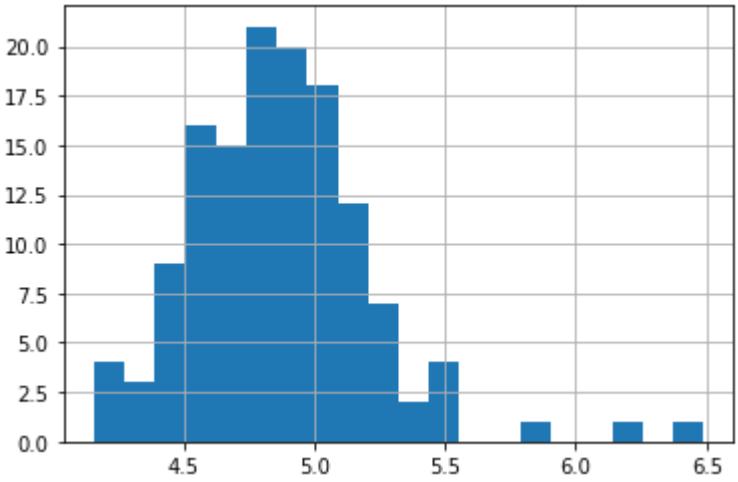
In [15]:
```
all_data['calories'].hist(bins=20)

plt.show()
```



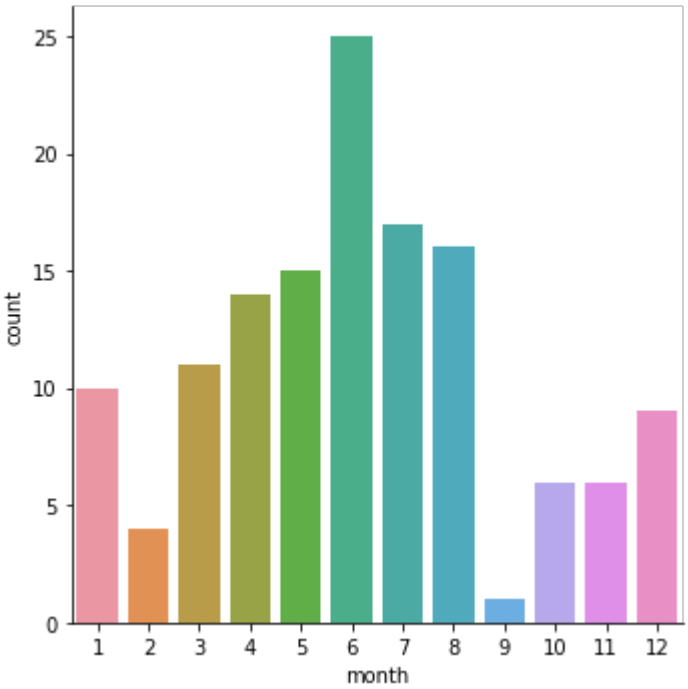
In [16]:
```
all_data['temperature'].hist(bins=20)

plt.show()
```



In [17]:
```
all_data['pace'].hist(bins=20)

plt.show()
```

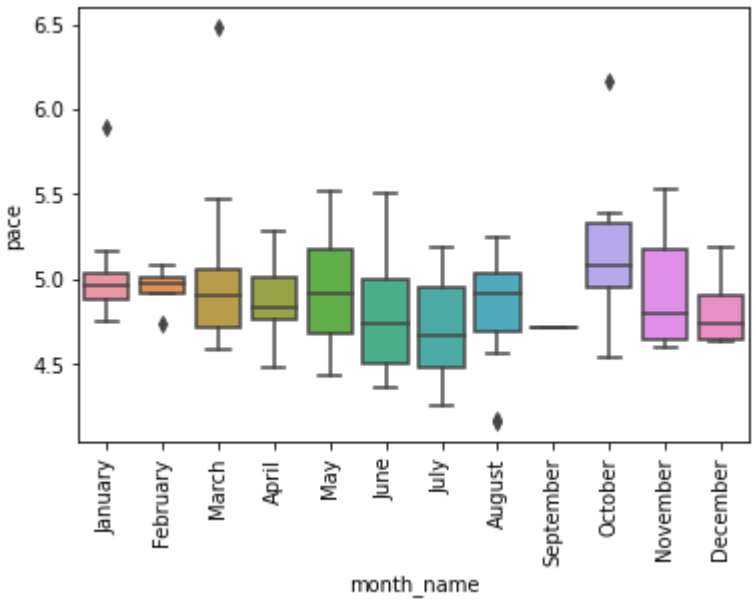


In [18]:
```
#Visualizing the number of runs by months
sns.catplot(x='month',data=all_data, kind='count')
plt.show()
```

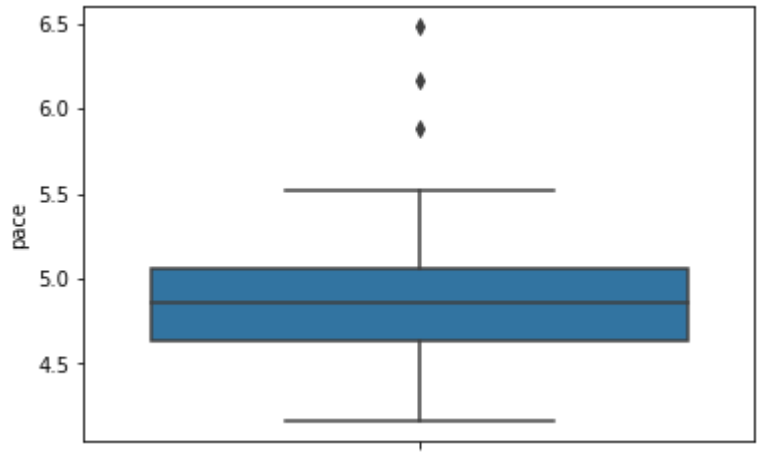


Identifying outliers

In [19]:
```
#creating boxplots
month_order=['January','February','March','April','May','June','July','August'
sns.boxplot(x='month_name',y='pace',data=all_data,order=month_order)
plt.xticks(rotation=90)
plt.show()
```



In [20]:
```
#month shows more outliers, maybe because for some months not enough data, so
sns.boxplot(y='pace', data=all_data)
plt.show()
```



In [21]:
```
all_data.shape
```

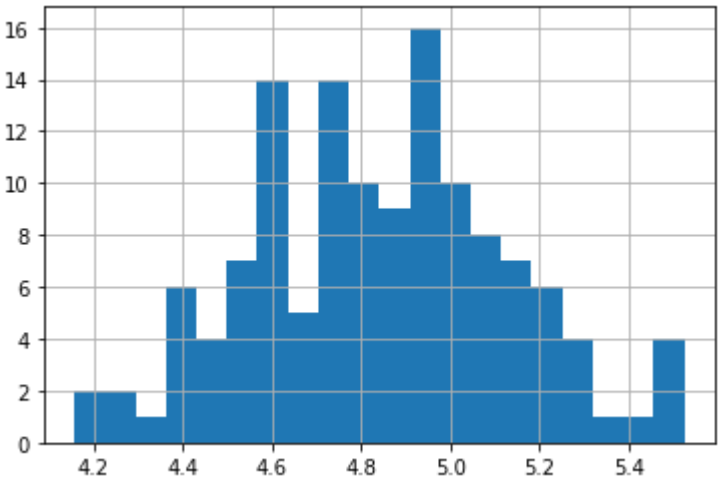
Out[21]: (134, 21)

In [22]:
```
#Finding outliers
from scipy.stats import iqr
iqr = iqr(all_data['pace'])
lower = np.quantile(all_data['pace'], 0.25) - 1.5 * iqr
upper = np.quantile(all_data['pace'], 0.75) + 1.5 * iqr

print(lower)
print(upper)

#showing outliers (3 in total)
outliers= all_data[((all_data['pace'] < lower)) | (all_data['pace'] > upper)]


all_data = all_data[all_data['pace'] > lower]
all_data =all_data[all_data['pace'] < upper]

#dataset got 3 rows less
all_data.shape
```

```
3.985785416666668
5.701668749999999
```

Out[22]: (131, 21)

In [23]:
```
all_data['pace'].hist(bins=20)

plt.show()
```



In [24]:
```
#deleting other non-needed columns
del all_data['sport_type_id']
del all_data['duration_per_km']
del all_data['calories']
```

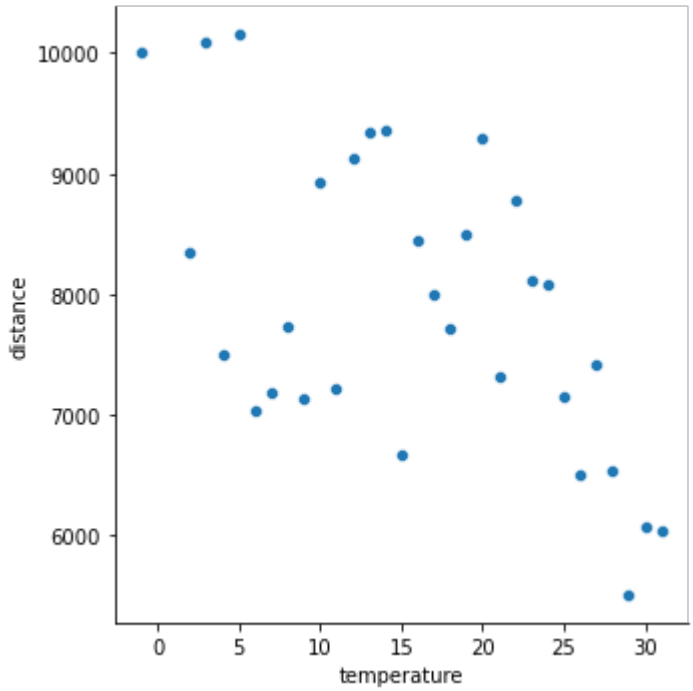
Outputting the cleaned dataframe

In [25]:
```
all_data.to_excel('Runtastic_cleaned.xlsx')
```

Plotting graphs and discovering relationships

Is there a relationship between distance and temperature?

In [26]:
```
# It looks like that distance and temperature are negative correlated. The hot
temp_pace = all_data.groupby('temperature')['distance'].mean()
sns.relplot(data=temp_pace, kind='scatter')
plt.show()
```



Which months have the best pace?

In [27]:
```
month_pace = all_data.groupby('month_name')['pace'].mean().sort_values(ascendi
month_pace
```

Out[27]:
```
month_name
July         4.711889
September    4.719350
June         4.758439
August       4.821081
December     4.822935
April        4.878093
March        4.889263
November     4.933122
January      4.935507
February     4.944287
May          4.949992
October      5.003730
Name: pace, dtype: float64
```

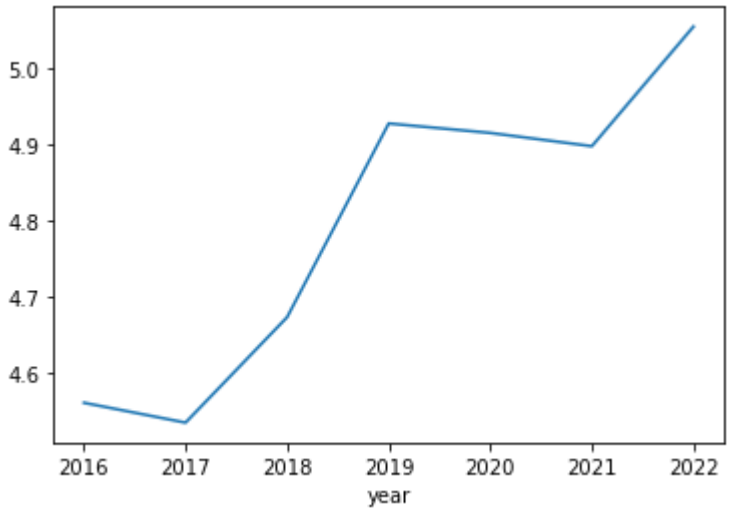
Which year has the best pace?

In [28]:
```
year_pace = all_data.groupby('year')['pace'].mean().sort_values(ascending=True
year_pace
```

Out[28]:
```
year
2017    4.534189
2016    4.560307
2018    4.672945
2021    4.897822
2020    4.915288
2019    4.927740
2022    5.055197
Name: pace, dtype: float64
```

In [29]:
```
# displaying pace by year
year_pace = all_data.groupby('year')['pace'].mean()
year_pace.plot()
```

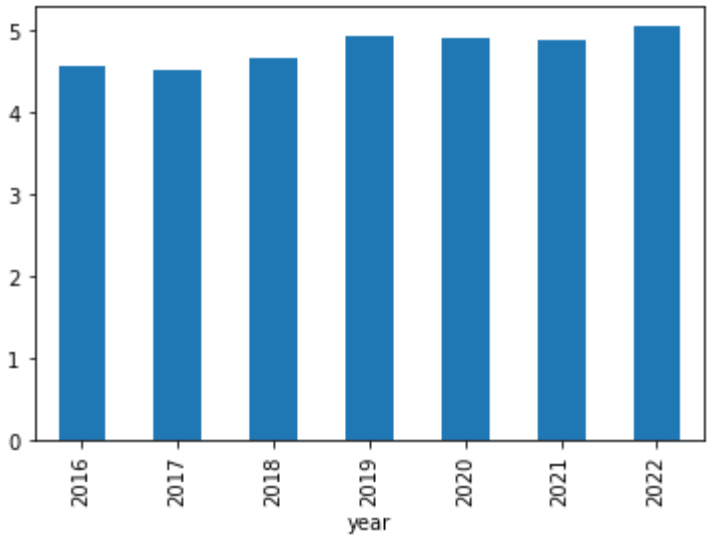
Out[29]:
```
<AxesSubplot:xlabel='year'>
```



Which year had the most runs?

In [30]:
```
year_runs = all_data.groupby('year')['count'].sum()
year_runs.sort_values(ascending=True)
year_pace.plot(kind='bar')
```

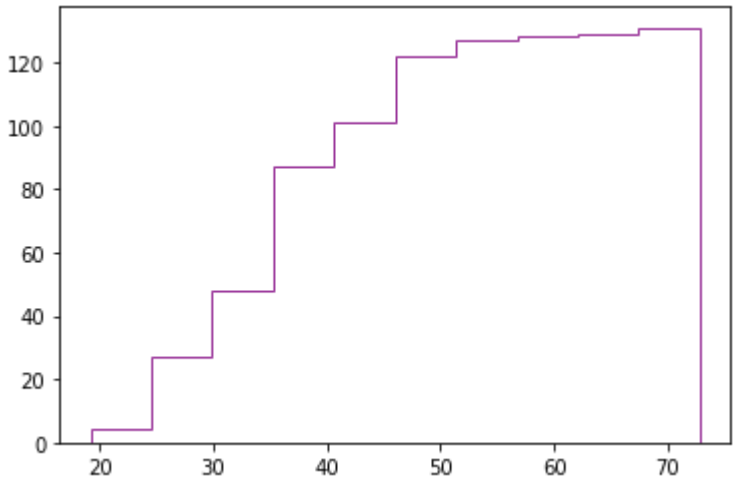
Out[30]: <AxesSubplot:xlabel='year'>



In [ ]:
```
sns.pairplot(data=year_runs)
plt.show()
```

In [48]:
```
pace = all_data['pace']
```

In [56]:
```
plt.hist(all_data['duration'], cumulative=True, label='CDF',
         histtype='step', alpha=0.8, color='purple')
plt.show()
```



In [ ]: