SWE 402 Assignment : Python for Data Analysis

Team Members: ¶

1. Khor Jia Cheng SWE1809651 (Leader)
2. Leong Wei Jun SWE1809654

Dataset Used for the Analysis

The dataset used for this assignmnent is from Kaggle. Link of the website : [https://www.kaggle.com/jesendo/malaysia-covid19 (https://www.kaggle.com/jesendo/malaysia-covid19)](https://www.kaggle.com/jesendo/malaysia-covid19)

Exploratory Data Analysis

Before we start for the data analysis,the dataset must be prepared and cleaned. First and foremost, excecute the code in the cell below to load the packages to run the rest of this notebook.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns                         #visualisation
import matplotlib.pyplot as plt               #visualisation
import numpy.random as nr
import math
%matplotlib inline

sns.set(color_codes=True)
```

The code in the cell below loads the datasets into their respectives dataframe, read the first five records, and check the length of the data.The datasets involves COVID-199 death cases, COVID-19 cases, type of test for COVID-19, and type of vaccination.

In [2]:
```
df_case = pd.read_csv("cases_malaysia.csv")
df_death = pd.read_csv("deaths_malaysia.csv")
df_test = pd.read_csv("tests_malaysia.csv")
df_vax = pd.read_csv("vax_malaysia.csv")
```

In [3]:
```
df_case.head(5)
```

Out[3]:

	date	cases_new	cases_import	cases_recovered	cases_active	cases_cluster	cases_unvax
0	2020-01-25	4	4	0	4	0.0	4.0
1	2020-01-26	0	0	0	4	0.0	0.0
2	2020-01-27	0	0	0	4	0.0	0.0
3	2020-01-28	0	0	0	4	0.0	0.0
4	2020-01-29	3	3	0	7	0.0	3.0

In [4]:
```
df_test.head(5)
```

Out[4]:

	date	rtk-ag	pcr
0	2020-01-24	0	2
1	2020-01-25	0	5
2	2020-01-26	0	14
3	2020-01-27	0	24
4	2020-01-28	0	53

In [5]:
```
df_death = df_death.drop(['deaths_bid','deaths_new_dod', 'deaths_bid_dod','deaths_unvax','deaths_pvax','deaths_fvax','deaths_boost','deaths_tat'], axis=1)
df_death.head(5)
```

Out[5]:

	date	deaths_new
0	2020-03-17	2
1	2020-03-18	0
2	2020-03-19	0
3	2020-03-20	1
4	2020-03-21	4

In [6]:
```
df_death.head(5)
```

Out[6]:

	date	deaths_new
0	2020-03-17	2
1	2020-03-18	0
2	2020-03-19	0
3	2020-03-20	1
4	2020-03-21	4

In [7]:
```
len(df_death)
```

Out[7]: 646

Removed Unused Data

The cell below drops the column of covid cases dataset and vaccination dataset that may not used for the analysis.

In [8]:
```
df_case = df_case.drop(['cases_cluster', 'cases_child','cases_adolescent','cas
es_adult','cases_elderly','cases_0_4','cases_5_11','cases_12_17','cases_18_29'
,'cases_30_39','cases_40_49','cases_50_59','cases_60_69','cases_70_79','cases_
80','cluster_import','cluster_religious','cluster_community','cluster_highRis
k','cluster_education','cluster_detentionCentre','cluster_workplace' ], axis=1
)
df_case.head(5)
```

Out[8]:

	date	cases_new	cases_import	cases_recovered	cases_active	cases_unvax	cases_pvax	c
0	2020-01-25	4	4	0	4	4.0	0.0	
1	2020-01-26	0	0	0	4	0.0	0.0	
2	2020-01-27	0	0	0	4	0.0	0.0	
3	2020-01-28	0	0	0	4	0.0	0.0	
4	2020-01-29	3	3	0	7	3.0	0.0	

In [9]:
```
df_vax = df_vax.drop(['daily_partial','daily_full','daily_partial_child', 'dai
ly_partial_child','daily_partial_child','daily_full_child','daily_booster','cu
mul_partial','cumul_full','cumul','cumul_partial_child','cumul_full_child','cu
mul_booster'], axis=1)
df_vax.head(5)
```

Out[9]:

	date	daily	pfizer1	pfizer2	pfizer3	sinovac1	sinovac2	sinovac3	astra1	astra2	astra3	s
0	2021-02-24	65	61	1	0	0	2	0	0	0	0	
1	2021-02-25	1151	1147	0	0	0	2	0	0	0	0	
2	2021-02-26	4068	4057	1	0	0	2	0	1	0	0	
3	2021-02-27	6716	6692	1	0	0	5	0	0	0	0	
4	2021-02-28	6717	6708	1	0	0	4	0	0	0	0	

Data Grouping

The cell below groups the total of different types of vaccine dose taken daily by combining multiple columns.

In [10]:
```
df_vax['pfizer'] =  df_vax['pfizer1']+ df_vax['pfizer2']+ df_vax['pfizer3']
df_vax['sinovac'] =  df_vax['sinovac1']+ df_vax['sinovac2']+ df_vax['sinovac3'
]
df_vax['astra'] =  df_vax['astra1']+ df_vax['astra2']+ df_vax['astra3']
df_vax['sinopharm'] =  df_vax['sinopharm1']+ df_vax['sinopharm2']+ df_vax['sin
opharm3']
df_vax['tot_cansino'] =  df_vax['cansino']+ df_vax['cansino3']
df_vax['pending'] =  df_vax['pending1']+ df_vax['pending2']+ df_vax['pending3'
]

df_vax = df_vax.drop(['pfizer1','pfizer2','pfizer3', 'sinovac1','sinovac2','si
novac3','astra1','astra2','astra3','sinopharm1','sinopharm2','sinopharm3','can
sino','cansino3','pending1','pending2','pending3'], axis=1)
df_vax
```

Out[10]:

	date	daily	pfizer	sinovac	astra	sinopharm	tot_cansino	pending
0	2021-02-24	65	62	2	0	0	0	1
1	2021-02-25	1151	1147	2	0	0	0	2
2	2021-02-26	4068	4058	2	1	0	0	7
3	2021-02-27	6716	6693	5	0	0	0	18
4	2021-02-28	6717	6709	4	0	0	0	4
...	...	...	...	...	...	...	...	...
297	2021-12-18	72164	3672	1906	108	128	134	66216
298	2021-12-19	79498	56218	10148	2095	84	209	10744
299	2021-12-20	146136	121474	21211	1689	249	119	1394
300	2021-12-21	168138	144477	19502	2104	206	110	1739
301	2021-12-22	177272	146761	19917	3010	219	187	7178

302 rows × 8 columns

Recode Name

Notice that one of the column names in df_test contain the '-' character. Python will not correctly recognize character strings containing '-'. Rather, such a name will be recognized as two character strings. The same problem will occur with column values containing many special characters including, '-', ',', '*', '/', '|', '>', '<', '@', '!' etc. If such characters appear in column names of values, they must be replaced with another character.

Execute the code in the cell below to replace the '-' characters by '_' :

In [11]:
```
df_test.columns = [str.replace('-', '_') for str in df_test.columns]
print(df_test.columns)
```

```
Index(['date', 'rtk_ag', 'pcr'], dtype='object')
```

Combining Multiple Dataframe to One Dataframe After removing the unused data, we may combining all of the dataframes to one dataframe called **df** using inner join with the use of **date** attribute.

In [12]:
```
df_vax_test = pd.merge(df_vax, df_test, on='date', how='inner')
df_vax_test
```

Out[12]:

	date	daily	pfizer	sinovac	astra	sinopharm	tot_cansino	pending	rtk_ag	pcr
0	2021-02-24	65	62	2	0	0	0	1	44841	30281
1	2021-02-25	1151	1147	2	0	0	0	2	47439	38490
2	2021-02-26	4068	4058	2	1	0	0	7	46207	34014
3	2021-02-27	6716	6693	5	0	0	0	18	34017	29550
4	2021-02-28	6717	6709	4	0	0	0	4	32557	26969
...	...	...	...	...	...	...	...	...	...	...
295	2021-12-16	174433	18946	3813	325	268	150	150931	91581	33616
296	2021-12-17	162223	9730	2690	144	253	138	149268	78413	31077
297	2021-12-18	72164	3672	1906	108	128	134	66216	62603	30918
298	2021-12-19	79498	56218	10148	2095	84	209	10744	67749	24185
299	2021-12-20	146136	121474	21211	1689	249	119	1394	114220	31012

300 rows × 10 columns

In [13]:
```
df_death_case= pd.merge(df_death, df_case, on='date', how='inner')
df_death_case
```

Out[13]:

	date	deaths_new	cases_new	cases_import	cases_recovered	cases_active	cases_unvax
0	2020-03-17	2	120	3	7	622	120.0
1	2020-03-18	0	117	8	11	728	117.0
2	2020-03-19	0	110	5	15	823	110.0
3	2020-03-20	1	130	6	12	940	130.0
4	2020-03-21	4	153	11	27	1062	153.0
...	...	...	...	...	...	...	...
641	2021-12-18	29	4083	34	5435	54000	1024.0
642	2021-12-19	19	3108	37	3701	53389	692.0
643	2021-12-20	43	2589	46	3810	52161	653.0
644	2021-12-21	57	3140	58	4278	51023	746.0
645	2021-12-22	29	3519	100	5118	49395	NaN

646 rows × 10 columns

In [14]:
```
df =  pd.merge(df_death_case, df_vax_test, on='date', how='inner')
df
```

Out[14]:

	date	deaths_new	cases_new	cases_import	cases_recovered	cases_active	cases_unvax
0	2021-02-24	12	3545	1	3331	30572	3545.0
1	2021-02-25	13	1924	6	3752	28738	1924.0
2	2021-02-26	10	2253	7	3085	27903	2253.0
3	2021-02-27	10	2364	1	3320	26937	2364.0
4	2021-02-28	9	2437	1	3251	26118	2437.0
...	...	...	...	...	...	...	...
295	2021-12-16	37	4262	36	4985	56156	955.0
296	2021-12-17	18	4362	28	5098	55380	969.0
297	2021-12-18	29	4083	34	5435	54000	1024.0
298	2021-12-19	19	3108	37	3701	53389	692.0
299	2021-12-20	43	2589	46	3810	52161	653.0

300 rows × 19 columns

After all of the dataframe had been joined into **df**, we check on the length of the dataframe, which is 300 records.

In [15]:
```
df.count()
```

Out[15]:
```
date               300
deaths_new         300
cases_new          300
cases_import       300
cases_recovered    300
cases_active       300
cases_unvax        300
cases_pvax         300
cases_fvax         300
cases_boost        300
daily              300
pfizer             300
sinovac            300
astra              300
sinopharm          300
tot_cansino        300
pending            300
rtk_ag             300
pcr                300
dtype: int64
```

In [16]:
```
df.tail(5)
```

Out[16]:

	date	deaths_new	cases_new	cases_import	cases_recovered	cases_active	cases_unvax
295	2021-12-16	37	4262	36	4985	56156	955.0
296	2021-12-17	18	4362	28	5098	55380	969.0
297	2021-12-18	29	4083	34	5435	54000	1024.0
298	2021-12-19	19	3108	37	3701	53389	692.0
299	2021-12-20	43	2589	46	3810	52161	653.0

Check the types of data

Here we check for the datatypes and found that cases_unvax, case_pvax, case_fvax, and case_boost are all stored as float. However, the date for three of these columns could be treated as int as the cases were all in whole number.

In [17]:
```
df.dtypes
```

Out[17]:
```
date                 object
deaths_new            int64
cases_new             int64
cases_import          int64
cases_recovered       int64
cases_active          int64
cases_unvax         float64
cases_pvax          float64
cases_fvax          float64
cases_boost         float64
daily                 int64
pfizer                int64
sinovac               int64
astra                 int64
sinopharm             int64
tot_cansino           int64
pending               int64
rtk_ag                int64
pcr                   int64
dtype: object
```

In [18]:
```
df['cases_unvax'] = df['cases_unvax'].astype(int)
df['cases_pvax'] = df['cases_pvax'].astype(int)
df['cases_fvax'] = df['cases_fvax'].astype(int)
df['cases_boost'] = df['cases_boost'].astype(int)
df.dtypes
```

Out[18]:
```
date               object
deaths_new          int64
cases_new           int64
cases_import        int64
cases_recovered     int64
cases_active        int64
cases_unvax         int64
cases_pvax          int64
cases_fvax          int64
cases_boost         int64
daily               int64
pfizer              int64
sinovac             int64
astra               int64
sinopharm           int64
tot_cansino         int64
pending             int64
rtk_ag              int64
pcr                 int64
dtype: object
```

Renaming Colums

In this instance, most of the column names are very confusing to read, so I just tweaked their column names. This is a good approach it improves the readability of the data set.

In [19]:
```
df = df.rename(columns={"deaths_new": "death_cases","cases_new": "new_cases",
"cases_import": "import_cases","cases_recovered": "recovered_cases","cases_act
ive": "active_cases","cases_unvax": "unvax_cases","cases_pvax": "pvax_cases",
"cases_fvax": "fvax_cases","cases_boost": "boostvax_cases","daily": "daily_va
x" })
df.head(10)
```

Out[19]:

	date	death_cases	new_cases	import_cases	recovered_cases	active_cases	unvax_cases
0	2021-02-24	12	3545	1	3331	30572	3545
1	2021-02-25	13	1924	6	3752	28738	1924
2	2021-02-26	10	2253	7	3085	27903	2253
3	2021-02-27	10	2364	1	3320	26937	2364
4	2021-02-28	9	2437	1	3251	26118	2437
5	2021-03-01	5	1828	7	2486	25456	1828
6	2021-03-02	6	1555	3	2528	24474	1555
7	2021-03-03	7	1745	2	2276	23939	1744
8	2021-03-04	5	2063	9	2922	23077	2062
9	2021-03-05	6	2154	5	3275	21948	2151

In [20]:
```
df.shape
```

Out[20]: (300, 19)

Check for Duplicate Data

The cell below was to check whether there is any duplicate data.

In [21]:
```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)

df.count()
```

```
number of duplicate rows:  (0, 19)
```

Out[21]:
```
date               300
death_cases        300
new_cases          300
import_cases       300
recovered_cases    300
active_cases       300
unvax_cases        300
pvax_cases         300
fvax_cases         300
boostvax_cases     300
daily_vax          300
pfizer             300
sinovac            300
astra              300
sinopharm          300
tot_cansino        300
pending            300
rtk_ag             300
pcr                300
dtype: int64
```

From the result above, there is no any duplicate data.

Check for Null Value

The cell below checks whether there is any null of missing values in the dataset. In this case, there is no any null or missing values in the dataset.

In [22]:
```
df.isnull().sum().sum()
```

Out[22]:
```
0
```

In [23]:
```
print(df.isnull().sum())
# That day de haven update
```

```
date               0
death_cases        0
new_cases          0
import_cases       0
recovered_cases    0
active_cases       0
unvax_cases        0
pvax_cases         0
fvax_cases         0
boostvax_cases     0
daily_vax          0
pfizer             0
sinovac            0
astra              0
sinopharm          0
tot_cansino        0
pending            0
rtk_ag             0
pcr                0
dtype: int64
```

In [24]:
```
df = df.dropna()    # Dropping the missing values.
df.count()
```

Out[24]:
```
date               300
death_cases        300
new_cases          300
import_cases       300
recovered_cases    300
active_cases       300
unvax_cases        300
pvax_cases         300
fvax_cases         300
boostvax_cases     300
daily_vax          300
pfizer             300
sinovac            300
astra              300
sinopharm          300
tot_cansino        300
pending            300
rtk_ag             300
pcr                300
dtype: int64
```

In [25]:
```
df.tail(5)
```

Out[25]:

	date	death_cases	new_cases	import_cases	recovered_cases	active_cases	unvax_cases
295	2021-12-16	37	4262	36	4985	56156	955
296	2021-12-17	18	4362	28	5098	55380	969
297	2021-12-18	29	4083	34	5435	54000	1024
298	2021-12-19	19	3108	37	3701	53389	692
299	2021-12-20	43	2589	46	3810	52161	653

In [26]:
```
print(df.isnull().sum())   # After dropping the values
```

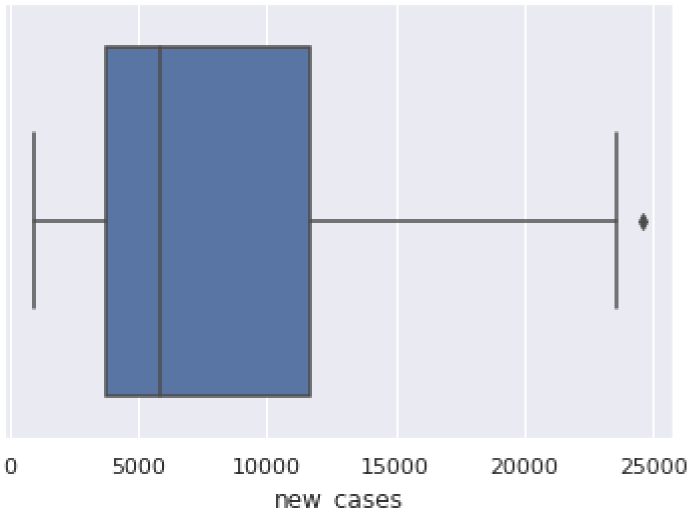
```
date               0
death_cases        0
new_cases          0
import_cases       0
recovered_cases    0
active_cases       0
unvax_cases        0
pvax_cases         0
fvax_cases         0
boostvax_cases     0
daily_vax          0
pfizer             0
sinovac            0
astra              0
sinopharm          0
tot_cansino        0
pending            0
rtk_ag             0
pcr                0
dtype: int64
```

Finding Outlier

An outlier is a point or set of points that are different from other points. Sometimes they can be very high or very low. It's often a good idea to detect outliers as it is one of the primary reasons for resulting in a less accurate model.Often outliers can be seen with visualizations using a box plot. Shown below are the box plot of new COVID-19 cases, and death cases. Herein all the plots, you can find some points are outside the box they are outliers. In this case, the outliers were not removed as the data for COVID-19 cases can be zero or very high.

In [27]:
```
sns.boxplot(x=df['new_cases'])
```

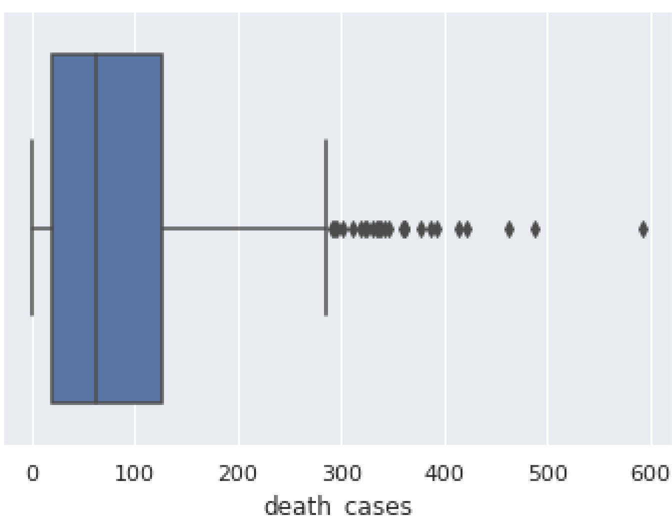
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4867ab39d0>



From the boxplot above, we can found that there is only little outliers (black points out of the box plot).

In [28]:
```
sns.boxplot(x=df['death_cases'])
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4866709d50>



From the boxplot above, we can found that there are a few outliers (black points out of the box plot).

In [29]:
```
df.describe()
```

Out[29]:

	death_cases	new_cases	import_cases	recovered_cases	active_cases	unvax_cases
count	300.000000	300.000000	300.00000	300.000000	300.000000	300.000000
mean	100.180000	8111.050000	13.29000	7938.380000	95734.203333	4650.690000
std	107.979843	6299.350336	10.30014	6303.910502	76647.724219	3559.815374
min	1.000000	941.000000	0.00000	1052.000000	14025.000000	653.000000
25%	19.000000	3741.250000	6.00000	3328.250000	37161.500000	1479.000000
50%	63.000000	5820.000000	10.00000	5568.000000	67044.000000	3662.500000
75%	126.750000	11685.750000	18.00000	11376.000000	136673.500000	7159.750000
max	592.000000	24599.000000	64.00000	24855.000000	263871.000000	12685.000000

From the result above, we can found that the maximum and the minimum value of the data were quite logic. For example, the minimum and the maximum number of death case due to Covid-19 could be 1 and 592, which is not too vary from the 3rd quartile.

In [30]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
death_cases            107.75
new_cases             7944.50
import_cases            12.00
recovered_cases       8047.75
active_cases         99512.00
unvax_cases           5680.75
pvax_cases            1632.00
fvax_cases            3955.25
boostvax_cases           0.00
daily_vax           247982.75
pfizer              136397.75
sinovac             114785.25
astra                24125.00
sinopharm              150.00
tot_cansino            454.50
pending                 95.25
rtk_ag               51947.00
pcr                  28838.75
dtype: float64
```

Plot Different Features Against One Other

The cell below create new columns which are **year** and **month** in the dataframe **df**.

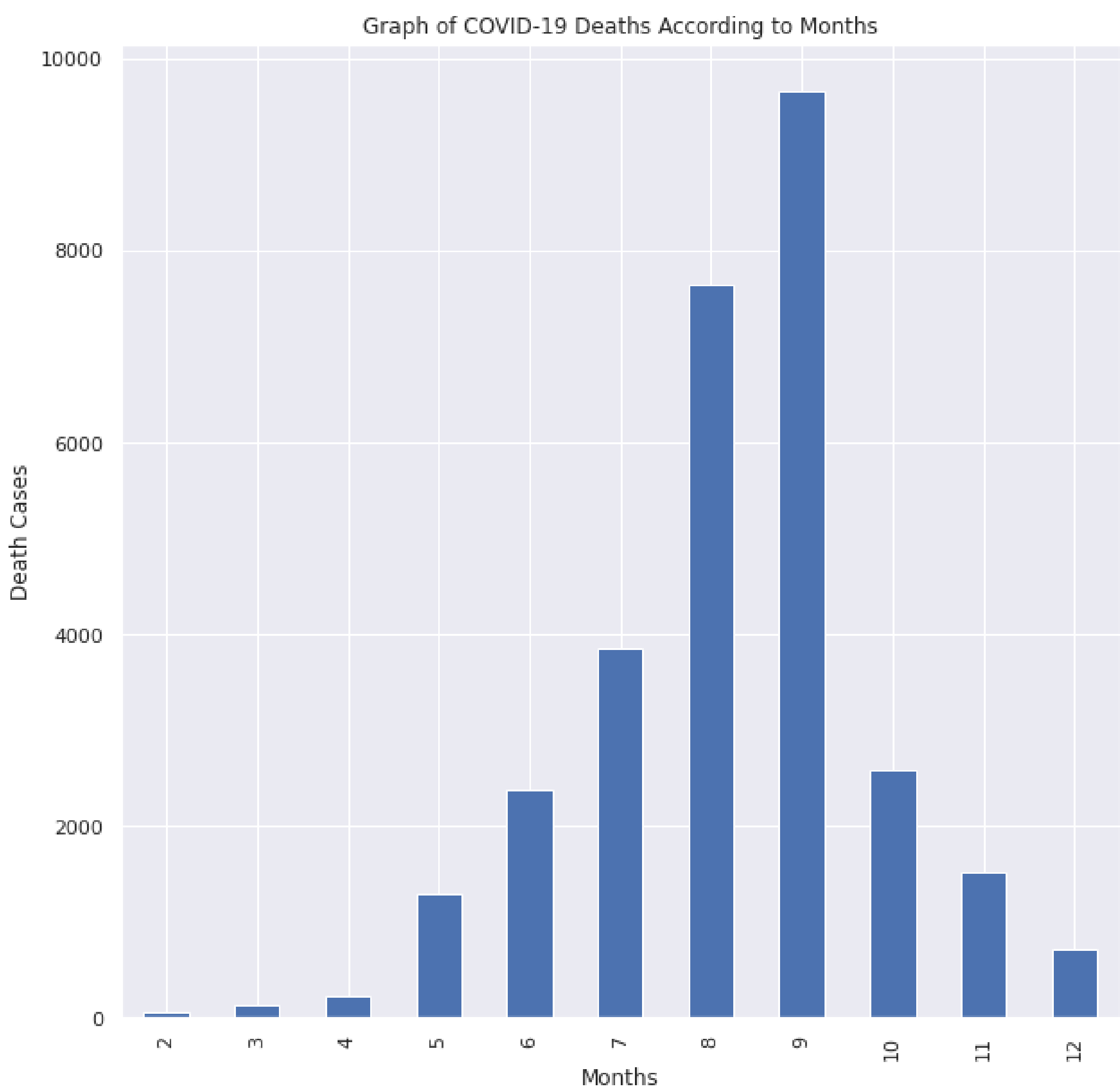
In [31]:
```
df['year'] = pd.DatetimeIndex(df['date']).year
df['month'] = pd.DatetimeIndex(df['date']).month
```

Bar Chart

Bar graphs are used to compare things between different groups or to track changes over time.In this case, the number of deaths caused by COVID-19, the number of active COVID-19 cases over months,comparison between COVID-19 tests and were visualize using bar chart.

In [32]:
```
fig = df.groupby('month').death_cases.sum().plot(kind='bar', title='Graph of C
OVID-19 Deaths According to Months', figsize=(10,10))
fig.set_ylabel("Death Cases")
fig.set_xlabel("Months")
```

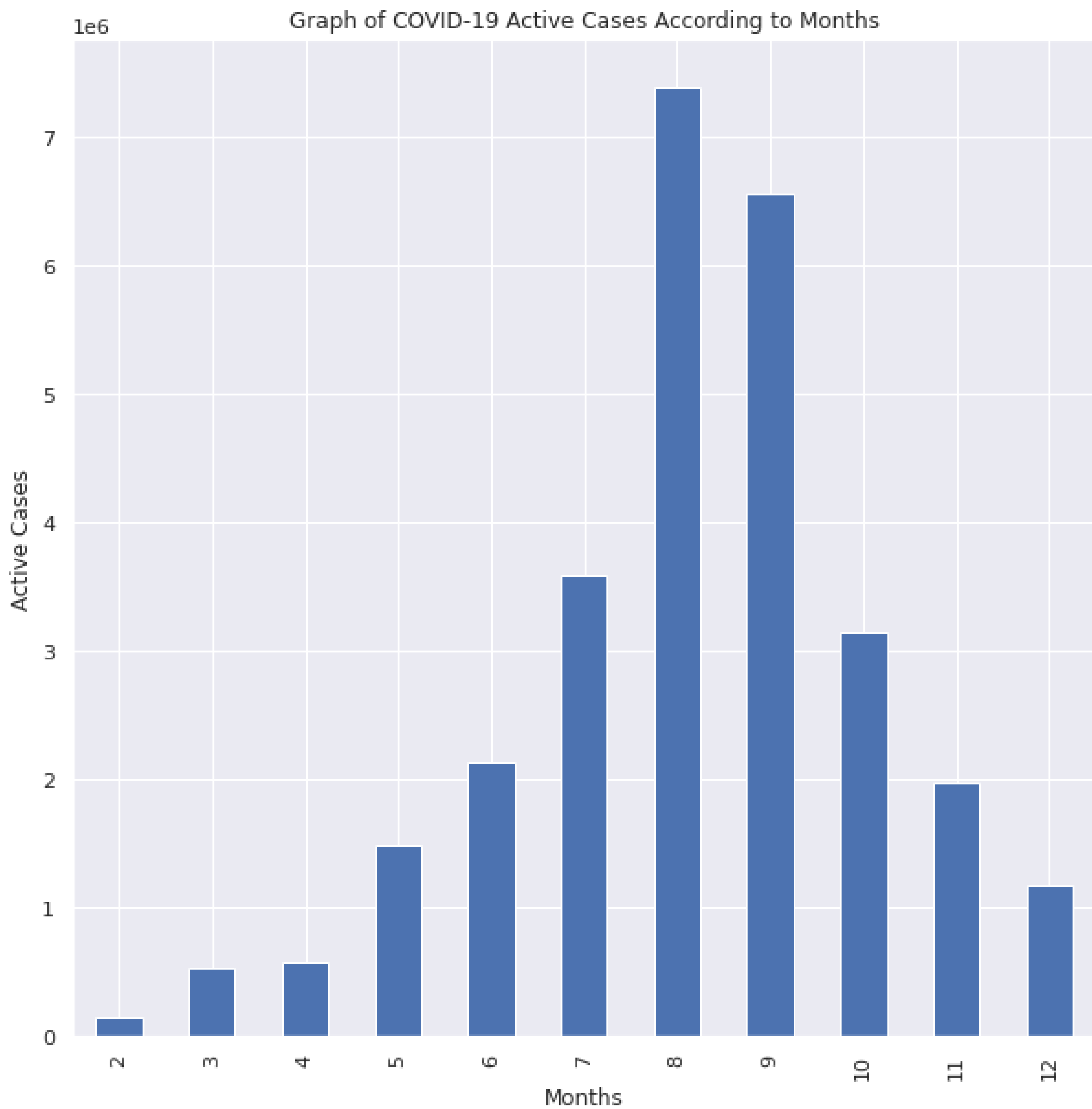
Out[32]:
```
Text(0.5, 0, 'Months')
```



Based on the graph of COVID-19 Deaths According to Months, we can found that the death cases caused by COVID-19 is highest in **September**, follow by **August**. Besides that, the death case in February is the least as there is only the last few days of the data were included.

In [33]:
```
fig = df.groupby('month').active_cases.sum().plot(kind='bar', title='Graph of COVID-19 Active Cases According to Months', figsize=(10,10))
fig.set_ylabel("Active Cases")
fig.set_xlabel("Months")
```

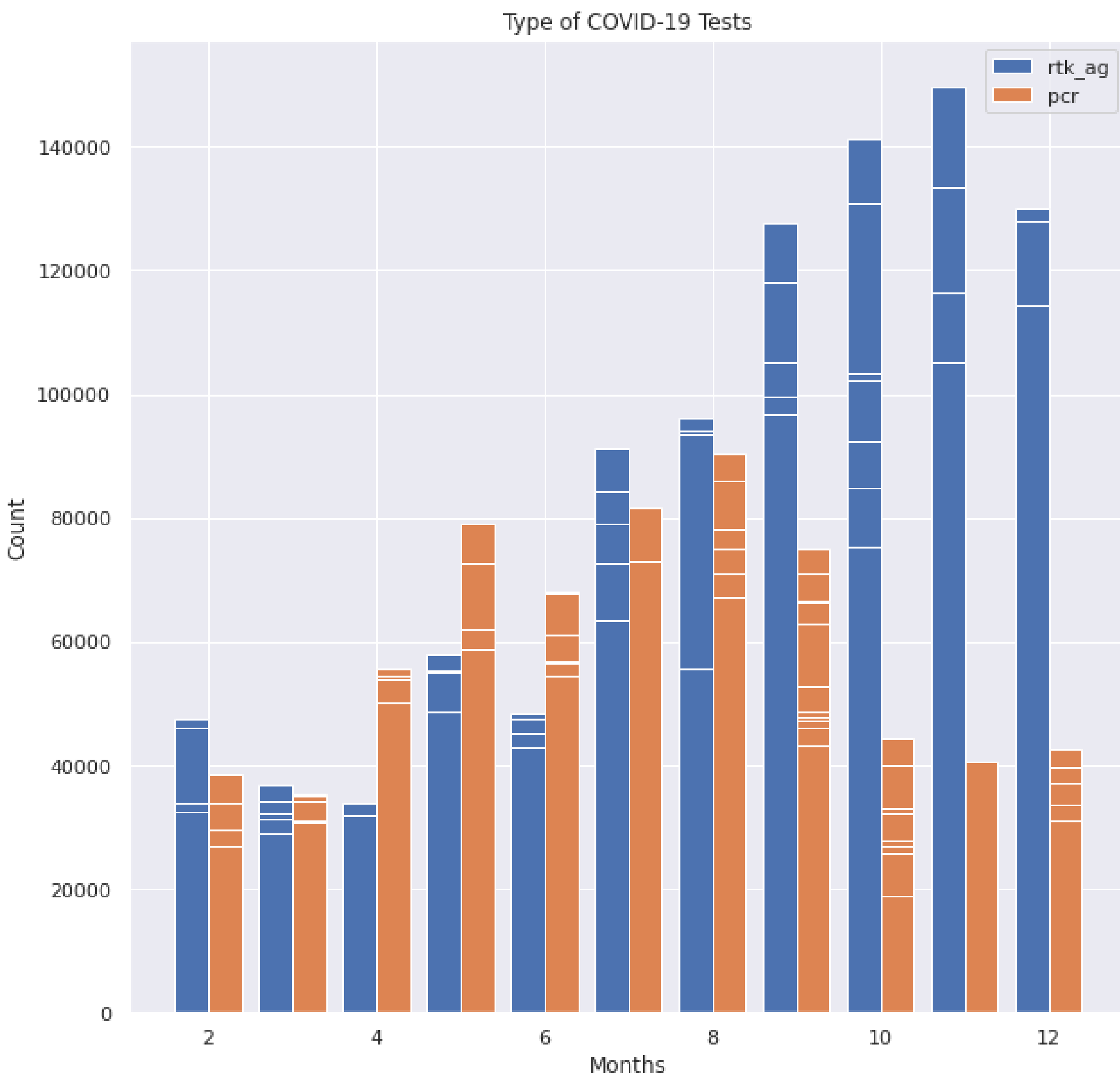
Out[33]: Text(0.5, 0, 'Months')



Based on the graph of Graph of COVID-19 Active Cases According to Months, we can found that in August, the COVID-19 Active Cases is highest, follow by the COVID-19 active cases in September. Besides that, the active case in February is the least as there is only the last few days of the data were included.

In [34]:
```
df.groupby('month').sum()

fig = plt.figure(figsize=(10,10))

plt.title("Type of COVID-19 Tests")
plt.bar(df['month'] - 0.2,df['rtk_ag'],0.4,label ='rtk_ag')
plt.bar(df['month']+ 0.2 ,df['pcr'],0.4,label ='pcr')
plt.xlabel("Months")
plt.ylabel("Count")

plt.legend()
plt.plot()
```

Out[34]: []



Based on the the bar chart above, we can found that most of the people prefer rtk-ag test as their COVID-19 tests from **July** and onwards while most of the people prefer to use pcr test during **May** and **June**

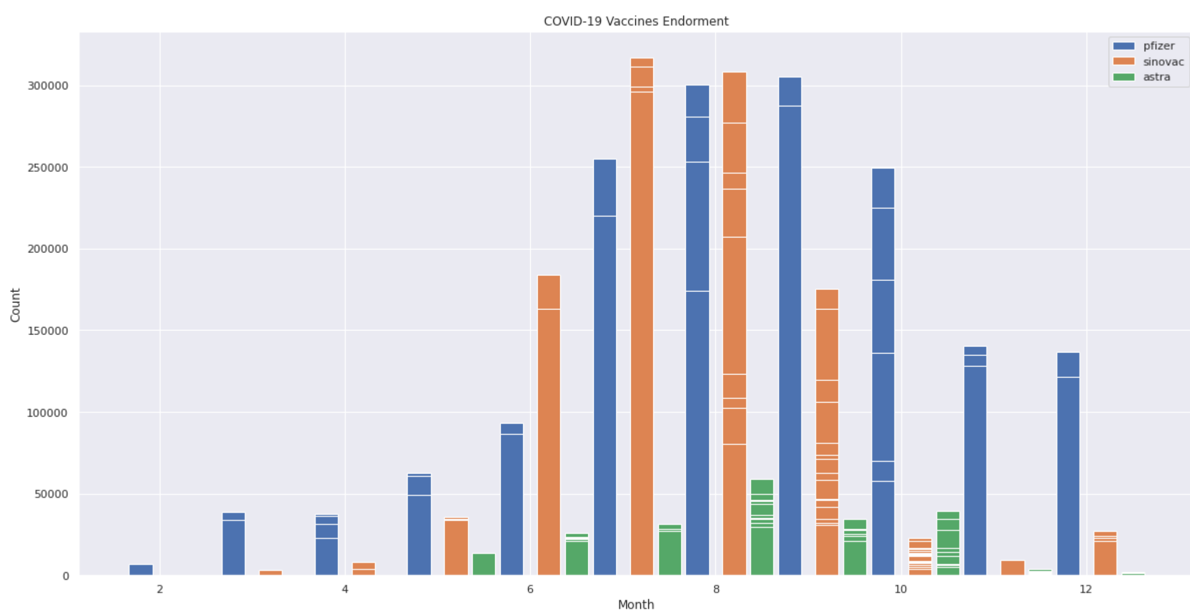
In [35]:
```
selected_columns = df[["month","pfizer","sinovac","astra","sinopharm","tot_cansino","pending"]]

df_vax_type = selected_columns.copy()

df_vax_type.head(5)
```

Out[35]:

	month	pfizer	sinovac	astra	sinopharm	tot_cansino	pending
0	2	62	2	0	0	0	1
1	2	1147	2	0	0	0	2
2	2	4058	2	1	0	0	7
3	2	6693	5	0	0	0	18
4	2	6709	4	0	0	0	4

In [36]:
```
df_vax_type.groupby('month').sum()

fig = plt.figure(figsize=(20,10))


plt.title("COVID-19 Vaccines Endorment")
plt.bar(df_vax_type['month'] - 0.2,df_vax_type['pfizer'],0.25,label ='pfizer')
plt.bar(df_vax_type['month']+ 0.2 ,df_vax_type['sinovac'],0.25,label ='sinovac')
plt.bar(df_vax_type['month']+ 0.5 ,df_vax_type['astra'],0.25,label ='astra')
plt.xlabel("Month")
plt.ylabel("Count")

plt.legend()
plt.plot()
```

Out[36]: []



Based on the bar chart above, we can found that majortity of the peoples prefer **Sinovac** for thier COVID-19 Vaccine endorsement, following by **Pfizer** and **AstraZeneca**.

Line Graph

Line graph was commonly used to create a graphical depiction of changes in values over time. In this case, the COVID-19 cases and the comparison between death cases, recovered cases and active cases were represented using line graph.

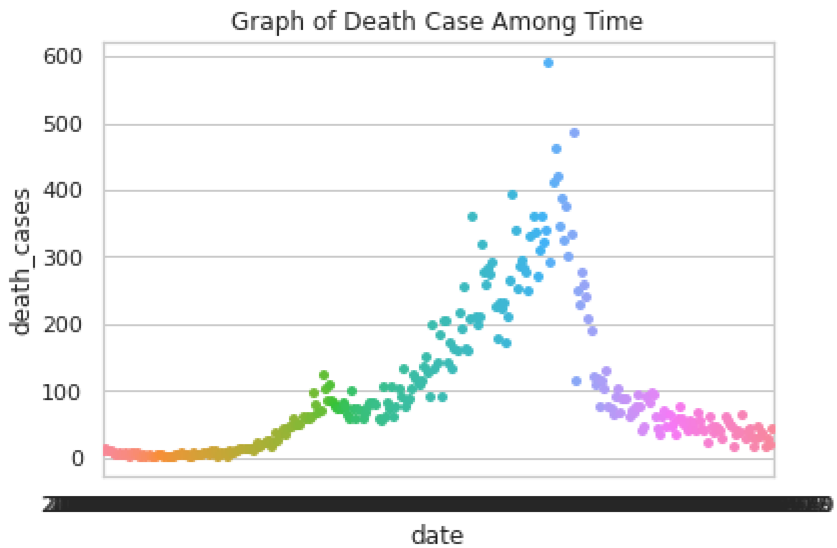
Copy of data to new dataframe called df_cases

In [37]:
```
# Group particular columns to df_cases.
selected_columns = df[["month","active_cases","new_cases","recovered_cases","death_cases"]]
df_cases = selected_columns.copy()

df_cases.head(5)
```

Out[37]:

	month	active_cases	new_cases	recovered_cases	death_cases
0	2	30572	3545	3331	12
1	2	28738	1924	3752	13
2	2	27903	2253	3085	10
3	2	26937	2364	3320	10
4	2	26118	2437	3251	9

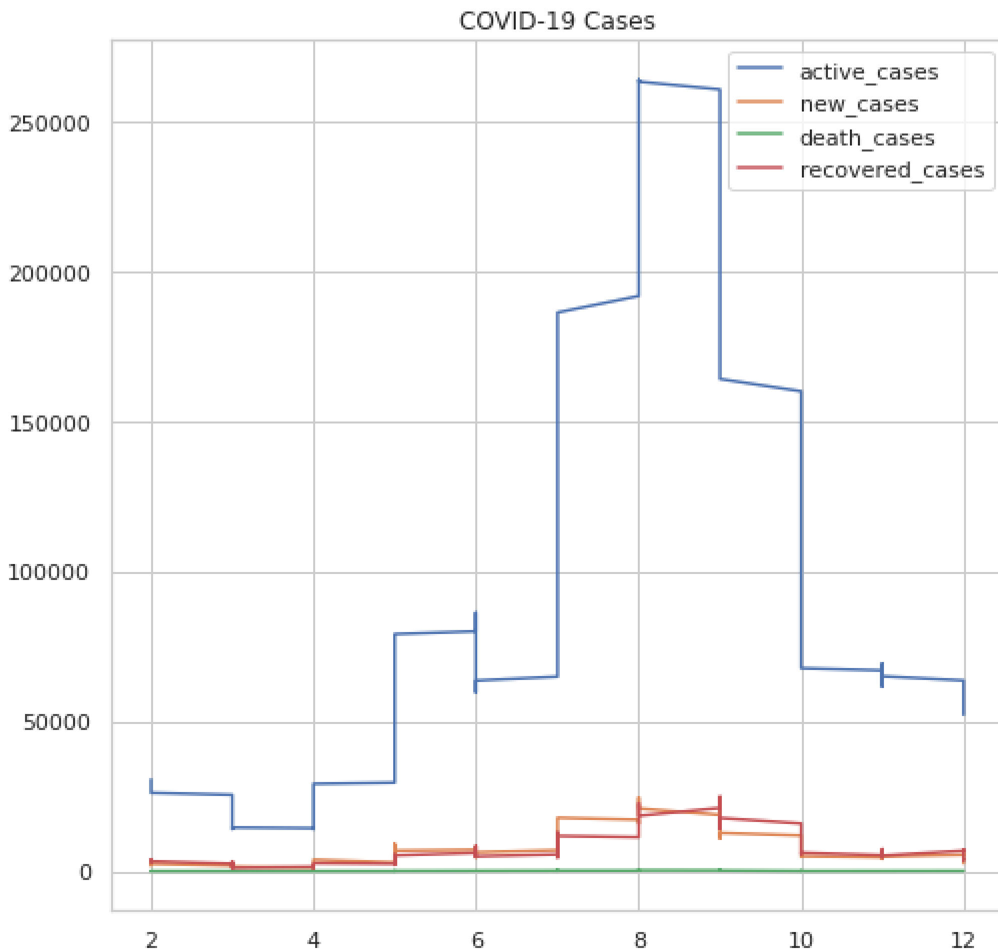
In [38]:
```
# use to set style of background of plot
sns.set(style="whitegrid")

# plotting strip plot with seaborn
# deciding the attributes of dataset on
# which plot should be made
ax = sns.swarmplot(x='date', y='death_cases', data=df)

# giving title to the plot
plt.title('Graph of Death Case Among Time')

# function to show plot
plt.show()
```



Based on the Graph of death case among time, we can found that there is an increase in COVID-19 death cases from 100+ to 500+. After a few days, the death case due to COVID-19 Pandemic decreased back to less than 100 of cases everyday.

In [39]:
```
df_cases.groupby('month').sum()

fig = plt.figure(figsize=(8,8))
plt.title("COVID-19 Cases")
plt.plot(df['month'],df['active_cases'],label ='active_cases')
plt.plot(df['month'],df['new_cases'],label ='new_cases')
plt.plot(df['month'],df['death_cases'],label ='death_cases')
plt.plot(df['month'],df['recovered_cases'],label ='recovered_cases')

plt.legend()
plt.plot()
```

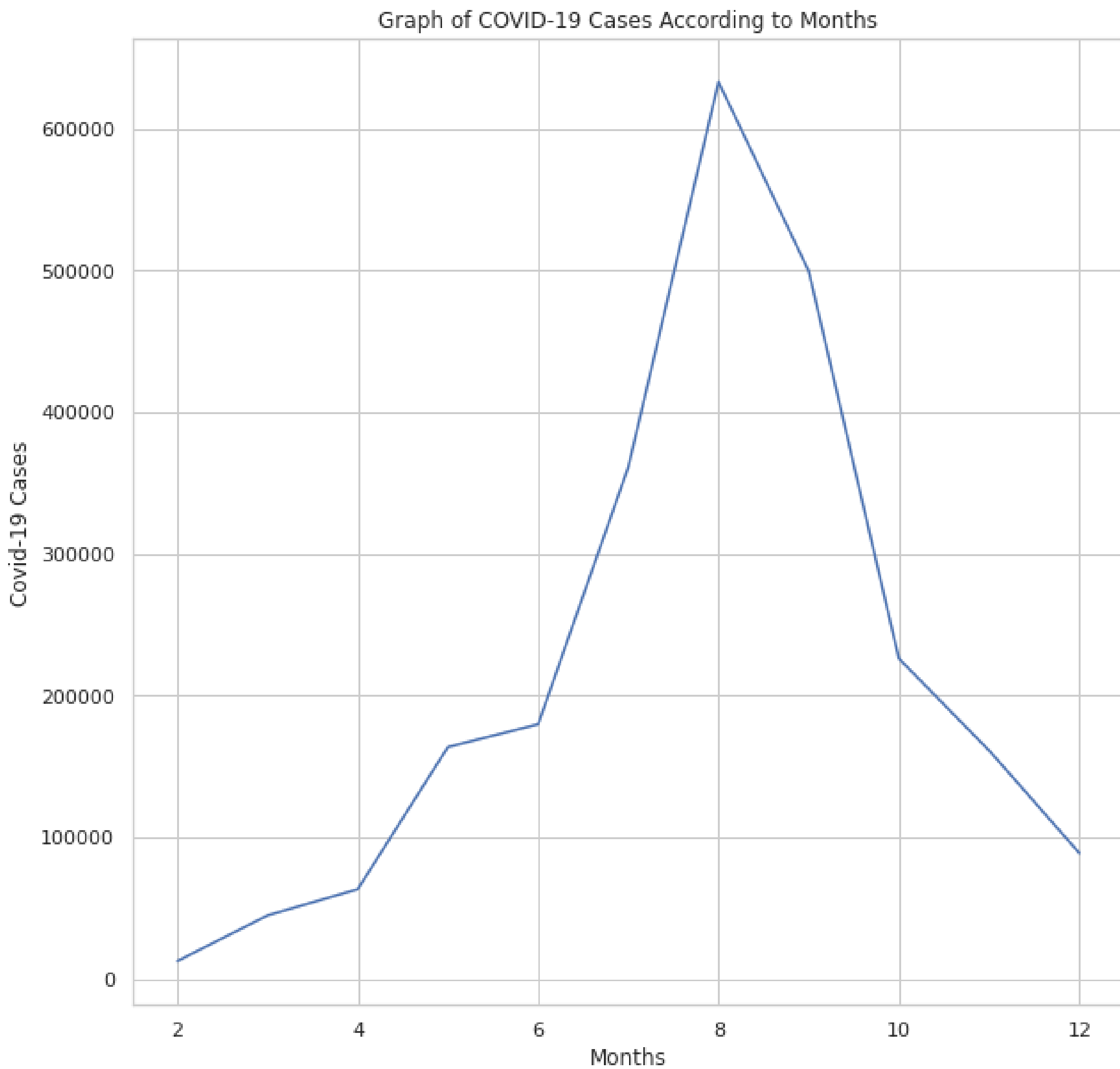
Out[39]: []



Based on the line graph above, we can found that the active case is largest in terms of numbers, following by new cases and recovered case where both of them were overlapping. Besides that, the death case due to COVID-19 Pandemic is the least.

In [40]:
```
fig = df.groupby('month').new_cases.sum().plot(title='Graph of COVID-19 Cases
 According to Months', figsize=(10,10))
fig.set_ylabel("Covid-19 Cases")
fig.set_xlabel("Months")
```

Out[40]: Text(0.5, 0, 'Months')



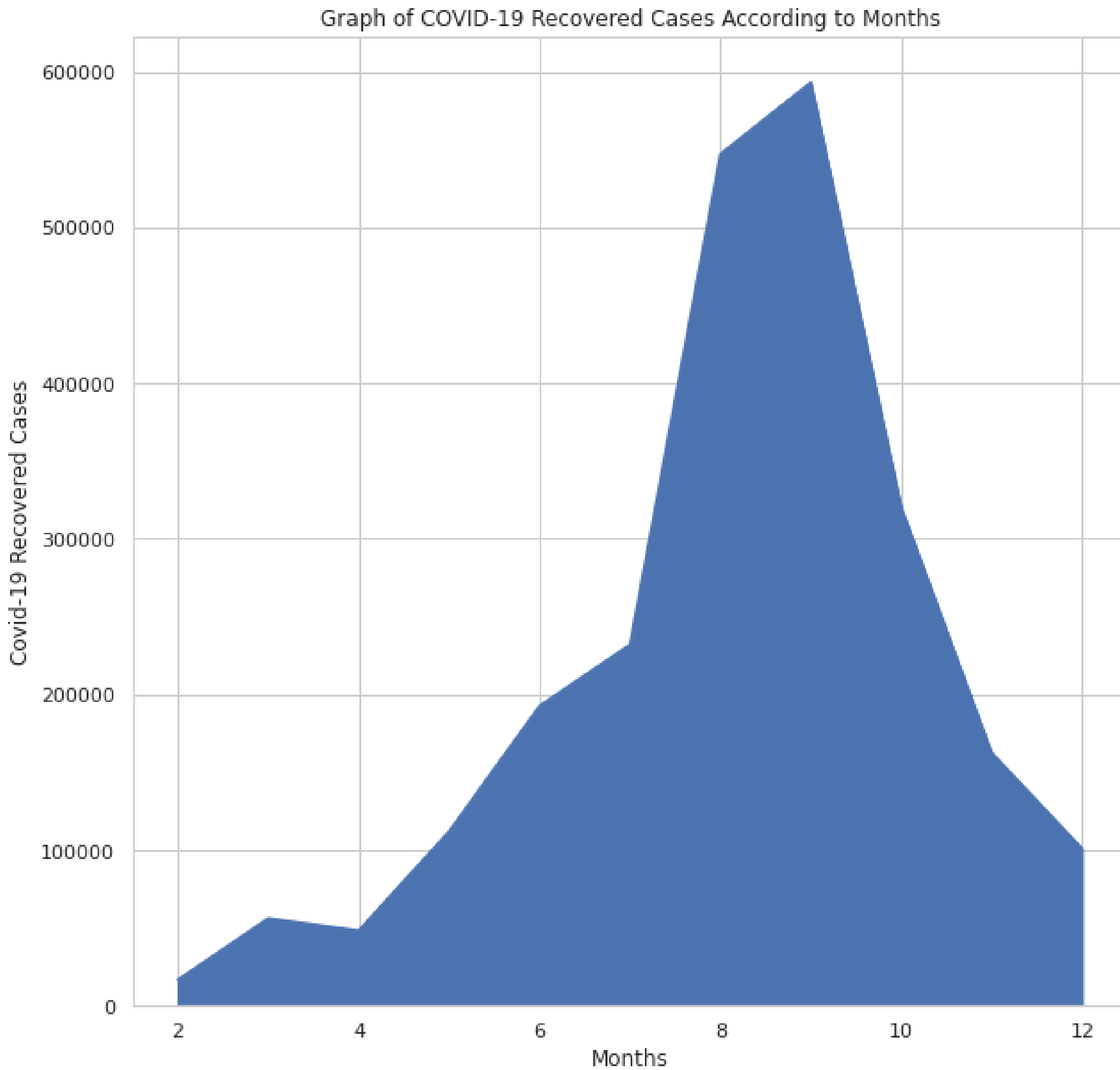
Based on the Graph of COVID-19 Cases According to Months, we can found that there is an increase of number of new COVID-19 cases from **February** and reached in the greatest number of cases in **August**. After that, the COVID-19 cases started to decrease back within 100000 in **December**.

Area Graph

Area graphy was widely used to show the rise and fall of various data series over tim and conveying total amounts over time as well as some sub-categorical breakdowns (but only to a point.In this case, area graph was used to show the Covid-19 recovered case according to month and Covid-19 vaccination endorsement according to months.

In [41]:
```
fig = df.groupby('month').recovered_cases.sum().plot(kind='area', title='Graph of COVID-19 Recovered Cases According to Months', figsize=(10,10))
fig.set_ylabel("Covid-19 Recovered Cases")
fig.set_xlabel("Months")
```

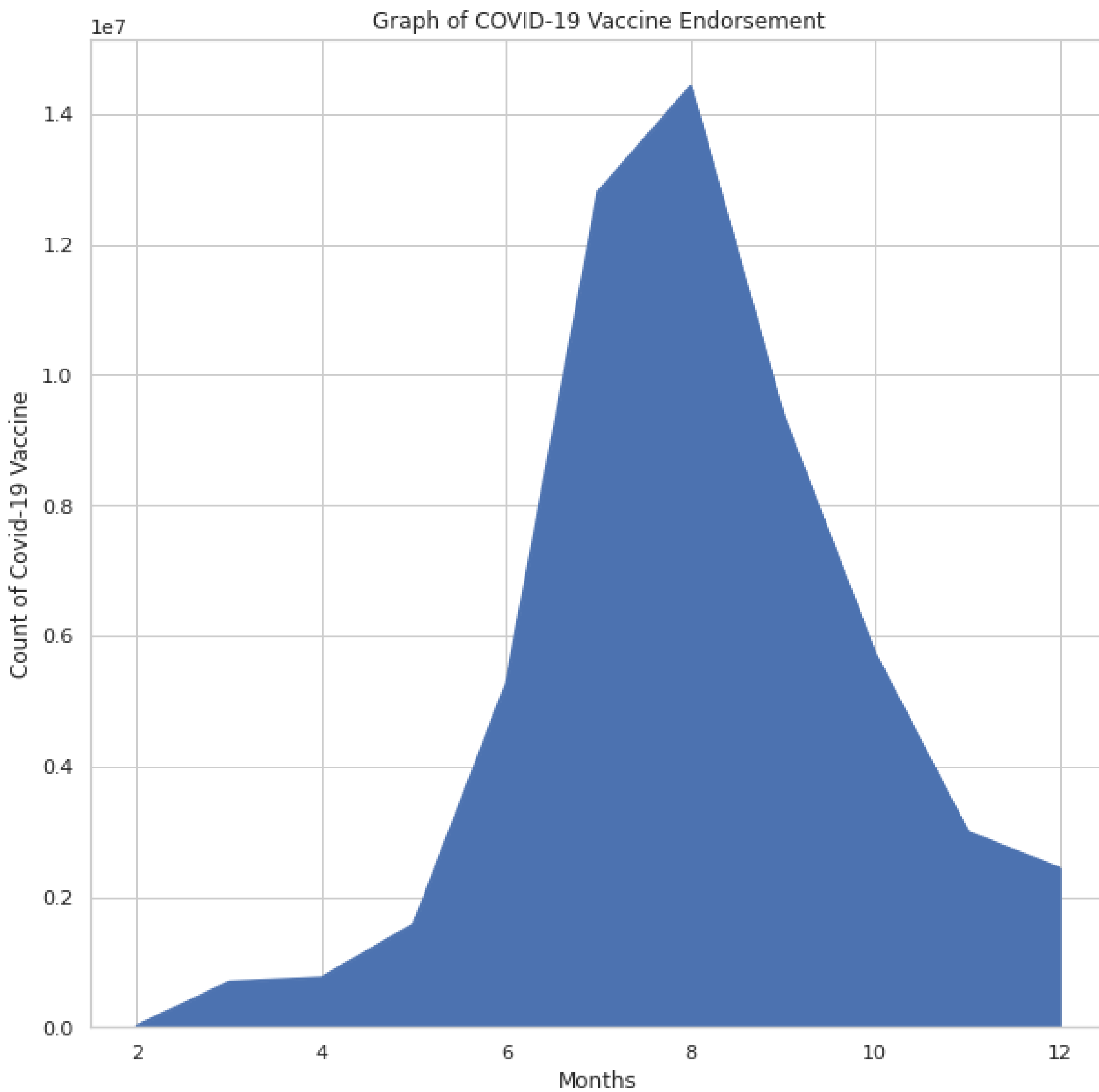
Out[41]:
```
Text(0.5, 0, 'Months')
```



Based on the are graph (Graph of COVID-19 Recovered Cases According to Months) above, we can found that there is an increase in COVID-19 recovered case from April to September, and then decrease back in December.

In [42]:
```
fig = df.groupby('month').daily_vax.sum().plot(kind='area', title='Graph of CO
VID-19 Vaccine Endorsement', figsize=(10,10))
fig.set_ylabel("Count of Covid-19 Vaccine")
fig.set_xlabel("Months")
```

Out[42]:
```
Text(0.5, 0, 'Months')
```



Based on the area graph (Graph of COVID-19 Vaccine Endorsement) above, we can found that there is an increase in daily COVID-19 Vaccine Endorsement from April to August, and then started to decrease back in December. This may be due to the reason that most of the people had taken their dose 1 and dose 2 vaccines.

Machine Learning

Linear Regresion

To build a supervised machine learning model based on the dataset to predict daily new COVID-19 cases, Linear Regression machine learning algorithm is used. Linear Regression is used to examine the relationship between one dependent variable and one or more independent variables and determine the strength of the predictor to predict the outcome. Therefore, in this section, **simple linear regression** and **multiple linear regression** will be used to examine the significance of the predictor. In order to do it, LinearRegression is imported from sklearn.linear_model library. To split the data set into train data set and testing set, train_test_split is imported from sklearn.model_selection. Finally, in order to evaluate the model performance, mean_squared_error and r2_score are imported from sklearn.metrics.

In [43]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
df
```

Out[43]:

	date	death_cases	new_cases	import_cases	recovered_cases	active_cases	unvax_cases
0	2021-02-24	12	3545	1	3331	30572	3545
1	2021-02-25	13	1924	6	3752	28738	1924
2	2021-02-26	10	2253	7	3085	27903	2253
3	2021-02-27	10	2364	1	3320	26937	2364
4	2021-02-28	9	2437	1	3251	26118	2437
...	...	...	...	...	...	...	...
295	2021-12-16	37	4262	36	4985	56156	955
296	2021-12-17	18	4362	28	5098	55380	969
297	2021-12-18	29	4083	34	5435	54000	1024
298	2021-12-19	19	3108	37	3701	53389	692
299	2021-12-20	43	2589	46	3810	52161	653

300 rows × 21 columns

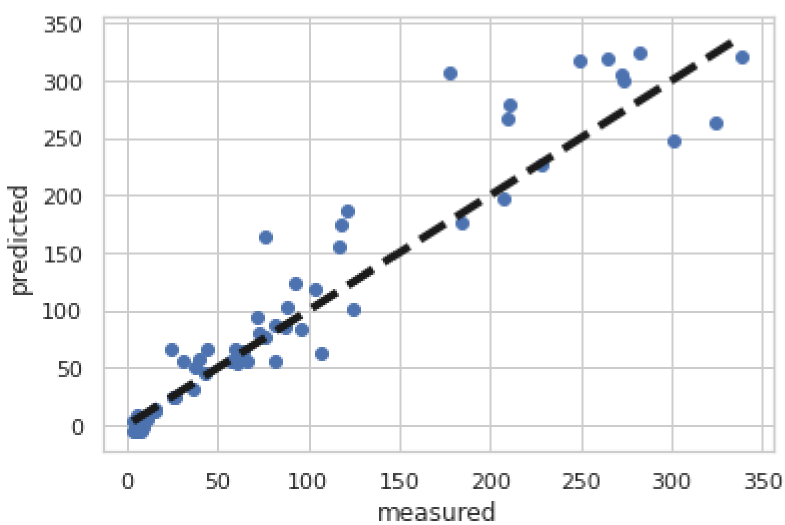
The active cases is declared as x variable which is an independent variable while death_cases, which is number of daily death cases is declared as y variable, a target variable. The dataset is split into 80% of training set and 20% of testing set. Then, fit it to the model reg. Finally, the model is evaluated using mean squared error and r2 score. A 87% of r2 score indicates 87% of data fits the model, means the model's performance is not bad.

In [44]:
```
x = df[['active_cases']]
y = df['death_cases']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
                                                    random_state=0)
# create linear regression object
reg = LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

y_predicted = reg.predict(X_test)

print("Mean squared error: %.2f" % mean_squared_error(y_test, y_predicted))
print('R²: %.2f' % r2_score(y_test, y_predicted))
```

```
Mean squared error: 1148.46
R²: 0.87
```

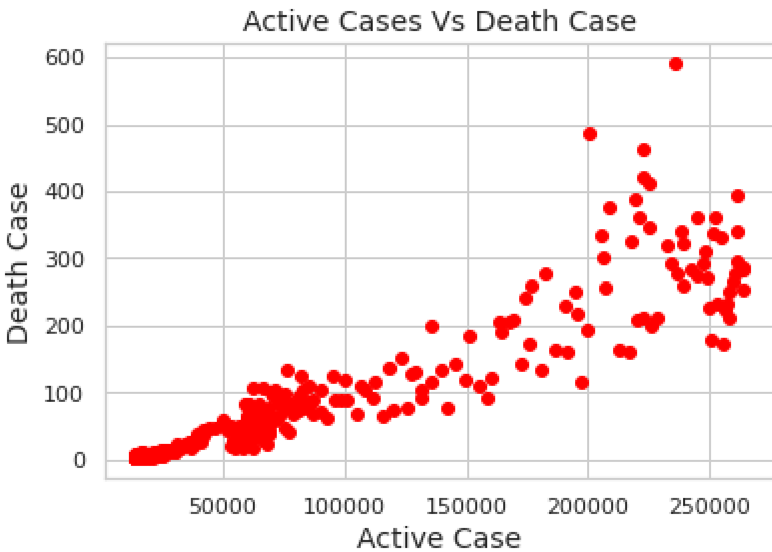
A scatter plot graph is plot to visualize the model performance.

In [45]:
```
fig, ax = plt.subplots()
ax.scatter(y_test, y_predicted)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=
4)
ax.set_xlabel('measured')
ax.set_ylabel('predicted')
plt.show()
```



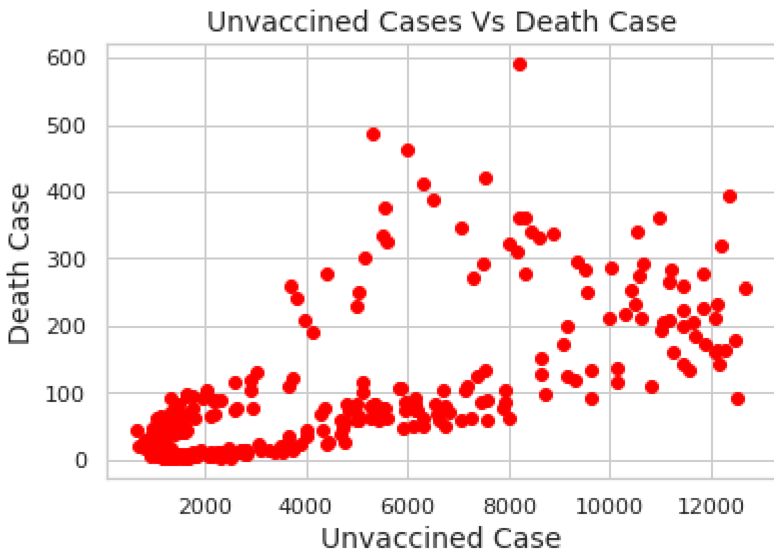
In order to find suitable independent variables to fit multiple linear regression model, a scatter plot of number of active cases against number of daily death cases are plotted to visualize their relationship.

In [46]:
```
plt.scatter(df['active_cases'], df['death_cases'], color='red')
plt.title('Active Cases Vs Death Case', fontsize=14)
plt.xlabel('Active Case', fontsize=14)
plt.ylabel('Death Case', fontsize=14)
plt.grid(True)
plt.show()
```



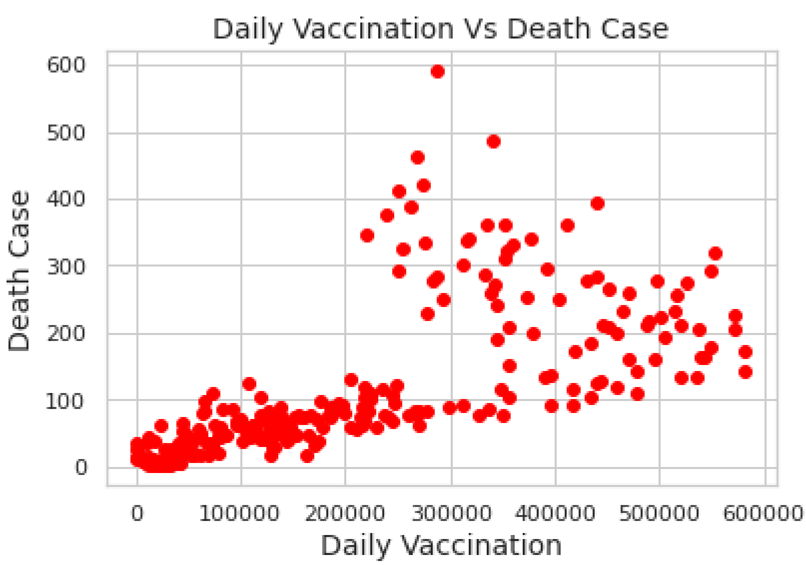
In order to find suitable independent variables to fit multiple linear regression model, a scatter plot of number of unvaccinated COVID-19 cases against number of daily death cases are plotted to visualize their relationship.

In [47]:
```
plt.scatter(df['unvax_cases'], df['death_cases'], color='red')
plt.title('Unvaccined Cases Vs Death Case', fontsize=14)
plt.xlabel('Unvaccined Case', fontsize=14)
plt.ylabel('Death Case', fontsize=14)
plt.grid(True)
plt.show()
```



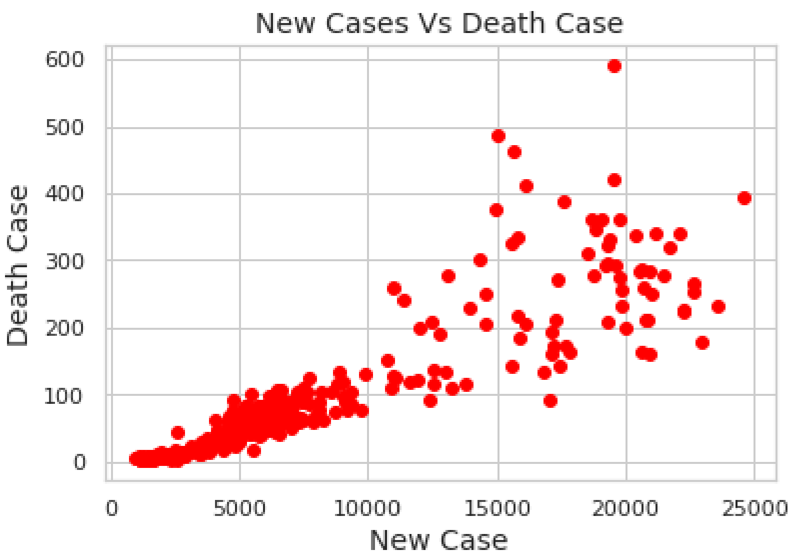
In order to find suitable independent variables to fit multiple linear regression model, a scatter plot of number of number of daily vaccinated taken against number of daily death cases are plotted to visualize their relationship.

In [48]:
```
plt.scatter(df['daily_vax'], df['death_cases'], color='red')
plt.title('Daily Vaccination Vs Death Case', fontsize=14)
plt.xlabel('Daily Vaccination', fontsize=14)
plt.ylabel('Death Case', fontsize=14)
plt.grid(True)
plt.show()
```



In order to find suitable independent variables to fit multiple linear regression model, a scatter plot of number of number of daily new COVID-19 cases against number of daily death cases are plotted to visualize their relationship.

In [49]:
```
plt.scatter(df['new_cases'], df['death_cases'], color='red')
plt.title('New Cases Vs Death Case', fontsize=14)
plt.xlabel('New Case', fontsize=14)
plt.ylabel('Death Case', fontsize=14)
plt.grid(True)
plt.show()
```



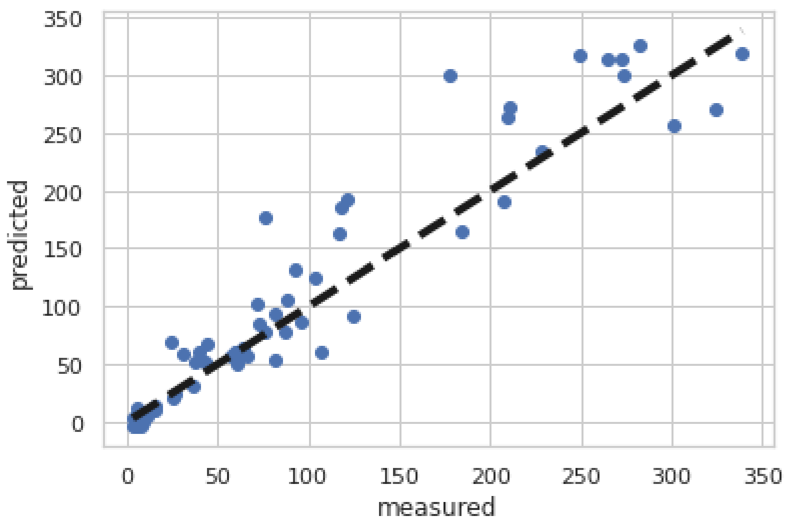
After plotting these scatter plot graphs, number of daily COVID-19 cases and number of active cases are selected as the independent variables for multiple Linear Regression model. A 86% of r2 score indicates 87% of data fits the model, means the model's performance is not bad. These two independent variables are possible to predict the daily death cases.

In [50]:
```
X = df.iloc[:,np.r_[2,5]]
Y = df.iloc[:,1]

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
                                                    random_state=0)
# create linear regression object
reg = LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

y_predicted = reg.predict(X_test)

print("Mean squared error: %.2f" % mean_squared_error(y_test, y_predicted))
print('R²: %.2f' % r2_score(y_test, y_predicted))
```

```
Mean squared error: 1226.95
R²: 0.86
```

A scatter plot graph is plot to visualize the model performance.

In [51]:
```
fig, ax = plt.subplots()
ax.scatter(y_test, y_predicted)
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=
4)
ax.set_xlabel('measured')
ax.set_ylabel('predicted')
plt.show()
```



Interactive Dashboard

In order to build interactive dashboard on Google Colab, plotly.express, display, HTML, interact, and widgets are imported. plotly.express library is used to plot the graph and chart while HTML is to display some numerical statistics. For interact imported from ipywidgets, it is used to perform interaction with the users. Then a data frame group by month is created for later use.

In [52]:
```
import plotly.express as px
from IPython.core.display import display, HTML
from ipywidgets import interact
import ipywidgets as widgets
df_month = df.groupby(['month']).sum()
df_month['month'] = df_month.index
df_month.reset_index(drop=True, inplace=True)

sorted_month_df = df_month.sort_values('new_cases', ascending= False)
```

bubble_chart() function is created and used to plot a scatter plot graph of month against monthly new cases. The user is able to select highest new cases month ranged from 0 to 11 which implemented using widgets.IntSlider(). The user able to select to highest new cases month by using the slider.

In [57]:
```
def bubble_chart(n):
  fig = px.scatter(sorted_month_df.head(n), x='month',y='new_cases', size='new
_cases',
                  color='month', hover_name='month',size_max=60)

  fig.update_layout(
    title=str(n) +" Highest New Cases Month",
    xaxis_title="Month",
    yaxis_title="Monthly New Cases",
    width = 700
    )
  fig.show();

interact(bubble_chart, n=widgets.IntSlider(min=0, max=11, step=1, value=5));
ipywLayout.display='none'
```

A function named stats_of_month is created and used to display number of news cases per month, recovered cases per month, death cases per month and vaccination took per month. The function also plot a bar chart of every day of the selected month against death cases. The user is able to interact by typing in the number which represents the month in the text box.

In [58]:
```
def stats_of_month(Month):
  n = int(Month)
  death_total = df_month.loc[df_month['month'] == n , 'death_cases'].sum()
  vac_total = int(df_month.loc[df_month['month'] == n , 'daily_vax'].sum())
  recovered_total = int(df_month.loc[df_month['month'] == n , 'recovered_case
s'].sum())
  new_cases = int(df_month.loc[df_month['month'] == n , 'new_cases'].sum())

  display(HTML("<div style = 'background-color: #504e4e; padding: 30px '>" +
              "<span style='color: red; font-size:30px;margin-left:20px;'> New
Cases: "  + str(new_cases) +"</span>" +
                "<span style='color: lightgreen; font-size:30px; margin-left:2
0px;'> Recovered Cases: " + str(recovered_total) + "</span>"+
             "<span style='color: red; font-size:30px;margin-left:20px;'> Deat
h Cases: " + str(death_total) + "</span>"+
              "<span style='color: #fff; font-size:30px;'> Vaccination took: "
+ str(vac_total) +"</span>" +
             "</div>")
       )
  df_case = df.loc[df['month']== n]
  return px.bar(
      df_case,
      x='date',
      y='death_cases',
      title= "Daily death case in month "+ str(n), # the axis names
      color_discrete_sequence=["blueviolet"],
      height=500,
      width=800
  )

interact(active_case_death_case, Month='8')
```

Out[58]: <function __main__.active_case_death_case>

A function named vaccination_type_bar_chart is created and used to display number of the selected vaccination type injected per month. The user can select the type of the vaccine in the drop down menu and number of selected vaccination type injected per month will be shown.

In [56]:
```
from ipywidgets.widgets import widget
def vaccination_type_bar_chart(vaccination_type):
  fig = px.bar(df, x='month', y=vaccination_type ,
                  color='month', hover_name='month')

  fig.update_layout(
    title="Endorsement of "+ vaccination_type +" Vaccine based on Month",
    xaxis_title="Month",
    yaxis_title="Count",
    width = 700
    )
  fig.show();

interact(vaccination_type_bar_chart,vaccination_type=['sinovac','pfizer','astr
a','sinopharm','tot_cansino' ]);
ipywLayout = widgets.Layout(border='solid 2px blue')
ipywLayout.display='none'
```

In [59]:
```
!pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-pac
kages (5.6.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python
3.7/dist-packages (from nbconvert) (1.5.0)
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.7/dist-p
ackages (from nbconvert) (2.11.3)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.
7/dist-packages (from nbconvert) (0.3)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-pack
ages (from nbconvert) (0.5.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dis
t-packages (from nbconvert) (5.1.1)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7/
dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-
packages (from nbconvert) (4.9.1)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-pa
ckages (from nbconvert) (0.7.1)
Requirement already satisfied: nbformat>=4.4 in /usr/local/lib/python3.7/dist
-packages (from nbconvert) (5.1.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packag
es (from nbconvert) (4.1.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-pack
ages (from nbconvert) (2.6.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/d
ist-packages (from jinja2>=2.4->nbconvert) (2.0.1)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/local/lib/pyth
on3.7/dist-packages (from nbformat>=4.4->nbconvert) (2.6.0)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.7/d
ist-packages (from nbformat>=4.4->nbconvert) (0.2.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-pa
ckages (from bleach->nbconvert) (1.15.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-pac
kages (from bleach->nbconvert) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/pyt
hon3.7/dist-packages (from packaging->bleach->nbconvert) (3.0.6)
```

In [ ]:
```

```

\n \n \n
\n \n \n\n"}, "metadata": {}}], "msg_id": "", "_model_module_version": "1.0.0", "_view_count": null, "_view_module_version": "1.0.0", "layout": "IPY_MODEL_fdc4e2f736e94be9b4d0f50c5ad3ca23", "_model_module": "@jupyter-widgets/output"}, "model_name": "OutputModel", "model_module_version": "1.0.0"}, "fdc4e2f736e94be9b4d0f50c5ad3ca23": {"model_module": "@jupyter-widgets/base", "state": {"_view_name": "LayoutView", "grid_template_rows": null, "right": null, "justify_content": null, "_view_module": "@jupyter-widgets/base", "_model_module_version": "1.2.0", "_view_count": null, "flex_flow": null, "min_width": null, "border": null, "align_items": null, "bottom": null, "_model_module": "@jupyter-widgets/base", "top": null, "grid_column": null, "width": null, "grid_row": null, "grid_auto_flow": null, "grid_area": null, "grid_template_columns": null, "flex": null, "_model_name": "LayoutModel", "justify_items": null, "overflow_x": null, "max_height": null, "align_content": null, "visibility": null, "overflow": null, "height": null, "min_height": null, "padding": null, "grid_auto_rows": null, "grid_gap": null, "overflow_y": null, "max_width": null, "display": null, "_view_module_version": "1.2.0", "align_self": null, "grid_template_areas": null, "object_position": null, "object_fit": null, "grid_auto_columns": null, "margin": null, "order": null, "left": null}, "model_name":

"LayoutModel", "model_module_version": "1.2.0"}, "c72382e4e695474aa25961917d65a5cb": {"model_module": "@jupyter-widgets/controls", "state": {"_view_name": "VBoxView", "_dom_classes": ["widget-interact"], "_model_name": "VBoxModel", "_view_module": "@jupyter-widgets/controls", "_model_module_version": "1.5.0", "_view_count": null, "_view_module_version": "1.5.0", "box_style": "", "layout": "IPY_MODEL_0bdedb0f70384606930551d85da69e94", "_model_module": "@jupyter-widgets/controls", "children": ["IPY_MODEL_6286653f2f0347ccbe14b2d04897868f", "IPY_MODEL_fe78846410c249608ab00df12266b398"]}, "model_name": "VBoxModel", "model_module_version": "1.5.0"}, "33facbda16474873a299d6ec85bf7bd3": {"model_module": "@jupyter-widgets/base", "state": {"_view_name": "LayoutView", "grid_template_rows": null, "right": null, "justify_content": null, "_view_module": "@jupyter-widgets/base", "_model_module_version": "1.2.0", "_view_count": null, "flex_flow": null, "min_width": null, "border": null, "align_items": null, "bottom": null, "_model_module": "@jupyter-widgets/base", "top": null, "grid_column": null, "width": null, "grid_row": null, "grid_auto_flow": null, "grid_area": null, "grid_template_columns": null, "flex": null, "_model_name": "LayoutModel", "justify_items": null, "overflow_x": null, "max_height": null, "align_content": null, "visibility": null, "overflow": null, "height": null, "min_height": null, "padding": null, "grid_auto_rows": null, "grid_gap": null, "overflow_y": null, "max_width": null, "display": null, "_view_module_version": "1.2.0", "align_self": null, "grid_template_areas": null, "object_position": null, "object_fit": null, "grid_auto_columns": null, "margin": null, "order": null, "left": null}, "model_name": "LayoutModel", "model_module_version": "1.2.0"}, "6286653f2f0347ccbe14b2d04897868f": {"model_module": "@jupyter-widgets/controls", "state": {"_view_name": "TextView", "style": "IPY_MODEL_17f69fff72d845ca88bd18af90663f0f", "_dom_classes": [], "description": "Month", "_model_name": "TextModel", "placeholder": "\u200b", "_view_module": "@jupyter-widgets/controls", "_model_module_version": "1.5.0", "value": "8", "_view_count": null, "disabled": false, "_view_module_version": "1.5.0", "continuous_update": true, "description_tooltip": null, "_model_module": "@jupyter-widgets/controls", "layout": "IPY_MODEL_6a8cdb9c23c24057a4292b7b96445f5e"}, "model_name": "TextModel", "model_module_version": "1.5.0"}, "7fa8f12e3a1647339474d5dbcca2363e": {"model_module": "@jupyter-widgets/controls", "state": {"_view_name": "StyleView", "_model_name": "DescriptionStyleModel", "description_width": "", "_view_module": "@jupyter-widgets/base", "_model_module_version": "1.5.0", "_view_count": null, "_view_module_version": "1.2.0", "_model_module": "@jupyter-widgets/controls"}, "model_name": "DescriptionStyleModel", "model_module_version": "1.5.0"}, "15f2fe96c111480f9b9b81bdd0bb0aad": {"model_module": "@jupyter-widgets/output", "state": {"_view_name": "OutputView", "_view_module": "@jupyter-widgets/output", "_dom_classes": [], "_model_name": "OutputModel", "outputs": [{"output_type": "display_data", "data": {"text/html": "\n\n\n
\n \n \n \n
\n \n
\n \n
\n\n"}, "metadata": {}}], "msg_id": "", "_model_module_version": "1.0.0", "_view_count": null, "_view_module_version": "1.0.0", "layout": "IPY_MODEL_8cf937a8a93c463c9518f8aebe66fad5", "_model_module": "@jupyter-widgets/output"}, "model_name": "OutputModel", "model_module_version": "1.0.0"}, "6a8cdb9c23c24057a4292b7b96445f5e": {"model_module": "@jupyter-widgets/base", "state": {"_view_name": "LayoutView", "grid_template_rows": null, "right": null, "justify_content": null, "_view_module": "@jupyter-widgets/base", "_model_module_version": "1.2.0", "_view_count": null, "flex_flow": null, "min_width": null, "border": null, "align_items": null, "bottom": null, "_model_module": "@jupyter-widgets/base", "top": null, "grid_column": null, "width": null, "grid_row": null, "grid_auto_flow": null, "grid_area": null, "grid_template_columns": null, "flex": null, "_model_name": "LayoutModel", "justify_items": null, "overflow_x": null, "max_height": null, "align_content": null, "visibility": null, "overflow": null, "height": null, "min_height": null, "padding": null, "grid_auto_rows": null, "grid_gap": null, "overflow_y": null, "max_width": null, "display": null, "_view_module_version": "1.2.0", "align_self": null, "grid_template_areas": null, "object_position": null, "object_fit": null, "grid_auto_columns": null, "margin": null, "order": null, "left": null}, "model_name": "LayoutModel", "model_module_version": "1.2.0"}, "17f69fff72d845ca88bd18af90663f0f": {"model_module": "@jupyter-widgets/controls", "state": {"_view_name": "StyleView", "_model_name": "DescriptionStyleModel", "description_width": "", "_view_module": "@jupyter-widgets/base", "_model_module_version": "1.5.0", "_view_count": null, "_view_module_version": "1.2.0", "_model_module": "@jupyter-widgets/controls"}, "model_name": "DescriptionStyleModel", "model_module_version": "1.5.0"}, "33700f14b35342f28a61f58cc874df88": {"model_module": "@jupyter-widgets/controls", "state": {"_view_name": "VBoxView", "_dom_classes": ["widget-interact"], "_model_name": "VBoxModel", "_view_module": "@jupyter-widgets/controls", "_model_module_version": "1.5.0", "_view_count": null, "_view_module_version": "1.5.0", "box_style": "", "layout": "IPY_MODEL_33facbda16474873a299d6ec85bf7bd3", "_model_module": "@jupyter-widgets/controls", "children": ["IPY_MODEL_ea035895f96045798a5dcadd2594eb5e", "IPY_MODEL_15f2fe96c111480f9b9b81bdd0bb0aad"]}, "model_name": "VBoxModel", "model_module_version": "1.5.0"}, "0bdedb0f70384606930551d85da69e94": {"model_module":

"@jupyter-widgets/base", "state": {"_view_name": "LayoutView", "grid_template_rows": null, "right": null, "justify_content": null, "_view_module": "@jupyter-widgets/base", "_model_module_version": "1.2.0", "_view_count": null, "flex_flow": null, "min_width": null, "border": null, "align_items": null, "bottom": null, "_model_module": "@jupyter-widgets/base", "top": null, "grid_column": null, "width": null, "grid_row": null, "grid_auto_flow": null, "grid_area": null, "grid_template_columns": null, "flex": null, "_model_name": "LayoutModel", "justify_items": null, "overflow_x": null, "max_height": null, "align_content": null, "visibility": null, "overflow": null, "height": null, "min_height": null, "padding": null, "grid_auto_rows": null, "grid_gap": null, "overflow_y": null, "max_width": null, "display": null, "_view_module_version": "1.2.0", "align_self": null, "grid_template_areas": null, "object_position": null, "object_fit": null, "grid_auto_columns": null, "margin": null, "order": null, "left": null}, "model_name": "LayoutModel", "model_module_version": "1.2.0"}, "cc78e0508c0f42a0a2a69ae65fc34c05": {"model_module": "@jupyter-widgets/controls", "state": {"_options_labels": ["sinovac", "pfizer", "astra", "sinopharm", "tot_cansino"], "_view_name": "DropdownView", "style": "IPY_MODEL_7fa8f12e3a1647339474d5dbcca2363e", "_dom_classes": [], "description": "vaccination_type", "_model_name": "DropdownModel", "index": 3, "_view_module": "@jupyter-widgets/controls", "_model_module_version": "1.5.0", "_view_count": null, "disabled": false, "_view_module_version": "1.5.0", "description_tooltip": null, "_model_module": "@jupyter-widgets/controls", "layout": "IPY_MODEL_915bbcc4605840f7bee3926ce8147ef2"}, "model_name": "DropdownModel", "model_module_version": "1.5.0"}, "1c091b9ad82746ca8933af4b7afdf1ef": {"model_module": "@jupyter-widgets/output", "state": {"_view_name": "OutputView", "_view_module": "@jupyter-widgets/output", "_dom_classes": [], "_model_name": "OutputModel", "outputs": [{"output_type": "display_data", "data": {"text/html": "\n\n\n
\n \n \n \n
\n \n
\n\n"}, "metadata": {}}], "msg_id": "", "_model_module_version": "1.0.0", "_view_count": null, "_view_module_version": "1.0.0", "layout": "IPY_MODEL_35b3712039b5491cb5f1605db5eb61fd", "_model_module": "@jupyter-widgets/output"}, "model_name": "OutputModel", "model_module_version": "1.0.0"}}