In [1]:

```
{
  "authors": [
    {
      "name": "Алексеев Андрей Сергеевич"
    }
  ],
  "group": "ИУ5-62Б",
  "kernelspec": {
    "name": "python3",
    "display_name": "Python 3 (ipykernel)",
    "language": "python"
  },
  "language_info": {
    "name": "python",
    "version": "3.9.7",
    "mimetype": "text/x-python",
    "codemirror_mode": {
      "name": "ipython",
      "version": 3
    },
    "pygments_lexer": "ipython3",
    "nbconvert_exporter": "python",
    "file_extension": ".py"
  },
  "title": "Анализ и прогнозирование временного ряда"
}
```

Out[1]:

```
{'authors': [{'name': 'Алексеев Андрей Сергеевич'}],
 'group': 'ИУ5-62Б',
 'kernelspec': {'name': 'python3',
  'display_name': 'Python 3 (ipykernel)',
  'language': 'python'},
 'language_info': {'name': 'python',
  'version': '3.9.7',
  'mimetype': 'text/x-python',
  'codemirror_mode': {'name': 'ipython', 'version': 3},
  'pygments_lexer': 'ipython3',
  'nbconvert_exporter': 'python',
  'file_extension': '.py'},
 'title': 'Анализ и прогнозирование временного ряда'}
```

In [2]:

```
import sys
sys.path.append(r"C:\Users\Admин")
sys.path.append(r"c:\users\admин\appdata\local\packages\pythonsoftwarefo
import sys
import warnings
warnings.filterwarnings('ignore')
from tqdm import tqdm

import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error

import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
from scipy.optimize import minimize

import matplotlib.pyplot as plt
```

In [3]:

```
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
from plotly import graph_objs as go
init_notebook_mode(connected = True)

def plotly_df(df, title = ''):
    data = []

    for column in df.columns:
        trace = go.Scatter(
            x = df.index,
            y = df[column],
            mode = 'lines',
            name = column
        )
        data.append(trace)

    layout = dict(title = title)
    fig = dict(data = data, layout = layout)
    iplot(fig, show_link=False)

dataset = pd.read_csv('StatewiseTestingDetails.csv', index_col=['Date'],
dataset = dataset.drop(columns=['State','Negative','Positive'])
plotly_df(dataset, title = "COVID-19 in India")
```

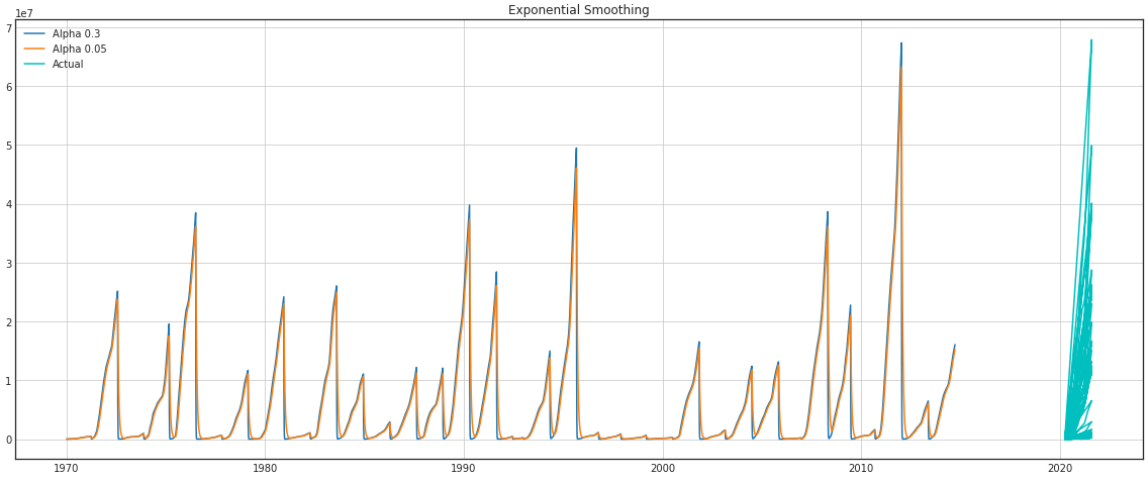
In [4]:

```
def exponential_smoothing(series, alpha):
```

```
    result = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        result.append(alpha * series[n] + (1 - alpha) * result[n-1])
    return result
```

In [5]:

```
with plt.style.context('seaborn-white'):
    plt.figure(figsize=(20, 8))
    for alpha in [0.3, 0.05]:
        plt.plot(exponential_smoothing(dataset['TotalSamples'], alpha),
    plt.plot(dataset['TotalSamples'], "c", label = "Actual")
    plt.legend(loc="best")
    plt.axis('tight')
    plt.title("Exponential Smoothing")
    plt.grid(True)
```



In [6]:

```
def double_exponential_smoothing(series, alpha, beta):
    result = [series[0]]
    for n in range(1, len(series)+1):
        if n == 1:
            level, trend = series[0], series[1] - series[0]
        if n >= len(series): # прогнозируем
            value = result[-1]
        else:
            value = series[n]
        last_level, level = level, alpha*value + (1-alpha)*(level+trend)
        trend = beta*(level-last_level) + (1-beta)*trend
        result.append(level+trend)
    return result
```

In [7]:

```
with plt.style.context('seaborn-white'):
    plt.figure(figsize=(20, 8))
    for alpha in [0.9, 0.02]:
        for beta in [0.9, 0.02]:
            plt.plot(double_exponential_smoothing(dataset['TotalSamples'
    plt.plot(dataset['TotalSamples'], label = "Actual")
    plt.legend(loc="best")
    plt.axis('tight')
    plt.title("Double Exponential Smoothing")
    plt.grid(True)
```

In [8]:

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn import cluster, datasets, mixture
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.metrics import homogeneity_completeness_v_measure
from sklearn.metrics import silhouette_score
from itertools import cycle, islice
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="ticks")
np.random.seed(100)
%matplotlib inline
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

In [9]:

```
cluster_n_samples = 1500

datasets_names = ['circles', 'moons', 'blobs', 'aniso', 'varied', 'no_st

def generate_datasets(n_samples):
    """
    Генерация набора данных на 1500 точек
    """
    noisy_circles, noisy_circles_y = datasets.make_circles(
        n_samples=n_samples, factor=.5, noise=.05)
    noisy_moons, noisy_moons_y = datasets.make_moons(n_samples=n_samples
    blobs, blobs_y = datasets.make_blobs(n_samples=n_samples, random_sta
    no_structure = np.random.rand(n_samples, 2)

    X_aniso, y_aniso = datasets.make_blobs(n_samples=n_samples, random_s
    transformation = [[0.6, -0.6], [-0.4, 0.8]]
    aniso = np.dot(X_aniso, transformation)

    varied, varied_y = datasets.make_blobs(
        n_samples=n_samples,
        cluster_std=[1.0, 2.5, 0.5],
        random_state=170)
```

```
    result_y = [noisy_circles_y, noisy_moons_y, varied_y, y_aniso, blobs_

    result_not_scaled = [noisy_circles, noisy_moons,
                         varied, aniso, blobs, no_structure]

    # Нормализуем признаки
    result = []
    for data in result_not_scaled:
        data_res = StandardScaler().fit_transform(data)
        result.append(data_res)

    return result, result_y
```

In [10]:
```
cluster_datasets, cluster_true_y = generate_datasets(cluster_n_samples)
```

In [11]:
```
# Сгенерировано 6 наборов данных
len(cluster_datasets)
```

Out[11]:
```
6
```

In [12]:
```
cluster_datasets[0].shape
```

Out[12]:
```
(1500, 2)
```

In [13]:
```
cluster_datasets[0]
```

Out[13]:
```
array([[-1.03872991, -0.19723691],
       [-0.78146692,  0.17887893],
       [-0.79616984, -0.22695504],
       ...,
       [-1.83374923,  0.31844795],
       [ 1.40784551,  1.12646296],
       [ 0.94318876,  0.35308948]])
```

In [14]:
```
len(cluster_true_y)
```

Out[14]:
```
5
```

In [15]:
```
cluster_true_y[0].shape
```

Out[15]:
```
(1500,)
```

In [16]:
```
cluster_true_y[0]
```

Out[16]:
```
array([1, 1, 1, ..., 0, 0, 1], dtype=int64)
```

In [17]:
```
for cluster in cluster_true_y:
    print(np.unique(cluster))
```

```
[0 1]
[0 1]
```

```
[0 1 2]
[0 1 2]
[0 1 2]
```

In [18]:

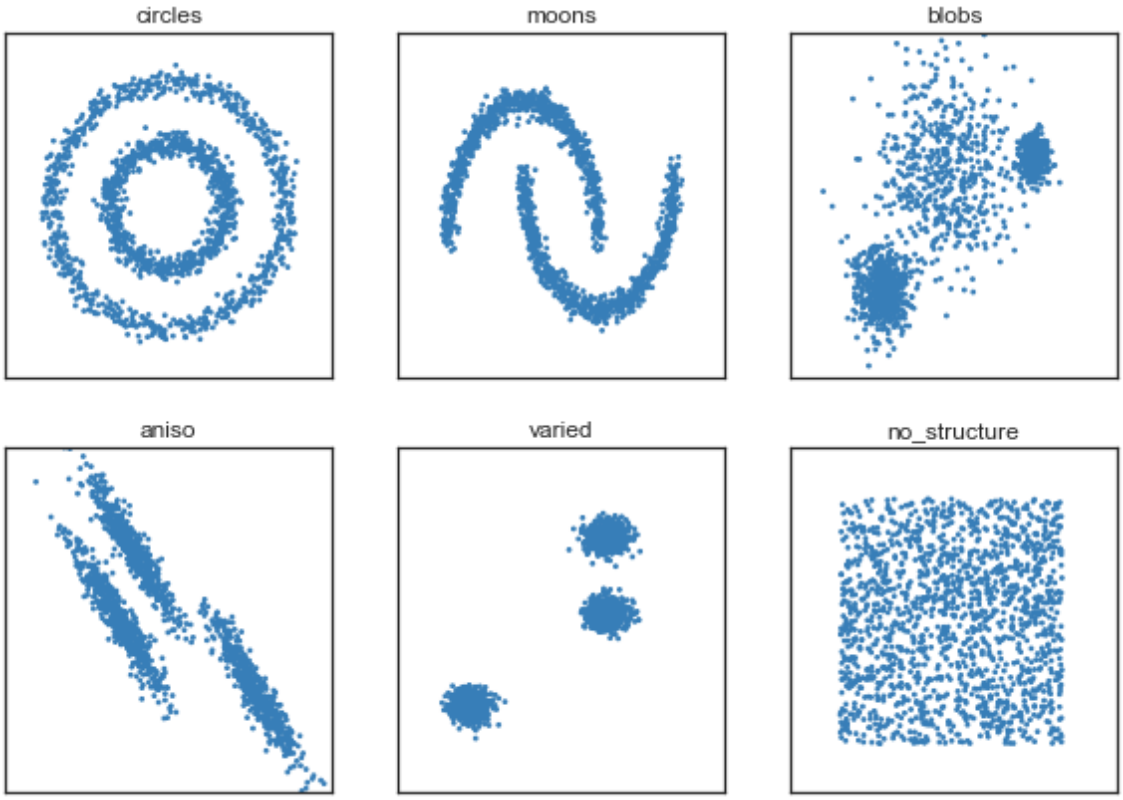
```
def visualize_clusters(cluster_datasets, cluster_results):
    """
    Визуализация результатов кластерного анализа
    """
    plt.subplots(figsize=(10,7))
    plot_num = 0
    for X, y_pred in zip(cluster_datasets, cluster_results):
        plot_num += 1
        plt.subplot(2, 3, plot_num)
        # Цвета точек как результат кластеризации
        colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4da
                                             '#f781bf', '#a65628', '#984
                                             '#999999', '#e41a1c', '#ded
                                      int(max(y_pred) + 1))))
        # черный цвет для выделяющихся значений
        colors = np.append(colors, ["#000000"])
        plt.scatter(X[:, 0], X[:, 1], s=3, color=colors[y_pred])
        plt.xlim(-2.5, 2.5)
        plt.ylim(-2.5, 2.5)
        plt.xticks(())
        plt.yticks(())
        plt.title(datasets_names[plot_num-1])

    plt.show()
```

In [19]:

```
cluster_results_empty = []
for i in range(6):
    cluster_results_empty.append(np.zeros(cluster_n_samples, dtype=int))
```

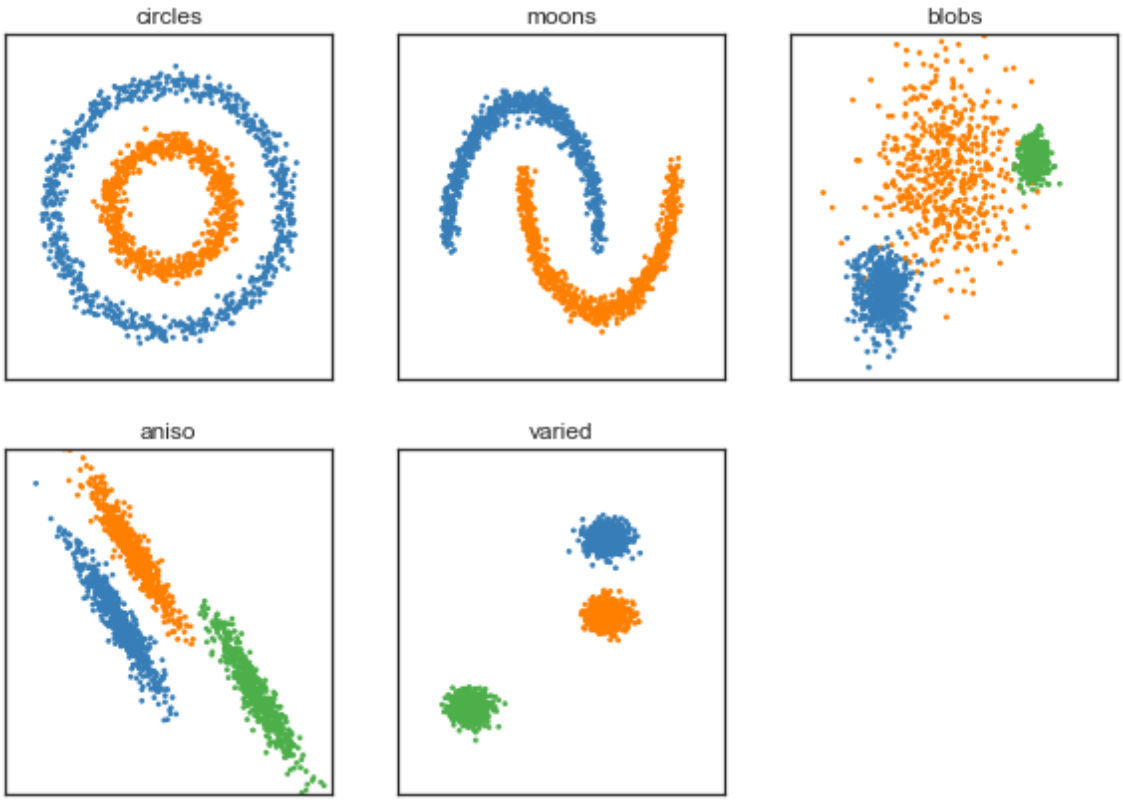
In [20]:

```
# Нет кластеров
visualize_clusters(cluster_datasets, cluster_results_empty)
```



In [21]:

```
# Эталонные значения кластеров
visualize_clusters(cluster_datasets, cluster_true_y)
```



In [22]:

```
def do_clustering(cluster_datasets, method):
    """
    Выполнение кластеризации для данных примера
    """
    cluster_results = []
```

```
    for X in cluster_datasets:
        temp_cluster = method.fit_predict(X)
        cluster_results.append(temp_cluster)
    return cluster_results
```

In [23]:

```
from sklearn.cluster import KMeans, MiniBatchKMeans
```

In [25]:

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_
from colab_pdf import colab_pdf
colab_pdf('pandas-assignment.ipynb')
```

```
---------------------------------------------------------------------------
--
ModuleNotFoundError                       Traceback (most recent call las
t)
C:\Users\7272~1\AppData\Local\Temp/ipykernel_5004/4081803837.py in <modul
e>
      1 get_ipython().system('wget -nc https://raw.githubusercontent.com/
brpy/colab-pdf/master/colab_pdf.py')
----> 2 from colab_pdf import colab_pdf
      3 colab_pdf('pandas-assignment.ipynb')

ModuleNotFoundError: No module named 'colab_pdf'
```

In [ ]: