PCA-on-Financial-Market-in-2020

May 30, 2021

1 Motivation

In 2020, COVID-19 has massively destroyed almost every portion of the globe. It is often daunting to believe that we could generalize price fluctuation of global market through simple regression, or a function of single x and single y. However, as Coronavirus has been the dominating disaster that has led to the scenes that are unprecedent, it may be possible to make very simple model that explains price fluctuation. The project entirely commits to understand if very simple model with a single input variable could explain the movement of global financial market.

2 Literature Review

While the main reason why I conduct this experiment is to understand how much COVID-19 could explain the variance of financial market, I quote a paper of Kritzman et al. (2010), 'Principal Components as a Measure of Systemic Risk' [1]. It indicates that Principal Component Analysis could extract and measure systemic risk, which is a ratio of systematic risk to idiosyncratic risk, because principal components, simplified eigenvectors, are the combinations of correlated inputs. As Principal Components Analysis does simplify dimensions of variables by taking orthogonal transformation of larger-dimensional vectors, it preserves almost entire original attributes of data while simplifing original data.

> "the absorption ratio, which equals the fraction of the total variance of a set of asset returns explained or "absorbed" by a fixed number of eigenvectors. The absorption ratio captures the extent to which markets are unified or tightly coupled. When markets are tightly coupled, they are more fragile in the sense that negative shocks propagate more quickly and broadly than when markets are loosely linked."

While it is the argument of Kritzman et al. (2010), it is my intention that I take principal components from data 'COVID-19 Confirmed' and 'Daily Return' so that we observe how much variance is explained by first principal component. I will extend the principal component into five, so that I can check the trend of the degree of explained variance in detail.

3 Null Hypothesis (H_0)

My null hypothesis is that countries whose COVID-19 fluctuation is stronger has stronger market fluctuation, and the direction of it is identical.

Let change in confirmed covid-19 as C and change in Price Fluctuation as P,

$$H_0 : \forall \left| C - P \right| < 0$$

In other words, change in COVID-19 and that in Daily Return are in same direction, or more intuitively, I just would like to check if increase in COVID-19 confirmation is paird with worst price fluctuation, and vise versa.

4 Principal Component Analysis

4.1 Load Packages and Define Functions

- Load Libraries
- Functions `z_score` and `date_convert` as something *pre-defined*.

```
In [1]: # Load Packages

        import yfinance as yf
        import csv
        import pandas as pd
        import numpy as np
        from datetime import date, time, timedelta
        import datetime
        from countryinfo import CountryInfo
        from datetime import datetime, timedelta
        from sklearn.decomposition import PCA
        from datapackage import Package
        from sklearn.cluster import KMeans

        today = datetime.today()
        yesterday = str(today - timedelta(2))[:10]

        # apply the z-score method in Pandas using the .mean() and .std() methods
        def z_score(df):
            # copy the dataframe
            df_std = df.copy()
            # apply the z-score method
            for column in df_std.columns:
                df_std[column] = (df_std[column] - df_std[column].mean()) / df_std[column].std

            return df_std

        # Convert Date
        def date_convert(dates):
            dates_return = []

            for date in dates:
                date = date.split("/")
                year = '20' + str(date[2])
```

```
            month = str(date[0])
            day = str(date[1])

            if int(month) < 10:
                month = '0' + month

            if int(day) < 10:
                day = '0' + day

            date = year + "-" + month + "-" + day
            dates_return.append(date)

        return dates_return

    start = "2020-01-22"
    end = "2021-01-01"

    # Matplotlib
    import matplotlib.pyplot as plt
    pd.set_option('max_rows', 500)
    pd.set_option('max_columns', 500)
    np.set_printoptions(suppress=True)

    %matplotlib inline
    plt.rcParams["figure.figsize"] = (16, 12)
    plt.style.use('seaborn-pastel')
    plt.rcParams['lines.linewidth'] = 1
    plt.figure(dpi=300)
    plt.rcParams['lines.color'] = 'b'
    plt.rcParams['axes.grid'] = True
    plt.tight_layout()
```

```
<Figure size 4800x3600 with 0 Axes>
```

4.2 Data Import

Import Data

4.2.1 Stock Indices

This part of code imports **39 Market Indices** that represent major financial market in the globe.

```
In [2]: # Import Market Indices
        SPY = yf.download("SPY", start, end)['Adj Close'].to_frame()
        Singapore = yf.download("^STI", start, end)['Adj Close'].to_frame()
        Dow = yf.download("^DJI", start, end)['Adj Close'].to_frame()
        Nasdaq = yf.download("^IXIC", start, end)['Adj Close'].to_frame()
```

```
FTSE100 = yf.download("^FTSE", start, end)['Adj Close'].to_frame()
FTSE250 = yf.download("^FTSE", start, end)['Adj Close'].to_frame()
FTSE350 = yf.download("^FTLC", start, end)['Adj Close'].to_frame()
FTAI = yf.download("^FTAI", start, end)['Adj Close'].to_frame()
N225 = yf.download("^N225", start, end)['Adj Close'].to_frame()
N500 = yf.download("^N500", start, end)['Adj Close'].to_frame()
N1000 = yf.download("^N1000", start, end)['Adj Close'].to_frame()
HSI = yf.download("^HSI", start, end)['Adj Close'].to_frame()
Taiwan = yf.download("^TWII", start, end)['Adj Close'].to_frame()
SSE = yf.download("000001.SS", start, end)['Adj Close'].to_frame()
Shenzhen = yf.download("399001.SZ", start, end)['Adj Close'].to_frame()
DAX = yf.download("^GDAXI", start, end)['Adj Close'].to_frame()
France = yf.download("^FCHI", start, end)['Adj Close'].to_frame()
Indonesia = yf.download("^JKSE", start, end)['Adj Close'].to_frame()
PSEI = yf.download("PSEI.PS", start, end)['Adj Close'].to_frame()
AORD = yf.download("^AORD", start, end)['Adj Close'].to_frame()
AXJO = yf.download("^AXJO", start, end)['Adj Close'].to_frame()
AXKO = yf.download("^AXKO", start, end)['Adj Close'].to_frame()
kospi = yf.download("^KS11", start, end)['Adj Close'].to_frame()
India = yf.download("^BSESN", start, end)['Adj Close'].to_frame()
NZ50 = yf.download("^NZ50", start, end)['Adj Close'].to_frame()
XAX = yf.download("^XAX", start, end)['Adj Close'].to_frame()
RUI = yf.download("^RUI", start, end)['Adj Close'].to_frame()
RUT = yf.download("^RUT", start, end)['Adj Close'].to_frame()
RUA = yf.download("^RUA", start, end)['Adj Close'].to_frame()
GSPTSE = yf.download("^GSPTSE", start, end)['Adj Close'].to_frame()
N100 = yf.download("^N100", start, end)['Adj Close'].to_frame()
N150 = yf.download("^N150", start, end)['Adj Close'].to_frame()
BFX = yf.download("^BFX", start, end)['Adj Close'].to_frame()
IMOEX = yf.download("IMOEX.ME", start, end)['Adj Close'].to_frame()
MERV = yf.download("^MERV", start, end)['Adj Close'].to_frame()
TA125 = yf.download("^TA125.TA", start, end)['Adj Close'].to_frame()
JNOU = yf.download("^JNOU.JO", start, end)['Adj Close'].to_frame()
AEX = yf.download("^AEX", start, end)['Adj Close'].to_frame()
ATOI = yf.download("^ATOI", start, end)['Adj Close'].to_frame()
BVSP = yf.download("^BVSP", start, end)['Adj Close'].to_frame()
MIB = yf.download("FTSEMIB.MI", start, end)['Adj Close'].to_frame()
ATX = yf.download("^ATX", start, end)['Adj Close'].to_frame()
ISEQ = yf.download("^ISEQ", start, end)['Adj Close'].to_frame()
NSEI = yf.download("^NSEI", start, end)['Adj Close'].to_frame()
MXX = yf.download("^MXX", start, end)['Adj Close'].to_frame()
SSMI = yf.download("^SSMI", start, end)['Adj Close'].to_frame()
STOXX50E = yf.download("^STOXX50E", start, end)['Adj Close'].to_frame()
MDAXI = yf.download("^MDAXI", start, end)['Adj Close'].to_frame()
SDAXI = yf.download("^SDAXI", start, end)['Adj Close'].to_frame()
HSCC = yf.download("^HSCC", start, end)['Adj Close'].to_frame()
HSCE = yf.download("^HSCE", start, end)['Adj Close'].to_frame()
KLSE = yf.download("^KLSE", start, end)['Adj Close'].to_frame()
```

```python
# Transform into Dataframe
df = pd.concat([
    Dow,
    Nasdaq,
    FTSE100,
    FTSE250,
    FTAI,
    N225,
    SSE,
    Shenzhen,
    DAX,
    France,
    Indonesia,
    PSEI,
    AXKO,
    kospi,
    NZ50,
    RUI,
    RUT,
    RUA,
    GSPTSE,
    N100,
    N150,
    BFX,
    IMOEX,
    MERV,
    TA125,
    JNOU,
    SPY,
    Singapore,
    AEX,
    ATOI,
    BVSP,
    MIB,
    ATX,
    ISEQ,
    MXX,
    STOXX50E,
    MDAXI,
    SDAXI,
    KLSE
], axis=1)

# Set Columns
df.columns=[
    'US-Dow',
    'US-Nasdaq',
```

```
    'GB-FTSE100',
    'GB-FTSE250',
    'GB-FTAI',
    'JP-N225',
    'CN-SSE',
    'CN-Shenzhen',
    'DE-DAX',
    'FR-FCHI',
    'ID-JKSE',
    'PH-PSEI',
    'AU-AXKO',
    'KR-KSII',
    'NZ-NZ50',
    'US-RUI',
    'US-RUT',
    'US-RUA',
    'CA-GSPTSE',
    'FR-N100',
    'FR-N150',
    'BE-BFS',
    'RU-IMOEX',
    'AR-MERV',
    'IL-TA125',
    'ZA-JNOU',
    'US-SPX',
    'SG-STI',
    'NL-AEX',
    'AU-ATOI',
    'BR-BVSP',
    'IT-MIB',
    'AT-ATX',
    'IE-ISEQ',
    'MX-MXX',
    'DE-Stoxx50E',
    'DE-MDAXI',
    'DE-SDAXI',
    'MY-KLSE'
]

# Calculating Percent Change

daily_return = df.diff(1)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

Preprocess Data (Eliminate Missing Values) As there are missing values in the dataframe I just generated, I use `fillna` function to eliminate missing values.

```
In [3]: # Eliminate Missing Values
        daily_return = df.fillna(method='ffill').fillna(method='bfill')

        # Normalize Data
        daily_return = z_score(daily_return)

        # Copy it for the future use
        daily_return1 = daily_return
```

4.2.2 COVID-19 Confirmed Data

```
In [4]: # COVID-19 Dataset

        states_url = "https://covidtracking.com/api/states/daily"
        us_url = "https://covidtracking.com/api/us/daily"
        case_threshold = 100

        cases = ["confirmed", "deaths", "recovered"]
        sheet = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19
        suffix = "_global.csv"
        df_list = []

        url_confirmed = sheet + "confirmed" + suffix

        df_confirmed = pd.read_csv(url_confirmed, header=0, escapechar="\\")
        df_confirmed1 = df_confirmed.drop(columns=["Lat", "Long"])
        df_confirmed = df_confirmed.drop(columns=["Lat", "Long"])

        df_confirmed = df_confirmed.groupby("Country/Region").agg("sum").T
        df_confirmed1 = df_confirmed1.groupby("Province/State").agg("sum").T

        # Preprocess Data
        dates = df_confirmed.index.tolist()
        dates = date_convert(dates)
        US = df_confirmed["US"].tolist()
        China = df_confirmed["China"].tolist()
        Germany = df_confirmed["Germany"].tolist()
        Japan = df_confirmed["Japan"].tolist()
        UK = df_confirmed["United Kingdom"].tolist()
        Korea = df_confirmed["Korea, South"].tolist()
        Australia = df_confirmed["Australia"].tolist()
        Austria = df_confirmed["Austria"].tolist()
        Denmark = df_confirmed["Denmark"].tolist()
        Greece = df_confirmed["Greece"].tolist()
        Finland = df_confirmed["Finland"].tolist()
```

```python
Ireland = df_confirmed["Ireland"].tolist()
Italy = df_confirmed["Italy"].tolist()
SouthAfrica = df_confirmed["South Africa"].tolist()
Spain = df_confirmed["Spain"].tolist()
Singapore = df_confirmed["Singapore"].tolist()
Russia = df_confirmed["Russia"].tolist()
NewZealand = df_confirmed["New Zealand"].tolist()
Canada = df_confirmed["Canada"].tolist()
France = df_confirmed["France"].tolist()
Netherlands = df_confirmed["Netherlands"].tolist()
Mexico = df_confirmed["Mexico"].tolist()
Brazil = df_confirmed["Brazil"].tolist()
Philippines = df_confirmed["Philippines"].tolist()
India = df_confirmed["India"].tolist()
Argentina = df_confirmed["Argentina"].tolist()
Indonesia = df_confirmed["Indonesia"].tolist()
Malaysia = df_confirmed["Malaysia"].tolist()
Israel = df_confirmed["Israel"].tolist()
Poland = df_confirmed["Poland"].tolist()
Afghanistan = df_confirmed["Afghanistan"].tolist()

data = [
    US, China, Japan,
    Korea, Australia, Austria,
    Germany, UK, Denmark,
    Greece, Italy, SouthAfrica,
    Spain, Singapore, Russia,
    NewZealand, Canada, France,
    Netherlands, Mexico, Philippines,
    India, Argentina, Indonesia,
    Malaysia, Israel, Poland,
    Brazil, Spain
]

# Country Codes
country_codes = [
    "US", "CN", "JP",
    "KR", "AU", "AT",
    "DE", "GB", "DK",
    "GR", "IT", "ZA",
    "ES", "SG", "RU",
    "NZ", "CA", "FR",
    "NL", "MX", "PH",
    "IN", "AR", "ID",
    "MY", "IL", "PL",
    "BR", "ES"
]
```

```
daily_confirmed = pd.DataFrame(data, index=country_codes, columns=dates).T.diff(1).repl
daily_confirmed = z_score(daily_confirmed)

for code in country_codes:
    population = CountryInfo(code).population()
    daily_confirmed[code] = daily_confirmed[code].div(population, axis=0)

daily_confirmed.index.name = 'Date'
daily_confirmed1 = daily_confirmed
```

4.2.3 Merge Dataframe

We merge two dataframe, which are **Daily Return** and **Daily Confirmed**.

```
In [5]: # List of Dates
        confirmed = daily_confirmed.index.tolist()
        returns = daily_return.index.tolist()

        # Build a list to include dates in common
        dates_common = []
        for date in returns:
            date = (str(date)[:10])
            if date in confirmed:
                dates_common.append(date)

        # Only leave dates in common from daily_confirmed
        for date in daily_confirmed.index:
            if date not in dates_common:
                daily_confirmed = daily_confirmed.drop(date)

        # Only leave dates in common from daily_return
        daily_return_index = []
        for var in daily_return.index.tolist():
            date = (str(var))[:10]
            if date not in dates_common:
                daily_return = daily_return.drop(var)

            else:
                daily_return_index.append(str(date))

        daily_return.index = daily_return_index
        daily_return.index.name = 'Date'

        # Now, merge them in same index
        df_merged = pd.concat([daily_return, daily_confirmed], axis=1)

        # Normalize
        df_merged = z_score(df_merged)
```

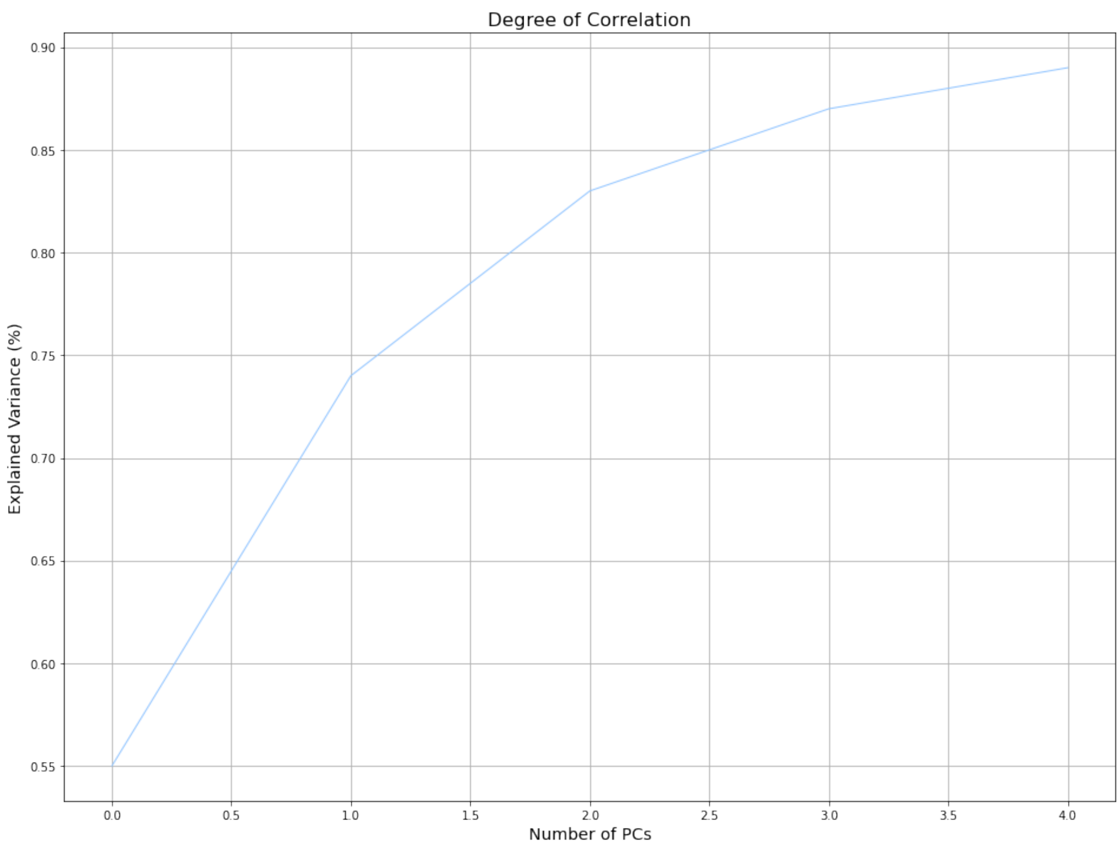
4.3 Extraction of Five Principal Components

```
In [6]: pca = PCA(5).fit(df_merged)
        daily_return_factors = pd.Series(index=df_merged.columns, data=pca.components_[0])
        print("Principal Components", pca.explained_variance_ratio_.round(2))
```

```
Principal Components [0.55 0.19 0.08 0.04 0.02]
```

4.4 Visualization of Explanatory Power for Each Principal Component

```
In [7]: variance_stock = PCA(5).fit(df_merged).explained_variance_ratio_.cumsum().round(2)
        plt.plot(variance_stock)
        plt.title('Degree of Correlation', fontsize = 16)
        plt.xlabel('Number of PCs', fontsize = 14)
        plt.ylabel('Explained Variance (%)', fontsize = 14)
        plt.show()
```



Principal Components seem to indicate approximately 55% is explained by the first principal component, whereas two principal components combinedly explain 70% of all variance. If we extend the degree of variance into five, almost 90% is explained.

5 Analysis on Eigenvectors

While our previous experiment clarified that **explained variance** caused correlation to have certain relationship to a certain extent, it does not appear to have as powerful as I originally expected. If my null hypothesis that

5.1 Meaning of Eigenvectors

Intuitively speaking, Principal Component Analysis is a technique to reduce dimensionality. Smaller dimension of features can be achieved by a linear combination of columns, which explain the maximum variation explained. This concept is what we used to understand `correlation` and `systemic risk` in the previous section of analysis. Each Principal Component Loading is an example of `unit vector`.

$$u := \min\left(\frac{1}{n}\sum_{i}^{n}(x_i^T x_i - (u_1^T x_i)^2)\right)$$

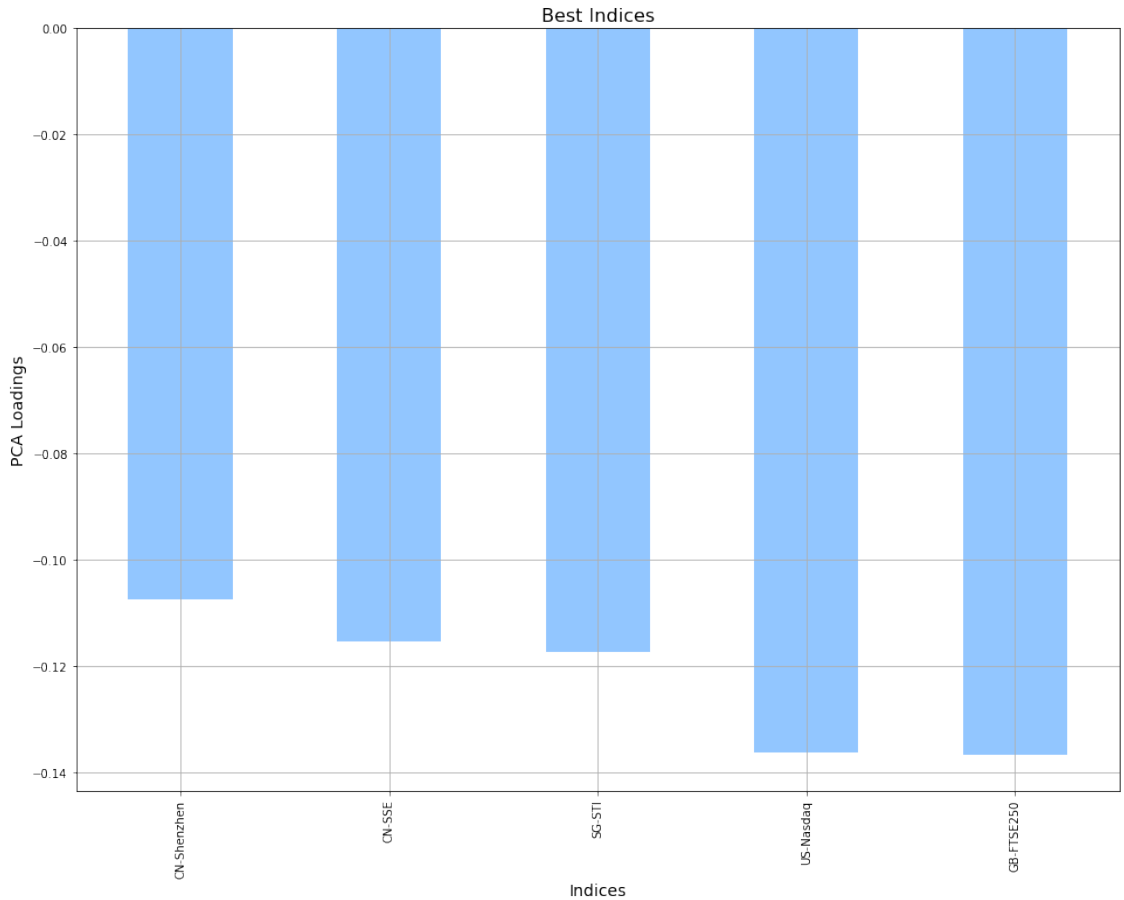
Principal Component Analysis aims for minimizing total distance of a unit vector whose perpendicular distance is minimized as a result. And it is the eigenvector of the covariance matrix of X.

$$Av = \lambda v$$

5.2 PCA Loadings as "Degree of Impact"

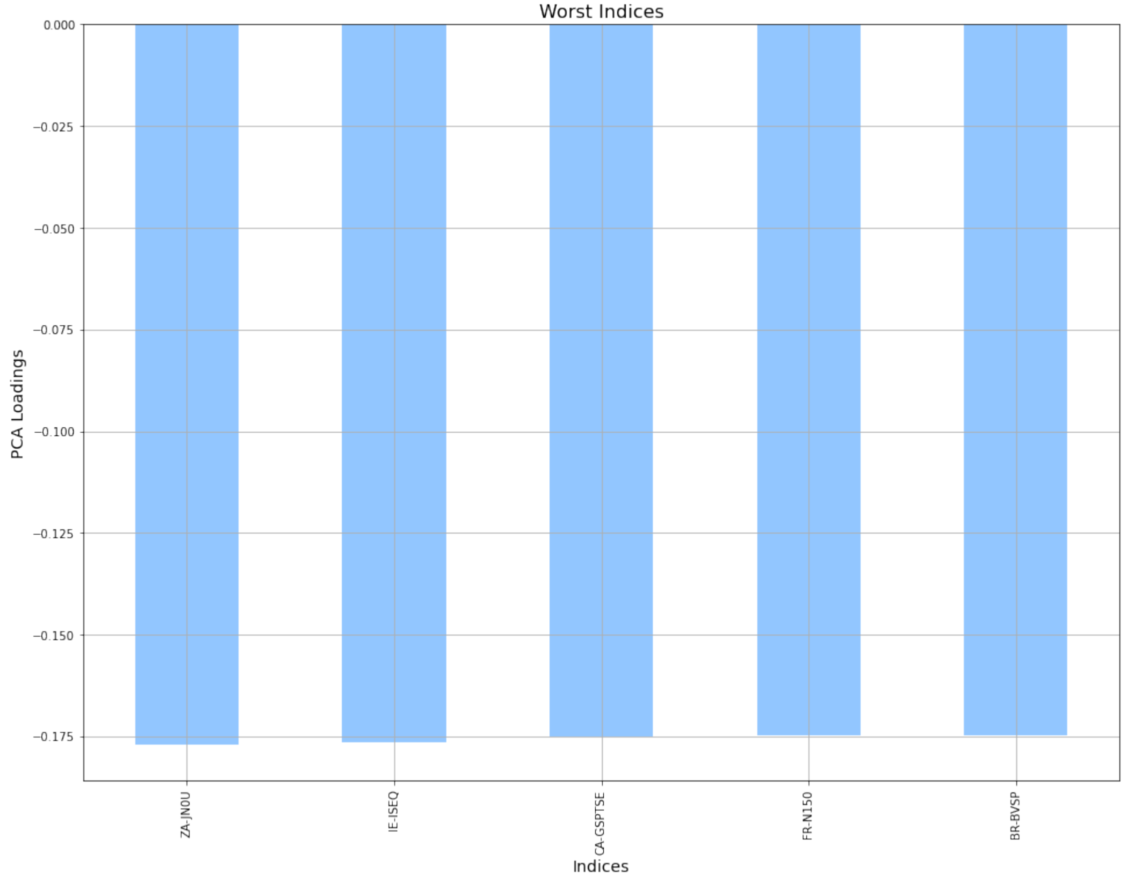
While it is about 'degree of impact', we may verify how large each extent is.

```
In [8]: pca_dr = PCA(1).fit(daily_return1)
        daily_return_factors = pd.Series(index=daily_return1.columns, data=pca_dr.components_[(

        daily_return_factors.nlargest(5).plot.bar()
        plt.title('Best Indices', fontsize=16)
        plt.xlabel('Indices', fontsize=14)
        plt.ylabel('PCA Loadings', fontsize = 14)
        # plt.savefig('Best Indices.png', dpi=300)

Out[8]: Text(0, 0.5, 'PCA Loadings')
```



In [9]:
```
pca_dr = PCA(1).fit(daily_return1)
daily_return_factors = pd.Series(index=daily_return1.columns, data=pca_dr.components_[(

daily_return_factors.nsmallest(5).plot.bar()
plt.title('Worst Indices', fontsize=16)
plt.xlabel('Indices', fontsize=14)
plt.ylabel('PCA Loadings', fontsize = 14)
```

Out[9]:
```
Text(0, 0.5, 'PCA Loadings')
```



6 Analysis

6.1 K-Means Clustering

```
In [10]: pca_covid = PCA(2).fit(daily_confirmed1)
         daily_confirmed1 = daily_confirmed1.diff(1)[1:]
         daily_confirmed_factors = pd.Series(index=daily_confirmed1.columns, data=pca_covid.co

         df1 = pd.DataFrame(daily_return_factors)
         df2 = pd.DataFrame(daily_confirmed_factors)

         df_list = []

         df1_temp = df1.T
         df2_temp = df2.T

         for index in df1_temp:
             nation = index.split("-")[0]
             indice = index.split("-")[1]
```

```
    factor_return = df1_temp[index]
    try:
        factor_input = df2_temp[nation]
    except:
        factor_input = df2_temp['FR'] + df2_temp['DE'] / 2

    df_list.append(
        [nation,
         indice,
         index,
         factor_input.values[0],
         factor_return.values[0]
        ]
    )

df = pd.DataFrame(
    df_list,
    columns=["Country", "Index", "Full-Name", "Confirmed", "Return"]
)

index = df.iloc[:,1].values.tolist()
df_default = df
df = df[['Confirmed', 'Return']]
df.index = index

kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 300,
    "random_state": 42
}

sse = []

kmeans = KMeans(
    init="random",
    n_clusters=3,
    n_init=100,
    max_iter=300,
    random_state=42
)

kmeans.fit(df)

print("The lowest SSE:", kmeans.inertia_)
print("Final Centroids:", kmeans.cluster_centers_)
print("Number of Iterations Required:", kmeans.n_iter_)
```

```
a = df[kmeans.labels_== 0]
b = df[kmeans.labels_== 1]
c = df[kmeans.labels_== 2]

a = a.index.values.tolist()
b = b.index.values.tolist()
c = c.index.values.tolist()

groups = pd.DataFrame([a, b, c], index=['Group A', 'Group B', 'Group C'])
groups = groups.T

print("RESULTS:")

groups
```

```
The lowest SSE: 0.08925300834959994
Final Centroids: [[-0.24552855 -0.15450734]
 [ 0.40383958 -0.13983685]
 [-0.02241021 -0.16070023]]
Number of Iterations Required: 5
RESULTS:
```

Out[10]:

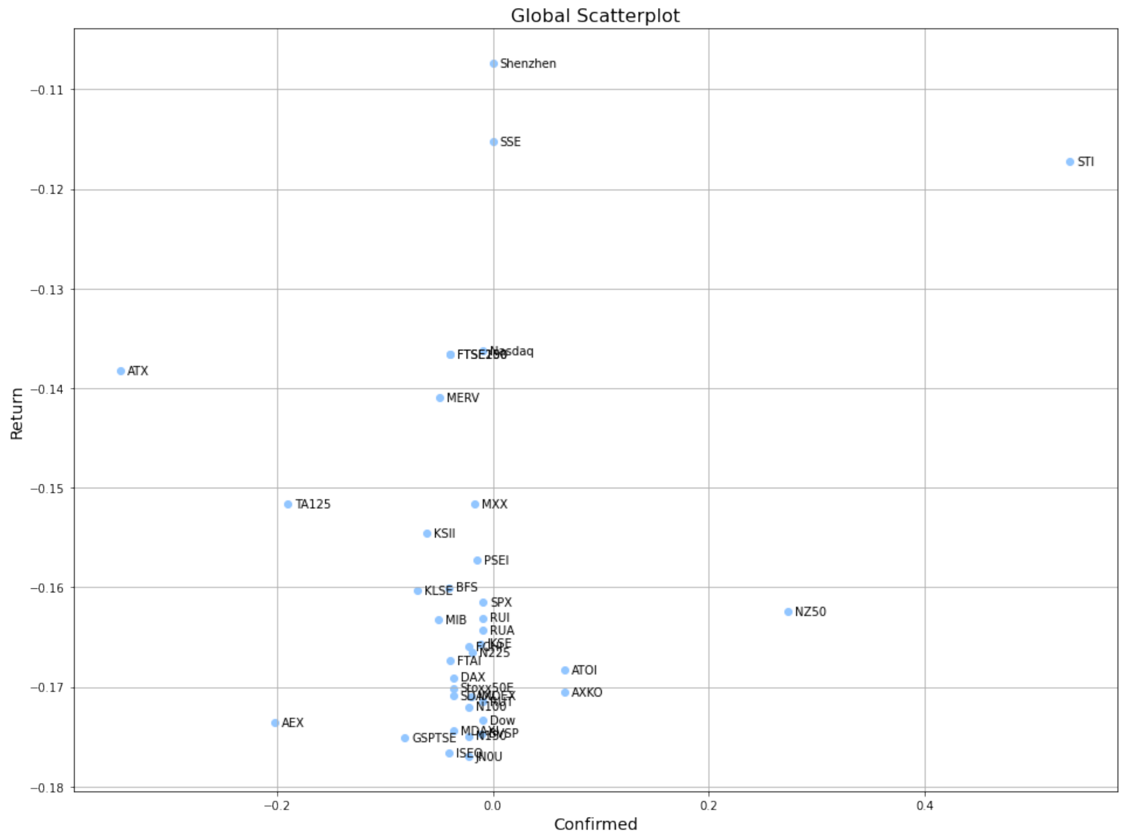
	Group A	Group B	Group C
0	TA125	NZ50	Dow
1	AEX	STI	Nasdaq
2	ATX	None	FTSE100
3	None	None	FTSE250
4	None	None	FTAI
5	None	None	N225
6	None	None	SSE
7	None	None	Shenzhen
8	None	None	DAX
9	None	None	FCHI
10	None	None	JKSE
11	None	None	PSEI
12	None	None	AXKO
13	None	None	KSII
14	None	None	RUI
15	None	None	RUT
16	None	None	RUA
17	None	None	GSPTSE
18	None	None	N100
19	None	None	N150
20	None	None	BFS
21	None	None	IMOEX
22	None	None	MERV
23	None	None	JNOU

```
24     None     None        SPX
25     None     None       ATOI
26     None     None       BVSP
27     None     None        MIB
28     None     None       ISEQ
29     None     None        MXX
30     None     None   Stoxx50E
31     None     None      MDAXI
32     None     None      SDAXI
33     None     None       KLSE
```

6.1.1 Visualization

As a data scientist, I add "visualization" to let my analysis more intuitive and straightforward.

```
In [11]: fig = plt.scatter(df[['Confirmed']].values, df[['Return']].values)

         labels = df.index.tolist()

         for x_pos, y_pos, label in zip(df[['Confirmed']].values, df[['Return']].values, labels
             plt.annotate(label,
                          xy = (x_pos, y_pos),
                          xytext=(6,0),
                          textcoords='offset points',
                          ha='left',
                          va='center'
                         )

         plt.title("Global Scatterplot", fontsize = 16)
         plt.xlabel('Confirmed', fontsize=14)
         plt.ylabel('Return', fontsize = 14)

Out[11]: Text(0, 0.5, 'Return')
```



6.2 Generalized Linear Model

Finally, I use generalized linear model, which is to have statistical regression to confirm correlation between spread of Coronavirus and trend of performance of market indices.

In [12]:
```
import statsmodels.formula.api as smf

GLSAR = smf.glm(
    data=df,
    formula='Return~Confirmed'
    ).fit()

GLSAR.summary()
```

Out[12]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                 Return   No. Observations:                   39
Model:                            GLM   Df Residuals:                       37
Model Family:                Gaussian   Df Model:                            1
```

```
Link Function:                 identity   Scale:                          0.00031360
Method:                            IRLS   Log-Likelihood:                     103.00
Date:                  Sun, 30 May 2021   Deviance:                         0.011603
Time:                          14:58:49   Pearson chi2:                       0.0116
No. Iterations:                       3
Covariance Type:              nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.1586      0.003    -55.367      0.000      -0.164      -0.153
Confirmed      0.0299      0.023      1.297      0.195      -0.015       0.075
==============================================================================
"""
```

6.3 Takeaways

To sum up, it appears that machine learning algorithms prove that Coronavirus-19 is not the appropriate factor to generalize the fluctuation in the stock market. There are a number of factors that provide some insights; **(1) strong monetary policy and fiscal policy globally done in 2020, (2) the gap between real variables and nominal variables in macroeconomy, and (3) sudden euphoria arose from sudden increase in asset price from massive money supply**.

7 Bibliography

[1] Kritzman, Mark and Li, Yuanzhen and Page, Sebastien and Rigobon, Roberto, Principal Components as a Measure of Systemic Risk (June 30, 2010). MIT Sloan Research Paper No. 4785-10, https://doi.org/10.3905/jpm.2011.37.4.112, Available at SSRN: https://ssrn.com/abstract=1633027 or http://dx.doi.org/10.2139/ssrn.1633027