Import library function

In [5]:
```
# Data Analysis
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import os
import missingno as msno

# Data View
pd.options.display.max_columns = 200

# Import Basic Visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Data Visualization : Plotly library
import chart_studio.plotly as py
import cufflinks as cf
cf.go_offline(connected = True )

import plotly.express as px

import plotly.graph_objects as go
import plotly.offline as pyo
pyo.init_notebook_mode()

from plotly.subplots import make_subplots
import plotly.figure_factory as ff
```

In [ ]:
```
# Plotly Dash
# Prerequsite
import dash
import dash_core_components as dcc
import dash_bootstrap_components as dbc
import dash_html_components as html
import dash_table
from dash.dependencies import Input, Output
from jupyter_dash import JupyterDash

# Data Analysis
import pandas as pd
import numpy as np

# Making Figure
import plotly.express as px
import plotly.graph_objects as go

# Operate Dash on jupyterlab
from jupyter_dash import JupyterDash
```

In [ ]:
```
# Data Preprocessing
from sklearn.model_selection import StratifiedKFold, train_test_split, cross_val_predict, GridSearchCV
from sklearn.preprocessing import PolynomialFeatures, StandardScaler, PolynomialFeatures

# Model Construction
from sklearn.pipeline import Pipeline

# Model


# Model Evaluation
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
from sklearn.model_selection import cross_val_score
```

Data glimpse function

1. Missing data

In [6]:
```
def missing(df) :
    missing_number = df.isnull().sum().sort_values(ascending = False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending = False)
    missing_values = pd.concat([missing_number, missing_percent], axis = 1, keys = ['Missing_number', 'Missing_percent'])
    return missing_values
```

2. Column categorize

In [ ]:
```
def categorize(df) :
    Quantitive_features = df.select_dtypes([np.number]).columns.tolist()
    Categorical_features = df.select_dtypes(exclude = [np.number]).columns.tolist()
    Discrete_features = [col for col in Quantitive_features if len(df[col].unique()) < 10]
    Continuous_features = [col for col in Quantitive_features if col not in Discrete_features]
    print(f"Quantitive feautres : {Quantitive_features} \nDiscrete features : {Discrete_features} \nContinous features : {Continuous_features} \nCategorical features : {Categorical_features}\n")
    print(f"Number of quantitive feautres : {len(Quantitive_features)} \nNumber of discrete features : {len(Discrete_features)} \nNumber of continous features : {len(Continuous_features)} \nNumber of categorical features : {len(Categorical_features)}")
```

3. Final code

In [ ]:
```
def missing(df) :
    """
    This function shows number of missing values and its percetages
    """
    missing_number = df.isnull().sum().sort_values(ascending = False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending = False)
    missing_values = pd.concat([missing_number, missing_percent], axis = 1, keys = ['Missing_number', 'Missing_percent'])
    return missing_values

def categorize(df) :
    """
    This function shows number of features by dtypes.
    Result of function is not always accruate because this result estimate dtypes before preprocessing.
    """
    Quantitive_features = df.select_dtypes([np.number]).columns.tolist()
    Categorical_features = df.select_dtypes(exclude = [np.number]).columns.tolist()
    Discrete_features = [col for col in Quantitive_features if len(df[col].unique()) < 10]
    Continuous_features = [col for col in Quantitive_features if col not in Discrete_features]
    print(f"Quantitive feautres : {Quantitive_features} \nDiscrete features : {Discrete_features} \nContinous features : {Continuous_features} \nCategorical features : {Categorical_features}\n")
    print(f"Number of quantitive feautres : {len(Quantitive_features)} \nNumber of discrete features : {len(Discrete_features)} \nNumber of continous features : {len(Continuous_features)} \nNumber of categorical features : {len(Categorical_features)}")

def unique(df) :
    """
    This function returns table storing number of unique values and its samples.
    """
    tbl = pd.DataFrame({'Columns' : df.columns, 'Number_of_Unique' : df.nunique().values.tolist(),
                        'Sample1' : df.sample(1).values.tolist()[0], 'Sample2' : df.sample(1).values.tolist()[0],
                        'Sample3' : df.sample(1).values.tolist()[0],
                        'Sample4' : df.sample(1).values.tolist()[0], 'Sample5' : df.sample(1).values.tolist()[0]})

    return tbl

def data_glimpse(df) :

    # Dataset preview
    print("1. Dataset Preview \n")
    display(df.head())
    print("-------------------------------------------------------------------------------\n")

    # Columns information
    print("2. Column Information \n")
    print(f"Dataset have {df.shape[0]} columns and {df.shape[1]} rows")
    print("\n")
    print(f"Dataset Column name : {df.columns.values}")
    print("\n")
    categorize(df)
    print("-------------------------------------------------------------------------------\n")

    # Basic information table
    print("3. Missing data table : \n")
    display(missing(df))
    print("-------------------------------------------------------------------------------\n")

    print("4. Number of unique value by column : \n")
    display(unique(df))
    print("-------------------------------------------------------------------------------\n")

    print("5. Describe table : \n")
    display(df.describe())
    print("-------------------------------------------------------------------------------\n")

    print(df.info())
    print("-------------------------------------------------------------------------------\n")
```

Visualization Analysis

1. Quantitive + Univariate

In [1]:
```
def Quantitive_Univariate_Plot(df, fea) :
    fig = make_subplots(rows = 1, cols = 2)

    fig.add_trace(go.Histogram(
        x = df[fea],
        name = 'Histogram'
        ),
        row = 1, col = 1
    )

    fig.add_trace(go.Box(
        y = df[fea],
        name = 'Box plot'
        ),
        row = 1, col = 2
    )
    fig.update_xaxes(title_text= "Value", row=1, col=1)
    fig.update_xaxes(title_text= fea, row=1, col=2)
    fig.update_yaxes(title_text= "Count", row=1, col=1)
    fig.update_yaxes(title_text= "Value", row=1, col=2)
    fig.show()
```

2. Categorical + Univarate

In [2]:
```
def Categorical_Features_Univarate(df, fea) :
    length = len(df[fea].value_counts().keys())
    colors = px.colors.sequential.RdBu[:length]
    fig = go.Figure()
    fig.add_trace(go.Bar(
        x = df[fea].value_counts(),
        y = df[fea].value_counts().keys(),
        orientation = 'h',
        marker_color = colors))
    fig.show()
```

3. Quantitive + Multivariate

In [3]:
```
def Quantitive_Multivariate(df, fea) :
    fig = go.Figure()
    fig.add_trace(
        go.Box(
            y = df_train_raw.loc[df.Risk_Flag == 1, fea],
            name = 'risk')
    )
    fig.add_trace(
        go.Box(
            y = df_train_raw.loc[df.Risk_Flag == 0, fea],
            name = 'non_risk')
    )
    fig.update_layout(
        {
            "title": {
                "text": "<b>Multivariate Analysis between {} and Risk_Flags</b>".format(fea),
                "x": 0.5,
                "y": 0.9,
                "font": {
                    "size": 15
                }
            },
            "xaxis": {
                "title": "Risk_Flags",
                "tickfont": {
                    "size": 10
                }
            },
            "yaxis": {
                "title": fea,
                "tickfont": {
                    "size": 10
                }
            },
            "template":'plotly_white'
        }
    )

    fig.show()
```

4. Categorical + Multivariate

In [4]:
```
def Categorical_Multivarate(df, fea) :
    fig = go.Figure()

    fig.add_trace(go.Bar(
        x = df[fea].unique(),
        y = df.loc[df.Risk_Flag == 0, fea].value_counts().values,
        name = 'non_risk',
        text = df.loc[df.Risk_Flag == 0, fea].value_counts().values,
        marker_color = px.colors.sequential.RdBu[0]
    )


    fig.add_trace(go.Bar(
        x = df[fea].unique(),
        y = df.loc[df.Risk_Flag == 1, fea].value_counts().values,
        name = 'risk',
        text = df.loc[df.Risk_Flag == 0, fea].value_counts().values,
        marker_color = px.colors.sequential.RdBu[7]
    )

    fig.show()
```

5. System function

5.1 Call list of data and merge them to one.

In [1]:
```
def processing_dataframe(filename) :

    doc = pd.read_excel(PATH + filename)

    # Drop value means count

    doc_drop = doc[(doc['대계열'] == '총계') | (doc['중계열'] == '계') | (doc['소계열'] == '계')]
    doc.drop(doc_drop.index, inplace = True)

    # Column selection
    doc = doc[['대계열', '중계열', '소계열', '전체']]

    # Column processing
    doc['전체'].fillna(0, inplace = True)
    doc['전체'] = doc['전체'].astype('int64')
    year_col = filename.split(".")[0]
    doc.rename(columns = {'전체' : year_col}, inplace = True)

    return doc

def generate_dataframe_by_path(PATH) :

    file_list = os.listdir(PATH)
    first_doc = True
    file_list.sort()

    for file in file_list :
        doc = processing_dataframe(file)
        if first_doc :
            final_doc, first_doc = doc, False
        else :
            final_doc = pd.merge(final_doc, doc, how = 'outer')

    return final_doc
```