OORJA
ISSN - 0974-7869 (Print)
ISSN - 2395-6771 (Online)

MODELING NON-LINEAR ASSOCIATIONS BETWEEN INDEPENDENT AND DEPENDENT VARIABLES USING ARTIFICIAL NEURAL NETWORKS IN PYTHON

Mr. Nachiket Talwar, Student, Department of Computer Science and Engineering, Vellore Institute of Technology, Vellore

Abstract

Purpose

The purpose of this paper is to present a method of data analysis that can analyze both linear and non-linear associations between independent and dependent variables. The program is developed to help researchers examine the behaviors of Asian consumers to help firms understand the factors influencing their buying behavior.

Design/methodology/approach

The ANN program was developed using Python. It is an end-to-end program that runs all the statistical tests to confirm data suitability for ANN modeling and calculate the relative importance of each identified independent variable.

Findings

This study presents the program and the reasoning why ANN as a method of data analysis is useful. The program was developed as a part of an internship to help researchers analyze data. As a result, this program's value comes from the fact that it has been used for data analysis in many studies, some of which have already been published in top-tier refereed international journals.

Research limitations/implications

The scope of the study is limited to arguing the usefulness of ANN for behavioral researchers and presenting the program. Although studies using this program for data analysis have been published, no data was collected and analyzed in this study.

The study objective of the study was to simply present this useful program that can help researchers to analyze non-linear and linear data which does not meet the multivariate requirements of (a)normality, (b) multicollinearity, (c) linearity, and (d) homoscedasticity.

Originality/value

This study makes an original contribution by presenting a program in a way that is easy to reproduce for researchers with different backgrounds to analyze the data. The objective of writing this program was to help researchers from Asia examine complex associations between the variables that impact consumers' decisions. However, the program can be used for analyzing associations in different contexts.

Keywords: Artificial neural network; dependent variable; independent variable; Python; structural equation modeling

Introduction

Research in social science, particularly in various functional management areas such as finance, marketing, operations, and human resource management, has become more challenging with time. The main reason behind it is the growing complexity of decision-making tools and situations, which have impacted the volume and the characteristics of data available for analysis. Typically, in research on different aspects of business management using primary data, scholars prefer to use structural equation modeling (SEM), which has two variants – covariance-based SEM (CB-SEM) and variance-based SEM (VB-SEM). Some of the recent studies that have used SEM for data analysis are - Hallikainen et al. (2022) for examining the effectiveness of personalized product recommendations and price promotions in the context of online grocery retailing, Malik et al. (2022) for examining the associations between economic cost, political trust, health status and willingness to pay for improved air quality, and Khan et al. (2022) for examining the association of supply chain connectivity (SCC), supply chain information sharing (SCIS), top management commitment (TMC) and green procurement and logistics acceptance (GPLA) with green supply chain management (GSCM), amongst others.

The guidance for substantive researchers provided by Anderson and Gerbing (1988) for the use of SEM suggests that it comprises a two-step modeling approach – one involving confirmatory factor analysis and another structural path analysis. Despite the continued popularity of SEM, there are certain challenges in analyzing data using SEM. To explain further, as discussed by scholars, SEM, particularly CB-SEM, has stringent data-related requirements in terms of sample size, outliers, and multivariate characteristics. Non-linearity is particularly challenging because the traditional SEM-based approach cannot be used. A way out is using the Artificial Neural Network Approach (ANN), which allows researchers to examine the difference in the relative influence of antecedents on outcome variables.

Responding to the growing need to analyze non-linear data, the present study presents a tool for applying the Artificial Neural Network Approach (ANN) developed using Python. Several studies have employed this tool for testing the relative influence of independent variables on dependent variables (e.g., M. Talwar et al., 2021; S.Talwar et al., 2021).

Background literature

Artificial neural networks (ANN)

Artificial neural networks are a concept that is part of deep learning. Deep learning is a part of machine learning but differs from them because deep learning uses artificial neural networks. The neural networks created can work on three types of learning- supervised, unsupervised and semi-supervised. Neural networks are a concept based on the human brain. The human brain is a gigantic mesh of wires that connect several neurons. ANNs are based on this in the sense that the neurons, when connected during the creation of a neural network, loosely resemble the human brain. A neural network can study data to extract relevant patterns to our questions. The neural network itself is a technique of data analysis. A neural network has loosely three layers- input layer, hidden layer, and output layer.

Different models can be created with a different number of hidden layers. These layers can have a different number of neurons. These hidden layers help determine the relationship between the various variables and find the desired output.

The input layer is the layer that takes the data to be analyzed as input and fits it so that the neural network can read the data and work on it. The hidden layer is the layer where the main work of the neural network takes place. The relationships present are found and understood in this layer. The output layer provides the neural network's final output based on the data given as input and the results obtained from the hidden layer. The hidden layer can have many layers- the more layers, the more the analysis performed on data, and the more accurate the results tend to be.

Neural networks are broadly divided into three types- Artificial Neural Networks (ANN), Convolution Neural Networks (CNN), and Recurrent Neural Networks (RNN). Convolution Neural Networks are predominantly used in Computer Vision and Image Processing- trying to perform analysis using images or identifying what is present in the image. Recurrent Neural Networks allow the output from one layer to be the input to the next layer. In general, the outputs of various layers are independent, but Recurrent Neural Networks provide a solution when the dependency is required.

Artificial Neural Networks have a variety of uses ranging from image recognition, speech recognition, and machine translation to medical diagnosis. They also have several advantages we must look out for, like the ability to store all the data in the network, excellent fault tolerance, and parallel processing ability. However, it also has limitations- hardware dependency, complex algorithm creation, and black-box systems, which provide probable results. The feasibility of neural networks has always been questioned due to their complexity. Thus, the creation and use of neural networks must be done carefully to provide the best possible results. As discussed in Talwar, Talwar, Tarjanne, & Dhir (2021), independent and dependent variables can have a more complex relationship than simple linear associations, and such non-linear associations can be examined and measured better through ANN, which can accommodate both types of associations.

Python

Python is a high-level programming language that is based on the English language. It came to fruition 31 years ago and has dominated computer programming thanks to its simple syntax structure, which is easy to replicate and understand for those who are not proficient in programming. Python has many packages that facilitate mathematical functions, predominantly done in R, from machine learning algorithms for graphical data representation. The various packages in python that were used in our model are-

(i) Pandas- This package provides a python program with data manipulation and analysis techniques.

(ii) Sklearn- It is formally known as scikit learn, and its primary use is to provide tools for machine learning and statistical modeling like regressors, and classifiers, among others.

(iii) Numpy- This package provides multi-dimensional array and matrix support and a large variety of high-level mathematical functions.

(iv) Openpyxl- This package enables python programs to read from or write to excel files.

(v) Statsmodels- It provides high-level statistic functions on top of numpy, matplotlib, and scipy.

(vi) Scipy- It stands for Scientific Python and provides python programs with functions for optimization and linear algebra, among others.

(vii) Random- This package provides various functions to generate random values.

(viii) SALib- It stands for Sensitivity Analysis Library and provides python programs with the tools to perform sensitivity analysis.

(ix) Factor_analyzer- This package allows users to perform exploratory factor analysis.

Each module above provided the necessary touches to create our model. Pandas are typically used to create data frames from various data sets we want to read into python programs. Sk learn acts as the heart of the model as it is what we utilize to create the Neural Network and split the dataset into test and training datasets to test and train the model. MLP Regressor is used from the SK learn library to create the neural network. However, this package also allows us to access machine learning models like linear regression, decision trees, and k- means clustering, to name a few. Numpy library provides us with tools to clean the data up. We want to try and create a model that is as efficient as possible. Using Numpy, we can clean up datasets so that the models or networks only need to process absolutely necessary data. Openpyxl is instrumental in documenting the results of our analysis. Documenting and storing the results clearly and concisely is important as the results can be understood easily if they are put together and explained properly. In the course of our sensitivity analysis, we needed help from two libraries to bring it to fruition- one being SA Lib and the other being stats models. The stats models library gave us access to Variance Inflation Factor (VIF), which is instrumental in performing sensitivity analysis. The SA Lib provides access to the tools used to perform the sensitivity analysis.

3. Model/Program Description

The program presented in this study utilizes the concept of neural networks. As a first step, the program runs a series of tests to prepare the data and check whether neural network usage is appropriate or not. To do so, first, the dataset is imported into Python and stored using Pandas. Pandas package provides the required dataframe function for this purpose. This package can be used to create a dataframe for data input from excel files. The dataframe gives a table-like presentation to the data, which it mimics from the excel sheet.

The data, after being input, must be cleaned before analyzing it. For cleaning the data, various columns having data or specific values that are of no consequence need to be identified and dropped. This stage requires a thorough investigation of the data to identify the redundant information in the dataframe and proceed to remove it.

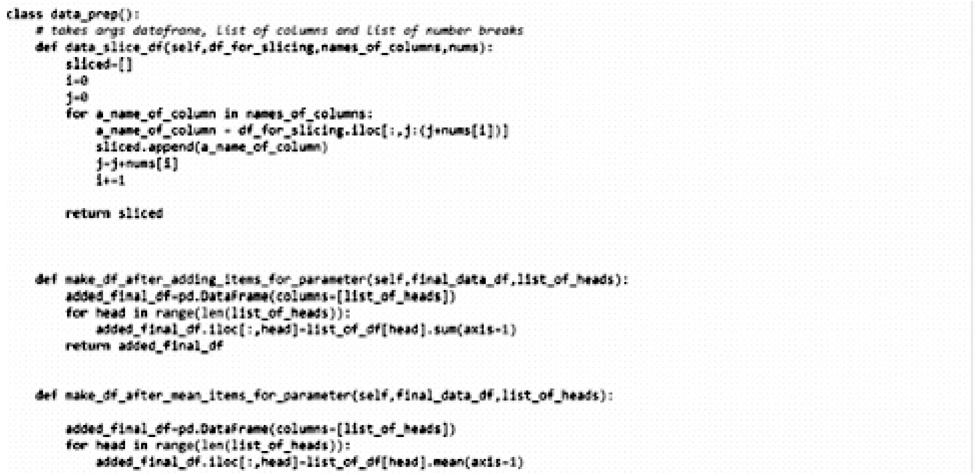
Cleaning the data is essential to ensure that there is no redundancy, which in turn is necessary to make the program as efficient as possible. If we provide the program with unnecessary data and pointless data to process, it slows down the program and increases the chances of error. Removing inconsequential values or columns also helps to make the data compact, i.e., provides good readability and allows the models to work faster. Data cleaning also helps in getting good accuracy as only necessary data will be processed, giving results as close as possible to the actual answer. In sum, data cleaning essentially involves working on following data redundancies:

- removal of null/NaN type of data, which cannot be processed by the model
- removal of mismatched data types (for example: text data in lieu of numbers needs to be removed)

Once the data is cleaned, the program proceeds to run several tests to confirm the suitability of the data for further analysis and reporting.

The next step after the data is cleaned and the variables of interest are identified is to prepare the data for further processing and analysis. To explain explicitly, the study takes the example of a hypothetical variable, say ITTN, comprising multiple items (ITTN1, ITTN2, ITTN3, ITTN4, ITTN5, ITTN6, ITTN7). Items may be dropped or carried forward on the basis of Cronbach Alpha values, i.e., some items may be dropped on a statistical basis. Assume that ITTN 3 and ITTN 5 need to be dropped in this regard. So, in the case of the proposed hypothetical variable, ITTN, only five items (ITTN1, ITTN2, ITTN4, ITTN6, ITTN7) will be taken forward. Means of the Likert-scale responses for each of these items are calculated and stored in a separate dataframe for future reference. This step is undertaken for each variable.

To execute this step, a Python Class is created with multiple functions to carry out specific tasks. The class entails the slicing function to remove unnecessary data and two functions to create a dataframe for the mean and sum of the values of the remaining variables, as shown in **Figure 1**. Cronbach Alpha is calculated in a separate Class: 'tests()', as presented in **Figure 2a**. Cronbach alpha shows how closely related the set of items measuring the same variable are. In the context of this study, it shows how coherent the items (ITTN1, ITTN2, ITTN3, ITTN4, ITTN5, ITTN6, ITTN7) are with one another. The stage is now set for checking whether the data is viable for neural network usage or not.

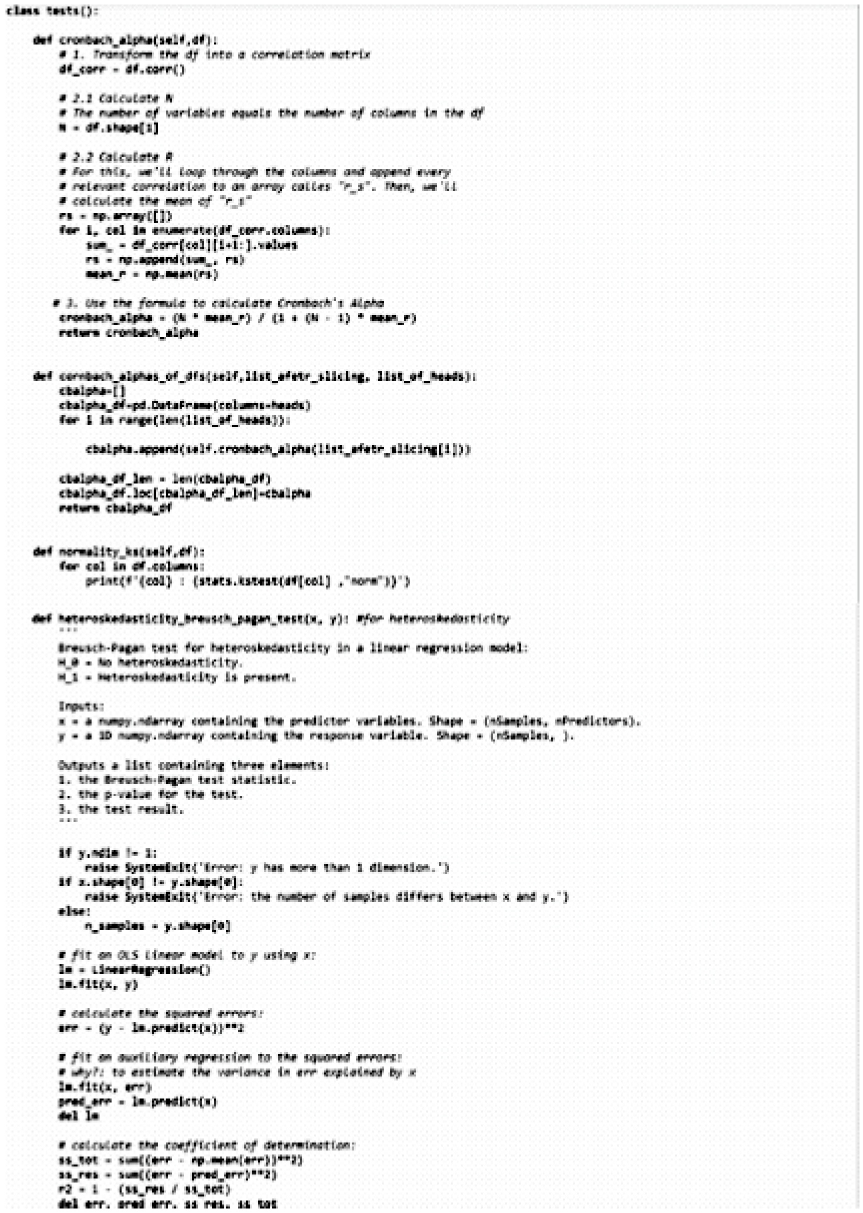


```
class data_prep():
    # takes args dataframe, list of columns and list of number breaks
    def data_slice_df(self,df_for_slicing,names_of_columns,nums):
        sliced=[]
        i=0
        j=0
        for a_name_of_column in names_of_columns:
            a_name_of_column = df_for_slicing.iloc[:,j:(j+nums[i])]
            sliced.append(a_name_of_column)
            j=j+nums[i]
            i+=1

        return sliced



    def make_df_after_adding_items_for_parameter(self,final_data_df,list_of_heads):
        added_final_df=pd.DataFrame(columns=[list_of_heads])
        for head in range(len(list_of_heads)):
            added_final_df.iloc[:,head]=list_of_df[head].sum(axis=1)
        return added_final_df


    def make_df_after_mean_items_for_parameter(self,final_data_df,list_of_heads):

        added_final_df=pd.DataFrame(columns=[list_of_heads])
        for head in range(len(list_of_heads)):
            added_final_df.iloc[:,head]=list_of_df[head].mean(axis=1)
```

Figure 1

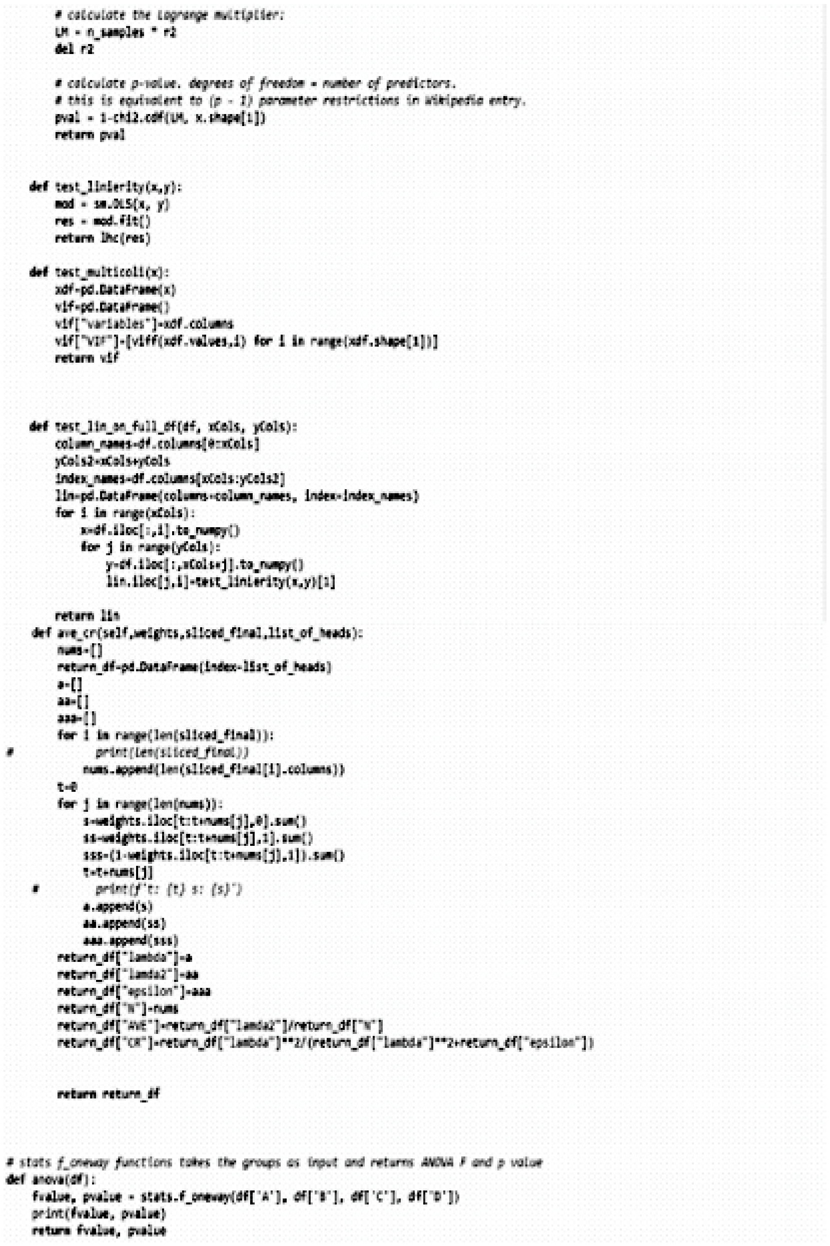
In the next step, the program executes a battery of tests for checking the multivariate characteristics of the data and the validity and reliability statistics. These tests are run in a separate Class: 'tests()', as presented in **Figures 2a and b**. Here, four multivariate assumptions are checked: normality, homoscedasticity, linearity, and multicollinearity. The Kolmogorov–Smirnov (K–S) test is performed to check the normality of the data provided. If the test outcomes shows that the data are not normally distributed, it indicates that using ANN is warranted. The next test is Breusch-Pagan's heteroscedasticity test which checks if the data is homoscedastic, i.e., adequately distributed around a fitted line. If the data is not homoscedastic, that means using ANN is justified. For checking linearity, the program used the ANOVA function. This function indicates if there is non-linear relationship between the variables of interest. Non-linearity also provides justification for using ANN, although it is capable of handling linear relationships as well. For diagnosing the fourth multivariate characteristic, multicollinearity, the program used Variant Inflation Factor (VIF). If the VIF values are in line with the recommended threshold, it indicates the absence of collinearity.

Coming to the validity and reliability tests, the ave_cr() function Class: 'tests()', as presented in Figures 2a and b. is provided.



```
class tests():

    def cronbach_alpha(self,df):
        # 1. Transform the df into a correlation matrix
        df_corr = df.corr()

        # 2.1 Calculate N
        # The number of variables equals the number of columns in the df
        N = df.shape[1]

        # 2.2 Calculate R
        # For this, we'll loop through the columns and append every
        # relevant correlation to an array calles "r_s". Then, we'll
        # calculate the mean of "r_s"
        rs = np.array([])
        for i, col in enumerate(df_corr.columns):
            sum_ = df_corr[col][i+1:].values
            rs = np.append(sum_, rs)
            mean_r = np.mean(rs)

       # 3. Use the formula to calculate Cronbach's Alpha
        cronbach_alpha = (N * mean_r) / (1 + (N - 1) * mean_r)
        return cronbach_alpha


    def cornbach_alphas_of_dfs(self,list_afetr_slicing, list_of_heads):
        cbalpha=[]
        cbalpha_df=pd.DataFrame(columns=heads)
        for i in range(len(list_of_heads)):

            cbalpha.append(self.cronbach_alpha(list_afetr_slicing[i]))

        cbalpha_df_len = len(cbalpha_df)
        cbalpha_df.loc[cbalpha_df_len]=cbalpha
        return cbalpha_df


    def normality_ks(self,df):
        for col in df.columns:
            print(f'{col} : {stats.kstest(df[col] ,"norm")}')

    def heteroskedasticity_breusch_pagan_test(x, y): #for heteroskedasticity
        '''
        Breusch-Pagan test for heteroskedasticity in a linear regression model:
        H_0 = No heteroskedasticity.
        H_1 = Heteroskedasticity is present.

        Inputs:
        x = a numpy.ndarray containing the predictor variables. Shape = (nSamples, nPredictors).
        y = a 1D numpy.ndarray containing the response variable. Shape = (nSamples, ).

        Outputs a list containing three elements:
        1. the Breusch-Pagan test statistic.
        2. the p-value for the test.
        3. the test result.
        '''

        if y.ndim != 1:
            raise SystemExit('Error: y has more than 1 dimension.')
        if x.shape[0] != y.shape[0]:
            raise SystemExit('Error: the number of samples differs between x and y.')
        else:
            n_samples = y.shape[0]

        # fit an OLS linear model to y using x:
        lm = LinearRegression()
        lm.fit(x, y)

        # calculate the squared errors:
        err = (y - lm.predict(x))**2

        # fit an auxiliary regression to the squared errors:
        # why?: to estimate the variance in err explained by x
        lm.fit(x, err)
        pred_err = lm.predict(x)
        del lm

        # calculate the coefficient of determination:
        ss_tot = sum((err - np.mean(err))**2)
        ss_res = sum((err - pred_err)**2)
        r2 = 1 - (ss_res / ss_tot)
        del err, pred_err, ss_res, ss_tot
```

Figure 2a: Class: 'tests()'

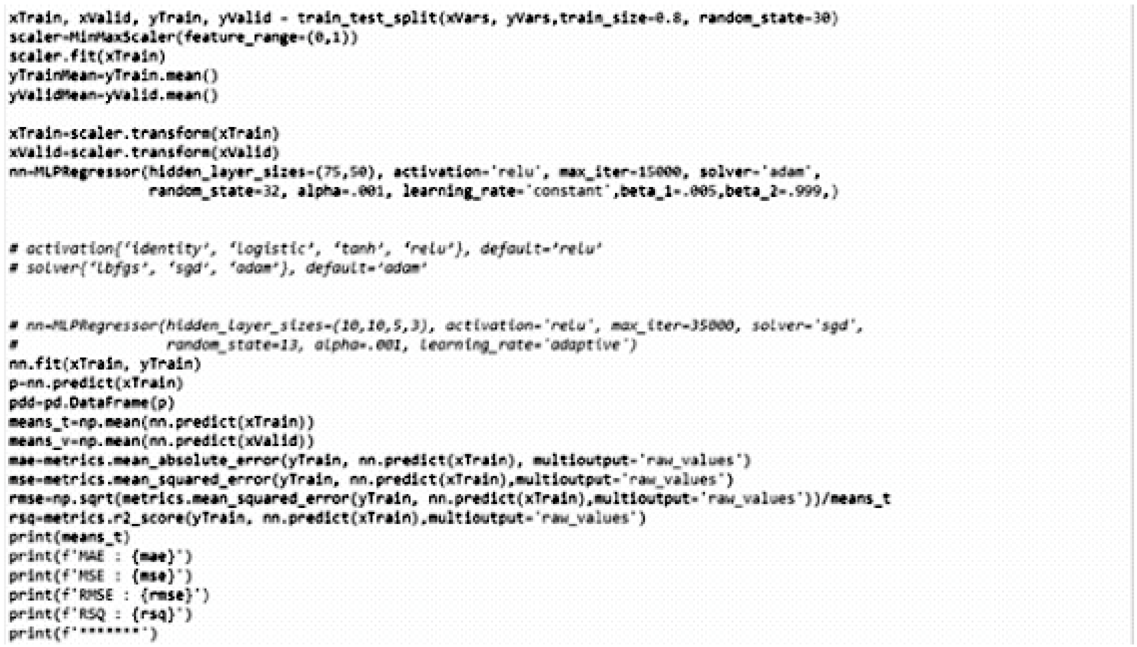


```
        # calculate the Lagrange multiplier:
        LM = n_samples * r2
        del r2

        # calculate p-value. degrees of freedom = number of predictors.
        # this is equivalent to (p - 1) parameter restrictions in Wikipedia entry.
        pval = 1-chi2.cdf(LM, x.shape[1])
        return pval


    def test_linierity(x,y):
        mod = sm.OLS(x, y)
        res = mod.fit()
        return lhc(res)

    def test_multicoli(x):
        xdf=pd.DataFrame(x)
        vif=pd.DataFrame()
        vif["variables"]=xdf.columns
        vif["VIF"]=[viff(xdf.values,i) for i in range(xdf.shape[1])]
        return vif



    def test_lin_on_full_df(df, xCols, yCols):
        column_names=df.columns[0:xCols]
        yCols2=xCols+yCols
        index_names=df.columns[xCols:yCols2]
        lin=pd.DataFrame(columns=column_names, index=index_names)
        for i in range(xCols):
            x=df.iloc[:,i].to_numpy()
            for j in range(yCols):
                y=df.iloc[:,xCols+j].to_numpy()
                lin.iloc[j,i]=test_linierity(x,y)[1]

        return lin
    def ave_cr(self,weights,sliced_final,list_of_heads):
        nums=[]
        return_df=pd.DataFrame(index=list_of_heads)
        a=[]
        aa=[]
        aaa=[]
        for i in range(len(sliced_final)):
#             print(len(sliced_final))
            nums.append(len(sliced_final[i].columns))
        t=0
        for j in range(len(nums)):
            s=weights.iloc[t:t+nums[j],0].sum()
            ss=weights.iloc[t:t+nums[j],1].sum()
            sss=(1-weights.iloc[t:t+nums[j],1]).sum()
            t=t+nums[j]
    #         print(f't: {t} s: {s}')
            a.append(s)
            aa.append(ss)
            aaa.append(sss)
        return_df["lambda"]=a
        return_df["lamda2"]=aa
        return_df["epsilon"]=aaa
        return_df["N"]=nums
        return_df["AVE"]=return_df["lamda2"]/return_df["N"]
        return_df["CR"]=return_df["lambda"]**2/(return_df["lambda"]**2+return_df["epsilon"])


        return return_df


# stats f_oneway functions takes the groups as input and returns ANOVA F and p value
def anova(df):
    fvalue, pvalue = stats.f_oneway(df['A'], df['B'], df['C'], df['D'])
    print(fvalue, pvalue)
    return fvalue, pvalue
```

Figure 2b: Class: 'tests()'

In the final step, the data is analyzed in neural networks. It comprises various steps. For this, the program separates the original dataset into x (independent) variables (say, O, C, E, A, N) and y (dependent) variables (say, ITTN). This data is used to train and test the neural network model. To execute, the dataset is divided into two parts, testing data and training data, with 80% of data used to train the model and the remaining 20% being used to test the model. The 80:20 bifurcation is not a standard protocol, but is usually done as a thumb rule. The model presented in this study can accommodate any bifurcation the researchers want (for example 70:30). Moving on, now the neurons and number of hidden layers need to be specified. The model allows researchers to specify their preferred number of neurons and hidden layers. For example, there can be a neural network with 75 and 50 neurons and two hidden layers, respectively. In addition, the model is flexible enough to allow researchers' specification of the following relevant values: activation, maximum iteration, solver, random state, alpha, learning rate, beta_1 and beta_2.

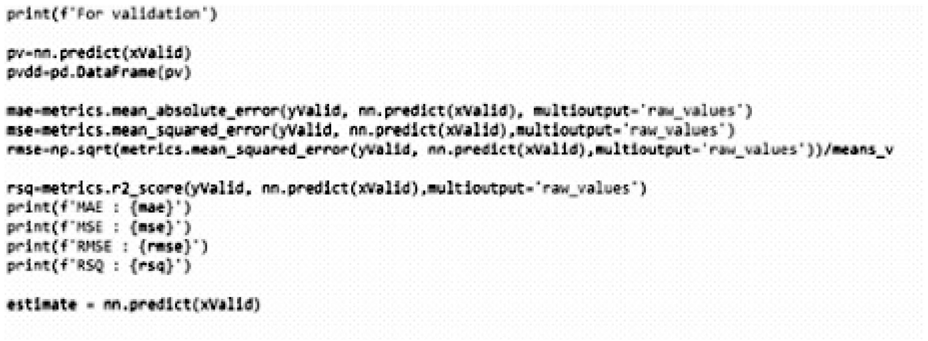
MLPRegressor was utilized to create the neural network as it is one of the best frameworks available for the purpose. After specification of all the above indicated values, the model is trained using the training dataset created by splitting the original dataset. The results of training the model are measured using four parameters: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R2_score (RSQ). Multiple model specifications are tested to arrive at the most robust model, which is confirmed from the value of RMSE being the lowest of the set. For this purpose, a function is provided to run the model multiple times. The relevant codes are presented in **Figure 3.**



```
xTrain, xValid, yTrain, yValid = train_test_split(xVars, yVars,train_size=0.8, random_state=30)
scaler=MinMaxScaler(feature_range=(0,1))
scaler.fit(xTrain)
yTrainMean=yTrain.mean()
yValidMean=yValid.mean()

xTrain=scaler.transform(xTrain)
xValid=scaler.transform(xValid)
nn=MLPRegressor(hidden_layer_sizes=(75,50), activation='relu', max_iter=15000, solver='adam',
                random_state=32, alpha=.001, learning_rate='constant',beta_1=.005,beta_2=.999,)


# activation{'identity', 'logistic', 'tanh', 'relu'}, default='relu'
# solver{'lbfgs', 'sgd', 'adam'}, default='adam'


# nn=MLPRegressor(hidden_layer_sizes=(10,10,5,3), activation='relu', max_iter=35000, solver='sgd',
#                 random_state=13, alpha=.001, learning_rate='adaptive')
nn.fit(xTrain, yTrain)
p=nn.predict(xTrain)
pdd=pd.DataFrame(p)
means_t=np.mean(nn.predict(xTrain))
means_v=np.mean(nn.predict(xValid))
mae=metrics.mean_absolute_error(yTrain, nn.predict(xTrain), multioutput='raw_values')
mse=metrics.mean_squared_error(yTrain, nn.predict(xTrain),multioutput='raw_values')
rmse=np.sqrt(metrics.mean_squared_error(yTrain, nn.predict(xTrain),multioutput='raw_values'))/means_t
rsq=metrics.r2_score(yTrain, nn.predict(xTrain),multioutput='raw_values')
print(means_t)
print(f'MAE : {mae}')
print(f'MSE : {mse}')
print(f'RMSE : {rmse}')
print(f'RSQ : {rsq}')
print(f'*******')
```

Figure 3: The ANN model

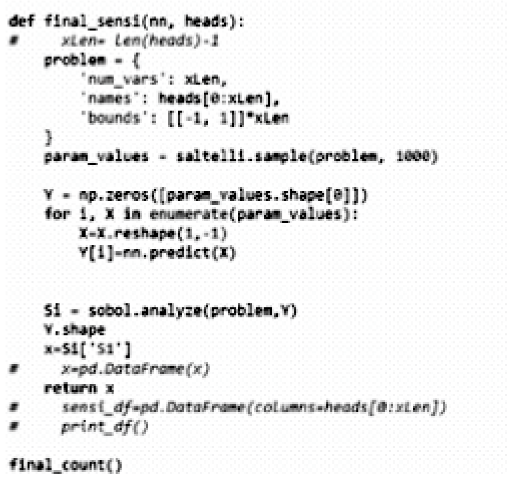
Once the model is trained and most robust model identified, the validation dataset is run on the model and the values of four parameters: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R2_score (RSQ) are generated. If these values are also acceptable as per the recommendation, then the model is considered valid. The code is presented in **Figure 4.**



```
print(f'For validation')

pv=nn.predict(xValid)
pvdd=pd.DataFrame(pv)

mae=metrics.mean_absolute_error(yValid, nn.predict(xValid), multioutput='raw_values')
mse=metrics.mean_squared_error(yValid, nn.predict(xValid),multioutput='raw_values')
rmse=np.sqrt(metrics.mean_squared_error(yValid, nn.predict(xValid),multioutput='raw_values'))/means_v

rsq=metrics.r2_score(yValid, nn.predict(xValid),multioutput='raw_values')
print(f'MAE : {mae}')
print(f'MSE : {mse}')
print(f'RMSE : {rmse}')
print(f'RSQ : {rsq}')

estimate = nn.predict(xValid)
```

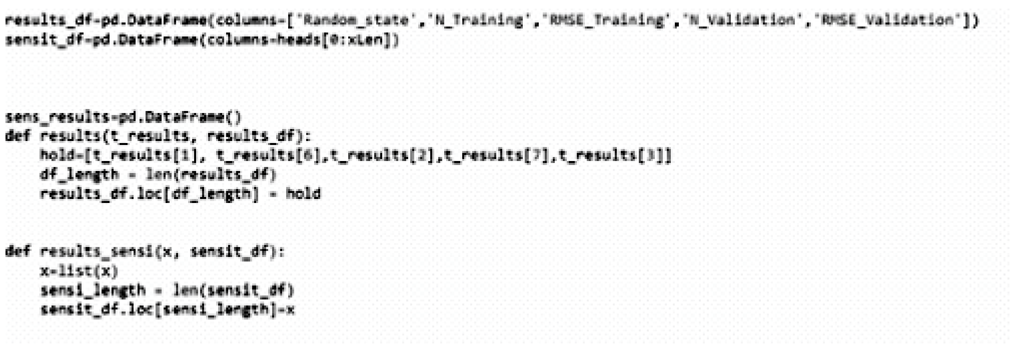
Figure 4 :Validation of the model

This is followed by sensitivity analysis, the code for which is presented in **Figures 5 a and b.** Sensitivity analysis is required to examine the strength of the influence of the identified independent variables on the dependent variables. A separate function is created for this purpose, named final_sensi(). The function uses SALib library of Python. Sensitivity analysis can be done with thousands of random inputs, and the model allows flexibility for the same. The function returns the normalized relative importance of each independent variable, which is finally reported as a percentage of the maximum relative importance.



```
def final_sensi(nn, heads):
#     xLen= len(heads)-1
    problem = {
        'num_vars': xLen,
        'names': heads[0:xLen],
        'bounds': [[-1, 1]]*xLen
    }
    param_values = saltelli.sample(problem, 1000)

    Y = np.zeros([param_values.shape[0]])
    for i, X in enumerate(param_values):
        X=X.reshape(1,-1)
        Y[i]=nn.predict(X)


    Si = sobol.analyze(problem,Y)
    Y.shape
    x=Si['S1']
#     x=pd.DataFrame(x)
    return x
#     sensi_df=pd.DataFrame(columns=heads[0:xLen])
#     print_df()

final_count()
```

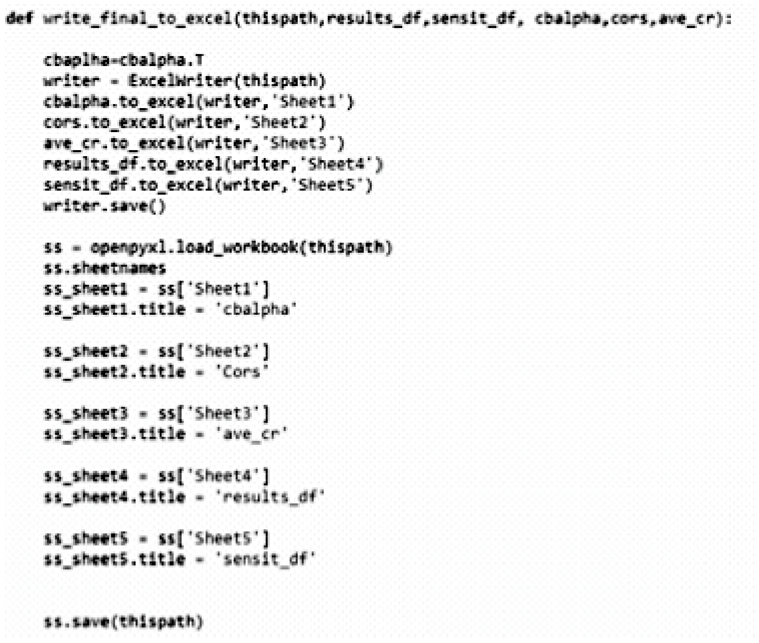
Figure 5a: Sensitivity Analysis



```
results_df=pd.DataFrame(columns=['Random_state','N_Training','RMSE_Training','N_Validation','RMSE_Validation'])
sensit_df=pd.DataFrame(columns=heads[0:xLen])



sens_results=pd.DataFrame()
def results(t_results, results_df):
    hold=[t_results[1], t_results[6],t_results[2],t_results[7],t_results[3]]
    df_length = len(results_df)
    results_df.loc[df_length] = hold


def results_sensi(x, sensit_df):
    x=list(x)
    sensi_length = len(sensit_df)
    sensit_df.loc[sensi_length]=x
```

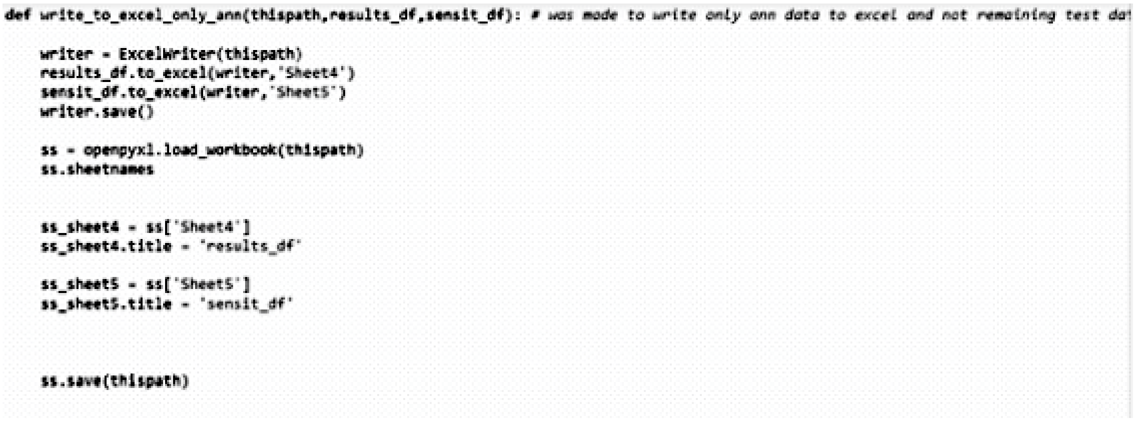
Figure 5b: Sensitivity analysis

The final step in neural network analysis is data tabulation, where the results of the functions and tests are saved in excel files for easy readability and understanding. This tabulation is done across several excel files, which store the results from multiple analyses performed, such as the Cronbach alpha test, the ave_cr test, and most importantly, the neural network and sensitivity analysis. Such systematic tabulation is helpful for researchers while reporting the results of their study clearly. The codes for this extensive tabulation exercise are presented in **Figures 6a and b.**



```
def write_final_to_excel(thispath,results_df,sensit_df, cbalpha,cors,ave_cr):

    cbaplha=cbalpha.T
    writer = ExcelWriter(thispath)
    cbalpha.to_excel(writer,'Sheet1')
    cors.to_excel(writer,'Sheet2')
    ave_cr.to_excel(writer,'Sheet3')
    results_df.to_excel(writer,'Sheet4')
    sensit_df.to_excel(writer,'Sheet5')
    writer.save()

    ss = openpyxl.load_workbook(thispath)
    ss.sheetnames
    ss_sheet1 = ss['Sheet1']
    ss_sheet1.title = 'cbalpha'

    ss_sheet2 = ss['Sheet2']
    ss_sheet2.title = 'Cors'

    ss_sheet3 = ss['Sheet3']
    ss_sheet3.title = 'ave_cr'

    ss_sheet4 = ss['Sheet4']
    ss_sheet4.title = 'results_df'

    ss_sheet5 = ss['Sheet5']
    ss_sheet5.title = 'sensit_df'


    ss.save(thispath)
```

Figure 6a : Data Tabulation



```
def write_to_excel_only_ann(thispath,results_df,sensit_df): # was mode to write only ann data to excel and not remaining test dat

    writer = ExcelWriter(thispath)
    results_df.to_excel(writer,'Sheet4')
    sensit_df.to_excel(writer,'Sheet5')
    writer.save()

    ss = openpyxl.load_workbook(thispath)
    ss.sheetnames


    ss_sheet4 = ss['Sheet4']
    ss_sheet4.title = 'results_df'

    ss_sheet5 = ss['Sheet5']
    ss_sheet5.title = 'sensit_df'



    ss.save(thispath)
```

Figure 6b : Data Tabulation

Discussion and Implication

CB-SEM is a popular method of data analysis, but it has stringent data requirements (e.g., Tandon et al., 2020). VB-SEM is also popular (e.g., Laato et al.,2020), particularly since it has less strict data requirements. In their paper, Talwar, Talwar, Kaur, Tripathy, & Dhir (2021) have discussed such requirements in detail and how data characteristics influence the choice of method of analysis. The authors cited prior studies to suggest that CB-SEM can be used only when the sample size is large with no outliers. The collected data meets the four multivariate characteristics criteria: (a)normality, (b) multicollinearity, (c) linearity, and (d) homoscedasticity, as discussed by Henseler et al. (2009), Hair et al. (2016), etc. Although VB-SEM does not have such strict requirements about sample size, it has similar multivariate characteristics requirements (Hew et al., 2019; Talwar, Talwar, Tarjanne, et al., 2021). Past studies have suggested that in the cases where data is not suitable for CB or VB-SEM, and the study aims to find out the influence of chosen independent variables on dependent variables, ANN should be used (Hew et al., 2019; Leong et al., 2020).

Taking note of this, the present study developed and presented an ANN program using Python. It is an end-to-end program that runs all the statistical tests to confirm data suitability for ANN modeling and calculate the relative importance of each identified independent variable. The program offers a data analysis method to analyze linear and non-linear associations between independent and dependent variables.

The program was developed as a part of an internship to help researchers analyze data. The program is expected to help researchers examine the behaviors of Asian consumers to help firms understand the factors influencing their buying behavior and other decisions. This program's value comes from the fact that it has been used for data analysis in many studies, some of which have already been published in top-tier refereed international journals. This study makes an original contribution by presenting a program in a way that is easy to reproduce for researchers with different backgrounds to analyze the data. Writing this program aimed to help researchers from Asia examine complex associations between the variables that impact consumers' decisions. However, the program can be used for analyzing associations in different contexts.

Limitations

The study aimed to support the usefulness of ANN for behavioral researchers and present the program that can help them analyze data through ANN. One of the study's limitations is that it just presents the model and does not collect and analyze data to exhibit the program's value. However, some recent studies have already used this program, proving its usefulness. The present study's objective was to simply present the program to researchers as a useful way of analyzing non-linear and linear data which does not meet the multivariate requirements of (a)normality, (b) multicollinearity, (c) linearity, and (d) homoscedasticity.

References:

- Anderson, J. C., & Gerbing, D. W. (1988). Structural equation modeling in practice: A review and recommended two-step approach. Psychological Bulletin, 103(3), 411-423. https://doi.org/10.1037/0033-2909.103.3.411
- Hallikainen, H., Luongo, M., Dhir, A., & Laukkanen, T. (2022). Consequences of personalized product recommendations and price promotions in online grocery shopping. Journal Of Retailing And Consumer Services, 69, 103088. https://doi.org/10.1016/j.jretconser.2022.103088
- Malik, S., Arshad, M., Amjad, Z., & Bokhari, A. (2022). An empirical estimation of determining factors influencing public willingness to pay for better air quality. Journal Of Cleaner Production, 133574. https://doi.org/10.1016/j.jclepro.2022.133574
- Khan, M., Ajmal, M., Jabeen, F., Talwar, S., & Dhir, A. (2022). Green supply chain management in manufacturing firms: A resource based viewpoint. Business Strategy And The Environment. https://doi.org/10.1002/bse.3207
- Talwar, M., Talwar, S., Kaur, P., Tripathy, N., & Dhir, A. (2021). Has financial attitude impacted the trading activity of retail investors during the COVID-19 pandemic? Journal of Retailing and Consumer Services, 58, 102341. doi: 10.1016/j.jretconser.2020.102341
- Talwar, S., Talwar, M., Tarjanne, V., & Dhir, A. (2021). Why retail investors traded equity during the pandemic? An application of artificial neural networks to examine behavioral biases. Psychology & Marketing, 38(11), 2142–2163. doi:10.1002/mar.21550
- Talwar, S., Talwar, M., Kaur, P., Singh, G., & Dhir, A. (2021). Why have consumers opposed, postponed, and rejected Innovations during a pandemic? A Study of mobile payment Innovations. Australasian Journal of Information Systems, 25. https://doi.org/ 10.3127/ ajis.v25i0.3201
- Talwar, S., Srivastava, S., Sakashita, M., Islam, N., & Dhir, A. (2021). Personality and travel intentions during and after the COVID-19 pandemic: An artificial neural network (ANN) approach. Journal of Business Research. doi:10.1016/j.jbusres.2021.12.002