In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
import plotly
import plotly.express as px
import plotly.graph_objs as go
```

Data Dictionary

People

- ID: Customer's unique identifier
- Year_Birth: Customer's birth year
- Education: Customer's education level
- Marital_Status: Customer's marital status
- Income: Customer's yearly household income
- Kidhome, Teenhome: Number of children\teenagers in customer's household
- Dt_Customer: Date of customer's enrollment with the company
- Recency: Number of days since customer's last purchase
- Complain: 1 if customer complained in the last 2 years, 0 otherwise

Products

- MntWines, MntFruits : Amount spent on wine\fruits in last 2 years

- MntWines, MntFruits : Amount spent on wine\fruits in last 2 years
- MntMeatProducts, MntFishProducts: Amount spent on meat\fish in last 2 years
- MntSweetProducts, MntGoldProds: Amount spent on sweets\gold in last 2 years

Promotion

- NumDealsPurchases: Number of purchases made with a discount
- AcceptedCmp1-5: 1 if customer accepted the offer in the 1st-5th campaign, 0 otherwise
- Response: 1 if customer accepted the offer in the last campaign, 0 otherwise

Place

- NumWebPurchases: Number of purchases made through the company's web site
- NumCatalogPurchases: Number of purchases made using a catalogue
- NumStorePurchases: Number of purchases made directly in stores
- NumWebVisitsMonth: Number of visits to company's web site in the last month

Original dataset citation: https://www.kaggle.com/imakash3011/customer-personality-analysis

In [4]:
```
market_df = pd.read_csv("./marketing_campaign.csv", sep='\t')
market_df.info()
market_df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
```

```
 15  NumDealsPurchases    2240 non-null   int64
 16  NumWebPurchases      2240 non-null   int64
 17  NumCatalogPurchases  2240 non-null   int64
 18  NumStorePurchases    2240 non-null   int64
 19  NumWebVisitsMonth    2240 non-null   int64
 20  AcceptedCmp3         2240 non-null   int64
 21  AcceptedCmp4         2240 non-null   int64
 22  AcceptedCmp5         2240 non-null   int64
 23  AcceptedCmp1         2240 non-null   int64
 24  AcceptedCmp2         2240 non-null   int64
 25  Complain             2240 non-null   int64
 26  Z_CostContact        2240 non-null   int64
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

Out[4]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenh
0	5524	1957	Graduation	Single	58138.0	0	
1	2174	1954	Graduation	Single	46344.0	1	
2	4141	1965	Graduation	Together	71613.0	0	
3	6182	1984	Graduation	Together	26646.0	1	
4	5324	1981	PhD	Married	58293.0	1	
...	...	...	...	...	...	...	
2235	10870	1967	Graduation	Married	61223.0	0	
2236	4001	1946	PhD	Together	64014.0	2	
2237	7270	1981	Graduation	Divorced	56981.0	0	
2238	8235	1956	Master	Together	69245.0	0	
2239	9405	1954	PhD	Married	52869.0	1	

2240 rows × 29 columns

Any missing value?

In [5]:
```
market_df.isnull().sum()
```

Out[5]:
```
ID                     0
Year_Birth             0
Education              0
Marital_Status         0
Income                24
Kidhome                0
Teenhome               0
Dt_Customer            0
Recency                0
MntWines               0
MntFruits              0
MntMeatProducts        0
MntFishProducts        0
```

```
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
AcceptedCmp3            0
AcceptedCmp4            0
AcceptedCmp5            0
AcceptedCmp1            0
AcceptedCmp2            0
Complain                0
Z_CostContact           0
Z_Revenue               0
Response                0
dtype: int64
```

In [16]:
```
market_df.nunique()
```

Out[16]:
```
ID                   2240
Year_Birth             59
Education               5
Marital_Status          8
Income               1974
Kidhome                 3
Teenhome                3
Dt_Customer           663
Recency               100
MntWines              776
MntFruits             158
MntMeatProducts       558
MntFishProducts       182
MntSweetProducts      177
MntGoldProds          213
NumDealsPurchases      15
NumWebPurchases        15
NumCatalogPurchases    14
NumStorePurchases      14
NumWebVisitsMonth      16
AcceptedCmp3            2
AcceptedCmp4            2
AcceptedCmp5            2
AcceptedCmp1            2
AcceptedCmp2            2
Complain                2
Z_CostContact           1
Z_Revenue               1
Response                2
dtype: int64
```

There are 24 missing values in the “Income” column. Categories "Z_CostContact" and "Z_Revenue" have the same value in all the rows, as a result, they are not going to contribute anything to the model building. So we drop missing values and categories that are not useful for this assignment: “ID”, “Z_CostContact”, and “Z_Revenue”.

In [17]:
```
market_df = market_df.dropna(axis=0)
market_df = market_df.drop(['ID','Z_CostContact', 'Z_Revenue'],a
```

In [18]:
```
market_df.isnull().sum()
```

Out[18]:
```
Year_Birth             0
Education              0
Marital_Status         0
Income                 0
Kidhome                0
Teenhome               0
Dt_Customer            0
Recency                0
MntWines               0
MntFruits              0
MntMeatProducts        0
MntFishProducts        0
MntSweetProducts       0
MntGoldProds           0
NumDealsPurchases      0
NumWebPurchases        0
NumCatalogPurchases    0
NumStorePurchases      0
NumWebVisitsMonth      0
AcceptedCmp3           0
AcceptedCmp4           0
AcceptedCmp5           0
AcceptedCmp1           0
AcceptedCmp2           0
Complain               0
Response               0
dtype: int64
```

Any duplicate value?

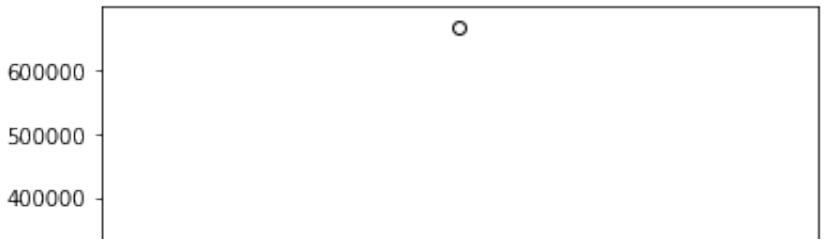
In [174…
```
market_df.duplicated().sum()
```

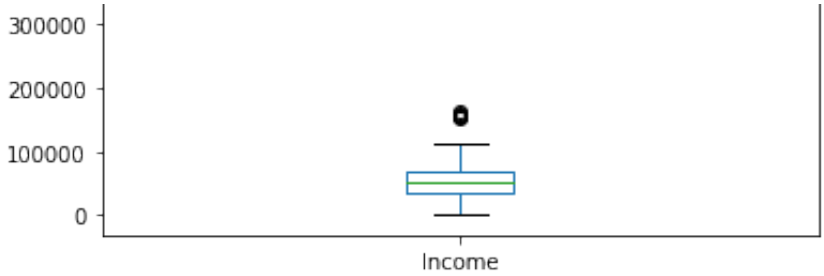
Out[174…
```
0
```

Any outliers?

In [20]:
```
market_df["Income"].plot(kind="box")
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1b20a6a00>
```

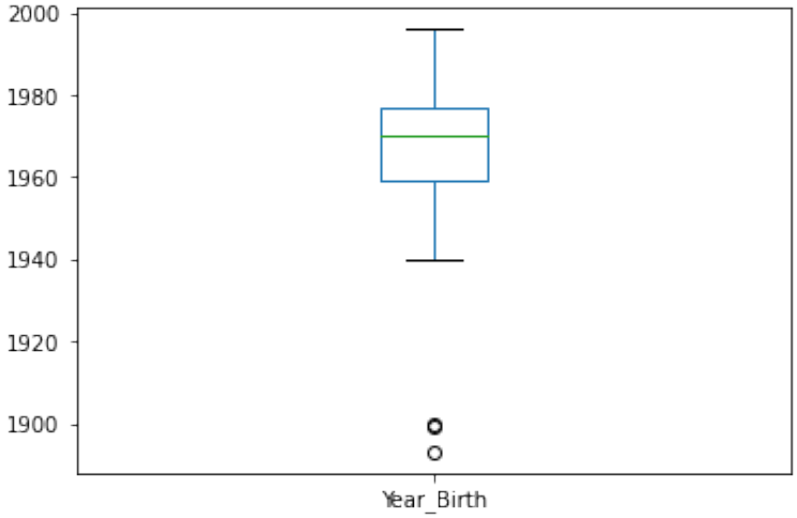




In [6]:
```
#The outliers in income are people who are extremely rich. I dec
market_df = market_df[market_df["Income"] < 500000]
```

In [7]:
```
market_df["Year_Birth"].plot(kind="box")
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fadf9161520>
```



In [8]:
```
#The outliers in year_birth are people who are extremely old. I
market_df = market_df[market_df["Year_Birth"] > 1920]
```

In [9]:
```
df = market_df.copy()
```

Numerical Statistics

In [25]:
```
df.describe()
```

Out[25]:

	Year_Birth	Income	Kidhome	Teenhome	Recency	2
count	2212.000000	2212.000000	2212.000000	2212.000000	2212.000000	
mean	1968.913653	51958.810579	0.441682	0.505877	49.019439	
std	11.701599	21527.278844	0.536955	0.544253	28.943121	
min	1940.000000	1730.000000	0.000000	0.000000	0.000000	
25%	1959.000000	35233.500000	0.000000	0.000000	24.000000	

50%	1970.000000	51371.000000	0.000000	0.000000	49.000000	
75%	1977.000000	68487.000000	1.000000	1.000000	74.000000	
max	1996.000000	162397.000000	2.000000	2.000000	99.000000	1

8 rows × 23 columns

Preprocess the dataset

In [10]:
```
#Add a variable "Age" in df: Customer current age
df['Age'] = 2014 - df["Year_Birth"]
#Calculate number of days a customer is engaged with the company
#subtract the last shopping date in the data from the customer's
df['Dt_Customer'] = pd.to_datetime(df.Dt_Customer)
df['Engaged_Days'] = -(df['Dt_Customer'] -  df['Dt_Customer'].ma
```

In [11]:
```
df.select_dtypes("object").value_counts()
```

Out[11]:
```
Education   Marital_Status
Graduation  Married           429
            Together          284
            Single            246
PhD         Married           190
Master      Married           138
Graduation  Divorced          119
PhD         Together          115
Master      Together          102
PhD         Single             96
2n Cycle    Married            80
Master      Single             75
2n Cycle    Together           56
PhD         Divorced           52
Master      Divorced           37
2n Cycle    Single             35
Graduation  Widow              35
PhD         Widow              24
2n Cycle    Divorced           22
Basic       Married            20
            Single             18
            Together           14
Master      Widow              11
2n Cycle    Widow               5
PhD         YOLO                2
Master      Alone               1
            Absurd              1
Graduation  Alone               1
            Absurd              1
Basic       Widow               1
PhD         Alone               1
Basic       Divorced            1
dtype: int64
```

In [12]:
```
#Combine different variables into one variable to reduce the num
```

```
df["Education"] = df["Education"].replace(["Graduation", "PhD",
df["Education"] = df["Education"].replace(["Basic"], "Undergradu
df['Marital_Status'] = df['Marital_Status'].replace(['Married',
df['Marital_Status'] = df['Marital_Status'].replace(['Divorced',
df['Kids'] = df['Kidhome'] + df['Teenhome']
df['Expenses'] = df['MntWines'] + df['MntFruits'] + df['MntMeatP
df['TotalAcceptedCampaign'] = df['AcceptedCmp1'] + df['AcceptedC
df['NumTotalPurchases'] = df['NumWebPurchases'] + df['NumCatalog
```

In [13]:

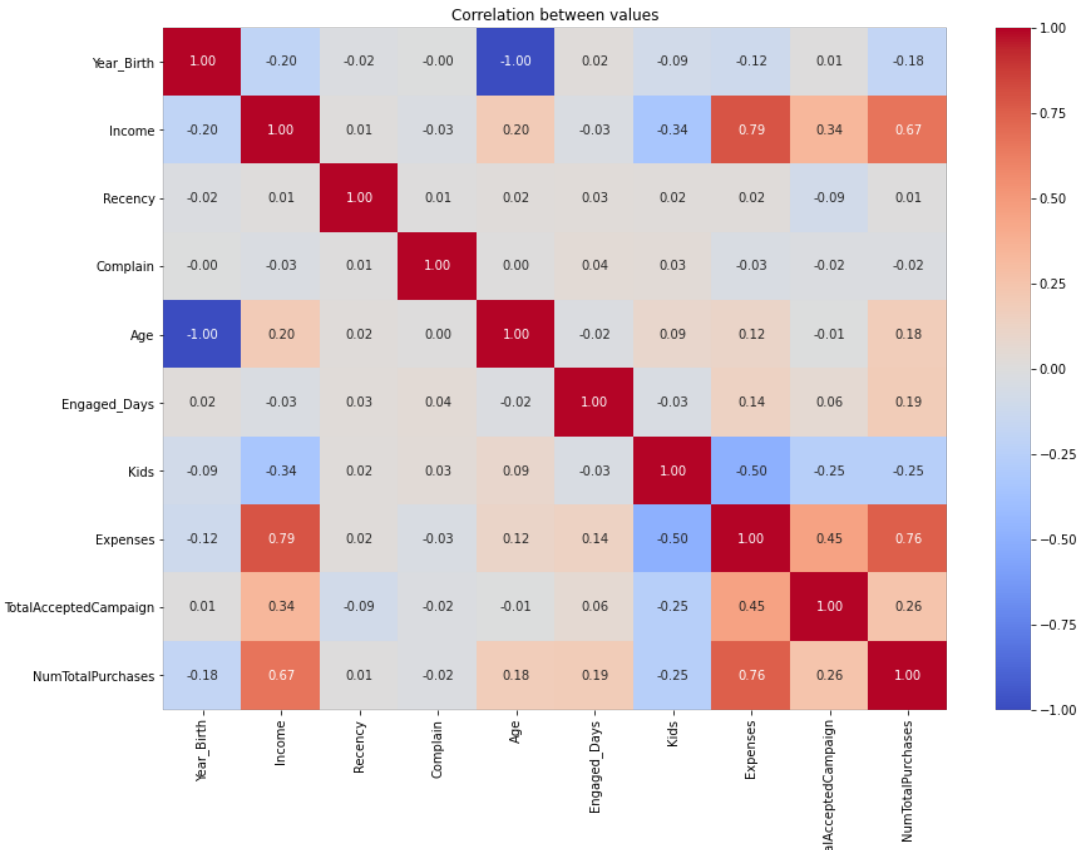
```
# Delete some columns to reduce dimension and complexity of the
col = ["Kidhome", "Teenhome","MntWines", "MntFruits", "MntMeatPr
df=df.drop(columns=col,axis=1)
df.head()
```

Out[13]:

	ID	Year_Birth	Education	Marital_Status	Income	Dt_Customer	Recenc
0	5524	1957	Graduate	Single	58138.0	2012-04-09	5
1	2174	1954	Graduate	Single	46344.0	2014-08-03	3
2	4141	1965	Graduate	Relationship	71613.0	2013-08-21	2
3	6182	1984	Graduate	Relationship	26646.0	2014-10-02	2
4	5324	1981	Graduate	Relationship	58293.0	2014-01-19	9

In [31]:

```
fig = plt.figure(figsize=(14,10))
plt.title('Correlation between values')
sns.heatmap(df.corr(), fmt= '.2f', vmin=-1, vmax=1, center=0,ann
```



When we look at the correlation table, all the data looks clean. Assuming a strong relationship above 0.70, it is obvious that there are some strong positive relationships between income and expenses; expenses, and the number of total purchases. The higher expenses are associated with higher income. Besides, there are some moderate relationships, such as kids' number between expenses and income having a moderate negative correlation. It is interesting, as I did the research, one journal gives me the answer: "The relationship between income and family size, which is hypothesized to be positive, often is negative in empirical studies. This perverse result is thought to occur because of the many correlations between income and other factors that affect fertility. In this research, these other factors--such as the net price of a child, the opportunity cost of the wife's time, and supply factors--are statistically controlled, and the income effect is positive and significant. When the net price of a child is not controlled, however, the income effect becomes negative and significant. " Citation: https://www.jstor.org/stable/2061527

Visualization Citation:https://plotly.com/python/

In [14]:
```
#Relationship between martial status and expenses
fig = px.bar(df, x='Marital_Status', y='Expenses', color="Marita
fig.show()
```

In [15]:
```
#Relationship between education and expenses
fig =  px.histogram (df, x = "Expenses", facet_row = "Education"
fig.show()
```

In [16]:
```
#Relationship between income and expenses
fig = px.histogram(df, x='Income', y='Expenses',color_discrete_s
fig.show()
```

In [17]:
```
#Relationship between customer engaged days and expenses
fig = px.histogram(df, x='Engaged_Days', y='Expenses',color_disc
fig.show()
```

In [18]:
```
#Relationship between number of kids in one customer's household
fig = px.bar(df, x='Kids', y='Expenses', color="Kids",title='Num
fig.show()
```

In [19]:
```
#Relationship between age and expenses
fig =  px.histogram(df, x='Age', y='Expenses',color_discrete_seq
fig.show()
```

PCA

In [21]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

Process high-dimensional data will cause high processing power and cost. The higher the number of features in this dataset, the more difficult to deal with. Thus it is important to do dimension reduction before we do the K-Means Clustering. I use PCA to preprocess the data to perform K-Means Clustering.

1. LabelEncoder(): encode categorical columns with value between 0 and n_classes-1,
2. StandardScaler(): standardization process by removing the mean and scaling to unit variance.
3. PCA() to reduce feature dimentions to 3-5
4. Elbow Method to determine the number of clusters in a data set.

In [22]:
```
drop_list = ['Dt_Customer','Year_Birth']
df.drop(drop_list, axis=1 ,inplace=True)
df1 = df.copy()
group = []
for i in df1.columns:
    if (df1[i].dtypes == "object"):
        group.append(i)

#print(group)

lbl_encode = LabelEncoder()
for i in group :
    df1[i]=lbl_encode.fit_transform(df1[[i]])
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validat
ion.py:63: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples, ), for example using rave
l().
```

In [23]:
```
scaler = StandardScaler()
scaler.fit(df1)
```

```
scaled_df = pd.DataFrame(scaler.transform(df), columns=df.colu
```

In [24]:
```
scaled_df.head()
```

Out[24]:

	ID	Education	Marital_Status	Income	Recency	Complain	Z_C
0	-0.018837	-0.158187	1.349603	0.287105	0.310353	-0.09552	
1	-1.050626	-0.158187	1.349603	-0.260882	-0.380813	-0.09552	
2	-0.444797	-0.158187	-0.740959	0.913196	-0.795514	-0.09552	
3	0.183824	-0.158187	-0.740959	-1.176114	-0.795514	-0.09552	
4	-0.080437	-0.158187	-0.740959	0.294307	1.554453	-0.09552	

In [25]:
```
# The number of dimensions as 3
pca = PCA(n_components=3)
pca.fit(scaled_df)
pca_data = pd.DataFrame(pca.transform(scaled_df), columns=["c1",
```

In [26]:
```
#pca.explained_variance_ratio_
```

In [27]:
```
x = pca_data["c1"]
y = pca_data["c2"]
z = pca_data["c3"]

fig = go.Figure(data=[go.Scatter3d(
    x=x,y=y,z=z,mode='markers',
    marker=dict(size=6,color=x,opacity=0.8))])

fig.update_layout( title={'text': "3D Plot of Size-Reduced Data"
        'x':0.5,'xanchor': 'center','yanchor': 'top'},
                  margin=dict(l=200, r=220, b=0, t=0))
fig.show()
```

K Means Clustering using Elbow Method

In [163…
```
from sklearn.cluster import KMeans
```

Find the optimal number of clusters

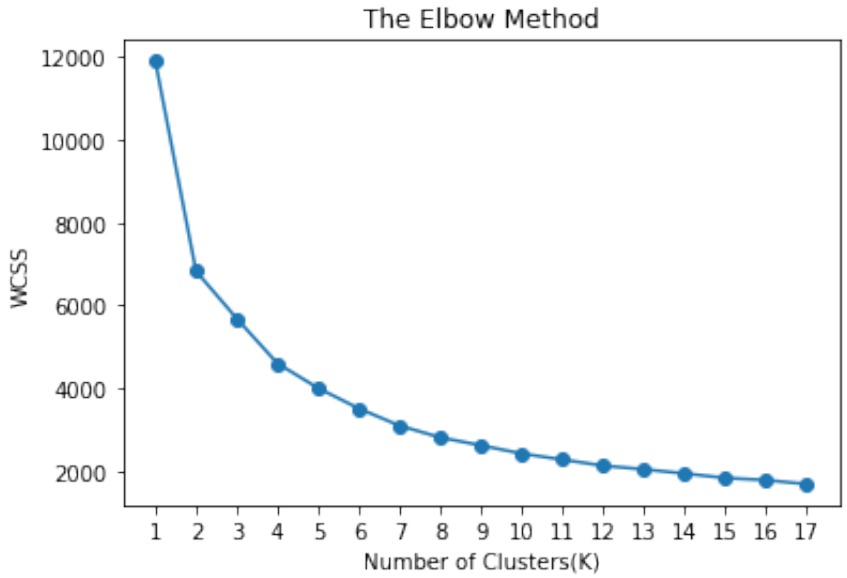
The number of clusters we select from a dataset cannot be random. Each cluster is formed by calculating and comparing the distances of data points within a cluster to its center.As a result, we calculate the Within-Cluster-Sum-of-Squares (WCSS) to find the right number of clusters. WCSS is the sum of squares of the distances of each data point in all clusters to their respective centers, and the goal is to minimize the sum. Assume there are n

observations in a dataset and we specify n number of clusters, which means k = n; so WCSS turns to 0 since data points themselves become centers and the distance will be 0, in turn this will perform a perfect cluster; but this is almost impossible as we have many clusters as the observations. Thus, we use Elbow point graph to find the optimum value for K by fitting the model in a range of values of K. We randomly initialize the K-Means algorithm for a range of K values and plot it against the WCSS for each K value.

In [181…

```
#https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of
wcss = []
k = range(1,18)
for i in k:
    model = KMeans(n_clusters=i)
    model.fit(pca_data)
    wcss.append(model.inertia_)

plt.plot(k, wcss, '-o')
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters(K)')
plt.ylabel('WCSS')
plt.xticks(k)
plt.show()
```



For the above-given graph, the optimum value for K would be 3. As we can see that with an increase in the number of clusters, the WCSS value decreases. We select the value for K, the "elbow", on the basis of the rate of decrease, to indicate the model fits best at that point. In the graph, from cluster 1 to 2 to 3 in the above graph there is a huge drop in WCSS. After 3 the drop is minimal, thus we chose 3 to be the optimal value for K. Based on the Elbow Method, we can find the optimal number of clusters is 3. According to: https://en.wikipedia.org/wiki/Elbow_method_(clustering)

In [195…

```
k_means = KMeans(n_clusters = 3, random_state = 50)
```

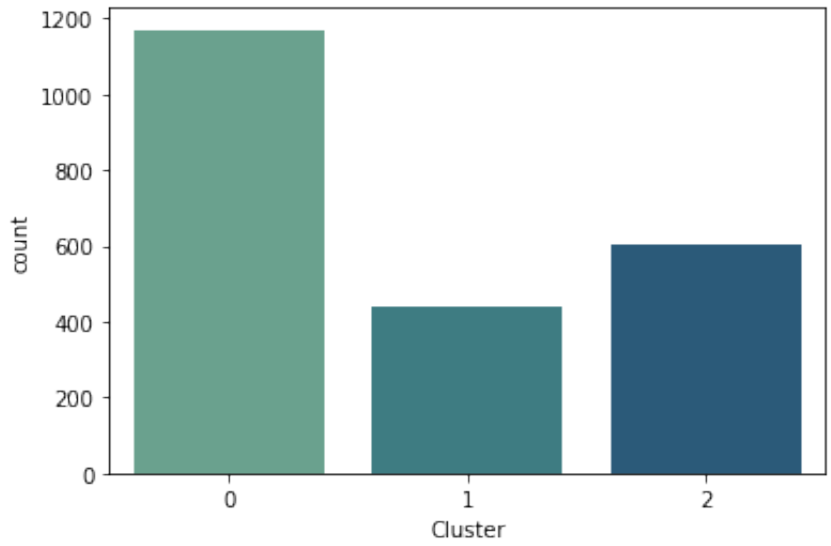
```
y_pred = k_means.fit_predict(pca_data)
pca_data['Cluster'] = y_pred
df['Cluster'] = y_pred
```

In [196…

```
sns.countplot(x=pca_data['Cluster'], palette = 'crest')
```

Out[196…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb16049bc10>
```

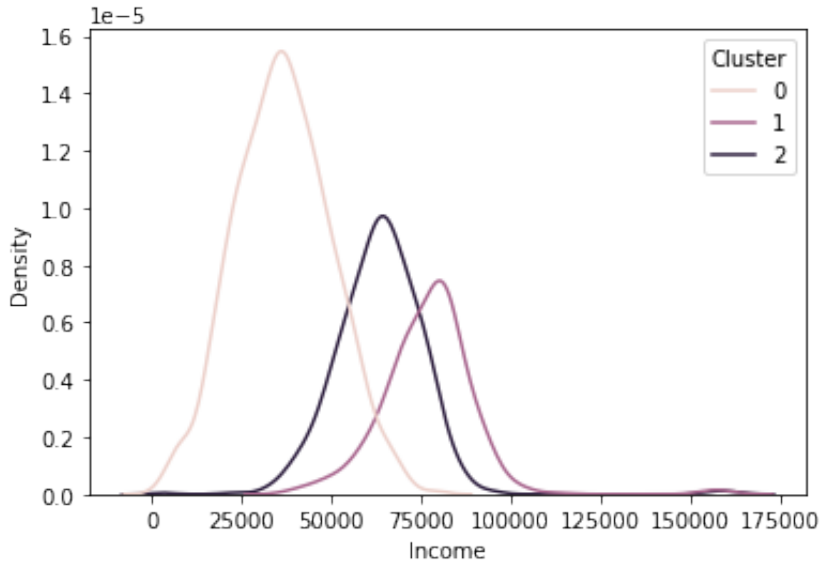


In [197…

```
sns.kdeplot(data=df, x='Income', hue='Cluster')
```

Out[197…

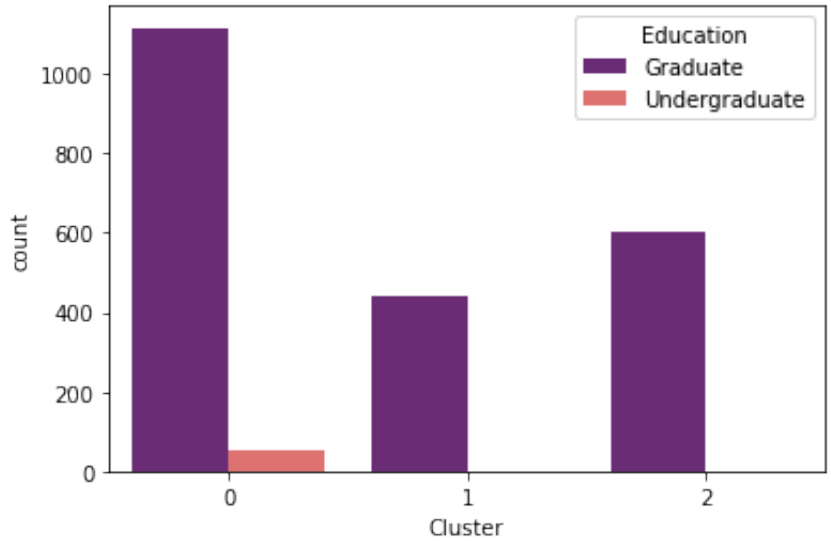
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb199190fa0>
```

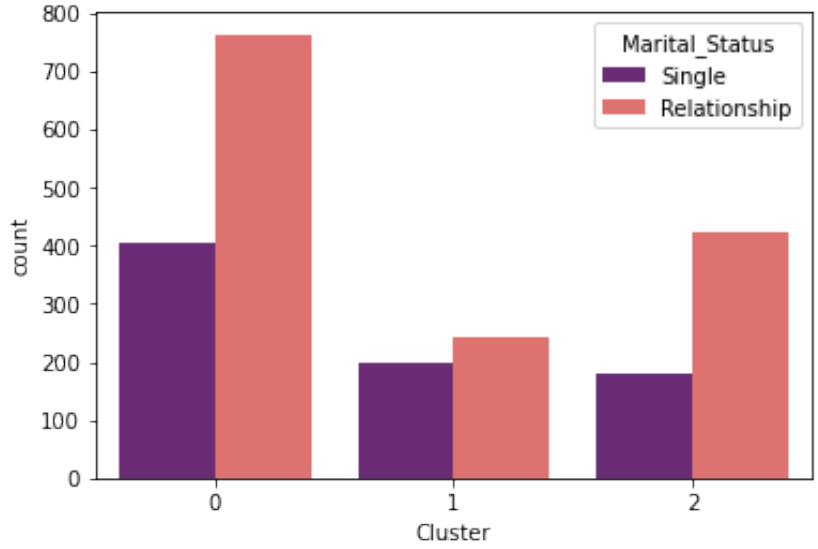


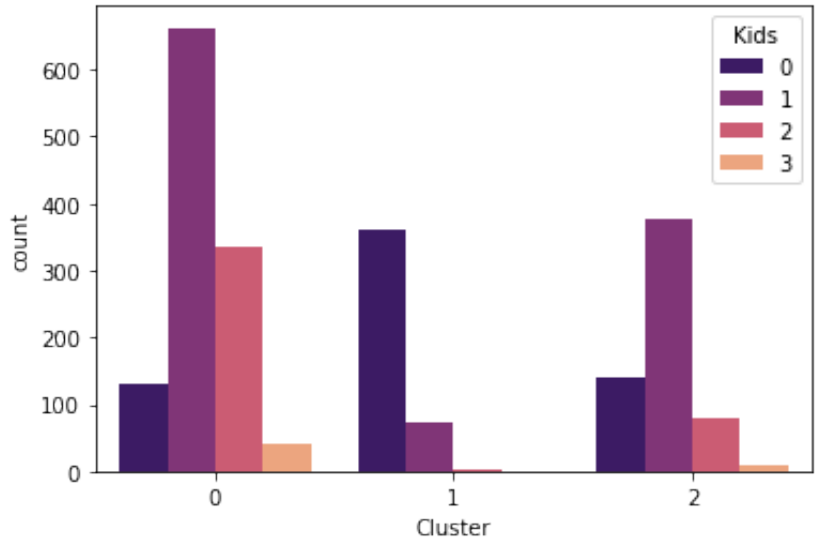
In [198…

```
profile = ['Education', 'Marital_Status', 'Kids']

for i in profile:
    plt.figure()
    sns.countplot(x='Cluster', data=df, hue=df[i],palette = 'mag
    plt.show()
```

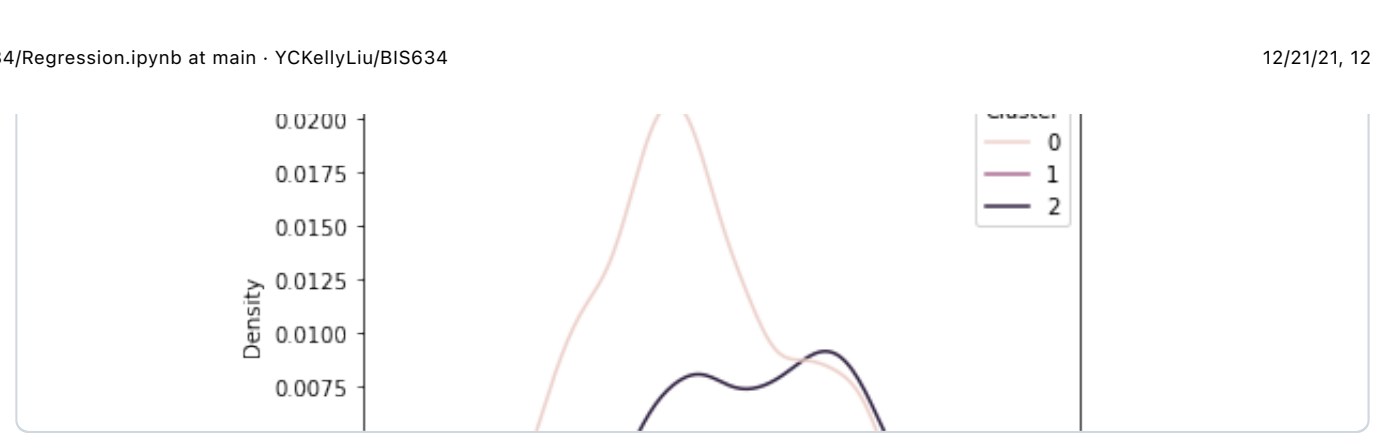






In [199…
```
sns.kdeplot(data=df, x='Age', hue='Cluster')
```

Out[199…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb19919d0a0>
```


0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
Density
0
1
2

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
data = pd.read_csv('./data.csv',index_col=False)
scaler = StandardScaler()
scaler.fit(data)
scaled_df = pd.DataFrame(scaler.transform(data), columns=data.columns)
print(scaled_df.head(5))
```

```
   Education  Marital_Status    Income   Recency  Complain       Age  \
0  -0.158187        1.349603  0.287105  0.310353  -0.09552  1.018352
1  -0.158187        1.349603 -0.260882 -0.380813  -0.09552  1.274785
2  -0.158187       -0.740959  0.913196 -0.795514  -0.09552  0.334530
3  -0.158187       -0.740959 -1.176114 -0.795514  -0.09552 -1.289547
4  -0.158187       -0.740959  0.294307  1.554453  -0.09552 -1.033114

   Engaged_Days      Kids  Expenses  TotalAcceptedCampaign  NumTotalPurch
ases
0      1.973583 -1.264598  1.676245               0.617244           1.31
7945
1     -1.665144  1.404572 -0.963297              -0.502808          -1.15
9273
2     -0.172664 -1.264598  0.280110              -0.502808           0.79
6425
3     -1.923210  0.069987 -0.920135              -0.502808          -0.89
8513
4     -0.822130  0.069987 -0.307562              -0.502808           0.53
5666
```

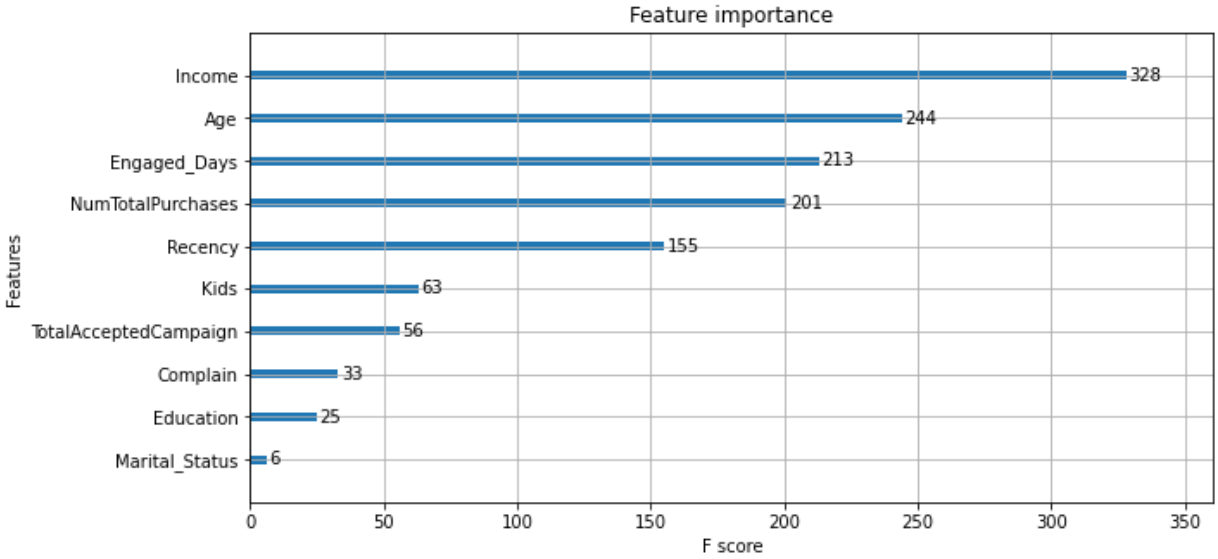
In [4]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
# X, y = data.iloc[:,4:],df.iloc[:,0]
#linear regression:y=ax+b
#https://xgboost.readthedocs.io/en/stable/
X,y = scaled_df[["Education","Marital_Status","Income","Recency","Compla
           "Engaged_Days","Kids","TotalAcceptedCampaign","NumTotalPurcha
print(y.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2212 entries, 0 to 2211
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Expenses  2212 non-null   float64
dtypes: float64(1)
memory usage: 17.4 KB
None
```

In [5]:
```
data_dmatrix = xgb.DMatrix(data=X,label=y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree
                max_depth = 10, alpha = 10, n_estimators = 200)#train 20
xg_reg.fit(X_train,y_train)
preds = xg_reg.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, preds))
```

In [6]:
```
print("RMSE: %f" % (rmse))
plt.rcParams['figure.figsize'] = [10, 5]
xgb.plot_importance(xg_reg)
plt.show()
```

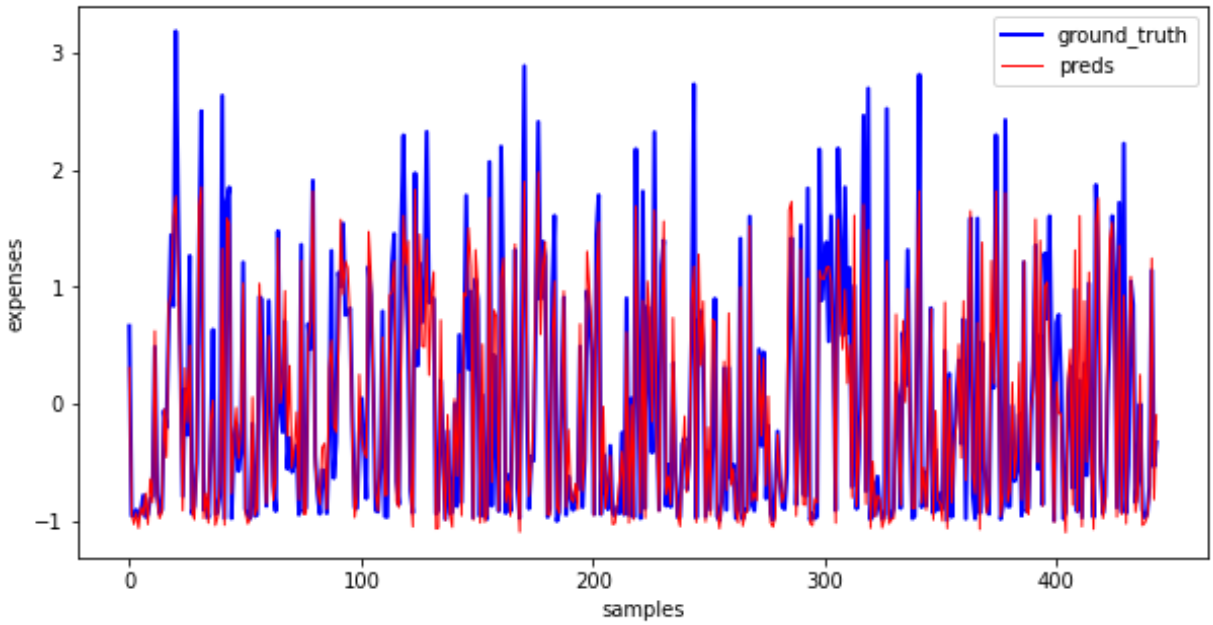
```
RMSE: 0.368855
```



Model Performance

In [8]:
```
x = np.linspace(0,len(preds),len(preds))
plt.plot(x,y_test,color='blue',linewidth="2" )
plt.plot(x,preds,color='red',linewidth="1" )
plt.legend(["ground_truth","preds"])
plt.xlabel("samples")
plt.ylabel("expenses")
```

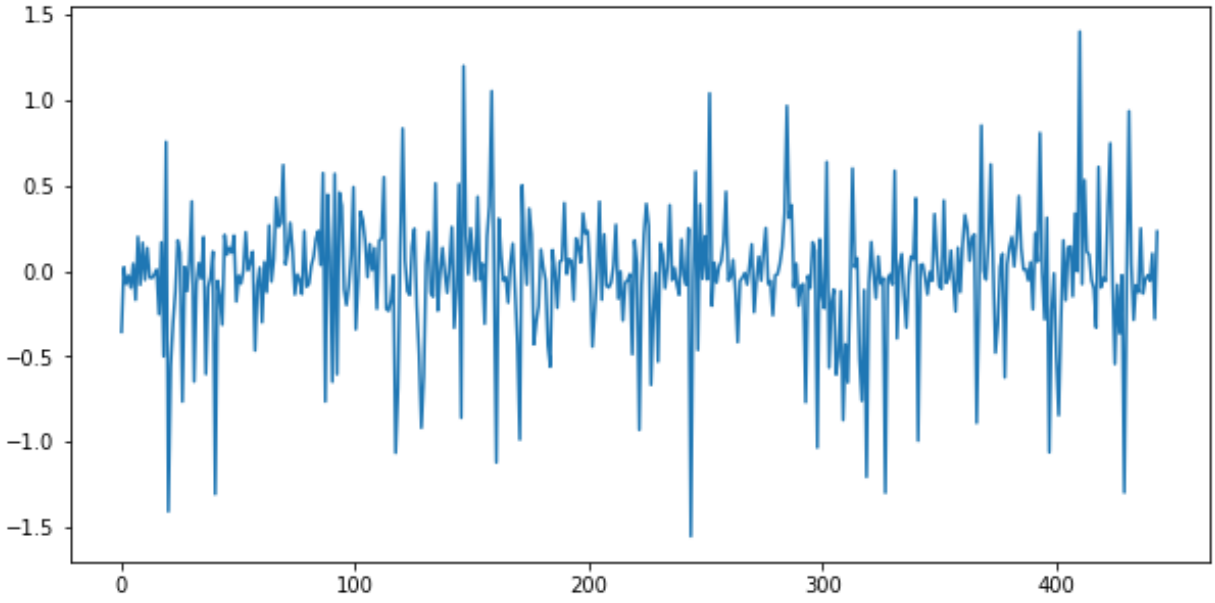
Out[8]:
```
Text(0, 0.5, 'expenses')
```



Model Loss

In [9]:
```
plt.plot(x,preds-y_test["Expenses"])
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x7fc708173910>]
```



In [3]:

```
#-------------------------------------------------------------------------
# Imports
#-------------------------------------------------------------------------
from flask import Flask, render_template, request
from flask import render_template_string, jsonify
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from plotly.utils import PlotlyJSONEncoder
import plotly.graph_objs as go
import logging
from logging import Formatter, FileHandler
import numpy as np
import pandas as pd
import pickle
import json
import os
import seaborn as sns
from datetime import datetime
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def PCA_analysis(data):
    global pca_data
    scaler = StandardScaler()
    scaler.fit(data)
    scaled_df = pd.DataFrame(scaler.transform(data), columns=data.columns
    # The number of dimensions as 3
    pca = PCA(n_components=3)
    pca.fit(scaled_df)
    pca_data = pd.DataFrame(pca.transform(scaled_df), columns=["c1", "c2
    x = pca_data["c1"]
    y = pca_data["c2"]
    z = pca_data["c3"]
    layout = {
    "margin": {
        "l": 200,
        "r": 220,
        "b": 0,
        "t": 0,
    },
    "title":{'text': "3D Plot of Size-Reduced Data",'y':0.9,
            'x':0.5,'xanchor': 'center','yanchor': 'top'},
    }
    data =[{
        "x":x.to_list(),"y":y.to_list(),"z":z.to_list(),"type":"scatter3
        "mode":"markers","marker":{"size":6,"color":x.to_list(),"opacity
    }]
    graphJSON = (jsonify([{"data":data, "layout":layout}]))
    return graphJSON

def Kmeans_analysis(pca_data):
```

```
    wcss = []
    k = range(1,18)
    if True:
        for i in k:
            model = KMeans(n_clusters=i)
            model.fit(pca_data)
            wcss.append(model.inertia_)
        layout = {
        "margin": {
            "l": 50,
            "r": 50,
            "b": 100,
            "t": 100,
            "pad": 4
        },
        "colorway" : ['#f3cec9', '#e7a4b6', '#cd7eaf', '#a262a9', '#6f4d
        "template":"seaborn",
        "title": "The Elbow Method",
        "xaxis":{"title":'Number of Clusters'},
        "yaxis":{"title":"WCSS"}
        }
        data =[{"x": list(k), "y": wcss, "type": "scatter","name":"The E
               }
              ]
        graphJSON = (jsonify([{"data":data, "layout":layout}]))
        return graphJSON
    else:
        print("error!")
        return "error!"

def Kmeans_results(ts,pca_data,k=4):
    ts_temp = ts
    k_means = KMeans(n_clusters = k, random_state = 50)
    y_pred = k_means.fit_predict(pca_data)
#     pca_data_temp['Cluster'] = y_pred
    ts_temp['Cluster'] = y_pred
    layout = {
        "margin": {
            "l": 50,
            "r": 50,
            "b": 100,
            "t": 100,
            "pad": 4
        },
        "colorway" : ['#f3cec9', '#e7a4b6', '#cd7eaf', '#a262a9', '#6f4d
        "template":"seaborn",
        "xaxis":{"title":'Cluster'},
        "yaxis":{"title":"Count"}
        }
    count = ts_temp['Cluster'].value_counts()
    data = [{"x": count.index.tolist(), "y": count.values.tolist(), "type
    graphJSON = (jsonify([{"data":data, "layout":layout}]))
    return graphJSON

def create_plot(ts,feature="Bar"):
    if feature=="Bar":
        x = [str(i)[:10] for i in ts.index.to_list()]
```

```
        layout = {
            "margin": {
                "l": 50,
                "r": 50,
                "b": 100,
                "t": 100,
                "pad": 4
            },
            "colorway" : ['#f3cec9', '#e7a4b6', '#cd7eaf', '#a262a9', '#
            "template":"seaborn"
            }
        data = [
            {"x": x, "y": ts["Education"].to_list(), "type": "bar","name
            {"x": x, "y": ts["Marital_Status"].to_list(), "type": "bar",
            {"x": x, "y": ts["Income"].to_list(), "type": "bar","name":"
            {"x": x, "y": ts["Recency"].to_list(), "type": "bar","name":
            {"x": x, "y": ts["Complain"].to_list(), "type": "bar","name"
            {"x": x, "y": ts["Age"].to_list(), "type": "bar","name":"Age
            {"x": x, "y": ts["Engaged_Days"].to_list(), "type": "bar","na
            {"x": x, "y": ts["Kids"].to_list(), "type": "bar","name":"Kid
            {"x": x, "y": ts["TotalAcceptedCampaign"].to_list(), "type":
            {"x": x, "y": ts["NumTotalPurchases"].to_list(), "type": "bar
            {"x": x, "y": ts["Expenses"].to_list(), "type": "bar","name"
        ]
        graphJSON = (jsonify([{"data":data, "layout":layout}]))
    if feature=="box":
        layout = {
            "margin": {
                "l": 50,
                "r": 50,
                "b": 100,
                "t": 100,
                "pad": 4
            },
            "colorway" : ['#f3cec9', '#e7a4b6', '#cd7eaf', '#a262a9', '#
            "template":"seaborn"
            }
        data = [
            {"y": ts["Education"].to_list(), "type": "box","name":"Educat
            {"y": ts["Marital_Status"].to_list(), "type": "box","name":"M
            {"y": ts["Income"].to_list(), "type": "box","name":"Income"},
            {"y": ts["Recency"].to_list(), "type": "box","name":"Recency
            {"y": ts["Complain"].to_list(), "type": "box","name":"Compla
            {"y": ts["Age"].to_list(), "type": "box","name":"Age"},
            {"y": ts["Engaged_Days"].to_list(), "type": "box","name":"Eng
            {"y": ts["Kids"].to_list(), "type": "box","name":"Kids"},
            {"y": ts["TotalAcceptedCampaign"].to_list(), "type": "box","n
            {"y": ts["NumTotalPurchases"].to_list(), "type": "box","name
            { "y": ts["Expenses"].to_list(), "type": "box","name":"Expens
        ]
        graphJSON = (jsonify([{"data":data, "layout":layout}]))
    if feature=="heatmap":
        layout = {
            "margin": {
                "l": 50,
                "r": 50,
                "b": 100,
```

```
                "t": 100,
                "pad": 40
            },
            "colorway" : ['#f3cec9', '#e7a4b6', '#cd7eaf', '#a262a9', '#
            "template":"seaborn"
            }
        ts = ts.corr()
        data = [{
        "z":[ts["Education"].to_list(),ts["Marital_Status"].to_list(),ts
            ts["Complain"].to_list(),ts["Age"].to_list(),ts["Engaged_Days
             ts["NumTotalPurchases"].to_list(),ts["Expenses"].to_list()]
        "x":["Education","Marital_Status","Income","Recency","Complain",
            "Engaged_Days","Kids","TotalAcceptedCampaign","NumTotalPurcha
        "y":["Education","Marital_Status","Income","Recency","Complain",
            "Engaged_Days","Kids","TotalAcceptedCampaign","NumTotalPurcha
        "square":True,"vmin":-1,"vmax":1,"center":0,"annot":True,"camp":
        }
        ]
        graphJSON = (jsonify([{"data":data}]))
    return graphJSON


sns.set(style="whitegrid")
#-----------------------------------------------------------------------
# App Config.
#-----------------------------------------------------------------------

app = Flask(__name__)

@app.route('/',methods=['GET', 'POST'])
def home():
    return render_template('pages/placeholder.home.html')

@app.route('/about')
def about():
    return render_template('pages/placeholder.about.html')

@app.route('/graphable')
def graphable():
    date = [str(i)[:10] for i in ts.index.to_list()]
    if request.method == 'POST':
        return render_template('pages/placeholder.eda.html',
                               listStatus = date,
                               data = (min(date),max(date)))
    else:
        select_start = request.form.get('date-select-start')
        select_end = request.form.get('date-select-end')
        return render_template('pages/placeholder.eda.html',
                               listStatus = date,
                               data = (select_start,select_end))

@app.route('/Cluster')
def Cluster():
    values = [str(i) for i in range(1,18)]
    return render_template('pages/placeholder.cluster.html',Kvalue=values

@app.route('/Prediction')
```

```
def Prediction():
    return render_template('pages/placeholder.model.html')

@app.route('/bar', methods=['GET', 'POST'])
def change_features():
    graphJSON= create_plot(ts,"Bar")
    return graphJSON

@app.route('/box', methods=['GET', 'POST'])
def box():
    graphJSON= create_plot(ts,"box")
    return graphJSON
@app.route('/heatmap', methods=['GET', 'POST'])
def heatmap():
    graphJSON= create_plot(ts,"heatmap")
    return graphJSON
@app.route('/pca', methods=['GET', 'POST'])
def pca():
    graphJSON= PCA_analysis(ts)
    return graphJSON
@app.route('/kmeans', methods=['GET', 'POST'])
def kmeans():
    global pca_data
    graphJSON= Kmeans_analysis(pca_data)
    return graphJSON
@app.route('/kmeans_results', methods=['GET', 'POST'])
def kmeans_results():
    global pca_data
    k_v = request.args.get('k_value')
    if k_v:
        graphJSON= Kmeans_results(ts,pca_data,k=int(k_v))
        return graphJSON
    else:
        graphJSON= Kmeans_results(ts,pca_data,k=4)
        return graphJSON
@app.route('/predection_result', methods=['GET', 'POST'])
def predection_result():
    Paras = request.args.get('Paras')
    input_x = json.loads(Paras)
    input_x = input_x[0]["data"]
    input_x = list(map(float,input_x))
    input_x = np.array(input_x).reshape(1,10)
    xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_by
                max_depth = 3, alpha = 10, n_estimators = 200)
    loaded_model = pickle.load(open("models/model.dat", "rb"))
    preds = loaded_model.predict(input_x)
    return str(preds[0])
#     global pca_data
#     graphJSON= Kmeans_analysis(pca_data)
#     return graphJSON
#---------------------------------------------------------------------
# Launch.
#---------------------------------------------------------------------

# Default port:
if __name__ == '__main__':
    path = "data/data.csv"
```

```
    ts = pd.read_csv(path,index_col=False)
    pca_data = None
    app.run(host="127.0.0.1",port="5000")
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production d
eployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/font-awesome-4.1.0.
min.css HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/bootstrap-3.1.1.min
.css HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/bootstrap-theme-3.1
.1.min.css HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/layout.main.css HTT
P/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/main.css HTTP/1.1"
200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/main.quickfix.css H
TTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/css/main.responsive.css
HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/js/libs/modernizr-2.8.2
.min.js HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/data_show.png HTTP/
1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/box_plot_yearbirth.
png HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/box_plot_income.png
HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/corr.png HTTP/1.1"
200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/pca.png HTTP/1.1" 2
00 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/kmeans.png HTTP/1.1
" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/fea_imps.png HTTP/1
.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/img/preds.png HTTP/1.1"
200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/js/libs/bootstrap-3.1.1
.min.js HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/js/plugins.js HTTP/1.1"
200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/js/script.js HTTP/1.1"
200 -
127.0.0.1 - - [21/Dec/2021 02:28:06] "GET /static/js/libs/jquery-1.11.1.m
in.js HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:07] "GET /static/ico/favicon.png HTTP/1.
1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:18] "GET /about HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:22] "GET /graphable HTTP/1.1" 200 -
127.0.0.1 - - [21/Dec/2021 02:28:22] "GET /static/js/jquery-1.11.1.min.js
```