```
import pandas as pd
import numpy as np
#reading data
data = pd.read_csv('Caravan.csv')
# dropping first column (of no use)
data.drop('Unnamed: 0', inplace=True, axis=1)
data
```

	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	MGODOV	MGODGE	MRELGE	MRELSA	MRELOV	MFALLEEN	MFGEKIND
0	33	1	3	2	8	0	5	1	3	7	0	2	1	2
1	37	1	2	2	8	1	4	1	4	6	2	2	0	4
2	37	1	2	2	8	0	4	2	4	3	2	4	4	4
3	9	1	3	3	3	2	3	2	4	5	2	2	2	3
4	40	1	4	2	10	1	4	1	4	7	1	2	2	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5817	36	1	1	2	8	0	6	1	2	1	2	6	5	3
5818	35	1	4	4	8	1	4	1	4	6	0	3	2	2
5819	33	1	3	4	8	0	6	0	3	5	1	4	3	3
5820	34	1	3	2	8	0	7	0	2	7	2	0	0	4
5821	33	1	3	3	8	0	6	1	2	7	1	2	1	4

5822 rows × 86 columns

```
#head of data
data.head(5)
```

	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	MGODOV	MGODGE	MRELGE	MRELSA	MREL
0	33	1	3	2	8	0	5	1	3	7	0	
1	37	1	2	2	8	1	4	1	4	6	2	
2	37	1	2	2	8	0	4	2	4	3	2	
3	9	1	3	3	3	2	3	2	4	5	2	
4	40	1	4	2	10	1	4	1	4	7	1	

5 rows × 86 columns

Let's see the Target Variable distribution:

```
cc = data['Purchase'].value_counts()
print(cc)
print("Purchased Percentage:")
print(cc[1]/(cc[1]+cc[0])*100)
```

```
No     5474
Yes     348
Name: Purchase, dtype: int64
Purchased Percentage:
5.977327378907591
```

Higly Imabalanced Dataset:

We can see that The number of Purchased records in dataset is very low ~ 6 %

```
X = data.iloc[:, :-1].values

#Separating the last column i.e. "Purchase" [Target Variable]
y = data.iloc[:,-1].values
```

```
# Since it is in Yes/No format, changing to 1/0
for i in range(0,data.shape[0]):
    #print(data.loc[i].at["Purchase"])
    if(data.loc[i].at["Purchase"] == "No"):
        y[i] = 0
    else:
        y[i] = 1

#Converting the type
y = y.astype('int')
y
```

```
array([0, 0, 0, ..., 1, 0, 0])
```

```
#Data Summary
data.describe()
```

	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	MGC
count	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.00C
mean	24.253349	1.110615	2.678805	2.991240	5.773617	0.696496	4.626932	1.069
std	12.846706	0.405842	0.789835	0.814589	2.856760	1.003234	1.715843	1.017
min	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.00C
25%	10.000000	1.000000	2.000000	2.000000	3.000000	0.000000	4.000000	0.00C
50%	30.000000	1.000000	3.000000	3.000000	7.000000	0.000000	5.000000	1.00C
75%	35.000000	1.000000	3.000000	3.000000	8.000000	1.000000	6.000000	2.00C
max	41.000000	10.000000	5.000000	6.000000	10.000000	9.000000	9.000000	5.00C

8 rows × 85 columns

Observation:

MOSTYPE i.e. Customer Sub-Type : Mean is around 24-25 [*young, low educated and young seniors*]

MGEMOMV i.e. Avg. Household Size: It is 2.6

MKOOPKLA i.e. Purchasing Power Class: 37-49% are in this class

```
###Try to find the row names with NaN values.
data.isnull().sum(axis=0)
```

```
MOSTYPE     0
MAANTHUI    0
MGEMOMV     0
MGEMLEEF    0
MOSHOOFD    0
           ..
APLEZIER    0
AFIETS      0
AINBOED     0
ABYSTAND    0
Purchase    0
Length: 86, dtype: int64
```

There is no NULL or nan Value in dataset.

Correlations:

```
# Correlations Matrix
data.corr()
```

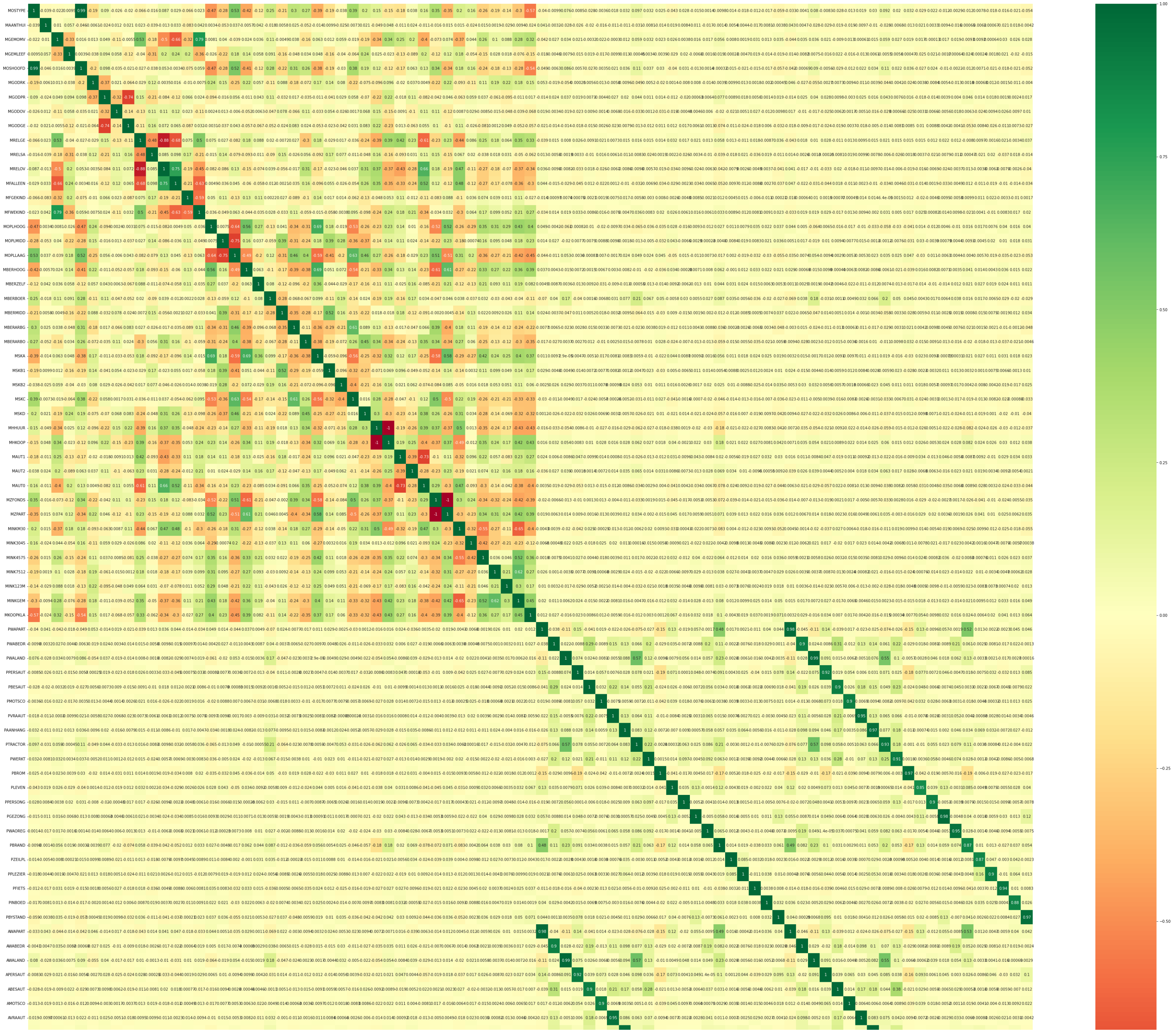
	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	MGODOV	MGODGE
MOSTYPE	1.000000	-0.038721	-0.021997	0.009454	0.992672	-0.193613	0.090399	-0.025642	-0.019505
MAANTHUI	-0.038721	1.000000	0.010102	0.056975	-0.045817	-0.006136	-0.024360	0.012056	0.020540
MGEMOMV	-0.021997	0.010102	1.000000	-0.328257	0.016115	0.013105	0.049356	-0.108650	-0.005527
MGEMLEEF	0.009454	0.056975	-0.328257	1.000000	0.003872	-0.037519	0.093654	0.057737	-0.119966
MOSHOOFD	0.992672	-0.045817	0.016115	0.003872	1.000000	-0.199186	0.098493	-0.034566	-0.021466
...	...	...	...	...	...	...	...	...	..
AZEILPL	0.007801	-0.006189	0.009234	0.000244	0.007099	-0.000675	0.013760	-0.023877	-0.008412
APLEZIER	-0.018162	0.000666	0.000644	-0.001791	-0.020683	0.011795	0.018468	0.009417	-0.026407
AFIETS	-0.015774	-0.020993	0.030330	0.020612	-0.017990	-0.001503	0.001906	0.025661	-0.011122
AINBOED	-0.021087	0.018304	0.025907	-0.020042	-0.020997	-0.011431	0.002392	-0.009734	0.007261
ABYSTAND	-0.053718	-0.004166	0.028384	-0.014540	-0.051723	-0.004009	0.016658	0.010127	-0.027291

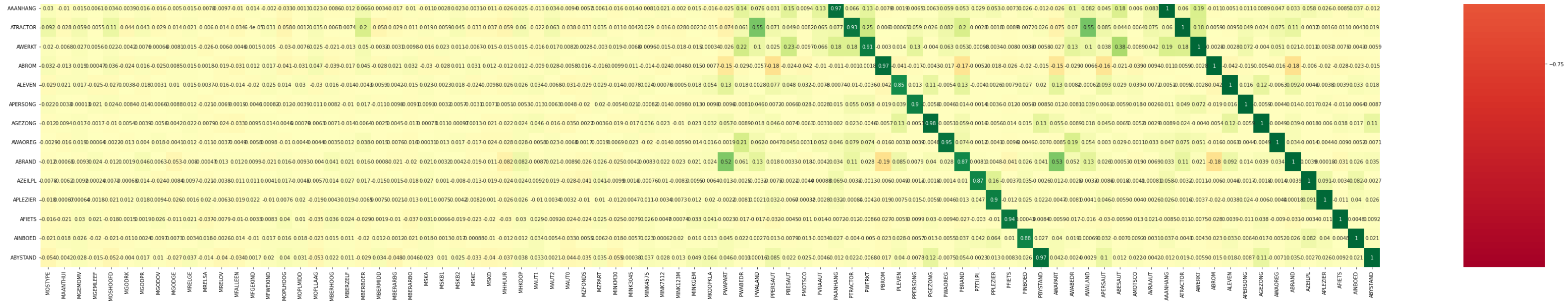
85 rows × 85 columns

```
#Correlation heatmap
import seaborn as sns
import matplotlib.pyplot as plt
#sns.heatmap(data.corr())

corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(60,60))
#plot heat map
sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd07e3178d0>
```




AAANHANG
ATRACTOR
AWERKT
ABROM
ALEVEN
APERSONG
AGEZONG
AWAOREG
ABRAND
AZEILPL
APLEZIER
AFIETS
AINBOED
ABYSTAND

Getting top Correlations

```
# pandas dataframe
df = pd.DataFrame(data = data)

#Getting the most absolute Correlations
def most_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    au_corr = au_corr.sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations")
arr = most_abs_correlations(data, 300)
print(arr)
```

```
Top Absolute Correlations
ABYSTAND  ABYSTAND    1.000000
PINBOED   PINBOED     1.000000
PPLEZIER  PPLEZIER    1.000000
PZEILPL   PZEILPL     1.000000
PBRAND    PBRAND      1.000000
                        ...
MRELGE    MAUT1       0.416807
MAUT1     MRELGE      0.416807
MOSHOOFD  MBERHOOG    0.412656
MBERHOOG  MOSHOOFD    0.412656
MSKB1     MOPLLAAG    0.406194
Length: 300, dtype: float64
```

```
#saving the Top 300 Correlations to csv to analyze
arr.to_csv("top_corr.csv")
```

From the CSV, WE can see that there is a correlation between:

MHHUUR [*Rented House*] vs MHKOOP [*Home owners*]

MOSTYPE [*Customer Sub Type*] and MOSHOOFD [*Customer Main Type*]

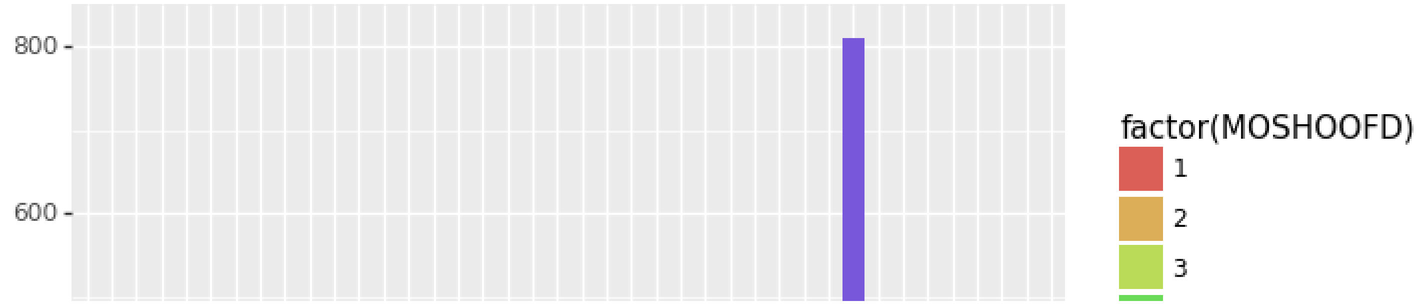
Avg. Household vs Singles and many more

Based on above correlations and taking into consideration some imporatant metrics such as Avg. income, Purchasing power etc, Let's explore their relationships with each other and target variable using ggplot

```
!pip install plotnine
from plotnine import ggplot, aes, geom_boxplot, geom_bar

# Customer Main type and Customer SubType
(
  ggplot(data)
  + aes(x="factor(MOSTYPE)", fill="factor(MOSHOOFD)")
  + geom_bar()
)
```

```
/usr/local/lib/python3.7/dist-packages/plotnine/utils.py:1246: FutureWarning: is_categorical is depreca
  if pdtypes.is_categorical(arr):
```



We can see that:

Customer Sub-Type 1−5 represents Customer Main Type 1

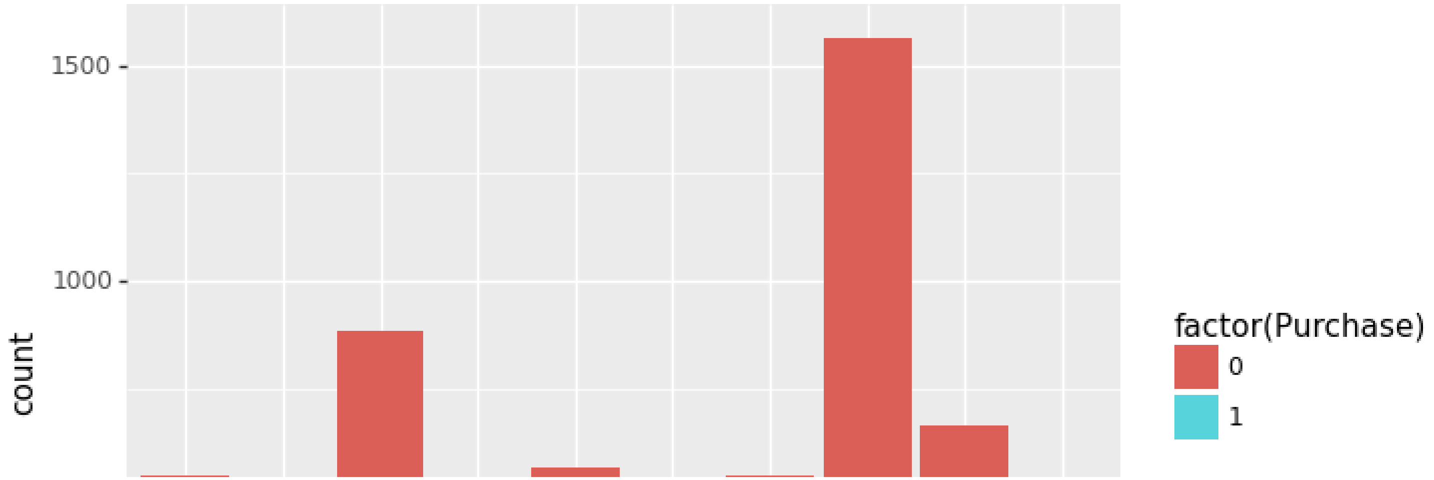
Customer Sub-Type 6−8 represents Customer Main Type 2

Customer Sub-Type 9−13 represents Customer Main Type 3

And So on...

```
# Customer Policies vs Customer Main type
(
  ggplot(data)
  + aes(x="factor(MOSHOOFD)", fill="factor(Purchase)")
  + geom_bar()
)
```

```
/usr/local/lib/python3.7/dist-packages/plotnine/utils.py:1246: FutureWarning: is_categorical is depreca
  if pdtypes.is_categorical(arr):
```



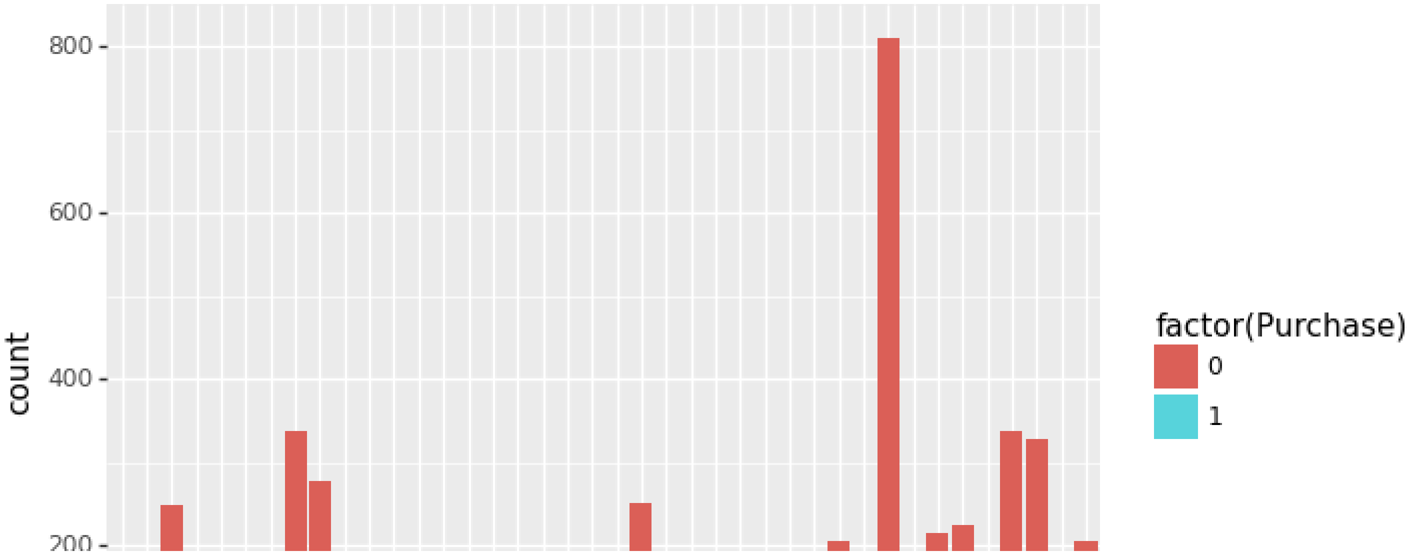
We can see that:

Maximum Purchase is in Customer Main type 8 [*Family with grown ups*]

Customer main type 4 (Career Loners) : 0

```
# Customer Policies vs Customer SubType
(
  ggplot(data)
  + aes(x="factor(MOSTYPE)", fill="factor(Purchase)")
  + geom_bar()
)
```

```
/usr/local/lib/python3.7/dist-packages/plotnine/utils.py:1246: FutureWarning: is_categorical is depreca
  if pdtypes.is_categorical(arr):
```



We see that:

Most Purchases : sub type 8 [Middle class families] and type 33 [Lower class large families].

Also Folowing subtypes have 0 purchases: 14 15 16 17 18 19 21 28 40

```
# Customer Policies vs Purchasing power

(
  ggplot(data)
  + aes(x="factor(MKOOPKLA)", fill="factor(Purchase)")
  + geom_bar()
)
```

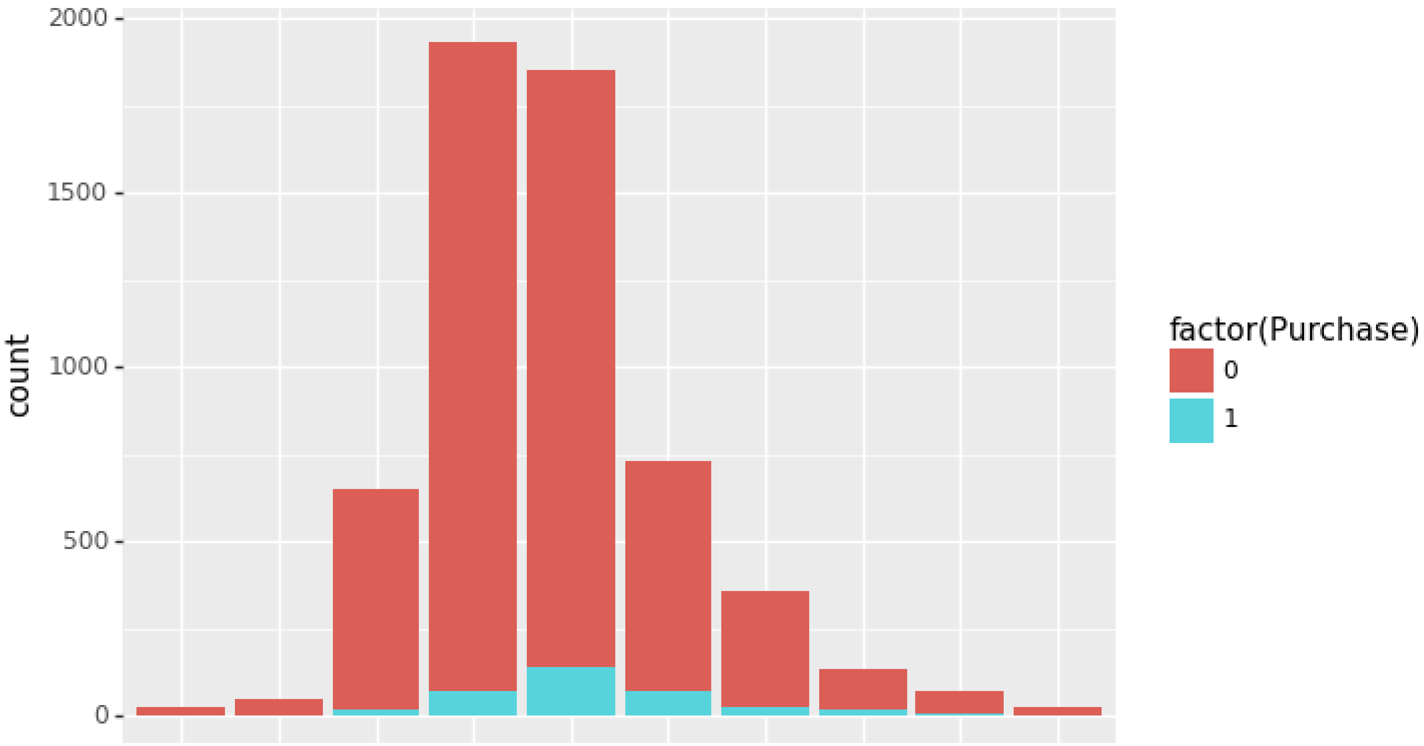
```
/usr/local/lib/python3.7/dist-packages/plotnine/utils.py:1246: FutureWarning: is_categorical is depreca
  if pdtypes.is_categorical(arr):
```

Most of the Purchases are in Medium to high Purchasing power class.

```
# Customer Policies vs AVg.Income

(
  ggplot(data)
  + aes(x="factor(MINKGEM)", fill="factor(Purchase)")
  + geom_bar()
)
```

```
/usr/local/lib/python3.7/dist-packages/plotnine/utils.py:1246: FutureWarning: is_categorical is depreca
  if pdtypes.is_categorical(arr):
```



Now converting the Customer main type and subtype columns to one hot encoding i.e. as they are categorical in original dataset

```
# one hot encoding
one_hot_encoded_data = pd.get_dummies(data, columns = ['MOSHOOFD', 'MOSTYPE'])
print(one_hot_encoded_data)
X = one_hot_encoded_data.iloc[:, :].values
X.shape

# Removing the Purchase column
index_no = one_hot_encoded_data.columns.get_loc('Purchase')
print(index_no)
X = np.delete(X, index_no, 1)  # delete "index_no"th column of X
X.shape
```

```
      MAANTHUI  MGEMOMV  MGEMLEEF  ...  MOSTYPE_39  MOSTYPE_40  MOSTYPE_41
0            1        3         2  ...           0           0           0
1            1        2         2  ...           0           0           0
2            1        2         2  ...           0           0           0
3            1        3         3  ...           0           0           0
4            1        4         2  ...           0           1           0
...        ...      ...       ...  ...         ...         ...         ...
5817         1        1         2  ...           0           0           0
5818         1        4         4  ...           0           0           0
5819         1        3         4  ...           0           0           0
5820         1        3         2  ...           0           0           0
5821         1        3         3  ...           0           0           0

[5822 rows x 134 columns]
83
(5822, 133)
```

```
import joblib, sys
sys.modules['sklearn.externals.joblib'] = joblib
```

Feature Selection:

Since there are total 134 columns, It would be infeasible to feed them into Logistic Regression or LDA.

Let's do the feature selction using Forward Feature selction and Separately Dimensionality reduction using PCA. And Compare the observations.

```
#For forward Feature Selecton
!pip install mlxtend
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.linear_model import LogisticRegression
```

```
lreg = LogisticRegression(max_iter = 10000)
#forward = True implies the Forward Feature selction
```

```
#orward  = false -> Backward Feature Selection
#k_features = No. of features to be selected
sfs1 = sfs(lreg, k_features=20, forward=True, verbose=2, scoring='neg_mean_squared_error')


sfs1 = sfs1.fit(X, y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 133 out of 133 | elapsed:    6.3s finished

[2021-12-03 19:54:40] Features: 1/20 -- score: -0.05977301889296934[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 132 out of 132 | elapsed:   12.1s finished

[2021-12-03 19:54:52] Features: 2/20 -- score: -0.059601345073226855[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 131 out of 131 | elapsed:   14.7s finished

[2021-12-03 19:55:07] Features: 3/20 -- score: -0.059601345073226855[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 130 out of 130 | elapsed:   19.8s finished

[2021-12-03 19:55:27] Features: 4/20 -- score: -0.059601345073226855[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 129 out of 129 | elapsed:   24.4s finished

[2021-12-03 19:55:51] Features: 5/20 -- score: -0.059601345073226855[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 128 out of 128 | elapsed:   27.6s finished

[2021-12-03 19:56:19] Features: 6/20 -- score: -0.059601345073226855[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 127 out of 127 | elapsed:   29.9s finished

[2021-12-03 19:56:49] Features: 7/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 126 out of 126 | elapsed:   36.2s finished

[2021-12-03 19:57:25] Features: 8/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done 125 out of 125 | elapsed:   42.0s finished

[2021-12-03 19:58:07] Features: 9/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend with
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 124 out of 124 | elapsed:   44.5s finished

[2021-12-03 19:58:51] Features: 10/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.4s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 123 out of 123 | elapsed:   45.1s finished

[2021-12-03 19:59:36] Features: 11/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 122 out of 122 | elapsed:   58.6s finished

[2021-12-03 20:00:35] Features: 12/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 121 out of 121 | elapsed:  1.1min finished

[2021-12-03 20:01:39] Features: 13/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.7s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 120 out of 120 | elapsed:  1.3min finished

[2021-12-03 20:02:55] Features: 14/20 -- score: -0.05942952376738493[Parallel(n_jobs=1)]: Using backend SequentialBackend wit
```

```
#Listing Selected Features by Forward Feature Selection
feat_names = list(sfs1.k_feature_names_)
print("Selected Features are:")
print(feat_names)
#Transforming the X to only keep the selected features
Xnew = sfs1.transform(X)
```

```
Selected Features are:
['0', '1', '2', '3', '4', '5', '6', '8', '10', '11', '12', '13', '16', '17', '20', '22', '23', '25', '44', '79']
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
Xnew = sc.fit_transform(Xnew)
```

```
#Splitting the data set into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(Xnew, y, test_size=0.25, random_state=2)

#Logistic regression
from sklearn.linear_model import LogisticRegression
clft = LogisticRegression(max_iter=1000)
clft.fit(X_train, y_train)
y_pred = clft.predict(X_test)

#Importing Accuracy and Confusion Matrix metrics
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix for Logistic Regression:')
print(cm)
acc_LR = accuracy_score(y_test, y_pred)
print('Accuracy for Logistic Regression')
print(acc_LR)


# LDA
print("\nLDA")
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
cl1ft = LDA()
cl1ft.fit(X_train, y_train)
y_pred = cl1ft.predict(X_test)
# print(y_pred)
# y_test
cm1 = confusion_matrix(y_test, y_pred)
print('Confusion Matrix for LDA')
print(cm1)
acc_LDA = accuracy_score(y_test, y_pred)
print('Accuracy for LDA')
print(acc_LDA)
```

```
Confusion Matrix for Logistic Regression:
```

```
[[1369    1]
 [  85    1]]
Accuracy for Logistic Regression
0.9409340659340659

LDA
Confusion Matrix for LDA
[[1366    4]
 [  84    2]]
Accuracy for LDA
0.9395604395604396
```

From the Confusion Matrix, We can see that the for both Logistic and LDA model is heavily biased towards the "No" Purchase class and the reason being the highly Imbalanced training Dataset.

The accuracy is almost similar for both model ~ 94% as both are classifying mostly into "No" purchase class only but the LDA performed slightly better in Confusion matrix and has comparatively less bias.

PCA based Dimesionality Reduction:

```
#PCA based dimentionality reduction
from sklearn.decomposition import PCA

#40 Components
pca = PCA(n_components=40)
Xnew = pca.fit_transform(X)


#Splitting the data set into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(Xnew, y, test_size=0.25, random_state=2)

#Logistic regression
from sklearn.linear_model import LogisticRegression
clpca = LogisticRegression(max_iter=1000)
```

```
clpca.fit(X_train, y_train)
y_pred = clpca.predict(X_test)


from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix for Logistic Regression')
print(cm)
acc_LR = accuracy_score(y_test, y_pred)
print('Accuracy for Logistic Regression')
print(acc_LR)


# LDA
print("\nLDA")
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
cl1pca = LDA()
cl1pca.fit(X_train, y_train)
y_pred = cl1pca.predict(X_test)
# print(y_pred)
# y_test
cm1 = confusion_matrix(y_test, y_pred)
print('Confusion Matrix for LDA')
print(cm1)
acc_LDA = accuracy_score(y_test, y_pred)
print('Accuracy for LDA')
print(acc_LDA)
```

```
Confusion Matrix for Logistic Regression
[[1368    2]
 [  86    0]]
Accuracy for Logistic Regression
0.9395604395604396

LDA
Confusion Matrix for LDA
[[1366    4]
 [  85    1]]
```

```
Accuracy for LDA
0.9388736263736264
```

The observation is similar to the Forward Feature Selection based Observation.

```
#End
```