Practical Work 1 - Design and Optimization of Machine Learning Models

Machine Learning Data (DAA)

Pg52676 - Catarina Costa
Pg54470 - Fernando Alves
Pg52694 - Marta Aguiar

TABLE OF CONTENTS

DATASET

Costumer Personality Analysis

- Customer Persona Understanding
- Targeted Product Modification
- Marketing Strategies Optimizations

DATASET ATTRIBUTES

People

- ID
- Year_Birth
- Education
- Marital_Status
- Income
- Kidhome
- Teenhome
- Dt_Custumer
- Recency
- Complain

Products

- MntWines
- MntFruits
- MntMeatProducts
- MntFishProducts
- MntSweetProducts
- MntGoldProds

Promotion

- NumDealsPurchases
- AcceptedComp1
- AcceptedComp2
- AcceptedComp3
- AcceptedComp4
- AcceptedComp5
- Response

Place

- NumWebPurchases
- NumCatalogPurchases
- NumStorePurchases
- NumWebVisitsMonth

DATA ANALYSING

We initially conducted a general analysis of the dataset

data.columns

```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Response'],
      dtype='object')
```

data.tail()

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWeb
2235	10870	1967	Graduation	Married	61223.0	0	1	13-06-2013	46	709	...	
2236	4001	1946	PhD	Together	64014.0	2	1	10-06-2014	56	406	...	
2237	7270	1981	Graduation	Divorced	56981.0	0	0	25-01-2014	91	908	...	
2238	8235	1956	Master	Together	69245.0	0	1	24-01-2014	8	428	...	
2239	9405	1954	PhD	Married	52869.0	1	1	15-10-2012	40	84	...	

5 rows × 29 columns

data.info()

```
<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
```

data.head()

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVisits
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58	635	...	
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38	11	...	
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	...	
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26	11	...	
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	...	

5 rows × 29 columns

data.shape

```
data.shape
```

```
(2240, 29)
```

DATA PROCESSING

1) Data conversion to correct dtypes:

- Convert the variable *Dt_Customer* to datetime64[ns] dtype

```
7   Dt_Customer            2240 non-null    object
```

```
data['Dt_Customer'] = pd.to_datetime(data['Dt_Customer'], format='%d-%m-%Y')
```

```
7   Dt Customer            2240 non-null    datetime64[ns]
```

2) Feature Engineering:

- Creation of the variables:
  - *Age*: Costumer's age in 2023
  - *Kids*: Sum of the Teenhome and Kidhome variables -> Represents the total number of children
  - *Spent:* Total spending on various items
  - Is_*Parent:* Binary variable

DATA PROCESSING

3) Segmenting data in groups:

- Marital_Status:
  - We regrouped the categorical variable 'Married' in two diferent groups: 'Single' and 'Together'
- Education:
  - We regrouped the categorical variable 'Education' in three diferent groups: 'Undergraduate', 'Graduate' and 'Postgraduate'

4) Rename Variables:

```
data=data.rename(columns={"MntWines": "Wines","MntFruits":"Fruits","MntMeatProducts":"Meat",
                          "MntFishProducts":"Fish","MntSweetProducts":"Sweets","MntGoldProds":"Gold",
                          "NumDealsPurchases": "DealsPurch", "NumWebPurchases": "WebPurch",
                          "NumCatalogPurchases" : "CatalogPurch", "NumStorePurchases": "StorePurch",
                          "NumWebVisitsMonth": "WebVisits"})
```

DATA PROCESSING

5) Drop data:

- We decided to drop some unused data for better organization:

```
to_drop = ["Year_Birth","Z_CostContact", "Z_Revenue",
           "Kidhome","Teenhome", "AcceptedCmp1",
           "AcceptedCmp2","AcceptedCmp3","AcceptedCmp4",
           "AcceptedCmp5", "Complain", "Response"]
```

DATA CLEANING

Age:

data.describe()

p4	AcceptedCmp5	AcceptedCmp1	AcceptedCmp2	Complain	Response	Age	Kids	Is_Parent	Spent
00	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000
54	0.072768	0.064286	0.013393	0.009375	0.149107	54.194196	0.950446	0.715179	605.798214
00	0.000000	0.000000	0.000000	0.000000	0.000000	27.000000	0.000000	0.000000	5.000000
00	0.000000	0.000000	0.000000	0.000000	0.000000	46.000000	0.000000	0.000000	68.750000
00	0.000000	0.000000	0.000000	0.000000	0.000000	53.000000	1.000000	1.000000	396.000000
00	0.000000	0.000000	0.000000	0.000000	0.000000	64.000000	1.000000	1.000000	1045.500000
00	1.000000	1.000000	1.000000	1.000000	1.000000	130.000000	3.000000	1.000000	2525.000000
28	0.259813	0.245316	0.114976	0.096391	0.356274	11.984069	0.751803	0.451430	602.249288

data["Age"].value_counts()

```
124      1
82       1
130      1
123      1
83       1
Name: count, dtype: int64
```

data_clean Age >= 100

Since there is a total of three people with the age above 100 years, we decided to drop all the data that had a "Age" bigger than 100.

```
data_clean = data.drop(data.loc[data['Age']>100].index, inplace = True)
print(data_clean)
```

DATA CLEANING

data.nunique

```
data.nunique()
```

```
ID                2237
Education            3
Marital_Status       2
Income            1971
Dt_Customer        663
Recency            100
Wines              775
Fruits             158
Meat               557
Fish               182
Sweets             177
Gold               213
DealsPurch          15
WebPurch            15
CatalogPurch        14
StorePurch          14
WebVisits           16
Age                 56
Kids                 4
Is_Parent            2
Spent             1054
dtype: int64
```

data.isna().null()

```
data.isna().any()
```

```
ID                False
Education         False
Marital_Status    False
Income             True
Dt_Customer       False
Recency           False
Wines             False
Fruits            False
Meat              False
Fish              False
Sweets            False
Gold              False
DealsPurch        False
WebPurch          False
CatalogPurch      False
StorePurch        False
WebVisits         False
Age               False
Kids              False
Is_Parent         False
Spent             False
dtype: bool
```

data.isnull().sum()

```
print("Total de valores nulos ")
print(data.isnull().sum())
```

```
Total de valores nulos
ID                 0
Education          0
Marital_Status     0
Income            24
Dt_Customer        0
Recency            0
Wines              0
Fruits             0
Meat               0
Fish               0
Sweets             0
Gold               0
DealsPurch         0
WebPurch           0
CatalogPurch       0
StorePurch         0
WebVisits          0
Age                0
Kids               0
Is_Parent          0
Spent              0
dtype: int64
```

data.duplicated().sum()

Check if the dataset has duplicate values

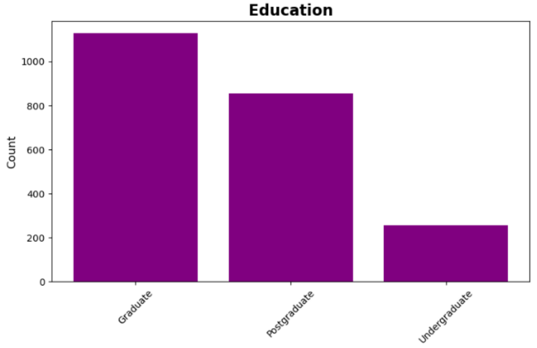
```
data.duplicated().sum()
```

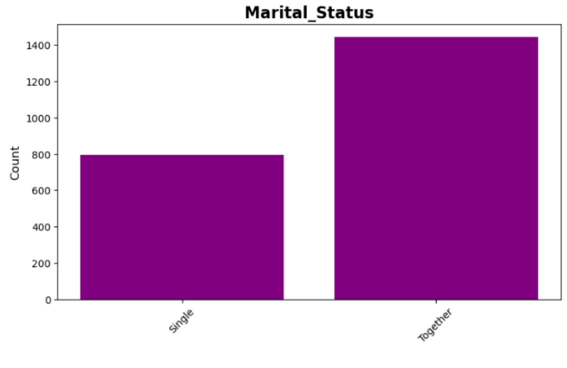
```
0
```

DATA EXPLORATION

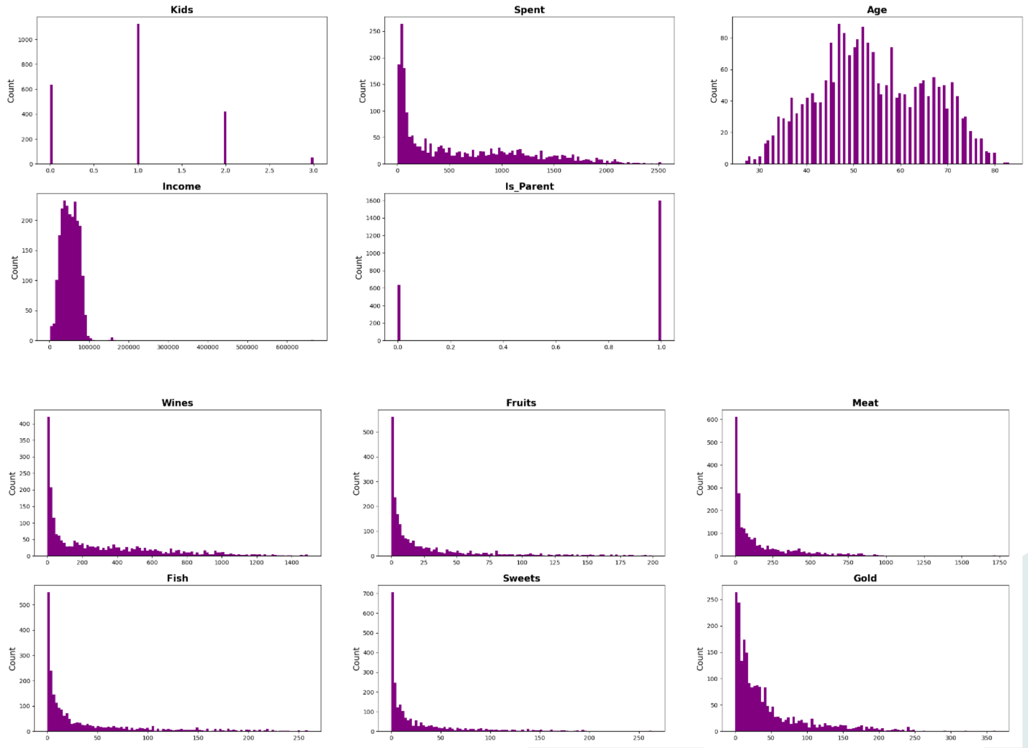
Visual Analysis: Distribution

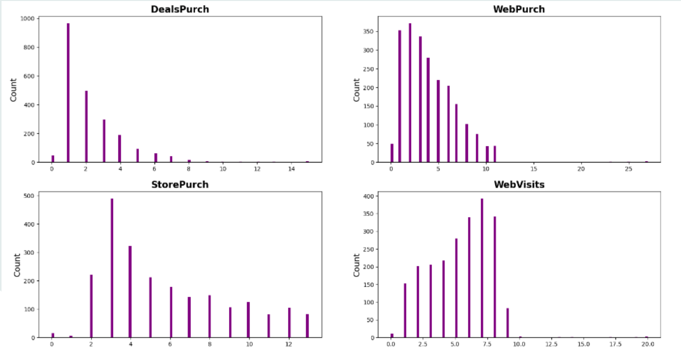
Categorical

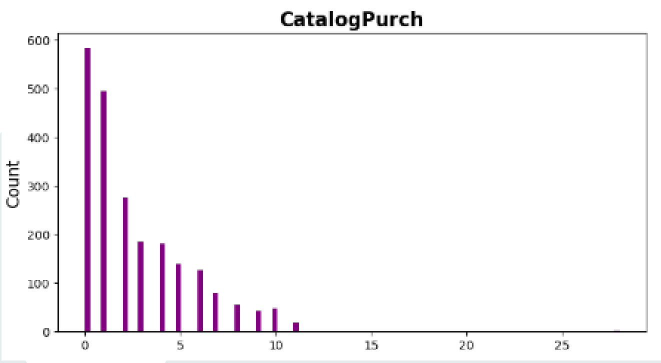




Numerical

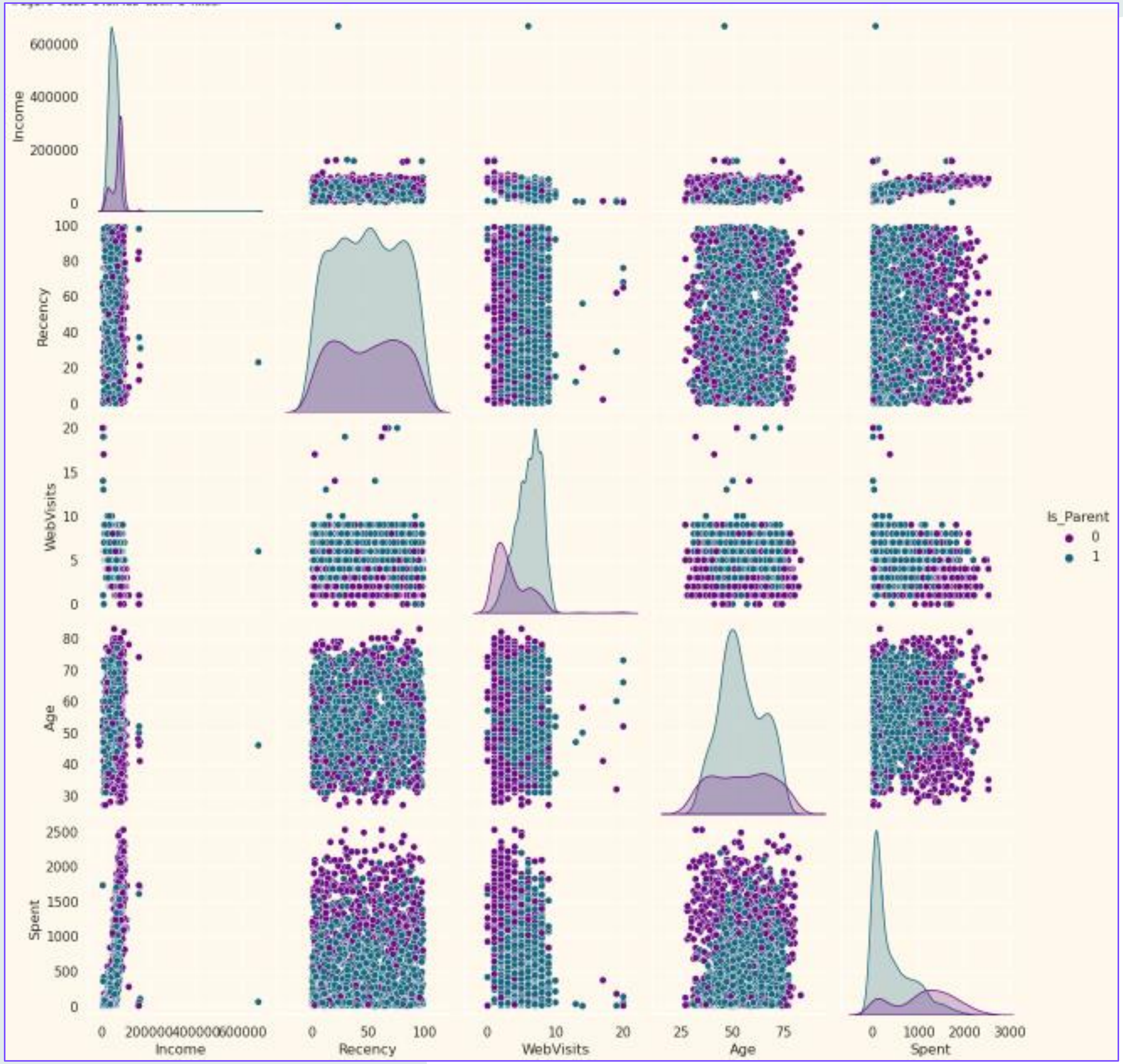






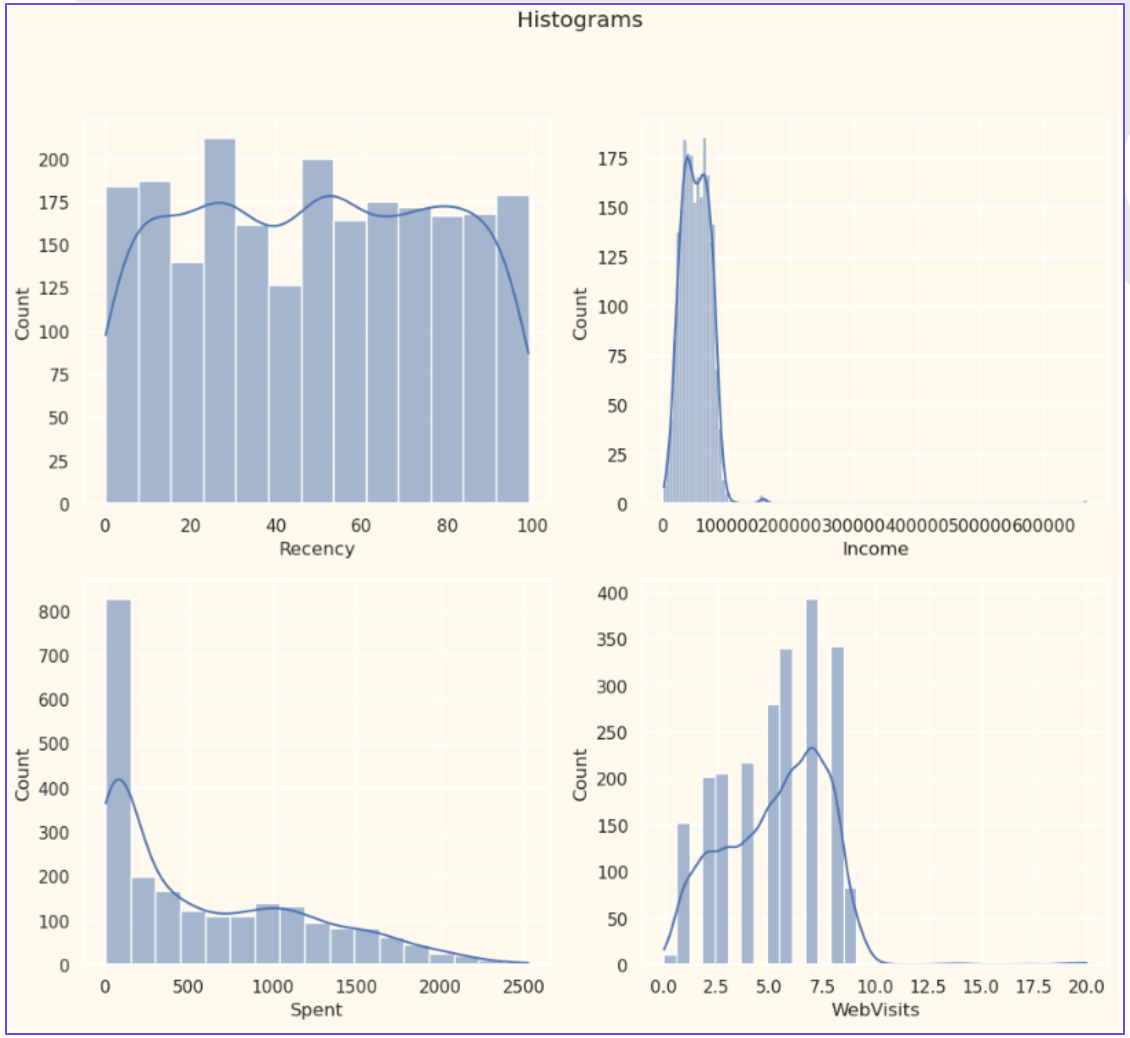
DATA EXPLORATION

Visual Analysis: Multivariable Analysis



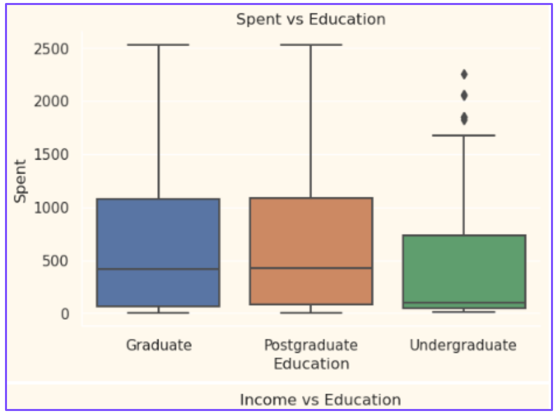
DATA EXPLORATION

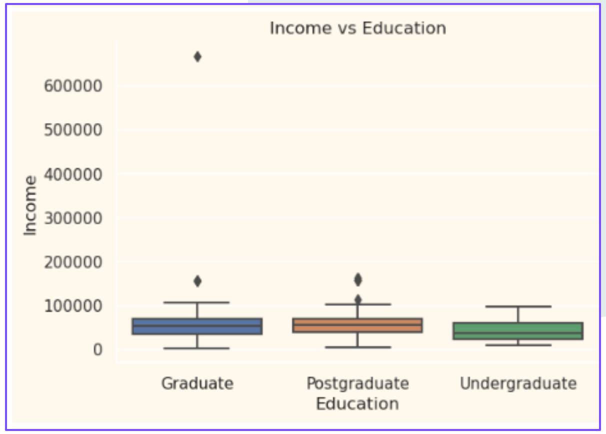
Dispersion Graphs:

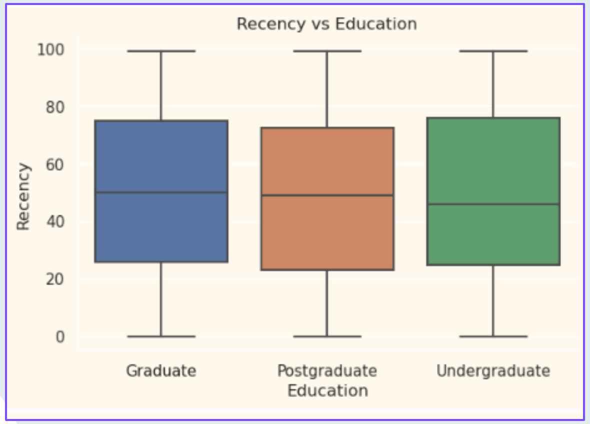


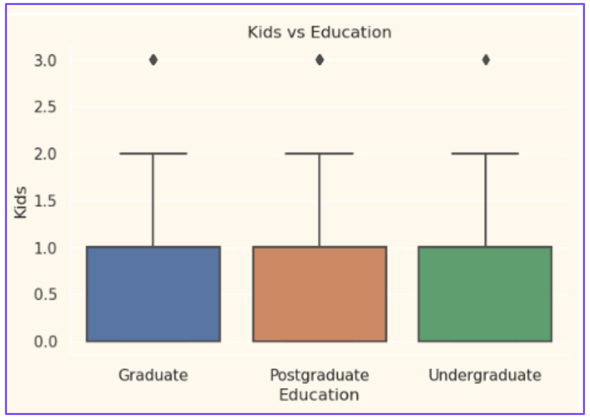
DATA EXPLORATION

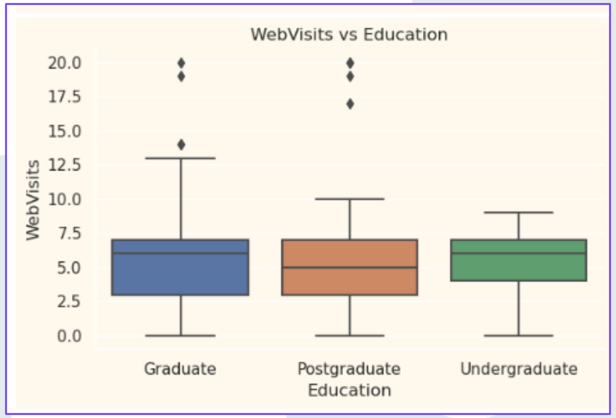
Numerical vs Categorical variables: *Education*





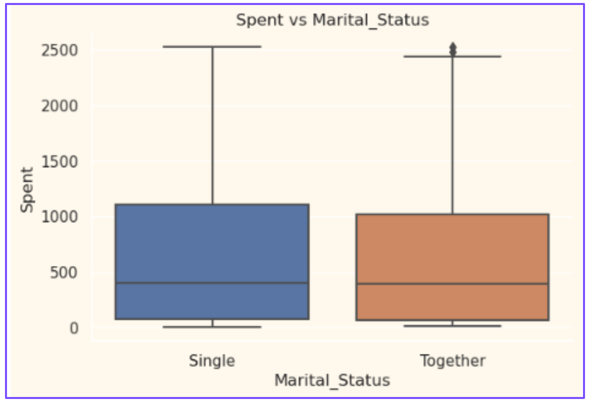


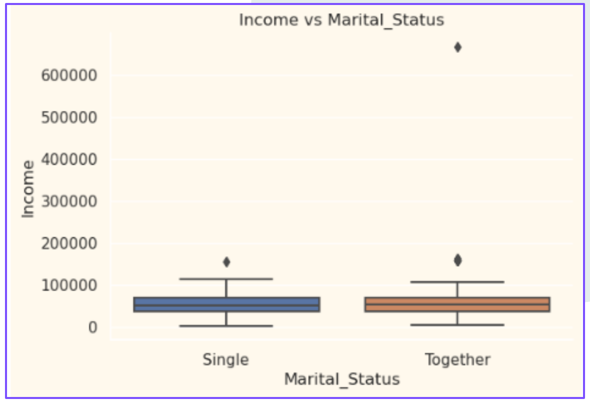


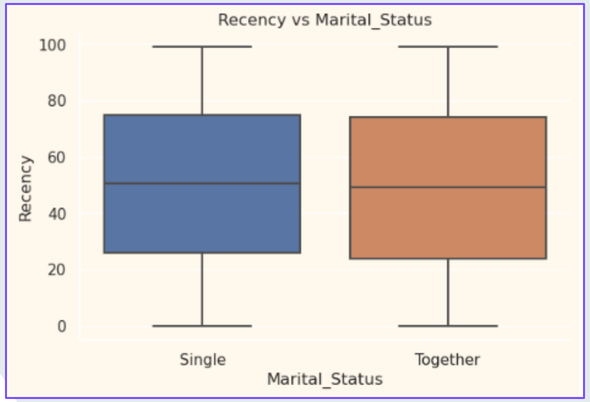


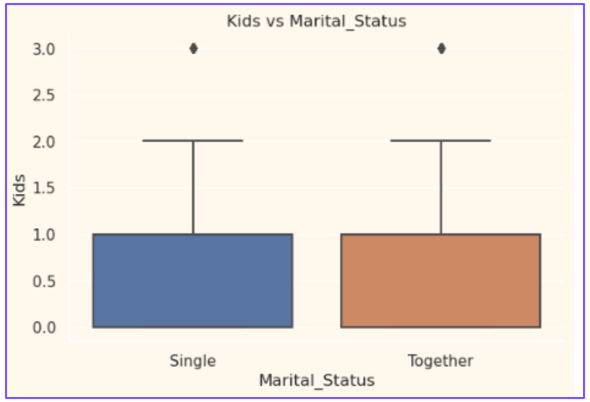
DATA EXPLORATION

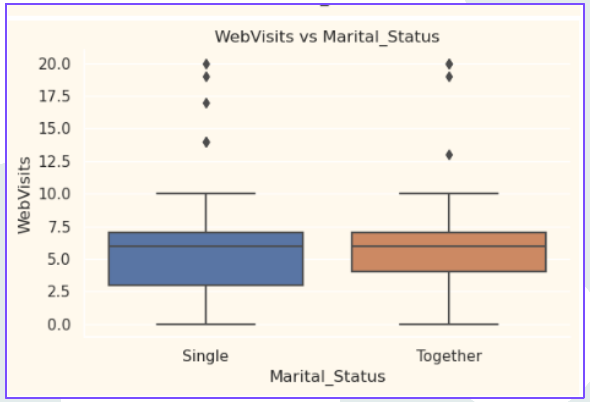
Numerical vs Categorical variables: *Marital_Status*





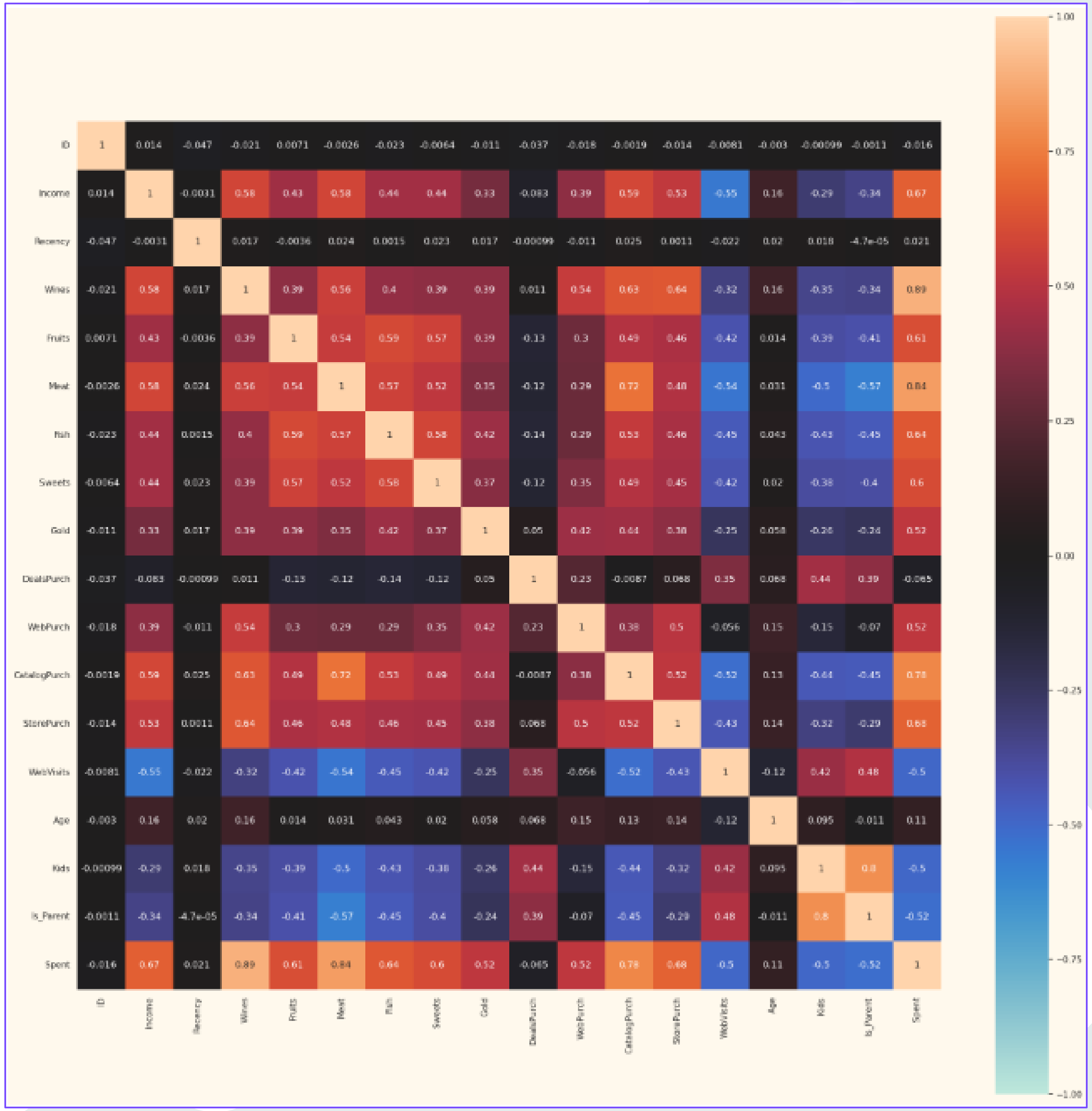






DATA EXPLORATION

Correlation Matrix



MODELING – CHOSEN MODELS

Suport Vector Classifier

Grid Search

Linear Regression

Decision Trees

MODELING – SVC and GRIDSEARCH

Regarding this two regression models, the data was treated identically.

1) Data Treatment:

- We created a new dataframe *df_feat* by removing the categorial data and the following target columns, for further training.

```
df_feat = data.drop(['Education', 'Dt_Customer', 'Marital_Status'], axis = 1)
df_target = data['Marital_Status']
```

2) Split Data Test:

- We split the features ('X') and target variable ('y') into training and testing sets;
- 25% of the data is reserved for testing;
- Using *random_state=2022* ensures consistent results across different runs.

```
X_train, X_test, y_train, y_test = train_test_split(df_feat, df_target, test_size=0.25,random_state=2022)
```

MODELING – SUPPORT VECTOR CLASSIFIER (SVC)

1) Data Training:

- Using Cross Validation approach with 10 folds

```
cross_valid_model = SVC(random_state=2022)
scores = cross_val_score(cross_valid_model, df_feat, df_target, cv=10)
```

```
0.65 accuracy with a standard deviation of 0.00
```

- Without Cross Validation

```
svc_model = SVC(random_state=2022, class_weight='balanced')
```

```
svc_model.fit(X_train,y_train)
```

2) Data Predictions:

```
svc_predictions = svc_model.predict(X_test)
```

```
print("%0.2f accuracy" % (accuracy_score(y_test, svc_predictions)))
0.51 accuracy
```

MODELING - GRIDSEARCH

1) Data Training:

- We used *RandomForestTreeClassifier*
- GridSearch with Random Forest Classifier involves using GridSearchCV, a hyperparameter tuning technique, to find the best hyperparameters for the Random Forest Classifier model. Random Forest builds multiple decision trees during training and outputs the mode of the classes for predictions.

```
gs_model = RandomForestClassifier(class_weight='balanced', random_state=2023)
```

2) Data Predictions:

```
gs_predictions = gs_model.predict(X_test)
```

```
gs_model.fit(X_train, y_train)
```

MODELING – SVC vs GRIDSEARCH

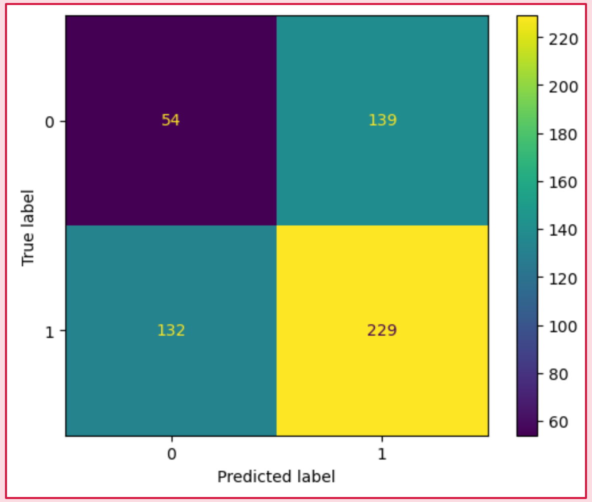
Support Vector Classification	GridSearch

Support Vector Classification

```
Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.28      0.28       193
           1       0.62      0.63      0.63       361

    accuracy                           0.51       554
   macro avg       0.46      0.46      0.46       554
weighted avg       0.51      0.51      0.51       554
```

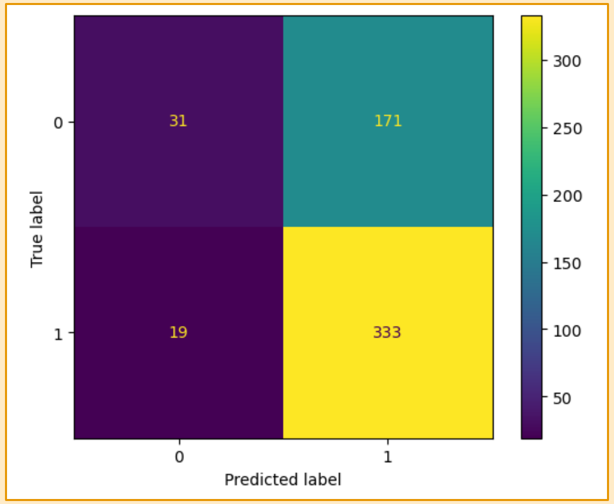


GridSearch

```
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.15      0.25       202
           1       0.66      0.95      0.78       352

    accuracy                           0.66       554
   macro avg       0.64      0.55      0.51       554
weighted avg       0.65      0.66      0.58       554
```



Conclusion:

- **Classification Report**

\- The model GridSearch has better values of precision and recall overall.

- **Confusion Matrix**

\- The percentage of true values (true positives and true negatives) predicted on the GridSearch model (~65.7%) are higher than the SVC model percentage (~51%).

We can conclude that the model GridSearch was a better approach to train and test data than the SVC.

MODELING – LINEAR REGRESSION

1) Data Treatment:

- We defined a feature matrix *X* by dropping the following columns;
- Our target variable, *y*, is set to *Spent*.

```
X = data.drop(['Education', 'Dt_Customer', 'Marital_Status'] + ['Kidhome', 'Teenhome'] + ['Wines', 'Fruits', 'Meat', 'Fish', 'Sweets', 'Gold'] + ['Spent'], axis=1)
y = data['Spent']
```

2) Split Data Test:

- We split the features ('X') and target variable ('y') into training and testing sets;
- 25% of the data is reserved for testing;
- Using *Random_state=2023* ensures consistent results across different runs.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=2023)
```

MODELING – LINEAR REGRESSION

3) Model Evaluation:

Initialization of a Linear Regression model object

```
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Initialization of a Linear Regression model object

```
print(lm.intercept_)
```

```
-59.68825869202374
```

Model coefficients

```
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

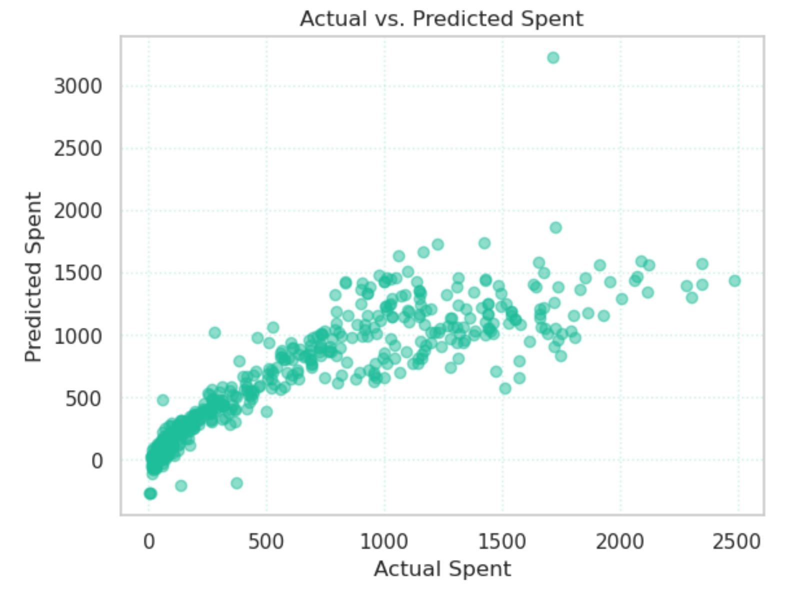
	Coefficient
Income	0.004122
Recency	0.100997
DealsPurch	-8.994111
WebPurch	37.424931
CatalogPurch	85.003446
StorePurch	42.733041
WebVisits	-2.426086
Age	-0.494622
Kids	-129.065452

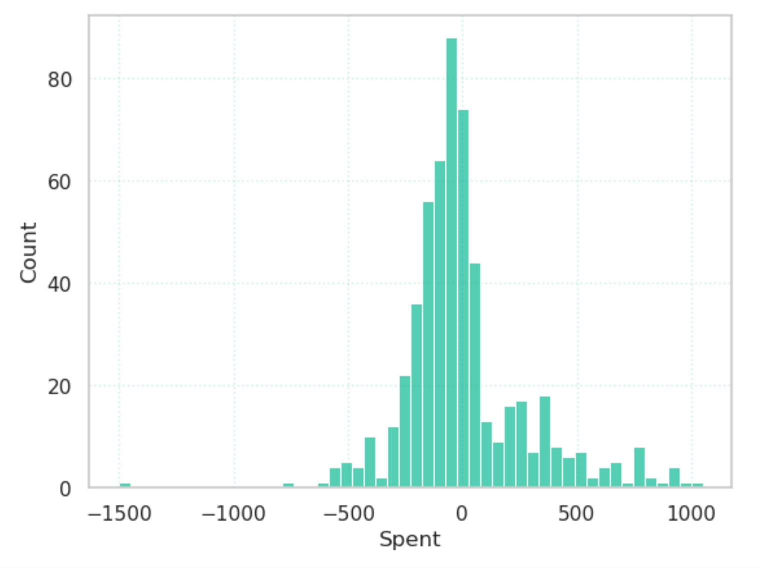
MODELING – LINEAR REGRESSION

4) Model Predictions:

```
predictions = lm.predict(X_test)
```

```
Mean Absolute Error:  187.23436989640257
Mean Squared Error:  75244.9668507371
RMSE:  274.3081603794118
```





After exploring the data using the linear regression model, we can conclude that this model is not the most suitable for exploring the data in the selected dataset. For this reason, we decided to continue exploring the data with other models.

MODELING – LINEAR REGRESSION

3) Model Evaluation: (Continuation)

Interpreting the coefficients:

- *Age:* Shows moderate negative impact.
- *WebVisits:* Negatively influences, but to a lesser extent.
- *Kids:* Displays a substantial negative impact having the largest negative coefficient.
- *DealsPurch:* Indicates a significant negative impact.

DealsPurch	-8.994111
WebVisits	-2.426086
Age	-0.494622
Kids	-129.065452

After observing the impact of the '**Age**', '**Kids**', '**DealsPurch**', '**WebVisits**' columns, we opted to remove them, since their impact on the model's predictive capacity appeared limited.

```
X = X.drop(['Age', 'Kids', 'DealsPurch', 'WebVisits'], axis=1)
```

Next, we splitted the data for testing again.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=2023)
```

MODELING – DECISION TREES

1) Data Treatment:

- We created a new dataframe *data2* by copying the original dataset data.
- We defined a feature matrix *X* by dropping columns and setting the target variable *y* to *Spent*.

```
data2 = data.copy()
X = data2.drop(['Marital_Status', 'Education', 'Dt_Customer', 'Spent'], axis=1)
y = data['Spent'].to_frame()
```

2) Split Data Test:

- We split the features ('X') and target variable ('y') into training and testing sets;
- 25% of the data is reserved for testing,
- Using *random_state=2024* ensures consistent results across different runs.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=2024)
```

MODELING – DECISION TREES

3) Model Analysis and Evaluation:

```
import time

def analise_model(model, model_name, y_train_analise=y_train):
    start_time = time.time()
    model.fit(X_train, y_train_analise)
    predictions = model.predict(X_test)
    print("time - {}".format(time.time()-start_time))
    predictions  = predictions.reshape(len(predictions),1 )
    # Métricas
    print(model)
    #print("Parâmetros:")
    #print(model.get_params())
    print("Mean Absolute Error: ", mean_absolute_error(y_test, predictions))
    print("Mean Squared Error: ", mean_squared_error(y_test, predictions, squared=True))
    print('RMSE: ', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

    r2 = r2_score(y_test, predictions)
    print('R² Score: ', r2)

   # sns.displot(y_test-predictions)
   # plt.show()
    ax = plt.axes()
    ax.plot([0, 500, 1000, 2000, 2500, 3000], [0, 500, 1000, 2000, 2500, 3000], 'r')
    plt.scatter(y_test,predictions)
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title(model_name)
    plt.annotate(f'R² Score: {r2:.4f}', xy=(0.7, 0.1), xycoords='axes fraction', fontsize=10, ha='center', color='blue')
    plt.show()
```

Interpretation:

Usage - The *analise_model* function can be used to analyse and visualize the performance of different regression models.

Time Execution - used to measure the execution time of fitting the model and making predictions.

Regression Metrics - Mean Absolute Error, Mean Squared Error, and RMSE (Root Mean Squared Error) are calculated and printed.

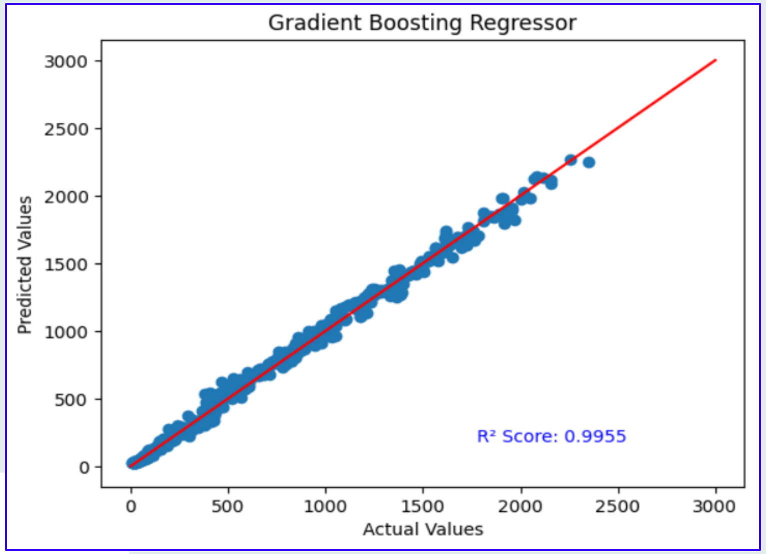
R^2 Score - The *R^2* score measures the goodness of fit.

MODELING – DECISION TREES

4) Model Selection and Evaluation:

```
dtr = DecisionTreeRegressor(random_state=2024)
rfr = RandomForestRegressor(n_estimators=20, max_depth=10, criterion='squared_error', random_state=2024)
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=2024)

analise_model(dtr, "Decision Tree Regressor")
analise_model(rfr, "Random Forest Regressor" , y_train.values.ravel())
analise_model(gbr, "Gradient Boosting Regressor")
```




```
time - 0.6810247898101807
GradientBoostingRegressor(random_state=2024)
Mean Absolute Error:  26.949907129257323
Mean Squared Error:  1583.0128107861642
RMSE:  39.78709352021286
R² Score:  0.99552490255378
```

From the three models (Decision Tree Regressor, Random Forest Regress, Gradient Boosting Regressor), the Gradient Boosting outperforms in terms of predictive accuracy (lowest *MAE, MSE, RMSE*, and highest R^2).

The choice of the best model depends on the trade-off between computational efficiency and predictive performance. Considering the balance of performance and execution time, Gradient Boosting seems to be a strong candidate for this regression task.

CONCLUSION

- After preparing the data analysis, processing, cleaning and exploration, we were able to have a global view of how the data in the chosen dataset is organized;
- We were able to see wich factors influence some people to spend more or less;
- With the training and testing models, we can say that the **DesicionTrees** model was the model that made the best dat prediction.

REFERENCES

Dataset: https://www.kaggle.com/datasets/imakash3011/customer-personality-analysis