Milestone 1: Business Understanding

GitHub and Live Website

Link to our GitHub Repository: https://github.com/Ettiene-Koekemoer/Machine-Learning-382-Project2-Group_J.git

Link to our Deployed Live Website: https://machine-learning-382-project2-group-j.onrender.com

Local website notebook and script are within the 'web_application.ipynb' and 'web_app.py' files respectively.

Problem Statement

Predict whether customers are likely to churn based on their past behaviour and demographics.

Data identification

In order for us to build a machine learning algorithm to predict customer churning, we will need a combination of features capturing the customer's interactions with our service as well as customer demographic information. Features that we will be uitilizing in our machine learning model will include:

- CustomerID
- Gender
- Age
- Income
- TotalPurchase
- NumOfPurchases
- Location
- MaritalStatus
- Education
- SubscriptionPlan
- Churn (label)

Hypothesis

We hypothesize the churn is influenced mainly by Gender, Age, Income, Total Purchases, Amount of Purchases, and SubscriptionPlan as the most, alongside combinations betweeen these features.

Businesses can prevent existing customers from leaving by being proactive and anticipating churn before it occurs.

The benefit of figuring out a firm's turnover rate is that it sheds light on how successfully it is keeping clients, which speaks to the caliber of service the company offers and its usefulness.

Collect and clean the data

We have collected raw data based on the desired features and target attributes for our churn prediction model. This raw data has been stored in the train.csv file in our data folder. We will now import this data into a dataframe and start cleaning the data.

Import

In [ ]:
```
# Supress warnings
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

import pandas as pd # data wrangling
import seaborn as sns # data visualization
import plotly.express as px
import matplotlib.pyplot as plt

# for cat features
from category_encoders import OneHotEncoder

from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.pipeline import make_pipeline

from skimpy import clean_columns
```

In [ ]:
```
df = pd.read_csv('./data/train.csv') #reading the data from the csv file to our
df.head() #display the first few data entries as well as column headings
```

Out[ ]:

	CustomerID	Gender	Age	Income	TotalPurchase	NumOfPurchases	Location	Mari
0	1	NaN	35	52850.0	1500	6.0	Urban	
1	2	Female	25	29500.0	800	3.0	Suburban	
2	3	Male	45	73500.0	2000	8.0	Rural	
3	4	Female	30	NaN	1200	5.0	Urban	
4	5	Male	55	80400.0	2500	9.0	Suburban	

We notice that our raw data has 10 features as well as a target feature called Churn. This data is not yet ready to be modelled and needs to be cleaned and prepared.

Preprocessing data

Removing irrelevent features

As we will not need to know the customer ID to determine if they will churn or not, it is not a relevent feature for machine learning modelling and can therefor be dropped.

In [ ]:
```
#removing the irrelevent feature
df.drop(
    columns='CustomerID',
    inplace=True
)

df.head() #inspecting the dataframe without the irrelevent feature
```

Out[ ]:

	Gender	Age	Income	TotalPurchase	NumOfPurchases	Location	MaritalStatus	Edu
0	NaN	35	52850.0	1500	6.0	Urban	Married	Ba
1	Female	25	29500.0	800	3.0	Suburban	NaN	
2	Male	45	73500.0	2000	8.0	Rural	Married	M
3	Female	30	NaN	1200	5.0	Urban	Single	Ba
4	Male	55	80400.0	2500	9.0	Suburban	Married	

Changing the target, Churn, to numeric values

We want to convert the target data type from string values to integer values for more accurate machine learning modelling.

In [ ]:
```
# Replacing the yes and no values with 1 and 0
df['Churn'].replace(
    {'Yes': 1, 'No': 0},
    inplace= True
)

df['Churn']
```

Out[ ]:
```
0      1
1      0
2      0
3      0
4      0
      ..
355    1
356    0
357    1
358    1
359    0
Name: Churn, Length: 360, dtype: int64
```

We have now converted the Churn datatype to int.

Data profiling

We will make use of the skimpy library to create a summary of desired data information.

In [ ]:
```
import skimpy as sk #importing the skimpy library

sk.skim(df) #create a summary of df information
```

skimpy summary

Data Summary

dataframe	Values
Number of rows	360
Number of columns	10

Data Types

Column Type	Count
string	5
int32	3
float64	2

number

column_name	NA	NA %	mean	sd	p0	p25
Age	0	0	37	10	19	28
Income	3	0.83	54000	19000	20000	35000
TotalPurchase	0	0	1600	590	500	1000
NumOfPurchases	4	1.11	6	2.1	2	4
Churn	0	0	0.3	0.46	0	0

string

column_name	NA	NA %	words per row
Gender	5	1.39	
Location	4	1.11	
MaritalStatus	5	1.39	
Education	0	0	
SubscriptionPlan	0	0	

End

Some key takeaways of this skimpy summary is that we have now have 5 numeric features(including the target), and 5 categorical features. We also notice that there are missing values for the features Income, NumOfPurchases, Gender, Location, and MaritalStatus. We will need to handle these missing features.

Handling missing values

In [ ]:
```
num_col = ['Income','NumOfPurchases'] #creating a list of the numeric features w
cat_col = ['Gender','Location','MaritalStatus'] #creating a list of categorical

for col1 in num_col: #for each of the columns in the list replace the missing va
    df[col1].fillna(
        df[col1]
        .dropna()
        .mean(),
        inplace= True
    )
```

```
for col2 in cat_col:
    df[col2].fillna( #replace the missing categorical values with the mode of th
        df[col2]
        .mode()[0],
        inplace= True
    )

df.isnull().sum()
```

Out[ ]:

```
Gender             0
Age                0
Income             0
TotalPurchase      0
NumOfPurchases     0
Location           0
MaritalStatus      0
Education          0
SubscriptionPlan   0
Churn              0
dtype: int64
```

In [ ]:

```
df.head()
```

Out[ ]:

	Gender	Age	Income	TotalPurchase	NumOfPurchases	Location	MaritalStatu
0	Female	35	52850.000000	1500	6.0	Urban	Marriec
1	Female	25	29500.000000	800	3.0	Suburban	Marriec
2	Male	45	73500.000000	2000	8.0	Rural	Marriec
3	Female	30	54273.529412	1200	5.0	Urban	Singl
4	Male	55	80400.000000	2500	9.0	Suburban	Marriec

We now have no missing values in our dataframe.

Checking the cardinality of categorical features

In [ ]:

```
df.select_dtypes('object').nunique()
```

Out[ ]:

```
Gender             2
Location           3
MaritalStatus      2
Education          4
SubscriptionPlan   3
dtype: int64
```

As our categorical features don't have very low or very high cardinality, we do not have to handle any feature cardinality.

High collinearity

We will now inspect the correlation between the features to detect any cases of high collinearity.

In [ ]:
```
corr_df = df.select_dtypes('number').corr()
corr_df
```

Out[ ]:

	Age	Income	TotalPurchase	NumOfPurchases	Churn
Age	1.000000	0.989016	0.991159	0.973570	-0.578108
Income	0.989016	1.000000	0.996362	0.979777	-0.576808
TotalPurchase	0.991159	0.996362	1.000000	0.980369	-0.569293
NumOfPurchases	0.973570	0.979777	0.980369	1.000000	-0.543626
Churn	-0.578108	-0.576808	-0.569293	-0.543626	1.000000

In [ ]:
```
fig = px.imshow(corr_df, color_continuous_scale='Spectral')
fig.update_layout(title='Heat Map: Correlation of Features', font=dict(size=12))
fig.show()
```

We notice that the highest collinearity is between TotalPurchase, Income, Age, and NumOfPurchases. As Income, TotalPurchase, and NumOfPurchases of the customer are important for churn predicitons, we can look at removing the Age feature for better model accuracy.

In [ ]:
```
#dropping feature with high collinearity
df.drop(
    columns= 'Age',
    inplace= True
)

df.head()
```

Out[ ]:

	Gender	Income	TotalPurchase	NumOfPurchases	Location	MaritalStatus	Edu
0	Female	52850.000000	1500	6.0	Urban	Married	Bac
1	Female	29500.000000	800	3.0	Suburban	Married	
2	Male	73500.000000	2000	8.0	Rural	Married	M
3	Female	54273.529412	1200	5.0	Urban	Single	Bac
4	Male	80400.000000	2500	9.0	Suburban	Married	

Storing the prepared data

Creating a prepare data function

We will now combine our data preparation code into a single function which will return a dataframe of prepared data ready for modelling.

In [ ]:
```
def prepare_data(path): #declaring the function with paramater path which will b
    prep_df = pd.read_csv(path) #reading the raw data from the path into a dataf
```

```
    #removing the irrelevent feature
    prep_df.drop(
        columns='CustomerID',
        inplace=True
    )

    # Replacing the yes and no values with 1 and 0
    prep_df['Churn'].replace(
        {'Yes': 1, 'No': 0},
        inplace= True
    )

    num_col = ['Income','NumOfPurchases'] #creating a list of the numeric featur
    cat_col = ['Gender','Location','MaritalStatus'] #creating a list of categori

    for col in num_col: #for each of the columns in the list replace the missing
        prep_df[col].fillna(
            prep_df[col]
            .dropna()
            .mean(),
            inplace= True
        )

    for col1 in cat_col:
        prep_df[col1].fillna( #replace the missing categorical values with the m
            prep_df[col1]
            .mode()[0],
            inplace= True
        )

    prep_df.drop(
        columns= 'Age',
        inplace= True
    )

    return clean_columns(prep_df)
```

Calling the prepare_data function

In [ ]:
```
prepared_df = prepare_data('./data/train.csv')
prepared_df.to_csv('./data/prepared_data.csv')
```

Milestone 2: Machine Learning Model Implementation

Data exploration

We will now explore our prepared data to gain more insights into their meaning and behaviour.

Univariate analysis

We will start our analysis by looking at the state and behaviour of our target, Churn.

In [ ]:
```
# Prepare data to display
labels = (
    prepared_df['churn']
    .astype('str')
    .str.replace('0','No', regex=True)
    .str.replace('1','Yes', regex=True)
    .value_counts()
)

# Create figure using Plotly
fig = px.bar(
    data_frame=labels,
    x=labels.index,
    y=labels.values,
    title=f'Class Imbalance',
    color=labels.index
)

# Add titles & Display figure
fig.update_layout(xaxis_title='Churn', yaxis_title='Number of Customers')
fig.show()
```

For business purposes, we want to focus on the customers that do churn. It is clear in this graph that the amount of customers that have churned is quite significant and the business would like to reduce this number.

Bivariate/Multi-variate analysis

Numeric Features

We will now visualise the relationships of the numeric features against our target to understand their behaviour and impact.

In [ ]:
```
plot_cols = ['income','total_purchase','num_of_purchases']

# Plot numeric features against target
plt.Figure(figsize=(3,4))
for col in plot_cols:
    fig = px.box(data_frame=prepared_df[plot_cols], x=col, color=prepared_df['ch
    fig.update_layout(xaxis_title=f'{col} Feature')
    fig.show()
```

After handling the outliers we concluded the following:

- Customers with lower income is more likely to churn
- Customers with lower total purchase amounts are churning
- Customers with lower number of purchases are also churning

Categorical features

In [ ]:
```
plot_columns = ['gender','location','marital_status','education','subscription_p
for plot in plot_columns:
    new_df = pd.DataFrame(
        prepared_df[[plot, 'churn']]
        .groupby(['churn'])
        .value_counts()
        .reset_index()
    )

    # Plot Category feature vs label
    fig = px.bar(
        data_frame=new_df,
        x=plot,
        y='count',
        facet_col='churn',
        color=new_df['churn'].astype(str), # convert it to string to avoid conti
        title=f'{plot} vs Target'
    )

    fig.update_layout(xaxis_title=plot, yaxis_title='Number of Customers')
    fig.show()
```

Focussing on the the customers that do churn we notice from the graphs that:

- More females are churning
- Customers from urban areas are churning the most
- More single customers are churning
- Customers with bachelor's degrees are churning the most
- Bronze level subscription plan customers are the ones that churn the most

Model Evaluation

Importing necessary libraries

In [ ]:
```
import numpy as np
import joblib
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_digits
from xgboost import XGBClassifier

accuracy_scores = []
precisions = []
f1_scores = []
recalls = []
mae_scores = []
```

Splitting the data

In [ ]:
```
target = 'churn'
x = prepared_df.drop(columns=[target], inplace=False)
y = prepared_df[target]

x_Train, x_Test, y_Train, y_Test = train_test_split(x, y, test_size=0.4, random_

print(
    f'Training dataset \
    \nx_Train: {x_Train.shape[0]/len(x)*100:.0f}% \ny_Train: {y_Train.shape[0]/l
    \n\nValidation dataset \
    \nx_Val: {x_Test.shape[0]/len(x)*100:.0f}% \ny_Val: {y_Test.shape[0]/len(x)*
)
```

```
Training dataset
x_Train: 60%
y_Train: 60%

Validation dataset
x_Val: 40%
y_Val: 40%
```

Base accuracy

In [ ]:
```
accuracy_Base = y_Train.value_counts(normalize=True).max()

print("Baseline Accuracy:", round(accuracy_Base, 2))
```

```
Baseline Accuracy: 0.71
```

Linear Regression model

In [ ]:
```
# Encode, build, and fit model
lin_pipeline = make_pipeline(
    OneHotEncoder(),
    StandardScaler(),
    LinearRegression()
)
# Define hyperparameters grid
param_grid = {
    'linearregression__fit_intercept': [True, False],
    'linearregression__positive': [True, False],
}
# Perform grid search cross-validation
lin_model = GridSearchCV(lin_pipeline, param_grid, cv=5, n_jobs=-1)
lin_model.fit(x_Train, y_Train)

# Train model
y_test_lin_prob = lin_model.predict(x_Test)
y_test_lin_pred = (y_test_lin_prob > 0.5).astype(int)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_lin_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_lin_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_lin_pred),4)),
```

```
f1_scores.append(round(f1_score(y_Test, y_test_lin_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_lin_pred),4))
```

Logistic Regression model

In [ ]:
```
# Encode, build, and fit model
log_pipeline = make_pipeline(
    OneHotEncoder(),
    StandardScaler(),
    LogisticRegression(max_iter=10000)
)
param_grid = {
    'logisticregression__C': [0.001, 0.01, 0.1, 1, 10, 100],
    'logisticregression__penalty': ['l1', 'l2'],
    'logisticregression__solver': ['liblinear', 'saga']
}
# Perform grid search cross-validation
log_Model = GridSearchCV(log_pipeline, param_grid, cv=5, n_jobs=-1)
log_Model.fit(x_Train, y_Train)

# Train model
y_test_log_pred = log_Model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_log_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_log_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_log_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_log_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_log_pred),4))
```

Decision Tree model

In [ ]:
```
tree_hyperparam = range(1, 8)

# List of scores for visualization
train_Scores = []
test_Scores = []

for i in tree_hyperparam:
    # Encode, build, and fit model
    tree_Model = make_pipeline(
        OneHotEncoder(),
        StandardScaler(),
        DecisionTreeClassifier(max_depth=i, random_state=42)
    )
    tree_Model.fit(x_Train, y_Train)

    # Training accuracy score
    train_Scores.append(tree_Model.score(x_Train, y_Train))

    # Testing accuracy score
    test_Scores.append(tree_Model.score(x_Test, y_Test))

tune_data = pd.DataFrame(
    data = {'Training': train_Scores, 'Testing': test_Scores},
    index=tree_hyperparam
)
```

```
fig = px.line(
    data_frame=tune_data,
    x=tree_hyperparam,
    y=['Training', 'Testing'],
    title="Decision Tree model training & testing curves"
)
fig.update_layout(xaxis_title ="Maximum Depth", yaxis_title="Accuracy Score")
fig.show()

y_test_tree_pred = tree_Model.predict(x_Test)

accuracy_scores.append(round(accuracy_score(y_Test, y_test_tree_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_tree_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_tree_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_tree_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_tree_pred),4))
```

Random Forest Classifier model

In [ ]:

```
# Encode, build, and fit model
forest_pipeline = make_pipeline(
    OneHotEncoder(),
    StandardScaler(),
    RandomForestClassifier(random_state=42)
    )
# Define hyperparameters grid
param_grid = {
    'randomforestclassifier__n_estimators': [50, 100, 200],
    'randomforestclassifier__max_depth': [None, 10, 20],
    'randomforestclassifier__min_samples_split': [2, 5, 10],
    'randomforestclassifier__min_samples_leaf': [1, 2, 4]
}
# Perform grid search cross-validation
forest_model = GridSearchCV(forest_pipeline, param_grid, cv=5, n_jobs=-1)
forest_model.fit(x_Train, y_Train)

# Train model
y_test_for_pred = forest_model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_for_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_for_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_for_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_for_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_for_pred),4))
```

Gaussian Naive Bayes model

In [ ]:

```
# Encode, build, and fit model
bayes_model = make_pipeline(
    OneHotEncoder(),
    StandardScaler(),
    GaussianNB()
    )
# No tuning or cross valdiation needed
bayes_model.fit(x_Train, y_Train)
```

```
# Train model
y_test_bay_pred = bayes_model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_bay_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_bay_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_bay_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_bay_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_bay_pred),4))
```

Gradient Boosting Classifier model

In [ ]:
```
# Encode, build and fit model
gbc = make_pipeline(
    OneHotEncoder(),
    StandardScaler(),
    GradientBoostingClassifier(n_estimators=300,
                                                                learning_rate=0.
                                                                random_state=100
                                                                max_features=5 )
)
gbc.fit(x_Train, y_Train)

# Train model
y_test_gbc_pred = gbc.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_gbc_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_gbc_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_gbc_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_gbc_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_gbc_pred),4))
```

XGBoost Classifier model

In [ ]:
```
# declare parameters
params = {
            'objective':'binary:logistic',
            'max_depth': 4,
            'alpha': 10,
            'learning_rate': 1.0,
            'n_estimators':100
        }

# Encode, build and fit model
xgb_model = make_pipeline(
    OneHotEncoder(),
    StandardScaler(),
    XGBClassifier(**params)
)
xgb_model.fit(x_Train, y_Train)

# Train model
y_test_xgb_pred = xgb_model.predict(x_Test)

# Populate evaluation metrics
```

```
accuracy_scores.append(round(accuracy_score(y_Test, y_test_xgb_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_xgb_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_xgb_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_xgb_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_xgb_pred),4))
```

In [ ]:
```
metrics_1 = {
        'Accuracy': accuracy_scores,
        'Precision': precisions,
        'F1-Score': f1_scores,
        'Recall': recalls,
        'MAE': mae_scores
    }

pd.DataFrame(
    data=metrics_1,
    index=['Linear Regression','Logistic Regression', 'Decision Tree','Random Fo
).sort_values(
    by='Accuracy',
    ascending=False
)
```

Out[ ]:

	Accuracy	Precision	F1-Score	Recall	MAE
Random Forest Classifier	0.9097	0.8636	0.8539	0.8444	0.0903
Gradient Boosting Classifier	0.8819	0.8043	0.8132	0.8222	0.1181
Decision Tree	0.8681	0.7708	0.7957	0.8222	0.1319
XGB Classifier	0.7917	0.7143	0.6250	0.5556	0.2083
Logistic Regression	0.7569	0.6087	0.6154	0.6222	0.2431
Linear Regression	0.7361	0.5714	0.5957	0.6222	0.2639
Gaussian Naive Bayes	0.6806	0.4943	0.6515	0.9556	0.3194

Here, Random Forest Classifier performs the best before feature engineering. We follow with saving this as our first model.

In [ ]:
```
# Save Model
joblib.dump(forest_model, './artifacts/model_1.pkl')
```

Out[ ]:
```
['./artifacts/model_1.pkl']
```

Feature engineering

We perform feature engineering by laying the income values into categories and getting the average per pruchase.

In [ ]:
```
engineered_df = prepared_df

# Bin income into brackets
bins = [0, 30000, 50000, 70000, float('inf')]
labels = ['Low Income', 'Medium Income', 'High Income', 'Very High Income']
engineered_df['income_bin'] = pd.cut(engineered_df['income'], bins=bins, labels=
```

```
# Added average feature
engineered_df['average_purchase'] = round(engineered_df['total_purchase'] / engi

# Reorder columns
engineered_df = engineered_df[['gender', 'income', 'income_bin', 'total_purchase
engineered_df
```

Out[ ]:

	gender	income	income_bin	total_purchase	num_of_purchases	average_pur
0	Female	52850.000000	High Income	1500	6.00000	
1	Female	29500.000000	Low Income	800	3.00000	
2	Male	73500.000000	Very High Income	2000	8.00000	
3	Female	54273.529412	High Income	1200	5.00000	
4	Male	80400.000000	Very High Income	2500	9.00000	
...	...	...	...	...	...	
355	Female	26000.000000	Low Income	750	5.97191	
356	Male	71000.000000	Very High Income	2100	8.00000	
357	Female	31000.000000	Medium Income	900	4.00000	
358	Male	51000.000000	High Income	1500	6.00000	
359	Female	72000.000000	Very High Income	2100	8.00000	

360 rows × 11 columns

Rebuilding the models with feature engineering

Following is the refitting and retraining of the previous models but with the feature engineered data.

In [ ]:

```
accuracy_scores = []
precisions = []
f1_scores = []
recalls = []
mae_scores = []

x = engineered_df.drop(columns=[target], inplace=False)
y = engineered_df[target]
```

```
x_Train, x_Test, y_Train, y_Test = train_test_split(x, y, test_size=0.4, random_

# Refit linear model
lin_model.fit(x_Train, y_Train)

# Train model
y_test_lin_prob = lin_model.predict(x_Test)
y_test_lin_pred = (y_test_lin_prob > 0.5).astype(int)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_lin_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_lin_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_lin_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_lin_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_lin_pred),4))


# Refit logistic model
log_Model.fit(x_Train, y_Train)

# Train model
y_test_log_pred = log_Model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_log_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_log_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_log_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_log_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_log_pred),4))


# Refit tree model
tree_Model.fit(x_Train, y_Train)

y_test_tree_pred = tree_Model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_tree_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_tree_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_tree_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_tree_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_tree_pred),4))


# Refit forest model
forest_model.fit(x_Train, y_Train)

# Train model
y_test_for_pred = forest_model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_for_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_for_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_for_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_for_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_for_pred),4))


# Refit bayes model
bayes_model.fit(x_Train, y_Train)
```

```
# Train model
y_test_bay_pred = bayes_model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_bay_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_bay_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_bay_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_bay_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_bay_pred),4))


# Refit gbc model
gbc.fit(x_Train, y_Train)

# Train model
y_test_gbc_pred = gbc.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_gbc_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_gbc_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_gbc_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_gbc_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_gbc_pred),4))


# Refit xgb model
xgb_model.fit(x_Train, y_Train)

y_test_xgb_pred = xgb_model.predict(x_Test)

# Populate evaluation metrics
accuracy_scores.append(round(accuracy_score(y_Test, y_test_xgb_pred),4)),
precisions.append(round(precision_score(y_Test, y_test_xgb_pred),4)),
recalls.append(round(recall_score(y_Test, y_test_xgb_pred),4)),
f1_scores.append(round(f1_score(y_Test, y_test_xgb_pred),4))
mae_scores.append(round(mean_absolute_error(y_Test,y_test_xgb_pred),4))

metrics_2 = {
        'Accuracy': accuracy_scores,
        'Precision': precisions,
        'F1-Score': f1_scores,
        'Recall': recalls,
        'MAE': mae_scores
    }

pd.DataFrame(
    data=metrics_2,
    index=['Linear Regression','Logistic Regression', 'Decision Tree','Random Fo
).sort_values(
    by='Accuracy',
    ascending=False
)
```

Out[ ]:

	Accuracy	Precision	F1-Score	Recall	MAE
Random Forest Classifier	0.8958	0.8571	0.8276	0.8000	0.1042
Gradient Boosting Classifier	0.8958	0.8261	0.8352	0.8444	0.1042
Decision Tree	0.8889	0.8222	0.8222	0.8222	0.1111
Logistic Regression	0.8611	0.7778	0.7778	0.7778	0.1389
XGB Classifier	0.8403	0.7292	0.7527	0.7778	0.1597
Linear Regression	0.7986	0.6667	0.6882	0.7111	0.2014
Gaussian Naive Bayes	0.6528	0.4737	0.6429	1.0000	0.3472

Random Forest Classifier still performs the best, although has lower metrics from more noise in its data.

This model is saved as our second model below.

In [ ]:

```
# Save Model
joblib.dump(forest_model, './artifacts/model_2.pkl')

# Loading Model
final_model = joblib.load('./artifacts/model_2.pkl')
```

Feature importances

The loaded model is now used to discover and save the feature importance values.

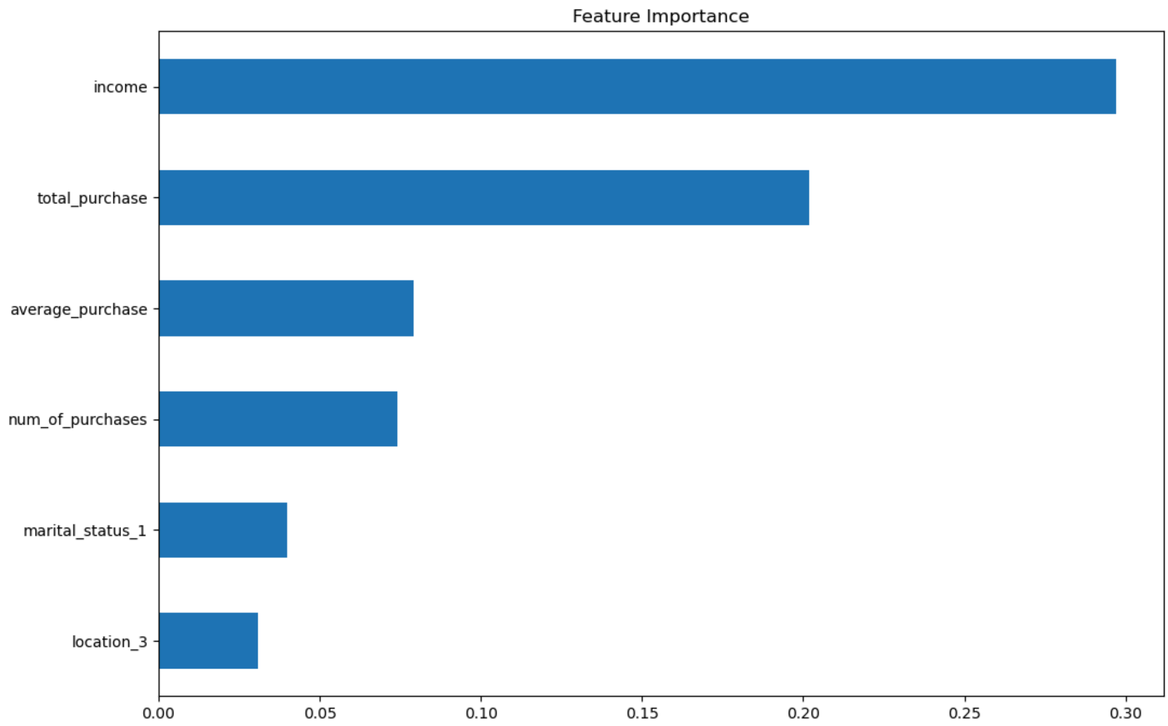
In [ ]:

```
best = final_model.best_estimator_
# Get coefficients of features
coefficients = best.named_steps.randomforestclassifier.feature_importances_

# Get feature names
features = best.named_steps["onehotencoder"].get_feature_names()

# Create a Series of features
feat_imp = pd.Series(data=coefficients, index=features)

plot_feat_imp = feat_imp.sort_values(ascending=True).tail(6)
plot_feat_imp.plot(kind="barh", figsize=(12,8))
plt.title("Feature Importance")
plt.show()
```



In [ ]:
```
# Saving importance values
feat_imp.to_csv('./artifacts/feature_importance.csv')
```

Creating a prediction function

Predictions are now done on test data without a target feature to test the loaded model.

In [ ]:
```
def make_predictions(csv_file):
    pred_model = joblib.load('./artifacts/model_2.pkl')

    pred_df = pd.read_csv(csv_file)

    #removing the irrelevent feature
    pred_df.drop(
        columns='CustomerID',
        inplace=True
    )

    num_col = ['Income','NumOfPurchases'] #creating a list of the numeric featur
    cat_col = ['Gender','Location','MaritalStatus'] #creating a list of categori

    for col in num_col: #for each of the columns in the list replace the missing
        pred_df[col].fillna(
            pred_df[col]
            .dropna()
            .mean(),
            inplace= True
        )

    for col1 in cat_col:
        pred_df[col1].fillna( #replace the missing categorical values with the m
            pred_df[col1]
            .mode()[0],
            inplace= True
        )
```

```
    pred_df.drop(
        columns= 'Age',
        inplace= True
    )

    pred_df = clean_columns(pred_df)

    label_enc = LabelEncoder()

    # Bin income into brackets
    bins = [0, 30000, 50000, 70000, float('inf')]
    labels = ['Low Income', 'Medium Income', 'High Income', 'Very High Income']
    pred_df['income_bin'] = pd.cut(pred_df['income'], bins=bins, labels=labels,

    # Added average feature
    pred_df['average_purchase'] = round(pred_df['total_purchase'] / pred_df['num

    # Reorder columns
    pred_df = pred_df[['gender', 'income', 'income_bin', 'total_purchase', 'num_

    predictions = pred_model.predict(pred_df)

    predictions = np.where(predictions == 1, 'yes', 'no')

    return predictions
```

The predictions get stored to the relevant file.

In [ ]:
```
test_df = make_predictions('./data/test.csv') #making predictions on test.csv da
np.savetxt('./artifacts/predictions.csv', test_df, delimiter=',', fmt='%s') #sav
```

GitHub and Live Website

Link to our GitHub Repository: https://github.com/Ettiene-Koekemoer/Machine-Learning-382-Project2-Group_J.git

Link to our Deployed Live Website: https://machine-learning-382-project2-group-j.onrender.com

Local website notebook and script are within the 'web_application.ipynb' and 'web_app.py' files respectively.