Predictive analytics to improve life insurance sales: A machine learning approach for Imperial Ltd.'s customer base.

Jaime Rubio Diaz
40425150

Contents

1. Introduction and Background

This project aims to identify the characteristics of customers who are most likely to purchase the life insurance product offered by Imperials Ltd. Using a large database of historical customer data, a robust predictive model will be built using Python. At the same time, a comprehensive descriptive analysis of the data set will be conducted to identify patterns related to life insurance purchasing behaviour. The overall objective is to provide valuable recommendations to Imperial Ltd, so that these can be used to increase sales of their life insurance product.

Shamsuddin et al., (2023) pursuing the same goal but working with an unbalanced dataset and using the SMOTE technique, concluded that among five machine learning classification algorithms, Neural Networks and Random Forests showed the highest accuracy. He et.al. (2020) studying customer churn in an insurance company, used an approach with six classification models, including Random Forest and Support Vector Machine.

2. Methodology

"The Cross Industry Standard Process for Data Mining" CRISP-DM is a comprehensive methodology which provides a clear framework to guide the entire data mining process, from business understanding to solution implementation (Wirth and Hipp, 2000).

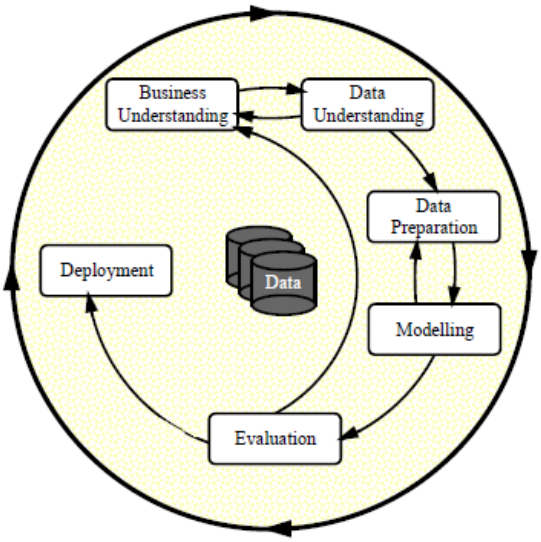


Figure 1. CRISP-DM
(Source: Wirth and Hipp, 2000)

Business Understanding

A life insurance policy is a pact between an insurance company and an individual, the individual pays premiums to the company during his or her lifetime and the insurer agrees to pay money to the designated beneficiaries in the event of the insured's death (Fontinelle, 2023). The price that policyholders pay during their lives to the company is determined primarily by their age and health.

Since the marketing team of the company want to increase sales of the life insurance product, it is crucial to identify those customers who are most likely to buy the product. By doing this, the company

can invest all its efforts in reaching valuable customers and minimize investments in reaching non-valuable customers (Verhoef and Donkers, 2001).

Data Understanding

The "sales_data" dataset comprises information on 40,000 customers, including 12 variables. One of these variables, named "flag" denotes whether the customer purchased the life insurance product.

Table 1. Data description

Customers Characteristics			**Missing values**
gender	Gender of the customer	Categorical (B)*	0
education	Educational level of the customer	Categorical (NB)*	741 (1.85%)
house_val	Value of customer house	Continuous	0
age	Age of the customer	Categorical (NB)	0
online	Whether or not the customer purchased online	Categorical (B)	0
marriage	Marital status of the customer	Categorical (B)	14027 (35.06%)
child		Categorical (B)	0
occupation	Occupation of the customer	Categorical (NB)	0
mortgage		Ordinal	0
house_owner		Categorical (B)	3377 (8.44%)
region		Categorical (NB)	0
fam_income		Ordinal	0
Target variable			
Purchased (flag)	Whether the customer purchased or not	Categorical (B)	0

*B = binary, NB = non-binary

As illustrate in Figure 2, the labels for various variable levels are unclear, requiring modifications during data pre-processing. The dataset also contains missing values. Additionally, certain variables expected to be binary, such as "gender" or “child" include an extra category labelled "Unknown." Given the context of these variables, "Unknown" is not a meaningful classification and will therefore be converted to “np.nan” for consistent imputation with other missing data. This approach is adopted because the presence of "Unknown" on certain variables equates to a lack of information. For example, if a customer had to consider the variable "preferred food": if "Unknown" is stored, this could suggest an absence of preference, which provides information. However, for a variable such as "sex", where biological categorizations recognize a binary classification, the label "Unknown" does not provide any information. Therefore, in the context of this data set, treating "Unkown" as missing data is the most reasonable approach. To evaluate associations across variables, various tests will be conducted based on the type of each variable, as detailed in Table 2.

Variables Types	Association method	Flag and
Categorical (B) - Categorical (B)	Phi coefficient	**Online, marriage, house_owner, gender, child**
Categorical (B) - Categorical (NB)	Cramer's V	**education, age, occupation, region**
Categorical (B) - Continuous	Point-biserial	**house_val**
Categorical (B) - Ordinal	Point-biserial	**fam_income, mortgage**

Table 2. Association methods

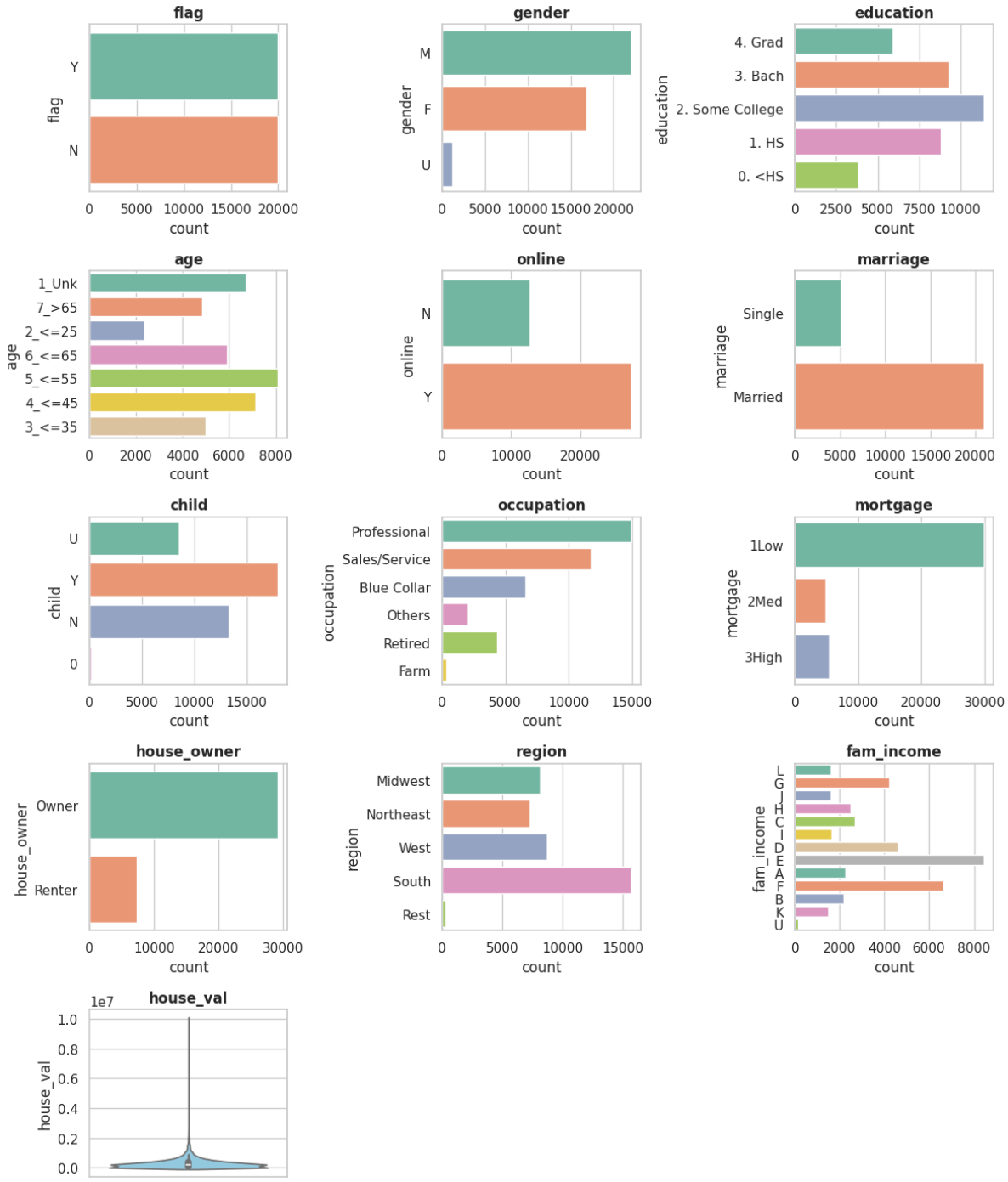


Figure 2. First visualizations

Data Preparation

Data preparation plays a key role in data analysis, accounting for a significant portion of the time spent, often more than 80%, underscoring its importance in improving results (Press, 2023). The processes involved in this part will be detailed in the following section. For this dataset, they include: 1. Changing some of the variable level names, 2. Imputing missing values by selecting the appropriate imputation technique.

Modelling

The relationship between interpretability and model flexibility is crucial; in the case of prediction, robust models such as Deep Learning should be prioritized, sacrificing some interpretability, however, to understand the underlying relationships, more interpretable models should be used, even if they are less robust (James, et al. 2023). For projects like this, balancing interpretability with flexibility is crucial to provide the company with understandable insights.

Figure 3. Interpretability vs Flexibility
Source: (James, 2023)

Table 3. Accuracy of the initial models

The accuracy of the initial models					
Logistic Regression	**Random Forest**	**Bagging**	**Gradient Decent**	**Support Vector Machine**	**Neural Networks**
0.55	0.65	0.633	0.50	0.68	0.68

Based on previous results, to enhance interpretability, it will be used Support Vector Machines and Random Forest for their balance between robustness and clarity; Neural Networks results will be included in the appendix.

Support Vector Machine	This algorithm transforms the data using a non-linear mapping to construct a hyperplane by performing a classification in this space, taking advantage of kernel functions (Hearst et al., 1998).
Random Forest	It is an algorithm that builds multiple decision trees, the method stands out for its ability to handle noisy data and its resistance to overfitting, in addition, it offers valuable internal estimates such as the importance of variables (Breiman, 2001).
Neural Networks	They are inspired by the brain's architecture, they consist of multiple layers and multiple activation functions that aim to forecast outcomes from inputs, functioning as advanced statistical models (Abdi et.al. 1999).

3. Data Pre-processing

Chai (2020) compared the results of the algorithms before and after performing data cleaning and concluded that small amounts of outliers or small amounts of incorrectly treated missing data can lead to significant changes in the model.

Firstly, some levels of some variables are modified in some cases to improve the informativeness of the variables and in other cases due to an error when they were stored.

Table 4. Changes in variables levels names

Variable	Original Value	New Value
education	'0. <HS'	'Lower than High School'
	'1. HS'	'High School'
	'2. Some College'	'College'
	'3. Bach'	'Bachelor'
	'4. Grad'	'Graduate'
gender	'F'	'Female'
	'M'	'Male'
	'U'	'Unknown'
age	'2_<=25'	'Young (0,25]'
	'3_<=35'	'Young Adult (25,35]'
	'4_<=45'	'Adult (35, 45]'
	'5_<=55'	'Adult II (45,55]'
	'6_<=65'	'Senior (55,65]'
	'7_>65'	Retired (more than 65)'
	'1_Unk'	'Unknown'
online	'Y'	'Yes'
	'N'	'No'
child	'0'	'No'
	'U'	'Unknown'
	'Y'	'Yes'
	'N'	'No'

Firstly, as detailed, "Unknown" values are converted to missing values for imputation.

Acuña and Rodriguez (2004) used mode imputation, case deletion, and KNN for the same purpose, assessing their impact on classification problems. Similarly, Faisal and Tutz (2022) compared methods for missing values in binary and multi-categorical variables, employing mode, MICE, and KNN among others, for datasets with 10, 20, and 30% missing values. In this dataset, missing values occur in a categorical variable, making K-nearest neighbour (KNN) unsuitable. Furthermore, data deletion is not viable due to the fact that "marriage" has 35% missing values, which would result in a significant loss of information. Thus, mode imputation is chosen to address the missing values.

4. Data Analytics

Descriptive Statistics and Visualisations

Table 5. Association coefficients

Association coefficients					
Age	0.22	**Region**	0.05	**Child**	0.05
Occupation	0.24	**Mortgage**	0.22	**House owner**	0.12
Education	0.25	**Online**	0.21	**Family Income**	0.23
Gender	0.18	**Marriage**	0.02	**House value**	0.16

After analysing associations between all predictors and the target variables, it appears they are not correlated. However, all the p-values are below 0.01, indicating statistical significance.

By a first simple descriptive analysis after the data pre-processing (Figure 5), it can be appreciated that life insurance purchasers are predominantly male, with a higher incidence of being married and situated within the adult to middle-aged categories. They typically have achieved higher education levels, most notably college and bachelor degrees followed by graduates, and are professionally employed. These customers tend to be homeowners, with a clear bias towards those with lower mortgages, suggesting a certain degree of financial stability. They are frequently located in the South region. The trend of completing transactions online is evident, reflecting a preference or convenience for digital engagement when purchasing life insurance products.

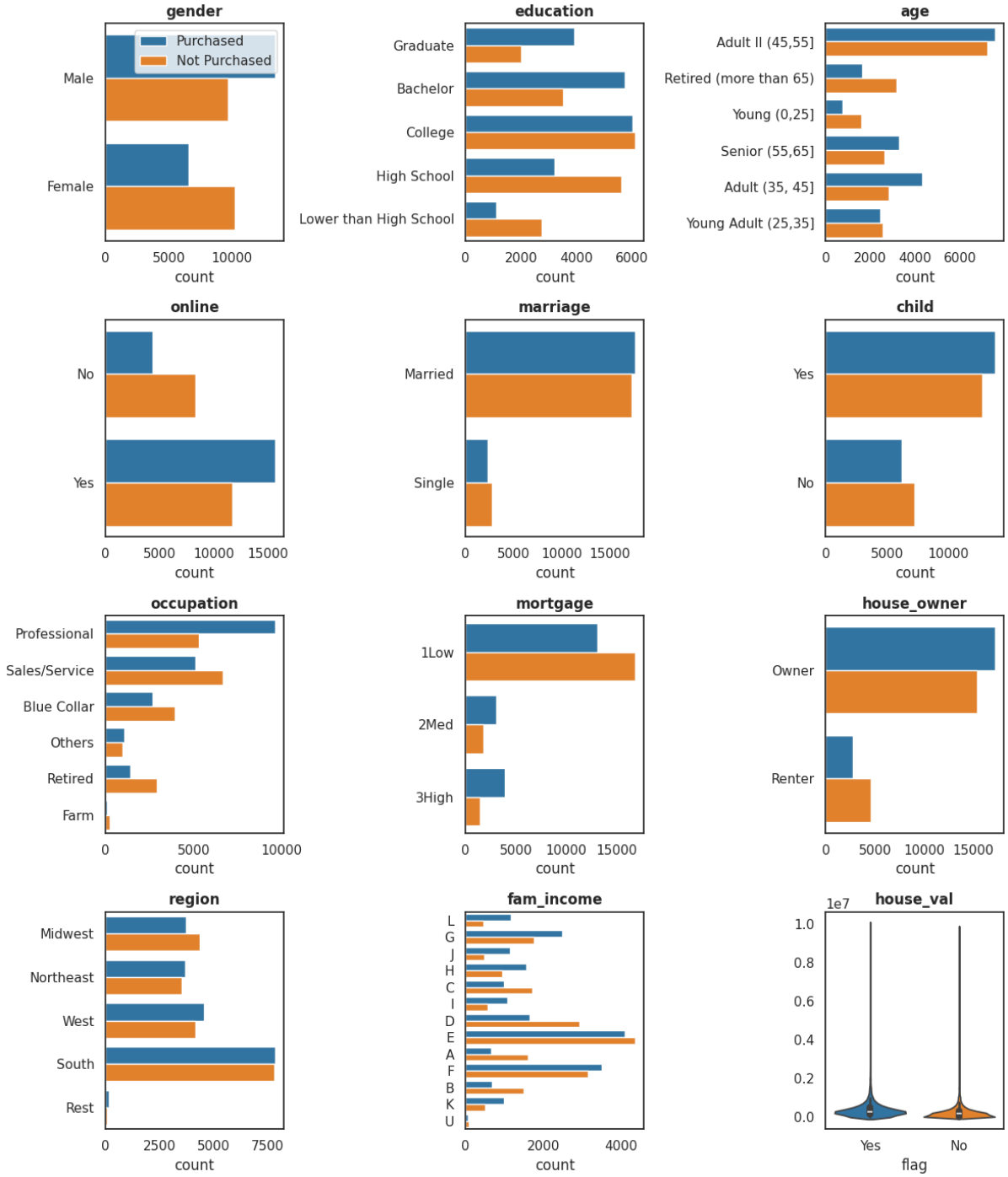
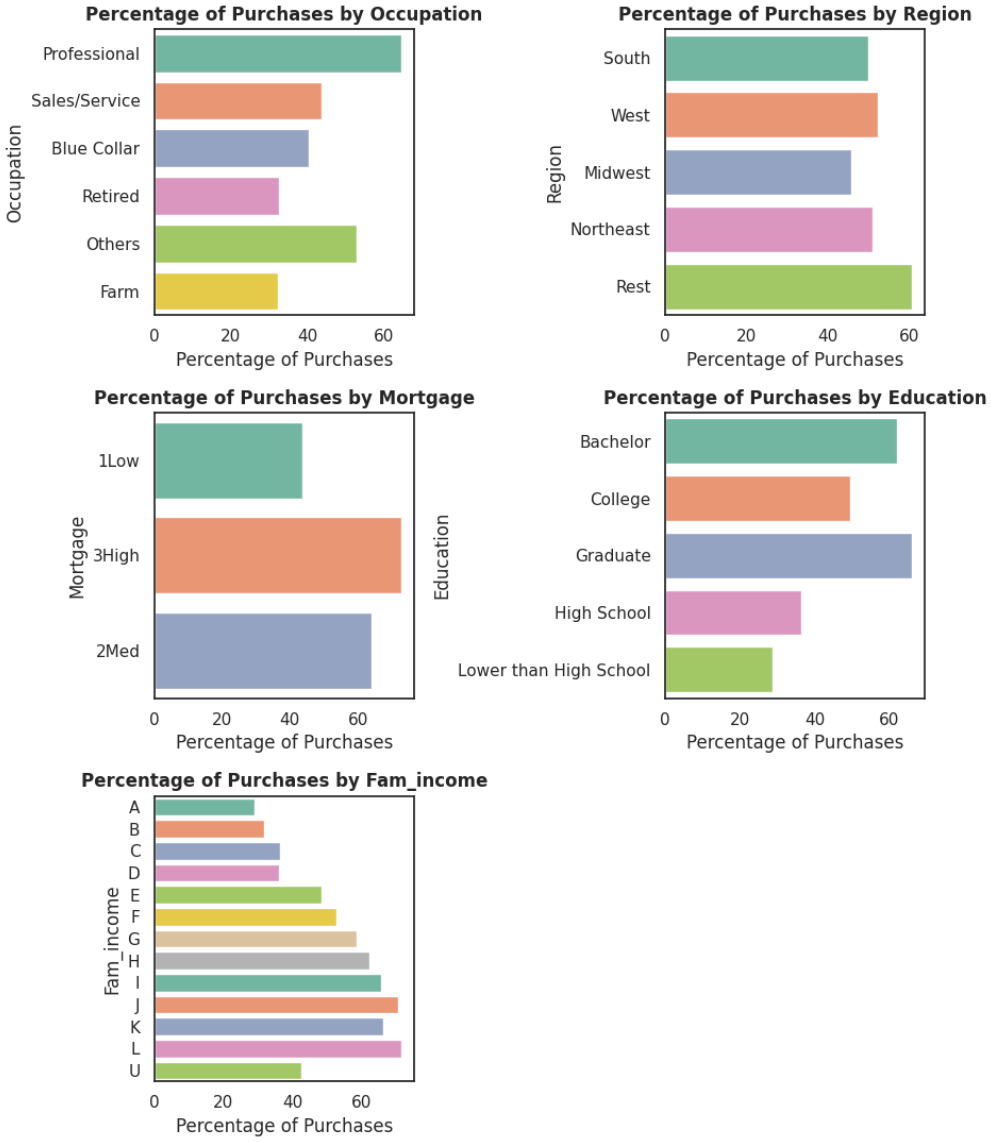


Figure 4. Descriptive analysis

Since the dataset comes from all customers of the insurance company, it is interesting to know what percentage of customers bought insurance in each category, and thus to know the conversion rate per category. The following variables present changes when comparing with the previous graphs.



In terms of occupation, it can be appreciated that professionals have the highest purchased rate with 64.5%, following by “Others” category. In terms of region, it was expected south as the category with the highest conversion rate. However, 60% of customers from the “Rest” category purchased the life insurance product, followed by west (52,3%) and Northeast (51,16%). The highest conversion rate is found in the "3High" mortgage category which indicates the top mortgage level. In terms of educational level, those individuals with a graduate level have more than 66% purchase rate. In addition, individuals with a family income of L, J, I and K are the highest subscribers.

Supervised Machine Learning

In Python it is necessary to transform categorical variables before training a machine learning algorithm. Hence, ordinal variables with a meaningful order are encoded using mapping technique to preserve their ordinal nature. On the other hand, all categorical variables without a natural order, except from the target variable, are converted into dummy variables, by dropping the first column, to avoid multicollinearity. The target variable is encoded by replacing “Yes” by 1 and “No” by 0. Algorithms such as SVM and NN are sensitive to variables with different scales. Therefore, it is

common practice to normalize the dataset using the robust scaler from the “sklearn” library before building those models.

Then the data is split into 80% for training and 20% for testing and their resulting accuracy were presented in Table 3. In this section, hyperparameter tuning will be conducted to improve the predictive accuracy of the models. However, it's essential to note that accuracy alone may not fully capture the performance of classifier algorithms. Therefore, a comprehensive evaluation of their performance will include other important metrics beyond accuracy such as:

Confusion Matrix	Table that compares the actual labels of the data with the labels predicted by the model. For a binary classification problem is a 2x2 matrix.
Recall	Fraction of positives that were correctly classified.
AUC-ROC Curve	It is the best metric to evaluate and compare classifiers performance, it is a curve that present the trade-off between the true positives and false positives.

(Hossin and Sulaiman, 2015)

To ensure effective prevention of overfitting while assessing the model's ability to generalize the data, k-fold cross-validation with a k-value of 10 is employed. This method allows a robust assessment of performance both before and after fitting the hyperparameters.

Random Forest Classifier

The accuracy for the initial model was 0.65, for Random Forest there are four hyperparameters that can be tuning: number of trees, maximum depth of trees, number of estimators considered at each split and minimum samples required to split nodes.

Table 6. Hyperparameter tuning random forest

Tried hyperparameters	Best performance
n of estimators’: 50, 100, 150	100
‘**max depth**: None, 5, 15	15
‘**min samples split’**: 2, 4, 6	2
‘**min samples leaf’**: 4, 5, 6	6

The following is the classification report for the model with the best performance.

Table 7. Performance report random forest

	precision	recall	f1-score	support
0	0.70	0.67	0.68	4017
1	0.68	0.71	0.70	3983
accuracy			0.69	8000
macro avg	0.69	0.69	0.69	8000
weighted avg	0.69	0.69	0.69	8000

The performance metrics demonstrates that the model has balanced performance across both classes. This conclusion is drawn from several observations. Regarding precision, the model shows a slightly better performance for class 0, with a precision of 0.70, compared to class 1. This indicates that when the model predicts an instance as belonging to class 0, it is slightly more likely to be correct than when it predicts an instance as belonging to class 1. In terms of recall for class 1 is higher (0.71) than that for class 0 (0.67), suggesting that the model is more effective at identifying true positives for class 1.

Support Vector Machine

The initial model had an accuracy of 0.68, as detailed in the methodology section. For the Support Vector Machine (SVM) algorithm, the kernel and C parameters can be tuned; additionally, if the kernel is nonlinear, the gamma parameter can be adjusted. Feature selection is crucial for SVM, as removing important features may increase accuracy, while adding irrelevant features can increase noise and reduce accuracy (Patle and Chouhan, 2013). For this dataset, "sklearn.feature_selection.RFECV" is used for feature selection. However, the results suggest using all features to build the model.

```
Optimal number of features: 26
Selected features: [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True
  True  True]
```

Figure 5. Feature selection SVM

Table 8. Hyperparameter tunning SVM

Tried hyperparameters	**Best performance**
'**kernel'**: linear, rbf, poly, sigmoid	linear
'**C'**: 0.01, 1, 5, 10	0.01
'gamma': 0.1, 0.01,'auto', 'scale'	

While Random Forest shows a 0.4 increase in accuracy after hyperparameter tuning, the accuracy of SVM remains almost unchanged. The classification report of the best performing model after hyperparameter tuning is shown below.

Table 9. Performance report SVM

	precision	recall	f1-score	support
0	0.67	0.69	0.68	3987
1	0.68	0.67	0.68	4013
accuracy			0.68	8000
macro avg	0.68	0.68	0.68	8000
weighted avg	0.68	0.68	0.68	8000

The performance metrics of the SVM model indicate a balanced performance across both classes too. With regards to precision, the model is marginally more likely to be accurate when predicting an instance as belonging to class 1 than it is for class 0. When looking at recall, the SVM model is more capable of identifying true positives within class 0 compared to class 1. Overall, the accuracy of the model stands at 0.68 for the dataset, highlighting a consistent level of performance across both precision and recall metrics for the two classes.

5. Results and Discussion

The results from k-fold cross-validation for both explained models demonstrate consistency, with similar average scores and low standard deviations. This suggests that both models possess strong generalization capabilities across the dataset.

Table 10. Cross-validation

K = 10		**Random Forest**	**Support Vector Machine**
Initial model	**Average accuracy**	0.648	0.679
	Standard deviation	0.008	0.006
After hyperparameter tuning	**Average accuracy**	0.686	0.684
	Standard deviation	0.006	0.006

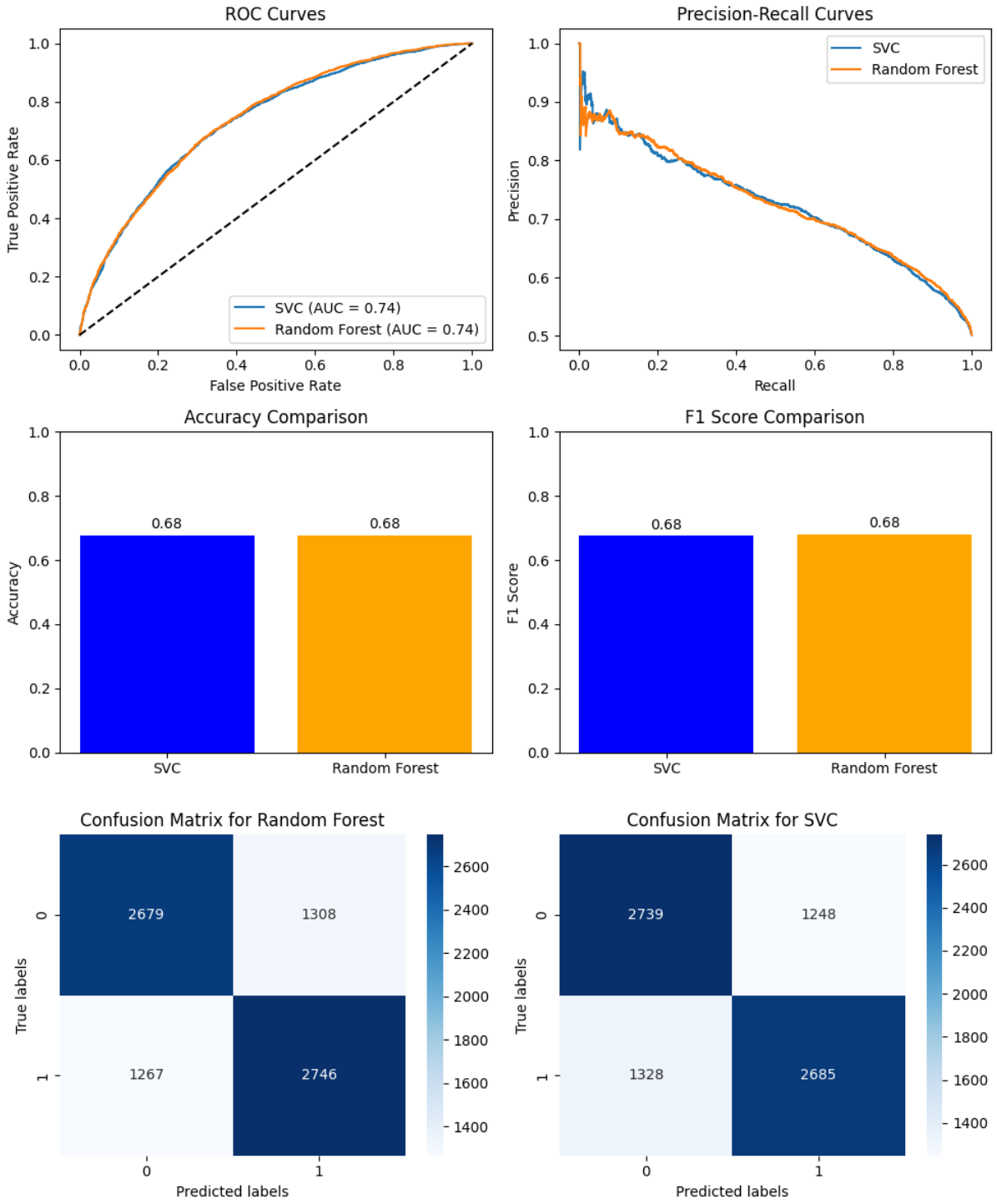


Figure 6. Performance Comparison

Comparing the performance of Random Forest and SVM after hyperparameter tuning, both models achieved similar overall metrics with an accuracy of 0.68. However, Random Forest shows a slightly better ability to identify class 1 than SVM, which is marginally more effective at detecting class 0. Given the main objective SVM model is slightly better for identifying customers likely to subscribe.

The accuracy can be compared with others researches such as: He, Y., Xiong, Y. and Tsai, Y. (2020) with the same objective but for all products offered by an insurance company, obtained an accuracy after hyperparameter tunning of 0.66 for Random Forest and 0.63 for SVM.

Feature Importance

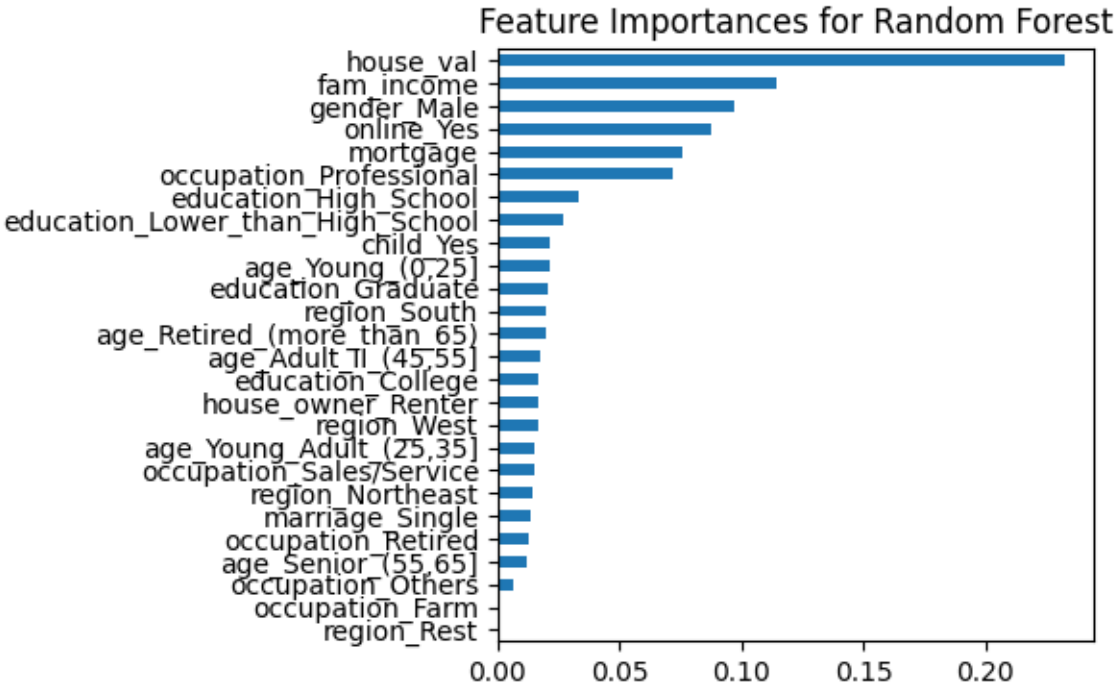


Figure 7. Features Importance Random Forest

The feature importance results for the random forest classifier show the significance of the variables in predicting purchases of the life insurance product. 'house_val' is the most influential, hence the value of a house has the greatest impact on purchased prediction. Other notable features include 'mortgage', 'fam_income', and 'gender_Male', illustrating the importance of financial stability family income, and gender. Lesser but still significant features relate to online purchases or professional occupation, among others. All variables contribute in some way to the model's decision-making process. For effective predictions using this Random Forest model is crucial knowing the variables of greatest importance.

Regarding SVM, feature importance is shown in the graph, similar to how it works in Random Forest; knowing the most important features of possible future customers is essential for effective predictions. Moreover, the graph confirms that all features are meaningful for the model, indicating the feature selection process was appropriate.

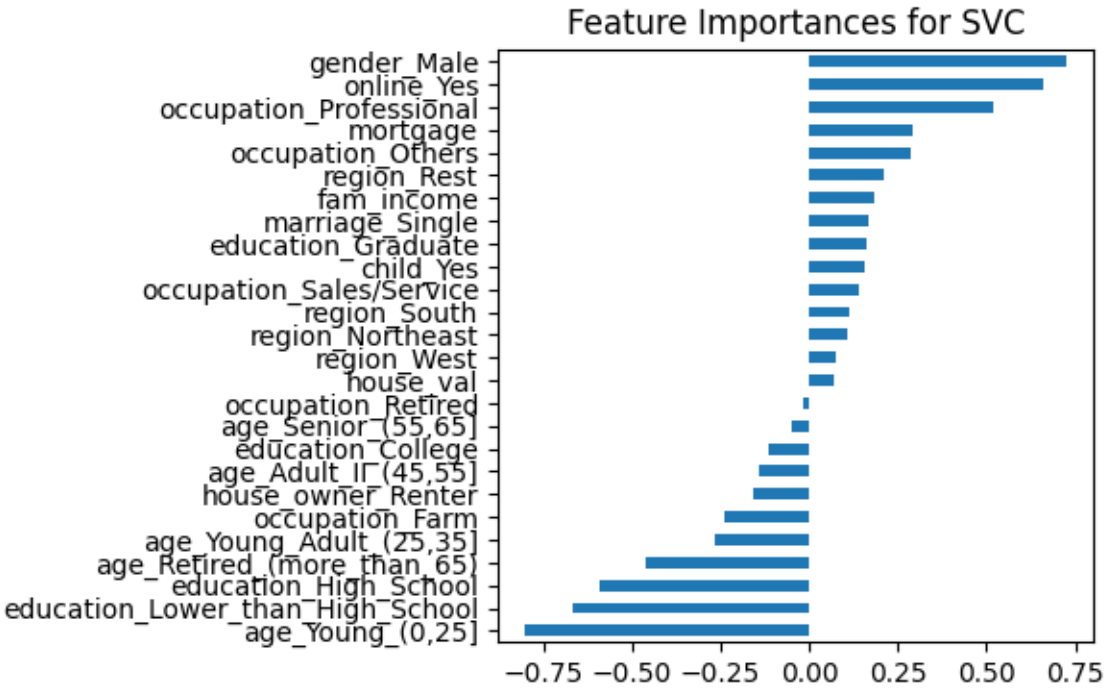


Figure 8. Features Importance SVM classifier

6. Conclusion and Recommendations

From the data analysis, it can be discovered that the insurance company should focus on targeting male customers, particularly those who are married, in the adult to middle-age category, and have higher education levels. Professionals show the highest purchase rate, suggesting occupation-based segmentation could be effective. Additionally, digital engagement strategies should be enhanced as online transactions are prevalent. Tailoring products to these demographics and preferences may improve conversion rates and customer acquisition. In addition, exploring the 'Others' and 'Rest' categories in occupation and region, which show high conversion rates, could reveal new market segments.

In terms of the performance of the analysed machine learning models, both exhibit an accuracy of 0.68. Moreover, after hyperparameter tuning, Neural Network also achieves this accuracy. Given business needs, these models can serve as baselines. However, if performance needs to be improved, data collection should be modified with a focus on addressing missing values and correcting inaccurate observations.

With the same purpose but with a different approach by doing k-means to segments the customers before building machine learning models, Kingawa and Hailu (2022), obtained an accuracy of 98,8% for a Deep Neural Network. This approach may be considered to improve model performance, since building predictive models for each segment obtained with k-means can result in better models because they are tailored to the specific characteristics of each segment.

7. References

Acuña, E., Rodriguez, C. (2004). The Treatment of Missing Values and its Effect on Classifier Accuracy. In: Banks, D., McMorris, F.R., Arabie, P., Gaul, W. (eds) Classification, Clustering, and Data Mining Applications. Studies in Classification, Data Analysis, and Knowledge Organisation. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17103-1_60 Available at: https://sci2s.ugr.es/sites/default/files/files/TematicWebSites/MVDM/IFCS04r.pdf Accessed: 09 February 2024.

Chai, C.P. (2020) 'The importance of data cleaning: Three visualization examples', CHANCE, 33(1), pp. 4–9. doi:10.1080/09332480.2020.1726112. Available at: https://www.tandfonline.com/doi/full/10.1080/09332480.2020.1726112?casa_token=CsRZRAVr-iAAAAAA%3AVvpFqNZXHFt22EolGbnIMSsihW0GozGUog3smoBV6LLuYXEvQ47ete-mW5A1as6bC6yjNew0vlA (Accessed: 08 February 2024).

Faisal, S. and Tutz, G. (2022) 'Nearest neighbor imputation for categorical data by weighting of attributes', Information Sciences, 592, pp. 306–319. doi:10.1016/j.ins.2022.01.056. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0020025522000895 . Accessed: 09 February 2024.

Fontinelle, A. (2023) Life insurance: What it is, how it works, and how to buy a policy, Investopedia. Available at: https://www.investopedia.com/terms/l/lifeinsurance.asp (Accessed: 06 February 2024).

He, Y., Xiong, Y. and Tsai, Y. (2020) 'Machine learning based approaches to predict customer churn for an insurance company', 2020 Systems and Information Engineering Design Symposium (SIEDS) [Preprint]. doi:10.1109/sieds49339.2020.9106691. Available at: https://ieeexplore.ieee.org/abstract/document/9106691 (Accessed: 02 March 2024)

Hearst, M.A. et al. (2002) 'Support Vector Machines', IEEE Xplore , 13(4), pp. 18–28. doi:10.1142/9789812776655_0002. Available at: https://ieeexplore.ieee.org/document/708428/citations#citations Accesed: 15 February 2024

Hossin, M. and Sulaiman, M.N. (2015) 'A review on evaluation metrics for data classification evaluations', International Journal of Data Mining & Knowledge Management Process, 5(2), pp. 1–10. doi:10.5121/ijdkp.2015.5201. Available at: https://www.academia.edu/download/37219940/5215ijdkp01.pdf. Accessed: (17 February 2024)

James, G. et al. (2023) An introduction to statistical learning: With applications in Python. Cham, Switzerland: Springer. Available at: https://www.statlearning.com (Accessed: 11 February 2024).

Kingawa, E.D. and Hailu, T.T. (2022) 'Customer churn prediction using machine learning techniques: The case of lion insurance', Asian Journal of Basic Science & Research, 04(04), pp. 60–73. doi:10.38177/ajbsr.2022.4407. Available at: https://www.academia.edu/download/98942839/74687.pdf (Accessed: 02 March 2024)

Press, G. (2023) Cleaning Big Data: Most time-consuming, least enjoyable data science task, survey says, Forbes. Available at: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/ (Accessed: 11 February 2024).

Shamsuddin, S.N., Ismail, N. and Nur-Firyal, R. (2023) 'Life insurance prediction and its sustainability using machine learning approach', Sustainability, 15(13). doi:10.3390/su151310737. Available at: https://www.researchgate.net/publication/372238469_Life_Insurance_Prediction_and_Its_Sustaina

bility_Using_Machine_Learning_Approach/link/64abf4f495bbbe0c6e25561c/download?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19 (Accessed: 06 February 2024)

Verhoef, P.C. and Donkers, B. (2001) 'Predicting customer potential value an application in the insurance industry', Decision Support Systems, 32(2), pp. 189–199. doi:10.1016/s0167-9236(01)00110-5 (Accessed: 06 February 2024).

Wirth, R. and Hipp, J. (2000) *CRISP-DM: Towards a Standard Process Model for Data Mining* [Online]. Available at: https://www.cs.unibo.it/~danilo.montesi/CBD/Beatriz/10.1.1.198.5133.pdf (Accessed: 06 February 2024).

Breiman, L. (2001) 'Random Forests', Kluwer Academic Publishers, pp. 5–32. Available at: https://link.springer.com/article/10.1023/A:1010933404324 (Accessed: 02 March 2024)

Abdi, H., Valentin, D. and Edelman, B. (1999) Neural networks. Thousand Oaks, CA: Sage Publications. pag. (1 -2).

Patle, A. and Chouhan, D.S. (2013) 'SVM kernel functions for classification', 2013 International Conference on Advances in Technology and Engineering (ICATE) [Online]. doi:10.1109/icadte.2013.6524743. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6524743. (Accessed: 03 March 2024)

Appendix I

Initial accuracy Neural Network = 0.680

After hyperparameter tunning = 0.685

Appendix II

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
import numpy as np

path = '/content/drive/MyDrive/data mining/sales_data.csv'
data = pd.read_csv(path)

data.info()

categorical_summary = {}

for column in data.select_dtypes(include=['object']):
    value_counts = data[column].value_counts()
    missing_values = data[column].isnull().sum()
    categorical_summary[column] = {"Value Counts": value_counts, "Missing Values": missing_values}

print(categorical_summary)

# first graphs

sns.set(style="whitegrid")
plt.figure(figsize=(12, 14))
```

```
categorical_columns = [col for col in data.columns if data[col].dtype == 'object']
numerical_columns = ['house_val']
rows, cols = 5, 3
for index, column in enumerate(categorical_columns):
    plt.subplot(rows, cols, index + 1)
    sns.countplot(y=column, data=data, palette="Set2")
    plt.title(f'{column}', fontweight='bold')
    plt.tight_layout()
plt.subplot(rows, cols, len(categorical_columns) + 1)
sns.violinplot(y=data['house_val'], color="skyblue")
plt.title(f'{numerical_columns[0]}', fontweight='bold')
plt.tight_layout()
plt.show()


## Data cleaning: Incorrect Data, Unkown values and missing values
def replace (data, column, old_value, new_value):
  data[column] = data[column].replace(old_value, new_value)
  return data
replace(data, "education", '0. <HS', 'Lower than High School')
replace(data, "education", '1. HS', 'High School')
replace(data, "education", '2. Some College', 'College')
replace(data, "education", '3. Bach', 'Bachelor' )
replace(data, "education", '4. Grad', 'Graduate' )
replace(data, "gender", 'M', 'Male')
replace(data, "gender", 'F', 'Female')
replace(data, "gender", 'U', 'Unknown')
replace(data, "age", '2_<=25', 'Young (0,25]')
replace(data, "age", '3_<=35', 'Young Adult (25,35]')
replace(data, "age", '4_<=45', 'Adult (35, 45]')
replace(data, "age", '5_<=55', 'Adult II (45,55]')
```

```
replace(data, "age", '6_<=65', 'Senior (55,65]')
replace(data, "age", '7_>65', 'Retired (more than 65)')
replace(data, "age", '1_Unk', 'Unknown')
replace(data, "online", "Y", "Yes")
replace(data, "online", "N", "No")
replace(data, "child", "Y", "Yes")
replace(data, "child", '0', 'No')
replace(data, "child", 'U', 'Unknown')
replace(data, "child", 'N', 'No')
replace(data, "flag", "Y", "Yes")
replace(data, "flag", "N", "No")
replace(data, "fam_income", "U", "Unknown")
data = data.replace('Unknown', np.nan)
variables = ['gender','age','child','education', 'marriage', 'house_owner', 'fam_income']
moda_imputer = SimpleImputer(strategy='most_frequent')
data[variables] = moda_imputer.fit_transform(data[variables])
data.info()
data.isnull().sum()
100 * data.isnull().sum() / len(data)


## DESCRIPTIVE ANALYSIS DATASET
from sklearn.preprocessing import LabelEncoder
from scipy.stats import chi2_contingency

## correlations
variables = ['flag', 'gender', 'online', 'marriage','child', 'house_owner']
data_encoded = pd.DataFrame()
le = LabelEncoder()
for var in variables:
    data_encoded[var] = le.fit_transform(data[var])
```

```
phi_coefficients = {}
for var in variables[1:]:
    crosstab = pd.crosstab(data_encoded[var], data_encoded['flag'])
    chi2, p, dof, expected = chi2_contingency(crosstab)
    phi = np.sqrt(chi2 / data.shape[0])
    phi_coefficients[var] = phi
phi_df = pd.DataFrame(list(phi_coefficients.items()), columns=['Variable', 'Phi Coefficient'])
print(phi_df)
from scipy.stats import pointbiserialr
data1 = data.copy()
data1['flag_encoded'] = data1['flag'].map({'Yes': 1, 'No': 0})
correlation, p_value = pointbiserialr(data1['flag_encoded'], data1['house_val'])
print(f"Correlación punto biserial: {correlation}")
print(f"Valor p: {p_value}")
variables = ['flag', 'fam_income']
data_encoded1 = pd.DataFrame()
le = LabelEncoder()
for var in variables:
    data_encoded1[var] = le.fit_transform(data[var])
correlation, p_value = pointbiserialr(data_encoded1['flag'], data_encoded1['fam_income'])
print(f"Correlación: {correlation}")
print(f"Valor p: {p_value}")

##graphs
sns.set(style="white")
plt.figure(figsize=(12, 14))
categorical_columns = [col for col in data.columns if data[col].dtype == 'object' and col != 'flag']
numerical_columns = ['house_val']
```

```
rows, cols = 4, 3
for index, column in enumerate(categorical_columns):
    ax = plt.subplot(rows, cols, index + 1)
    sns.countplot(y=column, data=data, palette="tab10", hue='flag')
    plt.title(f'{column}', fontweight='bold')
    ax.set_ylabel('')
    if index != 0:
        ax.get_legend().remove()
    else:
        handles, labels = ax.get_legend_handles_labels()
        new_labels = ['Purchased' if label == 'Yes' else 'Not Purchased' for label in labels]
        ax.legend(handles, new_labels)
    plt.tight_layout()
ax = plt.subplot(rows, cols, len(categorical_columns) + 1)
sns.violinplot(x='flag', y='house_val', data=data, palette="tab10")
plt.title(f'{numerical_columns[0]}', fontweight='bold')
ax.set_ylabel('')
plt.tight_layout()
handles, labels = ax.get_legend_handles_labels()
new_labels = ['Purchased' if label == 'Yes' else 'Not Purchased' for label in labels]
plt.show()


##percentajes
categorical_columns = [ 'occupation', 'region', 'mortgage', 'education', 'gender', 'online',
'house_owner', 'fam_income']  # etc.
conversion_rates = {}
for column in categorical_columns:
    contingency_table = pd.crosstab(data[column], data['flag'])
    conversion_rate = (contingency_table['Yes'] / (contingency_table['Yes'] +
contingency_table['No'])) * 100
    conversion_rates[column] = conversion_rate
```

```
conversion_df = pd.DataFrame(conversion_rates)
print(conversion_df)

## graphs conversion rate
sns.set(style="white")
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(9, 11))
axes = axes.flatten()
def plot_purchase_percentage_by_category_subplot(data, category, ax):
    total_per_category = data[category].value_counts()
    purchases_per_category = data[data['flag'] == "Yes"][category].value_counts()
    percentage_purchases = (purchases_per_category / total_per_category * 100).reset_index()
    percentage_purchases.columns = [category, 'Percentage of Purchases']
    sns.barplot(x='Percentage of Purchases', y=category, data=percentage_purchases, palette="Set2",
ax=ax)
    ax.set_title(f'Percentage of Purchases by {category.capitalize()}', fontweight='bold')
    ax.set_ylabel('')
    ax.set_xlabel('Percentage of Purchases')
    ax.set_ylabel(category.capitalize())
categories = ["occupation", "region", "mortgage", "education", "fam_income"]
for category, ax in zip(categories, axes[:5]):
    plot_purchase_percentage_by_category_subplot(data, category, ax)
axes[-1].axis('off')
plt.tight_layout()
plt.show()

## Models constructions
data.head()
data.shape
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
```

```
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import lightgbm as lgb
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import svm


## feature engineering and train and test data set

print(data['fam_income'].unique())
print(data['mortgage'].unique())

data['flag'].replace('Yes', 1, inplace = True)
data['flag'].replace('No', 0, inplace = True)

fam_income_mapping = {'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7, 'H': 8, 'I': 9, 'J': 10, 'K': 11, 'L': 12}
data['fam_income'].replace(fam_income_mapping, inplace=True)

mortgage_mapping = {'1Low': 1, '2Med': 2, '3High': 3}
data['mortgage'].replace(mortgage_mapping, inplace=True)

data_encoded = pd.get_dummies(data, drop_first=True)

data_encoded.columns = data_encoded.columns.str.replace(' ', '_')

X = data_encoded.drop('flag', axis=1)
y = data_encoded['flag']
```

```
print(X)


X_train, X_test, y_train, y_test =
train_test_split(X,y,train_size=0.80,test_size=0.20,random_state=404)


#comparing initial models


cols = ['Case','GradientDecent','Bagging','Random Forest','Logistic Regression']


resul = pd.DataFrame(columns=cols)

resul.set_index("Case",inplace=True)

resul.loc['Accuracy'] = [0,0,0,0]

resul.head()


gradientdecent   = SGDClassifier()

bagging   = BaggingClassifier()

randomforest    = RandomForestClassifier()

logistic    = LogisticRegression(solver='liblinear')


models = [gradientdecent,bagging,randomforest,logistic]


col = 0

for model in models:

   model.fit(X_train,y_train.values.ravel())

   resul.iloc[0,col] = model.score(X_test,y_test)

   col += 1


resul.head()
```

```
""" RANDOM FOREST"""

randomforest = RandomForestClassifier(random_state = 12, oob_score=True)

y_pred_1 = randomforest.fit(X_train, y_train).predict(X_test)
report_1 = classification_report(y_test, y_pred_1)
print("Performance Metrics Before Hyperparameter Tuning:")
print(report_1)

#Cross validation

from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

scores = cross_val_score(randomforest, X, y, cv=10)

print("Cross-Validation Scores:", scores)
print("Average Score:", scores.mean())
print("Standard Deviation:", scores.std())

#HYPERPARAMETER TUNNING

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import RobustScaler


pipeline = Pipeline([
    ('scaler', RobustScaler()),
```

```
    ('randomforest', RandomForestClassifier())
])


param_grid = {
    'randomforest__n_estimators': [50, 100, 150],
    'randomforest__max_depth': [None, 5 ,15],
    'randomforest__min_samples_split': [2, 4, 6],
    'randomforest__min_samples_leaf': [4, 5, 6]
}

grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train, y_train)

print("Best:", grid_search.best_params_)

best_model = grid_search.best_estimator_

random_forest = RandomForestClassifier(
    n_estimators=100,
    max_depth=15,
    min_samples_split=2,
    min_samples_leaf=6,
    random_state=78
)

random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)
report2 = classification_report(y_test, y_pred_rf)
print("\nPerformance Metrics After Hyperparameter Tuning:")
```

```
print(report2)

"""Cross validation after hyperparameter"""

scores = cross_val_score(random_forest, X, y, cv=10)

print("Cross-Validation Scores:", scores)
print("Average Score:", scores.mean())
print("Standard Deviation:", scores.std())



## SUPPORT VECTOR MACHINE


from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold

#feature selection

X_sub, _, y_sub, _ = train_test_split(X, y, test_size=0.8, stratify=y, random_state=42)


X_train_sub_scaled = scaler.fit_transform(X_sub)
X_sub_scaled = scaler.transform(X_sub)

clf = svm.SVC(kernel="linear")

rfecv = RFECV(estimator=clf, step=1, cv=StratifiedKFold(5), scoring='accuracy')

rfecv.fit(X_sub_scaled, y_sub)
```

```
print("Optimal number of features: %d" % rfecv.n_features_)


print('Selected features:', rfecv.support_)


## first model building

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_scaled = scaler.fit_transform(X)

clf = svm.SVC()

clf.fit(X_train_scaled, y_train)

y_pred = clf.predict(X_test_scaled)

print("Default hyperparameter values:")
print(clf.get_params())

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

#cross validation

scores1 = cross_val_score(clf, X_scaled, y, cv=10)
```

```
print("Cross-Validation Scores:", scores1)
print("Average Score:", scores1.mean())
print("Standard Deviation:", scores1.std())

#hyperparameter tunning

param_grid = {
    'kernel': [ 'linear', 'rbf', 'poly', 'sigmoid'],
    'C': [0.01,  1, 5, 10],
    'gamma': [0.1, 0.01,'auto', 'scale']
}

svc = SVC()

grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

print("Mejores hiperparámetros encontrados:")
print(grid_search.best_params_)

best_model = grid_search.best_estimator_

accuracy = best_model.score(X_test_scaled, y_test)
print("Precisión del mejor modelo en el conjunto de prueba:", accuracy)

#best model

best_kernel = 'linear'
best_C = 0.01
```

```
best_svc = SVC(kernel=best_kernel, C=best_C, random_state=404, probability=True)

best_svc.fit(X_train_scaled, y_train)

y_pred_4 = best_svc.predict(X_test_scaled)

print(classification_report(y_test, y_pred_4))

#cross validation

scores12 = cross_val_score(best_svc, X_scaled, y, cv=10)

print("Cross-Validation Scores:", scores12)
print("Average Score:", scores12.mean())
print("Standard Deviation:", scores12.std())



#models comparation

from sklearn.metrics import roc_curve, auc, precision_recall_curve, accuracy_score, f1_score

y_proba_svc = best_svc.predict_proba(X_test_scaled)[:, 1]
y_proba_rf = random_forest.predict_proba(X_test)[:, 1]

fpr_svc, tpr_svc, _ = roc_curve(y_test, y_proba_svc)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_proba_rf)

# Precision-Recall
precision_svc, recall_svc, _ = precision_recall_curve(y_test, y_proba_svc)
precision_rf, recall_rf, _ = precision_recall_curve(y_test, y_proba_rf)
```

```
# Accuracy y F1 Score
accuracy_svc = accuracy_score(y_test, y_pred_4)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_svc = f1_score(y_test, y_pred_4)
f1_rf = f1_score(y_test, y_pred_rf)

conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
conf_matrix_svc = confusion_matrix(y_test, y_pred_4)



models = ['SVC', 'Random Forest']
accuracies = [accuracy_svc, accuracy_rf]
f1_scores = [f1_svc, f1_rf]

fig, axs = plt.subplots(3, 2, figsize=(10, 12))

#  ROC
axs[0, 0].plot(fpr_svc, tpr_svc, label=f'SVC (AUC = {auc(fpr_svc, tpr_svc):.2f})')
axs[0, 0].plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {auc(fpr_rf, tpr_rf):.2f})')
axs[0, 0].plot([0, 1], [0, 1], 'k--', label='')
axs[0, 0].set_xlabel('False Positive Rate')
axs[0, 0].set_ylabel('True Positive Rate')
axs[0, 0].set_title('ROC Curves')
axs[0, 0].legend()

#  Precision-Recall
axs[0, 1].plot(recall_svc, precision_svc, label='SVC')
axs[0, 1].plot(recall_rf, precision_rf, label='Random Forest')
axs[0, 1].set_xlabel('Recall')
axs[0, 1].set_ylabel('Precision')
```

```
axs[0, 1].set_title('Precision-Recall Curves')
axs[0, 1].legend()

# Gráfico de barras de Accuracy
axs[1, 0].bar(models, accuracies, color=['blue', 'orange'])
axs[1, 0].set_title('Accuracy Comparison')
axs[1, 0].set_ylabel('Accuracy')
axs[1, 0].set_ylim([0, 1])
for i, acc in enumerate(accuracies):
    axs[1, 0].text(i, acc + 0.02, f'{acc:.2f}', ha='center')

# F1 Score
axs[1, 1].bar(models, f1_scores, color=['blue', 'orange'])
axs[1, 1].set_title('F1 Score Comparison')
axs[1, 1].set_ylabel('F1 Score')
axs[1, 1].set_ylim([0, 1])
for i, f1 in enumerate(f1_scores):
    axs[1, 1].text(i, f1 + 0.02, f'{f1:.2f}', ha='center')

# Confuision matrix
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap='Blues', ax=axs[2, 0])
axs[2, 0].set_title('Confusion Matrix for Random Forest')
axs[2, 0].set_xlabel('Predicted labels')
axs[2, 0].set_ylabel('True labels')

# Confuision matrix
sns.heatmap(conf_matrix_svc, annot=True, fmt="d", cmap='Blues', ax=axs[2, 1])
axs[2, 1].set_title('Confusion Matrix for SVC')
axs[2, 1].set_xlabel('Predicted labels')
axs[2, 1].set_ylabel('True labels')
```

```
plt.tight_layout()
plt.show()

#importance variables random forest

importances = pd.Series(random_forest.feature_importances_, index=X.columns)

print(importances)

importances_sorted = importances.sort_values()

plt.figure(figsize=(4, 4))
importances_sorted.plot(kind='barh')
plt.title('Feature Importances for Random Forest')
plt.show()

#importance variables SVM

column_names = X.columns



importances2 = pd.Series(best_svc.coef_[0], index=column_names)
importances2_sorted = importances2.sort_values()

plt.figure(figsize=(4, 4))
importances2_sorted.plot(kind='barh')
plt.title('Feature Importances for SVC')
plt.show()

print(importances2)
```

```
## neural network

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

#first model

model = Sequential([
   Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
   Dropout(0.5),
   Dense(64, activation='relu'),
   Dropout(0.5),
   Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
            loss='binary_crossentropy',
            metrics=['accuracy'])

model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_data=(X_test_scaled,
y_test))

loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f'Loss: {loss}, Accuracy: {accuracy}')

#hyperparameter tuning

import kerastuner as kt

def build_model(hp):
```

```
    model = Sequential([
        Dense(hp.Int('units', min_value=32, max_value=512, step=32), activation='relu',
input_shape=(X_train_scaled.shape[1],)),
        Dropout(hp.Float('dropout', min_value=0.0, max_value=0.5, step=0.1)),
        Dense(hp.Int('units', min_value=32, max_value=512, step=32), activation='relu'),
        Dropout(hp.Float('dropout', min_value=0.0, max_value=0.5, step=0.1)),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model


tuner = kt.Hyperband(build_model,
                     objective='val_accuracy',
                     max_epochs=10,
                     directory='my_dir',
                     project_name='hola',
                     overwrite=True)


tuner.search(X_train_scaled, y_train, epochs=10, validation_data=(X_test_scaled, y_test))
```