DATA MINING MGT7216

Assignment 1: Muhammad Muneeb Ullah Ansari - 40426685

WORD COUNT: 2169

Contents

Introduction and Background

The insurance sector increasingly relies on data-driven decision-making to enhance customer targeting, product personalization, and risk assessment. Ansell et al. (2007) describe how advancements in data collection and processing technologies have facilitated the adoption of sophisticated analytical methods in insurance. Kuhn and Johnson (2013) further emphasize the importance of machine learning in extracting valuable insights from complex datasets, enabling more accurate predictions and strategic business decisions. Predicting customer behavior in the insurance market presents several challenges, including data heterogeneity, privacy concerns, and the dynamic nature of customer preferences. Brown and Mues (2012) discuss the complexities of dealing with diverse data sources and formats, while Smith (2012) highlights the ethical considerations and regulatory constraints associated with using personal data for predictive modeling. Ratcliffe and Mccarty (2003) underscore the difficulty of capturing and adapting to changing consumer behaviors over time. Hastie, Tibshirani, and Friedman (2009) provide a foundational understanding of these techniques and their application in statistical learning. Bishop (2006) and Goodfellow, Bengio, and Courville (2016) delve into the specifics of neural networks and deep learning, illustrating their potential in handling high-dimensional and nonlinear data patterns. Delen, Walker, and Kadam (2005) examine the use of decision trees and logistic regression in predicting customer retention and acquisition in the insurance sector. Ngai et al. (2009) explore the application of data mining techniques, including clustering and classification, to segment customers and predict purchase likelihood. Tsai and Wu (2008) focus on the role of neural networks in analyzing and forecasting customer behavior in life insurance. Barocas and Selbst (2016) discuss the potential for bias and discrimination in algorithmic decision-making, while Zarsky (2016) examines the transparency and accountability of machine learning systems. Pagallo (2011) reviews the legal framework governing data use in predictive modeling, emphasizing the need for compliance with data protection laws.

Methodology

In view of CRISP-DM, my methodological approach as been very iterative in terms of data understanding and modeling. I started off with a solid business understanding through relevant literature and understanding the data through pre-processing and exploratory analysis. After data preparation and going into modeling, two of the performed models gave similar results despite being algorithms with varying computational power. This caused a cycle of back and forth from data understanding to modeling each time certain changes being made until a change was made large enough to differentiate the results of the models on some rational.

Missing values are usually identified in the data through summary statistics as NAs. However in this dataset, missing values were represented as an empty space or “ ”. Hence if the dataset had no missing values is very misleading and it is only after careful observation that these spaces can be imputed with any relevant observation. In our case, “Unknown” seemed to be most fitting as it encapsulates the nature of both “error during recording” and “field not relevant”.

Apart from the usual descriptive statistics, I opted for calculating correlations between variables. This would give me an idea of roughly what to expect as the outcome of my supervised learning algorithms.

It also provides some direction as to what variables might be more significant to visualize in the exploratory analysis after data preprocessing had been done. For that purpose I decided to opt for one-hot coding and calculate the correlation between each generated column. The result is in the form of a table but relevant correlations are extracted to make meaning. Correlation with the single numeric predictor was also done which did not require one-hot coding.

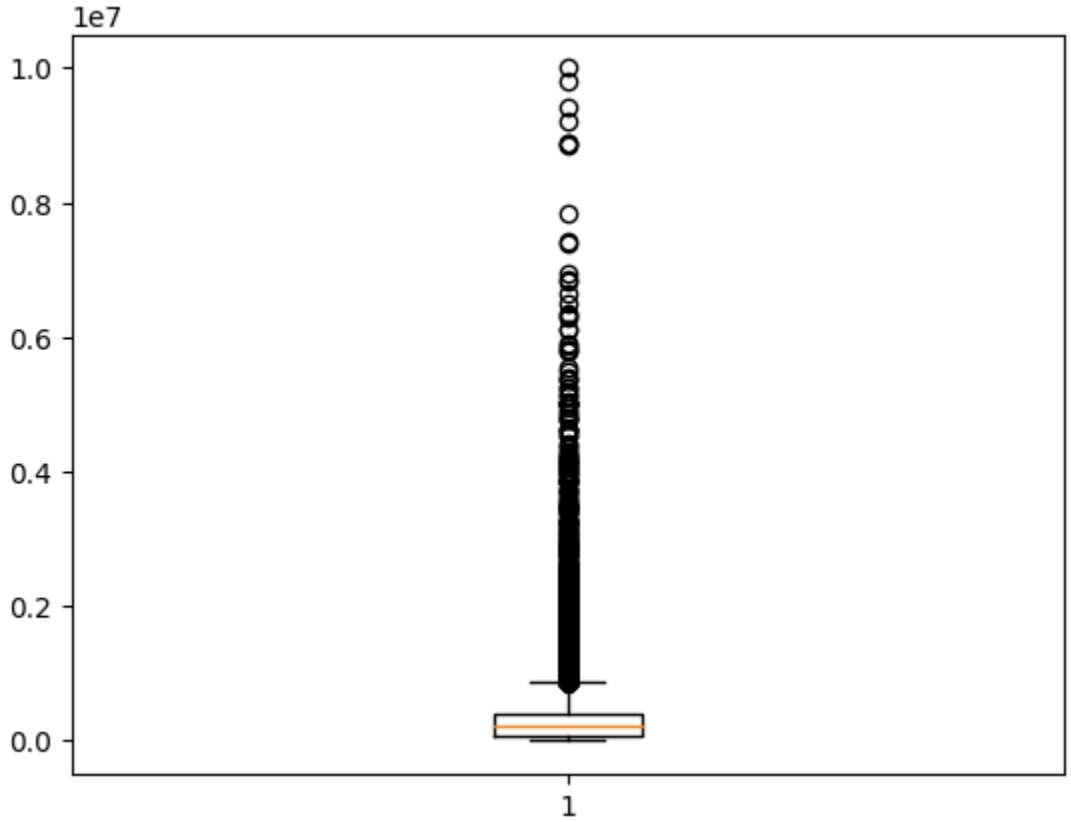
Visualizations were made keeping certain things in mind. During the preprocessing, house values displayed very extreme behavior with an unignorable amount of data points as outliers. This would make observing the distribution of house values very interesting. For someone interested to buy insurance, intuitively it makes sense that if they already don't have any financial obligations, they might be more willing to commit to an insurance provided that they do have a sizable portion of family income. This also combined with how valuable of a house they have makes the effect of the mortgage more prominent and is also something to shed light on. Whether more valuable house are owned by customers making purchases online or offline is also to take note of as this could indicate of what medium to opt for to reach the target audience.

Cutler et al. (2012) elaborates tinherent feature of bootstrapping and aggregating (bagging) in RF allows it to improve accuracy and control overfitting by averaging multiple decision trees' predictions. Hosmer, Lemeshow, and Sturdivant (2013) argue that LR is particularly effective for interpretability purposes, allowing researchers to understand the influence of each predictor on the outcome. Menard (2000) discusses how researchers conduct model diagnostics, including checking for multicollinearity through variance inflation factors (VIF) and assessing model fit. Breiman (2001) highlights the number of trees in the forest (n_estimators) and the number of features considered for splitting at each leaf node (max_features) as critical for ensuring model accuracy and preventing overfitting. Furthermore, Probst, Wright, and Boulesteix (2019) stress the importance of the min_samples_split and min_samples_leaf parameters, which control the minimum number of samples required to split an internal node and the minimum number of samples required to be at a leaf node, respectively. Genuer, Poggi, and Tuleau-Malot (2010) demonstrate how RF's built-in feature importance metrics can guide feature selection and dimensionality reduction, enhancing model interpretability and efficiency as detailed in Liaw and Wiener (2002).

Data Pre-Processing

The following were the data quality issues:

- Many extreme values in house_val, removed 0.33% of those values which were very far off and were not visible cluttered in a cluster.



- Levels in education were incorrectly named.

```
      education
        4. Grad
        3. Bach
2. Some College
2. Some College
2. Some College
```

- NaN in education were dropped.

```
      education
        4. Grad
        3. Bach
2. Some College
2. Some College
2. Some College
```

- Levels in age were incorrectly named.

```
   age
 1_Unk
 7_>65
2_<=25
2_<=25
 1_Unk
```

- Levels in mortgage were incorrectly named.

```
mortgage
    1Low
    1Low
    1Low
    1Low
    1Low
```

- Missing values in house_owner and marriage were removed.

```
house_owner
        NaN
      Owner
      Owner
        NaN
        NaN
```

```
marriage
     NaN
     NaN
     NaN
  Single
 Married
```

- Individual variables were converted to factors.

```
# Convert individual variables to categorical types
categorical_columns = ['flag', 'gender', 'education', 'age', 'online', 'marriage',
                       'child', 'occupation', 'mortgage', 'house_owner', 'region', 'fam_income']
for col in categorical_columns:
    data[col] = data[col].astype('category')
```

Data Analytics

Looking at the summary for house_val, it revealed that it had a very high mean which was pushed ahead by outliers even after a significant chunk had been removed.

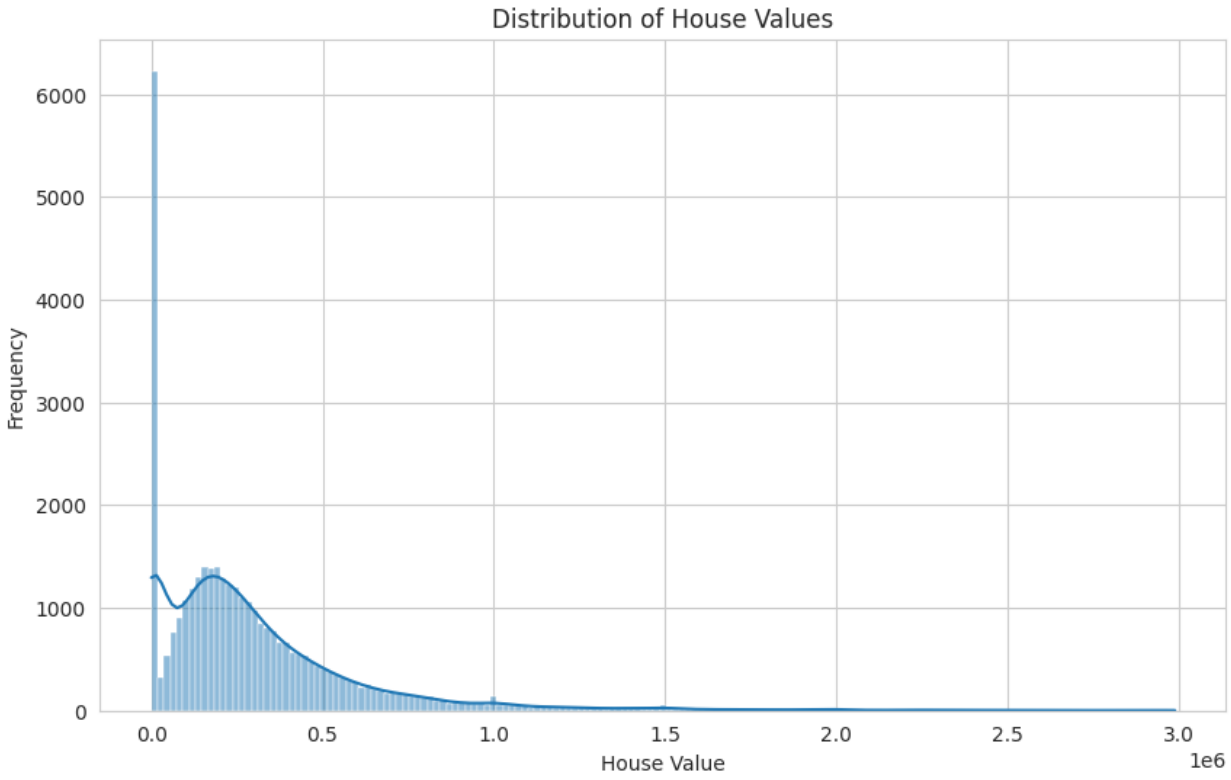
	house_val
count	4.000000e+04
mean	3.072138e+05
std	4.222146e+05
min	0.000000e+00
25%	8.065725e+04
50%	2.148720e+05
75%	3.937620e+05
max	9.999999e+06

After the data preprocessing, the data still almost had an equal sampling of the target variable. Majority of the clients owned their house, had a low mortgage and where married. When calculating the correlations of house_val with other variables, it was found out that it was slightly positively related to whether they had a high mortgage, whether they had an L income and whether they were a professional by occupation. It was also found out that it was slightly negatively correlated with whether the customer had a low mortgage and whether they rented their home.

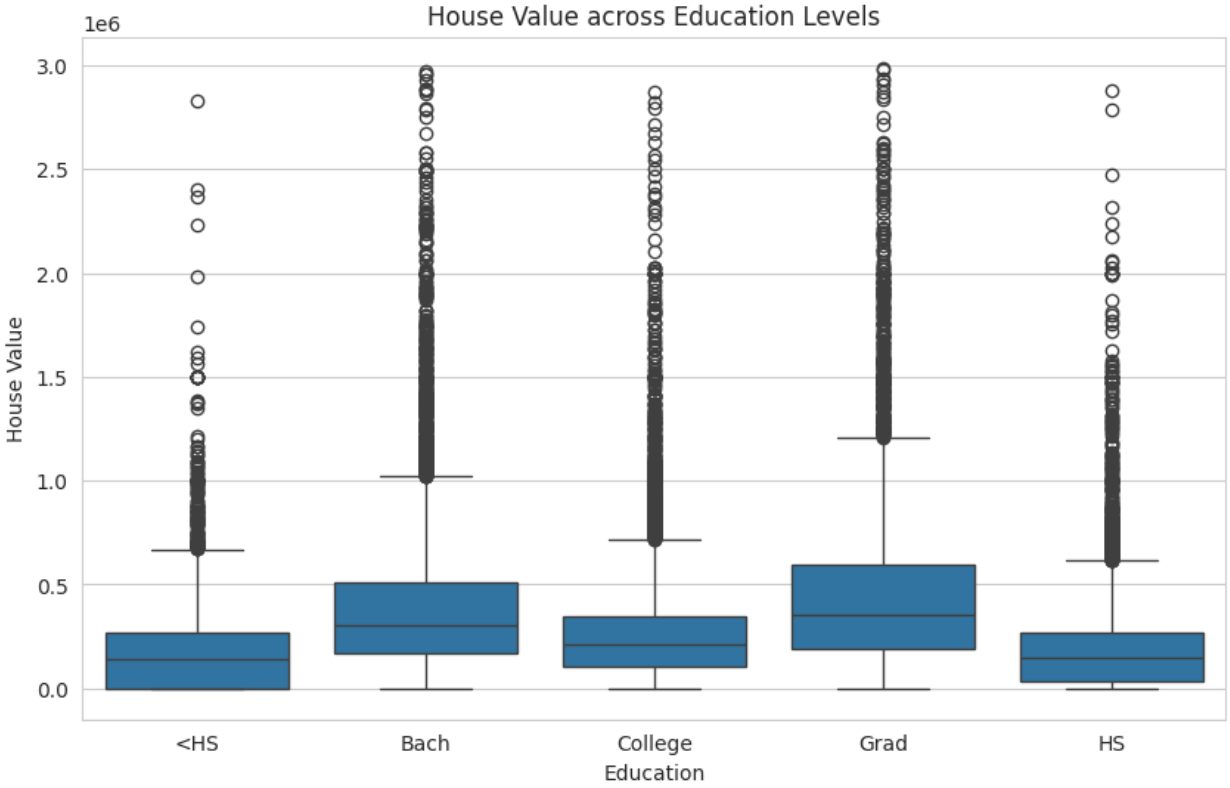
Top correlations among all other variables revealed important patterns. If a customer's age is +60, they are most probably retired. If a customer is single, there is a slight chance that they might not own the house and is a renter. Also there is also some evidence to suggest that if a customer rents the house, they might have a higher chance of having a lower mortgage.

Variable 1	Variable 2	Correlation	Positive
occupation_Retired	age_>65	0.775473	True
mortgage_Med	mortgage_Low	-0.628827	False
occupation_Sales/Service	occupation_Professional	-0.503238	False
child_Y	child_U	-0.463283	False
region_West	region_South	-0.419938	False
region_South	region_Northeast	-0.378369	False
fam_income_L	house_val	0.375801	True
mortgage_Low	house_val	-0.359412	False
education_College	education_Bach	-0.357800	False
house_owner_Renter	marriage_Single	0.351639	True
education_HS	education_College	-0.344436	False
house_owner_Renter	mortgage_Low	0.310046	True
education_HS	education_Bach	-0.301531	False
occupation_Professional	house_val	0.285189	True
occupation_Retired	occupation_Professional	-0.279089	False

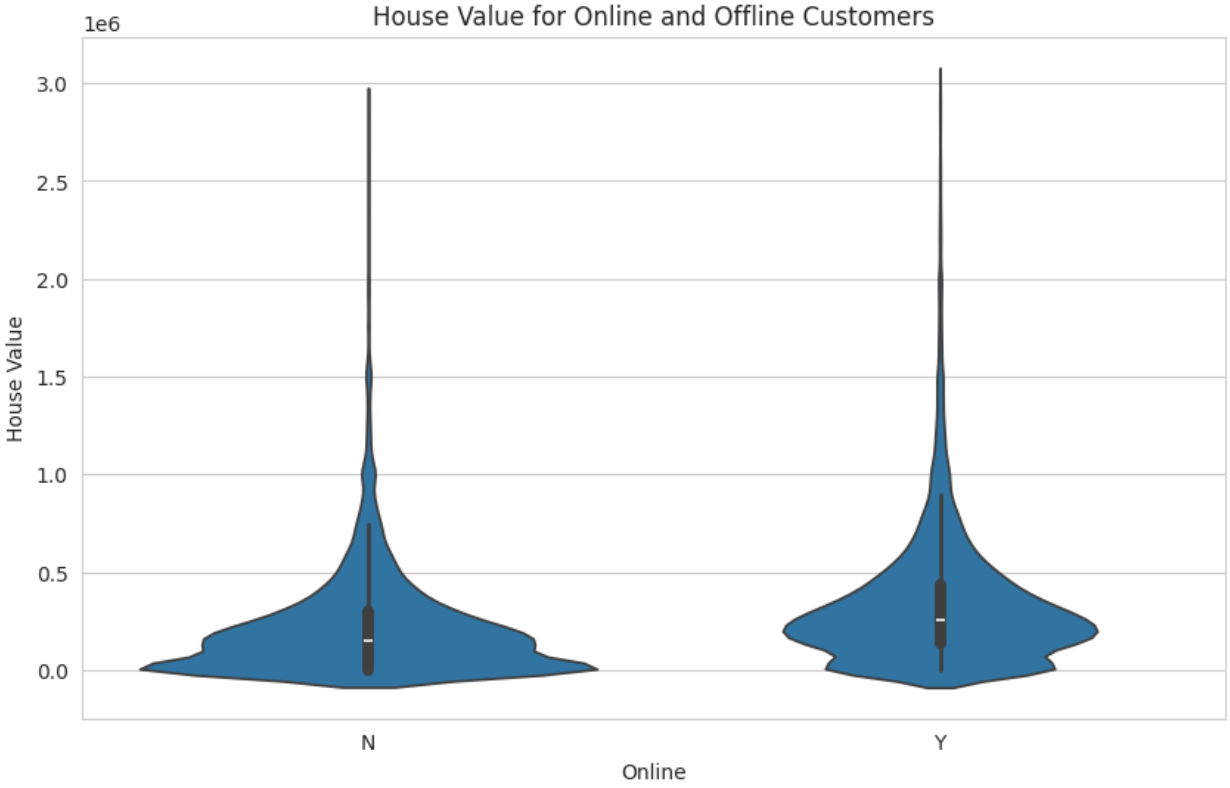
House values follow a distribution which is normal after a certain house value, but even then the curve is highly skewed. This is accounted due to the outliers which cannot be removed. We also see an abnormal peak in the curve which indicates that the values represented on the curves are outliers and the abnormal peak if closed is the actual distribution of house values if outliers are removed.



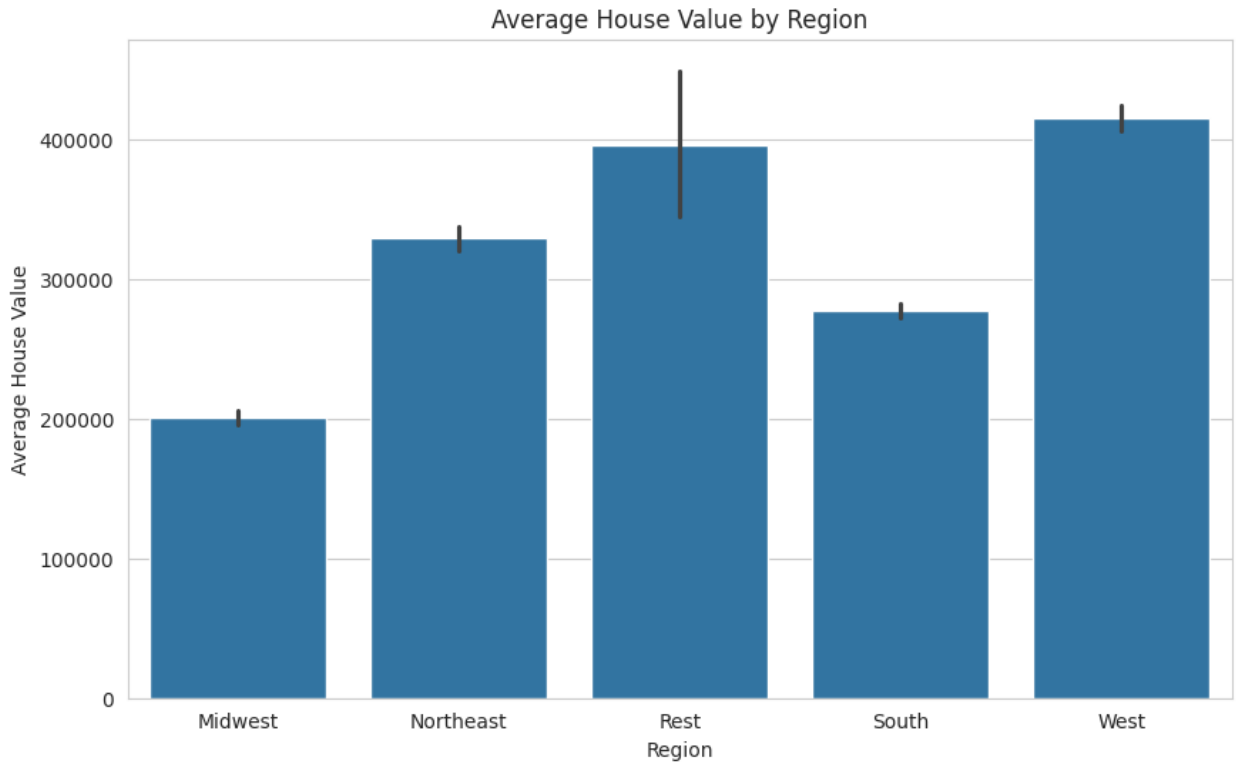
For each education level, we see that even though the range of house value is approximately similar due to outliers, Grads tend to have a higher house value followed by Bach. Even though Grads are not in the majority of the database, Bach comes in at a close second. Hence Bach might be an attractive target audience provided that they are more likely to convert.



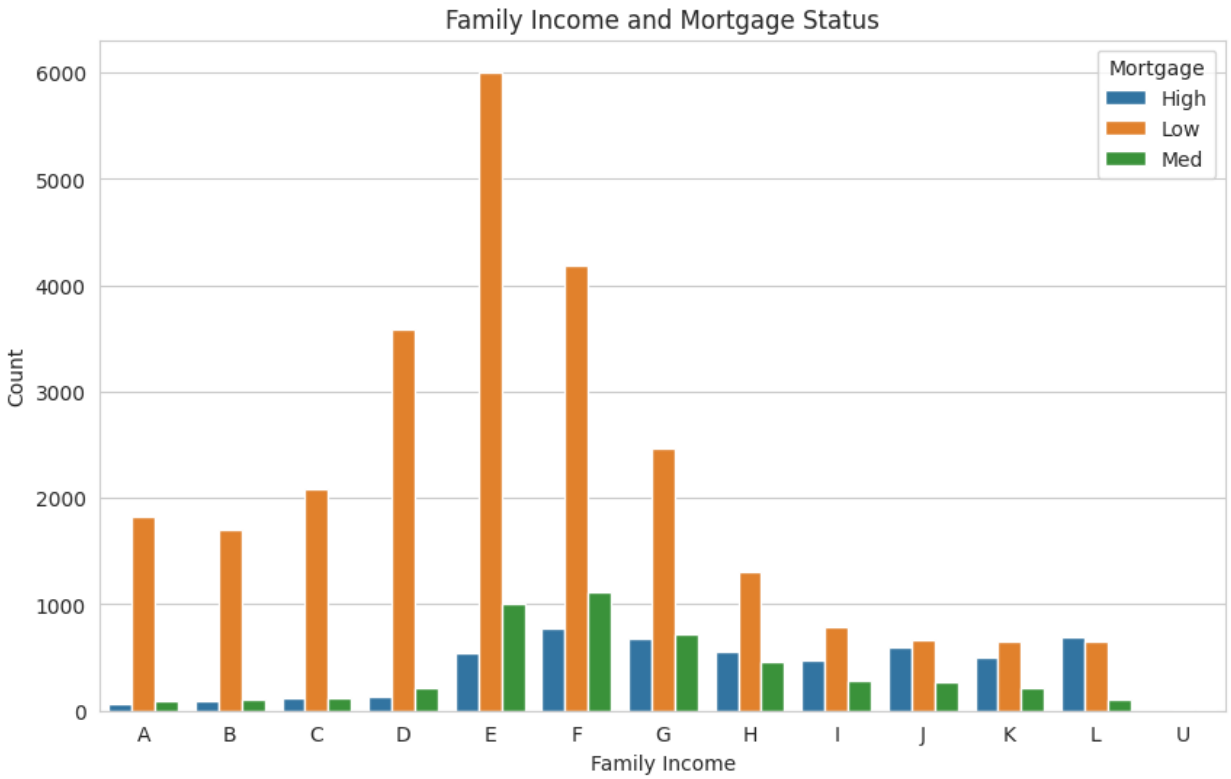
There is significant variability in house values among customers, as indicated by the extensive spread of the violins. This suggests that there are both lower-value and higher-value homes owned by customers across the education spectrum and regardless of their online status. The presence of wider parts of the violin at the bottom suggests a higher density of customers with lower house values.



The black lines at the top of each bar likely represent the error bars, which could indicate the range of the confidence interval for the mean estimate. The presence of these error bars suggests that there's variability around the mean house value estimates, which is natural in real-world data due to factors like heterogeneity in housing markets within each region. However interestingly this error for Rest Region is significantly large for all other regions. This implies that even though the rest of the regions don’t have as many counts of observations, some of them might have house values far exceeding any dominant region.



The following combined bar chart suggest that middle-income families are more likely to have higher mortgages, perhaps reflecting a middle-class squeeze or a trend where middle-income earners are more likely to invest in property that stretches their financial capabilities. Lower-income families might have less access to mortgage credit, resulting in fewer high mortgages. Higher-income families seem to have lower relative counts, which might imply that they either have no mortgage due to higher outright property ownership or they are less in number compared to the middle-income brackets.



Supervised Machine Learning

In total, we have formed 4 models. 2 for Random Forest and 2 for Logistic Regression. Each of these models was cross validated 5 times to ensure best model selection. For Random Forest, we chose to tune max depth of the trees to prevent model from becoming too overly complex and fitting to the noise in the data. We also controlled the number of trees because we wanted more trees as the model will become better at generalizing the data by averaging out biases and reducing variance.

For Logistic Regression, we opted for varying regularization strength to see results for models of varying complexity. We also varied the norm of penalization being l1 “lasso regression” and l2 “ridge regression” which penalizes coefficients in the model depending on how significant they are.

Summarizing the outcomes of each of the models, the logistic regression performed significantly well after tuning of hyperparameters. This was in terms of increased predictive accuracy from 55% to 69% as well as better precision and f1 score but a slightly lower recall.

Logistic Regression Model Before Tuning

```
Logistic Regression Model without Parameter Tuning
Accuracy: 0.547448647643833
              precision    recall  f1-score   support

           0       0.65      0.21      0.32      5396
           1       0.53      0.89      0.66      5363

    accuracy                           0.55     10759
   macro avg       0.59      0.55      0.49     10759
weighted avg       0.59      0.55      0.49     10759

[[1140 4256]
 [ 613 4750]]
```

Logistic Regression Model After Tuning

```
Logistic Regression Accuracy: 0.6867738637419835
              precision    recall  f1-score   support

           0       0.69      0.68      0.68      5396
           1       0.68      0.70      0.69      5363

    accuracy                           0.69     10759
   macro avg       0.69      0.69      0.69     10759
weighted avg       0.69      0.69      0.69     10759

[[3656 1740]
 [1630 3733]]
```

The coefficients after tuning increased in sizes for example gender_M from 2.1e-12 to 5.7e-01 and this changed the order of the top predictors by size.

Logistic Regression Coefficients Before Tuning

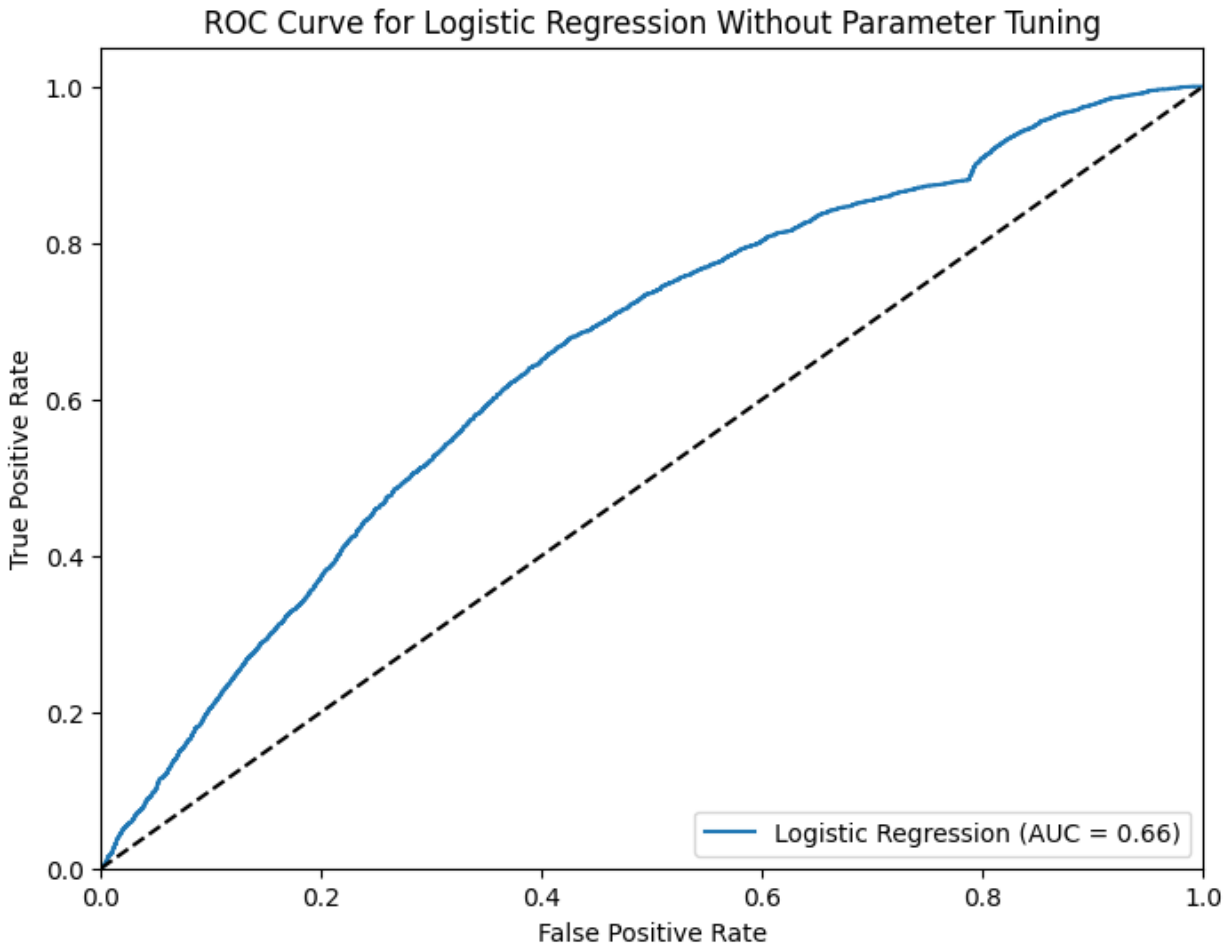
[ ]

	Feature	Coefficient
52	house_val	7.676577e-07
26	occupation_Professional	2.611135e-12
1	gender_M	2.174794e-12
17	marriage_Married	1.711900e-12
16	online_Y	1.569315e-12
29	mortgage_High	1.465946e-12
4	education_Bach	1.281186e-12
11	age_<=55	1.259121e-12
6	education_Grad	1.112532e-12
10	age_<=45	9.192310e-13
31	mortgage_Med	8.365872e-13
48	fam_income_J	4.176872e-13
45	fam_income_G	3.516750e-13
46	fam_income_H	3.349154e-13
50	fam_income_L	3.002583e-13
47	fam_income_I	2.686631e-13
49	fam_income_K	2.640666e-13
12	age_<=65	1.983253e-13
36	region_Rest	1.859488e-14
51	fam_income_U	-8.566057e-16
44	fam_income_F	-8.656615e-14
24	occupation_Farm	-1.039318e-13
22	child_Y	-1.917200e-13
38	region_West	-2.265250e-13
25	occupation_Others	-2.423195e-13
21	child_U	-3.465387e-13
32	house_owner_Owner	-3.508265e-13

Logistic Regression Coefficients After Tuning

	Feature	Coefficient
1	gender_M	5.714598e-01
11	age_<=55	3.684281e-01
26	occupation_Professional	3.615279e-01
10	age_<=45	2.985646e-01
6	education_Grad	2.572335e-01
12	age_<=65	2.258741e-01
29	mortgage_High	1.931587e-01
4	education_Bach	1.463585e-01
31	mortgage_Med	1.404577e-01
48	fam_income_J	1.105464e-01
28	occupation_Sales/Service	9.948486e-02
45	fam_income_G	8.010840e-02
18	marriage_Single	8.008933e-02
16	online_Y	4.405282e-02
21	child_U	4.318030e-02
46	fam_income_H	3.233802e-02
49	fam_income_K	2.792928e-02
25	occupation_Others	2.071705e-02
37	region_South	1.163149e-02
52	house_val	4.660530e-07
36	region_Rest	0.000000e+00
38	region_West	0.000000e+00
22	child_Y	0.000000e+00
51	fam_income_U	0.000000e+00

The ROC curve for the logistic regression without tuning had a slight bump at a false positive rate of 0.8 which was fixed after the hyperparameters were tuned.



As for the Random Forest, tuning increased the overall accuracy from 67% to 69% which was not as significant of an increase.

Random Forest Model Before Tuning

```
Random Forest Model without Parameter Tuning
Accuracy: 0.6668835393623943
              precision    recall  f1-score   support

           0       0.67      0.67      0.67      5396
           1       0.67      0.66      0.66      5363

    accuracy                           0.67     10759
   macro avg       0.67      0.67      0.67     10759
weighted avg       0.67      0.67      0.67     10759

[[3622 1774]
 [1810 3553]]
```

Random Forest Model After Tuning

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters for Random Forest: {'classifier__max_depth': 10, 'classifier__n_estimators': 200}
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters for Logistic Regression: {'classifier__C': 0.1, 'classifier__penalty': 'l1'}
Random Forest Accuracy: 0.694302444465099
              precision    recall  f1-score   support

           0       0.70      0.69      0.69      5396
           1       0.69      0.70      0.70      5363

    accuracy                           0.69     10759
   macro avg       0.69      0.69      0.69     10759
weighted avg       0.69      0.69      0.69     10759

[[3709 1687]
 [1602 3761]]
```

Similarly there were marginal increases in precision, recall and f1 scores. The importance of variables after tuning slightly decreased overall but house_val still remained as the variable with the largest importance. Other variables more or less stayed at their original rank of importance.

Random Forest Feature Importance Before Tuning

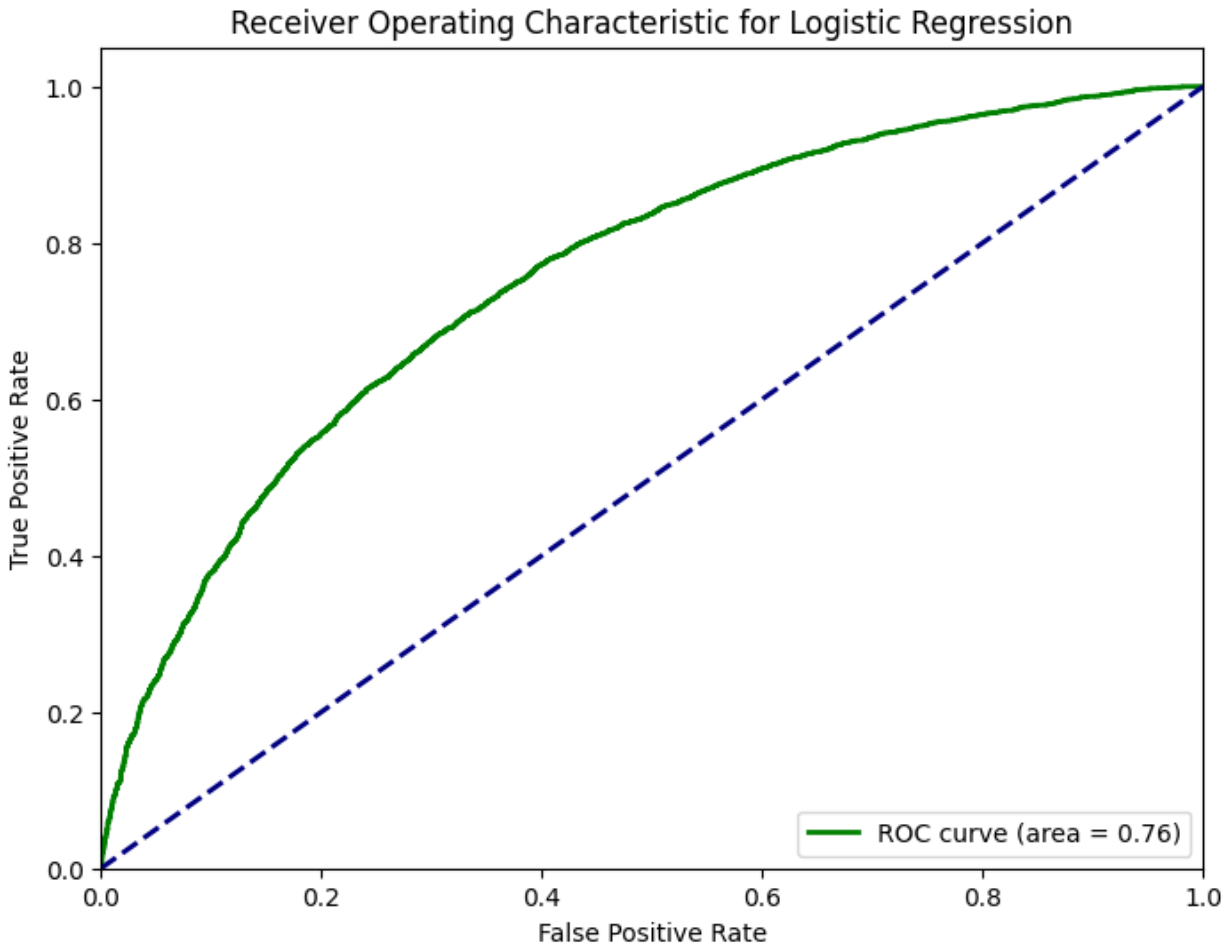
	Feature	Importance
52	remainder__house_val	0.215870
26	cat__occupation_Professional	0.029320
37	cat__region_South	0.026514
43	cat__fam_income_E	0.024709
1	cat__gender_M	0.023382
22	cat__child_Y	0.023085
34	cat__region_Midwest	0.022341
38	cat__region_West	0.022125
44	cat__fam_income_F	0.022081
35	cat__region_Northeast	0.022053
20	cat__child_N	0.021475
5	cat__education_College	0.020028
30	cat__mortgage_Low	0.019842
21	cat__child_U	0.019624
19	cat__marriage_nan	0.019547
4	cat__education_Bach	0.019453
7	cat__education_HS	0.018860
11	cat__age_<=55	0.018838
16	cat__online_Y	0.018069
15	cat__online_N	0.017824
45	cat__fam_income_G	0.017819
17	cat__marriage_Married	0.017800
0	cat__gender_F	0.017166
28	cat__occupation_Sales/Service	0.016514
42	cat__fam_income_D	0.016384
10	cat__age_<=45	0.016373

Random Forest Feature Importance After Tuning

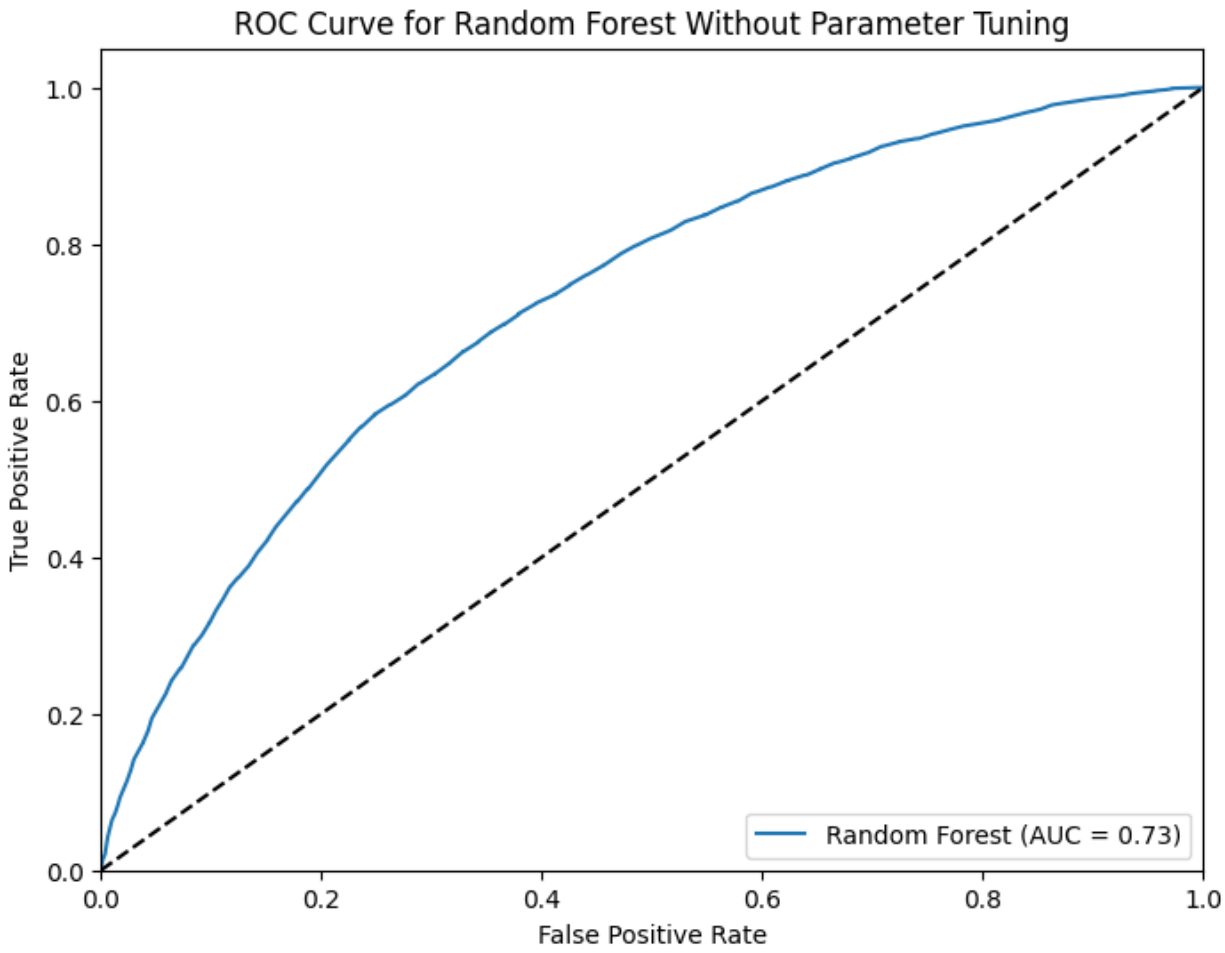
	Feature	Importance
52	house_val	0.147745
26	occupation_Professional	0.081667
1	gender_M	0.071083
30	mortgage_Low	0.061059
16	online_Y	0.060764
15	online_N	0.052417
0	gender_F	0.049898
19	marriage_nan	0.041261
17	marriage_Married	0.033836
29	mortgage_High	0.027899
7	education_HS	0.019830
11	age_<=55	0.019425
4	education_Bach	0.018400
6	education_Grad	0.017896
3	education_<HS	0.017658
14	age_Unknown	0.014978
13	age_>65	0.014783
27	occupation_Retired	0.013871
8	age_<=25	0.012084
10	age_<=45	0.011381
20	child_N	0.009685
22	child_Y	0.009220
43	fam_income_E	0.008994
31	mortgage_Med	0.008993
23	occupation_Blue Collar	0.008896
28	occupation_Sales/Service	0.008621
38	region_West	0.008306
12	age_<=65	0.008300

As for the ROC curves, for all except logistic regression before tuning, they had no anomaly and all were identical.

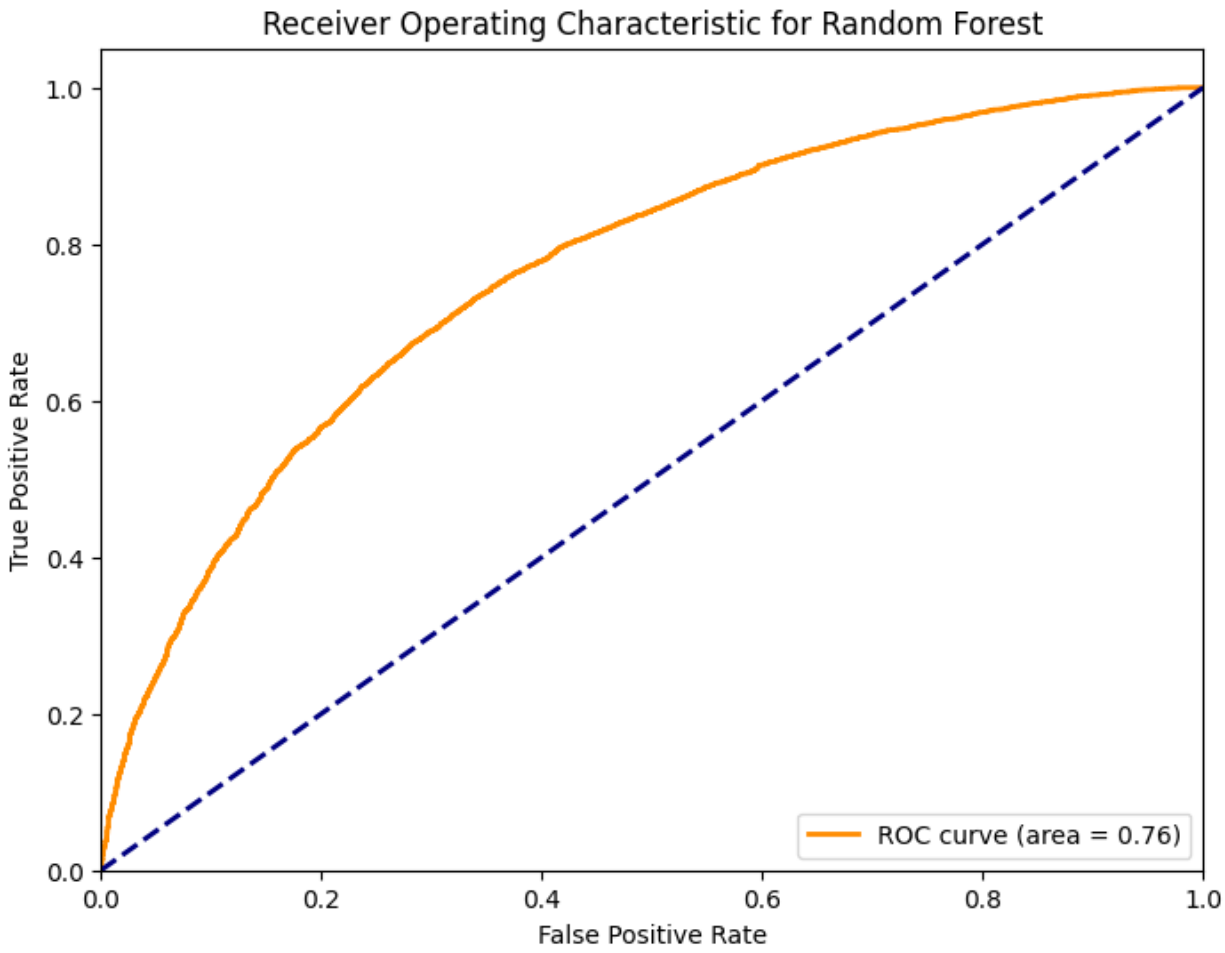
ROC Logistic Regression After Tuning



ROC Random Forest Before Tuning



ROC Random Forest After Tuning



Results and Discussion

Usually regularization methods cause the variables to reduce to zero. This was done for our regression and resulted in improved accuracy. However post tuning some coefficients significantly increased in size. This can be attributed to rebalancing of coefficients rather than reducing them to near zero. This could also be an attempt to correcting multicollinearity in the dataset. The model might increase certain coefficients to penalize others and maintain the overall predictive power. In the ROC curve for the bump at 0.8 false positive rate, the model might be classifying a subset of negative cases incorrectly as positive. The "bump" indicates this subset where the model is particularly sensitive or biased towards predicting positive or 1.

For the Random Forest models, the post tuning results suggest that over fitting and noise in the data is not a cause of concern because tuning due to the number of trees and depth of trees to reduce overfitting did not significantly increase the performance of the model. For our case, we want models that are better at predicting the positive class 1 which is a representation of the customer converting. Hence precision tells us how often a conversion is predicted correctly and sensitivity tells portion of actual positives correctly predicted. Hence either of the two models can be used but it is recommended to use the tuned logistic regression model solely because of less computationally demanded.

Conclusion and Recommendations

As the outcome of the analysis of all the predictive models, based on variable coefficients and importance, the recommendations provided to Imperials Ltd will be as follows:

- Try to target individuals with high house value, low mortgage, and high incomes.
- Individuals who are Professionals and are at least graduates have better conversion chances.
- Individuals who are already house owners are better to target than house renters.
- Target more men instead of women.

Appendix

```
import matplotlib.pyplot as plt
import numpy as np

from google.colab import files
import pandas as pd
import io


# Upload the file
uploaded = files.upload()
```

```
# Read the file into a DataFrame
if 'sales_data.csv' in uploaded:
    # Read the file into a pandas DataFrame
    data = pd.read_csv(io.BytesIO(uploaded['sales_data.csv']))
    # Show the first few rows of the DataFrame
    print(data.head())
else:
    # If the file isn't found, print an error message
    print("File not found. Please check the filename and upload again.")
```

```
# Structure and summary of the data
print(data.info())
print(data.describe())

# Check for missing values (NaN)
print(data.isna().sum().sum())

#Unique Values for each categorical variable
unique_flag = data['flag'].unique()
print(unique_flag)

unique_gender = data['gender'].unique()
print(unique_gender)

unique_education = data['education'].unique()
print(unique_education)

unique_age = data['age'].unique()
print(unique_age)
```

```
unique_online = data['online'].unique()
print(unique_online)

unique_marriage = data['marriage'].unique()
print(unique_marriage)

unique_child = data['child'].unique()
print(unique_child)

unique_occupation = data['occupation'].unique()
print(unique_occupation)

unique_mortgage = data['mortgage'].unique()
print(unique_mortgage)

unique_house_owner = data['house_owner'].unique()
print(unique_house_owner)

unique_region = data['region'].unique()
print(unique_region)

unique_fam_income = data['fam_income'].unique()
print(unique_fam_income)

# Boxplot for house_value to identify outliers
plt.boxplot(data['house_val'])
plt.show()

# Identifying and handling outliers in house_val
print((data['house_val'] > 3000000).sum())
print(((data['house_val'] > 3000000).sum() / len(data)) * 100)
```

```
# Removing outliers
data = data[data['house_val'] <= 3000000]

# Handling weird naming in 'education'
education_mapping = {
    "0. <HS": "<HS",
    "1. HS": "HS",
    "2. Some College": "College",
    "3. Bach": "Bach",
    "4. Grad": "Grad",
    "": np.nan # Convert empty strings to NaN
```

```
}
data['education'].replace(education_mapping, inplace=True)
data.dropna(subset=['education'], inplace=True) # Remove rows with NaN in
'education'

# Handling weird naming in 'age'
age_mapping = {
    "1_Unk": "Unknown",
    "2_<=25": "<=25",
    "3_<=35": "<=35",
    "4_<=45": "<=45",
    "5_<=55": "<=55",
    "6_<=65": "<=65",
    "7_>65": ">65"
}
data['age'].replace(age_mapping, inplace=True)

# Handling weird naming in 'mortgage'
mortgage_mapping = {
    "1Low": "Low",
    "2Med": "Med",
    "3High": "High"
}
data['mortgage'].replace(mortgage_mapping, inplace=True)

# Handling missing values in 'house_owner' and 'marriage'
data['house_owner'].replace("", np.nan, inplace=True)
data.dropna(subset=['house_owner'], inplace=True)

data['marriage'].replace("", "Unknown", inplace=True)

# Convert individual variables to categorical types
categorical_columns = ['flag', 'gender', 'education', 'age', 'online',
'marriage',
                       'child', 'occupation', 'mortgage', 'house_owner',
'region', 'fam_income']
for col in categorical_columns:
    data[col] = data[col].astype('category')
```

```
# Structure after conversions
print(data.info())
```

```
# Descriptive statistics for numeric variables
```

```
numeric_stats = data.describe()

# Frequency distribution for categorical variables
categorical_stats = data.describe(include='category')

print("Descriptive statistics for numeric variables:")
print(numeric_stats)

print("\nFrequency distribution for categorical variables:")
print(categorical_stats)
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Using one-hot encoding for all categorical variables including 'flag'
categorical_variables = ['flag', 'gender', 'education', 'age', 'online',
                         'marriage', 'child', 'occupation', 'mortgage',
                         'house_owner', 'region', 'fam_income']

encoded_data = pd.get_dummies(data, columns=categorical_variables,
drop_first=False)

# Correlation matrix
correlation_matrix = encoded_data.corr()

# 'house_val' variable
correlations_with_house_val =
correlation_matrix['house_val'].sort_values(ascending=False)

print("Correlations of all variables with 'house_val':")
print(correlations_with_house_val)

# Heatmap for correlations with 'house_val'
sorted_correlation_df =
encoded_data[correlations_with_house_val.index].corr()
plt.figure(figsize=(12, 10))
sns.heatmap(sorted_correlation_df[['house_val']], annot=True,
cmap='coolwarm', center=0)
plt.title('Correlation with House Value')
plt.show()
```

```
encoded_data = data.copy()

# Map binary categorical variables to numbers (for example, 'flag',
'online', etc.)
binary_mappings = {
    'Y': 1,
    'N': 0
}

encoded_data['flag_numeric'] = encoded_data['flag'].map(binary_mappings)

categorical_variables = ['gender', 'education', 'age', 'online',
                         'marriage', 'child', 'occupation', 'mortgage',
                         'house_owner', 'region', 'fam_income']

encoded_data = pd.get_dummies(encoded_data, columns=categorical_variables,
drop_first=True)

# Calculate the correlation matrix
correlation_matrix = encoded_data.corr()

correlation_pairs =
correlation_matrix.where(np.tril(np.ones(correlation_matrix.shape), k=-
1).astype(bool))

# Stack the data and reset index
correlation_stacked = correlation_pairs.stack().reset_index()

# Name the columns
correlation_stacked.columns = ['Variable 1', 'Variable 2', 'Correlation']

# Calculate whether each correlation is positive or negative
correlation_stacked['Positive'] = correlation_stacked['Correlation'] > 0

# Sort by the absolute value of 'Correlation', descending
correlation_stacked['AbsCorrelation'] =
correlation_stacked['Correlation'].abs()
top_correlations = correlation_stacked.sort_values('AbsCorrelation',
ascending=False).head(15)

# Drop the 'AbsCorrelation' column as it's no longer needed
top_correlations = top_correlations.drop('AbsCorrelation', axis=1)

# Display the top 15 correlations
print(top_correlations)
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set the aesthetic style of the plots
sns.set_style("whitegrid")

# Visualisation 1: Distribution of house values
plt.figure(figsize=(10, 6))
sns.histplot(data['house_val'], kde=True)
plt.title('Distribution of House Values')
plt.xlabel('House Value')
plt.ylabel('Frequency')
plt.show()

# Visualisation 2: Count plot for education levels
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='education')
plt.title('Count of Education Levels')
plt.xlabel('Education')
plt.ylabel('Count')
plt.show()

# Visualisation 3: Age distribution within each education level
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='education', y='house_val')
plt.title('House Value across Education Levels')
plt.xlabel('Education')
plt.ylabel('House Value')
plt.show()

# Visualisation 4: Relationship between online activity and house value
plt.figure(figsize=(10, 6))
sns.violinplot(data=data, x='online', y='house_val')
plt.title('House Value for Online and Offline Customers')
plt.xlabel('Online')
plt.ylabel('House Value')
plt.show()

# Visualisation 5: House value distribution by region
plt.figure(figsize=(10, 6))
sns.barplot(data=data, x='region', y='house_val')
plt.title('Average House Value by Region')
plt.xlabel('Region')
```

```
plt.ylabel('Average House Value')
plt.show()

# Visualisation 6: Family income vs. mortgage status
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='fam_income', hue='mortgage')
plt.title('Family Income and Mortgage Status')
plt.xlabel('Family Income')
plt.ylabel('Count')
plt.legend(title='Mortgage')
plt.show()
```

```
print(data.info)
print(encoded_data.info)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Map target variable 'flag' to numeric values
data['flag_numeric'] = data['flag'].map({'Y': 1, 'N': 0})

# Define features and target variable
X = data.drop(['flag', 'flag_numeric'], axis=1)  # Drop the original flag
and the encoded flag_numeric
y = data['flag_numeric']

# Define categorical features for encoding
categorical_features = X.select_dtypes(include=['object',
'category']).columns.tolist()

# One-hot encode categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_features)
```

```
    ],
    remainder='passthrough'
)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Random Forest Pipeline
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Logistic Regression Pipeline
lr_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(solver='liblinear',
random_state=42))
])

# Parameter grid for Random Forest
param_grid_rf = {
    'classifier__n_estimators': [100, 200, 300],  # Number of trees in the
forest
    'classifier__max_depth': [None, 10, 20, 30],  # Maximum depth of the
trees
}

# Parameter grid for Logistic Regression
param_grid_lr = {
    'classifier__C': [0.001, 0.01, 0.1, 1, 10],
    'classifier__penalty': ['l1', 'l2']
}

# GridSearch for Random Forest
grid_search_rf = GridSearchCV(rf_pipeline, param_grid_rf, cv=5, n_jobs=-1,
verbose=2)
grid_search_rf.fit(X_train, y_train)
print(f'Best parameters for Random Forest: {grid_search_rf.best_params_}')

# GridSearch for Logistic Regression
grid_search_lr = GridSearchCV(lr_pipeline, param_grid_lr, cv=5, n_jobs=-1,
verbose=2)
grid_search_lr.fit(X_train, y_train)
```

```
print(f'Best parameters for Logistic Regression:
{grid_search_lr.best_params_}')

# Evaluate Random Forest
y_pred_rf = grid_search_rf.predict(X_test)
print(f'Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf)}')
print(classification_report(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))

# Evaluate Logistic Regression
y_pred_lr = grid_search_lr.predict(X_test)
print(f'Logistic Regression Accuracy: {accuracy_score(y_test,
y_pred_lr)}')
print(classification_report(y_test, y_pred_lr))
print(confusion_matrix(y_test, y_pred_lr))
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities for the test set
y_probs_rf = grid_search_rf.predict_proba(X_test)[:, 1]  # probabilities
for the positive class

# Compute ROC curve and area under the curve for Random Forest
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_probs_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve for Random Forest
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label=f'ROC curve (area
= {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Random Forest')
plt.legend(loc="lower right")
plt.show()
```

```
# Predict probabilities for the test set
```

```
y_probs_lr = grid_search_lr.predict_proba(X_test)[:, 1]  # probabilities
for the positive class

# Compute ROC curve and area under the curve for Logistic Regression
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_probs_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Plot ROC curve for Logistic Regression
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, color='green', lw=2, label=f'ROC curve (area =
{roc_auc_lr:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```

```
#Models without parameter tuning

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Random Forest Pipeline without parameter tuning
rf_model_simple = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))  # Using
default parameters
])
# Train the Random Forest model
rf_model_simple.fit(X_train, y_train)
# Predictions with Random Forest
y_pred_rf_simple = rf_model_simple.predict(X_test)

# Evaluate Random Forest
print('Random Forest Model without Parameter Tuning')
print(f'Accuracy: {accuracy_score(y_test, y_pred_rf_simple)}')
print(classification_report(y_test, y_pred_rf_simple))
print(confusion_matrix(y_test, y_pred_rf_simple))
print('\n')
```

```
# Logistic Regression Pipeline without parameter tuning
lr_model_simple = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(solver='liblinear',
random_state=42))  # Using default parameters
])
# Train the Logistic Regression model
lr_model_simple.fit(X_train, y_train)
# Predictions with Logistic Regression
y_pred_lr_simple = lr_model_simple.predict(X_test)
# Evaluate Logistic Regression
print('Logistic Regression Model without Parameter Tuning')
print(f'Accuracy: {accuracy_score(y_test, y_pred_lr_simple)}')
print(classification_report(y_test, y_pred_lr_simple))
print(confusion_matrix(y_test, y_pred_lr_simple))
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities for the positive class with the Random Forest
model
y_probs_rf_simple = rf_model_simple.predict_proba(X_test)[:, 1]

# Compute ROC curve and area under the curve for Random Forest
fpr_rf_simple, tpr_rf_simple, _ = roc_curve(y_test, y_probs_rf_simple)
roc_auc_rf_simple = auc(fpr_rf_simple, tpr_rf_simple)

# Plot ROC curve for Random Forest without parameter tuning
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf_simple, tpr_rf_simple, label=f'Random Forest (AUC =
{roc_auc_rf_simple:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for reference
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Random Forest Without Parameter Tuning')
plt.legend(loc="lower right")
plt.show()
```

```
# Predict probabilities for the positive class with the Logistic
Regression model
y_probs_lr_simple = lr_model_simple.predict_proba(X_test)[:, 1]

# Compute ROC curve and area under the curve for Logistic Regression
fpr_lr_simple, tpr_lr_simple, _ = roc_curve(y_test, y_probs_lr_simple)
roc_auc_lr_simple = auc(fpr_lr_simple, tpr_lr_simple)

# Plot ROC curve for Logistic Regression without parameter tuning
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr_simple, tpr_lr_simple, label=f'Logistic Regression (AUC =
{roc_auc_lr_simple:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for reference
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression Without Parameter Tuning')
plt.legend(loc="lower right")
plt.show()
```

```
#Feature importance for Random Forest

rf_model_fitted = grid_search_rf.best_estimator_['classifier']

import numpy as np
import pandas as pd

rf_model_fitted = grid_search_rf.best_estimator_['classifier']

# Get feature names for the one-hot encoded categorical features
onehot_feature_names =
grid_search_rf.best_estimator_.named_steps['preprocessor'].named_transform
ers_['cat'].get_feature_names_out(input_features=categorical_features)

# Combine one-hot encoded feature names with non-categorical feature names
non_categorical_features = X.select_dtypes(exclude=['object',
'category']).columns.tolist()
all_feature_names = np.concatenate([onehot_feature_names,
non_categorical_features])

# Feature importances
importances = rf_model_fitted.feature_importances_
```

```
# Create DataFrame for easier interpretation
importances_df = pd.DataFrame({
    'Feature': all_feature_names,
    'Importance': importances
})

print(importances_df.sort_values(by='Importance', ascending=False))
```

```
#Logistic Regression Coeffecients

lr_model_fitted = grid_search_lr.best_estimator_['classifier']

# Get feature names after one-hot encoding from the ColumnTransformer
feature_names =
grid_search_lr.best_estimator_.named_steps['preprocessor'].named_transform
ers_['cat'].get_feature_names_out(categorical_features)
non_categorical_features = X.select_dtypes(exclude=['object',
'category']).columns.tolist()
all_feature_names = np.concatenate([feature_names,
non_categorical_features])

# Coefficients
coefficients = lr_model_fitted.coef_[0]

# Create DataFrame for easier interpretation
coefficients_df = pd.DataFrame({
    'Feature': all_feature_names,
    'Coefficient': coefficients
})

print(coefficients_df.sort_values(by='Coefficient', ascending=False))
```

```
#Importances of Random Forest without tuning
# Import necessary libraries
import numpy as np
import pandas as pd

importances =
rf_model_simple.named_steps['classifier'].feature_importances_
```

```
feature_names_transformed =
rf_model_simple.named_steps['preprocessor'].get_feature_names_out()

all_feature_names = feature_names_transformed

# Create a DataFrame for feature importances for better readability
importances_df = pd.DataFrame({
    'Feature': all_feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Display the DataFrame
print(importances_df)
```

```
#Coeffecients of Logistic Regression without Tuning

# Access the preprocessor from the pipeline
preprocessor = lr_model_simple.named_steps['preprocessor']

feature_names_transformed =
preprocessor.named_transformers_['cat'].get_feature_names_out()

non_categorical_features = X.select_dtypes(exclude=['object',
'category']).columns
all_feature_names = np.concatenate((feature_names_transformed,
non_categorical_features), axis=None)

# Retrieve the coefficients from the fitted logistic regression model
within the pipeline
coefficients = fitted_lr_model.coef_[0]

# Create a DataFrame for easier interpretation
coefficients_df = pd.DataFrame({
    'Feature': all_feature_names,
    'Coefficient': coefficients
}).sort_values(by='Coefficient', ascending=False)

print(coefficients_df)
```

References

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

Cutler, A., Cutler, D.R., and Stevens, J.R. (2012). Random forests. Ensemble Machine Learning, 157-175.

Genuer, R., Poggi, J.-M., and Tuleau-Malot, C. (2010). Variable selection using random forests. Pattern Recognition Letters, 31(14), 2225-2236.

Hosmer, D.W., Lemeshow, S., and Sturdivant, R.X. (2013). Applied Logistic Regression. Wiley.

Liaw, A., and Wiener, M. (2002). Classification and regression by randomForest. R News, 2(3), 18-22.

Menard, S. (2000). Applied Logistic Regression Analysis. Sage.

Probst, P., Wright, M.N., and Boulesteix, A.-L. (2019). Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9(3), e1301.

Ansell, J., et al. (2007). Handbook of Insurance. Springer.

Barocas, S., and Selbst, A.D. (2016). Big data's disparate impact. California Law Review, 104, 671.

Bishop, C.M. (2006). Pattern Recognition and Machine Learning. Springer.

Brown, I., and Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. Expert Systems with Applications, 39(3), 3446-3453.

Delen, D., Walker, G., and Kadam, A. (2005). Predicting breast cancer survivability: a comparison of three data mining methods. Artificial Intelligence in Medicine, 34(2), 113-127.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). The Elements of Statistical Learning. Springer.

Kuhn, M., and Johnson, K. (2013). Applied Predictive Modeling. Springer.

Ngai, E.W.T., et al. (2009). Application of data mining techniques in customer relationship management: A literature review and classification. Expert Systems with Applications, 36(2), 2592-2602.

Pagallo, U. (2011). The laws of robots: Crimes, contracts, and torts. Springer.

Ratcliffe, J.H., and Mccarty, W.P. (2003). Exploring the relationship between the spatial distribution of methamphetamine labs and residential property values: A predictive analysis approach. Journal of Research in Crime and Delinquency, 40(4), 335-355.

Smith, H.J. (2012). Information privacy and marketing: What the U.S. should (and shouldn't) learn from Europe. California Management Review, 54(3), 8-33.

Tsai, C.-F., and Wu, J.-W. (2008). Using neural network ensembles for bankruptcy prediction and credit scoring. Expert Systems with Applications, 34(4), 2639-2649.

Zarsky, T.Z. (2016). The trouble with algorithmic decisions: An analytic road map to examine efficiency and fairness in automated and opaque decision making. Science, Technology, & Human Values, 41(1), 118-132.