

# Project 7: Modeling Car Insurance Claims Outcome

Lawal's Project

2024-11-23

## Table of contents

1	Project Overview	2
2	Task	3
3	Data Source	3
4	Tools	4
5	Methodology: Steps/Explanations	4
6	Data Analysis	7
7	Result/Findings	14
8	Recommendations	14
9	Limitations	14
10	Conclusion	14



Figure 1: car

## 1 Project Overview

Insurance companies invest a lot of **time and money** into optimizing their pricing and accurately estimating the likelihood that customers will make a claim. In many countries insurance it is a legal requirement to have car insurance in order to drive a vehicle on public roads, so the market is very large!

Knowing all of this, On the Road car insurance have requested your services in building a model to predict whether a customer will make a claim on their insurance during the policy period. As they have very little expertise and infrastructure for deploying and monitoring machine learning models, they've asked you to identify the single feature that results in the best performing model, as measured by accuracy, so they can start with a simple model in production.

They have supplied you with their customer data as a csv file called `car_insurance.csv`, along with a table detailing the column names and descriptions below.

Table 1: Customer data

Column	Description
<code>id</code>	Unique client identifier
<code>age</code>	Client's age:
<code>gender</code>	Client's gender:
<code>driving_experience</code>	Years the client has been driving:
<code>education</code>	Client's level of education:
<code>income</code>	Client's income level:
<code>credit_score</code>	Client's credit score (between zero and one)
<code>vehicle_ownership</code>	Client's vehicle ownership status:
<code>vehcile_year</code>	Year of vehicle registration:
<code>married</code>	Client's marital status:
<code>children</code>	Client's number of children
<code>postal_code</code>	Client's postal code
<code>annual_mileage</code>	Number of miles driven by the client each year
<code>vehicle_type</code>	Type of car:
<code>speeding_violations</code>	Total number of speeding violations received by the client
<code>duis</code>	Number of times the client has been caught driving under the influence of alcohol
<code>past_accidents</code>	Total number of previous accidents the client has been involved in
<code>outcome</code>	Whether the client made a claim on their car insurance (response variable):

## 2 Task

- Identify the single feature of the data that is the best predictor of whether a customer will put in a claim (the `"outcome"` column), excluding the `"id"` column.
- Store as a DataFrame called `best_feature_df`, containing columns named `"best_feature"` and `"best_accuracy"` with the name of the feature with the highest accuracy, and the respective accuracy score.

## 3 Data Source

Data: The primary data used for this analysis is the `car_insurance.csv`, which can be downloaded [here](#). See Table 1 for the column names and descriptions.

## 4 Tools

Jupyter lab

## 5 Methodology: Steps/Explanations

The necessary libraries was imported, which were Pandas and logit from `statsmodels.formula.api`

### Reading in and exploring the dataset, including imputation of missing values

- The Original dataset was loaded, named `car`.
- The first function, `explore`, was designed to help analyze and clean a dataset by providing a detailed overview of its structure and content, and it also optionally imputes missing values. Here's a step-by-step explanation:

1. **Function creation and its arguments:** `data`, the DataFrame to analyze; `head_rows`, the number of rows to display from the start of the DataFrame (default: 5); `group_by_col`, the column used to group data for imputing missing values (default: None); `cols_to_impute`, the list of columns where missing values will be filled with the group mean (default: None).

```
def explore(data, head_rows=5, group_by_col=None, cols_to_impute=None):
```

2. **Function Task 1:** Prints information about the DataFrame, such as:

- Number of rows and columns.
- Data types of each column.
- Non-null counts for each column.

```
print("\n--- DataFrame Info ---\n")
data.info()
```

3. **Function Task 2:** Displays summary statistics for all columns, including:

- For numerical data: Mean, standard deviation, min, max, and percentiles.
- For categorical data: Frequency counts (mode) and unique counts.

```
print("\n--- Summary Statistics ---\n")
print(data.describe(include='all'))
```

3. **Function Task 3:** Displays the first `head_rows` rows (default: 5) of the DataFrame to give a preview of the data.

```
print(f"\n--- First {head_rows} Rows ---\n")
print(data.head(head_rows))
```

4. **Function Task 4:** Iterates over each column and prints the unique values present in it. Helps understand the distinct data points for each column.

```
print("\n--- Unique Values ---\n")
for col in data.columns:
    print(f"{col}: {data[col].unique()}")
```

5. **Function Task 5:** Fills missing values (NaN) in the specified columns (`cols_to_impute`) by grouping data based on `group_by_col` and calculating the mean for each group.

- Steps:
  - Groups the data by the column specified in `group_by_col`.
  - Calculates the mean for the columns listed in `cols_to_impute` for each group.
  - Fills missing values in each column by mapping the group means to the corresponding rows.
- Error Handling:
  - Ensures the function doesn't crash if the specified column is not found or if an error occurs during imputation.

```
if group_by_col and cols_to_impute:
    print("\n--- Imputing Missing Values ---\n")
    try:
        group_means = data.groupby(group_by_col)[cols_to_impute].mean().to_dict()
        for col in cols_to_impute:
            if col in data.columns:
                print(f"Imputing missing values in '{col}' based on group means of '{group_by_col}'")
                data[col] = data[col].fillna(data[group_by_col].map(group_means[col]))
            else:
                print(f"Column '{col}' not found in the dataset.")
    except Exception as e:
        print(f"Error while imputing missing values: {e}")
```

6. **Function Task 6:** After the imputation, checks and prints the count of missing values in each column to verify if gaps were successfully filled.

```
print("\n--- Any missing values again ? ---\n")
print(data.isna().sum())
```

## Finding the best performing model, with the highest accuracy.

- The second function, `best_logmodel`, was designed to identify the single best feature in a dataset for predicting a binary outcome using logistic regression with the `statsmodels` library. Here's a detailed explanation:

1. **Function creation and its arguments:** `data`, the input dataset for modeling as a pandas DataFrame; `outcome_column`, the target column (dependent variable) representing the outcome being predicted (default: 'outcome'); `id_column`, a unique identifier column to exclude from the analysis (default: 'id').

```
def best_logmodel(data, outcome_column='outcome', id_column='id'):
```

2. **Function Task:** Creates a new DataFrame (`data1`) by removing the `id_column` (not predictive) and the `outcome_column` (target variable) from the list of features. The remaining columns are treated as potential predictors.

```
data1 = data.drop(columns=[id_column, outcome_column])
```

3. **Initialize Tracking Variables:** `best_feature`, placeholder for the name of the feature with the highest accuracy and `best_accuracy`, tracks the best accuracy score encountered during the iteration.

```
best_feature = None
best_accuracy = 0
```

4. **Loop Through Each Feature:** Iterates through all the columns (features) in `data1` to evaluate their predictive power for the `outcome_column`.

```
for col in data1.columns:
```

5. **Create the Logistic Regression Formula:** Constructs a formula for logistic regression in the form "`outcome_column ~ feature_column`".

```
formula = f"{outcome_column} ~ {col}"
```

6. **Fit Logistic Regression Model:** Fits a logistic regression model for the current feature using the `logit` function from `statsmodels`. The `'disp=False'` argument suppresses output during model fitting.

```
model = logit(formula=formula, data=data).fit(dispatch=False)
```

7. **Generate Confusion Matrix:** Produces a confusion matrix for the logistic regression model's predictions.

```
confusion_matrix = model.pred_table()
```

- **Confusion Matrix Layout:**

```
[[TN, FP], # TN = True Negatives, FP = False Positives
 [FN, TP]] # FN = False Negatives, TP = True Positives
```

8. **Calculates the model's accuracy from the confusion matrix:**

- TP: True Positives (correctly predicted positives).
- TN: True Negatives (correctly predicted negatives).
- T: Total number of predictions.
- **Accuracy Formula:**

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total Predictions}}$$

9. **Update the Best Feature:** Compares the current feature's accuracy with the best accuracy seen so far. If the current feature has a higher accuracy, update `best_feature` and `best_accuracy`.

```
if accuracy > best_accuracy:
    best_feature = col
    best_accuracy = accuracy
```

10. **Store Results in a DataFrame:** Summarizes the results into a pandas DataFrame with:

- `best_feature`: The name of the feature with the highest accuracy.
- `best_accuracy`: The corresponding accuracy score.

```
best_feature_df = pd.DataFrame({
    "best_feature": [best_feature],
    "best_accuracy": [best_accuracy]
})
```

11. **Return the Results:** Returns the DataFrame so that the results can be used or displayed.

```
return best_feature_df
```

## 6 Data Analysis

```
# Import required modules
import pandas as pd
from statsmodels.formula.api import logit

# Start coding!
car = pd.read_csv("car_insurance.csv")

# Exploring the data
```

```

def explore(data, head_rows=5, group_by_col=None, cols_to_impute=None):
    """
    Explores the given DataFrame by displaying basic information, summary statistics,
    the first few rows, unique values, and imputes missing values with group means if specified.

    Parameters:
        data (pd.DataFrame): The DataFrame to explore.
        head_rows (int): Number of rows to display for the head of the DataFrame. Default is 5.
        group_by_col (str): Column name to group by for imputing missing values. Default is None.
        cols_to_impute (list): List of column names to impute missing values. Default is None.
    """
    print("\n--- DataFrame Info ---\n")
    data.info()

    print("\n--- Summary Statistics ---\n")
    print(data.describe(include='all')) # Include all data types in describe()

    print(f"\n--- First {head_rows} Rows ---\n")
    print(data.head(head_rows))

    print("\n--- Unique Values ---\n")
    for col in data.columns:
        print(f"{col}: {data[col].unique()}")

    # Impute missing values if group_by_col and cols_to_impute are specified
    if group_by_col and cols_to_impute:
        print("\n--- Imputing Missing Values ---\n")
        try:
            group_means = data.groupby(group_by_col)[cols_to_impute].mean().to_dict() # Group means
            for col in cols_to_impute:
                if col in data.columns:
                    print(f"Imputing missing values in '{col}' based on group means of '{group_by_col}'")
                    data[col] = data[col].fillna(data[group_by_col].map(group_means[col]))
                else:
                    print(f"Column '{col}' not found in the dataset.")
        except Exception as e:
            print(f"Error while imputing missing values: {e}")

    print("\n--- Any missing values again ? ---\n")
    print(data.isna().sum())

# Example usage
# explore(your_data, group_by_col="outcome", cols_to_impute=["credit_score", "annual_mileage"])

```



```
explore(car, group_by_col="outcome", cols_to_impute=["credit_score", "annual_mileage"])
```

```
def best_logmodel(data, outcome_column='outcome', id_column='id'):
```

```
    """
```

```
    Identifies the single best feature for predicting the outcome column using logistic regression
    with statsmodels. Calculates accuracy directly from the confusion matrix.
```

```
    Parameters:
```

```
        data (pd.DataFrame): The dataset containing features and the outcome column.
```

```
        outcome_column (str): The name of the target column.
```

```
        id_column (str): The name of the column to exclude from analysis.
```

```
    Returns:
```

```
        pd.DataFrame: A DataFrame with the best feature and its accuracy score.
```

```
    """
```

```
    # Exclude ID and outcome columns from columns set
```

```
    data1 = data.drop(columns=[id_column, outcome_column])
```

```
    best_feature = None
```

```
    best_accuracy = 0
```

```
    # Iterate through each columns
```

```
    for col in data1.columns:
```

```
        # Create formula for logistic regression
```

```
        formula = f"{outcome_column} ~ {col}"
```

```
        # Fit logistic regression model on the entire dataset
```

```
        model = logit(formula=formula, data=data).fit(dispatch=False)
```

```
        # Generate confusion matrix using pred_table()
```

```
        confusion_matrix = model.pred_table()
```

```
        # Calculate accuracy from confusion matrix
```

```
        TP = confusion_matrix[1, 1]
```

```
        TN = confusion_matrix[0, 0]
```

```
        T = confusion_matrix.sum()
```

```
        accuracy = (TP + TN) / T
```

```
        # Update the best feature if this one is better
```

```
        if accuracy > best_accuracy:
```

```
            best_feature = col
```

```
            best_accuracy = accuracy
```

```
    # Store results in a DataFrame
```

```

best_feature_df = pd.DataFrame({
    "best_feature": [best_feature],
    "best_accuracy": [best_accuracy]
})

return best_feature_df

# Example usage
# best_feature_df = best_logmodel(your_data)
# print(best_feature_df)

best_feature_df = best_logmodel(car)

print(best_feature_df)

--- DataFrame Info ---

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                    10000 non-null  int64
1   age                   10000 non-null  int64
2   gender                10000 non-null  int64
3   driving_experience     10000 non-null  object
4   education             10000 non-null  object
5   income                10000 non-null  object
6   credit_score           9018 non-null   float64
7   vehicle_ownership     10000 non-null  float64
8   vehicle_year          10000 non-null  object
9   married               10000 non-null  float64
10  children              10000 non-null  float64
11  postal_code           10000 non-null  int64
12  annual_mileage        9043 non-null   float64
13  vehicle_type          10000 non-null  object
14  speeding_violations    10000 non-null  int64
15  dui                   10000 non-null  int64
16  past_accidents        10000 non-null  int64
17  outcome               10000 non-null  float64
dtypes: float64(6), int64(7), object(5)
memory usage: 1.4+ MB

--- Summary Statistics ---

```

	id	age	gender	driving_experience	\
count	10000.000000	10000.000000	10000.000000	10000	
unique	NaN	NaN	NaN	4	
top	NaN	NaN	NaN	0-9y	
freq	NaN	NaN	NaN	3530	
mean	500521.906800	1.489500	0.499000	NaN	
std	290030.768758	1.025278	0.500024	NaN	
min	101.000000	0.000000	0.000000	NaN	
25%	249638.500000	1.000000	0.000000	NaN	
50%	501777.000000	1.000000	0.000000	NaN	
75%	753974.500000	2.000000	1.000000	NaN	
max	999976.000000	3.000000	1.000000	NaN	

	education	income	credit_score	vehicle_ownership	\
count	10000	10000	9018.000000	10000.000000	
unique	3	4	NaN	NaN	
top	high school	upper class	NaN	NaN	
freq	4157	4336	NaN	NaN	
mean	NaN	NaN	0.515813	0.697000	
std	NaN	NaN	0.137688	0.459578	
min	NaN	NaN	0.053358	0.000000	
25%	NaN	NaN	0.417191	0.000000	
50%	NaN	NaN	0.525033	1.000000	
75%	NaN	NaN	0.618312	1.000000	
max	NaN	NaN	0.960819	1.000000	

	vehicle_year	married	children	postal_code	annual_mileage	\
count	10000	10000.000000	10000.000000	10000.000000	9043.000000	
unique	2	NaN	NaN	NaN	NaN	
top	before 2015	NaN	NaN	NaN	NaN	
freq	6967	NaN	NaN	NaN	NaN	
mean	NaN	0.498200	0.688800	19864.548400	11697.003207	
std	NaN	0.500022	0.463008	18915.613855	2818.434528	
min	NaN	0.000000	0.000000	10238.000000	2000.000000	
25%	NaN	0.000000	0.000000	10238.000000	10000.000000	
50%	NaN	0.000000	1.000000	10238.000000	12000.000000	
75%	NaN	1.000000	1.000000	32765.000000	14000.000000	
max	NaN	1.000000	1.000000	92101.000000	22000.000000	

	vehicle_type	speeding_violations	duis	past_accidents	\
count	10000	10000.000000	10000.000000	10000.000000	
unique	2	NaN	NaN	NaN	
top	sedan	NaN	NaN	NaN	
freq	9523	NaN	NaN	NaN	

mean	NaN	1.482900	0.23920	1.056300
std	NaN	2.241966	0.55499	1.652454
min	NaN	0.000000	0.00000	0.000000
25%	NaN	0.000000	0.00000	0.000000
50%	NaN	0.000000	0.00000	0.000000
75%	NaN	2.000000	0.00000	2.000000
max	NaN	22.000000	6.00000	15.000000

	outcome
count	10000.000000
unique	NaN
top	NaN
freq	NaN
mean	0.313300
std	0.463858
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

--- First 5 Rows ---

	id	age	gender	driving_experience	education	income	\
0	569520	3	0	0-9y	high school	upper class	
1	750365	0	1	0-9y	none	poverty	
2	199901	0	0	0-9y	high school	working class	
3	478866	0	1	0-9y	university	working class	
4	731664	1	1	10-19y	none	working class	

	credit_score	vehicle_ownership	vehicle_year	married	children	\
0	0.629027	1.0	after 2015	0.0	1.0	
1	0.357757	0.0	before 2015	0.0	0.0	
2	0.493146	1.0	before 2015	0.0	0.0	
3	0.206013	1.0	before 2015	0.0	1.0	
4	0.388366	1.0	before 2015	0.0	0.0	

	postal_code	annual_mileage	vehicle_type	speeding_violations	duis	\
0	10238	12000.0	sedan		0	0
1	10238	16000.0	sedan		0	0
2	10238	11000.0	sedan		0	0
3	32765	11000.0	sedan		0	0
4	32765	12000.0	sedan		2	0

past_accidents	outcome
----------------	---------

0	0	0.0
1	0	1.0
2	0	0.0
3	0	0.0
4	1	1.0

--- Unique Values ---

```
id: [569520 750365 199901 ... 468409 903459 442696]
age: [3 0 1 2]
gender: [0 1]
driving_experience: ['0-9y' '10-19y' '20-29y' '30y+']
education: ['high school' 'none' 'university']
income: ['upper class' 'poverty' 'working class' 'middle class']
credit_score: [0.62902731 0.35775712 0.49314579 ... 0.47094023 0.36418478 0.43522478]
vehicle_ownership: [1. 0.]
vehicle_year: ['after 2015' 'before 2015']
married: [0. 1.]
children: [1. 0.]
postal_code: [10238 32765 92101 21217]
annual_mileage: [12000. 16000. 11000. 13000. 14000. 10000. 8000. nan 18000. 17000.
7000. 15000. 9000. 5000. 6000. 19000. 4000. 3000. 2000. 20000.
21000. 22000.]
vehicle_type: ['sedan' 'sports car']
speeding_violations: [ 0  2  3  7  6  4 10 13  1  5  9  8 12 11 15 17 19 18 16 14 22]
duis: [0 2 1 3 4 5 6]
past_accidents: [ 0  1  3  7  2  5  4  6  8 10 11  9 12 14 15]
outcome: [0. 1.]
```

--- Imputing Missing Values ---

```
Imputing missing values in 'credit_score' based on group means of 'outcome'
Imputing missing values in 'annual_mileage' based on group means of 'outcome'
```

--- Any missing values again ? ---

id	0
age	0
gender	0
driving_experience	0
education	0
income	0
credit_score	0
vehicle_ownership	0
vehicle_year	0

```

married          0
children         0
postal_code      0
annual_mileage   0
vehicle_type     0
speeding_violations 0
duis             0
past_accidents   0
outcome          0
dtype: int64
      best_feature  best_accuracy
0  driving_experience      0.7771

```

## 7 Result/Findings

- The analysis identified **driving\_experience** (indicating the years the client has been driving) as the best predictor of whether a customer will file a claim, with an accuracy score of 77.7%. This indicates that the model correctly predicted claims and non-claims in approximately 78 out of 100 cases, making this feature a significant factor in claim prediction.

## 8 Recommendations

None

## 9 Limitations

None

## 10 Conclusion

My analysis identified **driving\_experience** (years of driving) as the strongest predictor of claim submissions, achieving an accuracy score of 77.7%. This result highlights the importance of driving experience in assessing customer risk. The model correctly classified claims and non-claims in 78 out of 100 cases.

Logistic regression was used to evaluate the predictive power of individual features, and accuracy was calculated using a confusion matrix. The prominence of driving experience suggests that more experienced drivers may exhibit different risk profiles, which could guide targeted policy offerings.

I recommend incorporating this insight into your risk assessment models.

## 11 References

1. For loop in Intermediate Python Course for Associate Data Scientist in Python Career Track in DataCamp Inc by Hugo Bowne-Henderson.
2. Introduction to functions in Python in Intermediate Python Course for Associate Data Scientist in Python Career Track in DataCamp Inc by Hugo Bowne-Henderson.
3. Introduction to Regression with statsmodels in Python in Intermediate Python Course for Associate Data Scientist in Python Career Track in DataCamp Inc by Maarten Van den Broeck.
4. Exploratory Data Analysis in Python in Intermediate Python Course for Associate Data Scientist in Python Career Track in DataCamp Inc by Hugo Bowne-Henderson.
5. Python For Data Analysis 3E (Online) by Wes McKinney Click [here](#) to preview.