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!

(
`Source: https://www.accenture.com/_acnmedia/pdf-84/accenture-machine-leaning-insuran`
)

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.

The dataset

Column	Description
`id`	Unique client identifier
`age`	Client's age: • `0` : 16-25 • `1` : 26-39 • `2` : 40-64 • `3` : 65+
`gender`	Client's gender: • `0` : Female • `1` : Male
`driving_experience`	Years the client has been driving: • `0` : 0-9 • `1` : 10-19 • `2` : 20-29 • `3` : 30+
`education`	Client's level of education: • `0` : No education • `1` : High school • `2` : University
`income`	Client's income level: • `0` : Poverty • `1` : Working class • `2` : Middle class • `3` : Upper class
`credit_score`	Client's credit score (between zero and one)
`vehicle_ownership`	Client's vehicle ownership status: • `0` : Does not own their vehilce (paying off finance) • `1` : Owns their vehicle
`vehcile_year`	Year of vehicle registration: • `0` : Before 2015 • `1` : 2015 or later
`married`	Client's marital status:

Column	Description
	• `0` : Not married • `1` : Married
`children`	Client's number of children
`postal_code`	Client's postal code
`annual_mileage`	Number of miles driven by the client each year
`vehicle_type`	Type of car: • `0` : Sedan • `1` : Sports car
`speeding_violations`	Total number of speeding violations received by the client
`duis`	Number of times the client has been caught driving under the influence of alcohol
`past_accidents`	Total number of previous accidents the client has been involved in
`outcome`	Whether the client made a claim on their car insurance (response variable): • `0` : No claim • `1` : Made a claim

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

#Final goal: Identify the single feature that produces the best accuracy in
predicting whether a customer makes a claim.

#1. Reading in and exploring the dataset

df = pd.read_csv("car_insurance.csv")
# Checking the actual data:
#df.head(50)
print(df.columns)

# Checking Dtypes-Non missing values-etc
#df.info()

# Too lazy to fix D-types

# Checking Missing values
#df.isna().sum() / len(df) * 100



# Checking distributions:
#df["age"].value_counts()
#df["gender"].value_counts()
#df["driving_experience"].value_counts()
#df["education"].value_counts()
#df["income"].value_counts()
#print(df[ ((df["credit_score"]<0) | (df["credit_score"]>1)) ]["credit_score"])
#df["vehicle_ownership"].value_counts()
#df["vehicle_year"].value_counts()
#df["married"].value_counts()
#df["children"].value_counts()
#df["postal_code"].value_counts()
#df["annual_mileage"].value_counts()
#df["vehicle_type"].value_counts()
#df["speeding_violations"].value_counts()
#df["duis"].value_counts()
#df["past_accidents"].value_counts()
#df["outcome"].value_counts()
```

```
Index(['id', 'age', 'gender', 'driving_experience', 'education', 'income',
       'credit_score', 'vehicle_ownership', 'vehicle_year', 'married',
       'children', 'postal_code', 'annual_mileage', 'vehicle_type',
       'speeding_violations', 'duis', 'past_accidents', 'outcome'],
      dtype='object')
```

```
#2. Filling missing values

# missing values in credit_score and annual_mileage >5% so the best solution is to
impute
# and not droping the values

# credit_score with 9.82% missing values and annual_mileage with 9.57%
df["credit_score"].fillna(df["credit_score"].median(), inplace = True)
#df["credit_score"].isna().sum()

df["annual_mileage"].fillna(df["annual_mileage"].median(), inplace = True)
#df["annual_mileage"].isna().sum()
```

```
#3. Preparing for modeling

# Creating a list to store the models
models = []

# Storing the features as a variable
features = list(df.columns[1:-1])
#print(features)
```

```
#4. Building and storing the models

#Modeling with a for loop
for f in features:
    model = logit(f"outcome ~ {f}",data = df).fit()
    models.append(model)

#for m in models:
    #print(m.params)
```

```
Optimization terminated successfully.
         Current function value: 0.511794
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.615951
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.467092
         Iterations 8
Optimization terminated successfully.
         Current function value: 0.603742
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.531499
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.572649
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.552412
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.572668
         Iterations 6
Optimization terminated successfully.
         Current function value: 0.586659
         Iterations 5
Optimization terminated successfully.
         Current function value: 0.595431
```

```
#5. Measuring performance

actual_outcomes = df["outcome"]

model_accuracies = []
for model in models:
    #predict
    predicted_outcomes = np.round(model.predict())

    #compare results:
    resu = pd.DataFrame({
        "actual_outcome":actual_outcomes,
        "predicted_outcome":predicted_outcomes
    })
    resu = resu.value_counts(sort=False)
    #print(resu)

    #confusion matrix:
    conf_matrix = model.pred_table()
    #print(conf_matrix)

    #extract values from conf matrix:
    TN = conf_matrix[0,0]
    TP = conf_matrix[1,1]
    FN = conf_matrix[1,0]
    FP = conf_matrix[0,1]

    #calculate and append accuracy:
    accuracy = (TN + TP) / (TN + FN + FP + TP)
    model_accuracies.append(accuracy)
```

```python
#6. Finding the best performing model

#Identifying the index of accuracies with the largest score
best_accuracy = max(model_accuracies)
best_model_index = model_accuracies.index(best_accuracy)

#Mapping the highest accuracy to the feature
best_model = models[best_model_index]
best_feature = best_model.params.index[1].split('[')[0]
#print(best_feature)

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

print(best_feature_df)
```

```
         best_feature  best_accuracy
0  driving_experience         0.7771
```