HOMEWORK

FINAL PROJECT

Theofilus Arifin

Christofer Bryan N. K.

Ramlan Apriyansyah

Muhammad Iqbal

Hanifah Arrasyidah

Christopher Stephen

Muhammad Rizq N. A.

Ujang Pian

I. Modeling

Decision Tree

1. Penjelasan Model
   Dataset yang digunakan adalah dataset yang telah dilakukan preprocessing sebelumnya. Data dibagi menggunakan train_test_split dengan perbandingan 70:30.

Memisahkan Feature dan Target

```
X = dataset.drop(columns=['Response'])
y = dataset['Response']
```

Split Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

2. Hasil Evaluasi (sebelum hyperparameter tuning)
   Dalam hal ini kami berfokus pada metric evaluasi Recall, yaitu meminimalisir nilai dari False Negative. Kemudian menggunakan croos-validation untuk menguji apakah model sudah bisa dikatakan fit.

```
eval(dt)
```

```
Accuracy (test): 0.79
Precision (test): 0.73
Recall (test): 0.91
AUC (test-proba): 0.83
AUC (train-proba): 0.88
roc_auc (crossval train): 0.8766399663306157
roc_auc (crossval test): 0.831875445299015
```

Dari hasil tersebut dapat dilihat bahwa nilai Recall sudah cukup tinggi (91%), namun jika dilihat dari Cross Validation score, model dapat dikatakan mengalami Overfitting.

3. Hyperparameter Tuning
   Hyperparameter yang dilakukan tuning pada model ini yaitu max_depth, min_samples_split, min_samples_leaf, max_features, criterion, dan splitter. Kemudian menggunakan atribut RandomizedSearchCV model di-*fit* untuk mencari kombinasi hyperparameter terbaik.

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from scipy.stats import uniform
import numpy as np

# List of Hyperparameter
max_depth = [int(x) for x in np.linspace(1, 110, num=30)]
min_samples_split = [2, 5, 10, 50, 100]
min_samples_leaf = [1, 2, 4, 10, 20, 50]
max_features = ['auto', 'sqrt']
criterion = ['gini', 'entropy']
splitter = ['best', 'random']

hyperparameters = dict(max_depth=max_depth,
                       min_samples_split=min_samples_split,
                       min_samples_leaf=min_samples_leaf,
                       max_features=max_features,
                       criterion=criterion,
                       splitter=splitter
                       )

# inisialisasi model
dt=DecisionTreeClassifier(random_state=42)
model = RandomizedSearchCV(dt, hyperparameters, cv=5, scoring='recall')
model.fit(X_train, y_train)

# predict and evaluation
y_pred = model.predict(X_test)
eval(model)
```

4. Hasil Evaluasi setelah Hyperparameter Tuning
   Jika dilihat pada metric recall, dapat disimpulkan bahwa tidak ada perubahan score sebelum dan setelah dilakukan hyperparameter tuning. Namun dapat dilihat bahwa nilai (score) Cross Validation mengalami perubahan.
   Terlihat selisih nilai antara Cross Validation untuk data test dan data training lebih kecil sehingga dapat dikatakan bahwa model sudah tidak mengalami overfitting.

```
Metrics for the Test Set:
Accuracy: 0.79
Precision: 0.73
Recall: 0.91
F1-Score: 0.81

Metrics Using Cross Validation:
Mean ROC-AUC (Test): 0.82
Std ROC-AUC (Test): 0.00

Mean ROC-AUC (Train): 0.83
Std ROC-AUC (Train): 0.00
```

Logistic Regression

1. Penjelasan Model
   Dataset yang digunakan dalam model ini adalah dataset yang sudah dilakukan preprocessing dan membainya menjadi data train dan data test dengan perbandingan 70:30.

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

train = pd.read_csv(r'train_after_selection.csv')

X = train.drop(['Response', 'Unnamed: 0'], axis=1)
y = train['Response']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

2. Hasil Evaluasi Model

- Model Evaluasi
  Berikut merupakan untuk mengevaluasi model klasifikasi, khususnya pada kasus ini menggunakan Logistic Regression. Modelini melakukan prediksi pada data test, memberikan skor kinerja model, menghitung ROC_AUC, dan melakukan validasi silang dengan Stratified K-Fold.

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.linear_model import LogisticRegression

def eval_classification(model):
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)
    y_pred_proba = model.predict_proba(X_test)
    y_pred_proba_train = model.predict_proba(X_train)

    print("Accuracy (Test Set): %.2f" % accuracy_score(y_test, y_pred))
    print("Precision (Test Set): %.2f" % precision_score(y_test, y_pred))
    print("Recall (Test Set): %.2f" % recall_score(y_test, y_pred))
    print("F1-Score (Test Set): %.2f" % f1_score(y_test, y_pred))

    print("roc-auc (Test Proba): %.2f" % roc_auc_score(y_test, y_pred_proba[:, 1]))
    print("roc_auc (Train Proba): %.2f" % roc_auc_score(y_train, y_pred_proba_train[:, 1]))

    # StratifiedKFold for cross-validation with stratified sampling
    n_splits = 5
    cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

    # Perform cross-validation
    cv_test_results = cross_validate(model, X_test, y_test, scoring=['roc_auc'],
                                     cv=cv, return_train_score=False)
    cv_train_results = cross_validate(model, X_train, y_train, scoring=['roc_auc'],
                                      cv=cv, return_train_score=False)
```

```python
# Display cross-validation results
print(" ")
print("Metrics Using Cross Validation:")
print(f"Mean ROC-AUC (Test): {cv_test_results['test_roc_auc'].mean():.2f}")
print(f"Std ROC-AUC (Test): {cv_test_results['test_roc_auc'].std():.2f}")
print()
print(f"Mean ROC-AUC (Train): {cv_train_results['test_roc_auc'].mean():.2f}")
print(f"Std ROC-AUC (Train): {cv_train_results['test_roc_auc'].std():.2f}")
```

- Fit Model
  Kemudian menggunakan modul LogisticRegression dari scikit-learn untuk melatih model dengan solver 'liblinear' untuk menyelesaikan masalah optimasi yang muncul selama pelatihan model, setelah itu mengevaluasi kinerjanya dengan fungsi eval_classification yang telah dibuat sebelumnya.

```python
from sklearn.linear_model import LogisticRegression # import logistic regression dari sklearn
logreg = LogisticRegression(solver= 'liblinear') # inisiasi object dengan nama logreg
logreg.fit(X_train, y_train) # fit model regression dari data train
eval_classification(logreg)
```

- Hasil Evaluasi

```
Accuracy (Test Set): 0.78
Precision (Test Set): 0.70
Recall (Test Set): 0.98
F1-Score (Test Set): 0.82
roc-auc (Test Proba): 0.81
roc_auc (Train Proba): 0.82

Metrics Using Cross Validation:
Mean ROC-AUC (Test): 0.81
Std ROC-AUC (Test): 0.01

Mean ROC-AUC (Train): 0.82
Std ROC-AUC (Train): 0.00
```

3. Hasil Evaluasi Model Setelah Hyperparameter Tuning
   Pada Hyperparameter Tuning digunakan Randomized Search pada Logistic Regression dengan parameter penalty ('l1' atau 'l2') dan nilai C antara 0.0001 hingga 0.002. Kemudian mengevaluasi kinerjanya dengan fungsi eval_classification yang telah dibuat sebelumnya.

```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV

penalty = ['l1', 'l2']
C = [float(x) for x in np.linspace(0.0001, 0.002, 100)]
hyperparameters = dict(penalty=penalty, C=C)

logreg = LogisticRegression(solver= 'liblinear')
rs = RandomizedSearchCV(logreg, hyperparameters, scoring='roc_auc', random_state=1, cv=5, n_iter=50)
rs.fit(X_train, y_train)
eval_classification(rs)
```

```
Accuracy (Test Set): 0.78
Precision (Test Set): 0.70
Recall (Test Set): 0.98
F1-Score (Test Set): 0.82
roc-auc (Test Proba): 0.81
roc_auc (Train Proba): 0.82

Metrics Using Cross Validation:
Mean ROC-AUC (Test): 0.81
Std ROC-AUC (Test): 0.01

Mean ROC-AUC (Train): 0.82
Std ROC-AUC (Train): 0.00
```

K-Nearest Neighbors

1. Penjelasan Model

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_std = scaler.transform(X)
```

Semua feature wajib distandardisasi terlebih dahulu sebelum digunakan untuk training model KNN, karena algortima KNN berbasis perhitungan jarak antar data point.

```
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.2, random_state=1)
```

Dataset displit dengan perbandingan 80% data training dan 20% data testing.

```python
def eval_classification(model):
    y_pred = model.predict(X_test)
    y_pred_train = model.predict(X_train)

    print("Metrics for Test Set:")
    print("Accuracy (Test Set): %.2f" % accuracy_score(y_test, y_pred))
    print("Precision (Test Set): %.2f" % precision_score(y_test, y_pred))
    print("Recall (Test Set): %.2f" % recall_score(y_test, y_pred))
    print("F1-Score (Test Set): %.2f" % f1_score(y_test, y_pred))

    cv_roc_auc = cross_validate(model, X_std, y, cv=5, scoring='roc_auc', return_train_score=True)

    print("Metrics Using Cross Validation:")
    print('Mean ROC-AUC (Test): '+ str(cv_roc_auc['test_score'].mean()))
    print('Standard deviation ROC-AUC (Test): '+ str(cv_roc_auc['test_score'].std()))

    print('Mean ROC-AUC (Train): '+ str(cv_roc_auc['train_score'].mean()))
    print('Standard deviation ROC-AUC (Train): '+ str(cv_roc_auc['train_score'].std()))
```

Function di atas digunakan untuk mengevaluasi performa model KNN. Function mengeluarkan metrics seperti accuracy, precision, recall dan F1-score untuk data testing. Untuk memastikan semua subset dari data terpakai untuk training dan testing, function di atas juga mengimplementasikan cross validation dengan 5 fold. ROC-AUC score diambil dari rata-rata ROC-AUC score untuk training dan testing yang dihasilkan dari cross validation.

2. Hasil evaluasi sebelum hyperparameter tuning

```python
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
eval_classification(knn)
```

```
Metrics for Test Set:
Accuracy (Test Set): 0.78
Precision (Test Set): 0.73
Recall (Test Set): 0.87
F1-Score (Test Set): 0.80
Metrics Using Cross Validation:
Mean ROC-AUC (Test): 0.5516357975673574
Standard deviation ROC-AUC (Test): 0.0015958711066472381
Mean ROC-AUC (Train): 0.5657856819528391
Standard deviation ROC-AUC (Train): 0.0008022936040659267
```

3. Hyperparameter tuning

```
n_neighbors = list(range(1,51))
p=[1,2]
algorithm = ['auto', 'ball_tree', 'kd_tree', 'brute']
hyperparameters = dict(n_neighbors=n_neighbors, p=p, algorithm=algorithm)

knn = KNeighborsClassifier()
randomized_search = RandomizedSearchCV(knn, hyperparameters, scoring='roc_auc', random_state=45, cv=5)
randomized_search.fit(X_train, y_train)
```

Best hyperparameter after tuning

```
1 randomized_search.best_estimator_.get_params()
```

```
{'algorithm': 'brute',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 43,
 'p': 1,
 'weights': 'uniform'}
```

4. Hasil evaluasi setelah hyperparameter tuning

```
1 eval_classification(randomized_search.best_estimator_)
```

```
Metrics for Test Set:
Accuracy (Test Set): 0.80
Precision (Test Set): 0.73
Recall (Test Set): 0.94
F1-Score (Test Set): 0.82
Metrics Using Cross Validation:
Mean ROC-AUC (Test): 0.6416010007778299
Standard deviation ROC-AUC (Test): 0.0035782597563321465
Mean ROC-AUC (Train): 0.6468413971243454
Standard deviation ROC-AUC (Train): 0.0028485365536093576
```

AdaBoost

Hasil evaluasi sebelum hyperparameter tunning

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier()
clf.fit(X_train, y_train)
eval_classification(clf,X_train, X_test, y_train, y_test)
```

✓ 32.0s

```
Training Set:
Accuracy: 0.80
Precision: 0.73
Recall: 0.95
F1-Score: 0.82
ROC AUC: 0.85

Test Set:
Accuracy: 0.80
Precision: 0.73
Recall: 0.95
F1-Score: 0.82
ROC AUC: 0.84

Metrics Using Cross Validation:
Mean ROC-AUC (Test): 0.84
Std ROC-AUC (Test): 0.00

Mean ROC-AUC (Train): 0.85
Std ROC-AUC (Train): 0.00
```

1. **Hyperparameter tunning**

   Melakukan Hyperparameter TunNing dengan menggunakan metode RandomizedSearchCV, parameter yang di tunning antara lain :

   - n_estimator (start di nilai 50 dan stop di nilai 200),
   - Learning_rate (start di 0,001 dan stop di 0,1)
   - Algorithm (SAMME dan SAMME.R)

```python
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
import numpy as np
(variable) hyperparameters: dict[str, list[int] | list[float] | list[str]]
hyperparameters = dict(n_estimators = [int(x) for x in np.linspace(start = 50, stop = 200, num = 200)],
                       learning_rate = [float(x) for x in np.linspace(start = 0.001, stop = 0.1, num = 200)],
                       algorithm = ['SAMME', 'SAMME.R']
                      )

# Init model
ab = AdaBoostClassifier(random_state=42)
ab_tuned = RandomizedSearchCV(ab, hyperparameters, random_state=42, cv=5, scoring='recall')
ab_tuned.fit(X_train,y_train)

print(f'Best Hyperparameters: {ab_tuned.best_params_}')

# Predict & Evaluation
eval_classification(ab_tuned, X_train, X_test, y_train, y_test)
```

2. **Hasil setelah hyperparameter tunning**

   a. Best Hyperparameter

```
Best Hyperparameters: {'n_estimators': 98, 'learning_rate': 0.016422110552763818, 'algorithm': 'SAMME'}
```

   b. Evaluasi setelah hyperparameter tunning

5. Kesimpulan

Score evaluasi metric sebelum dilakukan Hyperparameter Tunning lebih baik dari pada setelah dilakukan Hyperparameter Tunning.

XGBoost

XGBoost, atau Extreme Gradient Boosting, merupakan algoritma machine learning yang terkenal karena kinerjanya yang sangat baik dalam berbagai kompetisi data science. Salah satu keunggulan utamanya adalah kemampuan untuk mengatasi overfitting melalui penerapan regularisasi L1 (LASSO) dan L2 (ridge). Selain itu, XGBoost dirancang untuk parallel dan distributed computing, memungkinkannya efisien dan scalable untuk dataset besar.

Berikut adalah tabel hasil dari sebelum dan sesudah hyperparameter tunning.

Metric	Sebelum Tuning	Sesudah Tuning
Accuracy	0.8	0.8
Precision	0.73	0.73
Recall	0.93	0.94
F1-Score	0.82	0.82
Mean ROC-AUC (Test)	0.83	0.84
Std ROC-AUC (Test)	0.01	0.01
Mean ROC-AUC (Train)	0.85	0.85
Std ROC-AUC (Train)	0	0

Model XGBoost menunjukkan peningkatan yang positif setelah hyperparameter tuning. Terjadi peningkatan pada Recall, F1-Score, dan Mean ROC-AUC pada Test set. Hal ini menunjukkan bahwa tuning parameter telah berhasil meningkatkan kemampuan model dalam mengidentifikasi positive instances dan kinerja keseluruhan.

CatBoost

CatBoost, singkatan dari Categorical Boosting, adalah algoritma gradient boosting yang dikembangkan oleh Yandex. Yang membedakan CatBoost adalah kemampuannya untuk menangani variabel kategorikal tanpa perlu preprocessing yang ekstensif. Algoritma ini dirancang untuk efisiensi tinggi dan kecepatan, dengan dukungan built-in untuk mencegah overfitting, mengurangi kebutuhan untuk penyesuaian manual terhadap hyperparameter.

Berikut adalah tabel hasil dari sebelum dan sesudah hyperparameter tunning.

Metric	Sebelum Tuning	Sesudah Tuning
Accuracy	0.80	0.80
Precision	0.73	0.73
Recall	0.93	0.94
F1-Score	0.82	0.82
Mean ROC-AUC (Test)	0.84	0.84
Std ROC-AUC (Test)	0.01	0.01
Mean ROC-AUC (Train)	0.85	0.85
Std ROC-AUC (Train)	0.00	0.00

CatBoost menunjukkan konsistensi kinerja antara sebelum dan sesudah hyperparameter tuning. Meskipun tidak ada peningkatan yang signifikan, model tetap mempertahankan tingkat performa yang baik, menunjukkan ketangguhan algoritma dalam menangani berbagai kondisi.

LightGBM

LightGBM, dikembangkan oleh Microsoft, adalah framework gradient boosting yang fokus pada pelatihan yang efisien dan distribusi pada dataset besar. Salah satu keunikan LightGBM terletak pada pertumbuhan pohon berbasis daun, yang dapat meningkatkan kecepatan konvergensi dan mengurangi biaya komputasi. Algoritma ini juga mendukung fitur kategorikal tanpa preprocessing tambahan, menjadikannya pilihan yang kuat untuk tugas machine learning pada skala besar.

Berikut adalah tabel hasil dari sebelum dan sesudah hyperparameter tunning.

Metric	Sebelum Tuning	Sesudah Tuning
Accuracy	0.80	0.80
Precision	0.73	0.73
Recall	0.94	0.94
F1-Score	0.82	0.82
Mean ROC-AUC (Test)	0.84	0.84
Std ROC-AUC (Test)	0.01	0.01

Mean ROC-AUC (Train)	0.85	0.85
Std ROC-AUC (Train)	0.00	0.00

LightGBM menunjukkan kinerja yang sangat konsisten sebelum dan sesudah hyperparameter tuning. Tingginya Recall dan Mean ROC-AUC pada Test set menunjukkan bahwa model ini sangat baik dalam mengidentifikasi positive instances dan memberikan hasil yang stabil.

II. Best Result

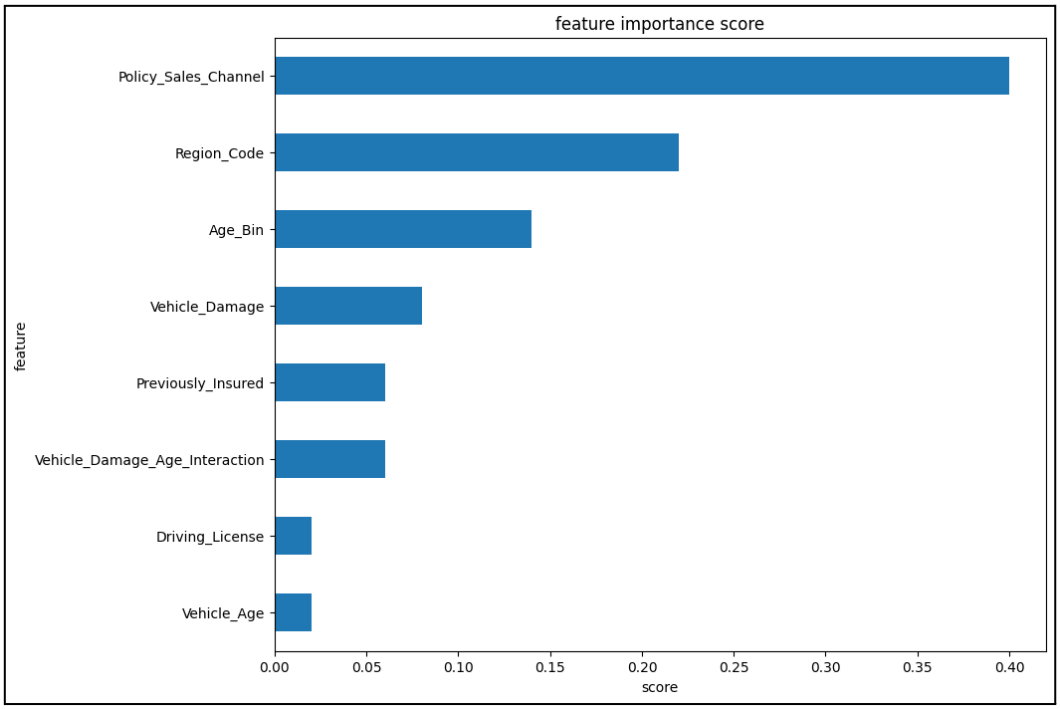
Metric	Ada Boost	Random Forest	XG Boost	Light GBM	Cat Boost	K nearest neighbors
Accuracy	0.8	0.8	0.8	0.8	0.8	0.8
Precision	0.73	0.73	0.73	0.73	0.73	0.73
Recall	0.95	0.93	0.94	0.94	0.93	0.94
F1-Score	0.82	0.82	0.82	0.82	0.82	0.82
Mean ROC-AUC (Test)	0.84	0.84	0.84	0.84	0.84	0.64
Std ROC-AUC (Test)	0	0.01	0.01	0.01	0.01	0
Mean ROC-AUC (Train)	0.85	0.85	0.85	0.85	0.85	0.65
Std ROC-AUC (Train)	0	0	0	0	0	0

Dari hasil eksperimen kami, terlihat bahwa Adaboosting mencapai nilai Recall tertinggi dibandingkan dengan algoritma lain yang diuji. Oleh karena itu, kami memilih untuk mengimplementasikan model menggunakan Adaboosting. Keputusan ini didasarkan pada prioritas kami untuk meningkatkan kemampuan model dalam mengidentifikasi pelanggan yang benar-benar tertarik, dengan menghindari False Negatives sebanyak mungkin.

Pemilihan Adaboosting diharapkan dapat memberikan keunggulan dalam konteks penawaran Asuransi Kendaraan, dengan meningkatkan akurasi dalam mengenali pelanggan yang berpotensi tertarik. Kami menyadari pentingnya Recall dalam situasi ini, dan keputusan ini sejalan dengan upaya kami untuk memaksimalkan kemampuan model dalam hal tersebut.

Penting untuk terus memantau dan merawat model guna memastikan kinerjanya tetap optimal dan sesuai dengan perkembangan bisnis. Kami berkomitmen untuk menjaga model ini agar tetap relevan dan efektif seiring berjalannya waktu.

III. Feature Importance



1. Policy Sales Channel:
   - Business Insight: Policy sales channel memiliki dampak besar terhadap hasil model, menunjukkan bahwa jalur penjualan tertentu memiliki pengaruh signifikan terhadap respons pelanggan terhadap penawaran asuransi kendaraan.
   - Rekomendasi: Perlu dilakukan analisis lebih lanjut untuk memahami karakteristik dan preferensi pelanggan yang menggunakan saluran penjualan tertentu. Upaya pemasaran dapat difokuskan pada peningkatan efektivitas saluran penjualan yang memberikan hasil positif.

2. Region Code:
   - Business Insight: Region code juga memiliki pengaruh yang signifikan pada model, menunjukkan bahwa lokasi geografis memainkan peran penting dalam respons pelanggan.
   - Rekomendasi: Analisis lebih lanjut mengenai karakteristik dan preferensi pelanggan di berbagai wilayah dapat membantu mengarahkan strategi pemasaran yang lebih lokal dan disesuaikan.

3. Age Bin:
   - Business Insight: Kategori usia (Age Bin) memengaruhi hasil model, menunjukkan bahwa segmentasi usia dapat menjadi faktor kunci dalam menentukan respons pelanggan.
   - Rekomendasi: Pemahaman lebih lanjut tentang preferensi dan kebutuhan asuransi kendaraan berdasarkan kategori usia dapat membimbing pengembangan produk yang lebih sesuai dan strategi pemasaran yang lebih efektif.

4. Vehicle Damage:

- Business Insight: Kondisi kerusakan kendaraan (Vehicle Damage) memiliki dampak besar pada respons pelanggan, menunjukkan bahwa pelanggan yang kendaraannya telah rusak mungkin lebih cenderung tertarik pada penawaran asuransi.
- Rekomendasi: Fokus pemasaran dapat ditempatkan pada segmen pelanggan yang belum memiliki asuransi dan telah mengalami kerusakan kendaraan sebelumnya.

5. Previously Insured:
- Business Insight: Status asuransi sebelumnya (Previously Insured) juga merupakan faktor kunci dalam respons pelanggan, menandakan bahwa pelanggan yang belum diasuransikan sebelumnya cenderung merespons lebih positif.
- Rekomendasi: Strategi pemasaran dapat lebih difokuskan pada pelanggan yang belum memiliki asuransi sebelumnya, dengan menekankan manfaat dan nilai tambah dari produk asuransi kendaraan.