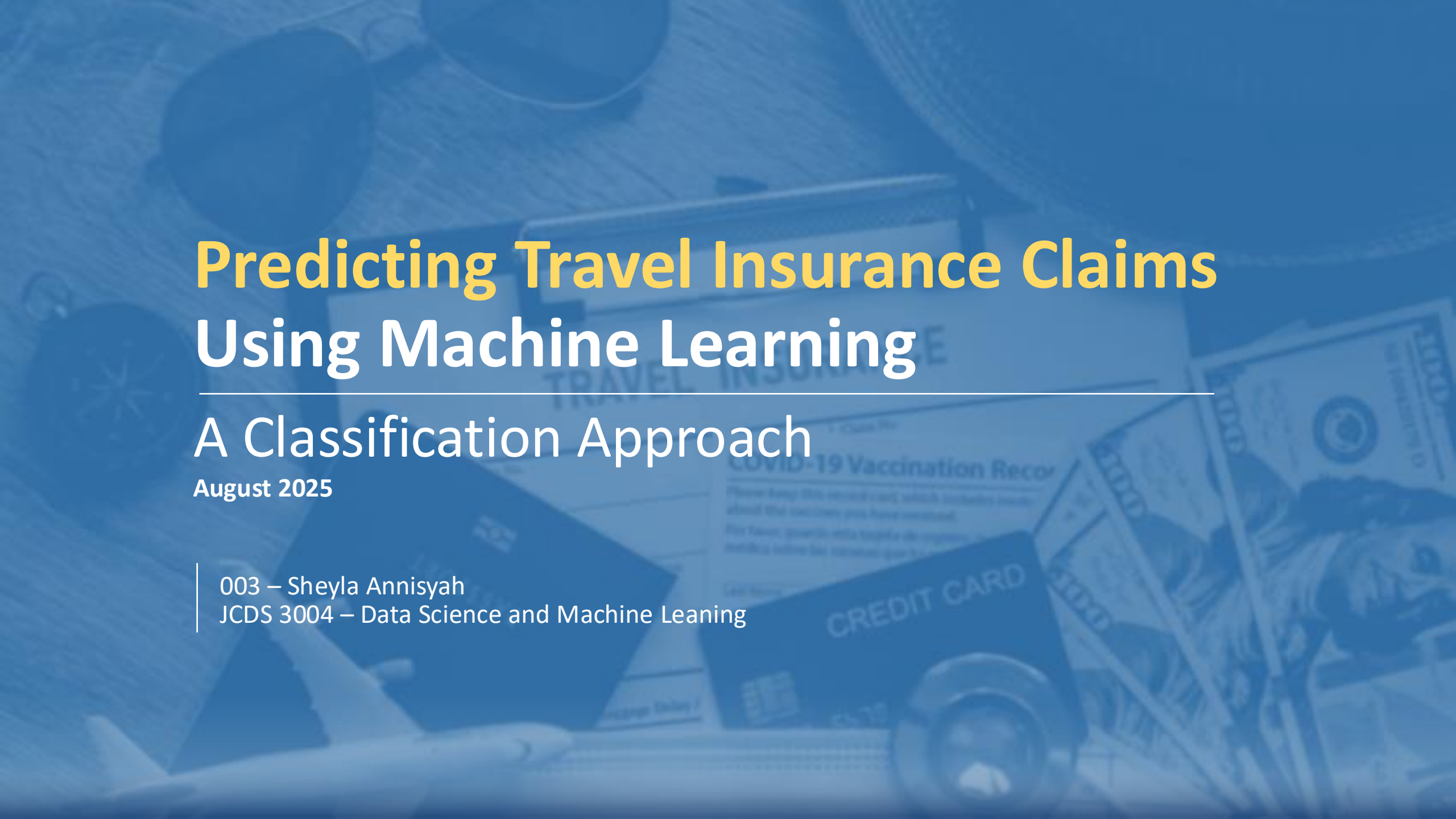


Predicting Travel Insurance Claims Using Machine Learning

A Classification Approach

August 2025

003 – Sheyla Annisyah
JCDS 3004 – Data Science and Machine Leaning

WHAT IS TRAVEL INSURANCE?

ASURANSI PERJALANAN

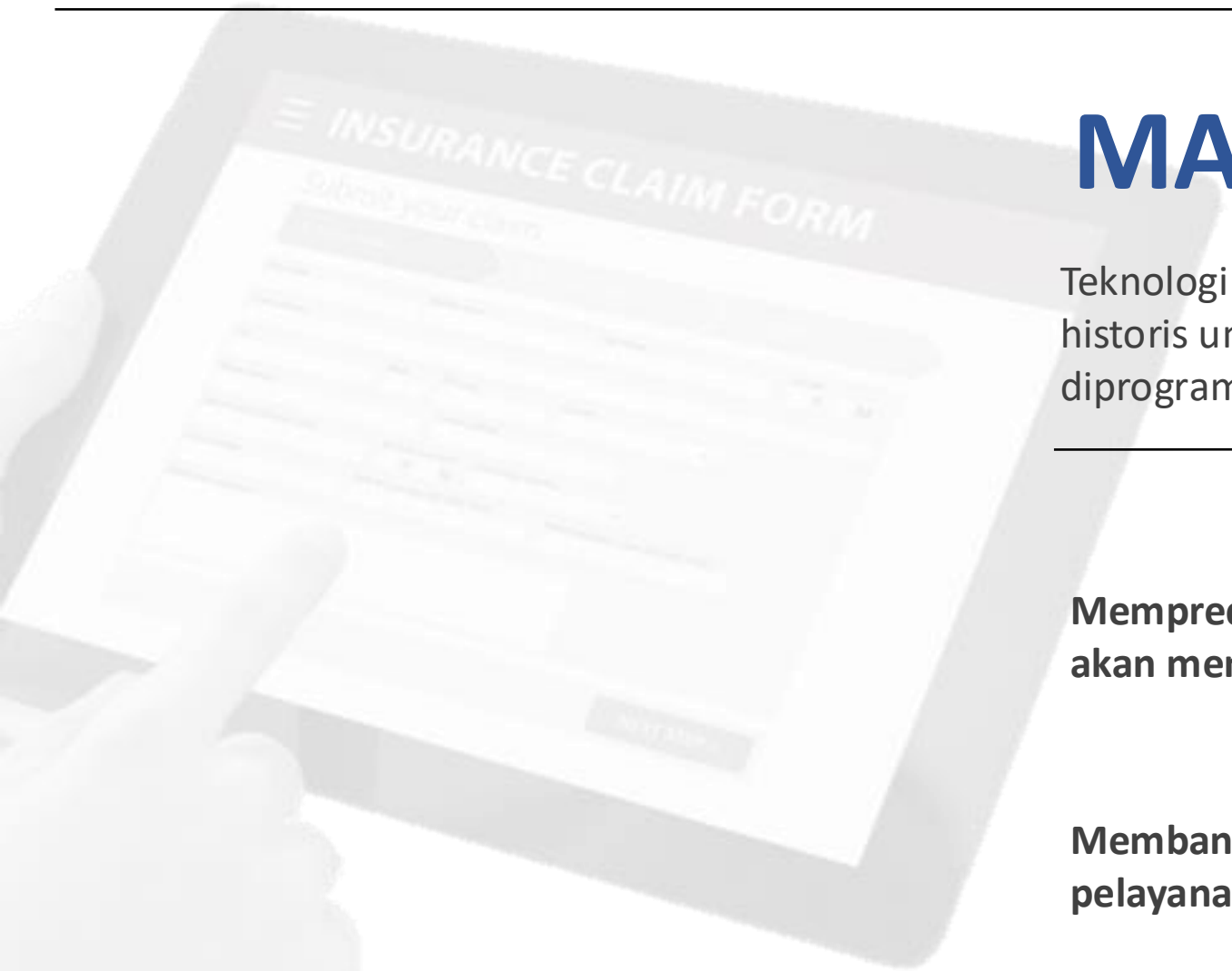
merupakan jenis asuransi yang memberikan perlindungan selama kita bepergian, baik di dalam negeri maupun di luar negeri.

Customer membayar premi → **Perusahaan memberikan perlindungan perjalanan** → **Customer Claim/Tidak asuransi**

PROBLEM STATEMENT

Tanpa strategi prediksi klaim, proses administrasi asuransi perjalanan menjadi tidak efisien dan berisiko **memboroskan sumber daya pada pemegang polis yang tidak pernah mengajukan klaim.**

HOW MACHINE LEARNING HELPS TRAVEL INSURANCE?



MACHINE LEARNING

Teknologi yang memungkinkan komputer belajar dari data historis untuk membuat prediksi atau keputusan tanpa diprogram secara eksplisit.

Memprediksi kemungkinan pemegang polis asuransi perjalanan akan mengajukan klaim

Membantu perusahaan mengoptimalkan strategi bisnis & pelayanan.

METRIC EVALUATION

	Predicted: No Claim	Predicted: Claim
Actual: No Claim	True Negative (TN)	False Positive (FP)
Actual: Claim	False Negative (FN)	True Positive (TP)

Type 1 Error (False Positive)

Definisi:
Model memprediksi seorang pelanggan akan mengajukan klaim, padahal kenyataannya tidak.

Konsekuensi:
Mengakibatkan biaya operasional sia-sia.

Type 2 Error (False Negative)

Definisi:
Model memprediksi seorang pelanggan tidak akan mengajukan klaim, kenyataannya mengajukan klaim.

Konsekuensi:
Perusahaan gagal mempersiapkan dana dan strategi untuk menanggung klaim dari pelanggan tersebut

DATA INFO

Feature	Description
Agency	Name of agency.
Agency Type	Type of travel insurance agencies.
Distribution Channel	Channel of travel insurance agencies.
Product Name	Name of the travel insurance products.
Gender	Gender of insured.
Duration	Duration of travel.
Destination	Destination of travel.
Net Sales	Amount of sales of travel insurance policies.
Commission (in value)	Commission received for travel insurance agency.
Age	Age of insured.
Claim	Claim status.

SUMBER DATA

Travel Insurance

Updated 7 years ago

https://www.kaggle.com/datasets/mhdzahier/travel-insurance

44.328 Data

11 Kolom

Yes Claim: 1.5%

No Claim: 98.5%

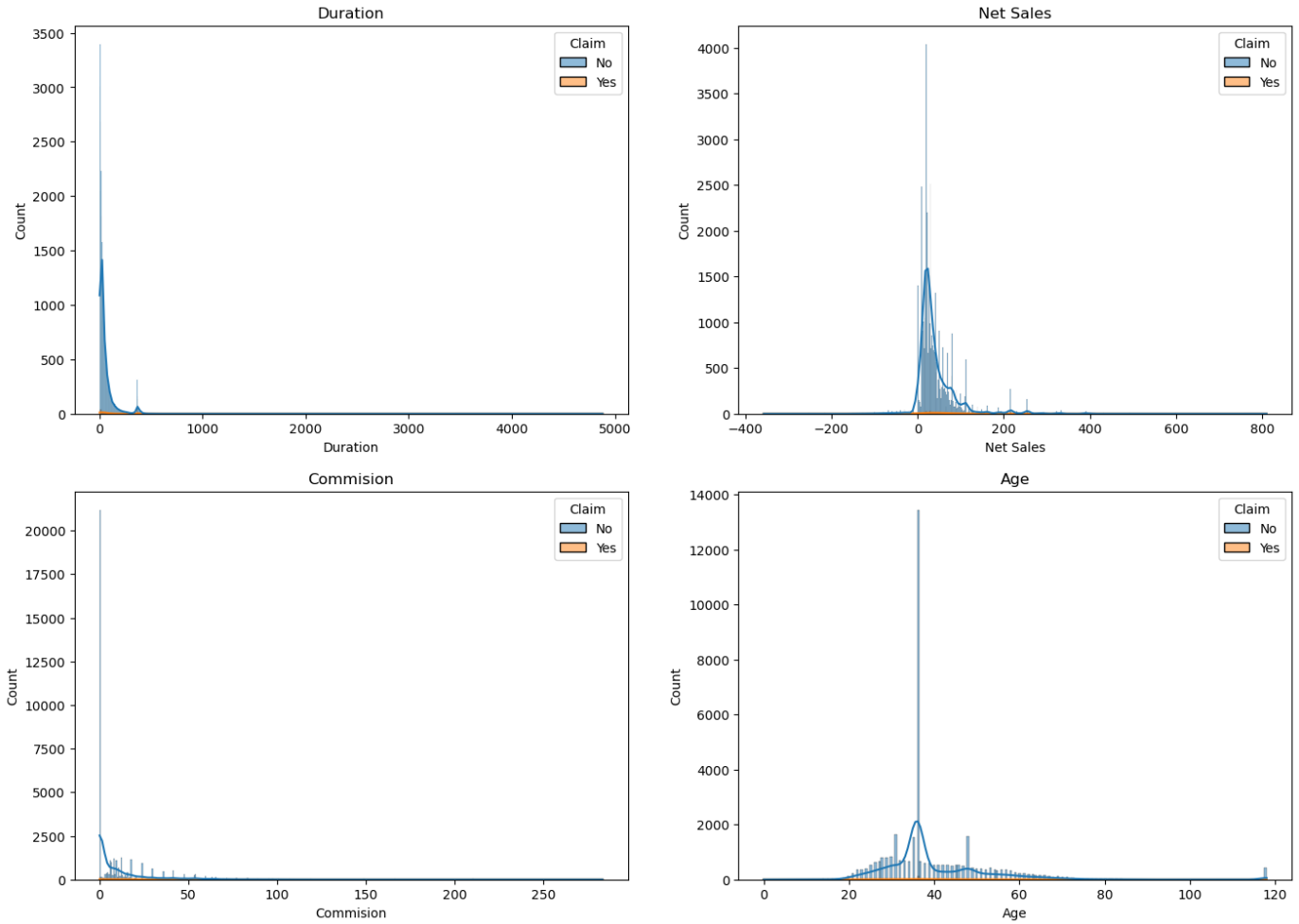
Data Imbalanced

Missing Value: 31.647 data jenis kelamin (Gender) → dihapus karena tidak memiliki korelasi pada kolom lainnya

Data Duplicate: 5.004 data yang sama → dihapus agar model tidak bias

Data Anomali: Menghapus data durasi yang minus

DATA OUTLIER



Data Sebelum Outlier Dihapus

Hapus: Duration > 365 Hari

Rata-rata durasi perjalanan yang ditanggung asuransi adalah 90-180 hari, tetapi beberapa asuransi menanggung 1 tahun.

Hapus: Age > 80

Rata-rata asuransi cover hingga umur 76 tahun. Rata-rata hidup manusia umumnya 80 tahun.

Hapus: Commision > 40%

Komisi umumnya yang diterima perusahaan asuransi berkisar 15-40%.

Hapus: Net Sales <-100 & >400 USD

Net sales rata-rata asuransi travel adalah 328 USD, net sales negative dapat terjadi karena adanya refund.

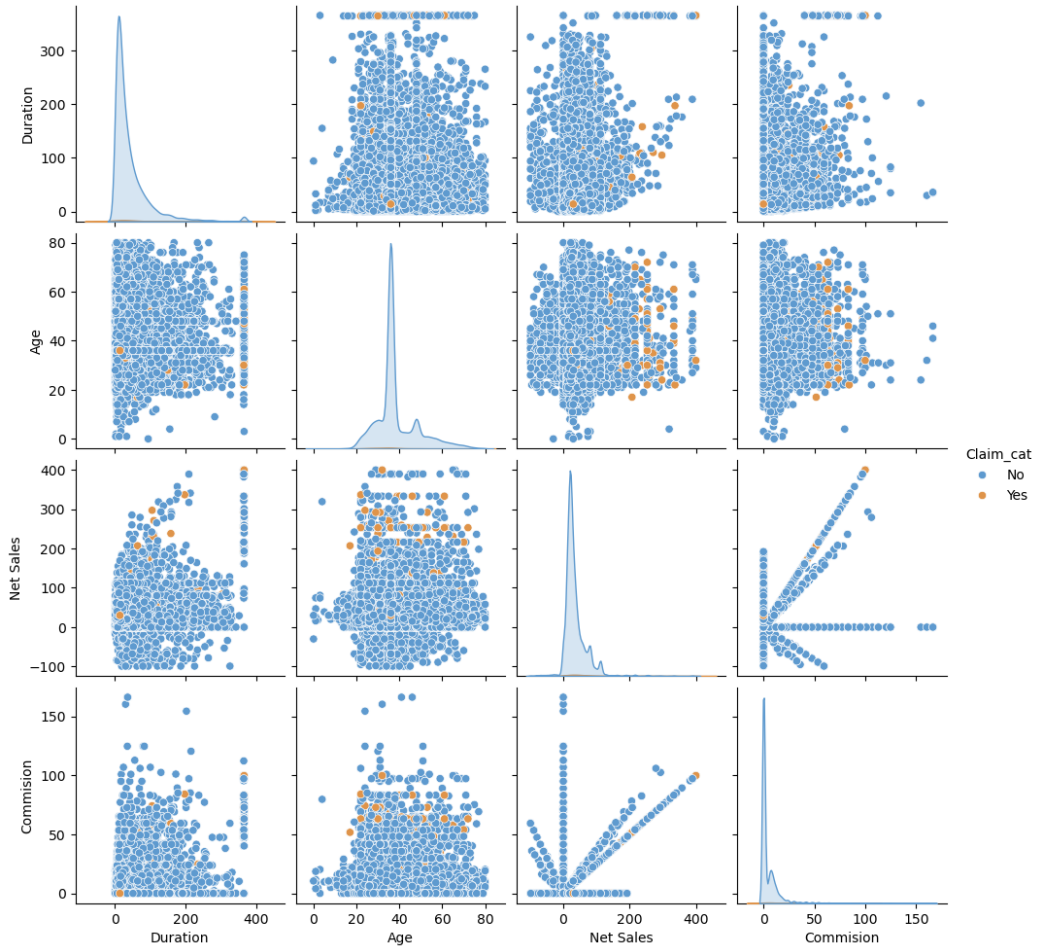
NUMERICAL DATA CORRELATION ANALYSIS

Destination
- Kebanyakan perjalanan pendek, baik yang mengajukan klaim maupun tidak.

Age
- Usia mayoritas pelanggan berada di 30–40 tahun.

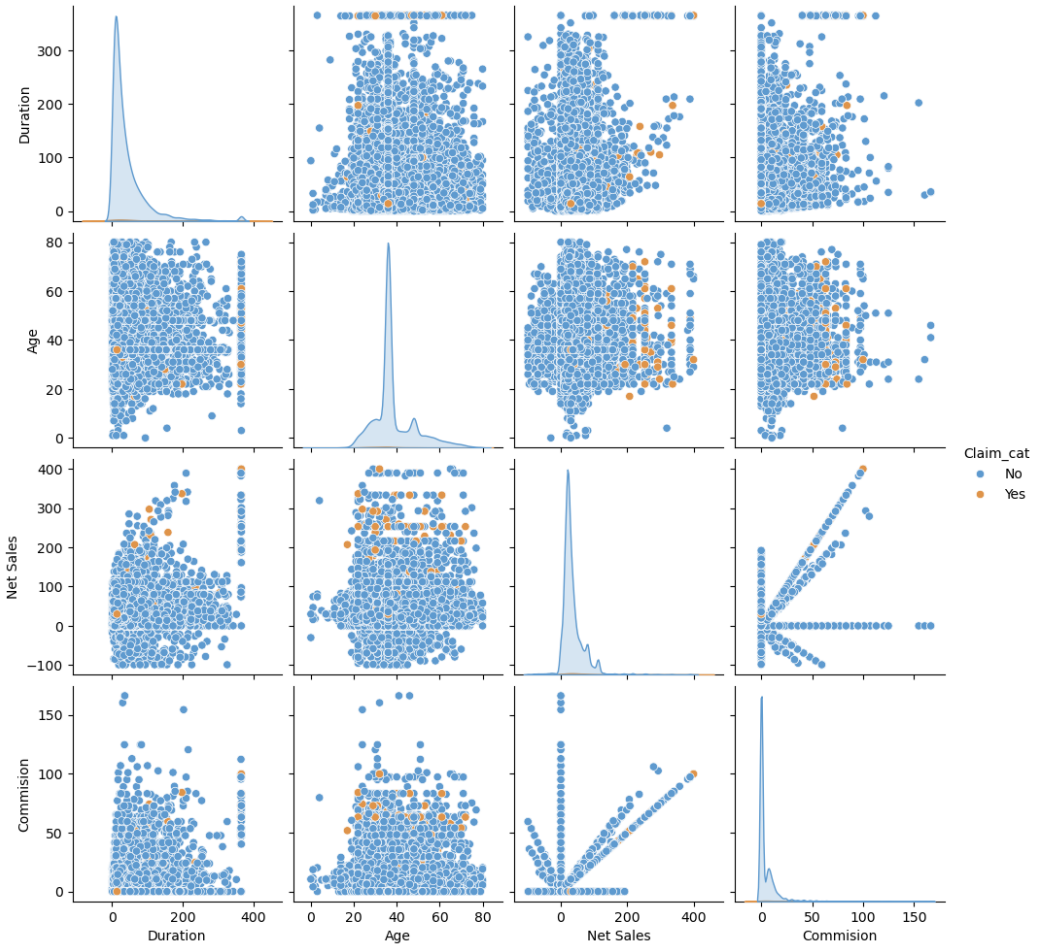
Net Sales
- Sebaran net Sales yang didapatkan perusahaan dari kelompok kedua kelompok adalah <100 USD (transaksi kecil)

Komisi
- Terlihat bahwa sebagian besar komisi yang dikeluarkan untuk agen travel adalah 0, hal ini logis karena Net sales yang didapatkan juga terbilang kecil (<100 USD).

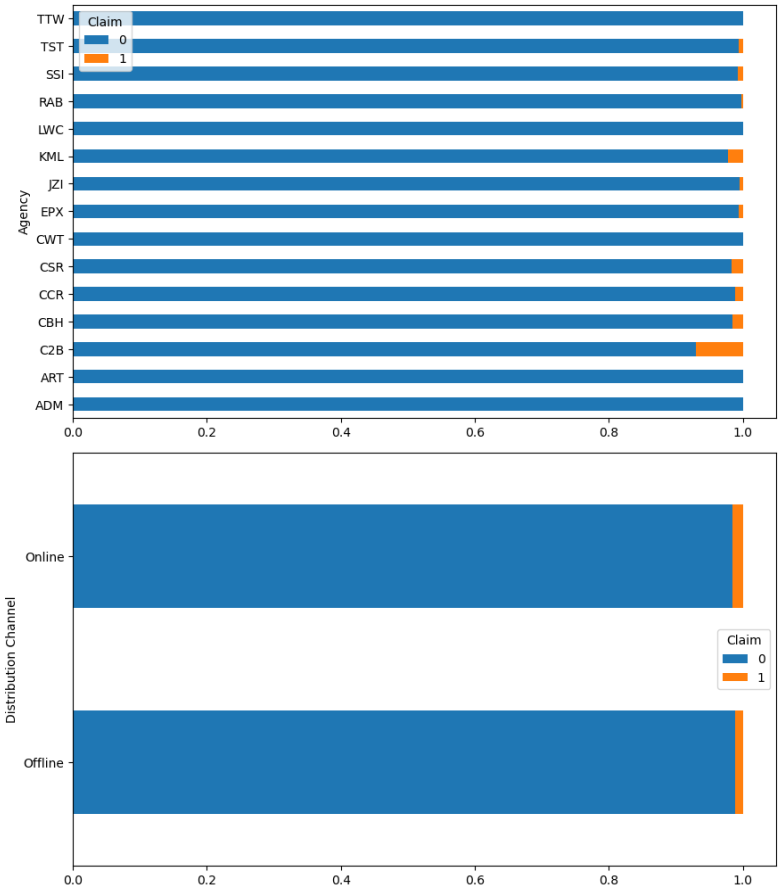


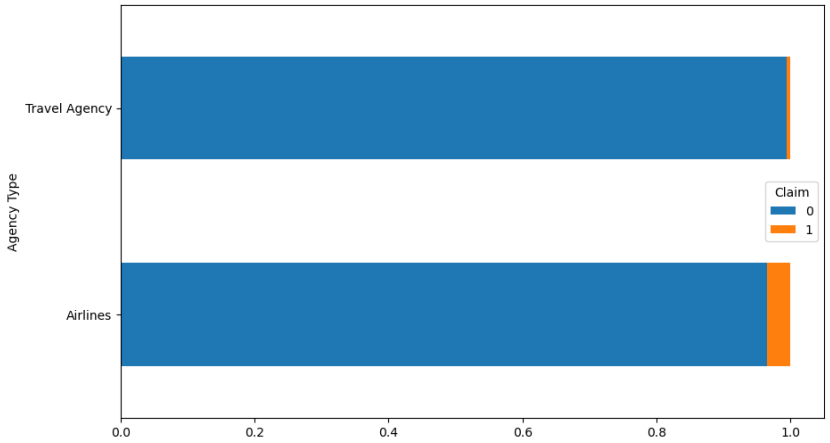
NUMERICAL DATA CORRELATION ANALYSIS

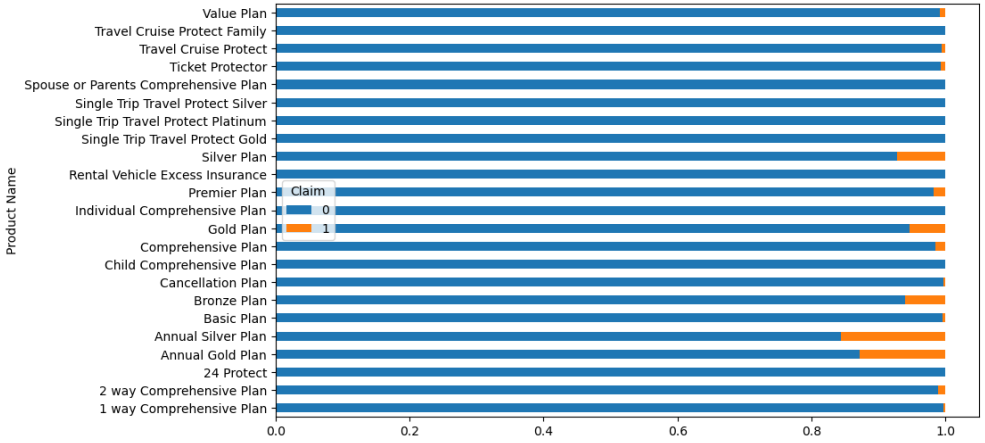
- **Duration vs Age →** Data tersebar cukup merata, tapi sebagian besar transaksi berada di durasi rendah (0–200). **Tidak ada tren kuat antara umur dan durasi.**
- **Duration vs Net Sales → Beberapa transaksi durasi rendah tapi penjualan tinggi.** Secara umum, klaim “Yes” muncul lebih sering pada durasi rendah–sedang dengan variasi net sales.
- **Duration vs Commission →** Mayoritas data berkonsentrasi di **durasi rendah dan komisi rendah.**
- **Age vs Net Sales →** Penjualan tinggi cenderung muncul pada **usia menengah (20–60).**
- **Age vs Commission → Komisi tersebar merata di seluruh usia.**
- **Net Sales vs Commission →** Ada pola diagonal dari (0,0) ke atas karena **komisi dihitung proporsional dari net sales.**



CATEGORICAL DATA CORRELATION ANALYSIS







Agency

- Beberapa agen seperti C2B **menunjukkan proporsi Yes Claim** yang relatif lebih besar dibanding agen lain.

Agensi Type

- **Airlines** menunjukkan proporsi klaim "Yes" **sedikit lebih tinggi** daripada Travel Agency

Distribution Channel

- Pelanggan **offline** sedikit lebih mungkin **mengajukan klaim** dibanding online.

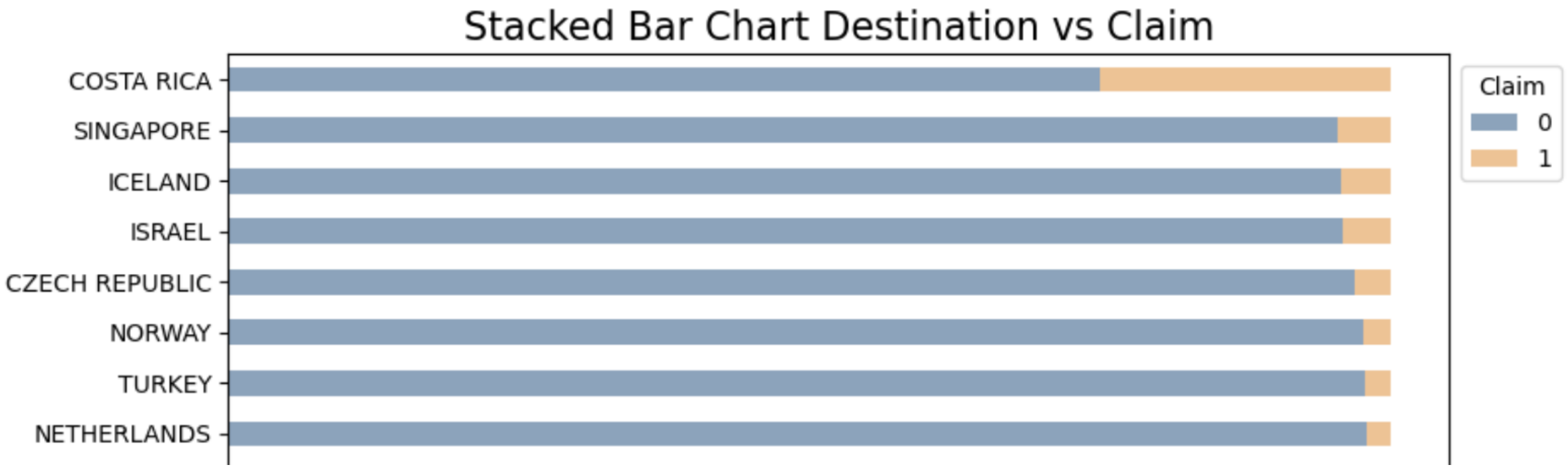
Product Name

- **Annual Travel Protect Gold** dan **Annual Gold Plan** terlihat **lebih banyak klaim** dibanding produk lain.

CATEGORICAL DATA CORRELATION ANALYSIS

Destination

Berdasarkan destination, ternyata klaim paling banyak berasal dari **Costa Rica, Singapore,** dan **Iceland**, yang dipengaruhi oleh jenis perjalanan dan aktivitas.



FEATURE ENGINEERING: ENCODING

Encoding

Encoding dilakukan pada data kategorikal.

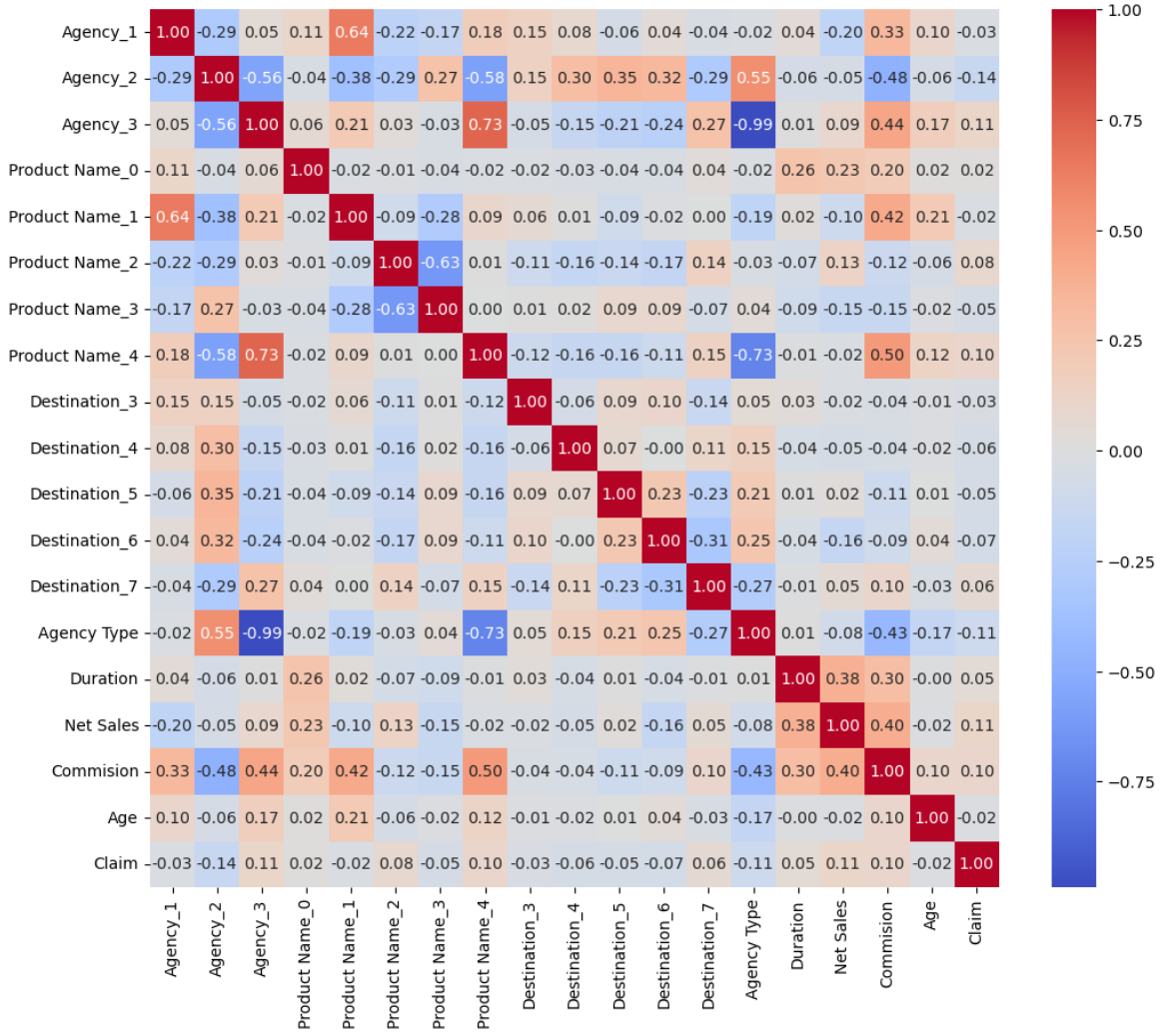
- One-hot encoding: Agency Type dan Distribution Channel
- Binary encoding: Agency, Product Name, dan Destination

Split Data

- 80% data dipakai untuk training
- 20% data dipakai untuk testing

Feature Selection

- SelectPercentile 80% untuk fitur terbaik
- Dari 23 Fitur dipilih menjadi **18 fitur terbaik**



MODELING & EVALUATION TEST

```
# Coba semua model klasifikasi
logreg = LogisticRegression(max_iter=4000, random_state=2023)
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier(random_state=2023)
rf = RandomForestClassifier(random_state=2023)
xgb = XGBClassifier(random_state=2023)
lgbm = LGBMClassifier(random_state=2023)

smote = SMOTE(random_state=2023)
scaler = RobustScaler()

logreg_penalized = LogisticRegression(class_weight = 'balanced', random_state=2023)
```

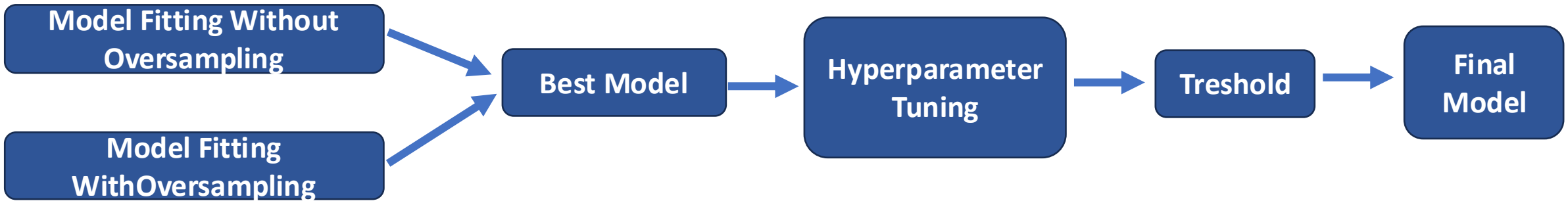
[733] Python

Algoritma

- Logistic Regression,
- KNN,
- Decision Tree,
- Random Forest,
- XGBoost,
- LightGBM

Evaluasi

- **Pipeline**
- **Stratified K-fold CrossVal**



MODELING FITTING WITHOUT OVERFITTING

```python
# Model Fitting tanpa oversampling
models = [logreg, knn, dt, rf, xgb, lgbm]

rocauc_mean_no_sampling = []
rocauc_std_no_sampling = []

for i in models:
    pipe_model_no_sampling = Pipeline([
        ('scaler', RobustScaler()),
        ('algorithm', i)
    ])

    skfold = StratifiedKFold(n_splits=5)

    model_no_sampling = cross_val_score(
        pipe_model_no_sampling,
        x_train_selected,
        y_train,
        cv=skfold,
        scoring='roc_auc',
        n_jobs=-1
    )

    rocauc_mean_no_sampling.append(model_no_sampling.mean())
    rocauc_std_no_sampling.append(model_no_sampling.std())

pd.DataFrame({
    'model': ['Logistic Regression', 'KNN', 'Decision Tree', 'Random Forest', 'XGBoost', 'LightGBM'],
    'Mean': rocauc_mean_no_sampling,
    'Std': rocauc_std_no_sampling,
}).set_index('model').sort_values(by='Mean', ascending=False)
```

Robust Scaller

Data train yang sudah memiliki feature selection

ROC-AUC MEAN

model	Mean	Std
Logistic Regression	0.831049224265373	0.013189874587767918
LightGBM	0.8030571290800002	0.025493509049893483
XGBoost	0.7804118805647401	0.020933669669594715
Random Forest	0.6953255028733654	0.0294044748019583
KNN	0.5910100772710132	0.013185320604786301
Decision Tree	0.5272813611348098	0.008430969501716106

- **Logistic Regression paling kuat dan stabil** di dataset tanpa oversampling.
- Model berbasis tree seperti **Random Forest dan Decision Tree kurang efektif** untuk data ini, mungkin karena imbalance.
- **Model boosting** (LightGBM, XGBoost) cukup baik tapi masih **di bawah Logistic Regression.**

MODELING FITTING WITHOUT OVERFITTING

```
models = [logreg, knn, dt, rf, xgb, lgbm]
rocauc_test_nosampling_noselect1 = []
rocauc_train_nosampling_noselect1 = []

def y_pred_func(model):
    estimator = Pipeline([
        ('scaler', scaler),
        ('algorithm', model)
    ])

    estimator.fit(x_train_selected,y_train)
    return estimator

for model in models:
    estimator = y_pred_func(model)
    y_pred_test = estimator.predict_proba(x_test_selected)[:, 1]
    y_pred_train = estimator.predict_proba(x_train_selected)[:, 1]
    rocauc_test_nosampling_noselect1.append(roc_auc_score(y_test, y_pred_test))
    rocauc_train_nosampling_noselect1.append(roc_auc_score(y_train, y_pred_train))

pd.DataFrame({
    'model': ['Logistic Regression','KNN','Decision Tree','Random Forest','XGBoost','LightGBM'],
    'Train ROC-AUC': rocauc_train_nosampling_noselect1,
    'Validation ROC-AUC': rocauc_mean_no_sampling,
    'Test ROC-AUC': rocauc_test_nosampling_noselect1,
}).set_index('model').sort_values(by='Validation ROC-AUC', ascending=False)
```

Test Set Validation

model	Train ROC-AUC	Validation ROC-AUC	Test ROC-AUC
Logistic Regression	0.8389210750037046	0.831049224265373	0.7870442576783189
LightGBM	0.9872716792328038	0.8030571290800002	0.7758693359769651
XGBoost	0.9861176482458947	0.7804118805647401	0.7519361550313194
Random Forest	0.9998816885770321	0.6953255028733654	0.6283237459587796
KNN	0.9764411389789249	0.5910100772710132	0.5591809898464336
Decision Tree	0.9998860411711881	0.5272813611348098	0.5261779526166902

- **Logistic Regression** → Stabil, mudah dijelaskan, performa test tertinggi (0.787).
- **LightGBM** → Overfit (train 0.987) tapi test cukup kuat (0.776).
- **XGBoost** → Mirip LightGBM, test lebih rendah (0.752).
- **Random Forest** → Overfit parah, test rendah (0.628).
- **KNN** → Overfit, test lemah (0.591).
- **Decision Tree** → Overfit ekstrem, test nyaris acak (0.52).

MODELING FITTING WITHOUT OVERFITTING

Final Test Validation

Logistic Regression

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	6464
1.0	0.00	0.00	0.00	98
accuracy			0.99	6562
macro avg	0.49	0.50	0.50	6562
weighted avg	0.97	0.99	0.98	6562

Decision Tree

	precision	recall	f1-score	support
0.0	0.99	0.98	0.98	6464
1.0	0.06	0.07	0.06	98
accuracy			0.97	6562
macro avg	0.52	0.53	0.52	6562
weighted avg	0.97	0.97	0.97	6562

XGBoost

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	6464
1.0	0.00	0.00	0.00	98
accuracy			0.98	6562
macro avg	0.49	0.50	0.50	6562
weighted avg	0.97	0.98	0.98	6562

KNN

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	6464
1.0	0.14	0.01	0.02	98
accuracy			0.98	6562
macro avg	0.56	0.50	0.51	6562
weighted avg	0.97	0.98	0.98	6562

Random Forest

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	6464
1.0	0.10	0.01	0.02	98
accuracy			0.98	6562
macro avg	0.54	0.50	0.51	6562
weighted avg	0.97	0.98	0.98	6562

LightGBM

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	6464
1.0	0.00	0.00	0.00	98
accuracy			0.98	6562
macro avg	0.49	0.50	0.50	6562
weighted avg	0.97	0.98	0.98	6562

Data imbalance jelas terlihat

- Label 1 (minoritas) hanya ada 98, sedangkan label 0 ada 6.464.
- Model cenderung memprediksi semua sebagai 0 → recall & f1 untuk label 1 sangat rendah.
- Accuracy tinggi (0.98–0.99) tapi ini menyesatkan karena sebagian besar **prediksi benar hanya karena label 0 dominan.**

Precision, recall, f1-score untuk label 1 sangat rendah

- Logistic Regression: recall = 0 → tidak menangkap satu pun kasus positif.
- KNN: recall = 0.01 → hampir tidak menangkap kasus positif.
- Decision Tree: recall = 0.07 → sedikit lebih baik tapi masih sangat buruk.

MODELING FITTING WITH OVERFITTING

```python
# Model Fitting dengan Oversampling
models = [logreg, knn, dt, rf, xgb, lgbm]

rocauc_mean_sampling_select1 = []
rocauc_std_sampling_select1 = []

for i in models:
    pipe_model_sampling = ImbPipeline([
        ('scaler', RobustScaler()),
        ('smote', SMOTE(random_state=42)),
        ('algorithm', i)
    ])

    skfold = StratifiedKFold(n_splits=5)

    model_sampling_select = cross_val_score(
        pipe_model_sampling,
        x_train_selected,
        y_train,
        cv=skfold,
        scoring='roc_auc',
        n_jobs=-1
    )

    rocauc_mean_sampling_select1.append(model_sampling_select.mean())
    rocauc_std_sampling_select1.append(model_sampling_select.std())

pd.DataFrame({
    'model': ['Logistic Regression', 'KNN', 'Decision Tree', 'Random Forest', 'XGBoost', 'LightGBM'],
    'Mean': rocauc_mean_sampling_select1,
    'Std': rocauc_std_sampling_select1,
}).set_index('model').sort_values(by='Mean', ascending=False)
```

Overfitting: SMOTE

ROC-AUC MEAN

model	Mean	Std
Logistic Regression	0.8304726043198254	0.00990290389890932
LightGBM	0.7981816989712099	0.01964052606271477
XGBoost	0.7815848088202931	0.01664662839009377
Random Forest	0.7445377934254723	0.009925165675504678
KNN	0.6838301438205542	0.007691476355880423
Decision Tree	0.5384787444709555	0.015856086914307347

Logistic Regression

- Mean ROC-AUC: 0.830 → paling tinggi dari semua model.
- Std: 0.0099 → sangat stabil antar fold → model konsisten.

LightGBM

- Mean: 0.798 → sedikit di bawah LogReg.
- Std: 0.0196 → agak lebih variatif, artinya performa lebih sensitif terhadap data split.

XGBoost

- Mean: 0.781 → mirip LightGBM tapi sedikit lebih rendah.
- Std: 0.0166 → variasi sedang.

MODELING FITTING WITHOUT OVERFITTING

```
# Data Training with oversampling
models = [logreg, knn, dt, rf, xgb, lgbm]
rocauc_test_sampling_select1 = []
rocauc_train_sampling_select1 = []

def y_pred_func(model):
    estimator = ImbPipeline([
        ('scaler', scaler),
        ('resample', smote),
        ('algorithm', model)
    ])

    estimator.fit(x_train_selected, y_train)
    return estimator

for model in models:
    estimator = y_pred_func(model)
    y_pred_test = estimator.predict_proba(x_test_selected)[:, 1]
    y_pred_train = estimator.predict_proba(x_train_selected)[:, 1]
    rocauc_test_sampling_select1.append(roc_auc_score(y_test,y_pred_test))
    rocauc_train_sampling_select1.append(roc_auc_score(y_train,y_pred_train))

pd.DataFrame({
    'model':['Logistic Regression', 'KNN', 'Decision Tree', 'Random Forest', 'XGBoost', 'LightGBM'],
    'Train Set': rocauc_train_sampling_select1,
    'Validation Set': rocauc_mean_sampling_select1,
    'Test Set': rocauc_test_sampling_select1,
}).set_index('model').sort_values(by='Validation Set',ascending=False)
```

Test Set Validation

model	# Train Set	# Validation Set	# Test Set
Logistic Regression	0.8414108577835364	0.8304726043198254	0.7835634408466357
LightGBM	0.8908612735255241	0.7981816989712099	0.774349142503536
XGBoost	0.9475730033908687	0.7815848088202931	0.7759853632046878
Random Forest	0.9998630911292746	0.7445377934254723	0.6873563472418671
KNN	0.995837139380155	0.6838301438205542	0.6120412583350171
Decision Tree	0.9998860411711881	0.5384787444709555	0.5277691831683168

Logistic Regression:
Stabil dari train ke test (0.841 → 0.784). Performa menurun sedikit, tapi tetap cukup konsisten, model tidak overfit.

LightGBM:
Cukup baik di train (0.891) tapi **turun lebih nyata di validation dan test** (0.798 → 0.774). Masih relatif stabil dibanding model kompleks lainnya.

XGBoost:
Sangat tinggi di train (0.948) tapi turun signifikan di validation (0.782) dan test (0.776), **menandakan overfitting sedang.**

MODELING FITTING WITHOUT OVERFITTING

Final Test Validation

Logistic Regression

	precision	recall	f1-score	support
0.0	0.99	0.80	0.89	6464
1.0	0.05	0.67	0.09	98
accuracy			0.80	6562
macro avg	0.52	0.74	0.49	6562
weighted avg	0.98	0.80	0.88	6562

LightGBM

	precision	recall	f1-score	support
0.0	0.99	0.98	0.98	6464
1.0	0.09	0.14	0.11	98
accuracy			0.97	6562
macro avg	0.54	0.56	0.55	6562
weighted avg	0.97	0.97	0.97	6562

XGBoost

	precision	recall	f1-score	support
0.0	0.99	0.98	0.98	6464
1.0	0.08	0.11	0.09	98
accuracy			0.97	6562
macro avg	0.53	0.55	0.54	6562
weighted avg	0.97	0.97	0.97	6562

Recall kelas 1 (positif) naik drastis di Logistic Regression

- Sebelum SMOTE: 0.00 → Sesudah SMOTE: 0.67, artinya **model sekarang jauh lebih sensitif menangkap klaim positif.**

Trade-off

- Akurasi turun (LogReg dari ~0.99 → 0.80) karena model **mulai "berani" memprediksi positif.**
- Precision kelas 1 turun (0.05) → **banyak false positive** (klaim negatif ikut diprediksi positif).

BEST MODEL: LOGISTIC REGRESSION, SMOTE

HYPERPARAMETER TUNING

Hyperparameter tuning → model optimal, stabil, dan mampu generalisasi dengan baik, terutama pada dataset kompleks atau tidak seimbang.

Best Model Without Tuning

```
Pipeline(steps=[('scaler', RobustScaler()),
                ('resample', SMOTE(random_state=2023)),
                ('algorithm',
                 LogisticRegression(max_iter=4000, random_state=2023))])
              precision    recall  f1-score   support

         0.0       0.99      0.81      0.89      6464
         1.0       0.05      0.67      0.09        98

    accuracy                           0.81      6562
   macro avg       0.52      0.74      0.49      6562
weighted avg       0.98      0.81      0.88      6562
```

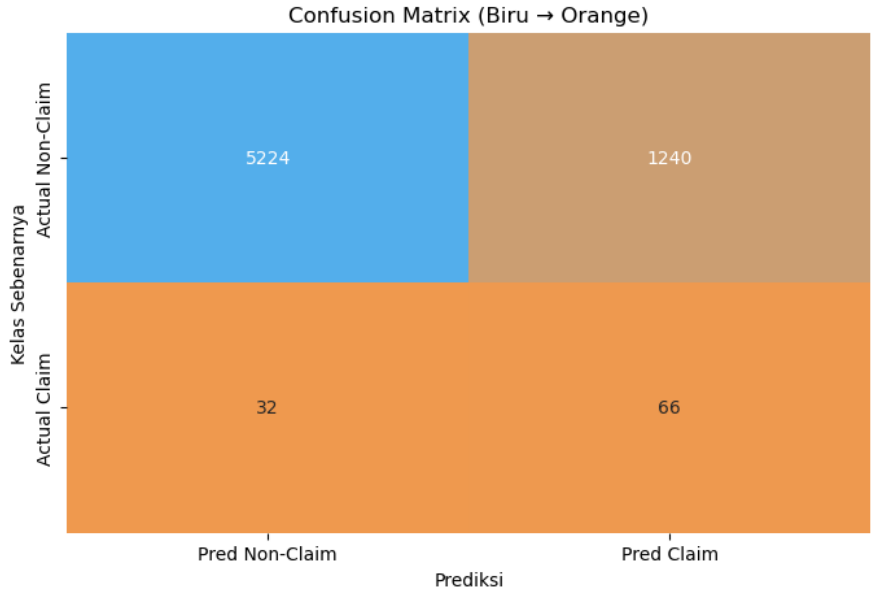
- **Kelas 0 (Non-claim)** → Precision **0.99**, Recall **0.81**, F1-score **0.89** → performa kuat di kelas mayoritas.
- **Kelas 1 (Claim)** → Precision **0.05**, Recall **0.67**, F1-score **0.09** → recall cukup tinggi, precision sangat rendah (banyak FP).
- **Accuracy** → **0.81**, namun **kurang representatif** karena data **imbalanced**.

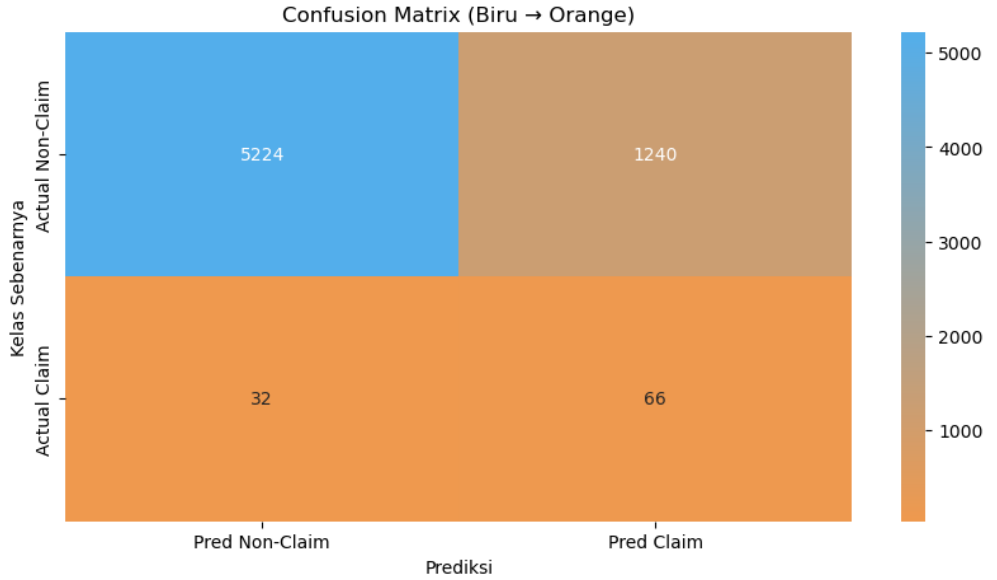
Best Model With Tuning

	precision	recall	f1-score	support
0.0	0.99	0.81	0.90	6464
1.0	0.05	0.66	0.10	98
accuracy			0.81	6562
macro avg	0.52	0.74	0.50	6562
weighted avg	0.98	0.81	0.88	6562

- **Kelas 0 (Non-claim)** → Precision **0.99**, Recall **0.81**, F1-score **0.90** → performa sangat baik.
- **Kelas 1 (Claim)** → Precision **0.05**, Recall **0.66**, F1-score **0.10** → recall cukup tinggi, precision sangat rendah (banyak FP).
- **Accuracy** → **0.81**, namun kurang representatif karena data **imbalanced**.

HYPERPARAMETER TUNING





Best Model Without Tuning

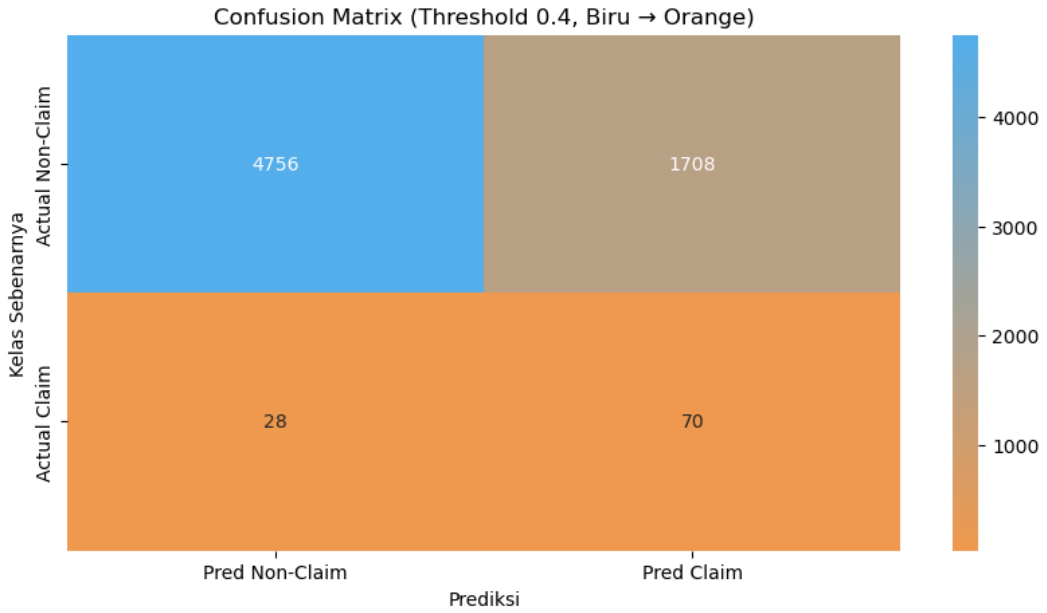
- Mayoritas prediksi benar pada **non-claim** (TN tinggi: 5224).
- **False Positive** cukup besar (1240) → banyak non-claim diprediksi jadi claim.
- **False Negative** sangat rendah (32) → klaim yang terlewat sedikit.
- **True Positive** masih rendah (66) → deteksi klaim benar masih terbatas.

Best Model With Tuning

- **True Negative (5224)** → Mayoritas non-claim diprediksi benar.
- **False Positive (1240)** → Masih banyak non-claim yang salah terdeteksi sebagai claim.
- **False Negative (32)** → Klaim yang terlewat sangat sedikit.
- **True Positive (66)** → Deteksi klaim benar masih rendah.

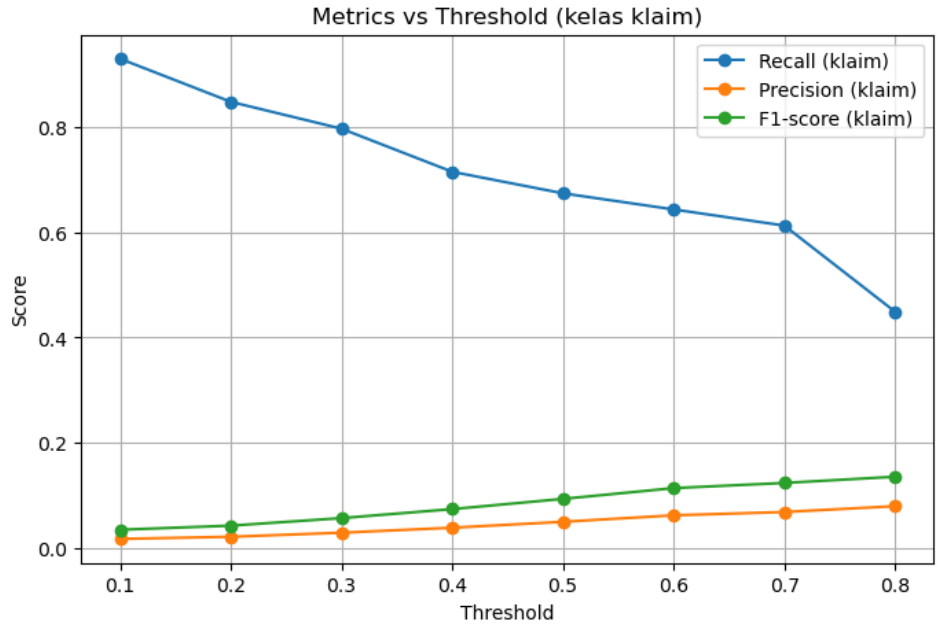
BEST MODEL: LOGISTIC REGRESSION, SMOTE, NO TUNING

THRESHOLD



Confusion Matrix (Threshold 0.4)

- **4756 (TN)** → Non-claim benar diprediksi sebagai non-claim.
- **1708 (FP)** → Non-claim salah diprediksi sebagai claim.
- **28 (FN)** → Claim salah diprediksi sebagai non-claim.
- **70 (TP)** → Claim benar diprediksi sebagai claim.

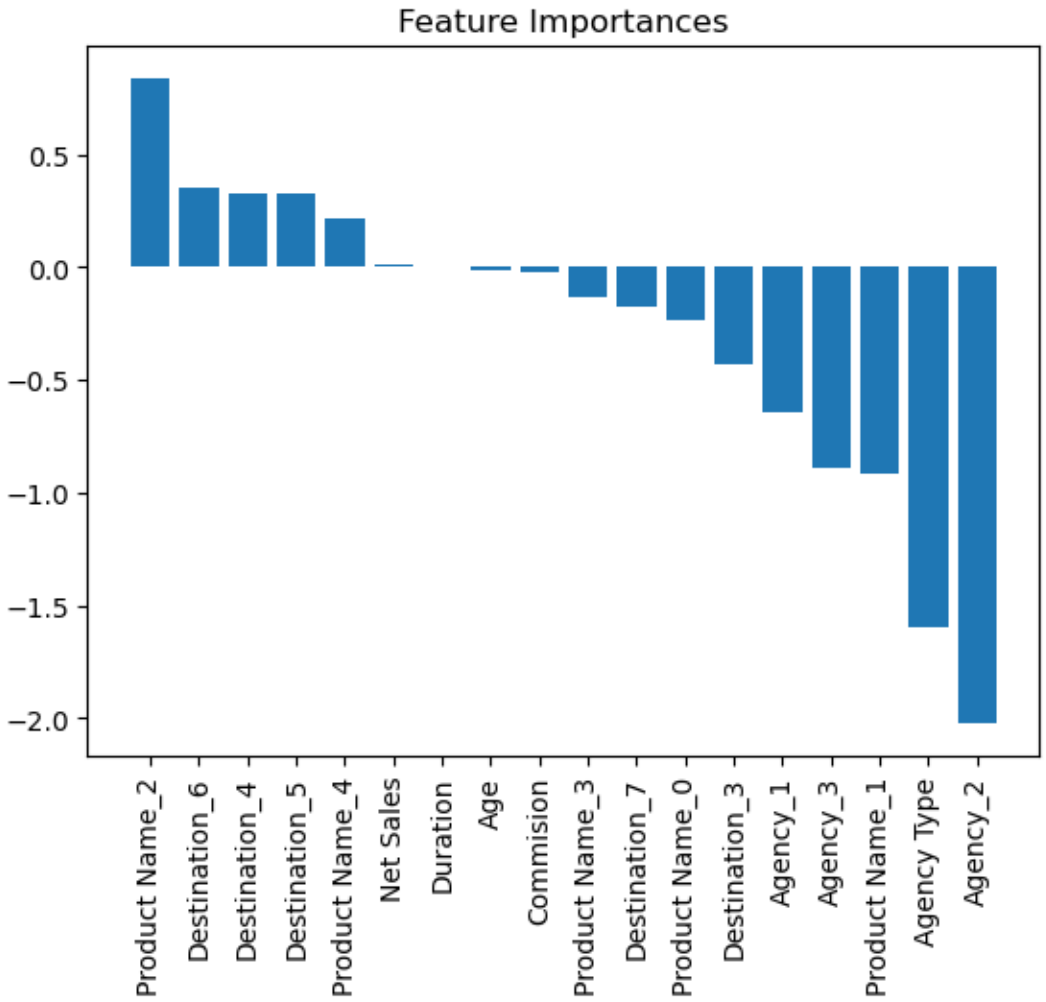


Threshold Analysis

- **0.3** → Recall klaim sangat tinggi (**0.80**), tapi banyak salah prediksi non-claim → recall non-claim **0.60**, accuracy **0.60**.
- **0.4** → Recall klaim masih tinggi (**0.71**), recall non-claim **0.74**, accuracy lebih seimbang (**0.74**).
- **0.5** → Recall klaim turun (**0.66**), recall non-claim naik (**0.81**), accuracy tertinggi (**0.81**).

BEST MODEL: LOGISTIC REGRESSION, SMOTE, NO TUNING, THRESHOLD 0.4

FEATURE IMPORTANCE



- **Product Name_2** → Kontributor terbesar meningkatkan prediksi klaim.
- **Agency_2** & **Product Name_7** → Nilai negatif signifikan, menurunkan prediksi klaim.
- **Nilai positif** → Fitur meningkatkan kemungkinan klaim.
- **Nilai negatif** → Fitur menurunkan kemungkinan klaim.
- **Nilai mendekati nol** → Pengaruh minimal pada prediksi.

CONCLUSION & RECOMMENDATION

CONCLUSION

	precision	recall	f1-score	support
Non-Claim	0.99	0.74	0.85	6464
Claim	0.04	0.71	0.08	98
accuracy			0.74	6562
macro avg	0.52	0.73	0.46	6562
weighted avg	0.98	0.74	0.84	6562

- **Akurasi**: 74% (keseluruhan prediksi benar).
- **Precision klaim**: 4% → banyak False Positive (96% prediksi klaim salah).
- **Recall klaim**: 71% → mayoritas klaim nyata terdeteksi, hanya 29% terlewat.
- Model **sensitif** dalam menangkap klaim (recall tinggi) tapi **tidak presisi**, berpotensi memproses klaim tidak valid.
- Untuk bisnis travel insurance, **menghindari klaim terlewat** lebih penting dibanding menghindari klaim tidak valid.

RECOMMENDATION

- Lengkapi data penting (mis. **Gender**) untuk kurangi klaim tidak valid.
- Tambah fitur terkait klaim (**tanggal pengajuan**, **harga polis**, **syarat & ketentuan**).
- Optimalkan fitur kuantitatif (**durasi perjalanan**, **umur**, **komisi**) agar tidak membebani model.
- Lanjutkan eksperimen algoritma, hyperparameter, metrik, & teknik resampling.

TERIMA KASIH

Github: https://github.com/SheylaAnn/ML_Travel_Insurance