Jagdsh Chand & Nisarg Shah – Model Selection – Assignment

Problem Statement

Global Insure, a leading insurance company, processes thousands of claims annually. However, a significant percentage of these claims turn out to be fraudulent, resulting in considerable financial losses. The company's current process for identifying fraudulent claims involves manual inspections, which is time-consuming and inefficient. Fraudulent claims are often detected too late in the process, after the company has already paid out significant amounts. Global Insure wants to improve its fraud detection process using data-driven insights to classify claims as fraudulent or legitimate early in the approval process. This would minimise financial losses and optimise the overall claims handling process.

Business Objective

Global Insure wants to build a model to classify insurance claims as either fraudulent or legitimate based on historical claim details and customer profiles. By using features like claim amounts, customer profiles and claim types, the company aims to predict which claims are likely to be fraudulent before they are approved.

Based on this assignment, you have to answer the following questions:

- How can we analyse historical claim data to detect patterns that indicate fraudulent claims?
- Which features are most predictive of fraudulent behaviour?
- Can we predict the likelihood of fraud for an incoming claim, based on past data?
- What insights can be drawn from the model that can help in improving the fraud detection process?

Assignment Steps

1) Data Preparation:

   - Loading the data set "insurance_claims.csv"
   - Checking the few data point

months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium
328	48	521585	2014-10-17	OH	250/500	1000	1406.91
228	42	342868	2006-06-27	IN	250/500	2000	1197.22
134	29	687698	2000-09-06	OH	100/300	2000	1413.14
256	41	227811	1990-05-25	IL	250/500	2000	1415.74
228	44	367455	2014-06-06	IL	500/1000	1000	1583.91

   - Getting the shape of the data set : (1000, 40)
   - Checking the count,mean,media,25%,50%,75%,max statistics of all columns

1.1) Data Cleaning:

- Data cleaning consists of the below steps :
- Handling the Null values such as “authorities_contacted”
- Null values were replaced by most frequent mode
- Identified and handled redundant values and columns
- Identify and drop any columns that are completely empty
- Columns such as “_c39” were empty so dropped them
- Identified the rows such as 'policy_deductable', 'umbrella_limit', 'incident_hour_of_the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'auto_year'
- The above rows had illogical, invalid and negative values.
- Removed the rows of columns 'police_report_available', 'collision_type', 'property_damage' as they had values ‘?’

2) Data Type Fixing:

- The data types of all the columns are shown below

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 0 to 998
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           800 non-null    int64
 1   age                          800 non-null    int64
 2   policy_bind_date             800 non-null    object
 3   policy_state                 800 non-null    object
 4   policy_csl                   800 non-null    object
 5   policy_deductable            800 non-null    int64
 6   policy_annual_premium        800 non-null    float64
 7   umbrella_limit               800 non-null    int64
 8   insured_sex                  800 non-null    object
 9   insured_education_level      800 non-null    object
 10  insured_occupation           800 non-null    object
 11  insured_hobbies              800 non-null    object
 12  insured_relationship         800 non-null    object
 13  capital-gains                800 non-null    int64
 14  capital-loss                 800 non-null    int64
 15  incident_date                800 non-null    object
 16  incident_type                800 non-null    object
 17  collision_type               800 non-null    object
 18  incident_severity            800 non-null    object
 19  authorities_contacted        800 non-null    object
 20  incident_state               800 non-null    object
 21  incident_city                800 non-null    object
 22  incident_hour_of_the_day     800 non-null    int64
 23  number_of_vehicles_involved  800 non-null    int64
 24  property_damage              800 non-null    object
 25  bodily_injuries              800 non-null    int64
 26  witnesses                    800 non-null    int64
 27  police_report_available      800 non-null    object
 28  total_claim_amount           800 non-null    int64
 29  injury_claim                 800 non-null    int64
 30  property_claim               800 non-null    int64
 31  vehicle_claim                800 non-null    int64
 32  auto_make                    800 non-null    object
 33  auto_model                   800 non-null    object
 34  auto_year                    800 non-null    int64
 35  fraud_reported               800 non-null    object
dtypes: float64(1), int64(15), object(20)
memory usage: 231.2+ KB
```

- We changed the data type of “policy_bind_date” and “incident_date” to date_time.”
- Rest of the columns were having perfect data type.

3) Train – Test Split :
   - First we detfine 'fraud_reported'
   - We split the data set into 70:30 ratio
   - X_train and Y_train for training data
   - X_validation and Y_validation for testing data

```
# Split the dataset into 70% train and 30% validation and use stratification on the target variable
X_train, X_validation, y_train, y_validation = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
# Reset index for all train and test sets
X_train = X_train.reset_index(drop=True)
X_validation = X_validation.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_validation = y_validation.reset_index(drop=True)
```

```
print(X_train.shape)
print(X_validation.shape)
print(y_train.shape)
print(y_validation.shape)
```

```
(560, 35)
(240, 35)
(560,)
(240,)
```

4) EDA on training data :
   - Performing univariate for the numerical variables
   - The below numerical variables were identified:

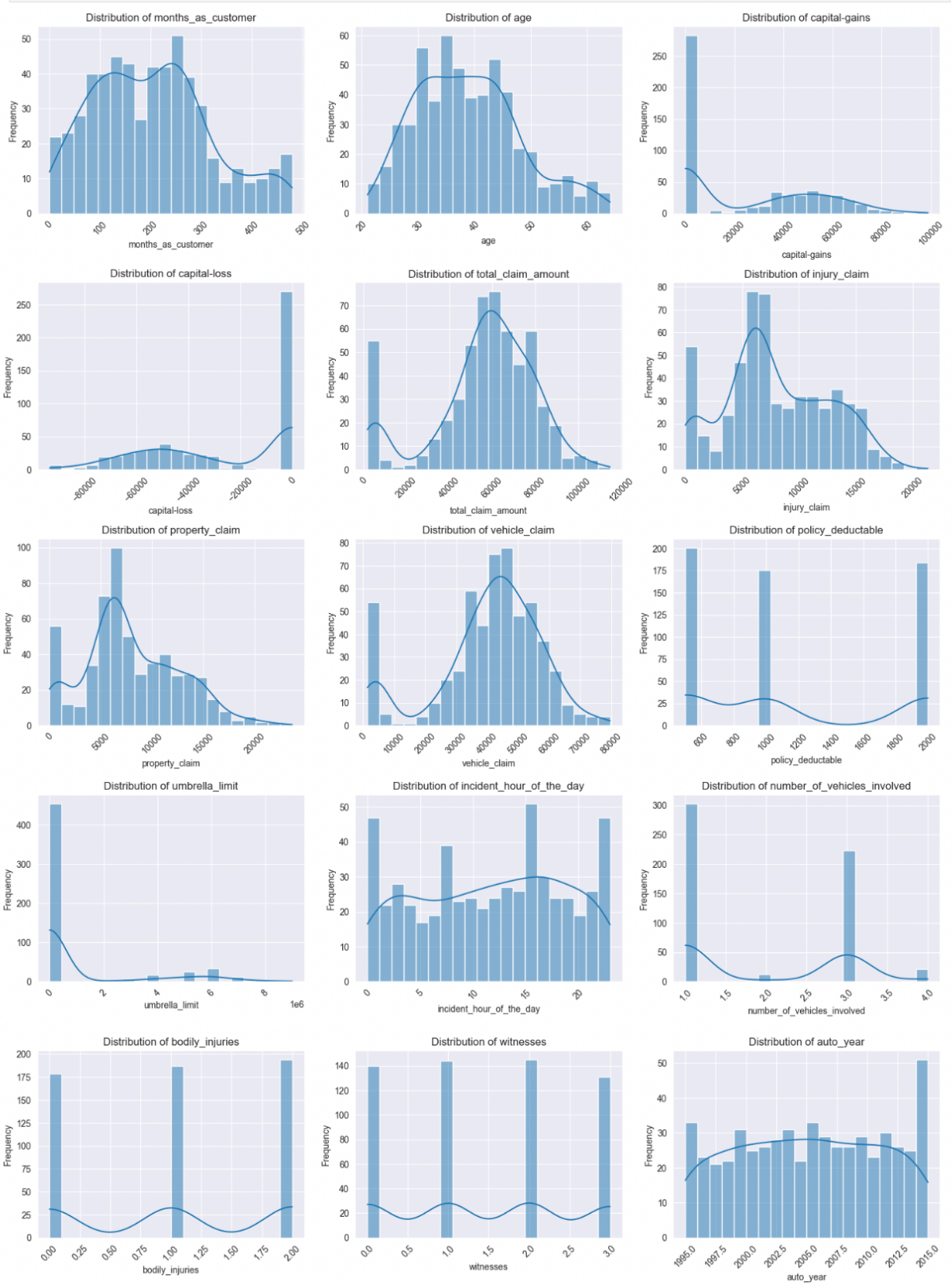
   - g_categorical_columns = ['policy_state', 'policy_csl', 'insured_sex', 'insured_education_level', 'insured_occupation', 'insured_hobbies', 'insured_relationship', 'incident_type', 'collision_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'property_damage', 'police_report_available', 'auto_make', 'auto_model']

   - g_unique_columns = ['policy_number', 'incident_location']

   - g_continuous_numerical = ['months_as_customer', 'age', 'capital-gains', 'capital-loss', 'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_claim']

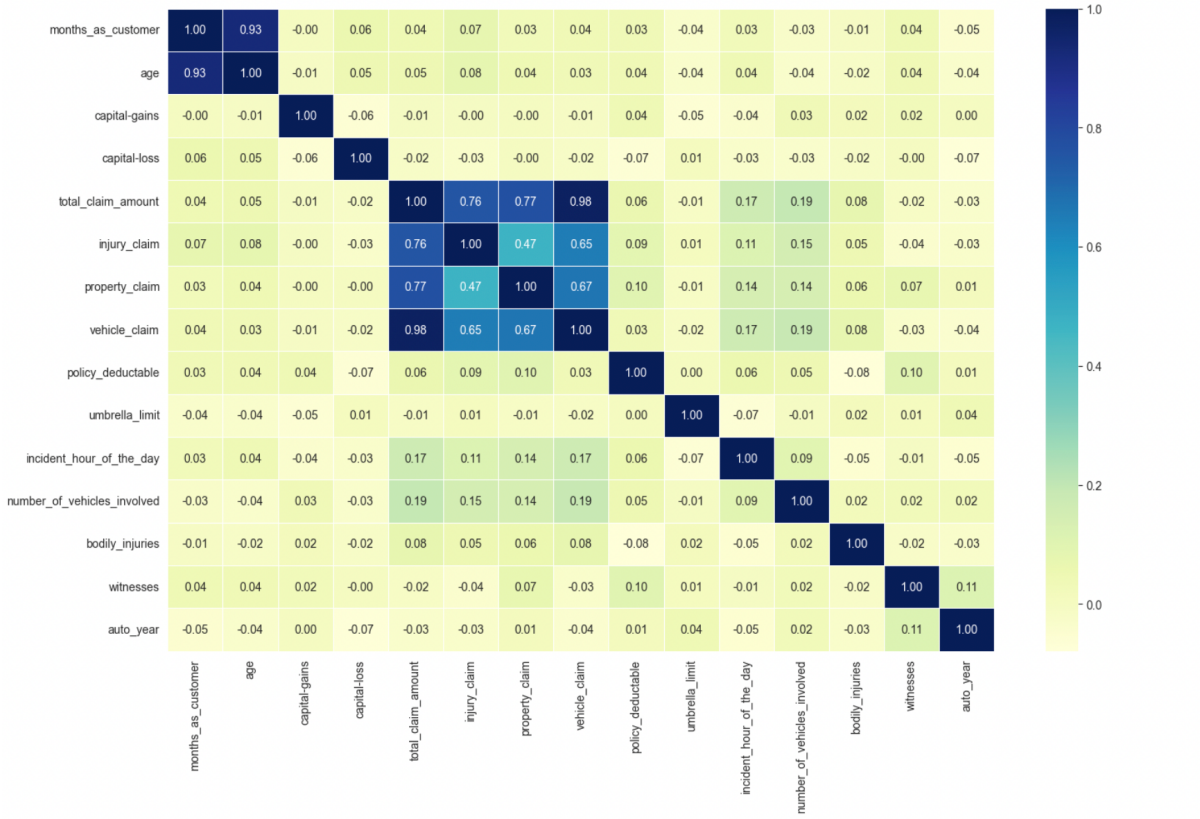
   - g_date_columns = ['policy_bind_date', 'incident_date']

   - g_numerical_columns = g_continuous_numerical + g_ordinal_numerical



4.1) Perform Correlation analysis :

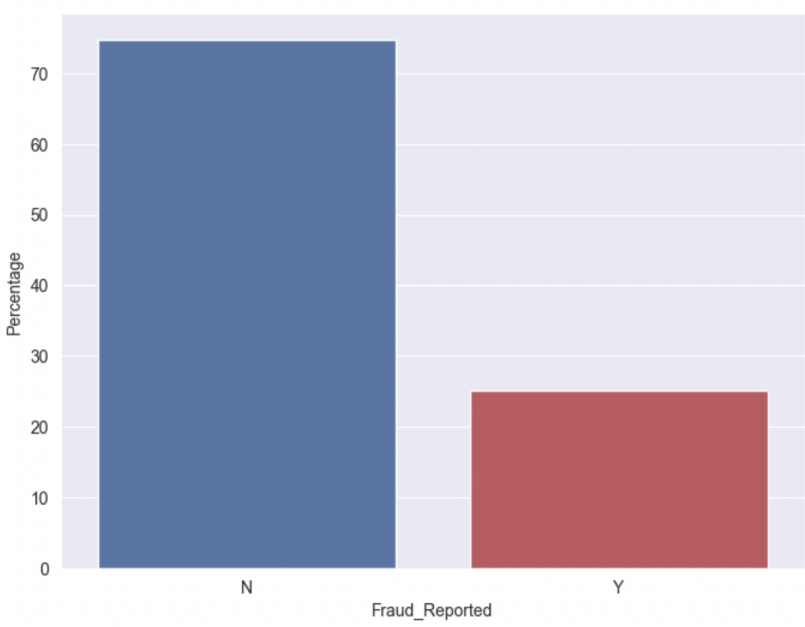
Investigated the relationships between numerical features to identify potential multicollinearity or dependencies. Visualised the correlation structure using an appropriate method to gain insights into feature relationships



The above figure shows the correlation between the columns.

4.2) Checking Class Balance:

- We checked the Fraud bar chart i.e Fraud committed ‘Y’ vs Fraud not committed ‘N‘.
- It can be seen that fraud committed is around 25%.

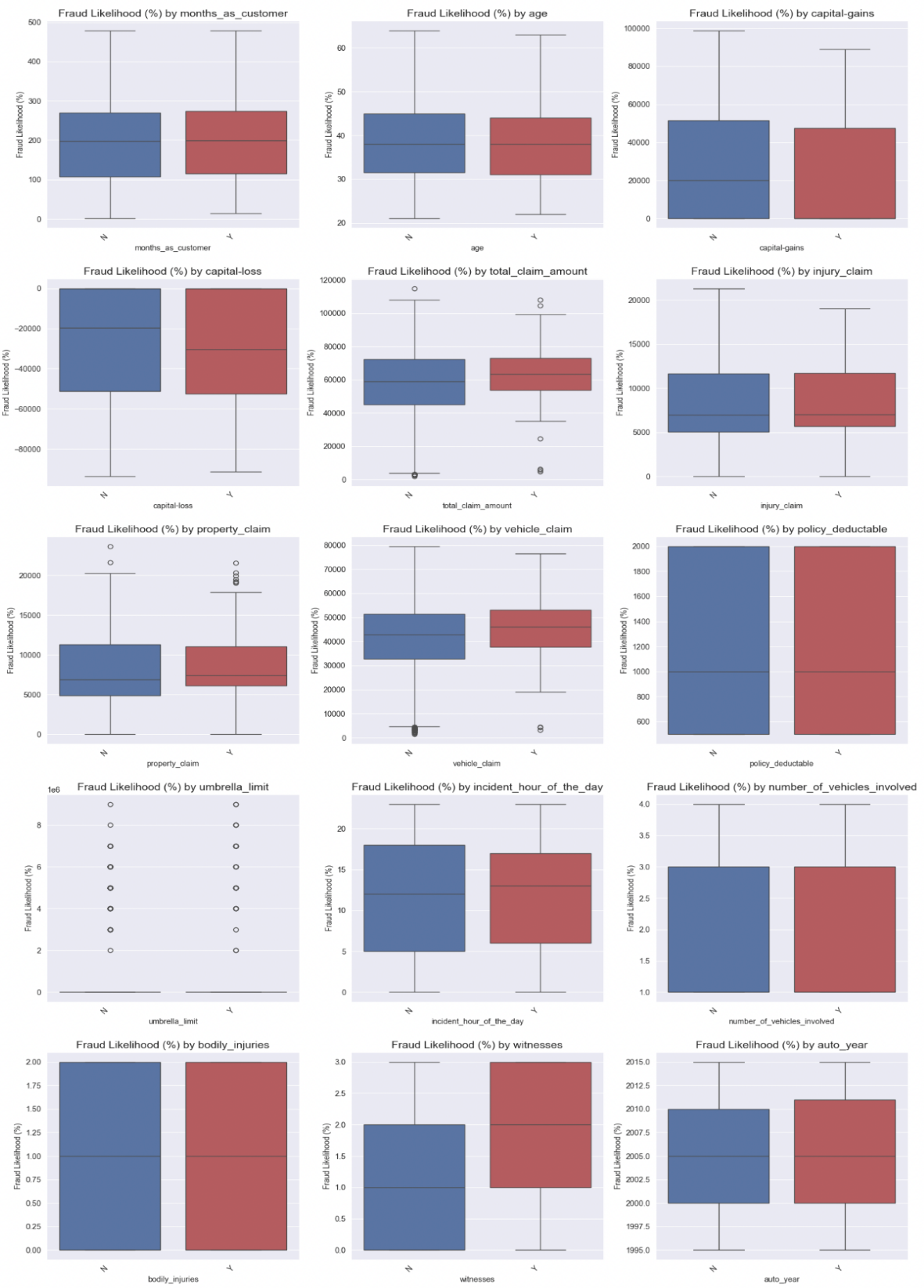


4.3) Performing Bi-variate analysis:

- Investigated the relationships between categorical features and the target variable by analysing the target event likelihood (for the 'Y' event) for each level of every relevant categorical feature

4.5) Target variable vs Numerical variables

- We found out the relationship between the target variable and numerical variable.



5) Feature Engineering:

- Used the RandomOverSampler technique to balance the data and handle class imbalance. This method increases the number of samples in the minority class by randomly duplicating them, creating synthetic data points with similar characteristics. This helps prevent the model from being biased toward the majority class and improves its ability to predict the minority class more accurately.

```python
# Import RandomOverSampler from imblearn library
from imblearn.over_sampling import RandomOverSampler

# Perform resampling on training data
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
X_resampled_df = pd.DataFrame(X_resampled, columns=X_train.columns)
y_resampled_df = pd.Series(y_resampled, name=y_train.name)
print(len(y_train), "->", len(y_resampled))
print("\nNew Class Distribution:")
print(y_resampled.value_counts())
print(y_resampled_df)
print(X_resampled_df)
# del X_train
# del y_train
# del X_resampled
# del y_resampled
```

```
560 -> 838
```

6) Feature Creation

- Created new features based on your understanding for both training and validation data
- “ Policy Tenure” : (months) [policy_bind-date] - [incident_date]
- “Incident Day”: Type (Weekend or weekday)
- “Claim consistency ratio” (total_claim_amount/policy_annual_premium > 80)
- “Injury vs Vehicle”: ratio (injury_claim/vehicle_claim)
- “ Capital activity” : (1 if Total capital returns ('capital-loss' + 'capital-gains') < 0 else 0)
- “Claim component balance” : (1 if (vehicle_claim/tail_claim_amount) mean > 70%)

```
X_resampled_df['Policy_Tenure'] = (X_resampled_df['incident_date'] - X_resampled_df['policy_bind_date']).dt.days
X_resampled_df['Incident_Day_Type'] = (X_resampled_df['incident_date'].dt.dayofweek >= 5).astype(int) ## 0 Weekda
X_resampled_df['Incident_Month'] = X_resampled_df['incident_date'].dt.month
X_resampled_df['Claim_consistency_ratio'] = (X_resampled_df['total_claim_amount'] / X_resampled_df['policy_annual
X_resampled_df['Injury_vs_Vehicle_ratio'] = X_resampled_df['injury_claim'] / X_resampled_df['vehicle_claim']
X_resampled_df['Capital_activity'] = (X_resampled_df['capital-loss'] + X_resampled_df['capital-gains']).astype(in
X_resampled_df['Claim_component_balance'] = (X_resampled_df['vehicle_claim'] / X_resampled_df['total_claim_amount

derived_variables = ['Policy_Tenure', 'Incident_Month', 'Claim_consistency_ratio', 'Injury_vs_Vehicle_ratio', 'Cl
g_numerical_columns = g_numerical_columns + derived_variables
del derived_variables
g_categorical_columns.append('Incident_Day_Type')
X_resampled_df
```

6.1) Handle redundant columns:

- It was found that columns [policy_bind_date', 'policy_deductable', 'policy_annual_premium', 'capital-gains', capital-loss', 'incident_date', 'vehicle_claim', 'auto_year', 'number_of_vehicles_involved', 'age'].

- Additionally we converted target variable “Fraud_Reported” from ‘Y’ and ‘N’ to ‘1’ and ‘0’ respectively.

```
# Create correlation matrix numerical columns against the target variable
train_numerical_corr_df = X_resampled_df[g_numerical_columns].copy()
train_numerical_corr_df['Fraud_Reported'] = y_resampled_df
train_numerical_corr_df['Fraud_Reported'] = train_numerical_corr_df['Fraud_Reported'].replace({'Y': 1, 'N': 0})
plt.figure(figsize=(16, 10))
train_numerical_corr_df.corrwith(train_numerical_corr_df['Fraud_Reported']).abs().sort_values(ascending=False)
corr = train_numerical_corr_df.corr()
# Plot Heatmap of the correlation matrix
sns.heatmap(corr,
            annot=True,
            cmap="YlGnBu",
            fmt=".2f",
            linewidths=1,
            cbar=True
            )
plt.tight_layout()
```

```
# Drop redundant columns from training and validation data
redundant_numerical_columns = ['policy_bind_date', 'policy_deductable', 'policy_annual_premium', 'capital-gains',
        'capital-loss', 'incident_date', 'vehicle_claim', 'auto_year', 'number_of_vehicles_involved', 'age']
X_resampled_df = X_resampled_df.drop(redundant_numerical_columns, axis=1)
X_validation = X_validation.drop(redundant_numerical_columns, axis=1)
g_numerical_columns = [col for col in g_numerical_columns if col not in redundant_numerical_columns]
```

6.2) Combine values in categorical columns:

- Combine categories that have low frequency or provide limited predictive information

- 1 if (X_resampled_df['incident_type'] == 'Multi-vehicle Collision') & ((X_resampled_df['incident_severity'] == 'Total Loss') | (X_resampled_df['incident_severity'] == 'Major Damage')) == valid_multi_collision_damage

- 1 if (X_resampled_df['incident_type'] == 'Single Vehicle Collision') & ((X_resampled_df['incident_severity'] == 'Total Loss') | (X_resampled_df['incident_severity'] == 'Major Damage')) else 0 == valid_single_collision_damage

- 1 if (X_resampled_df['authorities_contacted'] =='Police') & (X_resampled_df['police_report_available'] == 'NO') == Valid_police_report

- 1 if ((X_resampled_df['authorities_contacted'] =='Ambulance') & (X_resampled_df['bodily_injuries'] == 0) & (X_resampled_df['injury_claim'] > 0)) == Valid_injury_claim

```
X_resampled_df['valid_multi_collision_damage'] = (
    (X_resampled_df['incident_type'] == 'Multi-vehicle Collision') &
    ((X_resampled_df['incident_severity'] == 'Total Loss') | (X_resampled_df['incident_severity'] == 'Major Damag
X_resampled_df['valid_single_collision_damage'] = (
    (X_resampled_df['incident_type'] == 'Single Vehicle Collision') &
    ((X_resampled_df['incident_severity'] == 'Total Loss') | (X_resampled_df['incident_severity'] == 'Major Damag
X_resampled_df['Valid_police_report'] = (
    (X_resampled_df['authorities_contacted'] == 'Police') &
    (X_resampled_df['police_report_available'] == 'NO')).astype(int)
X_resampled_df['Valid_injury_claim'] = (
    (X_resampled_df['authorities_contacted'] == 'Ambulance') &
    (X_resampled_df['bodily_injuries'] == 0) & (X_resampled_df['injury_claim'] > 0)
).astype(int)

X_validation['valid_multi_collision_damage'] = (
    (X_validation['incident_type'] == 'Multi-vehicle Collision') &
    ((X_validation['incident_severity'] == 'Total Loss') | (X_validation['incident_severity'] == 'Major Damage'))
X_validation['valid_single_collision_damage'] = (
    (X_validation['incident_type'] == 'Single Vehicle Collision') &
    ((X_validation['incident_severity'] == 'Total Loss') | (X_validation['incident_severity'] == 'Major Damage'))
X_validation['Valid_police_report'] = (
    (X_validation['authorities_contacted'] == 'Police') &
    (X_validation['police_report_available'] == 'NO')).astype(int)
X_validation['Valid_injury_claim'] = (
    (X_validation['authorities_contacted'] == 'Ambulance') &
    (X_validation['bodily_injuries'] == 0) & (X_validation['injury_claim'] > 0)
).astype(int)

redundant_categorical_columns = ['authorities_contacted', 'bodily_injuries', 'injury_claim', 'incident_severity',
X_resampled_df = X_resampled_df.drop(redundant_categorical_columns, axis=1)
X_validation = X_validation.drop(redundant_categorical_columns, axis=1)
g_categorical_columns = [col for col in g_categorical_columns if col not in redundant_categorical_columns]
g_numerical_columns = [col for col in g_numerical_columns if col not in redundant_categorical_columns]
X_resampled_df
```

	months_as_customer	policy_state	policy_csl	umbrella_limit	insured_sex	insured_education_level	insured_occupation	ins
0	169	IL	100/300	0	MALE	PhD	craft-repair	
1	428	IL	100/300	0	FEMALE	High School	tech-support	
2	154	OH	100/300	0	FEMALE	JD	priv-house-serv	
3	252	IN	500/1000	0	FEMALE	JD	handlers-cleaners	
4	225	OH	500/1000	0	MALE	High School	exec-managerial	
...	...	...	...	...	...	...	...	
833	121	IN	500/1000	0	MALE	MD	priv-house-serv	
834	61	IL	250/500	4000000	FEMALE	Associate	tech-support	
835	154	IN	100/300	0	MALE	PhD	machine-op-inspct	
836	231	OH	250/500	0	FEMALE	High School	sales	
837	82	IL	100/300	0	MALE	PhD	adm-clerical	

838 rows × 30 columns

6.3) Combine values in categorical columns:

- Identified categorical columns for dummy variable creation

```
# Identify the categorical columns for creating dummy variables
g_categorical_columns
```

```
['policy_state',
 'policy_csl',
 'insured_sex',
 'insured_education_level',
 'insured_occupation',
 'insured_relationship',
 'collision_type',
 'incident_state',
 'incident_city',
 'property_damage',
 'police_report_available',
 'auto_make',
 'auto_model',
 'Incident_Day_Type']
```

6.4) Feature scaling

- Scaling numerical features to a common range to prevent features with larger values from dominating the model.

```
# Import the necessary scaling tool from scikit-learn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Scale the numeric features present in the training data
X_resampled_dummies_df[g_numerical_columns] = scaler.fit_transform(X_resampled_dummies_df[g_numerical_columns])
print(X_resampled_dummies_df)

# Scale the numeric features present in the validation data

X_dummies_validation_df[g_numerical_columns] = scaler.transform(X_dummies_validation_df[g_numerical_columns])
```

7) Model Building

Logistic Regression Model

- Feature Selection using RFECV – Identify the most relevant features using Recursive Feature Elimination with Cross-Validation.
- Model Building and Multicollinearity Assessment – Build the logistic regression model and analyse statistical aspects such as p-values and VIFs to detect multicollinearity.
- Model Training and Evaluation on Training Data – Fit the model on the training data and assess initial performance.
- Finding the Optimal Cutoff – Determine the best probability threshold by analysing the sensitivity-specificity tradeoff and precision-recall tradeoff.
- FInal Prediction and Evaluation on Training Data using the Optimal Cutoff – Generate final predictions using the selected cutoff and evaluate model performance.

Random Forest Model

- Get Feature Importances - Obtain the importance scores for each feature and select the important features to train the model.
- Model Evaluation on Training Data – Assess performance metrics on the training data.
- Check Model Overfitting using Cross-Validation – Evaluate generalisation by performing cross-validation.
- Hyperparameter Tuning using Grid Search – Optimise model performance by fine-tuning hyperparameters.
- Final Model and Evaluation on Training Data – Train the final model using the best parameters and assess its performance.

7.1.1 Import necessary libraries

```
# Import necessary libraries
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
```

7.1.2 Perform feature selection

Applied RFECV to identify the most relevant features.

```
# Apply RFECV to identify the most relevant features
from sklearn.ensemble import RandomForestClassifier

estimator = RandomForestClassifier(
    n_estimators=100,
    max_depth=5,
    random_state=42,
    n_jobs=-1
)

rfecv = RFECV(
    estimator=estimator,
    step=10,               # Changed step size for faster elimination
    cv=StratifiedKFold(5),
    scoring='f1',
    min_features_to_select=5,
    n_jobs=-1
)
rfecv.fit(X_resampled_dummies_df, y_resampled_dummies_df)

print(f"Optimal number of features: {rfecv.n_features_}")
print("-" * 50)
```

```
Optimal number of features: 95
--------------------------------------------------
```

```
# Display the features ranking by RFECV in a DataFrame
feature_ranking_data = list(zip(X_resampled_dummies_df.columns, rfecv.support_, rfecv.ranking_))
feature_ranking_df = pd.DataFrame(
    feature_ranking_data,
    columns=['Feature', 'Selected (Boolean)', 'RFECV Rank']
)
feature_ranking_df = feature_ranking_df.sort_values(by='RFECV Rank', ascending=True).reset_index(drop=True)
print(feature_ranking_df)
# Get the names of the selected features
optimal_features = X_resampled_dummies_df.columns[ rfecv.support_].tolist()
optimal_features = X_resampled_dummies_df.columns[ rfecv.support_]
print("Selected Optimal Features:")
print(optimal_features)
```

```
                        Feature  Selected (Boolean)  RFECV Rank
0             months_as_customer                True           1
1                auto_make_Audi                True           1
2    police_report_available_YES                True           1
3     police_report_available_NO                True           1
4           property_damage_YES                True           1
..                          ...                 ...         ...
120              auto_model_CRV               False           4
121           incident_state_PA               False           4
122            auto_model_Civic               False           4
123   auto_model_Grand Cherokee               False           4
124          auto_model_Impreza               False           4

[125 rows x 3 columns]
Selected Optimal Features:
Index(['months_as_customer', 'umbrella_limit', 'incident_hour_of_the_day',
       'witnesses', 'total_claim_amount', 'property_claim', 'Policy_Tenure',
       'Incident_Month', 'Claim_consistency_ratio', 'Injury_vs_Vehicle_ratio',
       'Capital_activity', 'Claim_component_balance',
       'valid_multi_collision_damage', 'valid_single_collision_damage',
       'Valid_police_report', 'Valid_injury_claim', 'policy_state_IL',
       'policy_state_IN', 'policy_state_OH', 'policy_csl_100/300',
       'policy_csl_250/500', 'policy_csl_500/1000', 'insured_sex_FEMALE',
       'insured_sex_MALE', 'insured_education_level_Associate',
       'insured_education_level_College',
       'insured_education_level_High School', 'insured_education_level_JD',
       'insured_education_level_MD', 'insured_education_level_Masters',
       'insured_education_level_PhD', 'insured_occupation_adm-clerical',
       'insured_occupation_armed-forces', 'insured_occupation_craft-repair',
       'insured_occupation_exec-managerial',
       'insured_occupation_farming-fishing',
       'insured_occupation_handlers-cleaners',
       'insured_occupation_machine-op-inspct',
       'insured_occupation_other-service',
       'insured_occupation_priv-house-serv',
       'insured_occupation_prof-specialty',
       'insured_occupation_protective-serv', 'insured_occupation_sales',
       'insured_occupation_tech-support',
       'insured_occupation_transport-moving', 'insured_relationship_husband',
       'insured_relationship_other-relative', 'insured_relationship_own-child',
       'insured_relationship_unmarried', 'insured_relationship_wife',
       'collision_type_Front Collision', 'collision_type_Rear Collision',
       'incident_state_NC', 'incident_state_NY', 'incident_state_OH',
       'incident_state_SC', 'incident_state_VA', 'incident_state_WV',
       'incident_city_Arlington', 'incident_city_Columbus',
       'incident_city_Hillsdale', 'incident_city_Northbend',
       'incident_city_Northbrook', 'incident_city_Riverwood',
       'incident_city_Springfield', 'property_damage_NO',
       'property_damage_YES', 'police_report_available_NO',
       'police_report_available_YES', 'auto_make_Audi', 'auto_make_BMW',
       'auto_make_Chevrolet', 'auto_make_Dodge', 'auto_make_Ford',
       'auto_make_Jeep', 'auto_make_Mercedes', 'auto_make_Saab',
       'auto_make_Suburu', 'auto_make_Toyota', 'auto_make_Volkswagen',
       'auto_model_95', 'auto_model_A3', 'auto_model_A5', 'auto_model_C300',
       'auto_model_Camry', 'auto_model_Corolla', 'auto_model_F150',
       'auto_model_Jetta', 'auto_model_MDX', 'auto_model_Neon',
       'auto_model_Passat', 'auto_model_Tahoe', 'auto_model_Ultima',
       'auto_model_Wrangler', 'auto_model_X6'],
      dtype='object')
```

7.2 Build Logistic Regression Model

After selecting the optimal features using RFECV, utilising these features to build a logistic regression model with Statsmodels.

7.2.1 Select relevant features and add constant in training data

```
# Select only the columns selected by RFECV
col_selected_rfe = X_resampled_dummies_df[col]
col_selected_rfe
```

	months_as_customer	umbrella_limit	incident_hour_of_the_day	witnesses	total_claim_amount	property_claim	Policy_Tenu
0	-0.316140	-0.482488	0.033636	1.415167	0.983559	1.160293	-0.4450
1	2.003286	-0.482488	-0.416162	0.474715	-2.398834	-1.503368	-1.6892
2	-0.450469	-0.482488	0.933233	1.415167	0.762285	1.236270	0.2616
3	0.427151	-0.482488	1.532965	0.474715	1.472986	1.549116	0.6451
4	0.185358	-0.482488	1.233099	0.474715	0.576640	1.075378	1.0871
...	...	...	...	...	...	...	
833	-0.745995	-0.482488	-0.116296	-0.465737	0.431312	-0.281033	1.4210
834	-1.283313	1.271632	0.333502	1.415167	1.886468	2.210562	1.1143
835	-0.450469	-0.482488	0.483435	-0.465737	0.533511	-1.784929	1.5500
836	0.239090	-0.482488	0.333502	-1.406189	0.613676	-0.017349	1.3177
837	-1.095252	-0.482488	0.333502	1.415167	0.636647	0.217286	0.6779

838 rows × 95 columns

```
# Import statsmodels and add a constant
import statsmodels.api as sm
X_train_sm = sm.add_constant(col_selected_rfe)
# Check the data
X_train_sm
```

	const	months_as_customer	umbrella_limit	incident_hour_of_the_day	witnesses	total_claim_amount	property_claim	Poli
0	1.0	-0.316140	-0.482488	0.033636	1.415167	0.983559	1.160293	
1	1.0	2.003286	-0.482488	-0.416162	0.474715	-2.398834	-1.503368	
2	1.0	-0.450469	-0.482488	0.933233	1.415167	0.762285	1.236270	
3	1.0	0.427151	-0.482488	1.532965	0.474715	1.472986	1.549116	
4	1.0	0.185358	-0.482488	1.233099	0.474715	0.576640	1.075378	
...	...	...	...	...	...	...	...	
833	1.0	-0.745995	-0.482488	-0.116296	-0.465737	0.431312	-0.281033	
834	1.0	-1.283313	1.271632	0.333502	1.415167	1.886468	2.210562	
835	1.0	-0.450469	-0.482488	0.483435	-0.465737	0.533511	-1.784929	
836	1.0	0.239090	-0.482488	0.333502	-1.406189	0.613676	-0.017349	
837	1.0	-1.095252	-0.482488	0.333502	1.415167	0.636647	0.217286	

838 rows × 96 columns

7.2.2 Fit logistic regression model

Fitting a logistic Regression model on X_train after adding a constant and output the summary.

```
# Fit a logistic Regression model on X_train after adding a constant and output the summary
logsk = LogisticRegression(C=1e9)
#logsk.fit(X_train[col], y_train)
logsk.fit(X_train_sm, y_resampled_dummies_df)
```

```
LogisticRegression(C=1000000000.0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
logm4 = sm.GLM(y_resampled_dummies_df,(sm.add_constant(col_selected_rfe)), family = sm.families.Binomial())
modres = logm4.fit()
logm4.fit().summary()
```

7.2.3 Evaluate VIF of features to assess multicollinearity

```
# Import 'variance_inflation_factor'
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# Make a VIF DataFrame for all the variables present
vif = pd.DataFrame()
vif['Features'] = X_resampled_dummies_df[col].columns
vif['VIF'] = [variance_inflation_factor(X_resampled_dummies_df[col].values, i) for i in range(X_resampled_dummies
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
41	insured_occupation_protective-serv	inf
58	incident_city_Arlington	inf
32	insured_occupation_armed-forces	inf
33	insured_occupation_craft-repair	inf
34	insured_occupation_exec-managerial	inf
...	...	...
15	Valid_injury_claim	1.28
92	auto_model_Ultima	1.27
7	Incident_Month	1.24
3	witnesses	1.23
0	months_as_customer	1.19

95 rows × 2 columns

7.2.4 Make predictions on training data

```
# Let's re-run the model using the selected variables
col_selected_rfe = X_resampled_dummies_df[col]
X_train_sm = sm.add_constant(col_selected_rfe)
logsk = LogisticRegression(C=1e9)
logsk.fit(X_train_sm, y_resampled_dummies_df)
logm4 = sm.GLM(y_resampled_dummies_df,(sm.add_constant(col_selected_rfe)), family = sm.families.Binomial())
modres = logm4.fit()
logm4.fit().summary()
```

Dep. Variable:	fraud_reported	**No. Observations:**	838
Model:	GLM	**Df Residuals:**	788
Model Family:	Binomial	**Df Model:**	49
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-440.17
Date:	Tue, 14 Oct 2025	**Deviance:**	880.34
Time:	16:31:56	**Pearson chi2:**	838.
No. Iterations:	21	**Pseudo R-squ. (CS):**	0.2852
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.4571	0.616	-0.742	0.458	-1.664	0.750
months_as_customer	0.1912	0.087	2.186	0.029	0.020	0.363
umbrella_limit	0.2297	0.090	2.560	0.010	0.054	0.406
incident_hour_of_the_day	-0.0849	0.089	-0.953	0.341	-0.259	0.090
witnesses	0.2509	0.090	2.780	0.005	0.074	0.428
total_claim_amount	0.3202	0.305	1.051	0.293	-0.277	0.917
property_claim	-0.1656	0.380	-0.436	0.663	-0.911	0.580
Policy_Tenure	-0.0357	0.092	-0.389	0.698	-0.216	0.144
Incident_Month	-0.1533	0.088	-1.732	0.083	-0.327	0.020
Claim_consistency_ratio	-0.1292	0.166	-0.777	0.437	-0.455	0.197
Injury_vs_Vehicle_ratio	-0.1367	0.320	-0.427	0.670	-0.765	0.491
Capital_activity	-5.062e-06	2.25e-06	-2.252	0.024	-9.47e-06	-6.57e-07
Claim_component_balance	0.3039	0.379	0.801	0.423	-0.440	1.048
valid_multi_collision_damage	1.8732	0.246	7.616	0.000	1.391	2.355
valid_single_collision_damage	2.3374	0.251	9.327	0.000	1.846	2.829
Valid_police_report	-0.2831	0.230	-1.230	0.219	-0.734	0.168
Valid_injury_claim	-0.0031	0.365	-0.009	0.993	-0.719	0.713
insured_relationship_husband	-0.5996	0.301	-1.994	0.046	-1.189	-0.010
insured_relationship_other-relative	0.2537	0.304	0.834	0.404	-0.342	0.850
insured_relationship_own-child	-0.5406	0.307	-1.762	0.078	-1.142	0.061
insured_relationship_unmarried	0.1883	0.313	0.602	0.547	-0.425	0.802
insured_relationship_wife	-0.0561	0.310	-0.181	0.856	-0.664	0.552
collision_type_Front Collision	-0.3077	0.236	-1.305	0.192	-0.770	0.154
collision_type_Rear Collision	0.3346	0.219	1.531	0.126	-0.094	0.763
incident_state_NC	0.1384	0.576	0.240	0.810	-0.990	1.267
incident_state_NY	-0.7502	0.555	-1.351	0.177	-1.838	0.338
incident_state_OH	0.2794	0.701	0.399	0.690	-1.094	1.653
incident_state_SC	-0.3258	0.553	-0.589	0.556	-1.409	0.758
incident_state_VA	-0.6077	0.586	-1.038	0.299	-1.756	0.540
incident_state_WV	-1.5462	0.575	-2.688	0.007	-2.673	-0.419
auto_make_BMW	0.0148	0.354	0.042	0.967	-0.679	0.708
auto_make_Chevrolet	-0.7623	0.467	-1.633	0.103	-1.678	0.153
auto_make_Dodge	-0.5088	0.447	-1.139	0.255	-1.384	0.366
auto_make_Ford	-1.6348	0.529	-3.090	0.002	-2.672	-0.598
auto_make_Jeep	-0.4772	0.604	-0.790	0.429	-1.661	0.706
auto_make_Mercedes	-0.5505	0.386	-1.428	0.153	-1.306	0.205
auto_make_Saab	-0.8822	0.412	-2.143	0.032	-1.689	-0.075
auto_make_Suburu	0.0851	0.333	0.255	0.798	-0.568	0.738
auto_make_Toyota	0.9987	0.473	2.111	0.035	0.071	1.926
auto_model_95	0.2564	0.634	0.405	0.686	-0.985	1.498
auto_model_C300	-0.0309	0.647	-0.048	0.962	-1.299	1.237
auto_model_Camry	-2.5110	0.832	-3.017	0.003	-4.142	-0.880
auto_model_Corolla	-22.6514	1.53e+04	-0.001	0.999	-3.01e+04	3e+04
auto_model_F150	1.8539	0.702	2.640	0.008	0.477	3.230
auto_model_MDX	-0.5635	0.483	-1.166	0.244	-1.511	0.384
auto_model_Neon	-0.2124	0.662	-0.321	0.748	-1.510	1.085
auto_model_Tahoe	-0.0614	0.654	-0.094	0.925	-1.344	1.221
auto_model_Ultima	-2.1434	0.869	-2.466	0.014	-3.847	-0.440
auto_model_Wrangler	-1.2927	0.847	-1.526	0.127	-2.953	0.368
auto_model_X6	0.3906	0.661	0.591	0.555	-0.905	1.686

- Predict the probabilities on the training data

```
# Predict the probabilities on the training data
y_resampled_dummies_df_predict = modres.predict(X_train_sm)
# Reshape it into an array
y_resampled_dummies_df_predict[:20]
```

```
0     0.489451
1     0.527320
2     0.423714
3     0.405732
4     0.833067
5     0.227228
6     0.549463
7     0.383317
8     0.931196
9     0.509928
10    0.241543
11    0.732491
12    0.734068
13    0.583530
14    0.232816
15    0.165385
16    0.922236
17    0.134351
18    0.502515
19    0.425169
dtype: float64
```

7.2.5 Creating a DataFrame that includes actual fraud reported flags, predicted probabilities, and a column indicating predicted classifications based on a cutoff value of 0.5

```
# Create a new DataFrame containing the actual fraud reported flag and the probabilities predicted by the model
y_train_pred_final = pd.DataFrame({'fraud':y_resampled_dummies_df.values, 'fraud_prob':y_resampled_dummies_df_pre
# Create new column indicating predicted classifications based on a cutoff value of 0.5
y_train_pred_final['fraud_predicted'] = y_train_pred_final.fraud_prob.map(lambda x: 1 if x > 0.5 else 0)
y_train_pred_final.head()
```

	fraud	fraud_prob	fraud_predicted
0	0	0.489451	0
1	0	0.527320	1
2	1	0.423714	0
3	0	0.405732	0
4	1	0.833067	1

7.2.6 Check the accuracy of the model

```
# Import metrics from sklearn for evaluation
from sklearn import metrics

# Check the accuracy of the model
print(metrics.accuracy_score(y_train_pred_final.fraud, y_train_pred_final.fraud_predicted))
```

```
0.7386634844868735
```

7.2.7 Create a confusion matrix based on the predictions made on the training data

```
# Create confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.fraud, y_train_pred_final.fraud_predicted )
print(confusion)
```

```
[[287 132]
 [ 87 332]]
```

7.2.8 Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and false negative
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

7.2.9 Calculate sensitivity, specificity, precision, recall and F1-score

```
# Calculate the sensitivity
sensitivity = TP / float(TP+FN)
print("sensitivity = ",sensitivity)
# Calculate the specificity
specificity = TN / float(TN+FP)
print("specificity = ",specificity)
# Calculate Precision
Precision = TP / float(TP+FP)
print("Precision = ",Precision)
# Calculate Recall
Recall = TP / float(TP+FN)
print("Recall = ",Recall)
```

```
sensitivity =  0.7923627684964201
specificity =  0.684964200477327
Precision =  0.7155172413793104
Recall =  0.7923627684964201
```

```
# Calculate F1 Score
F1_score = 2 * (Precision * Recall) / (Precision + Recall)
print("F1_score = ",F1_score)
```

```
F1_score =  0.7519818799546999
```

7.3 Find the Optimal Cutoff

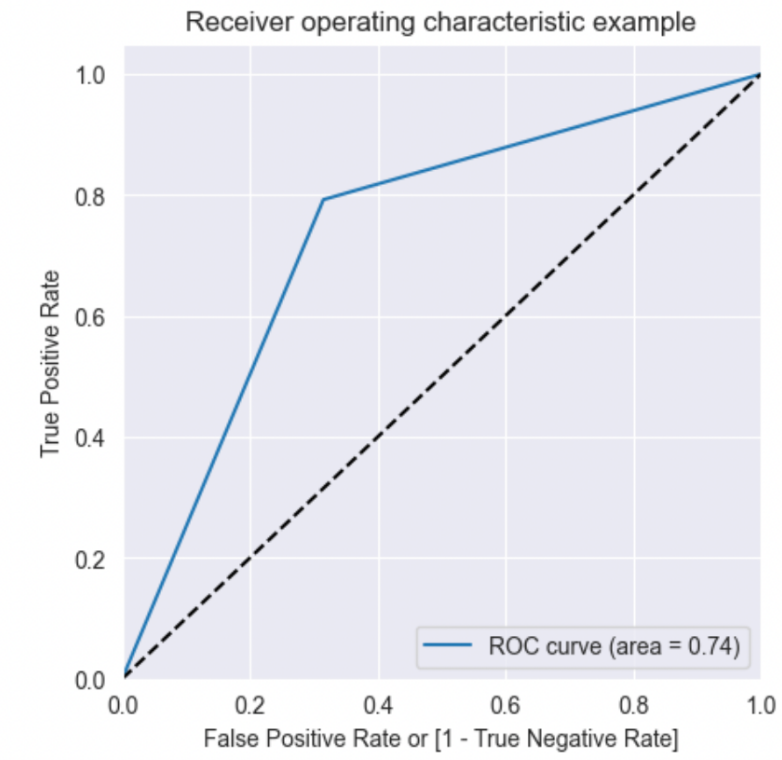
Find the optimal cutoff to improve model performance by evaluating various cutoff values and their impact on relevant metrics.

7.3.1 Plot ROC Curve to visualise the trade-off between true positive rate and false positive rate across different classification thresholds

```
# Import libraries or function to plot the ROC curve
from sklearn.metrics import roc_curve, roc_auc_score, RocCurveDisplay
# Define ROC function
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

```
# Call the ROC function
pr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.fraud, y_train_pred_final.fraud_predicted, drop_inter
draw_roc(y_train_pred_final.fraud, y_train_pred_final.fraud_predicted)
```



7.3.2 Predict on training data at various probability cutoffs

```
# Create columns with different probability cutoffs to explore the impact of cutoff on model performance
numbers = [float(x)/10 for x in range(10)]
for i in numbers:
    y_train_pred_final[i]= y_train_pred_final.fraud_prob.map(lambda x: 1 if x > i else 0)
y_train_pred_final.head()
```

	fraud	fraud_prob	fraud_predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.489451	0	1	1	1	1	1	0	0	0	0	0
1	0	0.527320	1	1	1	1	1	1	1	0	0	0	0
2	1	0.423714	0	1	1	1	1	1	0	0	0	0	0
3	0	0.405732	0	1	1	1	1	1	0	0	0	0	0
4	1	0.833067	1	1	1	1	1	1	1	1	1	1	0

7.3.3 Plot accuracy, sensitivity, specificity at different values of probability cutoffs

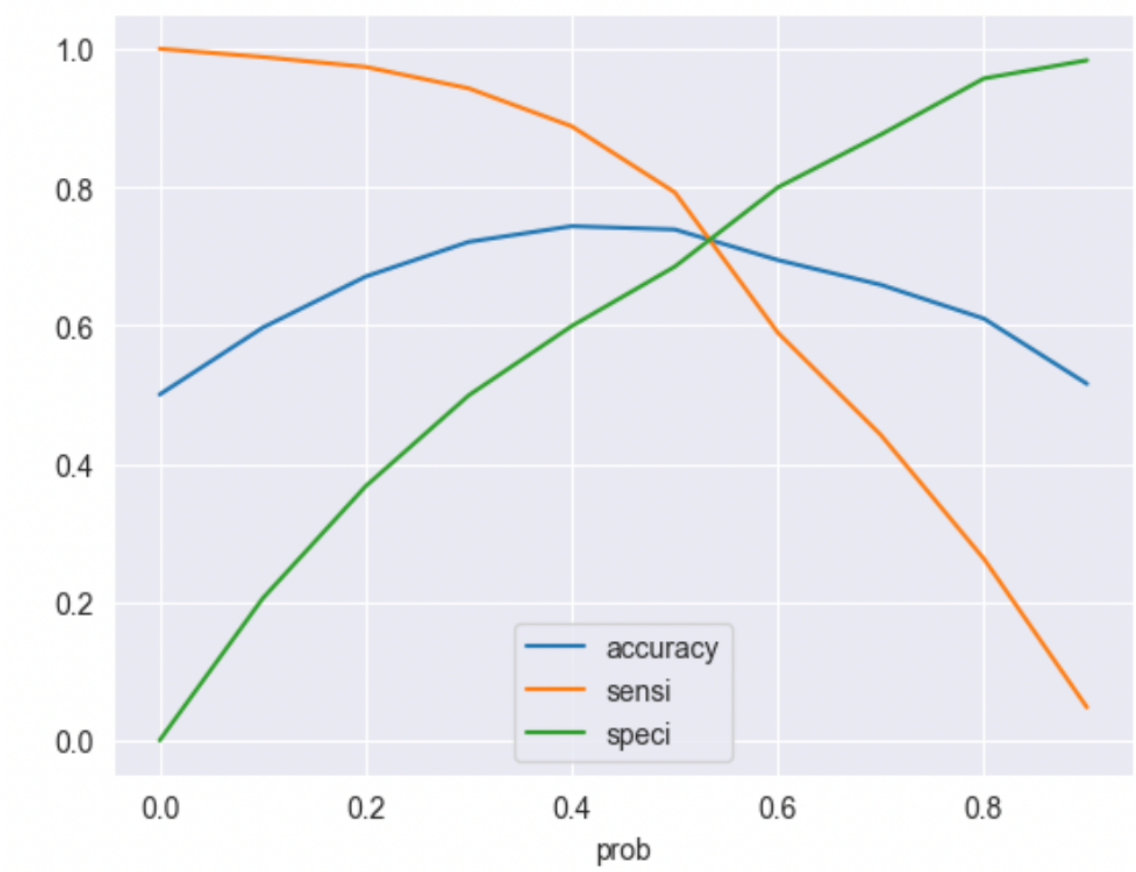
```
# Create a DataFrame to see the values of accuracy, sensitivity, and specificity at different values of probabili
# Now let's calculate accuracy sensitivity and specificity for various probability cutoffs.
cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
from sklearn.metrics import confusion_matrix

# TP = confusion[1,1] # true positive
# TN = confusion[0,0] # true negatives
# FP = confusion[0,1] # false positives
# FN = confusion[1,0] # false negatives

num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
for i in num:
    cm1 = metrics.confusion_matrix(y_train_pred_final.fraud, y_train_pred_final[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
print(cutoff_df)
```

```
     prob  accuracy     sensi     speci
0.0   0.0  0.500000  1.000000  0.000000
0.1   0.1  0.596659  0.988067  0.205251
0.2   0.2  0.670644  0.973747  0.367542
0.3   0.3  0.720764  0.942721  0.498807
0.4   0.4  0.743437  0.887828  0.599045
0.5   0.5  0.738663  0.792363  0.684964
0.6   0.6  0.694511  0.589499  0.799523
0.7   0.7  0.658711  0.441527  0.875895
0.8   0.8  0.609785  0.262530  0.957041
0.9   0.9  0.515513  0.047733  0.983294
```



7.3.4 Create a column for final prediction based on optimal cutoff

```
# Create a column for final prediction based on the optimal cutoff
#From above 0.5 seems to be the optimal cutoff
y_train_pred_final['final_predicted'] = y_train_pred_final.fraud_prob.map( lambda x: 1 if x > 0.6 else 0)
y_train_pred_final.head()
```

	fraud	fraud_prob	fraud_predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	final_predicted
0	0	0.489451	0	1	1	1	1	1	0	0	0	0	0	0
1	0	0.527320	1	1	1	1	1	1	1	0	0	0	0	0
2	1	0.423714	0	1	1	1	1	1	0	0	0	0	0	0
3	0	0.405732	0	1	1	1	1	1	0	0	0	0	0	0
4	1	0.833067	1	1	1	1	1	1	1	1	1	1	0	1

7.3.5 Calculate the accuracy

```
# Check the accuracy now
metrics.accuracy_score(y_train_pred_final.fraud, y_train_pred_final.final_predicted)
```

```
0.6945107398568019
```

7.3.6 Create confusion matrix

```
# Create the confusion matrix once again
confusion2 = metrics.confusion_matrix(y_train_pred_final.fraud, y_train_pred_final.final_predicted)
confusion2
```

```
array([[335,  84],
       [172, 247]])
```

7.3.7 Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and false negative
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

7.3.8 Calculate sensitivity, specificity, precision, recall and F1-score of the model

```
# Calculate the sensitivity
sensitivity = TP / float(TP+FN)
print("sensitivity = ",sensitivity)
# Calculate the specificity
specificity = TN / float(TN+FP)
print("specificity = ",specificity)
# Calculate Precision
Precision = TP / float(TP+FP)
print("Precision = ",Precision)
# Calculate Recall
Recall = TP / float(TP+FN)
print("Recall = ",Recall)
```

```
sensitivity =  0.5894988066825776
specificity =  0.7995226730310262
Precision =  0.7462235649546828
Recall =  0.5894988066825776
```

7.3.9 Plot precision-recall curve

```
# Plot precision-recall curve

#Precision TP / TP + FP
```

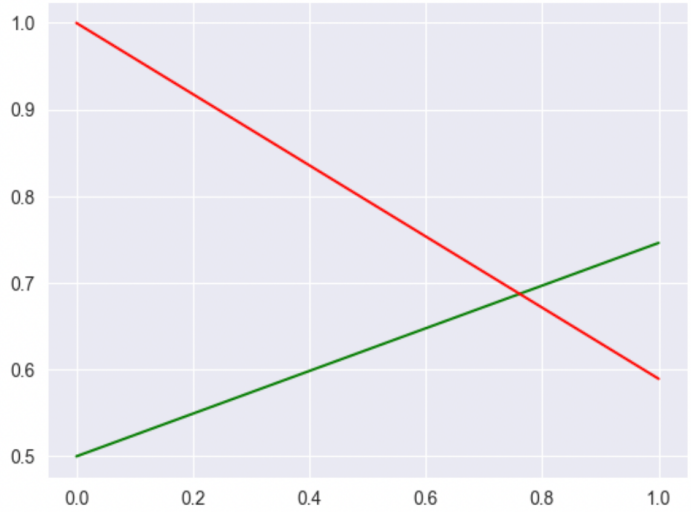
```
confusion[1,1]/(confusion[0,1]+confusion[1,1])
```

```
np.float64(0.7155172413793104)
```

```
# Recall = TP / TP + FN
confusion[1,1]/(confusion[1,0]+confusion[1,1])
```

```
np.float64(0.7923627684964201)
```

```
from sklearn.metrics import precision_score, recall_score
p, r, thresholds = precision_recall_curve(y_train_pred_final.fraud, y_train_pred_final.final_predicted)
plt.plot(thresholds, p[:-1], "g-")
plt.plot(thresholds, r[:-1], "r-")
plt.show()
```



7.4 Build Random Forest Model

7.4.1 Import necessary libraries

```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score, GridSearchCV
```

7.4.2 Build the random forest model

Started to build the Random Forest Model with standard params
Number of trees: 10 with each tree max depth of 4 with each node having max of 10 features.
After initializing I had fitted the model with train data

```
%%time
```

```
rf = RandomForestClassifier(n_estimators=10, max_depth=4, max_features=10,
random_state=100, oob_score=True)
rf.fit(X_train_sm, y_resampled_dummies_df)
```

Output:

```
CPU times: user 9.5 ms, sys: 1.16 ms, total: 10.7 ms
Wall time: 9.87 ms
RandomForestClassifier
RandomForestClassifier(max_depth=4, max_features=10, n_estimators=10,
                       oob_score=True, random_state=100)
```

7.4.3 Get feature importance scores and select important features

I used the random forest attribute

```
rf.feature_importances_
```

Output:

All the 50 features which was present, the score of importance is displayed in the second column.

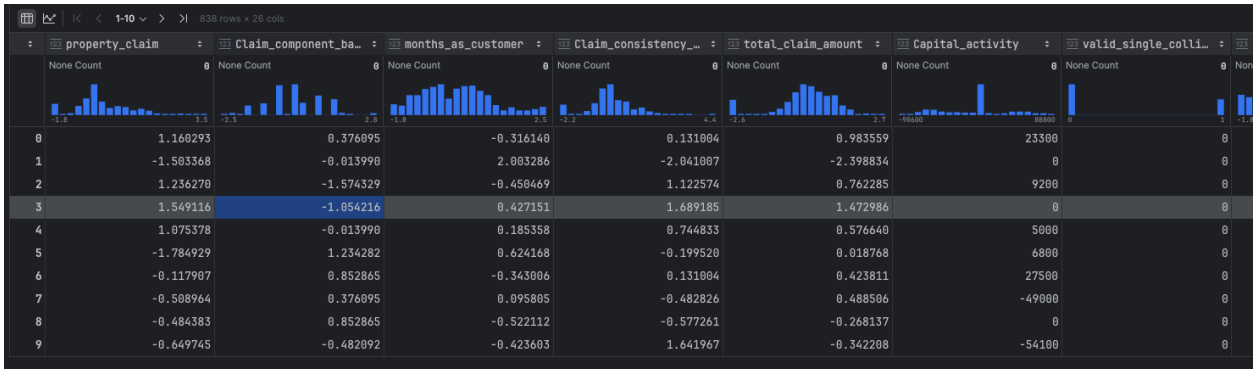
50 rows × 2 cols

	Feature	Importance
	None Count 0	None Count 0
	50 Unique values	
6	property_claim	0.155438
12	Claim_component_balance	0.123308
1	months_as_customer	0.081605
9	Claim_consistency_ratio	0.078829
5	total_claim_amount	0.073627
11	Capital_activity	0.069056
14	valid_single_collision_…	0.068310
7	Policy_Tenure	0.057579
3	incident_hour_of_the_day	0.055403
29	incident_state_WV	0.041323
13	valid_multi_collision_d…	0.036057
2	umbrella_limit	0.024352
10	Injury_vs_Vehicle_ratio	0.023260
8	Incident_Month	0.013368

Then I selected all the features with importance more than “0.004” which selected 26 features for the model to process.

```
# Select features with high importance scores
important_features = feature_importance_df[feature_importance_df['Importance'] >
0.004]
```

Output:



	property_claim	Claim_component_ba…	months_as_customer	Claim_consistency_…	total_claim_amount	Capital_activity	valid_single_colli…
	None Count 0	None Count 0	None Count 0	None Count 0	None Count 0	None Count 0	None Count 0
0	1.160293	0.376095	-0.316140	0.131004	0.983559	23300	0
1	-1.503368	-0.013990	2.003286	-2.041007	-2.398834	0	0
2	1.236270	-1.574329	-0.450469	1.122574	0.762285	9200	0
3	1.549116	-1.054216	0.427151	1.689185	1.472986	0	0
4	1.075378	-0.013990	0.185358	0.744833	0.576640	5000	0
5	-1.784929	1.234282	0.624168	-0.199520	0.018768	6800	0
6	-0.117907	0.852865	-0.343006	0.131004	0.423811	27500	0
7	-0.508964	0.376095	0.095805	-0.482826	0.488506	-49000	0
8	-0.484383	0.852865	-0.522112	-0.577261	-0.268137	0	0
9	-0.649745	-0.482092	-0.423603	1.641967	-0.342208	-54100	0

7.4.4 Train the model with selected features

Trained the model with new 26 features.

Code and Output

```
# Fit the model on the training data with selected features
rf.fit(X_train_important_feature, y_resampled_dummies_df)
```

✓ [229] 31ms

RandomForestClassifier

```
RandomForestClassifier(max_depth=4, max_features=10, n_estimators=10,
                       oob_score=True, random_state=100)
```

7.4.5 Generate predictions on the training data

Run prediction on the training data

```
y_train_pred = rf.predict(X_train_important_feature)
```

7.4.6 Check accuracy of the model

Using the sklearn metrics library calculated the accuracy of the train data.

```
# Check accuracy of the model
from sklearn.metrics import confusion_matrix, accuracy_score

train_accuracy = accuracy_score(y_resampled_dummies_df, y_train_pred)
print("Train Accuracy :", train_accuracy)
```

Output:

```
Train Accuracy : 0.7875894988066826
```

7.4.7 Create confusion matrix

Created a confusion matrix usng the SKlearn package

```
train_confusion = confusion_matrix(y_resampled_dummies_df, y_train_pred)
print("Train Confusion Matrix:")
print(train_confusion)
```

Output

```
Train Confusion Matrix:
[[291 128]
 [ 50 369]]
```

7.4.8 Create variables for true positive, true negative, false positive and false negative

```
TN, FP, FN, TP = train_confusion.ravel() # true negatives, false positives, false
negatives, true positive
```

7.4.9 Calculate sensitivity, specificity, precision, recall and F1-score of the model

Calculated the sensitivity aka recall, specificity, precision and F1 Score using the standard formula

```
recall = TP / float(TP + FN)

# Calculate the specificity
print(TN / float(TN+FP))


# Calculate Precision
precision = TP / float(TP + FP)


# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
```

Output

```
Sensitivity:
0.8806682577565632
Specificity:
0.6945107398568019
Precision:
0.7424547283702213
Recall:
0.8806682577565632
F1 Score:
0.8056768558951964
```

7.4.10 Check if the model is overfitting training data using cross validation

Used the sklearn model selection cross validate function to calculate the overfitting, Since in the previous step we calculated the sensitivity aka recall, specificity, precision and F1 Score, used the same scoring technique to say the F1 Score on train and test. As expected the model **is experiencing the Overfitting** as we can see 6 % greater in Training F1 Score and Test F1 Score.

```
scoring = ['f1', 'accuracy', 'precision', 'recall']

cv_results = cross_validate(
    estimator=rf,
    X=X_train_important_feature,
    y=y_resampled_dummies_df,
    cv=5,
    scoring=scoring,
    return_train_score=True,
    n_jobs=-1
)
train_f1_mean = np.mean(cv_results['train_f1'])
validation_f1_mean = np.mean(cv_results['test_f1'])
```

Output:

```
Training F1 Score: 0.8078
Test F1 Score: 0.7432
----------------------------------------
```

7.5 Hyperparameter Tuning

7.5.1 Use grid search to find the best hyperparameter values

Used the sklearn KFold and GridSearchCV for finding the best hyperparameters,

```
def use_grid_search_cv(X_train_grid_cv, y_train_grid_cv):
    folds = KFold(n_splits = 3, shuffle = True, random_state = 100)
    hyper_params = {
    'max_depth': [3, 5],
    'n_estimators': [10, 100, 200],
    'min_samples_split': [10, 20, 30],
    'min_samples_leaf': [5, 10],
}
    model_cv = GridSearchCV(estimator = rf,
                            param_grid = hyper_params,
                            scoring= 'f1',
                            cv = folds,
                            verbose = 1,
                            return_train_score=True)

    model_cv.fit(X_train_grid_cv, y_train_grid_cv)
    return model_cv.cv_results_
```

Output:

Copy pasting only the Mean score of test and train data as that was the metric used to derive the hyper params.

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
'mean_train_score': array([0.76724949, 0.78625979, 0.7937922 , 0.77018613, 0.7854358 ,
       0.78838074, 0.75904531, 0.78083776, 0.7797087 , 0.74370653,
       0.77425707, 0.78029202, 0.74370653, 0.77425707, 0.78029202,
       0.74560664, 0.77372709, 0.77940961, 0.83256469, 0.86427579,
       0.86601614, 0.84022438, 0.84851981, 0.84949217, 0.82792928,
       0.83578221, 0.83910987, 0.81568107, 0.83676993, 0.83625567,
       0.81568107, 0.83676993, 0.83625567, 0.80196311, 0.82946313,
       0.82304696]),
'mean_test_score': array([0.70257431, 0.74367229, 0.74235033, 0.72251254, 0.7346105 ,
       0.73948253, 0.68391885, 0.73249134, 0.73558367, 0.69725899,
       0.73134116, 0.7332476 , 0.69725899, 0.73134116, 0.7332476 ,
       0.67213262, 0.7280895 , 0.73769709, 0.73750827, 0.77656375,
       0.77919834, 0.74638236, 0.76396303, 0.77375593, 0.7359688 ,
       0.75168569, 0.76284918, 0.733822  , 0.76354368, 0.76568572,
       0.733822  , 0.76354368, 0.76568572, 0.72990791, 0.75483957,
       0.76710221]),
```

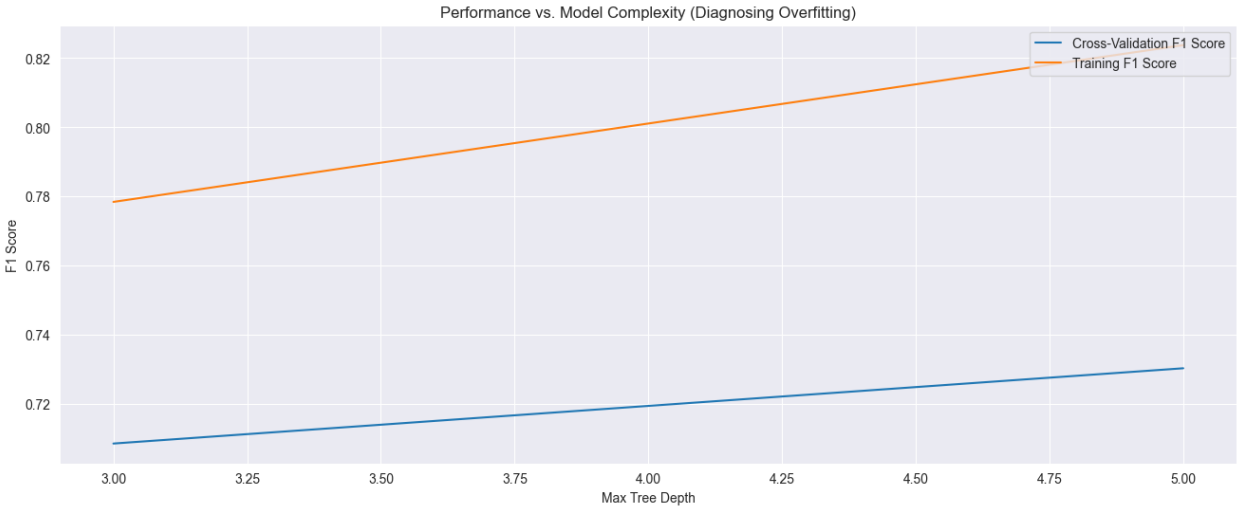
Plotting Cross Validation Results:

Plotted the cv results in graphical view to see the changes of the hyper parameters suggesting. As in the output it is evident that taining score is .82 while the test is .73 for max depth of 5, after that the model gets complex.

```
plt.figure(figsize=(16,6))

plot_param = 'param_max_depth'
grouped_results = pd.DataFrame(cv_results).groupby(plot_param)[['mean_test_score',
'mean_train_score']].mean().reset_index()
```

Output:



7.5.2 Build a random forest model based on hyperparameter tuning results

As per suggestion from the hyper tuning final result used the number of estimators as 100 with max depth 5 minimun samples split of 20 for each node, minimum samples for the leaf node is 5.

```
rf = RandomForestClassifier(n_estimators=100, max_depth=5, min_samples_split=20, min_samples_leaf=5, random_state=100, oob_score=True)
```

7.5.3 Make predictions on training data

```
# Make predictions on training data
rf.fit(X_train_important_feature, y_resampled_dummies_df)
y_train_pred = rf.predict(X_train_important_feature)
```

7.5.4 Check accuracy of Random Forest Model

Using the sklearn metrics library calculated the accuracy of the train data.

Output:

```
Train Accuracy : 0.8281622911694511
```

7.5.5 Create confusion matrix

Created a confusion matrix usng the SKlearn package

Output:

```
Train Confusion Matrix:
[[312 107]
 [ 37 382]]
```

7.5.6 Create variables for true positive, true negative, false positive and false

```
TN, FP, FN, TP = train_confusion.ravel() # true negatives, false positives, false negatives, true positive
```

7.5.7 Calculate sensitivity, specificity, precision, recall and F1-score of the model

Calculated the sensitivity aka recall, specificity, precision and F1 Score using the standard formula

```
recall = TP / float(TP + FN)

# Calculate the specificity
print(TN / float(TN+FP))
```

```
# Calculate Precision
precision = TP / float(TP + FP)


# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
```

Output:

```
Sensitivity:
0.9116945107398569
Specificity:
0.7446300715990454
Precision:
0.7811860940695297
Recall:
0.9116945107398569
F1 Score:
0.841409691629956
```

Rerun the Cross validation on train and test data to see the improvement in the F1 Score

```
scoring = ['f1', 'accuracy', 'precision', 'recall']

cv_results = cross_validate(
    estimator=rf,
    X=X_train_important_feature,
    y=y_resampled_dummies_df,
    cv=5,
    scoring=scoring,
    return_train_score=True,
    n_jobs=-1
)
train_f1_mean = np.mean(cv_results['train_f1'])
validation_f1_mean = np.mean(cv_results['test_f1'])
```

Output:

```
F1 Score after hyper tuning parameters
Training F1 Score: 0.8326
Validation F1 Score: 0.7529
--------------------------------------
```

Finding the optimal Threshold for increasing the precision:

During the fine tunning the hyper parameters we noticed that there was a low precision, however the tuning the hyper parameters, so we tried to see if we can increased the threshold for the model to classify as **“Fraud”** with increase in threshold. The output showed that default threshold was working fine as the threshold the model was struggling to identify the “Fraud” case as it lagged the necessary data.

```
thresholds = np.arange(0.5, 0.9, 0.05) # Check thresholds from 0.50 to 0.85
best_threshold = 0.5
best_f1 = 0
```

Output:

```
--- Threshold Optimization Results ---
Threshold: 0.50 | Precision: 0.3370 | Recall: 0.5082 | F1 Score: 0.4052
Threshold: 0.55 | Precision: 0.3208 | Recall: 0.2787 | F1 Score: 0.2982
Threshold: 0.60 | Precision: 0.2500 | Recall: 0.0820 | F1 Score: 0.1235
Threshold: 0.65 | Precision: 0.3333 | Recall: 0.0328 | F1 Score: 0.0597
Threshold: 0.70 | Precision: 0.0000 | Recall: 0.0000 | F1 Score: 0.0000
Threshold: 0.75 | Precision: 0.0000 | Recall: 0.0000 | F1 Score: 0.0000
Threshold: 0.80 | Precision: 0.0000 | Recall: 0.0000 | F1 Score: 0.0000
Threshold: 0.85 | Precision: 0.0000 | Recall: 0.0000 | F1 Score: 0.0000

Recommended optimal threshold (based on F1): 0.50
```

8. Prediction and Model Evaluation

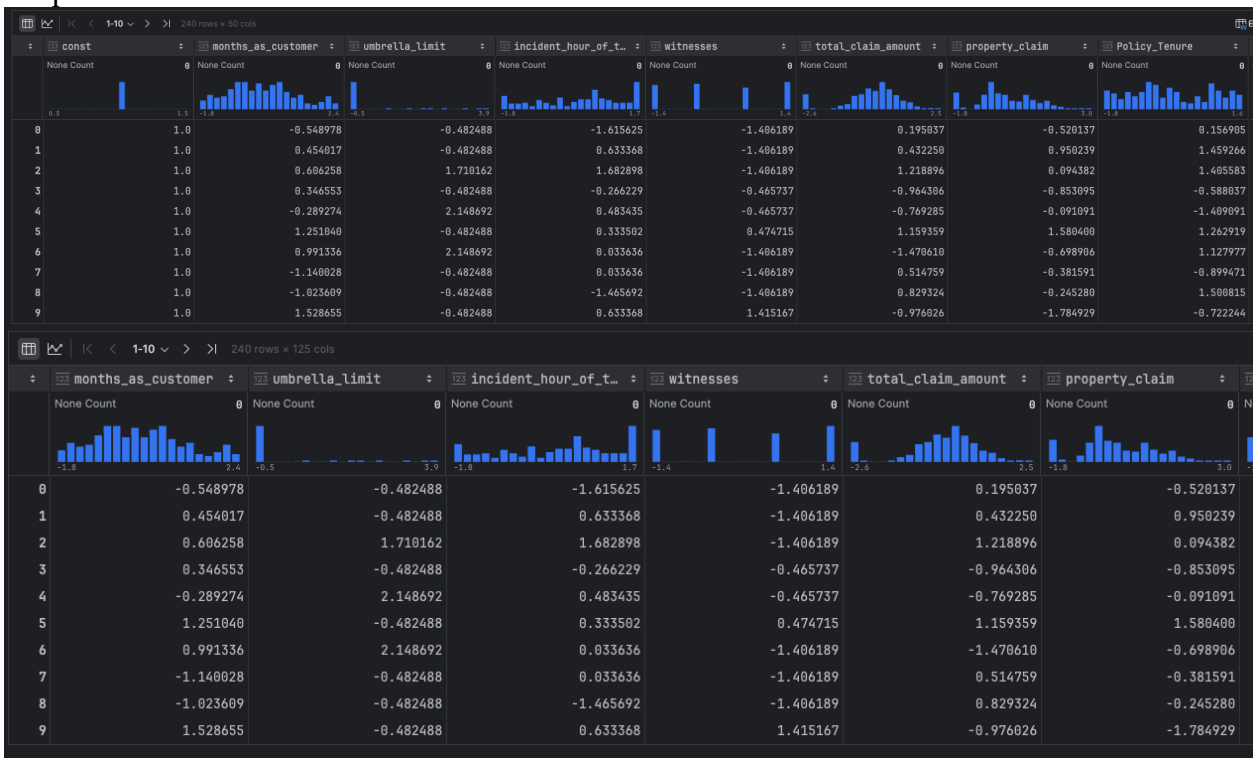
8.1 Make predictions over validation data using logistic regression model

8.1.1 Select relevant features for validation data and add constant

Listed the Validation DF and selected only the columns by Recursive Feature Elimanaiton Cross Validation (RFECV)

```
# Select the relevant features for validation data
# Select only the columns selected by RFECV
col_selected_rfe = X_dummies_validation_df[col]
# Add constant to X_validation
X_validation_sm = sm.add_constant(col_selected_rfe)
# Check the data
X_validation_sm
```

Output:



	const	months_as_customer	umbrella_limit	incident_hour_of_t…	witnesses	total_claim_amount	property_claim	Policy_Tenure
0	1.0	-0.548978	-0.482488	-1.615625	-1.406189	0.195037	-0.520137	0.156905
1	1.0	0.454017	-0.482488	0.633368	-1.406189	0.432250	0.950239	1.459266
2	1.0	0.606258	1.710162	1.682898	-1.406189	1.218896	0.094382	1.405583
3	1.0	0.346553	-0.482488	-0.266229	-0.465737	-0.964306	-0.853095	-0.588037
4	1.0	-0.289274	2.148692	0.483435	-0.465737	-0.769285	-0.091091	-1.409091
5	1.0	1.251040	-0.482488	0.333502	0.474715	1.159359	1.580400	1.262919
6	1.0	0.991336	2.148692	0.033636	-1.406189	-1.470610	-0.698906	1.127977
7	1.0	-1.140028	-0.482488	0.033636	-1.406189	0.514759	-0.381591	-0.899471
8	1.0	-1.023609	-0.482488	-1.465692	-1.406189	0.829324	-0.245280	1.508815
9	1.0	1.528655	-0.482488	0.633368	1.415167	-0.976026	-1.784929	-0.722244

	months_as_customer	umbrella_limit	incident_hour_of_t…	witnesses	total_claim_amount	property_claim
0	-0.548978	-0.482488	-1.615625	-1.406189	0.195037	-0.520137
1	0.454017	-0.482488	0.633368	-1.406189	0.432250	0.950239
2	0.606258	1.710162	1.682898	-1.406189	1.218896	0.094382
3	0.346553	-0.482488	-0.266229	-0.465737	-0.964306	-0.853095
4	-0.289274	2.148692	0.483435	-0.465737	-0.769285	-0.091091
5	1.251040	-0.482488	0.333502	0.474715	1.159359	1.580400
6	0.991336	2.148692	0.033636	-1.406189	-1.470610	-0.698906
7	-1.140028	-0.482488	0.033636	-1.406189	0.514759	-0.381591
8	-1.023609	-0.482488	-1.465692	-1.406189	0.829324	-0.245280
9	1.528655	-0.482488	0.633368	1.415167	-0.976026	-1.784929

8.1.2 Make predictions over validation data

```
# Make predictions on the validation data and store it in the variable
'y_validation_pred'
```

```
y_validation_predict = logsk.predict(X_validation_sm)
# Reshape it into an array
y_validation_predict[:20]
```

Output:

```
array([1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0])
```

8.1.3 Create DataFrame with actual values and predicted values for validation data

```
#  Create DataFrame with actual values and predicted values for validation data
y_validation_pred_final = pd.DataFrame({'fraud':y_dummies_validation_df.values,
'fraud_prob':y_validation_predict})
# Create new column indicating predicted classifications based on a cutoff value
of 0.5
y_validation_pred_final['fraud_predicted'] =
y_validation_pred_final.fraud_prob.map(lambda x: 1 if x > 0.6 else 0)
y_validation_pred_final.head()
```

Output:

5 rows × 3 cols

	fraud	fraud_prob	fraud_predicted
0	0	1	1
1	0	0	0
2	1	0	0
3	0	0	0
4	0	0	0

8.1.4 Make final prediction based on cutoff value

```
# Make final predictions on the validation data using the optimal cutoff
y_validation_pred_final['final_predicted'] =
y_validation_pred_final.fraud_prob.map( lambda x: 1 if x > 0.6 else 0)
y_validation_pred_final.head()
```

Output:

5 rows × 4 cols

	fraud	fraud_prob	fraud_predicted	final_predicted
0	0	1	1	1
1	0	0	0	0
2	1	0	0	0
3	0	0	0	0
4	0	0	0	0

8.1.5 Check the accuracy of logistic regression model on validation data

```
metrics.accuracy_score(y_validation_pred_final.fraud,
y_validation_pred_final.final_predicted)
```

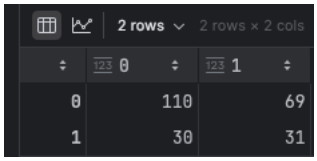
Output:

0.5875

8.1.6 Create confusion matrix

```
# Create the confusion matrix
confusion_validation = metrics.confusion_matrix(y_validation_pred_final.fraud,
y_validation_pred_final.final_predicted)
confusion_validation
```

Output:



	0	1
0	110	69
1	30	31

8.1.7 Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and false
negative
TP = confusion_validation[1,1] # true positive
TN = confusion_validation[0,0] # true negatives
FP = confusion_validation[0,1] # false positives
FN = confusion_validation[1,0] # false negatives
```

8.1.8 Calculate sensitivity, specificity, precision, recall and f1 score of the model

```
# Calculate the sensitivity
sensitivity = TP / float(TP+FN)
print("sensitivity = ",sensitivity)
# Calculate the specificity
specificity = TN / float(TN+FP)
print("specificity = ",specificity)
# Calculate Precision
Precision = TP / float(TP+FP)
print("Precision = ",Precision)
# Calculate Recall
Recall = TP / float(TP+FN)
print("Recall = ",Recall)
# Calculate F1 Score
log_reg_validation_f1_score = 2 * (log_reg_validation_Precision *
log_reg_validation_Recall) / (log_reg_validation_Precision +
log_reg_validation_Recall)
print("F1 Score = ", log_reg_validation_f1_score)
```

Output:

```
sensitivity =  0.5081967213114754
specificity =  0.6145251396648045
Precision =  0.31
Recall =  0.5081967213114754
F1 Score =  0.3850931677018634
```

8.2 Make predictions over validation data using random forest mode

8.2.1 Select the important features and make predictions over validation data

Used the important feature found in Hyper tunning on the validation / test data and predicted it

```
# Select the relevant features for validation data
print(X_validation_important_feature.columns)
# Make predictions on the validation data
y_validation_pred = rf.predict(X_validation_important_feature)
```

Output:

```
Index(['property_claim', 'Claim_component_balance', 'months_as_customer',
       'Claim_consistency_ratio', 'total_claim_amount', 'Capital_activity',
       'valid_single_collision_damage', 'Policy_Tenure',
       'incident_hour_of_the_day', 'incident_state_WV',
       'valid_multi_collision_damage', 'umbrella_limit',
       'Injury_vs_Vehicle_ratio', 'Incident_Month',
       'insured_relationship_other-relative', 'auto_model_Camry',
       'auto_make_Suburu', 'collision_type_Rear Collision',
       'auto_make_Mercedes', 'incident_state_SC', 'auto_model_95',
       'Valid_police_report', 'insured_relationship_husband', 'auto_model_X6',
       'auto_make_Dodge', 'insured_relationship_unmarried'],
      dtype='object')
```

8.2.2 Check accuracy of random forest model

```
# Check accuracy
validation_accuracy = accuracy_score(y_dummies_validation_df, y_validation_pred)
print("Validation Accuracy :", validation_accuracy)
```

Output:

```
Validation Accuracy : 0.6208333333333333
```

8.2.3 Create confusion matrix

```
# Create the confusion matrix
validation_confusion = confusion_matrix(y_dummies_validation_df,
y_validation_pred)
print("Validation Confusion Matrix:")
print(validation_confusion)
```

Output:

```
Validation Confusion Matrix:
[[118  61]
 [ 30  31]]
```

8.2.4 Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and false
negative
TP = confusion_validation[1,1] # true positive
TN = confusion_validation[0,0] # true negatives
FP = confusion_validation[0,1] # false positives
FN = confusion_validation[1,0] # false negatives
```

8.2.5 Calculate sensitivity, specificity, precision, recall and f1 score of the model

```
# Calculate the sensitivity
sensitivity = TP / float(TP+FN)
print("sensitivity = ",sensitivity)
# Calculate the specificity
specificity = TN / float(TN+FP)
print("specificity = ",specificity)
# Calculate Precision
Precision = TP / float(TP+FP)
print("Precision = ",Precision)
# Calculate Recall
Recall = TP / float(TP+FN)
print("Recall = ",Recall)
# Calculate F1 Score
```

Output:

```
Sensitivity:
0.5081967213114754
Specificity:
0.659217877094972
Precision:
0.33695652173913043
Recall:
0.5081967213114754
F1 Score:
0.40522875816993464
```

Evaluation and Conclusion- Write the conclusion.

Key Observations:

1. **Severe Overfitting in RF (High Variance):** The Random Forest's performance collapses dramatically on unseen data. Its training accuracy (0.82) and F1 mean score (0.83) are excellent, but the Validation Accuracy drops by 0.62 and the Validation Recall drops by 0.50 . The RF is memorizing the training data's noise.
2. **Weak Generalization (Low Precision):** Both models suffer from very low Validation Precision (0.31, 0.33). This means that when the model flags a claim as fraudulent, it is wrong about two-thirds of the time(high False Positives), indicating a high-cost liability for the company.
3. **Equivalent Recall:** Both models achieved an identical Recall on the validation set, suggesting they are equally effective at finding actual fraud cases, even though is far more complex.

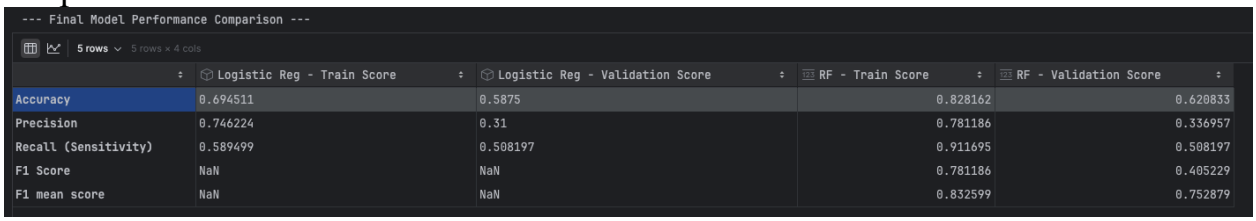
Conclusion and Final Recommendation

The ultimate goal of this project is to recommend a reliable and actionable model for fraud detection.

1. Final Model Choice: Logistic Regression

Despite its lower metrics, the **Logistic Regression (LR) model is the more stable classifier** in this comparison, as its scores are relatively closer (though not perfectly close) across the train/validation split.

Output:



--- Final Model Performance Comparison ---

5 rows 5 rows × 4 cols

	Logistic Reg - Train Score	Logistic Reg - Validation Score	RF - Train Score	RF - Validation Score
Accuracy	0.694511	0.5875	0.828162	0.620833
Precision	0.746224	0.31	0.781186	0.336957
Recall (Sensitivity)	0.589499	0.508197	0.911695	0.508197
F1 Score	NaN	NaN	0.781186	0.405229
F1 mean score	NaN	NaN	0.832599	0.752879