**Problem Statement**

**Objective:** To develop a predictive classification model that categorizes properties into "Easy," "Medium," and "Hard" risk categories. This classification will facilitate the automation of the insurance underwriting process by determining the ease of providing insurance and setting appropriate premiums.

**Importance:** Accurately classifying properties into risk categories is crucial for insurance companies as it enables them to:

1. **Set Appropriate Premiums:** Charge premiums that reflect the true risk associated with insuring a property.
2. **Improve Risk Management:** Identify high-risk properties and take necessary actions to mitigate potential losses.
3. **Streamline Underwriting:** Automate the decision-making process, reducing the time and effort required for manual assessments.

**Dataset**

**Features:**

* **Geographical and Environmental Factors:**
  + **Distance\_to\_Coast:** Distance of the property from the coast.
  + **Fire\_Risk\_Zone:** Risk of fire in the property's location (Low, Medium, High).
  + **Hurricane\_Risk\_Zone:** Risk of hurricanes in the property's location (Low, Medium, High).
  + **DIS\_WaterBody\_BU:** Distance to the nearest water body.
  + **DIS\_Firestation\_BU:** Distance to the nearest fire station.
* **Property Characteristics:**
  + **Footprint\_Area:** Area of the property's footprint.
  + **Story\_Num\_BU:** Number of stories in the building.
  + **Square\_Footage:** Total square footage of the property.
  + **Building\_Height\_BU:** Height of the building.
  + **Ground\_Height\_BU:** Height of the ground the building is on.
  + **Year\_Built:** Year the property was built.
  + **No\_of\_Buildings:** Number of buildings on the property.
  + **Construction:** Type of construction (Wood, Brick, Concrete).
  + **Roof\_Type\_RF:** Type of roof (Gable, Hip, Flat).
  + **Roof\_Material\_RF:** Material of the roof (Asphalt Shingles, Metal, Tile).
  + **Roof\_Condition\_RF:** Condition of the roof (New, Good, Worn).
  + **Roof\_Material\_Condition:** Condition of the roof material (Good, Moderate, Poor).
  + **Roof\_Evidence\_RF:** Evidence of roof damage (None, Minor, Major).
  + **Tree\_Height\_BU:** Height of the trees around the building.
  + **Elevation:** Elevation of the property.
* **Additional Property Features:**
  + **Solar\_Panels\_RF:** Presence of solar panels.
  + **Air\_Conditioner\_RF:** Presence of air conditioners.
  + **Skylights\_RF:** Presence of skylights.
  + **Chimneys\_RF:** Presence of chimneys.
  + **Tree\_Overhang\_RF:** Trees overhanging the roof.
  + **Gable\_Wall\_DI\_RF:** Condition of the gable wall (Intact, Damaged).
  + **DIS\_ClosestBuilding\_BU:** Distance to the closest building.
  + **DIS\_Vegetation\_BU:** Distance to vegetation.
  + **DIS\_Trees\_BU:** Distance to trees.
  + **Pool\_AR\_PA:** Area of the pool.
  + **Pool\_Enclosure\_PA:** Presence of pool enclosure.
  + **Temporary\_Pool\_PA:** Presence of a temporary pool.
  + **Trampoline\_PA:** Presence of a trampoline.
  + **Yard\_Debris\_PA:** Presence of yard debris (None, Some, A lot).
  + **Pools:** Presence of pools (None, Outdoor, Indoor, Both).
  + **Tree\_Coverage:** Percentage of tree coverage on the property.
* **Neighborhood Information:**
  + **Population\_Density:** Population density of the area.
  + **Median\_Income:** Median income of the area.
  + **Education\_Level:** Education level in the area (High School, Bachelor, Master, PhD).
  + **Age\_of\_Housing:** Age of the housing (New, Mid-Age, Old).
* **Policyholder Information:**
  + **Claim\_History:** Number of past insurance claims.
  + **Policy\_Number:** Unique policy number.
  + **Policy\_Start\_Date:** Start date of the policy.
  + **Policy\_End\_Date:** End date of the policy.
  + **Coverage\_Type:** Type of coverage (Standard, Comprehensive, Premium).
* **Insurance Policy Details:**
  + **Policy\_Coverage\_Amount:** Coverage amount of the policy.
  + **Deductible:** Deductible amount.
  + **Premium\_Amount:** Premium amount.
* **Target Variable:**
  + **Category:** Risk category (Easy, Medium, Hard).

**Solution Workflow**

**1. Data Preprocessing**

**Steps:**

1. Load the dataset.
2. Handle missing values by forward filling.
3. Convert categorical columns to numerical using one-hot encoding.
4. Convert date columns to datetime and create a duration feature.
5. Drop original date columns.
6. Encode the target variable.
7. Split the data into training and testing sets.

python

Copy code

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import joblib

# Load the dataset

file\_path = 'underwriter.xlsx'

df = pd.read\_excel(file\_path)

# Display the first few rows of the dataframe

print(df.head())

# Check for missing values

missing\_values = df.isnull().sum()

print(missing\_values[missing\_values > 0])

# Fill or drop missing values as necessary

df.fillna(method='ffill', inplace=True)

# Convert categorical columns to numerical using one-hot encoding

categorical\_columns = df.select\_dtypes(include=['object']).columns

df = pd.get\_dummies(df, columns=categorical\_columns, drop\_first=True)

# Convert date columns to datetime and create duration feature

df['Policy\_Start\_Date'] = pd.to\_datetime(df['Policy\_Start\_Date'])

df['Policy\_End\_Date'] = pd.to\_datetime(df['Policy\_End\_Date'])

df['Policy\_Duration'] = (df['Policy\_End\_Date'] - df['Policy\_Start\_Date']).dt.days

# Drop original date columns

df.drop(columns=['Policy\_Start\_Date', 'Policy\_End\_Date'], inplace=True)

# Ensure 'Category' column exists

if 'Category' not in df.columns:

raise KeyError("The 'Category' column is missing from the dataset.")

# Encode target variable

label\_encoder = LabelEncoder()

df['Category'] = label\_encoder.fit\_transform(df['Category'])

# Separate features and target variable

X = df.drop(columns=['Category'])

y = df['Category']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

print(f'Training data shape: {X\_train.shape}')

print(f'Testing data shape: {X\_test.shape}')

**2. Model Training and Tuning**

**Logistic Regression**

python

Copy code

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Define the parameter grid

param\_grid = {

'C': [0.001, 0.01, 0.1, 1, 10],

'solver': ['lbfgs', 'liblinear']

}

# Perform Grid Search

grid\_search = GridSearchCV(LogisticRegression(max\_iter=1000, penalty='l2', random\_state=42), param\_grid, cv=5, n\_jobs=-1, verbose=2)

grid\_search.fit(X\_train, y\_train)

print(f'Best Parameters: {grid\_search.best\_params\_}')

print(f'Best Cross-Validation Accuracy: {grid\_search.best\_score\_:.2f}')

# Retrain the model with best parameters

best\_logreg\_model = grid\_search.best\_estimator\_

best\_logreg\_model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred\_best\_logreg = best\_logreg\_model.predict(X\_test)

# Evaluate the model

accuracy\_best\_logreg = accuracy\_score(y\_test, y\_pred\_best\_logreg)

print(f'Best Logistic Regression Model Accuracy: {accuracy\_best\_logreg:.2f}')

print("Best Logistic Regression Model Classification Report:")

print(classification\_report(y\_test, y\_pred\_best\_logreg))

print("Best Logistic Regression Model Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred\_best\_logreg))

**Results:**

* **Best Parameters:** {'C': 10, 'solver': 'liblinear'}
* **Best Cross-Validation Accuracy:** 0.97
* **Accuracy on Test Set:** 0.96

**Classification Report:**

markdown

Copy code

precision recall f1-score support

Easy 0.96 0.98 0.97 106

Hard 0.93 1.00 0.96 96

Medium 0.98 0.89 0.93 98

accuracy 0.96 300

macro avg 0.96 0.96 0.96 300

weighted avg 0.96 0.96 0.96 300

Bottom of Form