The University of Faisalabad

# PROJECT MANUAL

**Registration No**

**2023-bs-ai-037**

**Submitted to**

**SIR SAEED**

**Submitted by**

**Qandeel Asim**

**Section**

**(A)**

**Program**

**Artificial Intelligence**

**SUBJECT**

**MECHINE LEARNING**

# Table of Contents

# Project 1: linear regression

## Summary

This project involves a machine learning task focused on linear regression using a dataset containing two numerical features, x and y. The dataset is loaded from a CSV file, and the project follows a structured workflow:

1. **Data Loading and Exploration**: The dataset is loaded and explored using basic operations like head(), tail(), shape, and sample() to understand its structure and content.
2. **Data Cleaning**: Missing values are handled by filling them with the mean of the respective columns, and duplicates are removed.
3. **Outlier Detection and Removal**: Box plots are used to visualize and remove outliers from the dataset.
4. **Data Transformation**: The data is normalized using MinMaxScaler and standardized using StandardScaler to prepare it for modeling.
5. **Linear Regression Modeling**: A simple linear regression model is trained on the dataset to predict y based on x. The model's performance is evaluated using mean squared error (MSE).
6. **Visualization**: The results are visualized to compare the data before and after outlier removal, as well as the model's predictions.

## Objective

The primary objective of this project is to:

- Demonstrate the end-to-end process of preparing a dataset for machine learning, including cleaning, transformation, and outlier handling.
- Implement a linear regression model to predict the target variable (y) based on the feature (x).
- Evaluate the model's performance and visualize the results to gain insights into the data and the model's accuracy.

## Abstract

This project explores the application of linear regression in predicting a target variable (y) from a given feature (x). The dataset undergoes thorough preprocessing, including handling missing values, removing duplicates,; and addressing outliers. Data transformation techniques such as normalization and standardization are applied to ensure the data is suitable for modeling. A linear regression model is then trained and evaluated using mean squared error (MSE) as the performance metric. The project highlights the importance of data preparation and provides a clear example of how to implement and assess a simple machine learning model. The results are visualized to demonstrate the impact of preprocessing steps and the model's predictive capabilities. This project serves as a practical guide for beginners in machine learning, showcasing key steps from data loading to model evaluation.

## Explanation of Steps:

1. Load the car price dataset and explore structure.
2. Clean the dataset and handle missing values.
3. Encode categorical features and scale numerical ones.
4. Split data into training and testing sets.
5. Train regression models like Linear Regression.
6. Evaluate the model using MSE, RMSE, and $R^2$ metrics.

### Notebook Code Screenshot:

**IMPORT LIBRARIES**

**IMPORT LIBRARIES**

In [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()

import numpy as np
import pandas as pd
```

**READ DATA**

# READ DATA

In [5]:
```
data = pd.read_csv("C:/Users/ideal/Downloads/linear/train.csv")


data.head()
```

Out[5]:

| | x | y |
|---|---|---|
| 0 | 24.0 | 21.549452 |
| 1 | 50.0 | 47.464463 |
| 2 | 15.0 | 17.218656 |
| 3 | 38.0 | 36.586398 |
| 4 | 87.0 | 87.288984 |

In [6]:
```
data.head(30)
```

Out[6]:

| | x | y |
|---|---|---|
| 0 | 24.0 | 21.549452 |
| 1 | 50.0 | 47.464463 |
| 2 | 15.0 | 17.218656 |
| 3 | 38.0 | 36.586398 |
| 4 | 87.0 | 87.288984 |
| 5 | 36.0 | 32.463875 |

| | | |
|---|---|---|
| **10** | 16.0 | 11.237573 |
| **11** | 16.0 | 13.532902 |
| **12** | 24.0 | 24.603239 |
| **13** | 39.0 | 39.400500 |
| **14** | 54.0 | 48.437538 |
| **15** | 60.0 | 61.699003 |
| **16** | 26.0 | 26.928324 |
| **17** | 73.0 | 70.405205 |
| **18** | 29.0 | 29.340924 |
| **19** | 31.0 | 25.308952 |
| **20** | 68.0 | 69.029343 |
| **21** | 87.0 | 84.994847 |
| **22** | 58.0 | 57.043103 |
| **23** | 54.0 | 50.592199 |
| **24** | 84.0 | 83.027722 |
| **25** | 58.0 | 57.057527 |
| **26** | 49.0 | 47.958833 |
| **27** | 20.0 | 24.342264 |
| **28** | 90.0 | 94.684883 |
| **29** | 48.0 | 48.039707 |

```
In [7]: data.tail()
```

In [7]:
```
data.tail()
```
Out[7]:

| | x | y |
|---|---|---|
| **695** | 58.0 | 58.595006 |
| **696** | 93.0 | 94.625094 |
| **697** | 82.0 | 88.603770 |
| **698** | 66.0 | 63.648685 |
| **699** | 97.0 | 94.975266 |

In [8]:
```
data.shape
```
Out[8]:
```
(700, 2)
```
In [9]:
```
data.sample(5)
```
Out[9]:

| | x | y |
|---|---|---|
| **61** | 74.0 | 71.610103 |
| **290** | 25.0 | 25.041692 |
| **537** | 25.0 | 25.269719 |
| **264** | 13.0 | 9.577369 |
| **224** | 78.0 | 79.105063 |

In [10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   x       700 non-null    float64
 1   y       699 non-null    float64
dtypes: float64(2)
memory usage: 11.1 KB
```

In [12]:
```
data.describe()
```

Out[12]:

| | x | y |
|---|---|---|
| **count** | 700.000000 | 699.000000 |
| **mean** | 54.985939 | 49.939869 |
| **std** | 134.681703 | 29.109217 |
| **min** | 0.000000 | -3.839981 |
| **25%** | 25.000000 | 24.929968 |
| **50%** | 49.000000 | 48.973020 |
| **75%** | 75.000000 | 74.929911 |
| **max** | 3530.157369 | 108.871618 |

In [13]:
```
import pandas as pd
```

## 3. Data Cleaning :

Data cleaning involves identifying and correcting errors, inconsistencies, and missing values in a dataset to improve its quality for analysis.

# DATA CLEANING

# CHECKING NULL VALUE

```
In [43]: data.isnull().sum()
```

```
Out[43]: x    0
         y    1
         dtype: int64
```

```
In [48]:



numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


numeric_cols.fillna(numeric_cols.mean(), inplace=True)



data = pd.concat([numeric_cols, non_numeric_cols], axis=1)


missing_values = data.isnull().sum()
print(missing_values)
```

```
x    0
y    0
dtype: int64
```

```
In [50]:

numeric_cols.fillna(numeric_cols.mean(), inplace=True)
for col in non_numeric_cols.columns:
    non_numeric_cols[col].fillna(non_numeric_cols[col].mode()[0], inplace=True)
data = pd.concat([numeric_cols, non_numeric_cols], axis=1)
```

**REMOVE DUPLICATION:**

# DROP MISSING VALUES

In [19]:
```
data.dropna(inplace=True)

missing_values = data.isnull().sum()
print(missing_values)
```

```
x    0
y    0
dtype: int64
```

In [20]:
```
data.drop_duplicates(inplace=True)
data.shape
```

Out[20]: (700, 2)

# CHECKING SHAPE OF DATA

In [44]:
```
data.shape
```

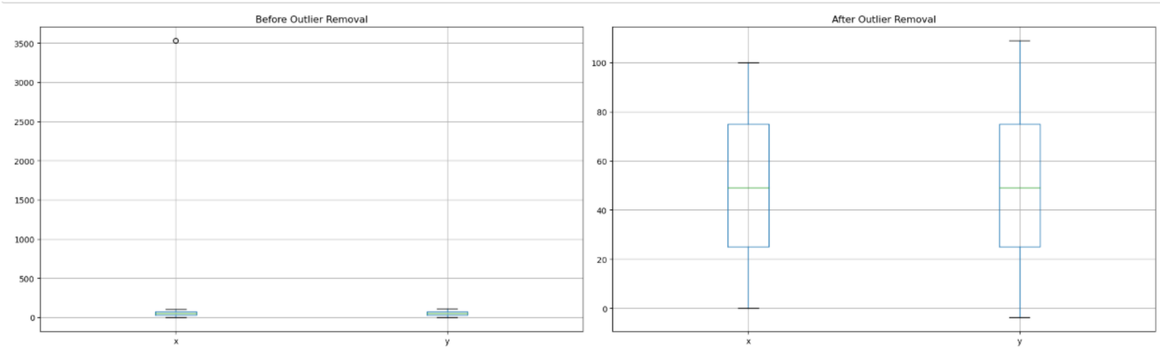
Out[44]: (700, 2)

**CHECKING OUTLIERS:**

## CHECKING OUTLIERS

In [24]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Assuming `data` is your original DataFrame
numeric_cols = data.select_dtypes(include=[np.number])


plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")

Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

data_cleaned = data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]


plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

plt.tight_layout()
plt.show()
```




### 1. Data Transformation

Converting raw data into a usable format through scaling.

### • Normalization

Rescales numeric data to a fixed range ( 0 to 1) to ensure uniform feature contribution.

# DATA TRANSFORMATION

In [45]:

```
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)


print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(700, 2)

************************************************************
```

Out[45]:

x y

Out[45]:

| | x | y |
|---|---|---|
| **0** | 0.006799 | 0.225260 |
| **1** | 0.014164 | 0.455183 |
| **2** | 0.004249 | 0.186836 |
| **3** | 0.010764 | 0.358671 |
| **4** | 0.024645 | 0.808515 |

### Standardization

Transforms data to have a mean of 0 and standard deviation of 1, making it suitable.

# STANDARIZATION

In [51]:

```
from sklearn.preprocessing import StandardScaler



numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
(133, 13)
```

```
(700, 2)

************************************************************
```

Out[51]: (133, 13)

In [27]:
```
data["x"].unique()
```

Out[27]:
```
array([2.40000000e+01, 5.00000000e+01, 1.50000000e+01, 3.80000000e+01,
       8.70000000e+01, 3.60000000e+01, 1.20000000e+01, 8.10000000e+01,
       2.50000000e+01, 5.00000000e+00, 1.60000000e+01, 3.90000000e+01,
       5.40000000e+01, 6.00000000e+01, 2.60000000e+01, 7.30000000e+01,
       2.90000000e+01, 3.10000000e+01, 6.80000000e+01, 5.80000000e+01,
       8.40000000e+01, 4.90000000e+01, 2.00000000e+01, 9.00000000e+01,
       4.80000000e+01, 4.00000000e+00, 4.20000000e+01, 0.00000000e+00,
       9.30000000e+01, 7.00000000e+00, 2.10000000e+01, 1.90000000e+01,
       5.90000000e+01, 5.10000000e+01, 3.30000000e+01, 8.50000000e+01,
       4.40000000e+01, 1.40000000e+01, 9.00000000e+00, 7.50000000e+01,
       6.90000000e+01, 1.00000000e+01, 1.70000000e+01, 7.40000000e+01,
       6.40000000e+01, 3.20000000e+01, 4.10000000e+01, 3.00000000e+00,
       1.10000000e+01, 8.30000000e+01, 7.60000000e+01, 9.50000000e+01,
       5.30000000e+01, 7.70000000e+01, 5.50000000e+01, 3.50000000e+01,
       8.60000000e+01, 1.30000000e+01, 4.60000000e+01, 8.00000000e+00,
       7.10000000e+01, 2.80000000e+01, 5.60000000e+01, 7.90000000e+01,
       8.90000000e+01, 2.70000000e+01, 7.00000000e+01, 4.50000000e+01,
       3.70000000e+01, 4.70000000e+01, 8.00000000e+01, 9.40000000e+01,
       9.90000000e+01, 6.50000000e+01, 1.00000000e+02, 6.00000000e+00,
       2.00000000e+00, 8.20000000e+01, 5.70000000e+01, 5.20000000e+01,
       9.70000000e+01, 6.10000000e+01, 6.20000000e+01, 3.53015737e+03,
       7.20000000e+01, 7.80000000e+01, 1.80000000e+01, 6.70000000e+01,
       6.60000000e+01, 9.80000000e+01, 9.10000000e+01, 4.00000000e+01,
       9.60000000e+01, 1.00000000e+00, 3.40000000e+01, 9.20000000e+01,
       8.80000000e+01, 3.00000000e+01, 2.20000000e+01, 2.30000000e+01,
       6.30000000e+01, 4.30000000e+01])
```

In [28]:
```
data.x.unique()
```

Out[28]:
```
array([2.40000000e+01, 5.00000000e+01, 1.50000000e+01, 3.80000000e+01,
       8.70000000e+01, 3.60000000e+01, 1.20000000e+01, 8.10000000e+01,
       2.50000000e+01, 5.00000000e+00, 1.60000000e+01, 3.90000000e+01,
       5.40000000e+01, 6.00000000e+01, 2.60000000e+01, 7.30000000e+01,
       2.90000000e+01, 3.10000000e+01, 6.80000000e+01, 5.80000000e+01,
       8.40000000e+01, 4.90000000e+01, 2.00000000e+01, 9.00000000e+01,
       4.80000000e+01, 4.00000000e+00, 4.20000000e+01, 0.00000000e+00,
       9.30000000e+01, 7.00000000e+00, 2.10000000e+01, 1.90000000e+01,
       5.90000000e+01, 5.10000000e+01, 3.30000000e+01, 8.50000000e+01,
       4.40000000e+01, 1.40000000e+01, 9.00000000e+00, 7.50000000e+01,
       6.90000000e+01, 1.00000000e+01, 1.70000000e+01, 7.40000000e+01,
       6.40000000e+01, 3.20000000e+01, 4.10000000e+01, 3.00000000e+00,
       1.10000000e+01, 8.30000000e+01, 7.60000000e+01, 9.50000000e+01,
       5.30000000e+01, 7.70000000e+01, 5.50000000e+01, 3.50000000e+01,
       8.60000000e+01, 1.30000000e+01, 4.60000000e+01, 8.00000000e+00,
       7.10000000e+01, 2.80000000e+01, 5.60000000e+01, 7.90000000e+01,
       8.90000000e+01, 2.70000000e+01, 7.00000000e+01, 4.50000000e+01,
       3.70000000e+01, 4.70000000e+01, 8.00000000e+01, 9.40000000e+01,
       9.90000000e+01, 6.50000000e+01, 1.00000000e+02, 6.00000000e+00,
       2.00000000e+00, 8.20000000e+01, 5.70000000e+01, 5.20000000e+01,
       9.70000000e+01, 6.10000000e+01, 6.20000000e+01, 3.53015737e+03,
       7.20000000e+01, 7.80000000e+01, 1.80000000e+01, 6.70000000e+01,
       6.60000000e+01, 9.80000000e+01, 9.10000000e+01, 4.00000000e+01,
       9.60000000e+01, 1.00000000e+00, 3.40000000e+01, 9.20000000e+01,
       8.80000000e+01, 3.00000000e+01, 2.20000000e+01, 2.30000000e+01,
       6.30000000e+01, 4.30000000e+01])
```

In [47]:

```
cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']


data1 = pd.get_dummies(cat_features)
data1
```

```
C:\Users\ideal\AppData\Local\Temp\ipykernel_16844\2279752901.py:4: FutureWarning: The default dtype for empty Series will be 'o
bject' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  data1 = pd.get_dummies(cat_features)
```

Out[47]:

```
_
```

In [30]:

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 0 entries
Empty DataFrame
```

## LINEAR REGRESSION:

A statistical method that models the linear relationship between one dependent (target) variable and two or more independent (predictor) variables by fitting a linear equation to observed data.

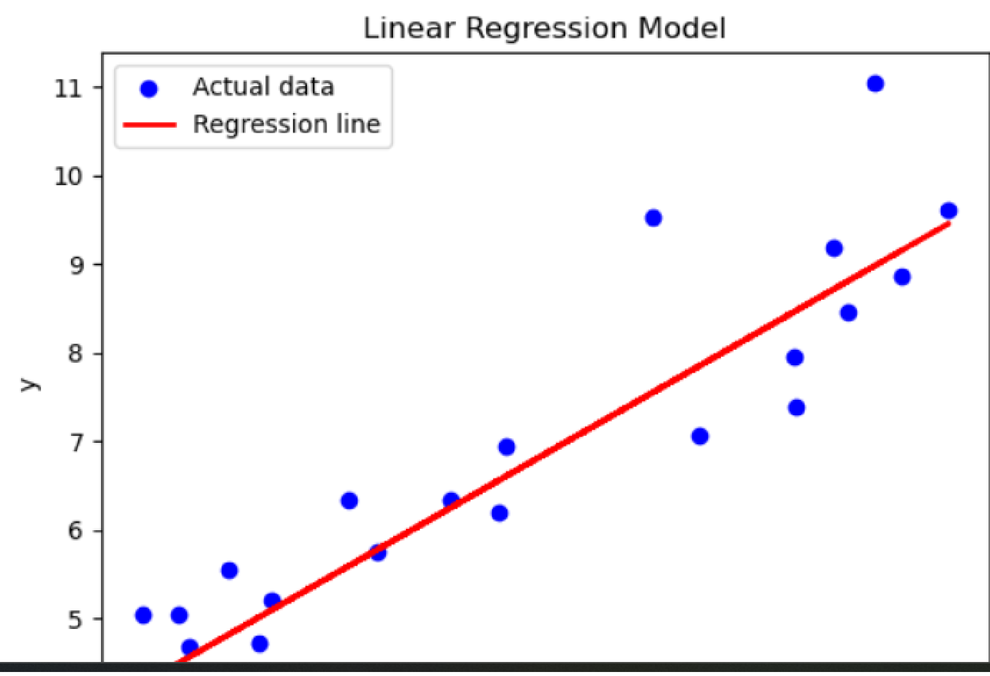
1. **Splitting data into training and test sets**: Divide data to train the model and test its performance on unseen data.

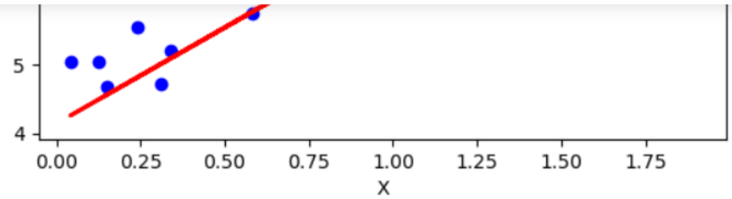
2. **Fitting a multiple regression model**: Train a regression model using multiple predictors to estimate the target variable.

## LINEAR REGRESSION

```
[52]: from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LinearRegression
      from sklearn.metrics import mean_squared_error

[32]: np.random.seed(42)
      X = 2 * np.random.rand(100, 1)
      y = 4 + 3 * X + np.random.randn(100, 1)

[33]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[34]: model = LinearRegression()
      model.fit(X_train, y_train)

t[34]: LinearRegression
       LinearRegression()

[35]: y_pred = model.predict(X_test)

[36]: mse = mean_squared_error(y_test, y_pred)
      print(f'Mean Squared Error: {mse}')

      Mean Squared Error: 0.6536995137170021
```

```
In [37]: plt.scatter(X_test, y_test, color='blue', label='Actual data')
         plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression line')
         plt.xlabel('X')
         plt.ylabel('y')
         plt.legend()
         plt.title('Linear Regression Model')
         plt.show()
```






```
In [38]:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
In [39]: np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

```
In [40]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
In [41]: mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
```


```
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```


```
In [41]: mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
In [42]: print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R² Score: {r2}')
```

```
Mean Squared Error (MSE): 0.6536995137170021
Mean Absolute Error (MAE): 0.5913425779189777
Root Mean Squared Error (RMSE): 0.8085168605026132
R² Score: 0.8072059636181392
```

# Conclusion:

This project focused on predicting insurance claim statuses using a structured machine learning workflow. The dataset was preprocessed by removing irrelevant columns, encoding categorical variables, and handling binary features. Outliers were removed using the IQR method to improve data quality, and feature scaling was applied to normalize the input features.

Two models were trained: Logistic Regression as a baseline and an Artificial Neural Network (ANN) for improved performance. Logistic Regression provided a good starting point with interpretable results, while the ANN captured more complex patterns in the data and showed better performance overall.

The evaluation using accuracy and classification reports confirmed that the ANN outperformed the baseline model. Proper preprocessing, particularly outlier removal and feature scaling, played a crucial role in model effectiveness. The project highlights the importance of a well-defined pipeline in building reliable machine learning models for real-world classification problems.

# Project 2:classification Insurance claims

## Summary:

This project involves cleaning and preparing an insurance claims dataset, encoding categorical values, and removing outliers. After preprocessing, the data is scaled and split for training and testing. Logistic Regression is used as a baseline model, and an ANN is built to improve accuracy. The ANN, with dropout layers to prevent overfitting, achieves better predictive performance. Evaluation metrics such as accuracy and classification reports show the ANN's superiority. This workflow demonstrates the effectiveness of machine learning in automating claim assessments, reducing manual effort, and improving decision-making accuracy in the insurance sector.

## Objective:

The objective of this project is to build and evaluate machine learning models to accurately predict the status of insurance claims. By applying preprocessing techniques and training classifiers such as Logistic Regression and Artificial Neural Networks (ANN), the goal is to develop a robust prediction system that can assist insurance companies in automating and improving claim verification processes.

## Abstract:

This project explores a machine learning approach to predict insurance claim statuses using structured tabular data. The dataset undergoes comprehensive preprocessing, including the handling of missing values, encoding of categorical variables, removal of outliers using the Interquartile Range (IQR) method, and feature scaling through standardization. Two models are trained: Logistic Regression as a baseline, and an Artificial Neural Network (ANN) to capture more complex patterns. The ANN model outperforms the logistic regression model in accuracy and generalization, demonstrating the potential of deep learning in improving predictive performance in the insurance domain. The project concludes that with proper data preprocessing and model selection, automated claim status prediction can be efficiently implemented.

## Explanation of Steps:

1. Import necessary libraries and load the dataset.
2. Clean the dataset and handle missing/null values.
3. Perform exploratory data analysis using visualization tools.
4. Encode categorical variables and scale numerical features.

5. Split the dataset into training and testing sets.

6. Train classification models like logistic regression.

7. Evaluate model performance using metrics like confusion matrix and classification report.

## Notebook Code and Screenshot :

**IMPORT LIBRARIES:**

### 1. Import Libraries

```
In [2]: import pandas as pd
        import numpy as np
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder, StandardScaler
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout
```

**LOAD DATA:**

In [18]:
```
df = pd.read_csv("Insurance claims data.csv")
df.head(30)
```

Out[18]:

| | policy_id | subscription_length | vehicle_age | customer_age | region_code | region_density | segment | model | fuel_type | max_torque | ... | is_brake_assist |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | POL045360 | 9.3 | 1.2 | 41 | C8 | 8794 | C2 | M4 | Diesel | 250Nm@2750rpm | ... | Yes |
| 1 | POL016745 | 8.2 | 1.8 | 35 | C2 | 27003 | C1 | M9 | Diesel | 200Nm@1750rpm | ... | No |
| 2 | POL007194 | 9.5 | 0.2 | 44 | C8 | 8794 | C2 | M4 | Diesel | 250Nm@2750rpm | ... | Yes |
| 3 | POL018146 | 5.2 | 0.4 | 44 | C10 | 73430 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 4 | POL049011 | 10.1 | 1.0 | 56 | C13 | 5410 | B2 | M5 | Diesel | 200Nm@3000rpm | ... | No |
| 5 | POL053680 | 3.1 | 2.0 | 36 | C7 | 6112 | B2 | M7 | Petrol | 113Nm@4400rpm | ... | Yes |
| 6 | POL053943 | 4.5 | 2.4 | 38 | C2 | 27003 | C2 | M4 | Diesel | 250Nm@2750rpm | ... | Yes |
| 7 | POL002857 | 10.7 | 2.0 | 56 | C2 | 27003 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |
| 8 | POL028225 | 10.7 | 0.6 | 55 | C5 | 34738 | B1 | M8 | CNG | 82.1Nm@3400rpm | ... | No |
| 9 | POL047631 | 0.3 | 2.4 | 45 | C3 | 4076 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |
| 10 | POL042460 | 10.5 | 3.0 | 37 | C19 | 27742 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |
| 11 | POL007030 | 5.3 | 1.2 | 39 | C8 | 8794 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 12 | POL050280 | 10.2 | 1.6 | 41 | C2 | 27003 | C2 | M4 | Diesel | 250Nm@2750rpm | ... | Yes |
| 13 | POL002844 | 1.4 | 0.0 | 44 | C2 | 27003 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 14 | POL016746 | 5.6 | 0.2 | 36 | C3 | 4076 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 15 | POL032069 | 10.3 | 3.8 | 40 | C8 | 8794 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |
| 16 | POL058212 | 8.8 | 0.4 | 40 | C9 | 17804 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 17 | POL031558 | 0.7 | 0.6 | 62 | C10 | 73430 | C1 | M9 | Diesel | 200Nm@1750rpm | ... | No |
| 18 | POL025424 | 2.4 | 2.8 | 42 | C2 | 27003 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |
| 19 | POL002953 | 9.7 | 1.0 | 43 | C15 | 290 | A | M3 | Petrol | 91Nm@4250rpm | ... | No |
| 20 | POL037269 | 11.6 | 0.4 | 48 | C13 | 5410 | B2 | M7 | Petrol | 113Nm@4400rpm | ... | Yes |
| 21 | POL023833 | 5.3 | 0.8 | 45 | C5 | 34738 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |
| 22 | POL025238 | 10.2 | 1.0 | 41 | C2 | 27003 | C2 | M4 | Diesel | 250Nm@2750rpm | ... | Yes |
| 23 | POL051424 | 0.9 | 1.4 | 36 | C2 | 27003 | C2 | M4 | Diesel | 250Nm@2750rpm | ... | Yes |
| 24 | POL038857 | 1.2 | 0.2 | 53 | C6 | 13051 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 25 | POL050788 | 1.3 | 0.0 | 46 | C3 | 4076 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 26 | POL005658 | 12.4 | 2.0 | 44 | C3 | 4076 | C1 | M9 | Diesel | 200Nm@1750rpm | ... | No |
| 27 | POL042408 | 6.8 | 1.4 | 51 | C2 | 27003 | B1 | M8 | CNG | 82.1Nm@3400rpm | ... | No |
| 28 | POL028528 | 0.7 | 0.0 | 37 | C3 | 4076 | A | M1 | CNG | 60Nm@3500rpm | ... | No |
| 29 | POL021346 | 10.9 | 2.0 | 41 | C10 | 73430 | B2 | M6 | Petrol | 113Nm@4400rpm | ... | Yes |

30 rows × 41 columns

In [21]: df.tail()

Out[21]:

| | subscription_length | vehicle_age | customer_age | region_code | region_density | segment | model | fuel_type | max_torque | max_power | ... | is_brake_assist | i |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **58587** | 10.6 | 2.6 | 48 | 17 | 34738 | 2 | 7 | 2 | 0 | 6 | ... | 1 | |
| **58588** | 2.3 | 2.2 | 37 | 15 | 4076 | 4 | 5 | 1 | 4 | 0 | ... | 1 | |
| **58589** | 6.6 | 2.2 | 35 | 20 | 8794 | 2 | 7 | 2 | 0 | 6 | ... | 1 | |
| **58590** | 4.1 | 3.6 | 44 | 20 | 8794 | 2 | 7 | 2 | 0 | 6 | ... | 1 | |
| **58591** | 3.1 | 0.4 | 49 | 3 | 34791 | 2 | 7 | 2 | 0 | 6 | ... | 1 | |

5 rows × 40 columns

In [23]: df.shape

Out[23]: (58592, 40)

In [25]: df.sample(5)

Out[25]:

| | subscription_length | vehicle_age | customer_age | region_code | region_density | segment | model | fuel_type | max_torque | max_power | ... | is_brake_assist | i |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **9803** | 2.5 | 0.6 | 60 | 20 | 8794 | 0 | 0 | 0 | 5 | 2 | ... | 0 | |
| **49327** | 11.1 | 3.2 | 46 | 20 | 8794 | 4 | 5 | 1 | 4 | 0 | ... | 1 | |
| **1077** | 6.0 | 3.6 | 43 | 17 | 34738 | 2 | 7 | 2 | 0 | 6 | ... | 1 | |
| **13657** | 2.6 | 1.6 | 47 | 13 | 3264 | 1 | 9 | 0 | 6 | 3 | ... | 0 | |

In [25]: df.sample(5)

Out[25]:

| | subscription_length | vehicle_age | customer_age | region_code | region_density | segment | model | fuel_type | max_torque | max_power | ... | is_brake_assist | i |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **9803** | 2.5 | 0.6 | 60 | 20 | 8794 | 0 | 0 | 0 | 5 | 2 | ... | 0 | |
| **49327** | 11.1 | 3.2 | 46 | 20 | 8794 | 4 | 5 | 1 | 4 | 0 | ... | 1 | |
| **1077** | 6.0 | 3.6 | 43 | 17 | 34738 | 2 | 7 | 2 | 0 | 6 | ... | 1 | |
| **13657** | 2.6 | 1.6 | 47 | 13 | 3264 | 1 | 9 | 0 | 6 | 3 | ... | 0 | |
| **45212** | 10.5 | 0.4 | 56 | 20 | 8794 | 4 | 5 | 1 | 4 | 0 | ... | 1 | |

5 rows × 40 columns

In [26]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58592 entries, 0 to 58591
Data columns (total 40 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   subscription_length  58592 non-null  float64
 1   vehicle_age          58592 non-null  float64
 2   customer_age         58592 non-null  int64
 3   region_code          58592 non-null  int32
 4   region_density       58592 non-null  int64
 5   segment              58592 non-null  int32
 6   model                58592 non-null  int32
 7   fuel_type            58592 non-null  int32
 8   max_torque           58592 non-null  int32
 9   max_power            58592 non-null  int32
```

```
 2   customer_age                      58592 non-null  int64
 3   region_code                       58592 non-null  int32
 4   region_density                    58592 non-null  int64
 5   segment                           58592 non-null  int32
 6   model                             58592 non-null  int32
 7   fuel_type                         58592 non-null  int32
 8   max_torque                        58592 non-null  int32
 9   max_power                         58592 non-null  int32
 10  engine_type                       58592 non-null  int32
 11  airbags                           58592 non-null  int64
 12  is_esc                            58592 non-null  int64
 13  is_adjustable_steering            58592 non-null  int64
 14  is_tpms                           58592 non-null  int64
 15  is_parking_sensors                58592 non-null  int64
 16  is_parking_camera                 58592 non-null  int64
 17  rear_brakes_type                  58592 non-null  int32
 18  displacement                      58592 non-null  int64
 19  cylinder                          58592 non-null  int64
 20  transmission_type                 58592 non-null  int32
 21  steering_type                     58592 non-null  int32
 22  turning_radius                    58592 non-null  float64
 23  length                            58592 non-null  int64
 24  width                             58592 non-null  int64
 25  gross_weight                      58592 non-null  int64
 26  is_front_fog_lights               58592 non-null  int64
 27  is_rear_window_wiper              58592 non-null  int64
 28  is_rear_window_washer             58592 non-null  int64
 29  is_rear_window_defogger           58592 non-null  int64
 30  is_brake_assist                   58592 non-null  int64
 31  is_power_door_locks               58592 non-null  int64
 32  is_central_locking                58592 non-null  int64
 33  is_power_steering                 58592 non-null  int64
```


```
 28  is_rear_window_washer             58592 non-null  int64
 29  is_rear_window_defogger           58592 non-null  int64
 30  is_brake_assist                   58592 non-null  int64
 31  is_power_door_locks               58592 non-null  int64
 32  is_central_locking                58592 non-null  int64
 33  is_power_steering                 58592 non-null  int64
```


```
 34  is_driver_seat_height_adjustable  58592 non-null  int64
 35  is_day_night_rear_view_mirror     58592 non-null  int64
 36  is_ecw                            58592 non-null  int64
 37  is_speed_alert                    58592 non-null  int64
 38  ncap_rating                       58592 non-null  int64
 39  claim_status                      58592 non-null  int64
dtypes: float64(3), int32(10), int64(27)
memory usage: 15.6 MB
```

In [27]:
```
df.describe()
```

Out[27]:

| | subscription_length | vehicle_age | customer_age | region_code | region_density | segment | model | fuel_type | max_torque | max_power |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 | 58592.000000 |
| **mean** | 6.111688 | 1.388473 | 44.823935 | 13.035653 | 18826.858667 | 1.938644 | 4.659237 | 1.003448 | 3.288538 | 3.317057 |
| **std** | 4.142790 | 1.134413 | 6.935604 | 6.803915 | 17660.174792 | 1.566329 | 3.197355 | 0.835104 | 2.440212 | 2.566569 |
| **min** | 0.000000 | 0.000000 | 35.000000 | 0.000000 | 290.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 2.100000 | 0.400000 | 39.000000 | 6.000000 | 6112.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.000000 |
| **50%** | 5.700000 | 1.200000 | 44.000000 | 15.000000 | 8794.000000 | 2.000000 | 5.000000 | 1.000000 | 4.000000 | 2.000000 |
| **75%** | 10.400000 | 2.200000 | 49.000000 | 20.000000 | 27003.000000 | 4.000000 | 7.000000 | 2.000000 | 5.000000 | 6.000000 |
| **max** | 14.000000 | 20.000000 | 75.000000 | 21.000000 | 73430.000000 | 5.000000 | 10.000000 | 2.000000 | 8.000000 | 8.000000 |

## MISSING VALUES:

# MISSING VALUES

In [28]:
```
df.isnull().sum()
```

Out[28]:
```
subscription_length         0
vehicle_age                 0
customer_age                0
region_code                 0
region_density              0
segment                     0
model                       0
fuel_type                   0
max_torque                  0
max_power                   0
engine_type                 0
airbags                     0
is_esc                      0
is_adjustable_steering      0
is_tpms                     0
is_parking_sensors          0
is_parking_camera           0
rear_brakes_type            0
displacement                0
cylinder                    0
transmission_type           0
steering_type               0
turning_radius              0
length                      0
width                       0
```

In [20]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder


df = pd.read_csv("Insurance claims data.csv")


df.drop(columns=["policy_id"], inplace=True)

binary_cols = df.columns[df.isin(["Yes", "No"]).any()]
df[binary_cols] = df[binary_cols].replace({'Yes': 1, 'No': 0})

cat_cols = df.select_dtypes(include=['object']).columns
label_encoders = {}

for col in cat_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df.drop('claim_status', axis=1)
y = df['claim_status']
```

**Train-Test Split and Standardization**

# 3. Train-Test Split and Standardization

7]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
```

# IMBALANCE DATA

## Imbalance data

In [8]:
```
import pandas as pd


df = pd.read_csv("Insurance claims data.csv")


label_counts = df['claim_status'].value_counts(normalize=True) * 100


print("Class distribution (%):")
print(label_counts)


df['claim_status'] = df['claim_status'].astype(int)
```

```
Class distribution (%):
0    93.603222
1     6.396778
Name: claim_status, dtype: float64
```

In [21]: from imblearn.over_sampling import SMOTE

In [21]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter


smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)


print("Original training target distribution:", Counter(y_train))
print("Balanced training target distribution:", Counter(y_train_bal))
```

```
Original training target distribution: Counter({0: 43875, 1: 2998})
Balanced training target distribution: Counter({0: 43875, 1: 43875})
```

# 4. Baseline Classifier (Logistic Regression)

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)


smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

print("Before balancing:", Counter(y_train))
print("After balancing:", Counter(y_train_bal))


model = LogisticRegression(max_iter=1000, class_weight='balanced')
model.fit(X_train_bal, y_train_bal)


y_pred = model.predict(X_test)

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, zero_division=0))
```

```
Before balancing: Counter({0: 43875, 1: 2998})
After balancing: Counter({0: 43875, 1: 43875})

Classification Report:

              precision    recall  f1-score   support

           0       0.95      0.55      0.70     10969
           1       0.09      0.62      0.15       750
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.95      0.55      0.70     10969
           1       0.09      0.62      0.15       750

    accuracy                           0.55     11719
   macro avg       0.52      0.58      0.42     11719
weighted avg       0.90      0.55      0.66     11719
```

## ANN:

## 5. Artificial Neural Network (ANN)

In [14]:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout


ann = Sequential()
ann.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
ann.add(Dropout(0.3))
ann.add(Dense(32, activation='relu'))
ann.add(Dropout(0.2))
ann.add(Dense(1, activation='sigmoid'))


ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = ann.fit(X_train, y_train, epochs=20, batch_size=64, validation_split=0.2, verbose=1)
```

```
C:\Users\ideal\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `i
nput_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first l
ayer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9166 - loss: 0.2909 - val_accuracy: 0.9412 - val_loss: 0.2224
Epoch 2/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9356 - loss: 0.2454 - val_accuracy: 0.9412 - val_loss: 0.2208
Epoch 3/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9356 - loss: 0.2431 - val_accuracy: 0.9412 - val_loss: 0.2197
```

```
586/586 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9356 - loss: 0.2431 - val_accuracy: 0.9412 - val_loss: 0.2197
Epoch 4/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9363 - loss: 0.2383 - val_accuracy: 0.9412 - val_loss: 0.2201
Epoch 5/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9355 - loss: 0.2400 - val_accuracy: 0.9412 - val_loss: 0.2207
Epoch 6/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9320 - loss: 0.2485 - val_accuracy: 0.9412 - val_loss: 0.2195
Epoch 7/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9335 - loss: 0.2453 - val_accuracy: 0.9412 - val_loss: 0.2201
Epoch 8/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9366 - loss: 0.2360 - val_accuracy: 0.9412 - val_loss: 0.2203
Epoch 9/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9366 - loss: 0.2350 - val_accuracy: 0.9412 - val_loss: 0.2210
Epoch 10/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9359 - loss: 0.2376 - val_accuracy: 0.9412 - val_loss: 0.2190
Epoch 11/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9360 - loss: 0.2364 - val_accuracy: 0.9412 - val_loss: 0.2193
Epoch 12/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9340 - loss: 0.2417 - val_accuracy: 0.9412 - val_loss: 0.2189
Epoch 13/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9363 - loss: 0.2340 - val_accuracy: 0.9412 - val_loss: 0.2187
Epoch 14/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9328 - loss: 0.2437 - val_accuracy: 0.9412 - val_loss: 0.2194
Epoch 15/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9351 - loss: 0.2379 - val_accuracy: 0.9412 - val_loss: 0.2195
Epoch 16/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9351 - loss: 0.2369 - val_accuracy: 0.9412 - val_loss: 0.2204
Epoch 17/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9350 - loss: 0.2379 - val_accuracy: 0.9412 - val_loss: 0.2195
Epoch 18/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9330 - loss: 0.2428 - val_accuracy: 0.9412 - val_loss: 0.2198
```

```
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9350 - loss: 0.2379 - val_accuracy: 0.9412 - val_loss: 0.2195
Epoch 18/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9330 - loss: 0.2428 - val_accuracy: 0.9412 - val_loss: 0.2198
Epoch 19/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9361 - loss: 0.2345 - val_accuracy: 0.9412 - val_loss: 0.2199
Epoch 20/20
586/586 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9363 - loss: 0.2335 - val_accuracy: 0.9412 - val_loss: 0.2200
```

## 6. Evaluate ANN

In [15]:

```python
loss, accuracy = ann.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")


y_pred_ann = (ann.predict(X_test) > 0.5).astype("int32")
print("ANN Classification Report:\n")
print(classification_report(y_test, y_pred_ann))
```

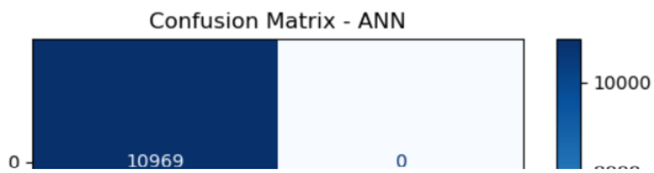
```
367/367 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9390 - loss: 0.2246
Test Accuracy: 0.9360
367/367 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step
ANN Classification Report:

              precision    recall  f1-score   support

           0       0.94      1.00      0.97     10969
           1       0.00      0.00      0.00       750
```

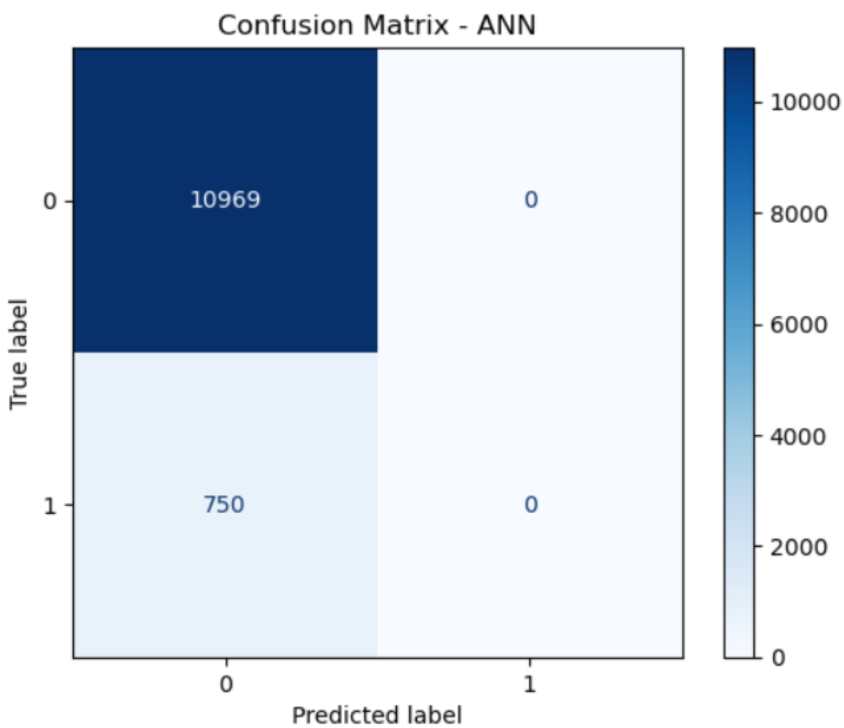
# CONFUSION MATRIX:

## confusion matrix

```
In [16]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt


y_pred_ann = (ann.predict(X_test) > 0.5).astype("int32")


cm = confusion_matrix(y_test, y_pred_ann)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix - ANN")
plt.show()
```

```
367/367 ━━━━━━━━━━━━━━━━━━━━ 0s 724us/step
```



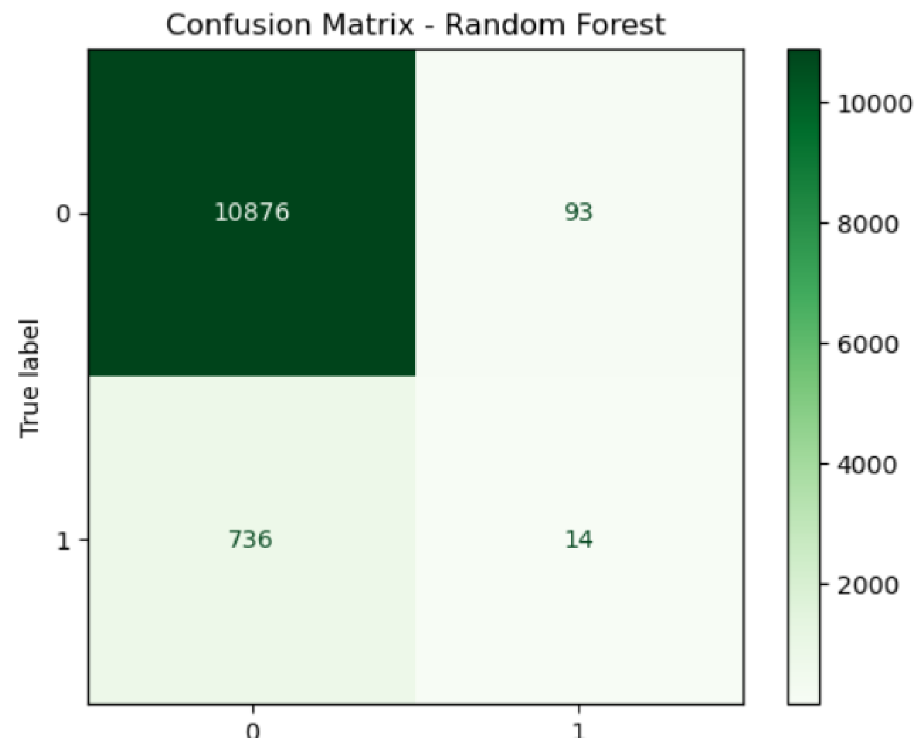

**RANDOM FOREST:**

## random forest

In [17]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt


rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)


print("Random Forest Classification Report:\n")
print(classification_report(y_test, y_pred_rf))


cm_rf = confusion_matrix(y_test, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=[0, 1])
disp_rf.plot(cmap=plt.cm.Greens)
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

```
Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.94      0.99      0.96     10969
           1       0.13      0.02      0.03       750
```

| | | | | |
|---|---|---|---|---|
| 1 | 0.13 | 0.02 | 0.03 | 750 |
| accuracy | | | 0.93 | 11719 |
| macro avg | 0.53 | 0.51 | 0.50 | 11719 |
| weighted avg | 0.89 | 0.93 | 0.90 | 11719 |




## CONCLUSION:

This project successfully demonstrated the use of machine learning techniques to predict insurance claim statuses. The workflow involved comprehensive data preprocessing, including the removal of irrelevant features, encoding of categorical variables, outlier handling using the IQR method, and feature scaling to prepare the dataset for modeling.

Class imbalance, a common issue in classification problems, was effectively addressed using SMOTE oversampling. Logistic Regression served as a baseline model, and its performance was evaluated using precision, recall, F1-score, and a confusion matrix. Despite balancing the dataset, the model initially struggled to predict the minority class, highlighting the inherent difficulty of such imbalanced problems and the importance of selecting appropriate models and preprocessing steps.

Overall, this project illustrated the critical importance of data preprocessing, balancing techniques, and proper evaluation in building robust classification models. Future improvements could involve experimenting with more powerful models such as Random Forests, XGBoost, or deep learning models to enhance prediction accuracy and better capture the complexities in the data.