# Credit Score Analysis

Group 7

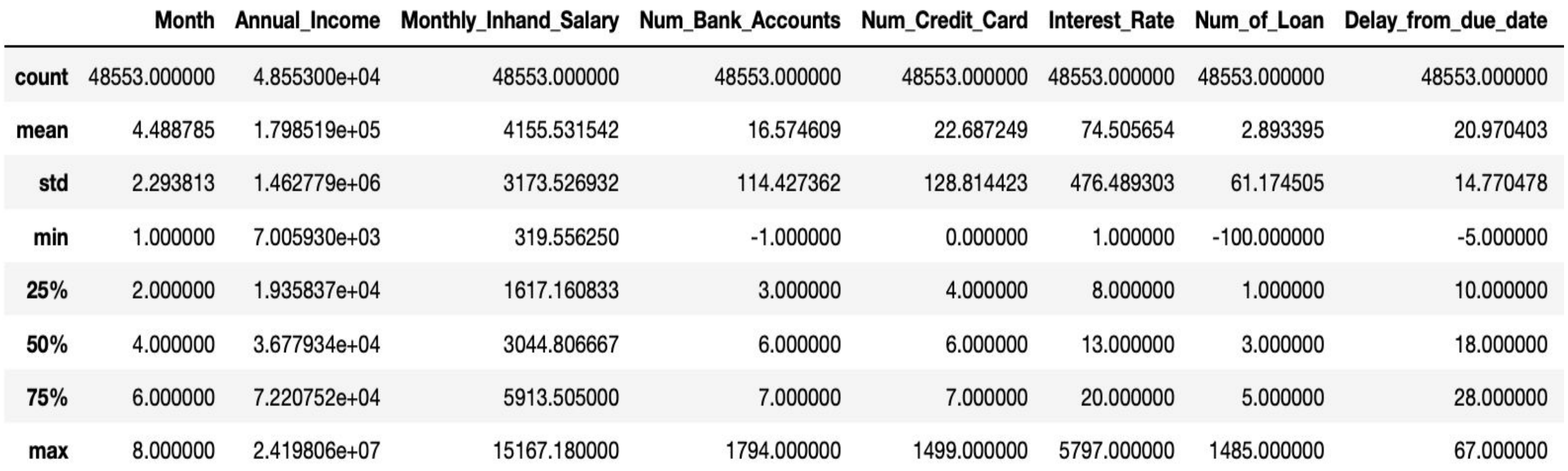
# Category

- Data Exploration Analysis
- Database
- Machine Learning Models

# Data Exploration Analysis

- Data Overview
- Interest Rate vs Credit Score
- Average Interest Rate vs Occupation
- Number of Delayed Payments vs Salary

# Dataset Overview



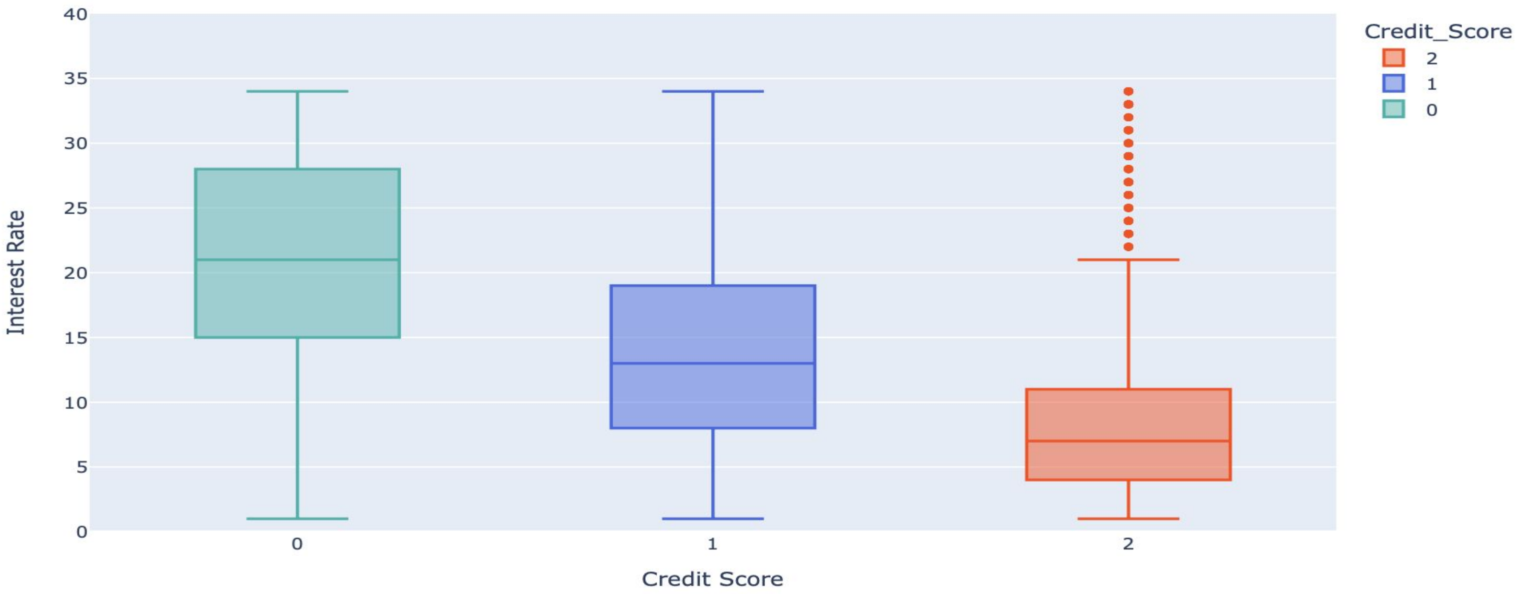
| | Month | Annual_Income | Monthly_Inhand_Salary | Num_Bank_Accounts | Num_Credit_Card | Interest_Rate | Num_of_Loan | Delay_from_due_date |
|---|---|---|---|---|---|---|---|---|
| **count** | 48553.000000 | 4.855300e+04 | 48553.000000 | 48553.000000 | 48553.000000 | 48553.000000 | 48553.000000 | 48553.000000 |
| **mean** | 4.488785 | 1.798519e+05 | 4155.531542 | 16.574609 | 22.687249 | 74.505654 | 2.893395 | 20.970403 |
| **std** | 2.293813 | 1.462779e+06 | 3173.526932 | 114.427362 | 128.814423 | 476.489303 | 61.174505 | 14.770478 |
| **min** | 1.000000 | 7.005930e+03 | 319.556250 | -1.000000 | 0.000000 | 1.000000 | -100.000000 | -5.000000 |
| **25%** | 2.000000 | 1.935837e+04 | 1617.160833 | 3.000000 | 4.000000 | 8.000000 | 1.000000 | 10.000000 |
| **50%** | 4.000000 | 3.677934e+04 | 3044.806667 | 6.000000 | 6.000000 | 13.000000 | 3.000000 | 18.000000 |
| **75%** | 6.000000 | 7.220752e+04 | 5913.505000 | 7.000000 | 7.000000 | 20.000000 | 5.000000 | 28.000000 |
| **max** | 8.000000 | 2.419806e+07 | 15167.180000 | 1794.000000 | 1499.000000 | 5797.000000 | 1485.000000 | 67.000000 |

Tool: Python

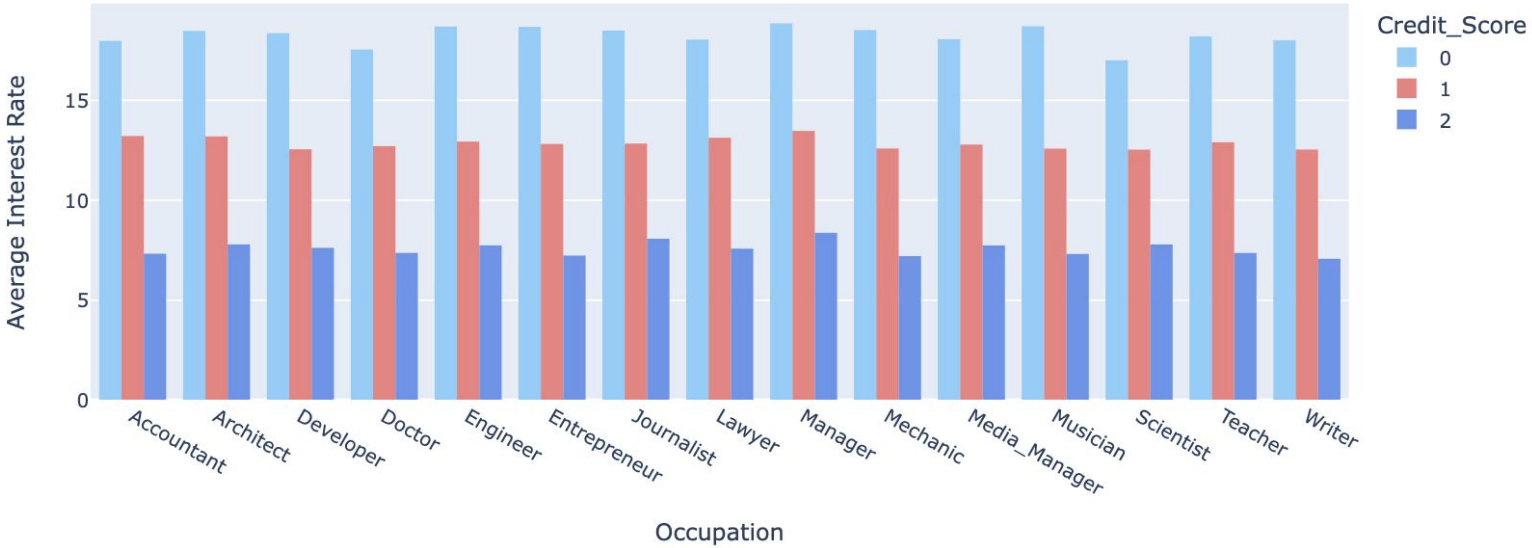
Size: 48553

Target: Credit Score

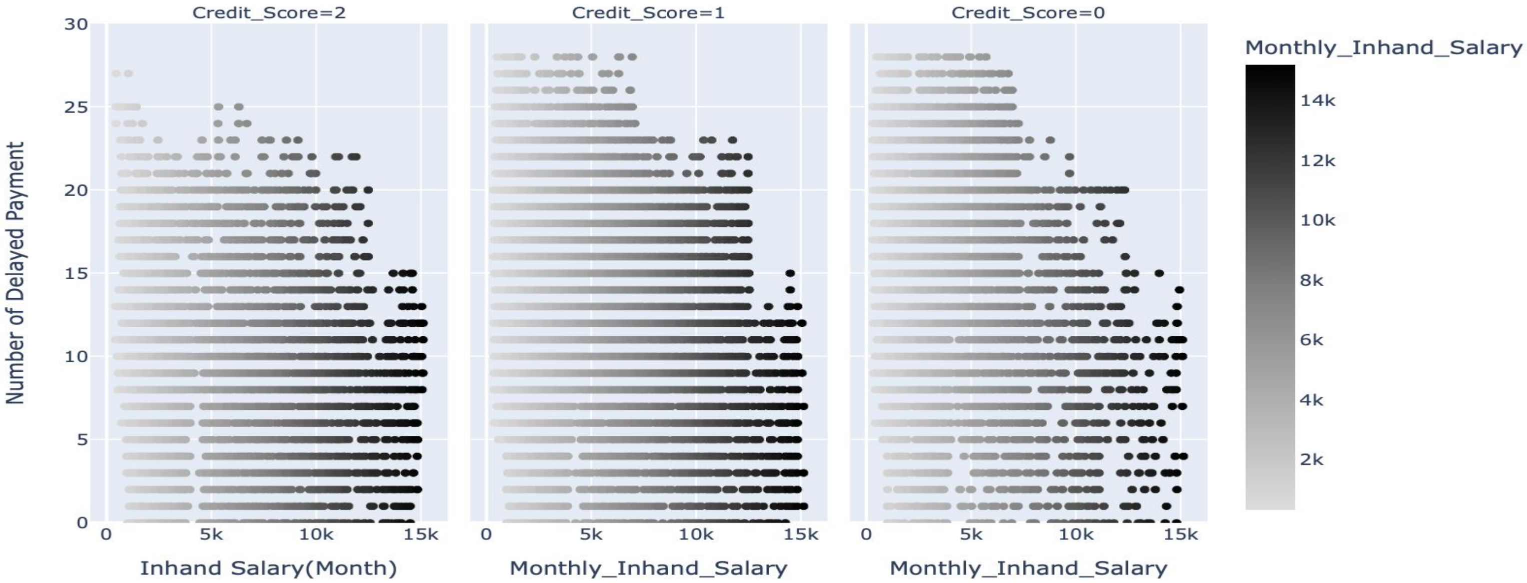
# Interest Rate vs Credit Score



# Average Interest Rate vs Occupation



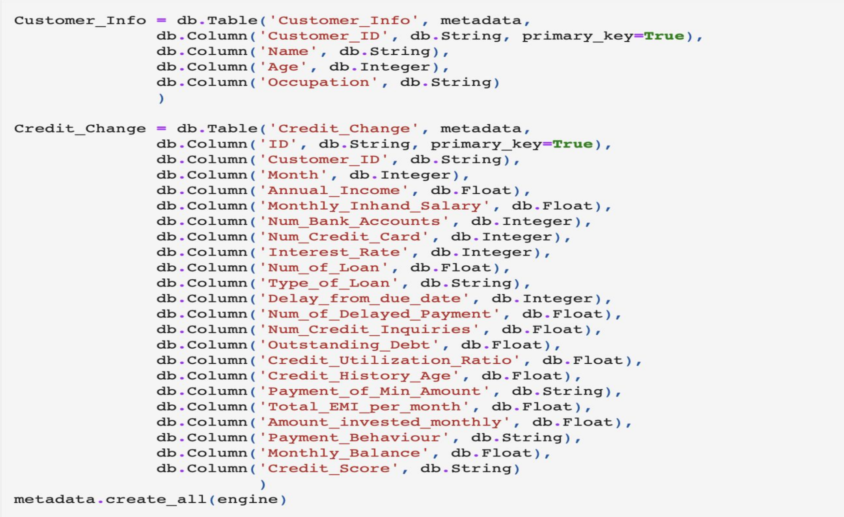
# Number of Delayed Payments vs Salary



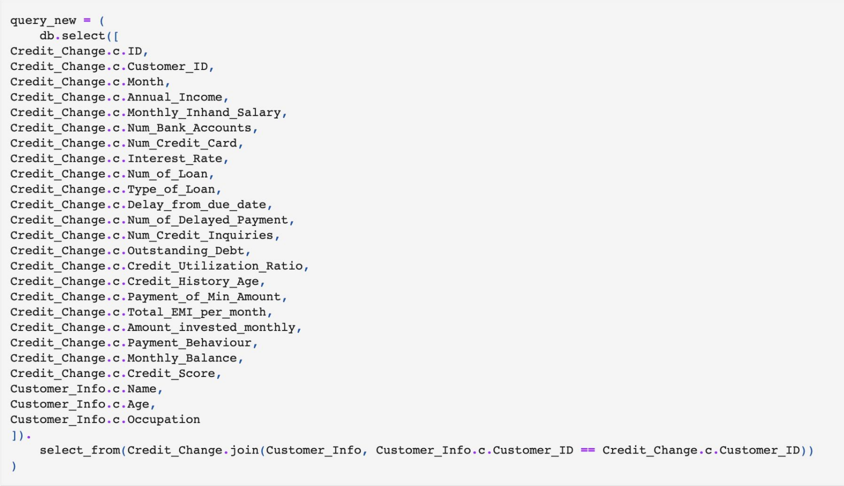
# Database

Tool: Python & SQLAlchemy
Process:

- Import datasets (Customer Info & Credit Change)
- Combine two tables
- Export csv file



```
Customer_Info = db.Table('Customer_Info', metadata,
              db.Column('Customer_ID', db.String, primary_key=True),
              db.Column('Name', db.String),
              db.Column('Age', db.Integer),
              db.Column('Occupation', db.String)
              )

Credit_Change = db.Table('Credit_Change', metadata,
              db.Column('ID', db.String, primary_key=True),
              db.Column('Customer_ID', db.String),
              db.Column('Month', db.Integer),
              db.Column('Annual_Income', db.Float),
              db.Column('Monthly_Inhand_Salary', db.Float),
              db.Column('Num_Bank_Accounts', db.Integer),
              db.Column('Num_Credit_Card', db.Integer),
              db.Column('Interest_Rate', db.Integer),
              db.Column('Num_of_Loan', db.Float),
              db.Column('Type_of_Loan', db.String),
              db.Column('Delay_from_due_date', db.Integer),
              db.Column('Num_of_Delayed_Payment', db.Float),
              db.Column('Num_Credit_Inquiries', db.Float),
              db.Column('Outstanding_Debt', db.Float),
              db.Column('Credit_Utilization_Ratio', db.Float),
              db.Column('Credit_History_Age', db.Float),
              db.Column('Payment_of_Min_Amount', db.String),
              db.Column('Total_EMI_per_month', db.Float),
              db.Column('Amount_invested_monthly', db.Float),
              db.Column('Payment_Behaviour', db.String),
              db.Column('Monthly_Balance', db.Float),
              db.Column('Credit_Score', db.String)
                        )
metadata.create_all(engine)
```



```
query_new = (
    db.select([
Credit_Change.c.ID,
Credit_Change.c.Customer_ID,
Credit_Change.c.Month,
Credit_Change.c.Annual_Income,
Credit_Change.c.Monthly_Inhand_Salary,
Credit_Change.c.Num_Bank_Accounts,
Credit_Change.c.Num_Credit_Card,
Credit_Change.c.Interest_Rate,
Credit_Change.c.Num_of_Loan,
Credit_Change.c.Type_of_Loan,
Credit_Change.c.Delay_from_due_date,
Credit_Change.c.Num_of_Delayed_Payment,
Credit_Change.c.Num_Credit_Inquiries,
Credit_Change.c.Outstanding_Debt,
Credit_Change.c.Credit_Utilization_Ratio,
Credit_Change.c.Credit_History_Age,
Credit_Change.c.Payment_of_Min_Amount,
Credit_Change.c.Total_EMI_per_month,
Credit_Change.c.Amount_invested_monthly,
Credit_Change.c.Payment_Behaviour,
Credit_Change.c.Monthly_Balance,
Credit_Change.c.Credit_Score,
Customer_Info.c.Name,
Customer_Info.c.Age,
Customer_Info.c.Occupation
]).
    select_from(Credit_Change.join(Customer_Info, Customer_Info.c.Customer_ID == Credit_Change.c.Customer_ID))
)
```

# Model Building

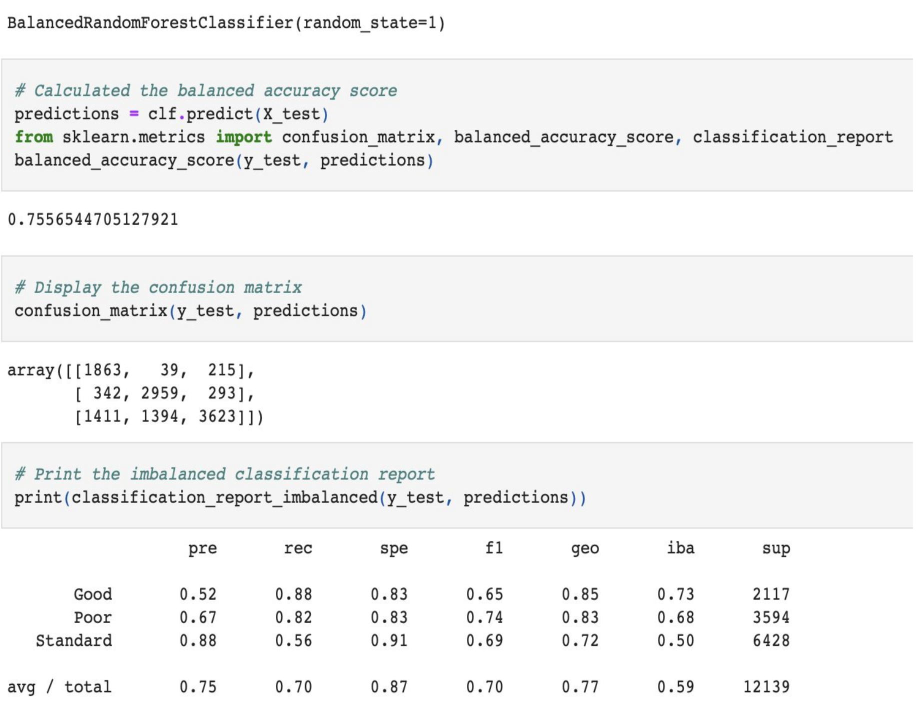
- Supervised Machine Learning
- Deep Machine Learning

# Supervised Machine Learning

Labeled Data >> Supervised Learning Model

Predict Outcome >> Classification

Algorithms : BalancedRandomForestClassifier



```
BalancedRandomForestClassifier(random_state=1)
```

```
# Calculated the balanced accuracy score
predictions = clf.predict(X_test)
from sklearn.metrics import confusion_matrix, balanced_accuracy_score, classification_report
balanced_accuracy_score(y_test, predictions)
```

```
0.7556544705127921
```

```
# Display the confusion matrix
confusion_matrix(y_test, predictions)
```

```
array([[1863,   39,  215],
       [ 342, 2959,  293],
       [1411, 1394, 3623]])
```

```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, predictions))
```

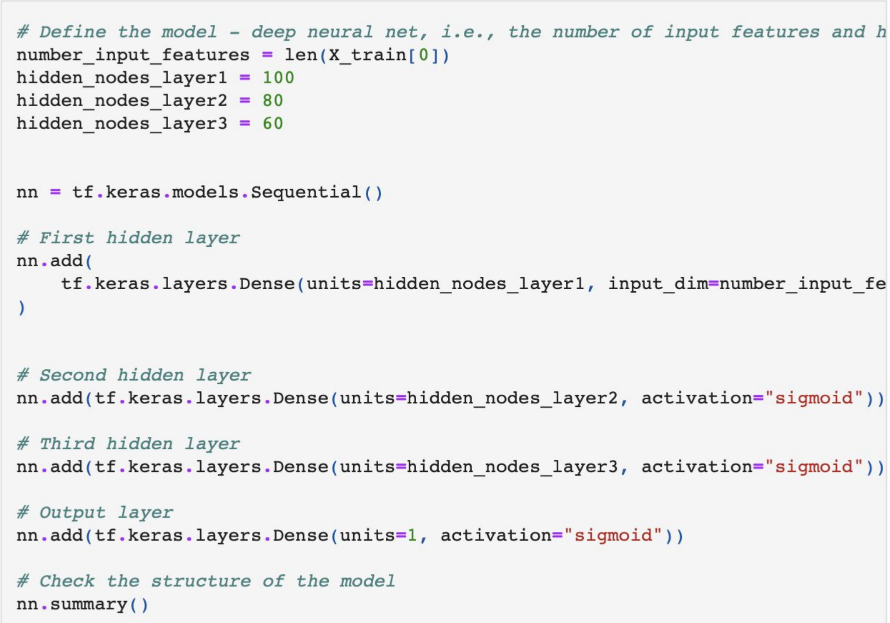
```
                   pre       rec       spe        f1       geo       iba       sup

       Good       0.52      0.88      0.83      0.65      0.85      0.73      2117
       Poor       0.67      0.82      0.83      0.74      0.83      0.68      3594
   Standard       0.88      0.56      0.91      0.69      0.72      0.50      6428

avg / total       0.75      0.70      0.87      0.70      0.77      0.59     12139
```

# Deep Machine Learning

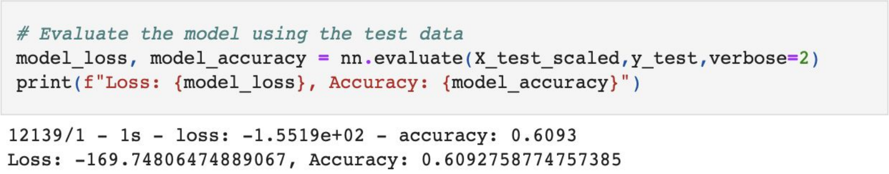
3 hidden layers

Neurons : 100, 80, 60

Epochs: 50



```
# Define the model - deep neural net, i.e., the number of input features and h
number_input_features = len(X_train[0])
hidden_nodes_layer1 = 100
hidden_nodes_layer2 = 80
hidden_nodes_layer3 = 60


nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_fe
)


# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="sigmoid"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="sigmoid"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```



```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
12139/1 - 1s - loss: -1.5519e+02 - accuracy: 0.6093
Loss: -169.74806474889067, Accuracy: 0.6092758774757385
```

# Model Evaluation - Ensemble Learning

- Are robust against overfitting
- be used to rank the importance of input variables
- Can handle thousands of input variables without variable deletion
- Are robust to outliers and nonlinear data
- Run efficiently on large datasets

# References

- https://machinelearningmastery.com/why-use-ensemble-learning/
- https://www.kaggle.com/datasets/parisrohan/credit-score-classification