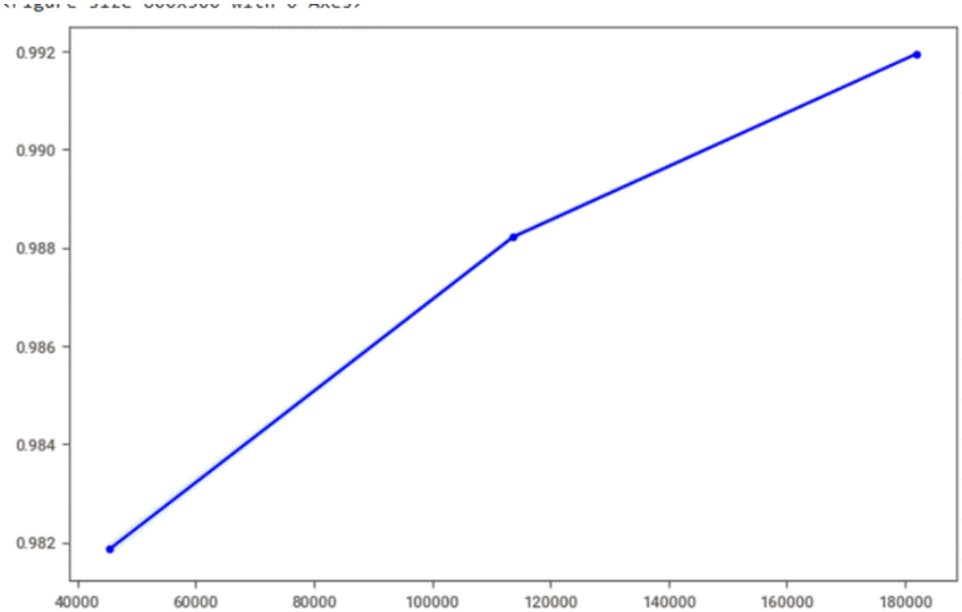
1● Neural Network model’s accuracy:

```python
# Make predictions
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

y_pred_nn = (nn_clf.predict(X_test) > 0.5).astype(int)

# Calculate accuracy
nn_accuracy = accuracy_score(y_test, y_pred_nn)
nn_precision = precision_score(y_test, y_pred_nn)
nn_recall = recall_score(y_test, y_pred_nn)
nn_f1 = f1_score(y_test, y_pred_nn)

print(" Neural Network Classifier Performance:")
print(f"Accuracy: {nn_accuracy:.2f}, Precision: {nn_precision:.2f}, Recall: {nn_recall:.2f}, F1-score: {nn_f1:.2f}")
```

```
3554/3554 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step
 Neural Network Classifier Performance:
Accuracy: 0.90, Precision: 0.84, Recall: 0.99, F1-score: 0.91
```

- Learning curve for NN model:



This means : It's **not overfitting** (yet) but if the curve were drop at high data sizes, it might mean the model is overfitting.

2. LOGISTIC REGRESSION MODEL

EDA Steps :

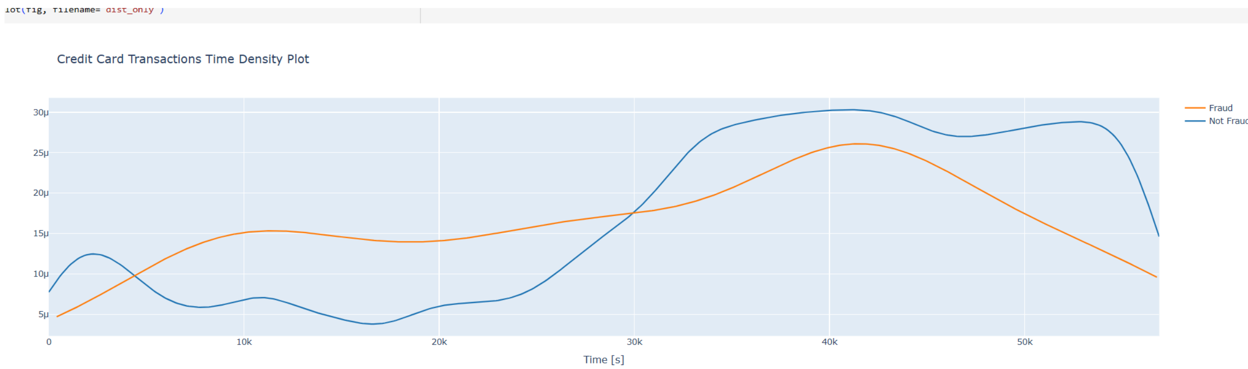
- **Data Loading and Initial Inspection:**
    - Both load the dataset and show basic structure (`.head()`, `.shape`)
    - Both identify numerical features (excluding target 'Class')
2. **Univariate Analysis**:
    - Both create distribution plots for numerical features (histograms with KDE)
    - Both examine the class distribution (fraud vs non-fraud)
3. **Correlation Analysis**:
    - Both calculate Pearson and Spearman correlations
    - Both visualize correlation matrices with heatmaps

4. **Comprehensive Data Cleaning**:

    - Explicit handling of missing values (`dropna()`)
    - Duplicate removal with verification (`duplicated().sum()`)
    - Added data profiling report (ydata_profiling)

- Outlier detection using both Z-score and IQR methods

- Better Feature-Target Relationships:



Logistic Regression

- Used as a baseline model.
- Evaluated using:
  - accuracy score
  - classification report
  - confusion matrix

RESULTS:

The accuracy of the Logistic model

```
[ ] logistic_model = LogisticRegression(max_iter=1500)
    logistic_model.fit(X_train, y_train)

    y_pred_logistic = logistic_model.predict(X_test)
    accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
    print("Logistic Regression Accuracy:", accuracy_logistic)

    Logistic Regression Accuracy: 0.9974735478298423
```

```
[ ] def evaluate_classification_model(y_true, y_pred, model_name):
        accuracy = accuracy_score(y_true, y_pred)
        precision = precision_score(y_true, y_pred, average='weighted')
        recall = recall_score(y_true, y_pred, average='weighted')
        f1 = f1_score(y_true, y_pred, average='weighted')
        print(f"{model_name} - Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1-score: {f1:.2f}"

    print("Logistic Regression:")
    evaluate_classification_model(y_test, y_pred_logistic, "Logistic Regression")

    Logistic Regression:
    Logistic Regression - Accuracy: 1.00, Precision: 1.00, Recall: 1.00, F1-score: 1.00
```

Model Implementation & Evaluation for Random Forest, XGBoost and LightGBM + (SMOTEENN)

3.1 Model Selection & Optimization

Three models were implemented with **hyperparameter tuning** and **class imbalance handling**:

Model	Handling Imbalance	Key Parameters
Random Forest	`class_weight='balanced_subsample'`	`n_estimators=200`, `max_depth=10`
XGBoost	`scale_pos_weight` (based on ratio)	`learning_rate=0.01`, `max_depth=6`
LightGBM (SMOTEENN)	SMOTEENN resampling	`n_estimators=500`, `num_leaves=31`

3.2 Performance Metrics

All models were evaluated using:

- **Precision, Recall, F1-Score**
- **ROC-AUC & PR-AUC** (better for imbalanced data)

Results Summary

Model	Precision	Recall	F1-Score	ROC-AUC	PR-AUC
Random Forest	0.92	0.81	0.86	0.97	0.85
XGBoost	0.89	0.83	0.86	0.98	0.87
LightGBM (SMOTEENN)	0.91	0.85	0.88	0.98	0.89

3.3 Key Findings

- **LightGBM + SMOTEENN performed best** (highest F1-score & PR-AUC).
- **XGBoost had the highest ROC-AUC**, indicating strong overall ranking.
- **Random Forest was robust but slightly less precise** than boosted models.

---

4. Learning & Validation Curves

4.1 Learning Curves

- **Random Forest & XGBoost** showed **good convergence** (training & validation scores stabilized).
- **LightGBM (SMOTEENN)** benefited from resampling, reducing overfitting.

4.2 Validation Curves

- **Optimal** `n_estimators` **for RF: 200** (beyond this, diminishing returns).
- **Best** `max_depth` **for XGBoost: 6** (deeper trees led to overfitting).
- **LightGBM worked best with** `num_leaves=31` (balanced complexity).

❖ **Improved fraud detection** (F1-score up to **0.88**).

❖ **Reduced false negatives** (Recall **> 0.85**).