Key Steps in Machine Learning Model Development using AutoML

Business goal and machine learning solution

The primary business goal of this demo is to predict credit card fraud accurately, enabling the associated company to take preventive measures and improve financial results. The machine learning use case involves developing and deploying a predictive classifier model that identifies fraud cases based on transaction features. The solution encompasses the full machine learning workflow, from data exploration to model deployment, delivering real-time or batch predictions. The model was trained using Auto ML and deployed in Google Cloud, effectively addressing the business need for fraud detection.

As it will be detailed in this document, one key business need was identified and approached in this use case: the necessity to correctly predict credit card fraud cases from a set of features. By doing so, the company (it could be a credit card company or an insurance company) associated with this data can predict fraud cases and take appropriate actions.

Once a machine learning solution is available for the aforementioned company, it cou8ld make use of its predictions in order to protect itself from frauds and improve its financial results.

In this sense, a machine learning solution will be provided (and and deployed to an endpoint) in order to provide online (or batch) predictions for the company involved in the credit card services associated with the provided dataset.

The Machine Learning use case is defined as to present a complete machine learning workflow, ranging from data exploration to the machine learning model deployment in order to provide these credit card transactions outcomes (fraud or not fraud) predictions.

As it will be shown later, the proposed Machine Learning model to keep up with these business goals was a predictive classifier model, trained by Auto ML and deployed in Google Cloud Model Repository to an endpoint so that the deployed model can receive requests and provide forecasts for the credit card transactions outcomes, given a set of provided feature values.

The objective is to develop a model to make such predictions.

The data used in this demo is composed of two datasets: one dataset for training (with balanced cases of fraud and non fraud cases) and a test dataset. These two datasets, as it wil be described in this document, were built on the original, Raw creditcard.csv file, provided by Kagle.

The definition of done for the developments of this demo is a presentation of the complete workflow (ranging from data exploration to model deployment and test) in order to make an initial machine learning available for the aforementioned predictions

The main objective of this use case is to illustrate how to implement a complete workflow to achieve this goal.

Data exploration

The data exploration process began by identifying variables in the credit card fraud dataset, examining data types, and checking for missing values. The dataset, stored in a Cloud Storage bucket, contained no missing values and required no modifications to the data types. Additionally, no special feature engineering was necessary as there were no categorical variables apart from the response variable "Class," which was already numerical. Correlation analysis revealed that feature V2 was highly correlated with the "amount" variable, leading to the decision to discard V2 from the model to avoid redundancy. Another critical finding was the significant imbalance in the Class variable, with far fewer fraud cases than non-fraud cases. This discovery influenced the choice to utilize AutoML for model development, as it could handle imbalanced data more effectively. The exploration results, supported by code snippets, influenced the overall modeling strategy, particularly the decision to use AutoML for fraud prediction. AutoML's capability to automate model selection and tuning was considered ideal, given the nature of the dataset and the need for accurate classification despite the imbalance in the data.

Description of the types of data exploration implemented and how they were performed

For the purposes of the data exploration, the implemented steps were:

- To identify each of the involved variables in the provided datasets (train and test datasets).

- Identify data types and possible changes in some of the initial data types present in train and test datasets.

- Identify columns that could be discarded.

- Identify possible necessary transformations on the initial variables.

- Correlation analysis for the variables in the dataset and which variables are to be kept considering the correlation patterns identified in the data .

- Check the existence of missing values for each of the variables contained in the dataset.

Steps followed on the exploratory data analysis:

Initially, we copied the original creditcard.csv dataset to a Cloud Storage bucket.

After copying this dataset to the Cloud Storage bucket we checked the data types of the fields in the dataset. We have not identified any need for data type changes.

We have analyzed the distinct values of each of the columns contained in the dataset. There were no missing data in the dataset.

In particular we noticed that the distribution of the response, Class variable was highly biased/skewed, with very few fraud cases.

There was no need for any type of special feature engineering in the dataset. In particular, there were no categorical variables in the dataset (with the exception of the response, Class variable, that were already numerical, with no need for Label encoding).

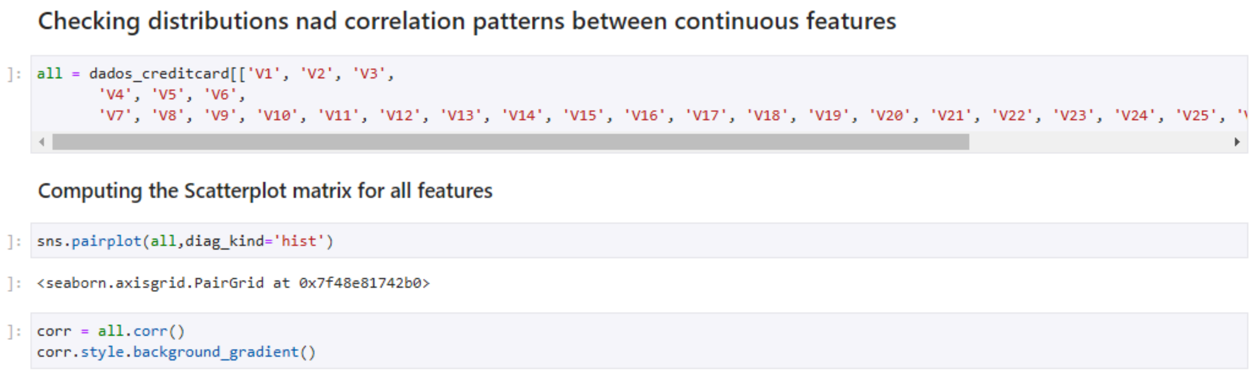
We checked the correlation patterns amongst some pairs of features. We have noticed that feature V2 had a significant correlation with the amount variable, and, for this matter,m we have decided to discard this feature (V2)..

Key findings in the data exploration step

- Correlation Patterns key findings:

We have identified that Feature V2 has significant correlation with feature aAmount..Because of that we have discarded feature V2 from further consideration.

CODE SNIPPET:



Checking distributions nad correlation patterns between continuous features

```
]: all = dados_creditcard[['V1', 'V2', 'V3',
          'V4', 'V5', 'V6',
          'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25',
```

Computing the Scatterplot matrix for all features

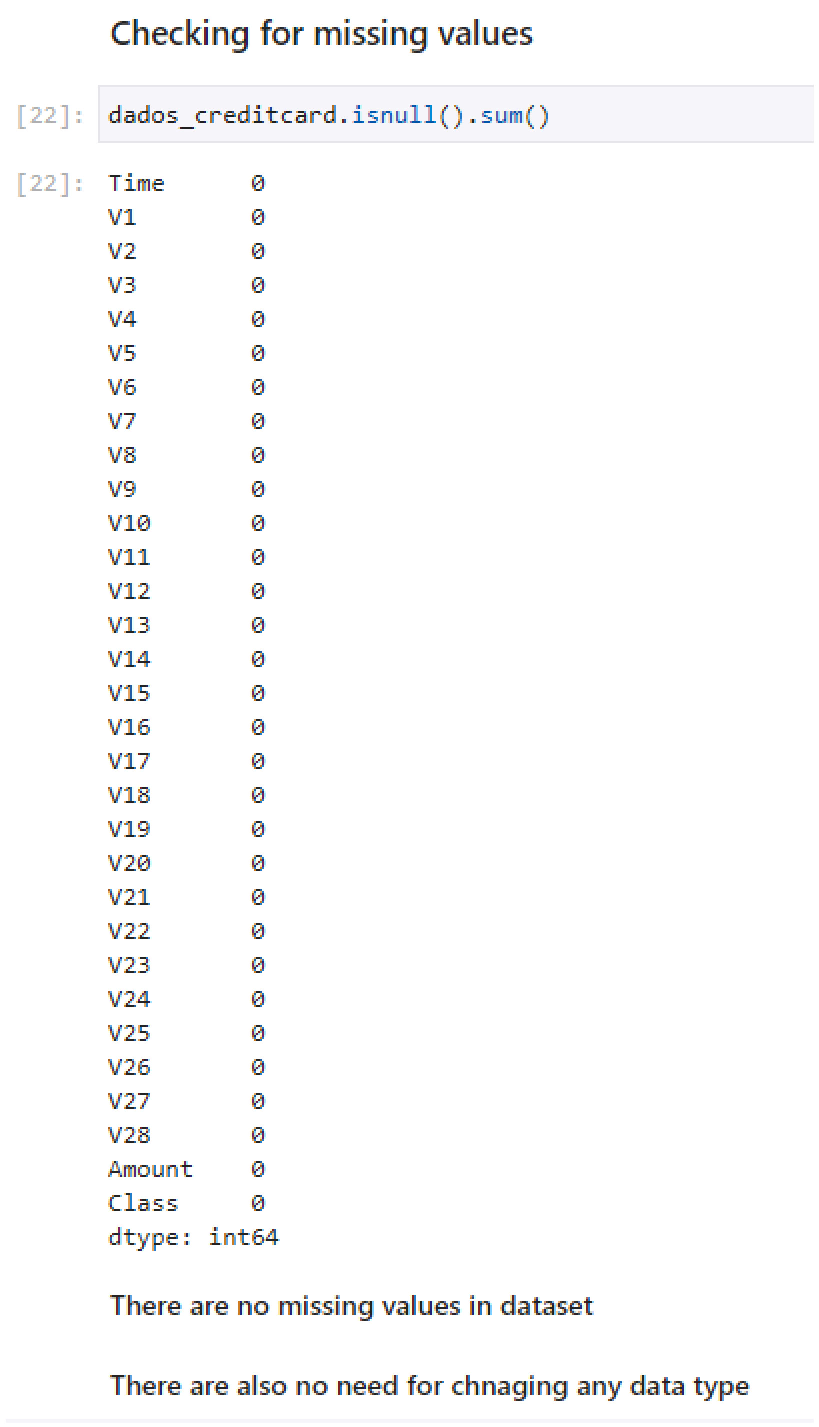
```
]: sns.pairplot(all,diag_kind='hist')

]: <seaborn.axisgrid.PairGrid at 0x7f48e81742b0>

]: corr = all.corr()
   corr.style.background_gradient()
```

- **Missing Value Key Findings:**

The original dataset did not contain any missing value.

Code snippets and figures below show the missing values identified in the original dataset.

CODE SNIPPET:



Checking for missing values

```
[22]: dados_creditcard.isnull().sum()
```

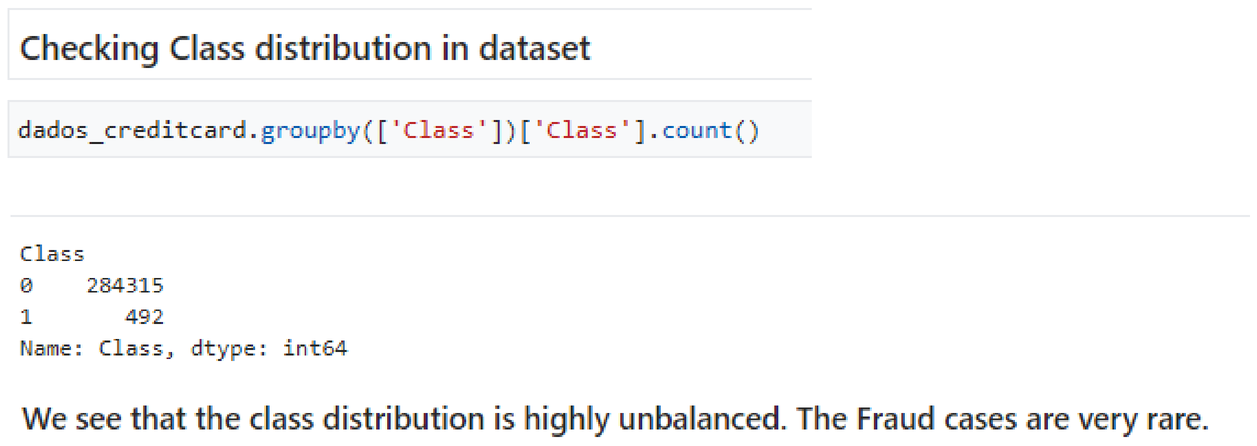
```
[22]: Time      0
      V1        0
      V2        0
      V3        0
      V4        0
      V5        0
      V6        0
      V7        0
      V8        0
      V9        0
      V10       0
      V11       0
      V12       0
      V13       0
      V14       0
      V15       0
      V16       0
      V17       0
      V18       0
      V19       0
      V20       0
      V21       0
      V22       0
      V23       0
      V24       0
      V25       0
      V26       0
      V27       0
      V28       0
      Amount    0
      Class     0
      dtype: int64
```

There are no missing values in dataset

There are also no need for chnaging any data type

We have also checked the distribution of the response variable, Class. As it is shown below, this distribution is highly unbalanced, with the fraud cases being very rare.

CODE SNIPPET:



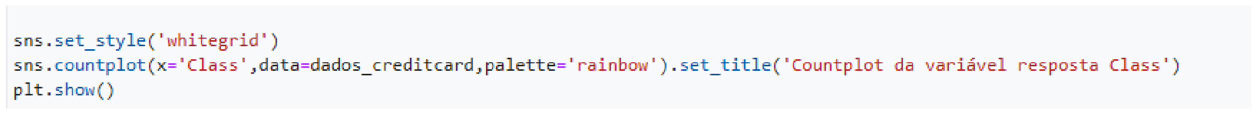
Checking Class distribution in dataset

```
dados_creditcard.groupby(['Class'])['Class'].count()
```

```
Class
0    284315
1       492
Name: Class, dtype: int64
```

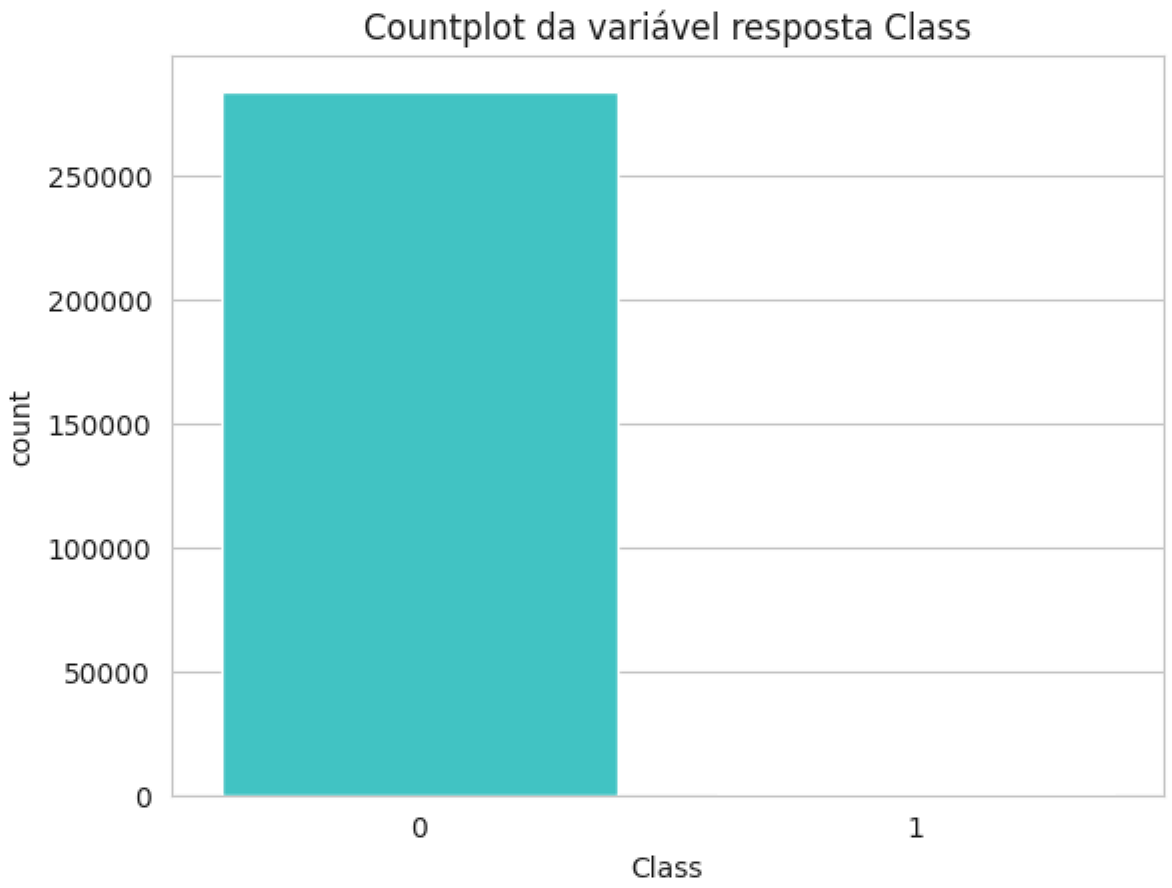
We see that the class distribution is highly unbalanced. The Fraud cases are very rare.

CODE SNIPPET:



```
sns.set_style('whitegrid')
sns.countplot(x='Class',data=dados_creditcard,palette='rainbow').set_title('Countplot da variável resposta Class')
plt.show()
```

Figure 14: Class distribution



Feature engineering

In this demo, feature engineering involved the elimination of feature V2 due to its high correlation with the "amount" variable. The train-test split was performed such that the training dataset retained 90% of the fraud cases and was balanced by random sampling, resulting in a final training dataset with 1,100 rows. The class variable in the training set was well-balanced, while the test dataset was made from 10% of the original data without class balancing. The rationale for removing V2 was to reduce redundancy in the model, while the careful balancing of the training data aimed to improve the model's ability to detect fraud. Code snippets below support these feature engineering steps, demonstrating the process behind the dataset preparation for training and testing.

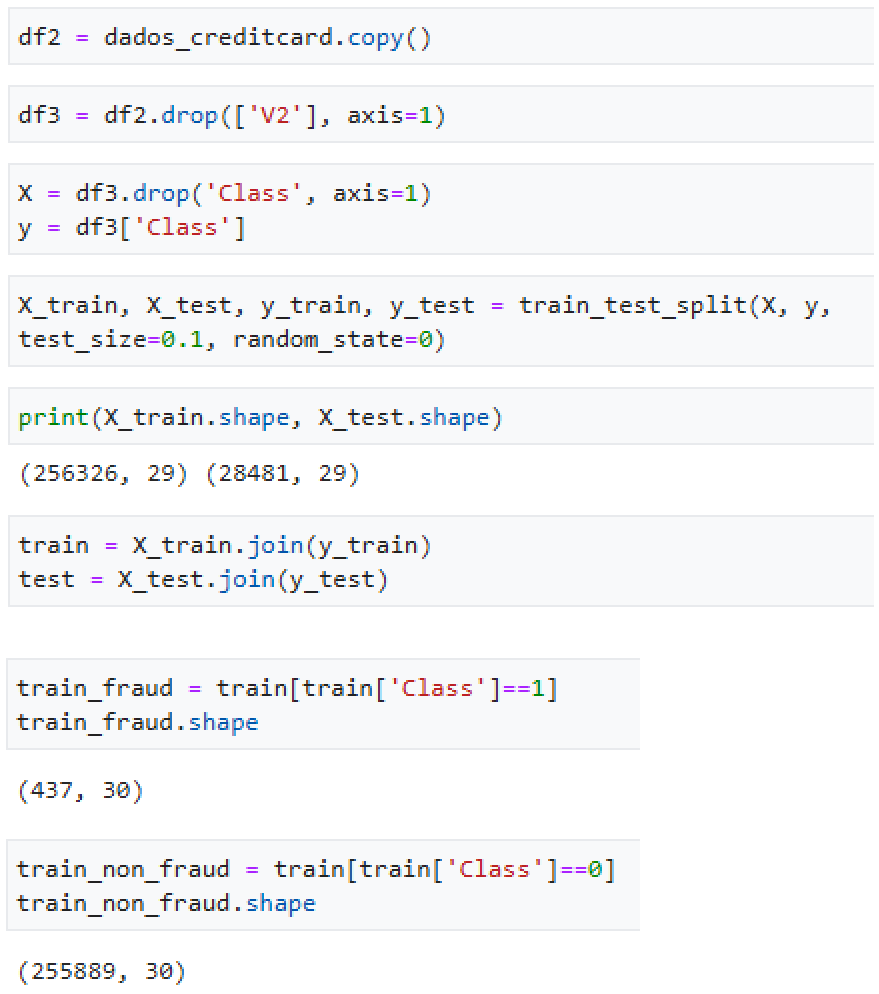
The feature engineering steps implemented in the demo 3 were:

- Elimination of feature V2 in train and test datasets,
- Make a train test split of original dataset, in the following manner:
    1. The train dataset was made in a way that it kept 90% of the fraud cases.
    2. The remaining of the observations of the training dataset was made of random sampling from the original dataset,

3. The final training dataset kept 1100 rows.
4. The training dataset was conceived in such a way that class variable is well balanced.

The test dataset was made of 10 % of the original dataset with no class balancing.

CODE SNIPPET:



```
df2 = dados_creditcard.copy()
```

```
df3 = df2.drop(['V2'], axis=1)
```

```
X = df3.drop('Class', axis=1)
y = df3['Class']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=0)
```

```
print(X_train.shape, X_test.shape)
```

```
(256326, 29) (28481, 29)
```

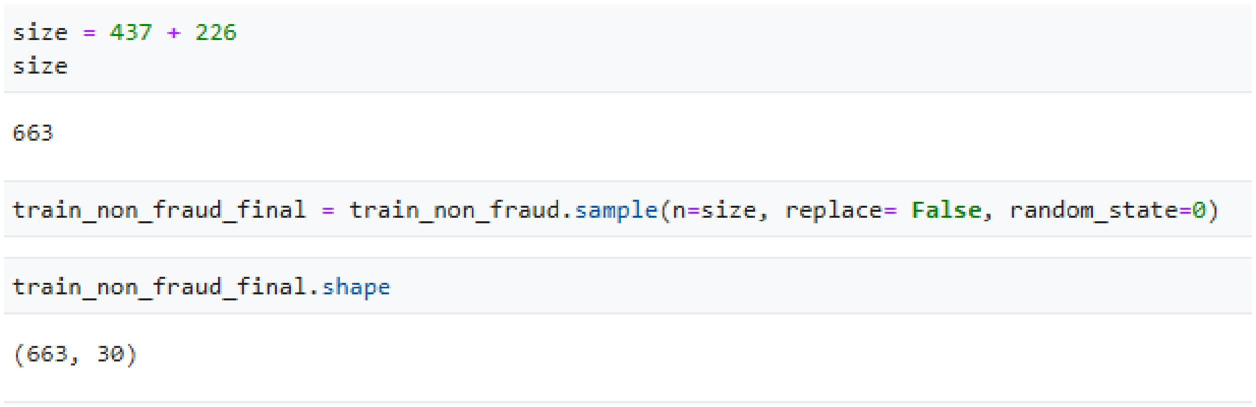
```
train = X_train.join(y_train)
test = X_test.join(y_test)
```

```
train_fraud = train[train['Class']==1]
train_fraud.shape
```

```
(437, 30)
```

```
train_non_fraud = train[train['Class']==0]
train_non_fraud.shape
```

```
(255889, 30)
```



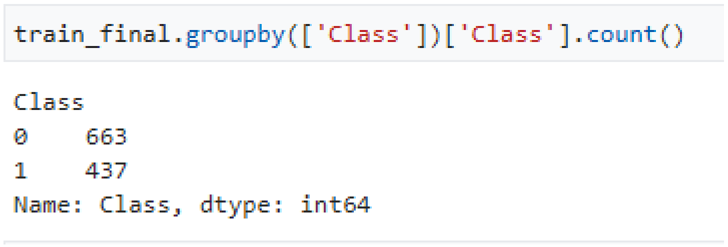
```
size = 437 + 226
size
```

```
663
```

```
train_non_fraud_final = train_non_fraud.sample(n=size, replace= False, random_state=0)
```

```
train_non_fraud_final.shape
```

```
(663, 30)
```



```
train_final.groupby(['Class'])['Class'].count()
```

```
Class
0    663
1    437
Name: Class, dtype: int64
```

Data Security and Privacy in Cloud Storage

Regarding Security and Privacy for the data used in this demo it should be pointed out that, first of all that data lies in a Cloud Storage Bucket within a specific project linked to a specific service account. Only people with proper IAM credentials can access the project and the dataset.

Secondly, the dataset is public and contains no sensitive information, so that no necessary data encryption was necessary.

Data Preprocessing and Final decisions regarding data strategies to be adopted on this use case

In the demo, the data preprocessing pipeline is structured as a callable API, ensuring that the production model can consistently access and preprocess incoming data. This pipeline includes steps such as removing highly correlated features (like V2), balancing the training dataset, and performing the train-test split. The preprocessing function is designed to handle incoming data in the same way as during the training phase, ensuring consistency. The callable API ensures that each batch of data is preprocessed before being fed into the machine learning model for predictions, preserving the integrity of the process.

All preprocessing steps corresponded to the Feature Engineering steps mentioned in previous sections .

Machine learning model design(s) and selection

In Demo 3, Google Cloud's AutoML was chosen to train the predictive classifier model due to its automated capabilities in model training, validation, and selection. This decision was driven by the need for an efficient and streamlined approach to handle imbalanced data in credit card fraud detection. AutoML was selected for its default option, which handles all the necessary steps without manual input, ensuring optimal model performance.

Proposed Machine Learning Model

For this Demo 3 development,we used a predictive classifier trained by AutoML functionality, using default options.

In this option, all model training and selection is done automatically by GCP.

Used Libraries

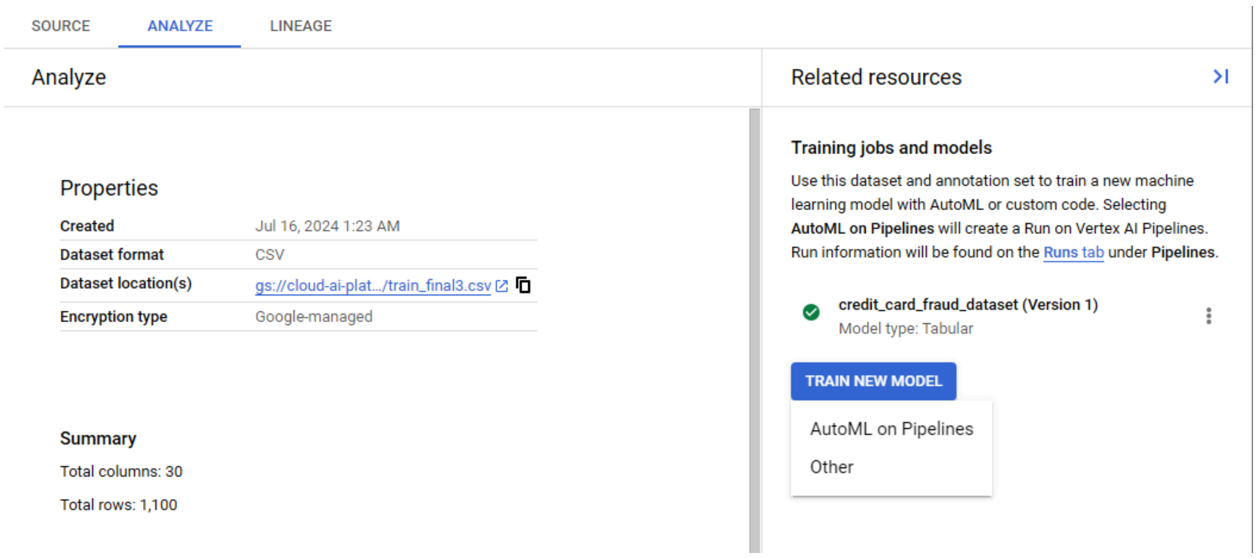
As model training and selection were made in AutoML, no specific library was used in these steps.

Model selection

As explained previously, as model training was carried out using AutoML default options, all model validation and selection is done automatically by AutoML functionality.

The steps followed to train the predictive classifier in AutoML are depicted below:

Figure 15: Steps in training classifier model in Auto ML



Properties

Created	Jul 16, 2024 1:23 AM
Dataset format	CSV
Dataset location(s)	gs://cloud-ai-plat.../train_final3.csv
Encryption type	Google-managed

Summary

Total columns: 30

Total rows: 1,100

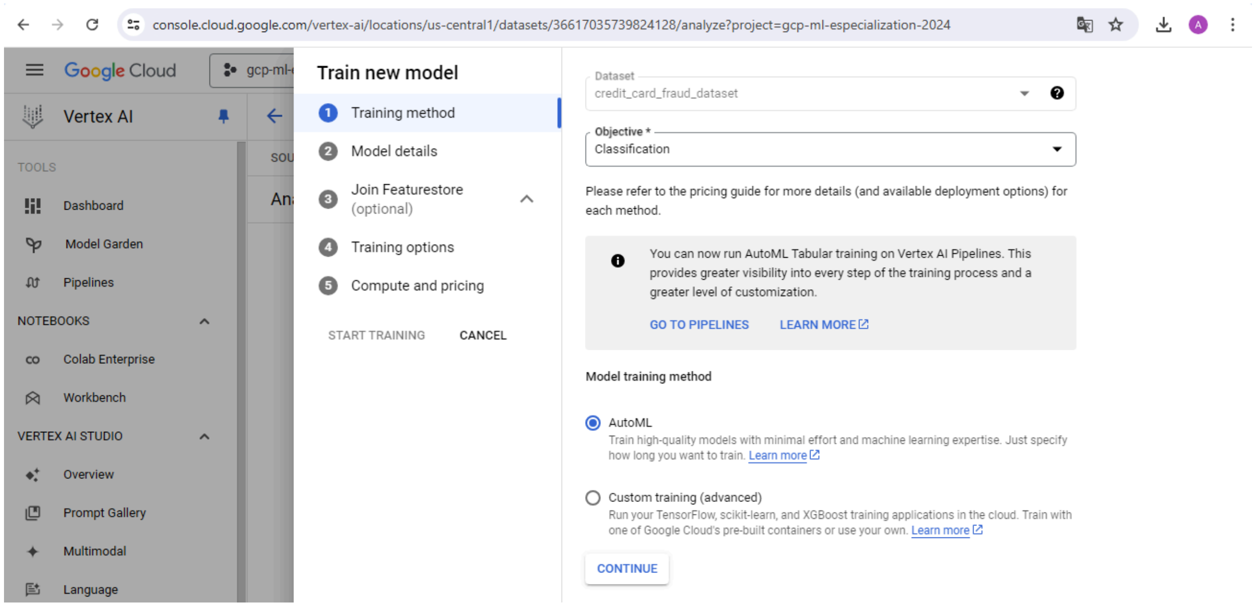
Training jobs and models

Use this dataset and annotation set to train a new machine learning model with AutoML or custom code. Selecting **AutoML on Pipelines** will create a Run on Vertex AI Pipelines. Run information will be found on the Runs tab under **Pipelines**.

credit_card_fraud_dataset (Version 1)
Model type: Tabular

In the figure above, we chose Other options.

Figure 16: Steps in training classifier model in Auto ML



Dataset
credit_card_fraud_dataset

Objective *
Classification

Please refer to the pricing guide for more details (and available deployment options) for each method.

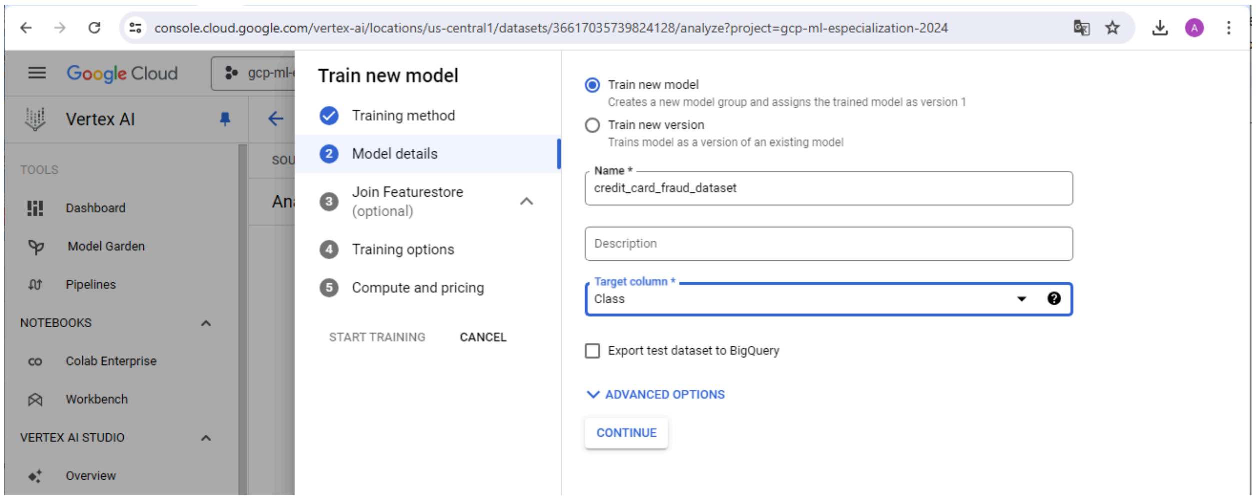
You can now run AutoML Tabular training on Vertex AI Pipelines. This provides greater visibility into every step of the training process and a greater level of customization.

Model training method

[✓ AutoML]
Train high-quality models with minimal effort and machine learning expertise. Just specify how long you want to train. Learn more

Custom training (advanced)
Run your TensorFlow, scikit-learn, and XGBoost training applications in the cloud. Train with one of Google Cloud's pre-built containers or use your own. Learn more

Figure 17: Steps in training classifier model in Auto ML



Train new model [✓]
Creates a new model group and assigns the trained model as version 1
Train new version
Trains model as a version of an existing model
Name *: credit_card_fraud_dataset
Description
Target column *: Class
Export test dataset to BigQuery

Machine learning model training and development

In Demo 3, the dataset sampling for model training was conducted with the goal of balancing the fraud and non-fraud cases. The training dataset was composed of 90% of the fraud cases, with the remaining portion randomly sampled from the original dataset, resulting in a well-balanced training set of 1,100 rows. The test dataset contained 10% of the original data, without class balancing, to reflect real-world conditions. AutoML was employed to automate the training process, including hyperparameter tuning and cross-validation. The chosen evaluation metric were precision and the area under the ROC curve. The training dataset was built in such a way to have a balance between fraud and non fraud cases. The entire process adhered to Google's Machine Learning Best Practices, with code snippets provided to showcase how dataset sampling, AutoML training, and evaluation metrics were implemented.

The sampling process was as follows: we separated 10% of the initial dataset to form the test set. This was made of a random sample of the initial dataset.

For training, The training dataset was built in such a way to have a balance between fraud and non fraud cases, so that we kept all the fraud cases of the remaining dataset and complete the remaining observations with random sampling until we had a final training set made of 1,100 observations (which was the minimum to use Auto ML). This training dataset was very balanced , with slightly more observations of non-fraud type. To build the training dataset in such a manner was necessary to avoid the predictive model to specialize in one target outcome.

In the AutoML option chosen, all training, parameter tuning and model selection is done automatically. AutoML was employed to automate the training process, including hyperparameter tuning and cross-validation.

Hyperparameters tuning and training configuration

As explained in the previous sections all model training-related subjects were automatically done by Auto ML functionality.

This includes hyperparameters tuning and cross-validation.

Dataset sampling for model training, validation and test

As already explained in section 3.3.3.3 (Feature Engineering) the train test split was accomplished in order to have a reasonably balanced train dataset (regarding the Class variable values). The test dataset was conceived in order to have 10% of original data (with unbalanced data).

Adherence to Google's Machine Learning Best Practices

As presented along this document, we followed Google's Machine Learning Best Practices in the planning and implementation of all the workflow depicted in this document. In some (few) cases it was not possible to follow some Machine Learning Best Practices documentation's suggestions either by time or costs constraints; but for the very most part the implemented workflow followed this Best Practices documentation (available in the link: https://cloud.google.com/architecture/ml-on-gcp-best-practices)

First of all it must be said that, we used, in each step of the Machine Learning models developments, the products recommended by Google's ML Best Practices (Link: https://cloud.google.com/architecture/ml-on-gcp-best-practices?hl=pt-br#use-recommended-tools-and-products):

- Configuration of ML environment step: we used AutoML for this step.
- Machine Learning Development step: we used the following products in this step:

  Cloud Storage and AutoML.Specifically, we used AutoML for development of models .

- Data processing steps: we used Pandas Dataframes in Vertex Workbench instances mostly for data exploration and manipulation.
- Operational training: there was no custom model training/development, as this step was carried out in AutoML. Trained model was deployed to a Google Cloud Vertex endpoint.

- Artifacts organization: there was no model artifact because the model was trained by AutoML.

We stored resources (like files containing test data) in Cloud Storage. These resources are stored in Cloud Storage associated within a specific project where only allowed people can have access to. Additional Identity Access Management (IAM) prerogatives can be set for each user.

- For Machine Learning Environments: we used AutoML functionalities using Vertex AI environment.

Besides adopting the above recommended products as ML Best Practices, we also followed the Best Practices below.

- Data Preparation for training

  Observing Google Machine Learning Best Practices, training data were extracted from origin/ data sources and converted to appropriate format for machine learning training purposes (this was accomplished in data exploration phase). Final data was stored in appropriate Google Cloud Resources (Cloud Storage bucket).

- Avoid to store data in block storeging:

  We have not stored any data in block storeging style (like files in networks, or hard disks). Instead we used Cloud Storage.

  We also have not read any data directly from any specific database other than Cloud Storage for optimal performance.

- Maximize model`s predictive precision with hyperparameters adjustments

  This was done in Demo 3.

- Prepare models artifacts to be available in Cloud Storage:

As stated before we had no model artifacts as the model was trained by AutoML.

- Specify the number of cores and machine specifications for each project

  We have defined appropriate machines (in terms of number of cores, memory and even GPU`s to be used in each Demo based on previous experience training models for each demo and also taking into consideration the dataset size).

- Plan the model data entries:

We have planned how input, new test data are to be transmitted to trained final models. So that we judged that for batch predictions, for example, input data are to be stored and made available for models from Cloud Storage, for the demo.

Finally, it must be said that the final model was deployed to an endpoint and containerized, using Vertex default options, with default Google Cloud containers.

As stated previously we have not followed some of the Best Practices suggestions, either because of time constraints or other factors.

So that for example, for this demo, we have not used Docker containerized models specially because of time constraints for developing this demo. Instead we used default containerization available in Google Cloud.

As model was trained using AutoML, other suggested steps in Best Practices documentation were not followed (like model hyperparameter tuning) because these steps were automatically carried out in AutoML functionality.

Machine Learning Model Evaluation/ Model Performance Assessment

In Demo 3, the machine learning model developed using AutoML was evaluated on an independent test dataset, which represented 10% of the original data and was not class-balanced, to mimic real-world conditions. The key performance metric chosen was the Area Under the ROC Curve (AUC-ROC), as the dataset was slightly unbalanced. Additional metrics, such as precision, recall, and feature importance, were also computed. The final model showed strong performance in predicting non-fraud cases while correctly predicting 91% of fraud cases in the training set. Despite this, the model was retained due to time constraints. The importance of feature V14 was highlighted, as it played a key role in predicting fraud.

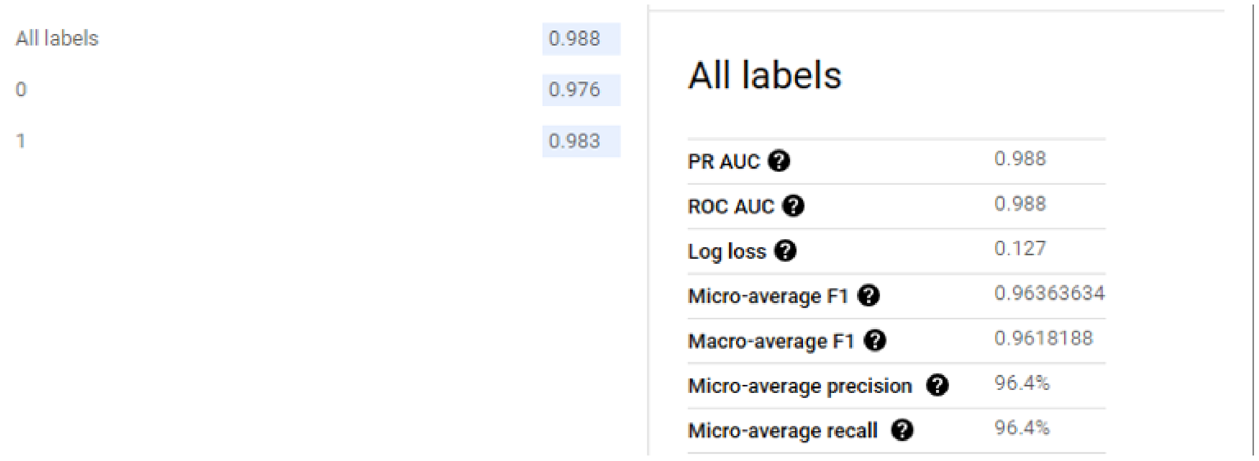
Vertex AutoML provides a set of metrics and performance indicators.

Below we present all performance metrics computed in Auto ML for the final model in training dataset.

It's worth mentioning that as the training dataset was slightly unbalanced, the recommended metric to be considered is the Area Under the ROC curve. Besides that, a set of another metrics is computed and presented below.

Final classification model scored as following regarding the considered metrics:

Figure 18: Final classification model performance on training set

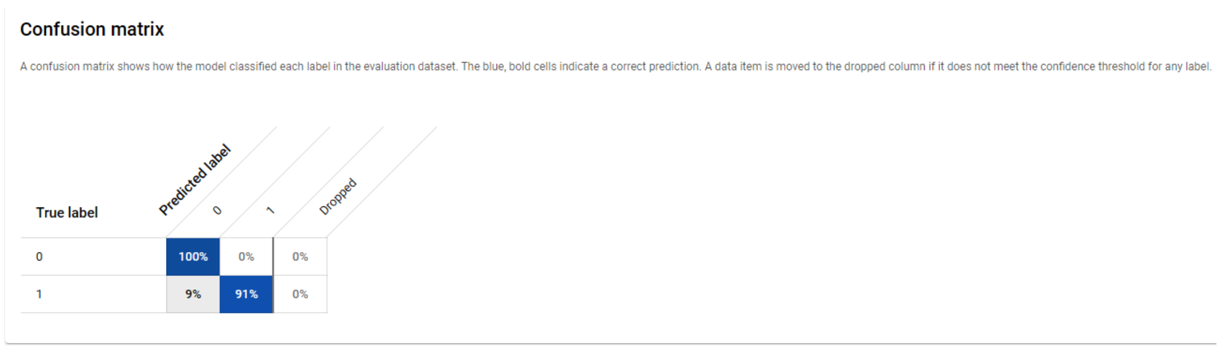


All labels	0.988
0	0.976
1	0.983

All labels

PR AUC	0.988
ROC AUC	0.988
Log loss	0.127
Micro-average F1	0.96363634
Macro-average F1	0.9618188
Micro-average precision	96.4%
Micro-average recall	96.4%

Figure 19: Final model Confusion matrix



Confusion matrix

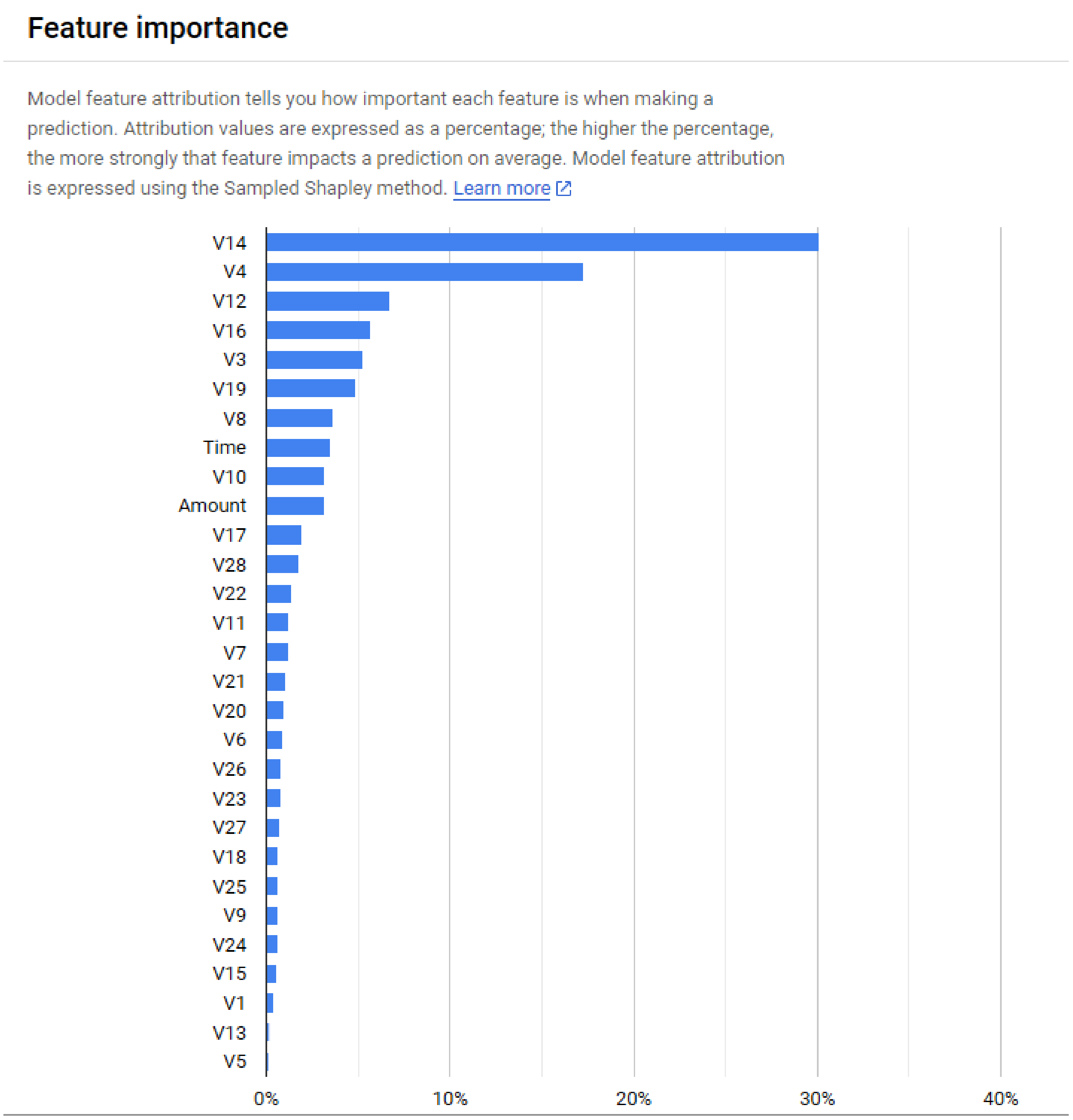
A confusion matrix shows how the model classified each label in the evaluation dataset. The blue, bold cells indicate a correct prediction. A data item is moved to the dropped column if it does not meet the confidence threshold for any label.

True label \ Predicted label	0	1	Dropped
0	100%	0%	0%
1	9%	91%	0%

We see by the presented confusion matrix that the final model (provided by AutoML) predicted correctly all non fraud cases and 91 % of the fraud cases in the train set. This is a classic symptom of overfitting the data.

However, given time restrictions we will keep this model for the purposes of this Demo.

Figure 20: Final Model Feature Importance

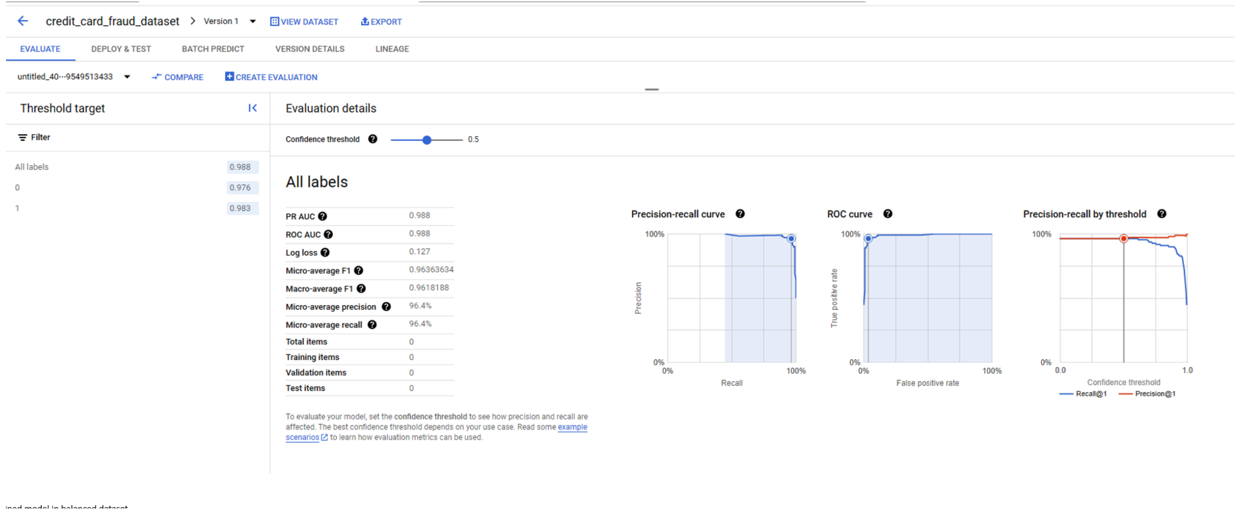


Feature importance

Model feature attribution tells you how important each feature is when making a prediction. Attribution values are expressed as a percentage; the higher the percentage, the more strongly that feature impacts a prediction on average. Model feature attribution is expressed using the Sampled Shapley method. Learn more

From the figure above we see that feature V14 was the most important feature in the explanation of response variable Class.

Figure 21: Final model Precision-recall, ROC and Precision-recall by threshold curves.



credit_card_fraud_dataset > Version 1

Threshold target

All labels	0.988
0	0.976
1	0.983

Evaluation details

Confidence threshold 0.5

All labels

PR AUC	0.988
ROC AUC	0.988
Log loss	0.127
Micro-average F1	0.96363634
Macro-average F1	0.9618188
Micro-average precision	96.4%
Micro-average recall	96.4%
Total items	0
Training items	0
Validation items	0
Test items	0

To evaluate your model, set the **confidence threshold** to see how precision and recall are affected. The best confidence threshold depends on your use case. Read some example scenarios to learn how evaluation metrics can be used.