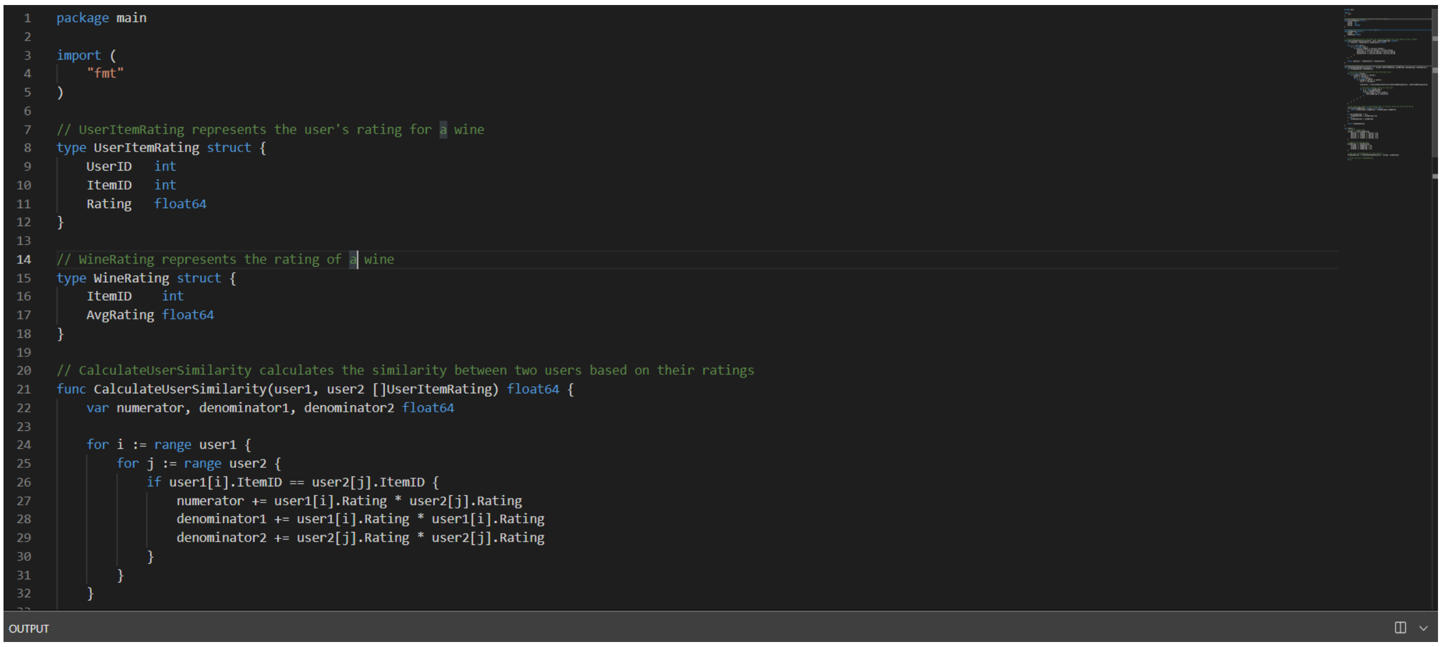
Weekly progress report – March 16 (6th report)

In the last series:

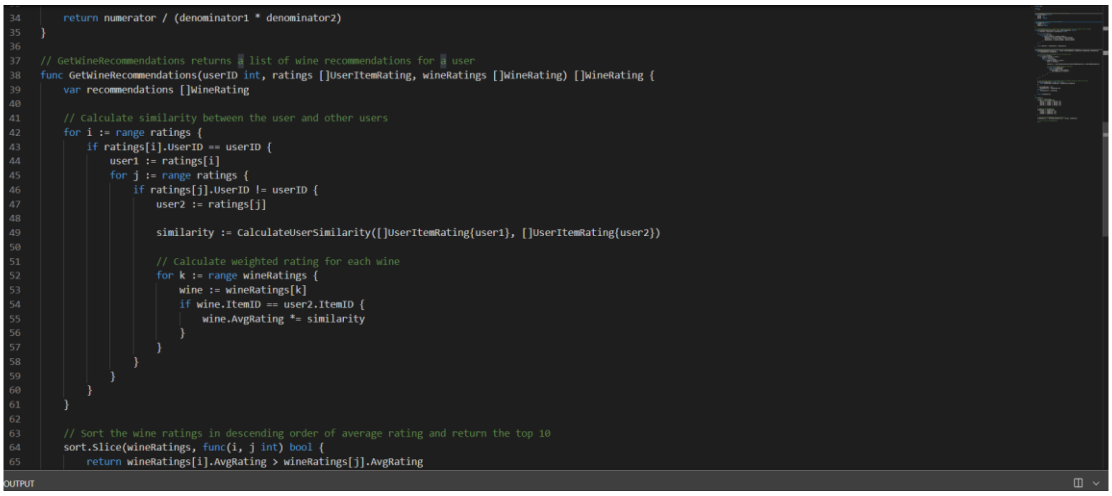
I stopped on 4th step while calculating users similarity. I think i'll continue it later and come back with trained ML models next week. See you in 6th report.

Done for this week: calculated users similarity

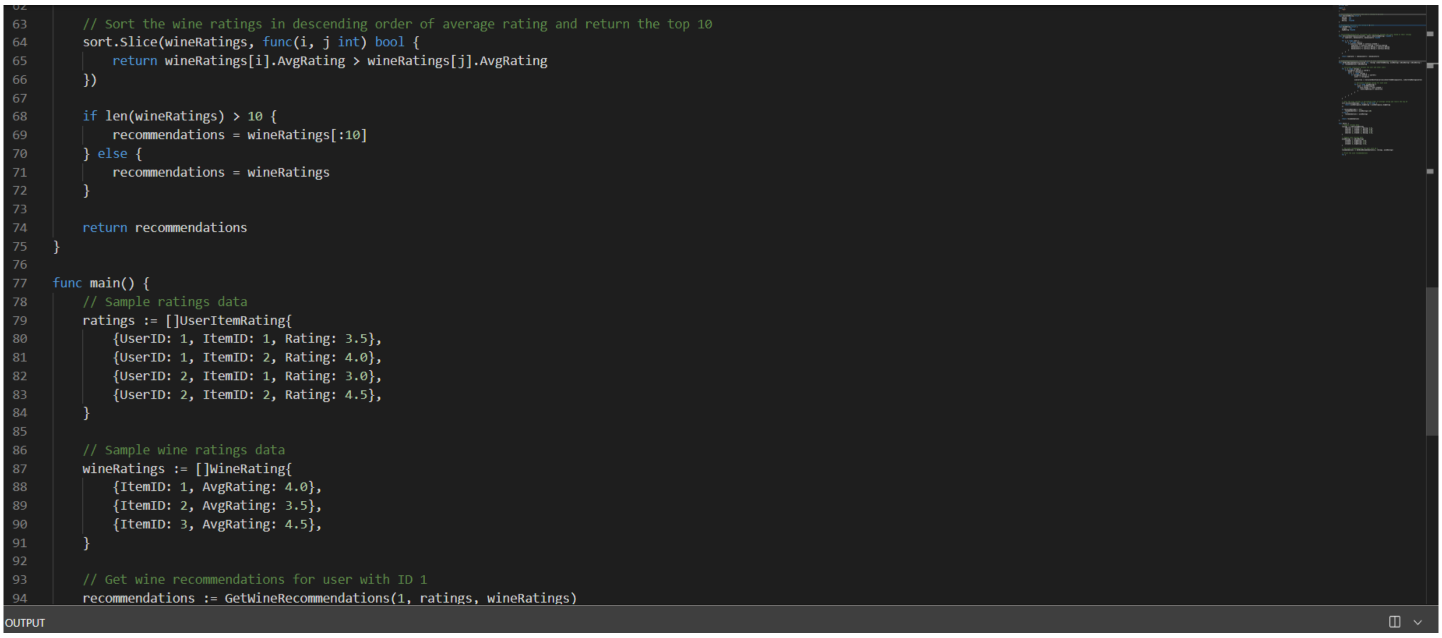
1. Recommendation Algorithm: need to choose a recommendation algorithm. There are several recommendation algorithms such as collaborative filtering, content-based filtering, and hybrid filtering. I I used collaborative filtering.



```go
package main

import (
    "fmt"
)

// UserItemRating represents the user's rating for a wine
type UserItemRating struct {
    UserID   int
    ItemID   int
    Rating   float64
}

// WineRating represents the rating of a wine
type WineRating struct {
    ItemID    int
    AvgRating float64
}

// CalculateUserSimilarity calculates the similarity between two users based on their ratings
func CalculateUserSimilarity(user1, user2 []UserItemRating) float64 {
    var numerator, denominator1, denominator2 float64

    for i := range user1 {
        for j := range user2 {
            if user1[i].ItemID == user2[j].ItemID {
                numerator += user1[i].Rating * user2[j].Rating
                denominator1 += user1[i].Rating * user1[i].Rating
                denominator2 += user2[j].Rating * user2[j].Rating
            }
        }
    }
```



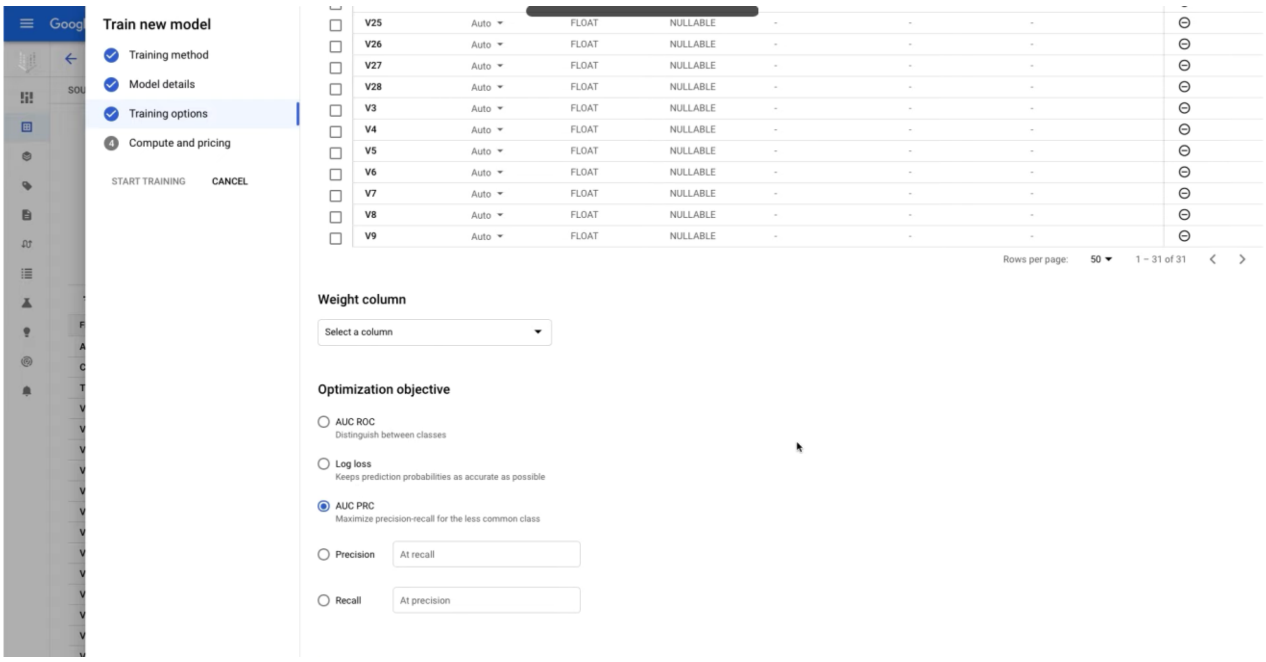
```go
    return numerator / (denominator1 * denominator2)
}

// GetWineRecommendations returns a list of wine recommendations for a user
func GetWineRecommendations(userID int, ratings []UserItemRating, wineRatings []WineRating) []WineRating {
    var recommendations []WineRating

    // Calculate similarity between the user and other users
    for i := range ratings {
        if ratings[i].UserID == userID {
            user1 := ratings[i]
            for j := range ratings {
                if ratings[j].UserID != userID {
                    user2 := ratings[j]

                    similarity := CalculateUserSimilarity([]UserItemRating{user1}, []UserItemRating{user2})

                    // Calculate weighted rating for each wine
                    for k := range wineRatings {
                        wine := wineRatings[k]
                        if wine.ItemID == user2.ItemID {
                            wine.AvgRating *= similarity
                        }
                    }
                }
            }
        }
    }

    // Sort the wine ratings in descending order of average rating and return the top 10
    sort.Slice(wineRatings, func(i, j int) bool {
        return wineRatings[i].AvgRating > wineRatings[j].AvgRating
```



```go
    // Sort the wine ratings in descending order of average rating and return the top 10
    sort.Slice(wineRatings, func(i, j int) bool {
        return wineRatings[i].AvgRating > wineRatings[j].AvgRating
    })

    if len(wineRatings) > 10 {
        recommendations = wineRatings[:10]
    } else {
        recommendations = wineRatings
    }

    return recommendations
}

func main() {
    // Sample ratings data
    ratings := []UserItemRating{
        {UserID: 1, ItemID: 1, Rating: 3.5},
        {UserID: 1, ItemID: 2, Rating: 4.0},
        {UserID: 2, ItemID: 1, Rating: 3.0},
        {UserID: 2, ItemID: 2, Rating: 4.5},
    }

    // Sample wine ratings data
    wineRatings := []WineRating{
        {ItemID: 1, AvgRating: 4.0},
        {ItemID: 2, AvgRating: 3.5},
        {ItemID: 3, AvgRating: 4.5},
    }

    // Get wine recommendations for user with ID 1
    recommendations := GetWineRecommendations(1, ratings, wineRatings)
```

OUTPUT

ML models on process



☐	V25	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V26	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V27	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V28	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V3	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V4	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V5	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V6	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V7	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V8	Auto	FLOAT	NULLABLE	-	-	-	⊖
☐	V9	Auto	FLOAT	NULLABLE	-	-	-	⊖

Rows per page: 50   1 – 31 of 31

Weight column

Select a column

Optimization objective

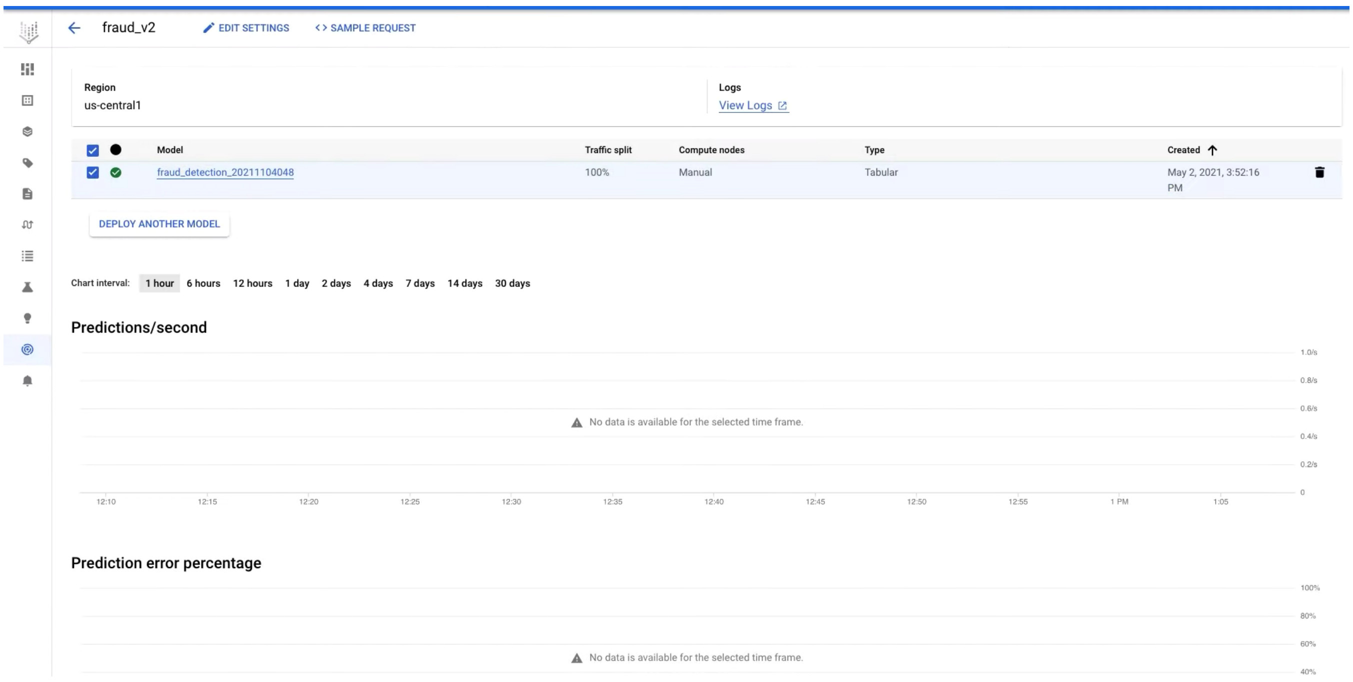
○ AUC ROC
Distinguish between classes

○ Log loss
Keeps prediction probabilities as accurate as possible

● AUC PRC
Maximize precision-recall for the less common class

○ Precision   At recall

○ Recall   At precision



Region
us-central1

Logs
View Logs

	Model	Traffic split	Compute nodes	Type	Created
✓	fraud_detection_20211104048	100%	Manual	Tabular	May 2, 2021, 3:52:16 PM

Chart interval: 1 hour 6 hours 12 hours 1 day 2 days 4 days 7 days 14 days 30 days

Predictions/second

No data is available for the selected time frame.

Prediction error percentage

No data is available for the selected time frame.

Next week i'll send the results of implementing a collaborative filtering algorithm for a wine recommendation engine