Τμήμα Εφαρμοσμένης Πληροφορικής
Ακαδημαϊκό έτος 2025-2026
Διδάσκουσα: Δρ. Δημαρά Ασημίνα

Ανάλυση Δεδομένων Μεγάλου Όγκου (Big Data)
Πρώτη Ομαδική Εργασία:
Flight Delay Analytics με Apache Spark

	Ον/μο	Αρ. Μητρώου
Μέλος 1ο	Παναγιωτίδης Θεόδωρος	ics22071
Μέλος 2ο	Μπλιώνα Αλίκη	ics23099

HELLENIC REPUBLIC
UNIVERSITY OF MACEDONIA

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. Εισαγωγή & Περιγραφή Δεδομένων 2
   1.1. Σκοπός της Εργασίας 2
   1.2. Περιγραφή & Καθαρισμός Δεδομένων 2
2. Αποτελέσματα Χρονομέτρησης 2
   2.1. Πίνακας Χρόνων: Spark RDD API 3
   2.2. Πίνακας Χρόνων: Spark DataFrame API 3
3. Συγκριτική Ανάλυση RDD vs DataFrame (Θέμα 3) 3
   3.1.Ευκολία Υλοποίησης 3
   3.2. Χρόνος Εκτέλεσης 4
   3.3. Εκφραστικότητα 4
   3.4. Δυνατότητες βελτιστοποίησης (Catalyst Optimizer) 4
4. Παρουσίαση Γραφικών Αποτελεσμάτων 5
   4.1. Γράφημα Top-10 Διαδρομών (Θέμα 4) 5
   4.2. Γράφημα Μέσης Καθυστέρησης ανά Ώρα (Θέμα 5) 5
5. Συμπεράσματα 7
   5.1. Τεχνική Σύγκριση & Απόδοση (θέμα 3) 8
   5.2. Πηγές και εργαλεία 8

1. Εισαγωγή & Περιγραφή Δεδομένων

1.1. Σκοπός της Εργασίας

Στην παρούσα εργασία αξιολογήσαμε συγκριτικά δύο APIs του Apache Spark: του Resilient Distributed Dataset (RDD) API και του DataFrame API. Ο τρόπος με τον οποίο πραγματοποιήσαμε την αξιολόγηση, είναι μέσα από την επίλυση του προβλήματος ανάλυσης δεδομένων (εύρεση μέσης καθυστέρησης πτήσεων) όπου εστιάσαμε, όπως παρουσιάζουμε σε επόμενη ενότητα, στην ευκολία υλοποίησης και αποδοτικότητας κάθε μιας προσέγγισης, χρησιμοποιώντας περιβάλλον PySpark. Εργαστήκαμε πάνω στο κομμάτι του κώδικα μέσω Google Colab.

1.2. Περιγραφή & Καθαρισμός Δεδομένων

Dataset: Χρησιμοποιήσαμε το αρχείο flights_2000.csv, το οποίο περιέχει ιστορικά δεδομένα πτήσεων.
Βασικοί Καθαρισμοί: Για να εξασφαλίσουμε την αξιοπιστία της ανάλυσης, εφαρμόσαμε τα ακόλουθα φίλτρα/καθαρισμούς σε όλα τα σενάρια (RDD και DataFrame):

- **Φιλτράρισμα Ακυρωμένων Πτήσεων:** Αποκλείσαμε όλες τις εγγραφές όπου η στήλη CANCELED ήταν ίση με 1.
- **Χειρισμός Κενών Τιμών:** Αφαιρέσαμε τις γραμμές με μη διαθέσιμες τιμές στις κρίσιμες στήλες: DEP_DELAY (Καθυστέρηση Αναχώρησης), ORIGIN_AIRPORT, DEST_AIRPORT, και SCHED_DEP.

2. Αποτελέσματα Χρονομέτρησης

Σε αυτή την ενότητα, παρουσιάζουμε τα αποτελέσματα των πέντε επαναληπτικών εκτελέσεων για κάθε API, καθώς και τον τελικό μέσο όρο των τριών ενδιάμεσων τιμών (αφαιρώντας τη μέγιστη και την ελάχιστη).

2.1. Πίνακας Χρόνων: Spark RDD API

Εκτέλεση	Συνολικός χρόνος (sec)	Χρόνος Πρώτου Action (sec)
Run 1	6.7539756298065186	1.2808096408843994
Run 2	2.9461259841918945	0.7913703918457031
Run 3	1.8939969539642334	0.6277713775634766

Run 4	2.0272767543792725	0.7611854076385498
Run 5	1.9199185371398926	0.6695282459259033
Μέσος Όρος (3 runs)	2.2978 δευτ.	0.7407 δευτ.

2.2. Πίνακας Χρόνων: Spark DataFrame API

Εκτέλεση	Συνολικός χρόνος (sec)	Χρόνος Πρώτου Action (sec)
Run 1	1.2807269096374512	0.6118130683898926
Run 2	1.779979944229126	0.9208300113677979
Run 3	1.7033429145812988	0.6779623031616211
Run 4	1.0217230319976807	0.509432315826416
Run 5	0.7245790958404541	0.35425448417663574
Μέσος Όρος (3 runs)	1.3353s	0.5997s

3. Συγκριτική Ανάλυση RDD vs DataFrame (Θέμα 3)

3.1.Ευκολία Υλοποίησης

Η χρήση του DataFrame API οδηγεί σε κώδικα που είναι πιο εύκολος στη συγγραφή, λιγότερο επιρρεπής σε λάθη και επιτρέπει να επικεντρωνόμαστε στη λογική της ανάλυσης. Σε αυτό το συμπέρασμα καταλήξαμε κατασκευάζοντας πίνακα σύγκρισης μεταξύ των δύο APIs:

	RDD API	**DataFrame API**
Ασφάλεια Τύπων	Μη τυποποιημένο, δηλαδή χειρίζεται γενικά Python objects και δε γνωρίζει τη σημασία των πεδίων.	Τυποποιημένο και στηλοειδές. Το Spark γνωρίζει τον τύπο δεδομένων κάθε στήλης άρα επιτρέπει βελτιστοποιήσεις μνήμης και εντοπίζει σφάλματα τύπων.
Parsing/Καθαρισμός	Χειροκίνητο parsing συμβολοσειρών .map(lambda line: line.split(',')). Επιρρεπές σε λάθη και χρονοβόρο	Χάρη στο Schema διαβάζει αυτόματα το CSV, αναγνωρίζει τις στήλες και μετατρέπει τους τύπους δεδομένων. Ο καθαρισμός κενών τιμών γίνεται με χρήση συνάρτησης na.drop().

3.2. Χρόνος Εκτέλεσης

Το DataFrame API εμφανίζει σταθερά χαμηλότερο μέσο Συνολικό Χρόνο Εκτέλεσης σε σχέση με το RDD API. Αυτή η διαφορά είναι αποτέλεσμα των μηχανισμών βελτιστοποίησης που χρησιμοποιεί το DataFrame.

Αναλυτικότερα, ο τρόπος που πραγματοποιήσαμε τη χρονομέτρηση και καταλήξαμε στο συγκεκριμένο συμπέρασμα είναι:

Βασιστήκαμε στην αρχή Lazy Evaluation, όπου το Spark δεν εκτελεί τους μετασχηματισμούς τη στιγμή που ορίζονται, αλλά κατασκευάζει ένα Λογικό Πλάνο Εκτέλεσης (DAG). Η πραγματική εκτέλεση πυροδοτείται μόνο όταν κληθεί μια ενέργεια.

Για τον λόγο αυτό, επιλέξαμε να χρονομετρείται αποκλειστικά το Action που υλοποιεί ολόκληρο τον υπολογιστικό κύκλο τόσο στο κομμάτι των RDDs όσο και σε αυτό των DataFrames.

Συγκεκριμένα:

- Στο RDD API η χρονομέτρηση τοποθετήθηκε αυστηρά γύρω από την εντολή take(10), διότι αυτή είναι το Action που αναγκάζει το Spark να εκτελέσει το DAG από την αρχή (ανάγνωση από δίσκο) έως το τέλος, προκειμένου να επιστρέψει τα 10 κορυφαία αποτελέσματα στον Driver. Συνεπώς, ο χρόνος που μετράται αντιπροσωπεύει τον πραγματικό Συνολικό Χρόνο επεξεργασίας
- Στο DataFrame API το χρονόμετρο τοποθετήθηκε γύρω από την εντολή write.csv() καθώς αυτή είναι το Action που ενεργοποιεί τον Catalyst Optimizer για να εκτελέσει το βελτιστοποιημένο φυσικό πλάνο.

3.3. Εκφραστικότητα

Η Δηλωτική φύση του DataFrame API κάνει τον κώδικα πολύ πιο συνοπτικό, ευανάγνωστο και ευκολότερο στη συντήρηση σε σχέση με τις χαμηλού επιπέδου lambda συναρτήσεις των RDDs.

	RDD API	**DataFrame API**
Προσέγγιση	Επιτακτική	Δηλωτική
Στόχος	Ορίζουμε το πώς θα γίνει ο υπολογισμός	Ορίζουμε τι αποτέλεσμα θα έχουμε.
Παράδειγμα της άσκησης	Για τον υπολογισμό μέσου όρου, χρησιμοποιούμε την combineByKey για να ορίσουμε τον τρόπο υπολογισμού του (Άθροισμα, Πλήθος)	Χρησιμοποιούμε το aggregation: avg_delays_df . (υψηλού επιπέδου)

Δομή	Δεν υπάρχει δομή. Τα δεδομένα είναι απλά σειρές (RDDs) ή ζεύγη (key, value)	Τα δεδομένα οργανώνονται σε στήλες με συγκεκριμένο Schema

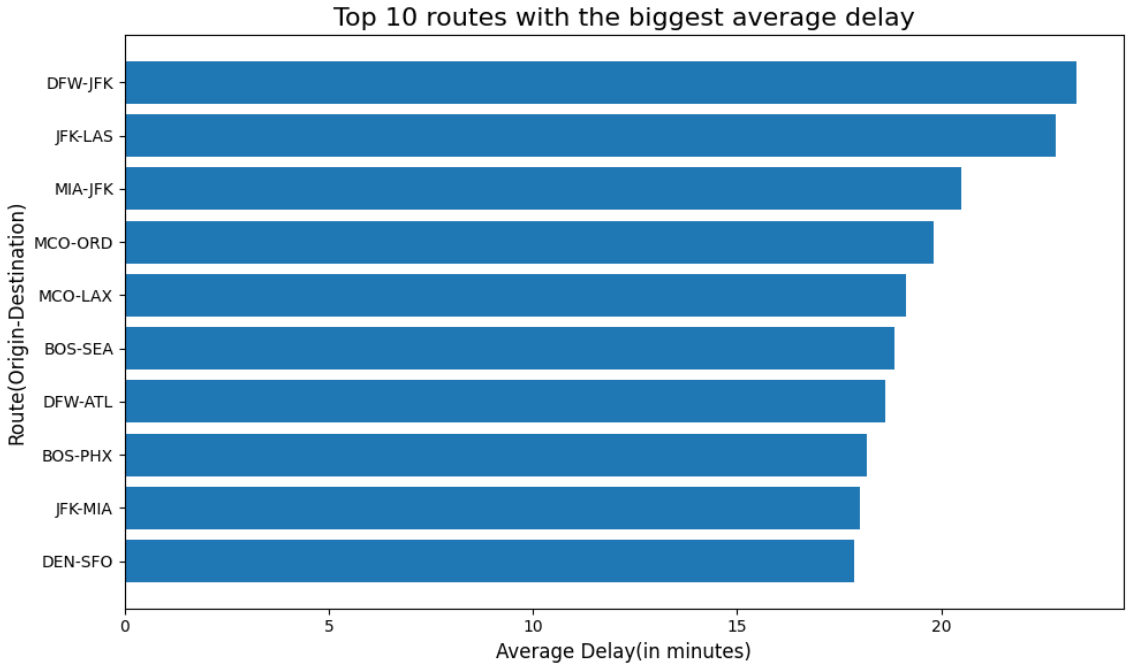
3.4. Δυνατότητες βελτιστοποίησης (Catalyst Optimizer)

Το κύριο ανταγωνιστικό πλεονέκτημα του DataFrame έναντι του RDD είναι ο Catalyst Optimizer. Δηλαδή:

- Το **RDD API** δεν έχει εσωτερική γνώση της δομής των δεδομένων, άρα οι βελτιστοποιήσεις είναι περιορισμένες.
- Το **DataFrame API** δημιουργεί ένα Λογικό Σχέδιο για κάθε query. Ο Catalyst Optimizer, βλέποντας το σχέδιο, βελτιώνει τη σειρά των λειτουργιών. Στη συνέχεια, δημιουργεί το βέλτιστο Φυσικό Σχέδιο για εκτέλεση. Αυτή η διαδικασία εξηγεί γιατί το DataFrame είναι θεμελιωδώς ταχύτερο.

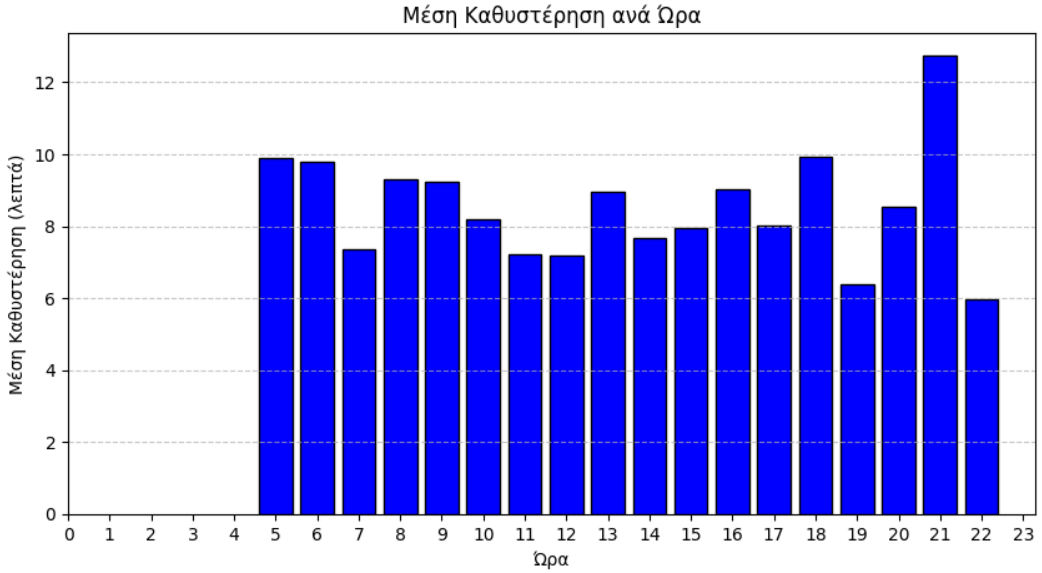
4. Παρουσίαση Γραφικών Αποτελεσμάτων

4.1. Γράφημα Top-10 Διαδρομών (Θέμα 4)

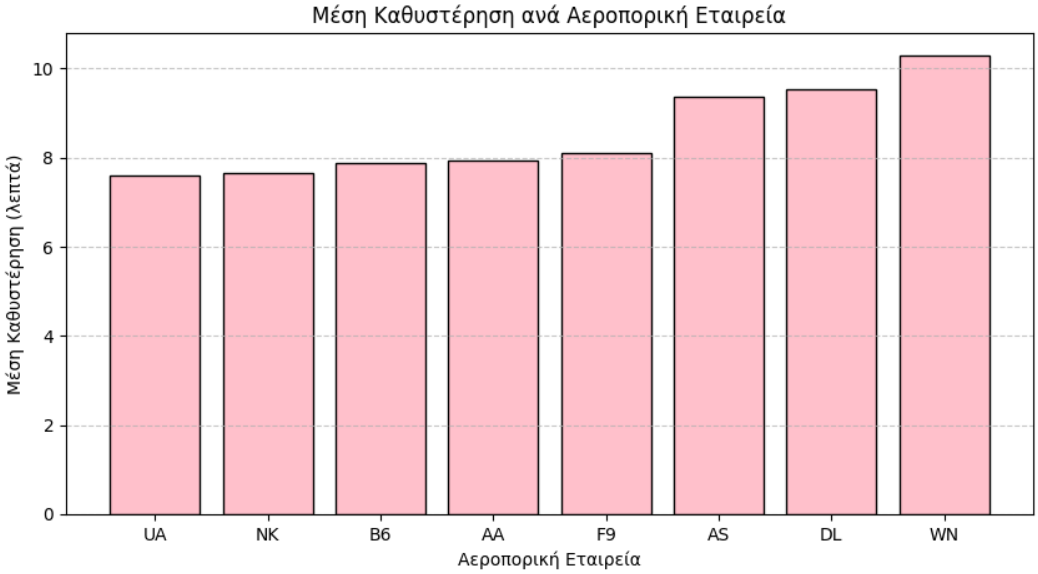


Όπως φαίνεται στο παραπάνω γράφημα, η ανάλυση των δεδομένων πτήσεων ανέδειξε τις 10 διαδρομές με τη μεγαλύτερη μέση καθυστέρηση αναχώρησης. Παρατηρούμε ότι η διαδρομή DFW-JFK έχει τη χειρότερη επίδοση με μέση καθυστέρηση 23,3 λεπτά. Είναι ενδιαφέρον ότι οι καθυστερήσεις κυμαίνονται από 17 έως 23,3 λεπτά, υποδεικνύοντας συγκεκριμένα προβλήματα σε αυτές τις συνδέσεις.

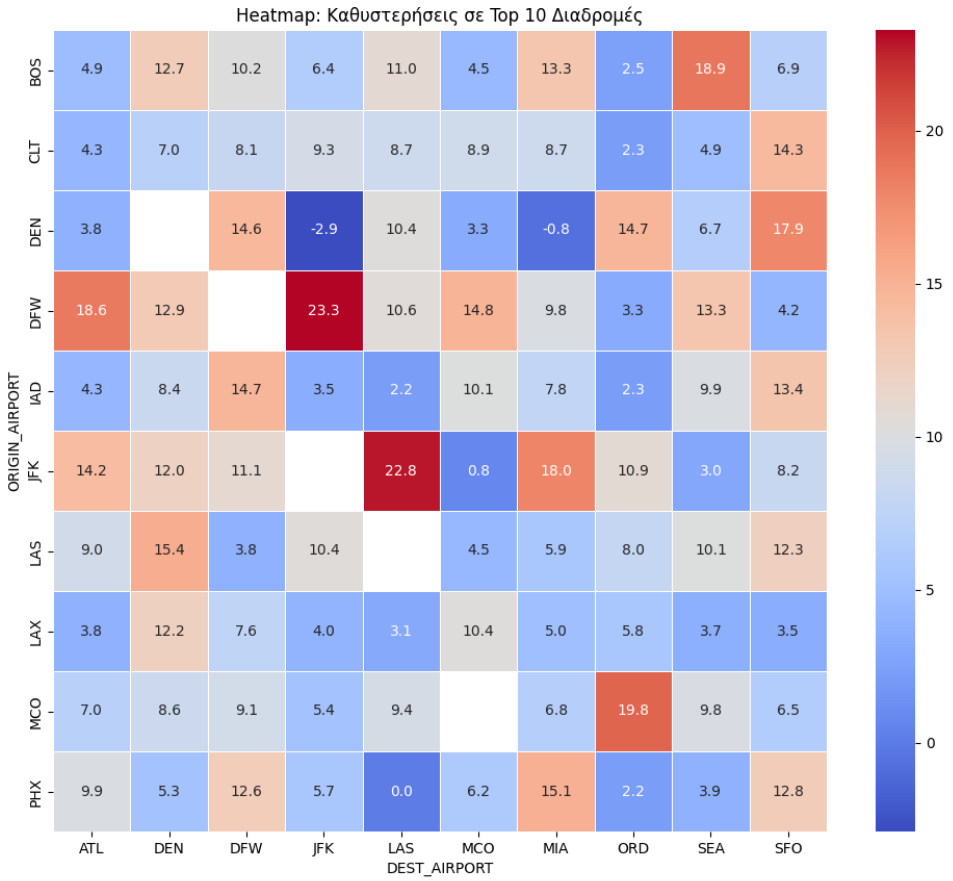
4.2. Γράφημα Μέσης Καθυστέρησης ανά Ώρα (Θέμα 5)



Ανάλυση ανά Ώρα: Όπως φαίνεται στο γράφημα, οι καθυστερήσεις αυξάνονται σταδιακά κατά τη διάρκεια της ημέρας, με τις βραδινές πτήσεις (μετά τις 18:00) να εμφανίζουν τη μεγαλύτερη ασυνέπεια.



Ανάλυση ανά Αεροπορική: Το γράφημα συγκρίνει την απόδοση των εταιρειών. Παρατηρούμε σημαντικές αποκλίσεις, με ορισμένες εταιρείες (UA) να έχουν πολύ μικρότερο μέσο χρόνο καθυστέρησης σε σχέση με άλλες (WN).



Heatmap Διαδρομών: Το Heatmap απεικονίζει τις καθυστερήσεις μεταξύ των 10 πιο πολυσύχναστων αεροδρομίων. Τα θερμά χρώματα (κόκκινο) υποδεικνύουν ζεύγη αεροδρομίων με συστηματικά προβλήματα (π.χ. η διαδρομή DFW -> JFK), ενώ τα μπλε χρώματα δείχνουν ομαλές πτήσεις.

5. Συμπεράσματα

5.1. Τεχνική Σύγκριση & Απόδοση (θέμα 3)

Με βάση τις ποσοτικές μετρήσεις και την ποιοτική ανάλυσης που πραγματοποιήσαμε: Το **Spark DataFrame API** αποδείχθηκε η **καταλληλότερη** προσέγγιση για την ανάλυση μεγάλων δεδομένων. Το DataFrame API προσέφερε μικρότερους χρόνους εκτέλεσης, χάρη στον Catalyst Optimizer του Spark που βελτιστοποιεί αυτόματα τα πλάνα εκτέλεσης. Επιπλέον, η δηλωτική σύνταξη των DataFrames έκανε τον κώδικα πιο ευανάγνωστο, και ευκολότερο στη συντήρηση σε σχέση με την πολυπλοκότητα των lambda functions στα RDDs. Συνεπώς, για δομημένα δεδομένα, τα DataFrames αποτελούν τη βέλτιστη επιλογή.

5.2. Πηγές και εργαλεία

Στο πλαίσιο της εργασίας, χρησιμοποιήθηκαν:

- *Διαλέξεις & Σημειώσεις Μαθήματος*: Κατανόηση της θεωρίας των RDDs και του τρόπου λειτουργίας του Spark Context.
- *Apache Spark Documentation*: Χρησιμοποιήθηκε για την τεκμηρίωση των μεθόδων combineByKey, reduceByKey και των παραμέτρων του write.csv.
- *Generative AI (ChatGPT / Gemini):* Χρησιμοποιήθηκε ως βοηθός προγραμματισμού με τους εξής τρόπους:
  1) Debugging: Για την επεξήγηση μηνυμάτων σφάλματος κατά την εκτέλεση στο Colab.
  2) Syntax Reference: Για τη σύνταξη εντολών της βιβλιοθήκης matplotlib και seaborn (στο Θέμα 4 & 5), ώστε να παραχθούν τα γραφήματα.
  3) Code Optimization: Για προτάσεις βελτίωσης της δομής του κώδικα, ειδικά στη διαχείριση των imports και στη δημιουργία των συναρτήσεων χρονομέτρησης.
  4) Θεωρητική Επεξήγηση: Για την καλύτερη κατανόηση της έννοιας "Lazy Evaluation" και του ρόλου του Catalyst Optimizer στη σύγκριση RDD vs DataFrame.

Τεχνολογικά Μέσα και Διαχωρισμός Εργασιών:

1. Το πρώτο μέλος (ics23099) ανέλαβε το κομμάτι των RDDs και την επέκταση κώδικα (θέμα 1 και θέμα 5). Αξιοποίησε φορητό υπολογιστή με CPU: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (2.42 GHz), Sockets: 1, Cores: 4, Logical Processors: 8. 8GB RAM. GPU: Intel(R) Iris(R) Xe Graphics.

2. Το δεύτερο μέλος (ics22071) ανέλαβε το τμήμα των DataFrames και την Οπτικοποίηση (θέμα 2 και θέμα 4). Αξιοποίησε φορητό υπολογιστή με CPU: AMD Rysen 5 3500U with Radeon Vega Mobile Gfx,2100 Mhz, 4 Core(s), 8 Logical Processor(s). 12GB RAM. GPU: AMD Radeon(TM) Vega 8 Graphics.
3. Η κοινή εργασία αφορά το κομμάτι της συγγραφής της τελικής αναφοράς και τη σύγκριση των δύο μοντέλων που απαιτεί το θέμα 3 της άσκησης. Η σύνθεση του κώδικα έγινε σε ένα κοινό Google Colab notebook.