BIRMINGHAM CITY
University

CMP6230 - Assignment 2

Data Management and MLOps Activities Log Report

Lewis Higgins - Student ID 22133848

CMP6230 - Data Management and Machine Learning Operations

Module Coordinator: Sima Iranmanesh

Contents

Introduction

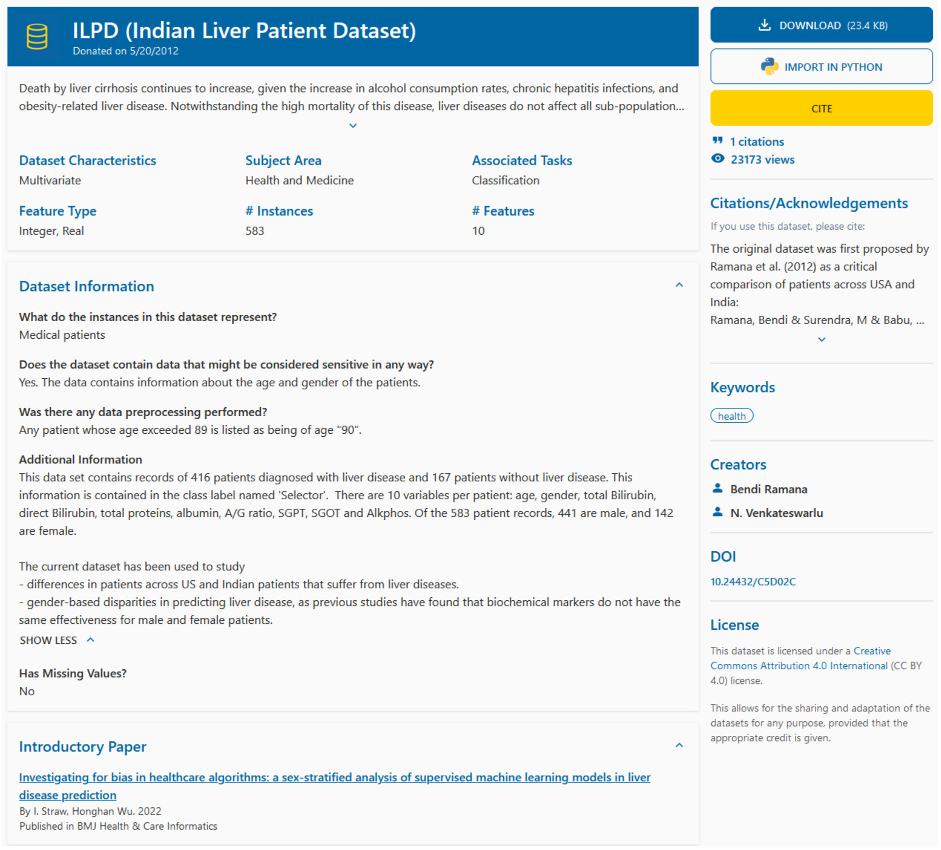
This report aims to design and document a five-stage Machine Learning Operations pipeline, from the initial data ingestion to the monitoring of the produced model, beginning with the identification of three candidate datasets for the pipeline to be conducted with, and continuing into descriptions of what each stage of the pipeline consists of, and how it will be implemented with the one chosen dataset using a wide variety of software that is also used within industry. When this plan is implemented at a later stage, it will provide a clear framework to create an automated MLOps pipeline with continuous integration and deployment.

Candidate Data Sources

For the first stage of the pipeline, data ingestion, three data sources will be identified in order to find the one that would be most optimal for the production and deployment of a machine learning model to complete a supervised learning task.

1.1 Candidate 1 - Indian Liver Patient Dataset

This dataset (Bendi Ramana and N. Venkateswarlu, 2022) consists of real data sourced from hospitals northeast of Andhra Pradesh in India. It was obtained from the UCI Machine Learning Repository, and has been previously used by Straw and Wu (2022) in their analysis of sex-related bias in supervised learning models. The UCI ML Repository is a popular host of datasets used by students, educators and researchers worldwide for machine learning (UCI Machine Learning Repository, 2024), and hosts these datasets on the cloud for public download and usage, as long as credit is given.



ILPD (Indian Liver Patient Dataset)

Donated on 5/20/2012

Death by liver cirrhosis continues to increase, given the increase in alcohol consumption rates, chronic hepatitis infections, and obesity-related liver disease. Notwithstanding the high mortality of this disease, liver diseases do not affect all sub-population...

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Health and Medicine	Classification
Feature Type	**# Instances**	**# Features**
Integer, Real	583	10

Dataset Information

What do the instances in this dataset represent?
Medical patients

Does the dataset contain data that might be considered sensitive in any way?
Yes. The data contains information about the age and gender of the patients.

Was there any data preprocessing performed?
Any patient whose age exceeded 89 is listed as being of age "90".

Additional Information
This data set contains records of 416 patients diagnosed with liver disease and 167 patients without liver disease. This information is contained in the class label named 'Selector'. There are 10 variables per patient: age, gender, total Bilirubin, direct Bilirubin, total proteins, albumin, A/G ratio, SGPT, SGOT and Alkphos. Of the 583 patient records, 441 are male, and 142 are female.

The current dataset has been used to study
- differences in patients across US and Indian patients that suffer from liver diseases.
- gender-based disparities in predicting liver disease, as previous studies have found that biochemical markers do not have the same effectiveness for male and female patients.

SHOW LESS

Has Missing Values?
No

Introductory Paper

Investigating for bias in healthcare algorithms: a sex-stratified analysis of supervised machine learning models in liver disease prediction
By I. Straw, Honghan Wu. 2022
Published in BMJ Health & Care Informatics

DOWNLOAD (23.4 KB)

IMPORT IN PYTHON

CITE

1 citations
23173 views

Citations/Acknowledgements

If you use this dataset, please cite:

The original dataset was first proposed by Ramana et al. (2012) as a critical comparison of patients across USA and India:
Ramana, Bendi & Surendra, M & Babu, ...

Keywords

health

Creators

Bendi Ramana
N. Venkateswarlu

DOI

10.24432/C5D02C

License

This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

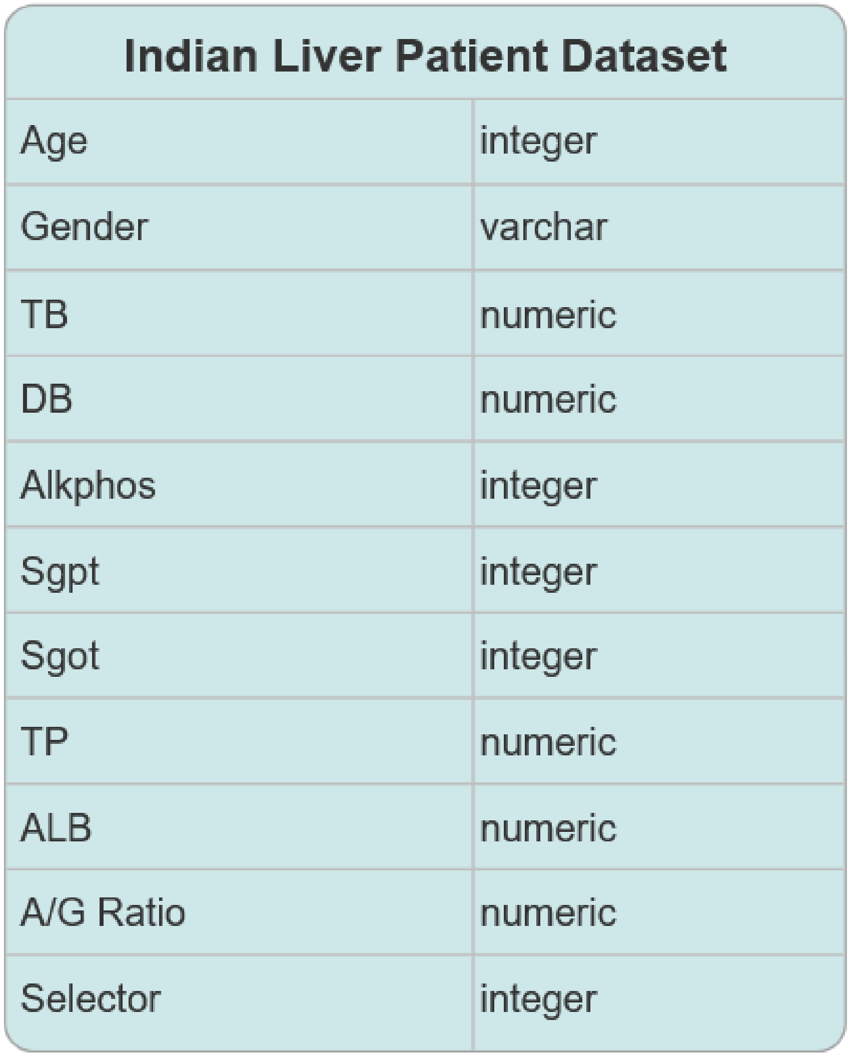
This allows for the sharing and adaptation of the datasets for any purpose, provided that the appropriate credit is given.

Figure 1.1: A snapshot of the dataset's UCI repository page.

This dataset in particular aims to assist in the diagnosis of liver disease due to increasing mortality rates from conditions like liver cirrhosis, and contains 584 records with 10 features as well as the "Selector" classification column, where those without liver disease are classed as 1, and those with liver disease are classed as 2. For the purposes of the ML model, these can be changed to 0 and 1 respectively. The dataset is a single flat-file Comma-Separated

Values (CSV) file, which stores data by separating each column with commas and each row with line breaks. This CSV file uses a One Big Table (OBT) schema, as seen in the entity relationship diagram in Figure 1.2, wherein all of the data within this dataset is stored in a single table. The OBT schema is a denormalised schema that is useful for simple querying due to there being no need for table joins. However, it is prone to data duplication and redundancy, which can increase necessary storage requirements.

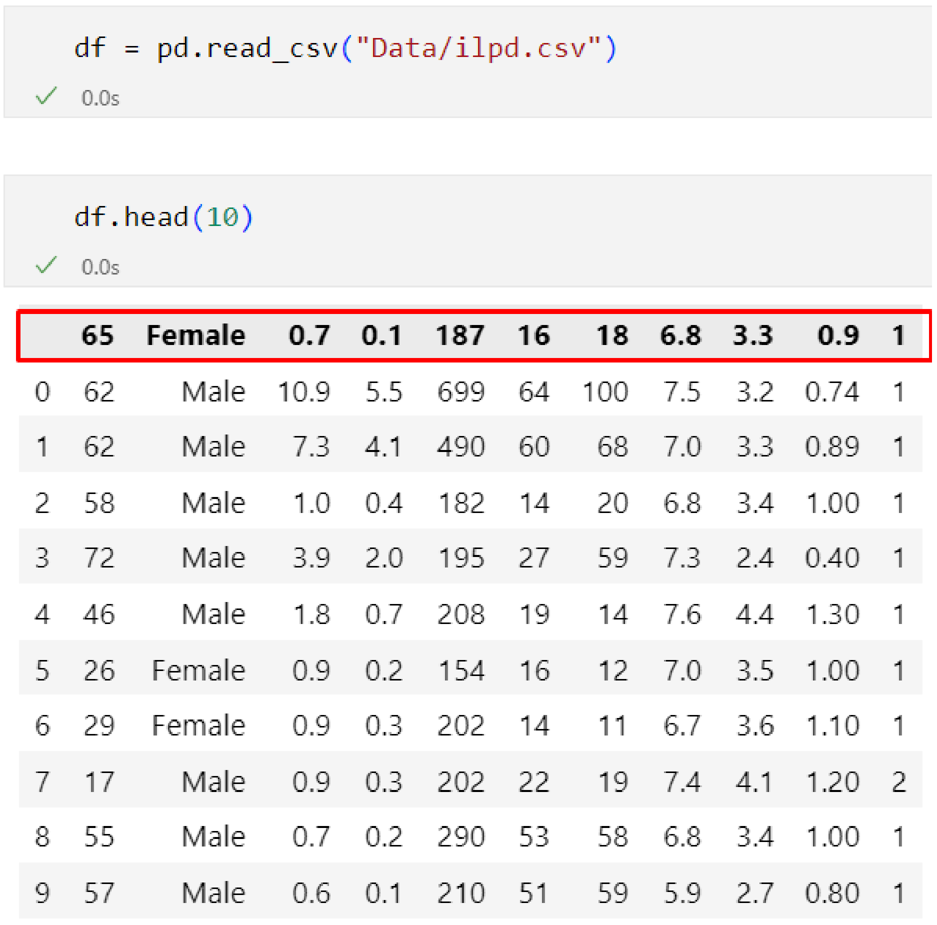
Descriptions of the columns in the dataset, as well as the associated data types, can be found in Table 1.1.



Indian Liver Patient Dataset	
Age	integer
Gender	varchar
TB	numeric
DB	numeric
Alkphos	integer
Sgpt	integer
Sgot	integer
TP	numeric
ALB	numeric
A/G Ratio	numeric
Selector	integer

Figure 1.2: An entity relationship diagram of the Indian Liver Patient Dataset.

A minor issue with this file is that it has no headers in its CSV file, meaning that when imported, Pandas will interpret the first row of data as the names of the columns, though this can be combated by adding the "names" argument when calling Pandas' "read_csv" function, seen below in Figure 1.3a.

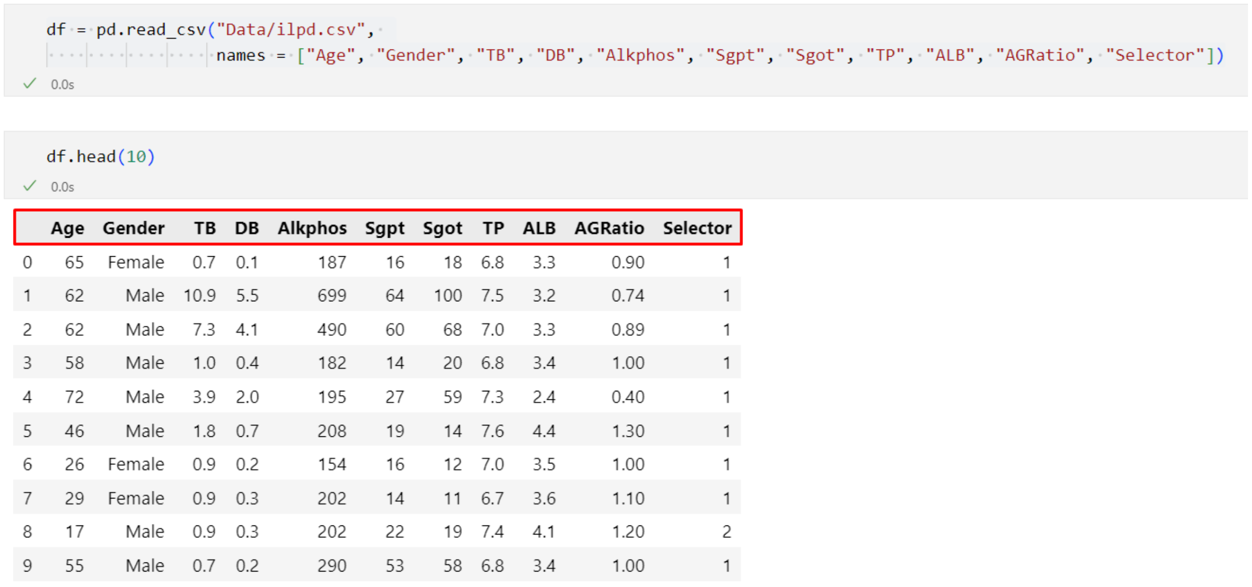


```
df = pd.read_csv("Data/ilpd.csv")
✓ 0.0s
```

```
df.head(10)
✓ 0.0s
```

	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.9	1
0	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
1	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
2	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
3	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1
4	46	Male	1.8	0.7	208	19	14	7.6	4.4	1.30	1
5	26	Female	0.9	0.2	154	16	12	7.0	3.5	1.00	1
6	29	Female	0.9	0.3	202	14	11	6.7	3.6	1.10	1
7	17	Male	0.9	0.3	202	22	19	7.4	4.1	1.20	2
8	55	Male	0.7	0.2	290	53	58	6.8	3.4	1.00	1
9	57	Male	0.6	0.1	210	51	59	5.9	2.7	0.80	1

(a) Importing without supplying column names.



```
df = pd.read_csv("Data/ilpd.csv",
                 names = ["Age", "Gender", "TB", "DB", "Alkphos", "Sgpt", "Sgot", "TP", "ALB", "AGRatio", "Selector"])
✓ 0.0s
```

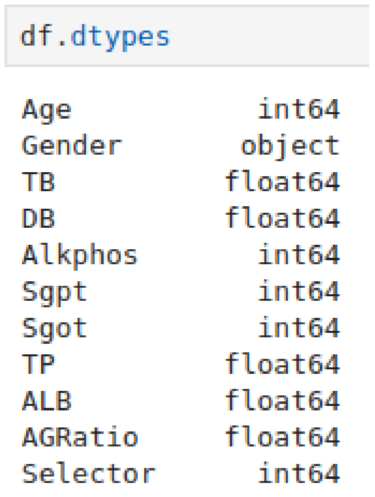
```
df.head(10)
✓ 0.0s
```

	Age	Gender	TB	DB	Alkphos	Sgpt	Sgot	TP	ALB	AGRatio	Selector
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1
5	46	Male	1.8	0.7	208	19	14	7.6	4.4	1.30	1
6	26	Female	0.9	0.2	154	16	12	7.0	3.5	1.00	1
7	29	Female	0.9	0.3	202	14	11	6.7	3.6	1.10	1
8	17	Male	0.9	0.3	202	22	19	7.4	4.1	1.20	2
9	55	Male	0.7	0.2	290	53	58	6.8	3.4	1.00	1

(b) Importing with the column names.

Figure 1.3: Importing and viewing the head of the erroneous CSV using Pandas. The column headers are highlighted in a red box.

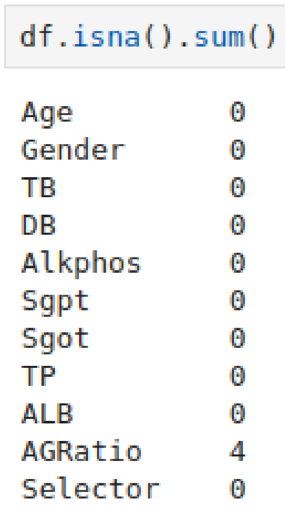
A preliminary analysis of the file to ascertain the data types of each column, seen in Figure 1.4, also revealed that there were 4 missing values in the A/G ratio column. It is possible that these missing values could be imputed rather than deleted, as it may be possible to calculate what the A/G ratio of these rows would have been in the Data Preprocessing stage of a pipeline.



```
df.dtypes
```

```
Age           int64
Gender       object
TB          float64
DB          float64
Alkphos       int64
Sgpt          int64
Sgot          int64
TP          float64
ALB         float64
AGRatio     float64
Selector      int64
```

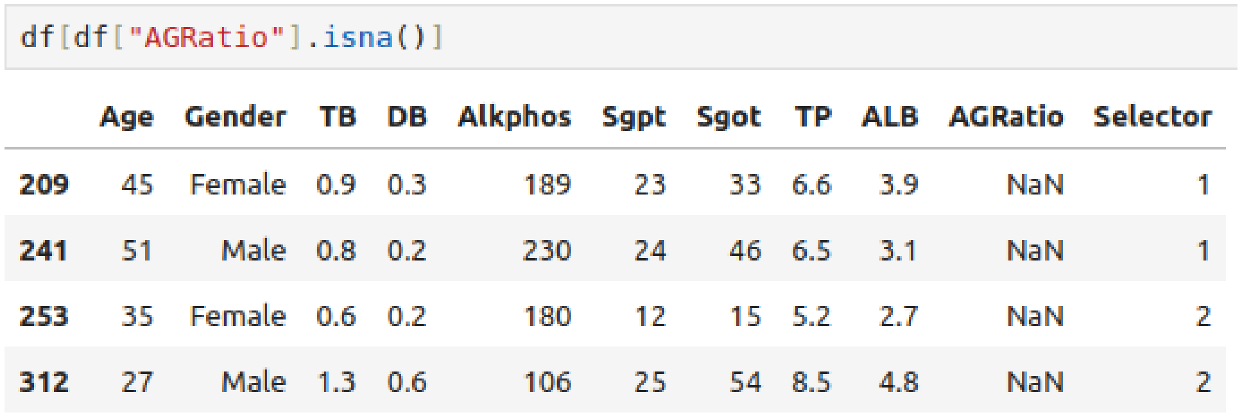
Figure 1.4: The data types of the Indian Liver Patient Dataset.



```
df.isna().sum()
```

```
Age         0
Gender      0
TB          0
DB          0
Alkphos     0
Sgpt        0
Sgot        0
TP          0
ALB         0
AGRatio     4
Selector    0
```

(a) Four missing values are identified.



```
df[df["AGRatio"].isna()]
```

	Age	Gender	TB	DB	Alkphos	Sgpt	Sgot	TP	ALB	AGRatio	Selector
209	45	Female	0.9	0.3	189	23	33	6.6	3.9	NaN	1
241	51	Male	0.8	0.2	230	24	46	6.5	3.1	NaN	1
253	35	Female	0.6	0.2	180	12	15	5.2	2.7	NaN	2
312	27	Male	1.3	0.6	106	25	54	8.5	4.8	NaN	2

(b) The four rows in question.

Figure 1.5: The identification of four missing values in the A/G ratio column.

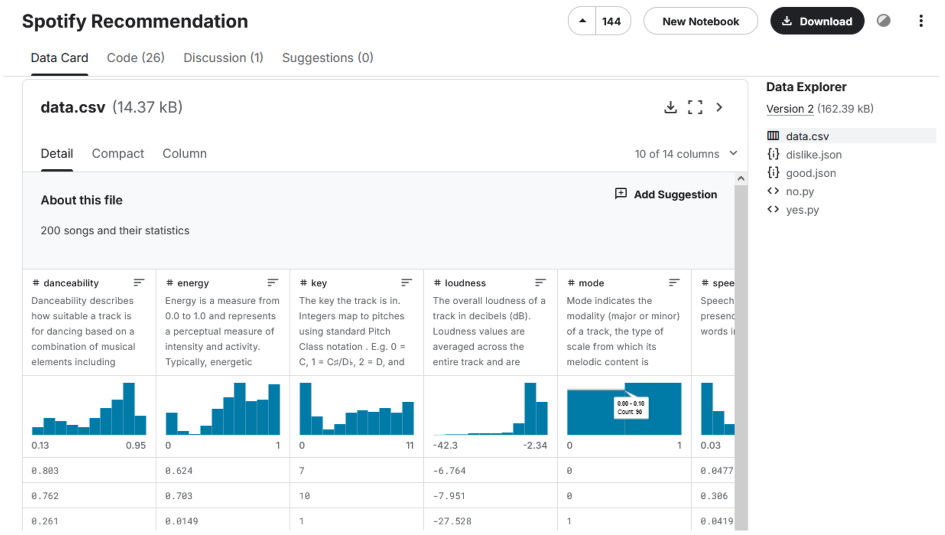
Column	Description	Measurement level
Age	The patient's age. **Ages of 90 or over were listed as 90 before this dataset was published, which could introduce a bias in the machine learning model.**	Ratio
Gender	The patient's gender, either "Male" or "Female".	Nominal
TB	Total bilirubin. Bilirubin is a substance produced by the liver, and a high presence of it may be indicative of liver problems (Mayo Clinic, 2024).	Ratio
DB	Direct bilirubin. This is a slightly different form of bilirubin that is formed after the liver has processed it.	Ratio
Alkphos	Levels of alkaline phosphate - an enzyme in the body produced by the liver. Too much may indicate liver disease. (Cleveland Clinic, 2024)	Ratio
Sgpt	Another enzyme found in the liver, where too much can indicate liver problems.	Ratio
Sgot	Levels of AST in the blood, where too much indicates liver problems.	Ratio
TP	Total proteins.	Ratio
ALB	Albumin - a protein in blood plasma. Too little of this may indicate liver problems.	Ratio
A/G Ratio	The ratio of albumin to globulin, which is another blood protein.	Ratio
Selector	The classifier, indicating if the person has liver disease. The target column for the ML model.	Nominal

Table 1.1: The descriptions of each column in the Indian Liver Patient Dataset.

This dataset could be used to solve a binary classification problem using the ten predictor variables and the ground truth Selector column, which will be used in measuring the accuracy of the model. There is a clear positive purpose for developing such a model; as previously mentioned, mortality rates from liver disease are high, and an early diagnosis that could leverage the power of machine learning can greatly enhance the odds of successful treatment.

1.2 Candidate 2 - Spotify Likes Dataset

This dataset (Vergnou, 2024) was sourced from Kaggle, a platform similar to the UCI ML repository in its purpose for students and researchers that acts as a search engine for datasets, but also allows its users to host competitions, upload their machine learning models, and also upload their own Python notebooks (Kaggle, 2024). This dataset is stored on their servers on the cloud, and is free to download and use under a Creative Commons Public Domain license (Creative Commons, 2024).



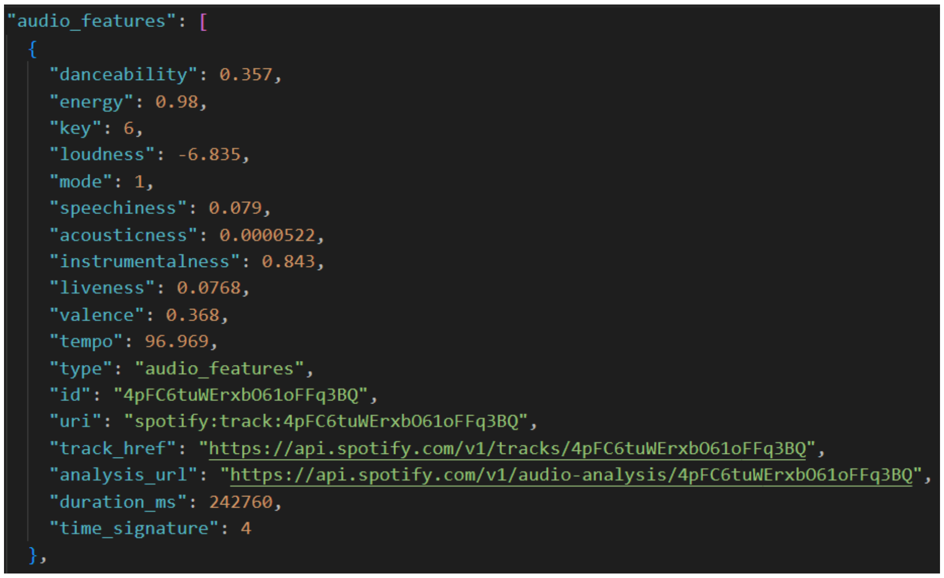
About this file

200 songs and their statistics

# danceability	# energy	# key	# loudness	# mode	# spee
Danceability describes how suitable a track is for dancing based on a combination of musical elements including	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic	The key the track is in. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is	Speech presenc words i
0.13 – 0.95	0 – 1	0 – 11	-42.3 – -2.34	0 – 1	0.03
0.803	0.624	7	-6.764	0	0.0477
0.762	0.703	10	-7.951	0	0.306
0.261	0.0149	1	-27.528	1	0.0419

Figure 1.6: A snapshot of the Spotify dataset's Kaggle page.

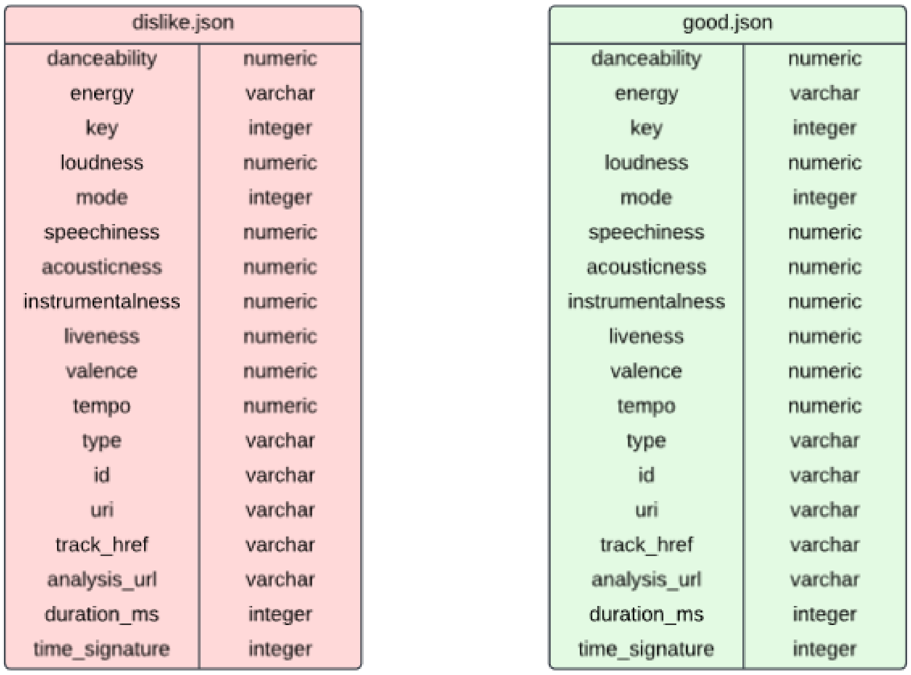
The data itself is split over two JavaScript Object Notation (JSON) files, but also fully present in an included CSV file, with all three utilising a One Big Table schema. The download also includes two Python files, which have the JSON data stored in Python dictionaries for ease of access, though these will not be used in this brief analysis. JSON files store data in **key-value pairs**, such as in the example snippet of this dataset depicted in Figure 1.7.



```
"audio_features": [
  {
    "danceability": 0.357,
    "energy": 0.98,
    "key": 6,
    "loudness": -6.835,
    "mode": 1,
    "speechiness": 0.079,
    "acousticness": 0.0000522,
    "instrumentalness": 0.843,
    "liveness": 0.0768,
    "valence": 0.368,
    "tempo": 96.969,
    "type": "audio_features",
    "id": "4pFC6tuWErxb061oFFq3BQ",
    "uri": "spotify:track:4pFC6tuWErxb061oFFq3BQ",
    "track_href": "https://api.spotify.com/v1/tracks/4pFC6tuWErxb061oFFq3BQ",
    "analysis_url": "https://api.spotify.com/v1/audio-analysis/4pFC6tuWErxb061oFFq3BQ",
    "duration_ms": 242760,
    "time_signature": 4
  },
```

Figure 1.7: A snippet of the JSON data, viewed in Visual Studio Code.

Every row in the JSON files is part of the single "audio_features" key, and each new row is separated by curly braces {}. Each column is then given as a key-value pair, such as the first row in Figure 1.7, where "danceability" is the key, and 0.352 is the associated value. This dataset does consist of real data, sourced from the author's personal liked songs directly via the Spotify API. There are 195 rows of data, with 100 liked songs, and 95 disliked songs. Liked and disliked songs are separated into two JSON files, named "dislike" and "good". The two JSON files have 18 features, as depicted in Figure 1.8.

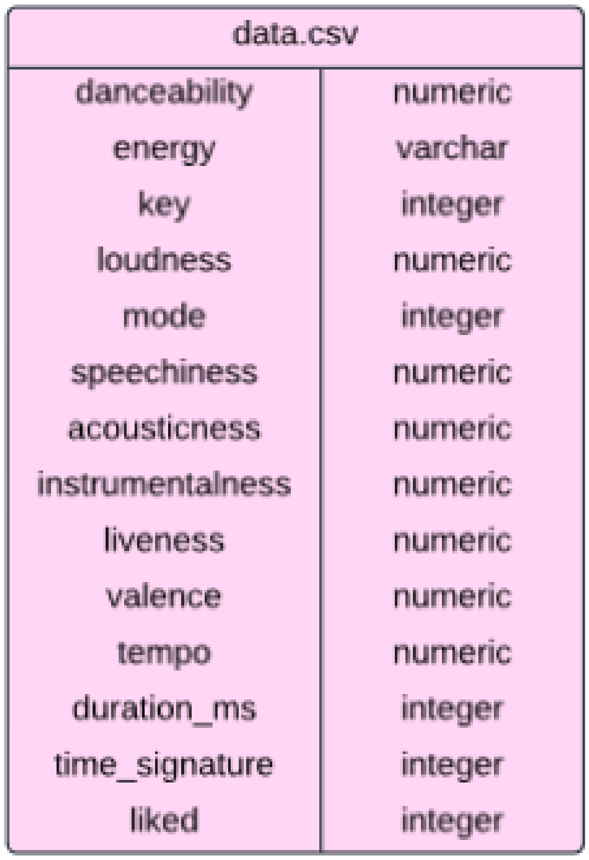


dislike.json	
danceability	numeric
energy	varchar
key	integer
loudness	numeric
mode	integer
speechiness	numeric
acousticness	numeric
instrumentalness	numeric
liveness	numeric
valence	numeric
tempo	numeric
type	varchar
id	varchar
uri	varchar
track_href	varchar
analysis_url	varchar
duration_ms	integer
time_signature	integer

good.json	
danceability	numeric
energy	varchar
key	integer
loudness	numeric
mode	integer
speechiness	numeric
acousticness	numeric
instrumentalness	numeric
liveness	numeric
valence	numeric
tempo	numeric
type	varchar
id	varchar
uri	varchar
track_href	varchar
analysis_url	varchar
duration_ms	integer
time_signature	integer

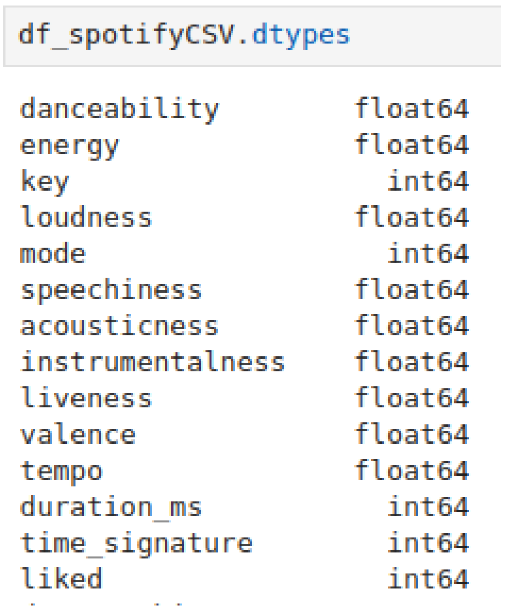
Figure 1.8: An entity relationship diagram of the two JSON files.

This dataset has been used to create machine learning models before, most notably by its own author, who has a public GitHub repository showcasing their work (Vergnou, 2021). Before publicising this data, however, the author had done some preprocessing of their own, having included the additional CSV file, produced as a result of merging the two JSON files into one CSV and removing unnecessary columns, as depicted in Figure 1.9. Therefore, my preliminary Pandas analysis of the data types and missing values will only be performed on this CSV, seen in Figures 1.10 and 1.12.



data.csv	
danceability	numeric
energy	varchar
key	integer
loudness	numeric
mode	integer
speechiness	numeric
acousticness	numeric
instrumentalness	numeric
liveness	numeric
valence	numeric
tempo	numeric
duration_ms	integer
time_signature	integer
liked	integer

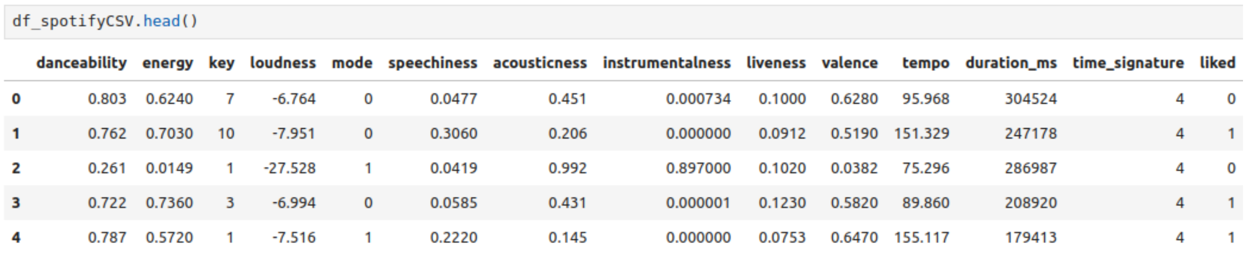
Figure 1.9: An entity relationship diagram of the author's preprocessed CSV file.



```
df_spotifyCSV.dtypes
```

```
danceability        float64
energy              float64
key                   int64
loudness            float64
mode                  int64
speechiness         float64
acousticness        float64
instrumentalness    float64
liveness            float64
valence             float64
tempo               float64
duration_ms           int64
time_signature        int64
liked                 int64
```

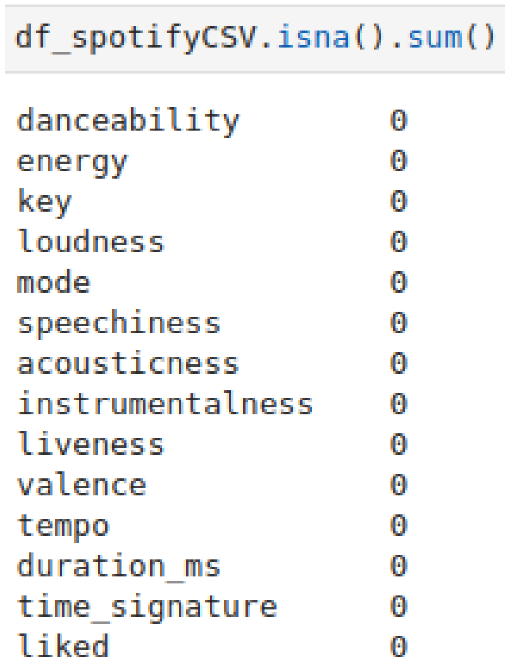
Figure 1.10: The data types of the Spotify Likes Dataset.



```
df_spotifyCSV.head()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	liked
0	0.803	0.6240	7	-6.764	0	0.0477	0.451	0.000734	0.1000	0.6280	95.968	304524	4	0
1	0.762	0.7030	10	-7.951	0	0.3060	0.206	0.000000	0.0912	0.5190	151.329	247178	4	1
2	0.261	0.0149	1	-27.528	1	0.0419	0.992	0.897000	0.1020	0.0382	75.296	286987	4	0
3	0.722	0.7360	3	-6.994	0	0.0585	0.431	0.000001	0.1230	0.5820	89.860	208920	4	1
4	0.787	0.5720	1	-7.516	1	0.2220	0.145	0.000000	0.0753	0.6470	155.117	179413	4	1

Figure 1.11: The head of the dataset.



```
df_spotifyCSV.isna().sum()

danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_ms         0
time_signature      0
liked               0
```

Figure 1.12: No missing values in the dataset.

While a machine learning model to solve a binary classification problem could be trained on this dataset to identify if the author would like a song, it has significantly less of a positive impact than Candidates 1 and 3, as this dataset is the author's subjective belief rather than objective fact that can be applied to other people. Nevertheless, the descriptions of each column can be found in Table 1.2.

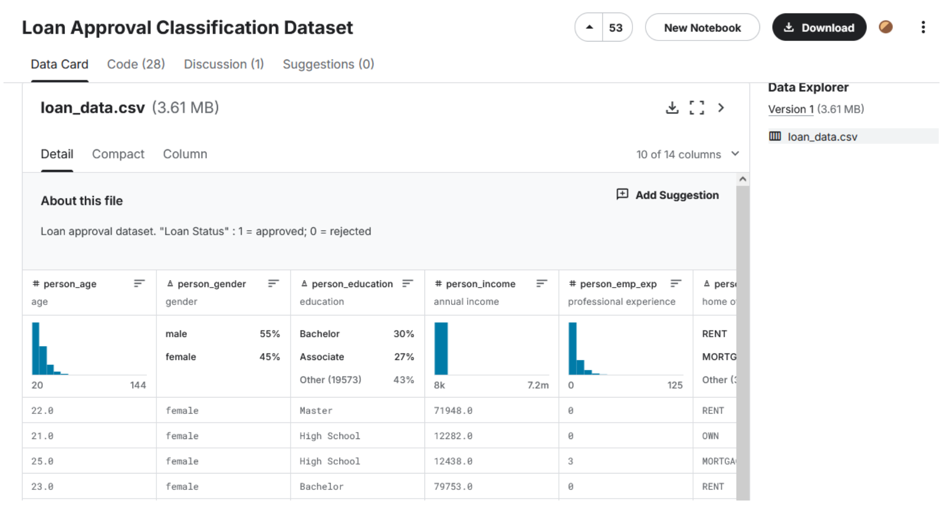
Column	Description	Measurement level
Danceability	How suitable a song is for dancing measured from 0.0 to 1.0.	Ratio
Energy	The intensity and activity of a song. For example, death metal is high energy, whereas classical music is low intensity. 1.0 is the most energetic.	Ratio
Key	The musical key the song is in, converted to an integer using standard pitch class notation.(Butterfield, 2024)	Ratio
Loudness	The averaged decibel volume of a song, typically between -60 and 0 dB.	Interval
Mode	Whether a song is in major or minor scale. 1 is major and 0 is minor.	Nominal
Speechiness	The calculated presence of spoken words in a song.	Ratio
Acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.	Ratio
Instrumentalness	A confidence measure from 0.0 to 1.0 of whether a song has no vocals.	Ratio
Liveness	A confidence measure from 0.0 to 1.0 of whether a live audience can be heard as part of a song.	Ratio
Valence	A confidence measure from 0.0 to 1.0 of the musical positiveness of a song.	Ratio
Tempo	The beats per minute of a song.	Ratio
Duration_MS	The duration of a song in milliseconds.	Ratio
Time signature	The estimated time signature of the song.	Ratio
Liked	The target variable, indicative of whether the author liked the song or not.	Ratio
Type	Always "audio_features". Not a relevant predictor.	Nominal
ID	Spotify's own unique ID for a song. Not a relevant predictor.	Nominal
URI	Spotify's URI for the song. Not a relevant predictor.	Nominal
Track HREF	A link to the song on Spotify's API. Not a relevant predictor.	Nominal
Analysis URL	A link to the song's audio analysis data. Not a relevant predictor.	Nominal

Table 1.2: The descriptions of each column in the Spotify songs dataset (Spotify, 2024). Red columns are only present in the CSV, whereas green columns are only present in the JSONs.

These measurements and their descriptions are sourced from Spotify's API, and are automatically calculated when songs are uploaded to the service. The ground truth of the dataset is present in the CSV file as the "liked" classifier column, and a train/test split can be implemented for predictions, which is aided by the fact that this dataset is well balanced (100 liked to 95 disliked). However, its small size and the fact that the model would only be able to predict one person's specific music taste make this dataset a poor candidate.

1.3 Candidate 3 - Loan Approval Classification Dataset

This dataset (Lo, 2024), similarly to Candidate 2, was sourced from Kaggle's cloud servers under an Apache 2.0 license, which states that the dataset can be used as long as credit is given to the original author. The dataset the physical form of a flat-file CSV, with the logical structure being the One Big Table data schema.



loan_data.csv (3.61 MB)

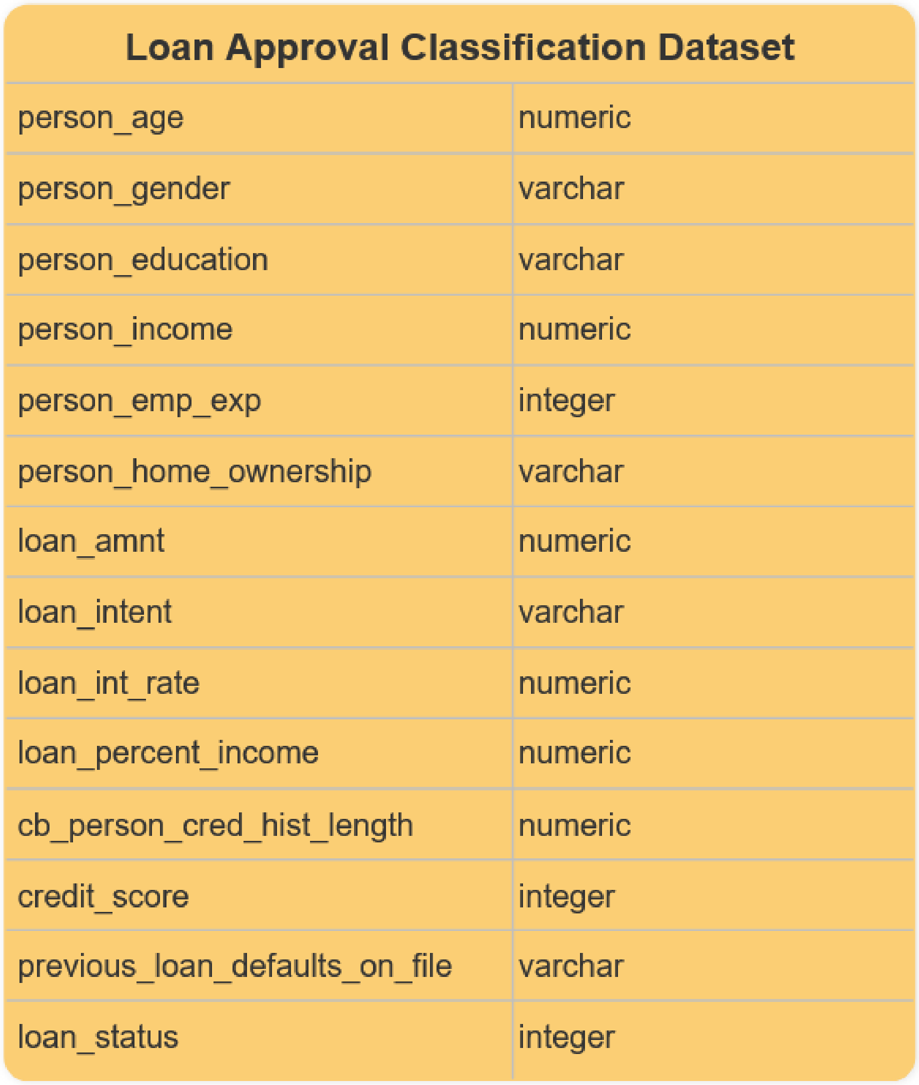
About this file

Loan approval dataset. "Loan Status" : 1 = approved; 0 = rejected

# person_age age	A person_gender gender	A person_education education	# person_income annual income	# person_emp_exp professional experience	A perso home o
20 144	male 55% female 45%	Bachelor 30% Associate 27% Other (19573) 43%	8k 7.2m	0 125	RENT MORTG Other (
22.0	female	Master	71948.0	0	RENT
21.0	female	High School	12282.0	0	OWN
25.0	female	High School	12438.0	3	MORTGA
23.0	female	Bachelor	79753.0	0	RENT

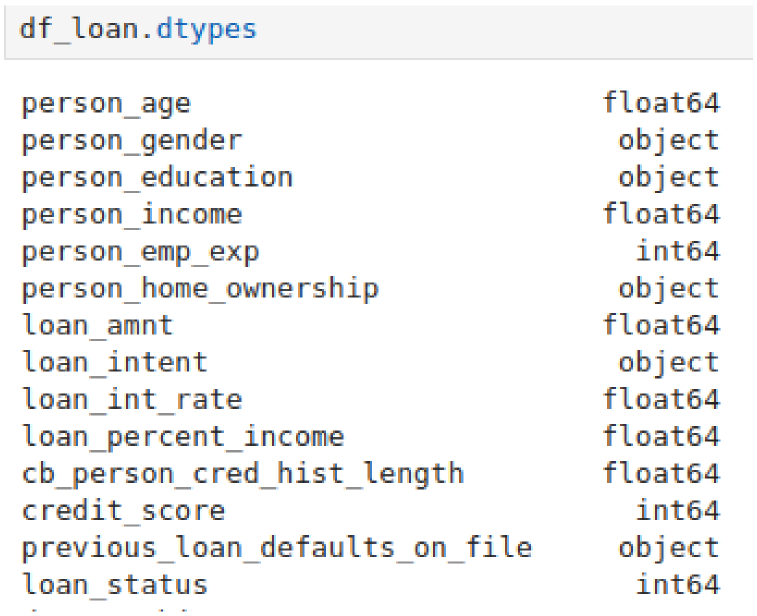
Figure 1.13: A snapshot of the Loan dataset's Kaggle page.

Unlike Candidates 1 and 2, this dataset does not consist of real data, and instead consists of synthetic data. This is likely due to the fact that this dataset, if it used real data, would contain extremely personal information that could not be shared online due to legislation such as GDPR, which is further discussed in Section 2.7. This particular dataset is an enhanced version of a different credit risk dataset, which also did not provide an original source and is also presumably synthetic data. The enhancements added were three additional columns, for the hypothetical person's gender, education level and credit score. Additionally, the author balanced the dataset using Synthetic Minority Oversampling Technique (SMOTE), an oversampling algorithm designed to balance datasets to produce higher quality machine learning models (Microsoft, 2024). It can therefore be established that this dataset was created solely for research and analysis purposes, including for the creation of supervised learning models. The dataset consists of 45,000 records and 14 features, with one of these being the ground truth target variable "loan_status", which is whether the person should be given a loan or not. 31 notebooks on Kaggle created by the site's users utilise this dataset, with many of these choosing to solve the binary classification problem that it presents, including works published by authors such as (Gupta et al., 2021). The dataset is physically stored as a flat-file CSV, under a One Big Table (OBT) logical data schema. The data types for each column can be seen in the entity relationship diagram and Pandas code in Figures 1.14 and 1.4, and descriptions of each column can be seen in Table 1.3.



Loan Approval Classification Dataset	
person_age	numeric
person_gender	varchar
person_education	varchar
person_income	numeric
person_emp_exp	integer
person_home_ownership	varchar
loan_amnt	numeric
loan_intent	varchar
loan_int_rate	numeric
loan_percent_income	numeric
cb_person_cred_hist_length	numeric
credit_score	integer
previous_loan_defaults_on_file	varchar
loan_status	integer

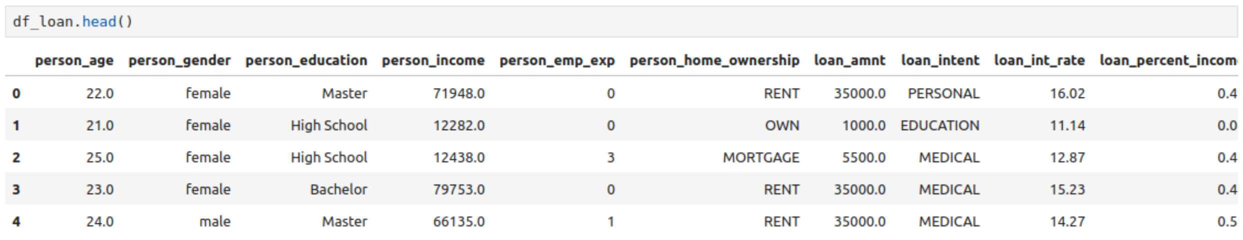
Figure 1.14: An entity relationship diagram of the Loan Approval Classification Dataset.



```
df_loan.dtypes
```

```
person_age                          float64
person_gender                        object
person_education                     object
person_income                       float64
person_emp_exp                        int64
person_home_ownership                object
loan_amnt                           float64
loan_intent                          object
loan_int_rate                       float64
loan_percent_income                 float64
cb_person_cred_hist_length          float64
credit_score                          int64
previous_loan_defaults_on_file       object
loan_status                           int64
```

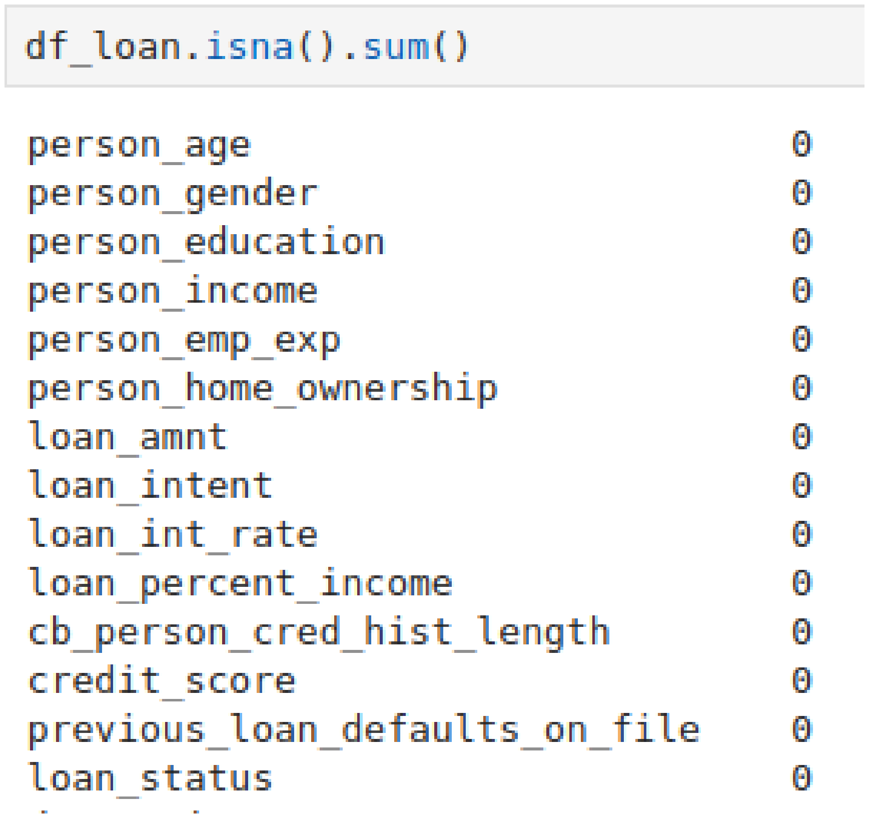
Figure 1.15: The data types of the Loan Approval Classification Dataset.



```
df_loan.head()
```

	person_age	person_gender	person_education	person_income	person_emp_exp	person_home_ownership	loan_amnt	loan_intent	loan_int_rate	loan_percent_incom
0	22.0	female	Master	71948.0	0	RENT	35000.0	PERSONAL	16.02	0.4
1	21.0	female	High School	12282.0	0	OWN	1000.0	EDUCATION	11.14	0.0
2	25.0	female	High School	12438.0	3	MORTGAGE	5500.0	MEDICAL	12.87	0.4
3	23.0	female	Bachelor	79753.0	0	RENT	35000.0	MEDICAL	15.23	0.4
4	24.0	male	Master	66135.0	1	RENT	35000.0	MEDICAL	14.27	0.5

Figure 1.16: The head of the dataset.



```
df_loan.isna().sum()
```

```
person_age                        0
person_gender                     0
person_education                  0
person_income                     0
person_emp_exp                    0
person_home_ownership             0
loan_amnt                         0
loan_intent                       0
loan_int_rate                     0
loan_percent_income               0
cb_person_cred_hist_length        0
credit_score                      0
previous_loan_defaults_on_file    0
loan_status                       0
```

Figure 1.17: No missing values in the dataset.

Column	Description	Measurement level
person_age	The age of the person.	Ratio
person_gender	The person's gender.	Nominal
person_education	The person's highest level of education.	Ordinal
person_emp_exp	The person's years of employment experience.	Ratio
person_home_ownership	Home ownership status (for example rent, own, mortgage)	Nominal
loan_amnt	The amount of money requested.	Ratio
loan_intent	The purpose of the loan.	Nominal
loan_int_rate	The interest rate of the loan.	Ratio
loan_percent_income	Loan amount as a percentage of the person's yearly income.	Ratio
cb_person_cred_hist_length	Length of credit history in years.	Ratio
credit_score	Credit score of the person.	Ratio
previous_loan_defaults_on_file	If the person has defaulted on a loan before.	Nominal
loan_status	Whether the loan should be approved. 1 if yes, 0 if no.	Nominal

Table 1.3: The descriptions of each column in the dataset.

This dataset is also frequently updated, with its most recent update occurring on the 29th of October. This would mean it may be more suited to an OLTP database system, which will be further discussed in Section 2.2.

1.4 Chosen dataset

Of the three candidates presented, the most suitable for a machine learning operations pipeline would be Candidate 3, the loan approval dataset. As mentioned in Section 1.3, this dataset possesses many predictor variables and an adequate amount of data to train a supervised learning classification model to classify whether an individual should be allowed a loan or not. While the data in this dataset is synthetic due to its real equivalent being highly protected under data protection legislation, the model trained from said synthetic data could be applied to real data using what it has learned, and could greatly expedite the process of loan approvals.

As previously mentioned, the dataset is hosted on Kaggle's cloud database, in the physical structure of a CSV file, using a One Big Table (OBT) data schema for its logical structure. Libraries such as Pandas natively work with these types of files which will allow for quick ingestion. When the dataset is ingested, it will be stored in a MariaDB Columnstore instance, which is further described in Section 2. For this pipeline, the data will be kept in the OBT schema due to the simplicity and efficiency of queries performed on it, and also to ensure maximum data integrity that could be lost if the logical structure were modified. There were other options that could have been used instead of OBT, represented in Table 1.4

Schema	Description	Positives	Negatives
Star (Kaminsky, 2024)	Stores data across a main fact table for measured or transactional data, and dimension tables for data related to the fact data, which can be visualised like a star.	Simple to understand, few joins needed, very scalable.	Data updates would require updates of multiple rows in multiple tables. Adding new columns (dimensions) after initial creation can be difficult because the schema is denormalised.
Snowflake (GeeksForGeeks, 2023)	An extended form of the Star schema that further branches its dimension tables into a hierarchy, appearing more like a snowflake than a star due to the multiple branches formed.	Unlike a star schema, data is normalised, allowing data to be added easier than a star schema.	More complex to query due to the increased amount of tables, therefore requiring more processing power.
Normalised Relational Model (MIcrosoft, 2024)	Heavily organises data, and emphasises the relationships between tables.	Keeps minimal or no redundant data whatsoever, reducing storage requirements. In doing so, this also heightens data integrity.	As with other schemas that use many tables, queries can be slowed by the need for multiple joins.

Table 1.4: An analysis of alternative data schemas.

Planning the MLOps Pipeline

All machine learning operations (MLOps) follow a five-step repeatable pipeline, outlined in Figure 2.1, where the output of one stage becomes the input of the next.

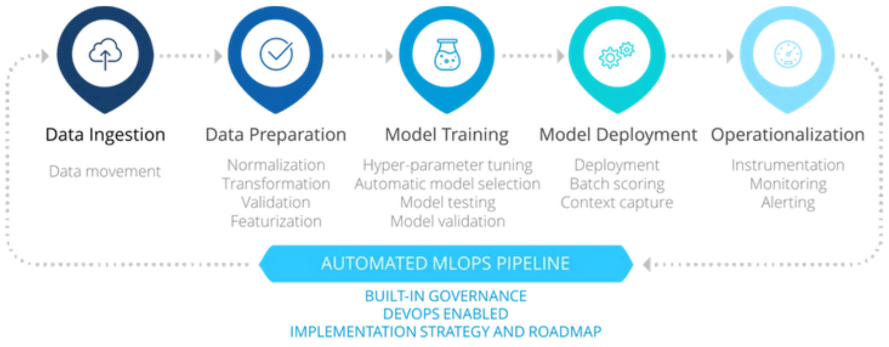


Figure 2.1: The five key steps in an MLOps pipeline (InCycle Software, 2024).

The pipeline begins with raw data and finishes with a trained machine learning model, and is often repeated at certain intervals, which could be as simple as once a day, or it could be repeated as new data becomes available. This repetition is performed automatically, so that the final model can become progressively more accurate. Because the process must be repeatable and automated, it is essential that data is validated to ensure that one run of the pipeline where the data may have been corrupted somehow would not cause issues, which would quickly spiral out of control as the pipeline is repeated again and again. Overall, MLOps pipelines standardise the development and deployment process of machine learning models, ensuring continuous integration (CI) and continuous delivery (CD) and enhancing collaboration between data scientists and development teams.

2.1 Software to be used in the pipeline

A wide variety of software will be used across the MLOps pipeline, with an overall glossary of them documented in Table 2.1. Descriptions on how they will specifically be used in each stage of the pipeline can be found in each stage's respective section.

Software/Library	Overview
Conda	A package and environment management system which allows for software developers and data scientists to easily control the libraries used in their code (Anaconda, 2024). Allows for the creation of **virtual environments**, which are contained structures where packages can be installed. The use of virtual environments allows for specific versions of packages to be kept, for instance with one environment storing an older version of a library for compatibility purposes, with another storing a newer version to be used in a different project. Were it not for these virtual environments, developers would constantly have to uninstall and reinstall specific package versions, wasting considerable amounts of time. Conda also hosts its own repositories to obtain packages from, and a major benefit of Conda occurs during the package installation process, which is that it will identify dependencies and version clashes between packages and solve them automatically, once again saving considerable amounts of time. In this project, Conda will be used as the environment and package manager to install and contain the libraries used in the development process. To do so, the Miniconda distribution will be installed, as this is a smaller distribution to save hard disk space and download time, but still contains the key Conda backend, as seen in Figure 2.2.
Apache Airflow	A platform used to orchestrate workflows and pipelines entirely in Python code (Apache, 2024c). Airflow creates Directed Acyclic Graphs (DAGs), which map out the order and dependencies of each task in a pipeline, and runs tasks in the order specified within the DAG, while also accounting for dependencies. Also provides many useful features such as task scheduling, which is especially useful for the automation of an MLOps pipeline, as well as automatic failure handling, where actions can automatically be performed on a task's failure, such as stopping the pipeline to prevent wasting computational resources. Airflow also provides a convenient UI, accessible by using the command "airflow webserver", which will host a web UI where tasks can be run, paused or stopped, as well as viewed in a tree view that maps the sequence and dependencies of tasks. Airflow also provides "Operators", which handle running code such as Python and Bash scripts. Airflow will be used within this project at all stages of the pipeline in order to manage the overall execution and structure of tasks. Operators will be used for the execution of all Python or Bash commands throughout the pipeline. By using Airflow in this way, the pipeline can become completely automated with continuous integration and continuous deployment.

Docker	An open-source containerisation system used to enhance the portability of applications by distributing them as self-contained, lightweight instances that come with everything they need to run immediately without needing to worry about system incompatibility (AWS (2024c), Docker (2022)) and minimise issues where software can run on one computer but not another. Docker could be described as working similarly to a virtual machine, though it is considerably more lightweight as it is not virtualising hardware.
MariaDB Columnstore	A columnar storage engine designed for the processing of petabytes of data with high performance and real-time response regardless of dataset size (MariaDB, 2024). While primarily intended for OLAP databases, it also facilitates OLTP databases. A Docker container for this software will be used in this pipeline.
Redis	An in-memory data store used to cache data in a machine's RAM rather than its persistent storage. The benefits of this are that said data can be loaded many times faster than if it were loaded from a hard drive. This does therefore mean that Redis will not be used to store persistent data, and it will instead be used to transfer data between different tasks in an Airflow DAG, as Airflow tasks are independent from each other. Redis will solve this by having relevant data loaded into memory before the task ends, at which point it will be read by the following task. A Docker container for this software will be used in this pipeline. Also includes a Python library of the same name that allows for access to the Redis store from Python scripts.
Pandas	A library used widely in industry for data analytics. It is capable of importing data of various types and storing them in a "DataFrame" object, which preprocessing operations can be performed on. It also allows the exporting of data in multiple formats (pandas, 2024).
SQLAlchemy	A library that allows SQL engine connections to be made from within a Python file. In doing so, data can be read from and stored into SQL databases (SQLAlchemy, 2024).
Scikit-learn	A large open-source Python library containing many different functionalities for machine learning (scikit-learn, 2024), such as encoders to convert strings to numerical equivalents, scalers to normalise and standardise the data to reduce variance (described further in Section 2.3), as well as containing methods to split data into training and testing sets, and fit, train and predict with various different machine learning algorithms (described further in Section 2.4).
FastAPI	A high-performance framework for developing APIs in Python, proclaimed as "One of the fastest Python frameworks available" (FastAPI, 2024). Facilitates an API for clients to access the backend ML model later in the pipeline.

Uvicorn	A low-level webserver implementation in Python with high performance (Uvicorn, 2024).
MLFlow	An open-source platform for the tracking and monitoring of machine learning models, storing each iteration of the model so that they may be reproduced at a later point (MLFlow, 2024), which is especially useful while performing hyperparameter tuning on the model, which is where small details of the model are changed such as the number of decision trees in a Random Forest. Also facilitates REST APIs, discussed in Section 2.5.
Apache Arrow PyArrow	Software with a Python library interface that allows for the serialization of objects (Apache, 2024b). This is important in synergy with Redis and Pandas, as Redis is unable to directly store Pandas dataframes, so they must first be serialized, converting them to a bytes format that Redis can store in memory.
Pickle	A base package in Python that functions similarly to Arrow in the serialization of data (Python, 2024). Pickle specifically will be used to serialize machine learning models.
Great Expectations	A Python library used for data validation, where "expectations" can be set for a dataset, such as all data of a column being numeric, and they will be assessed (GX, 2024).

Table 2.1: Descriptions of software and libraries across the pipeline.

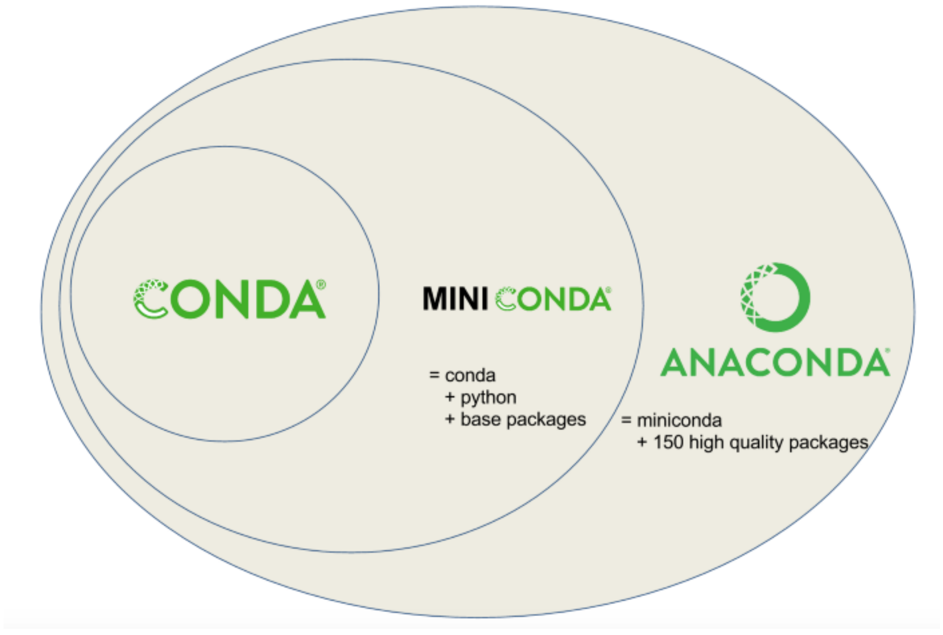


Figure 2.2: A comparison of the primary Conda distributions (TowardsDataScience, 2021)

2.2 Data Ingestion

2.2.1 Description

The first step of any machine learning pipeline is data ingestion. This refers to the process of obtaining the input data from its original source and transferring it to a relevant storage medium, such as a database or data warehouse, to be used in later stages. This stage is undertaken by data scientists. It is of vital importance that data is not lost or corrupted during ingestion, as this stage is the baseline for all future stages in the pipeline, and any issues here will directly impact all future stages, as previously discussed. Furthermore, when ingesting data, it is important to understand what type of system this data will be used in, of which there are two options: Online Analytics Processing Systems (OLAP), and Online Transactional Processing Systems (OLTP), as well as how it will be loaded into the chosen system, either using an ELT (Extract, Load, Transform) or ETL (Extract, Transform, Load) methodology.

OLAP and OLTP

OLAP	OLTP
Designed for complex queries and data analysis, primarily data reading.	Designed for lots of short, fast queries ("transactions") for both reading and writing.
Typically store massive amounts of data, sometimes petabytes. for extremely detailed analysis.	Usually store less data for speed purposes.
Usually historical data, infrequently updated.	Typically real-time data that is updated often.
Uses database schemas such as star or snowflake schema to allow for queries using many joins.	Uses normalised or denormalised models, such as One Big Table, minimising joins and maximising speed.
Slow response times, measured in minutes.	Fast response times, measured in milliseconds.

Table 2.2: A comparison of OLAP and OLTP systems (AWS, 2024b).

As mentioned in Table 2.2, OLAP systems are designed for complex and deep data analytics, which helps companies perform tasks such as analysing customer trends, while OLTP systems aim for maximum speed to complete quick transactions, which is necessary in situations like processing payments and orders.

ELT and ETL

ELT and ETL are both acronyms for the order of processes taken when ingesting data, with "Transform" either happening before or after the data is loaded into a storage medium like a data warehouse or data lake. The extract phase refers to the initial gathering of the data from its original source, such as Kaggle for the selected dataset in this report. The transform phase refers to early modifications made to the data, such as formatting or cleaning, and the load phase refers to the transportation of the data from its original storage medium (a CSV on Kaggle's cloud servers in this case) to a more optimised and efficient data warehouse to be used in the execution of the pipeline.

Regardless of whether ETL or ELT is used, the data is still extracted, loaded and transformed. However, the decision of which order to use can be determined from the data itself, with smaller datasets that may need complex transformation from their raw form being more suited to ETL, whereas large datasets with less transformation needed can be better with ELT (Smallcombe, 2024).

ELT	ETL
Data is transformed **after** being loaded into another storage medium.	Data is transformed **before** being loaded into another storage medium.
Useful if the data warehouse is a more modern system with good processing power.	Useful if the data warehouse has limited processing abilities of its own, typically seen in older or otherwise less powerful systems such as those on virtual machines.
Lower latency in the ingestion phase because data is immediately loaded.	Higher latency in the ingestion phase, because the data is being processed first, adding an extra time overhead where the pipeline could be held up.
Simpler to set up due to the immediate loading of the data.	Can be complex to set up and maintain, as the transformation would occur outside the warehouse.

Table 2.3: A comparison of ETL and ELT methodologies (Bartley (2024), AWS (2024a)).

It can be surmised from the analysis of each method in Table 2.3 that ETL is best applied to older systems with limited processing power, whereas ELT is better in more modern systems. It can additionally be argued that performing the ETL order of operations blurs the line between the ingestion and preprocessing stages, as the data is transformed before it is actually loaded into the system and fully ingested, whereas ELT has a clear split between the ingestion and preprocessing of the data.

2.2.2 In this project

The candidate dataset's raw CSV file, downloaded directly from Kaggle, will act as the input for this stage. It utilises the One Big Table schema, and it was previously established in Section 1.3 that it contains no missing values. The size of the dataset is the largest of the three candidates but is still small in comparison to those used in industry, being only 3.5MB in comparison to gigabytes and petabytes as previously mentioned in Table 2.2. Because of this small size, processing operations performed on the dataset will be very fast, even with limited processing power, so the ETL methodology will be used to ingest the data into a MariaDB Columnstore OLTP database for quick and efficient loading and querying, where the end user will be able to quickly supply predictor variables and receive a fast response.

To do so, some of the software originally mentioned in Table 2.1 will be used, seen in Table 2.4.

Software/Library	Usage for ingestion
Docker	In this stage, Docker will be used to host a container of MariaDB Columnstore.
MariaDB Columnstore	In this stage, the Columnstore will be hosted on a Docker container at port 3306 (Docker Hub, 2024), and will be used as the OLTP storage medium of the ingested data.
Pandas	In the ingestion phase, it will be used to import the dataset's CSV file, and export it as SQL to the MariaDB Columnstore through the use of SQLAlchemy.
SQLAlchemy	Alongside Pandas, SQLAlchemy will export the DataFrame to the MariaDB Columnstore instance.

Table 2.4: Software to be used in ingestion.

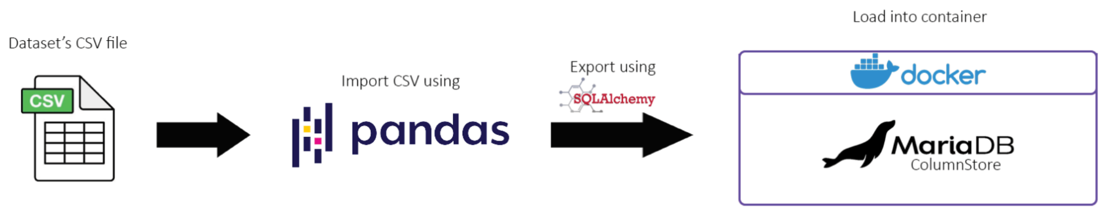


Figure 2.3: A diagram of the planned ingestion process.

2.3 Data Preprocessing

2.3.1 Description

After the data has been ingested, the preprocessing stage begins, and is often also conducted by Data Scientists. This stage encompasses the cleaning, integration and transformation of the data in order to optimise the dataset for model development.

Cleaning refers to the identification of missing, inaccurate or malformed data within the dataset, as well as its removal or imputation where possible.

Integration is often seen in datasets with multiple tables or that have been retrieved from multiple sources, and refers to the combination of the retrieved data into a single flat file.

Transformation, also known as feature engineering, is a considerable aspect of data preprocessing which refers to the manipulation and formatting of the data, such as changing the formats of columns from numeric dates to proper date data types, as well as the handling of categorical data, such as genders, which may originally be strings. Strings cannot be interpreted by machine learning models, and therefore they are encoded into numbers using techniques such as label encoding, which converts the unique values in a column to a numerical representation, such as male being 0 and female being 1. Data is also normalised and standardised in this stage, meaning that numerical data is reduced to being between 0 and 1 to adjust the overall scale of the data. This is especially useful with algorithms such as K-Nearest Neighbours, where large differences in distance between data can mislead the classification algorithm (IBM, 2021b).

Once these tasks have all been completed, the dataset will be ready to be used for model development.

2.3.2 In this project

The preprocessing in this project will take the ingested data as input, from which point it will be transformed, with columns such as "person_gender" encoded to categorically labelled numerical equivalents. To do so, some of the software and libraries originally mentioned in Table 2.1 will be used, shown below in Table 2.5. After this, the processed dataset will be output to Redis, to be used as the input for the model development stage.

Software/Library	Usage for preprocessing
Pandas	Will be used alongside SQLAlchemy to load the dataset from the MariaDB Columnstore instance hosted on Docker, as well as for the actual transformation of the data using the methods provided by Pandas dataframes.
SQLAlchemy	Will be used alongside Pandas to query and receive data from the MariaDB Columnstore instance.
Scikit-learn	Will be used to normalise and standardise the data, as well as encode any string data into numerical equivalents.

Docker	Will host the MariaDB Columnstore container. Additionally for this section, it will also host a container for Redis to store the processed dataset.
Redis	Will store the processed dataframe in memory. Cannot directly store the dataframe as mentioned in Table 2.1, so it must be serialised first using Arrow.
Arrow	Will serialise the processed dataframe so that it can then be stored by Redis.

Table 2.5: Descriptions of software to be used for preprocessing.

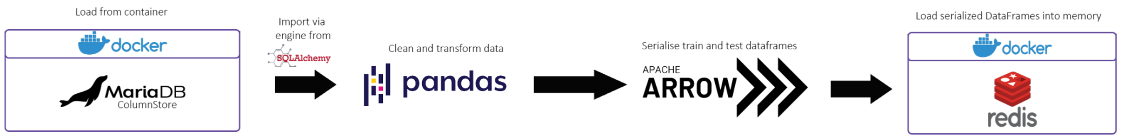


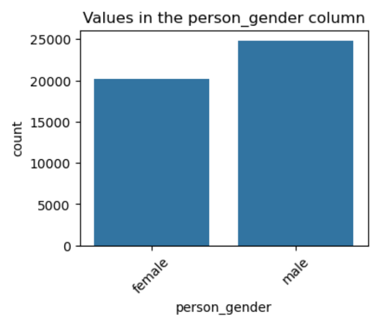
Figure 2.4: A diagram of the planned preprocessing stage.

2.3.3 Data analysis

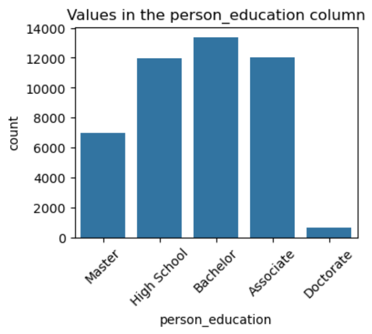
To build a machine learning model for a dataset, it greatly helps to understand the data itself.

Distributions and outliers

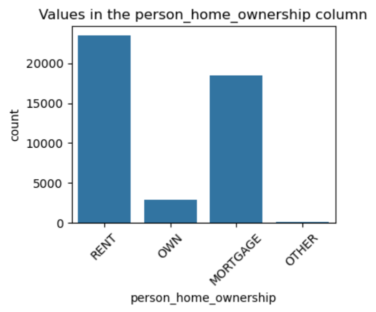
Understanding the data means to know how the data is typically distributed, and what constitutes an outlier. This can be done through visualisations such as box plots for numerical data and bar charts for categorical data.



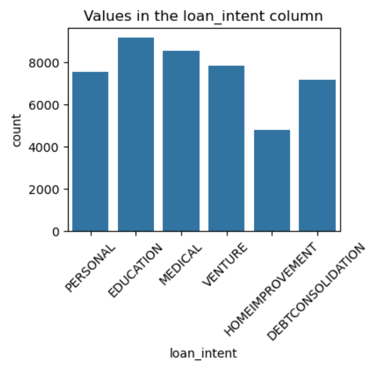
(a) Genders



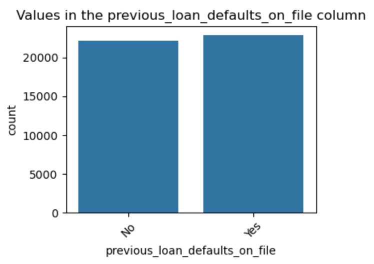
(b) Education levels



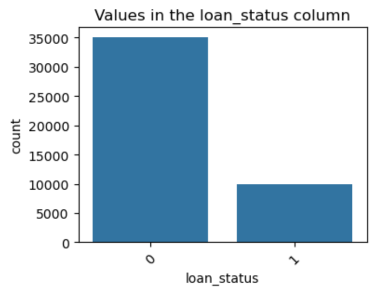
(c) Home ownership



(d) Loan intents



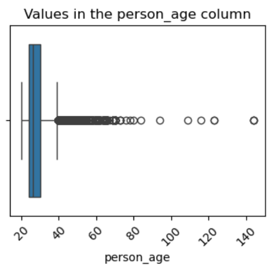
(e) Previously defaulted



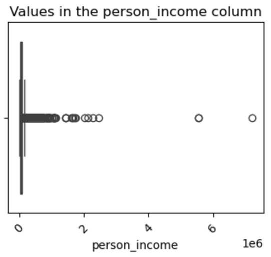
(f) Loan status

Figure 2.5: Bar charts of the six categorical rows.

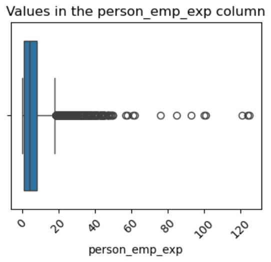
A notable observation is that the dataset is imbalanced, containing fewer records of granted loans than denied loans.



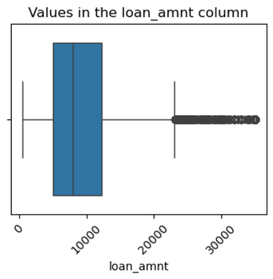
(a) Age



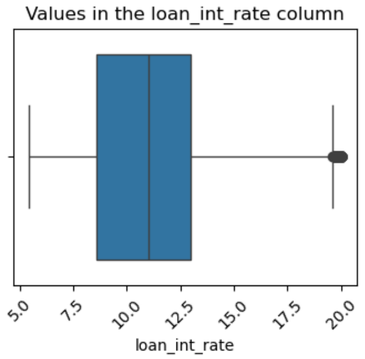
(b) Yearly income



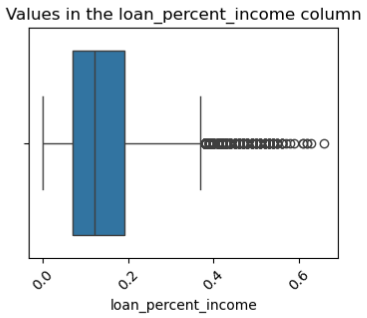
(c) Previous jobs



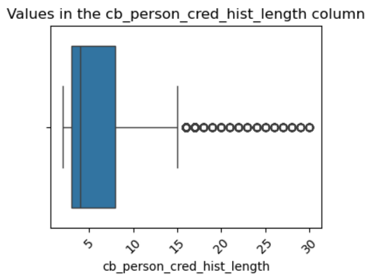
(d) Loan amount



(e) Interest rate



(f) Percentage of income



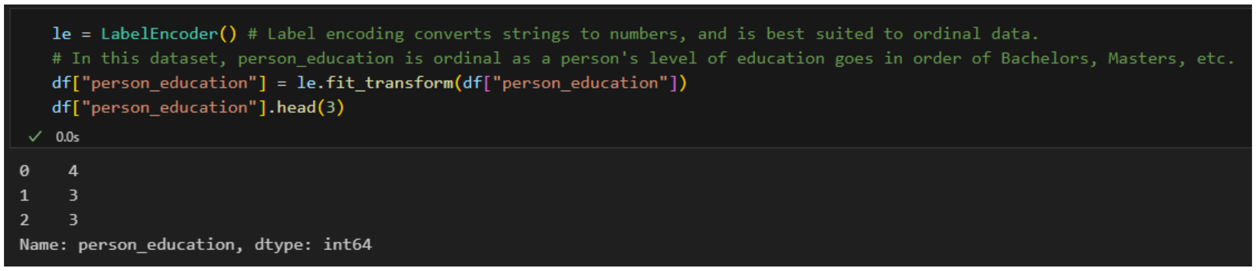
(g) Credit history length

Figure 2.6: Box plots of all numerical columns.

There are many outliers in this dataset, most notably in the income column, where very few are earning millions. Also, the age column has some outliers of extremely old people, including one aged 140, who cannot exist as the oldest person known was 112 before they died (Sky News, 2024). This can therefore likely be attributed to erroneous generation due to this dataset being synthetic, and records with considerable outliers like this can be safely removed in the implementation's preprocessing phase.

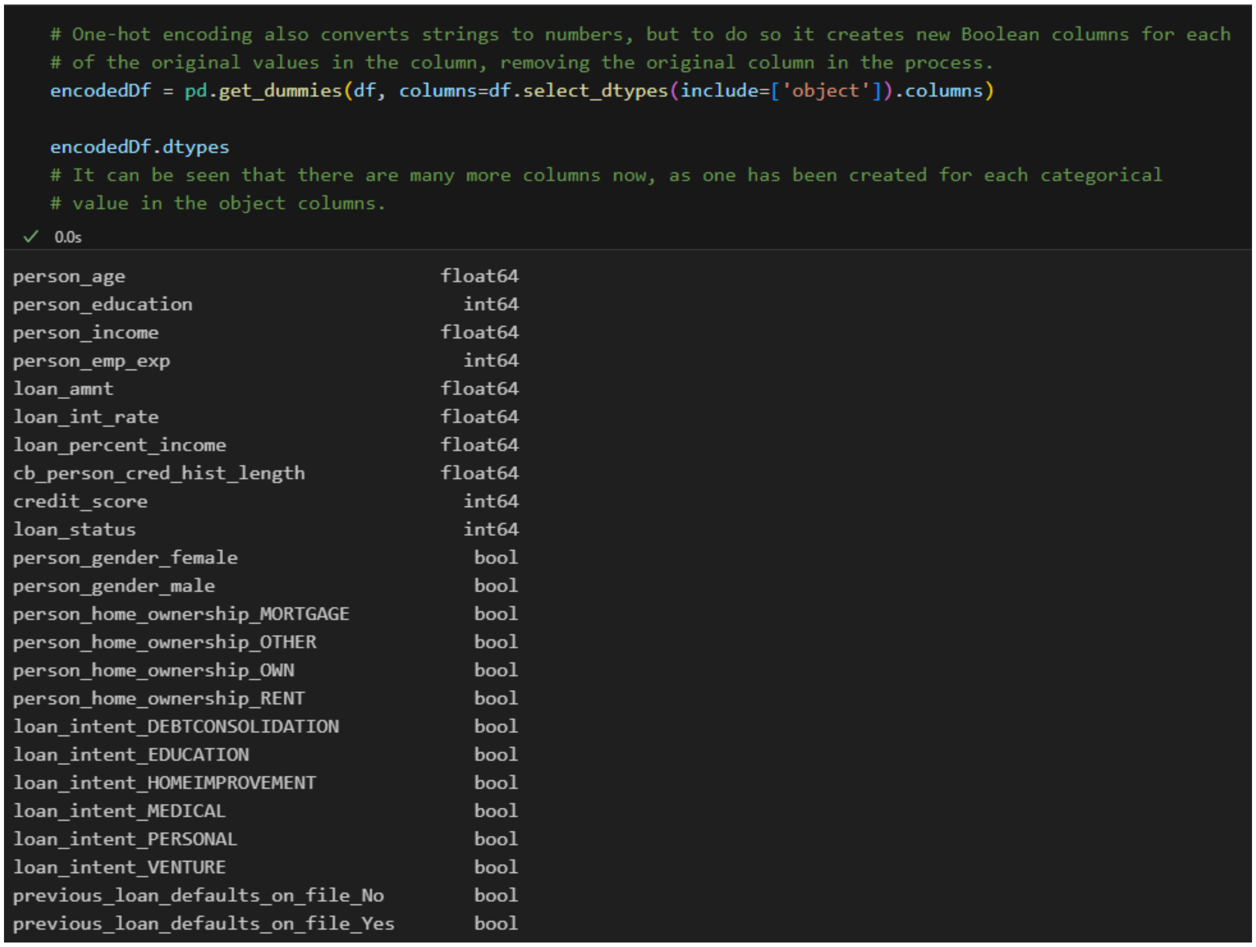
Encoding

Machine learning models cannot directly interpret string data. Therefore, strings must be encoded to numerical representations through either label encoding or one-hot encoding. Label encoding is best suited to ordinal data as it will maintain the ordering of said data, such as levels of education. One-hot encoding is better suited to nominal data without an inherent order, such as the intent of a loan. Therefore, this dataset requires the use of both types of encoding, shown in Figures 2.7 and 2.8



```
le = LabelEncoder() # Label encoding converts strings to numbers, and is best suited to ordinal data.
# In this dataset, person_education is ordinal as a person's level of education goes in order of Bachelors, Masters, etc.
df["person_education"] = le.fit_transform(df["person_education"])
df["person_education"].head(3)
✓ 0.0s
0    4
1    3
2    3
Name: person_education, dtype: int64
```

Figure 2.7: Using label encoding for the person_education column.



```
# One-hot encoding also converts strings to numbers, but to do so it creates new Boolean columns for each
# of the original values in the column, removing the original column in the process.
encodedDf = pd.get_dummies(df, columns=df.select_dtypes(include=['object']).columns)

encodedDf.dtypes
# It can be seen that there are many more columns now, as one has been created for each categorical
# value in the object columns.
✓ 0.0s
person_age                        float64
person_education                    int64
person_income                     float64
person_emp_exp                      int64
loan_amnt                         float64
loan_int_rate                     float64
loan_percent_income               float64
cb_person_cred_hist_length        float64
credit_score                        int64
loan_status                         int64
person_gender_female                 bool
person_gender_male                   bool
person_home_ownership_MORTGAGE       bool
person_home_ownership_OTHER          bool
person_home_ownership_OWN            bool
person_home_ownership_RENT           bool
loan_intent_DEBTCONSOLIDATION        bool
loan_intent_EDUCATION                bool
loan_intent_HOMEIMPROVEMENT          bool
loan_intent_MEDICAL                  bool
loan_intent_PERSONAL                 bool
loan_intent_VENTURE                  bool
previous_loan_defaults_on_file_No    bool
previous_loan_defaults_on_file_Yes   bool
```

Figure 2.8: Using one-hot encoding for the person_education column.

Correlations

The easiest way to view the correlations between variables is through a correlation matrix, seen below in Figure 2.9, which cannot be created unless data is first encoded. However, an immediate observation is the sheer size of the matrix due to an unfortunate side effect from one-hot encoding creating entirely new columns, increasing the dimensionality of the dataset.

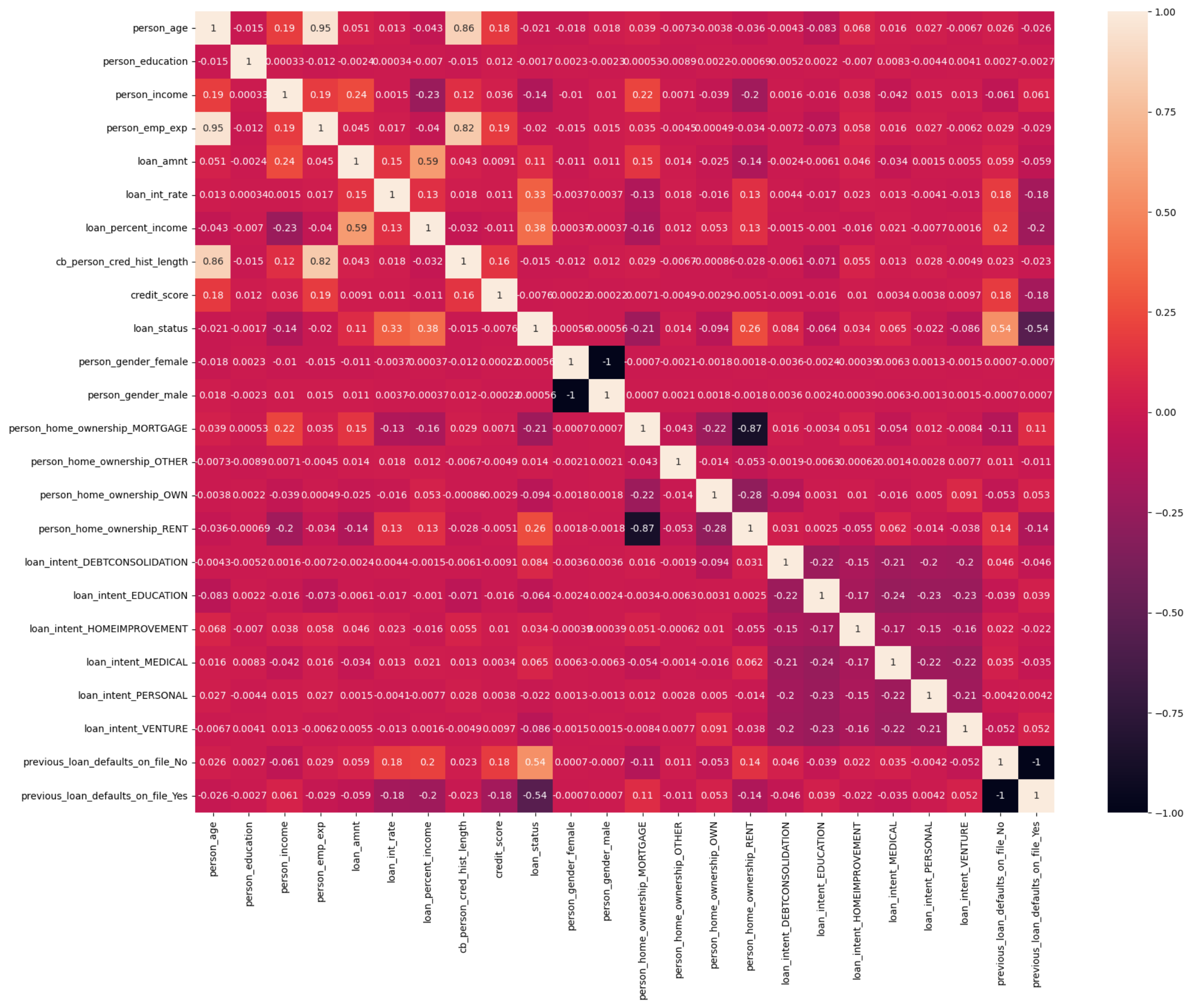


Figure 2.9: The full correlation matrix for the encoded dataset.

Despite this, key insights can still be made about the likelihood of a loan being granted, such as:

- Loans are most likely to be approved if the person has never defaulted on a loan before.
  - They are least likely to be approved if the person has defaulted before.
- The interest rate positively correlates with the loan's approval.
- If the person rents their home, they are more likely to be approved.
  - They are less likely to be approved if they have a mortgage.
- Loans are more likely to be approved if they are a large percentage of the person's yearly income.

2.4 Model Development

2.4.1 Description

The model development stage accepts the processed dataset from the preprocessing stage as input and leverages machine learning algorithms to solve the problem in question, either a classification problem where data will be identified as being of a certain category (class), or a regression problem where unknown data can be predicted. Both of these problems require the model to be "fitted" and "trained". These refer to the utilisation of the processed dataset for the recognition of patterns, associations and correlations within the data. To fit and train the data, it is split into two sets: a training set, consisting of a large majority of the data ($\tilde{8}0\%$), and a testing set which uses the remaining minority. The algorithm will then use what it has learned from the training set to make predictions on the testing set, from which the accuracy of the model can be ascertained. These processes often yield better results with larger datasets, as the algorithm will have more information to learn and make predictions based on, which is why it is preferred to not remove data from the dataset unless strictly necessary.

This stage is conducted by machine learning engineers, and its output is that of the trained machine learning model, which can then be deployed.

2.4.2 In this project

The dataset will first be loaded from Redis and deserialised by Arrow. The dataset poses a classification problem, and as such, the Scikit-Learn Python library will be particularly key in this stage for its implementation of a Random Forest algorithm, which is reputed as one of the most accurate models in many scenarios. However, it can have a steep processing time depending on how many decision trees it is told to create.

Software/Library	Usage for development
Arrow	Will deserialize the stored dataframe from Redis after the preprocessing stage.
Pandas	Will store the dataframe deserialized by Arrow.
Redis	Stores the processed dataset to be retrieved at the beginning of this stage.
Scikit-learn	Will be used for its implementations of various algorithms, most notably a Random Forest Classifier. Will also provide useful metrics such as accuracy on the training and test data.
MLFlow	Will be used to store and track each iteration of the model that is produced as this phase is repeated.

Table 2.6: Descriptions of software to be used for model development.

2.5 Model Deployment

2.5.1 Description

The developed model from the previous stage of the pipeline can then be integrated into an actual environment, and can be utilised as a tool to make decisions. This stage is where software engineers will make the model available for use, and where the model will therefore begin to be provided with unseen data, which refers to data outside of the original training dataset. Models are typically deployed using Representational State Transfer APIs, better known as REST APIs (RedHat, 2024). REST APIs make use of typical frontend web HTTP requests (GET, POST, PUT, DELETE) from the client to give instructions to the backend machine learning model (RestfulAPI, 2023). A significant benefit of using REST APIs is the massive portability benefits provided; using a REST API means that the model can provide results to a wide variety of devices such as Windows PCs, Macs, and even phones, as anything that can make HTTP requests can interface with one.

2.5.2 In this project

The produced model will be hosted on a webserver via Uvicorn, and the REST API will be implemented via FastAPI. These two Python packages will provide an interface where the user can input the predictor variables of the dataset (age, credit score, etc.) to receive the model's output result.

Software/Library	Usage for deployment
FastAPI	Will be used to provide the API between the user and the ML model, where the user will give data to API endpoints and the model will supply a prediction.
Uvicorn	Will be used to host the webserver that the API will run on.
MLFlow	Will be used as a "bridge" of sorts between the user and the model, and will access and load it as well as retrieving any necessary artifacts to process the data that the user supplies.

Table 2.7: Descriptions of software to be used for model deployment.

2.6 Model Monitoring

2.6.1 Description

After the model is deployed, its performance is continuously monitored by data scientists. The monitoring process consists of the analysis of the model's results via metrics such as those found in Table 2.8 By monitoring the model, any issues with it can be quickly identified and the pipeline can be restarted to yield a higher accuracy model, which can be monitored again.

Metric	What is it?	Type
Accuracy	The number of correct predictions divided by the total amount of predictions.	Classification
Precision	The ratio of correctly predicted positives to the total amount of positives in the dataset.	Classification
F1-score	A measurement calculated from a model's accuracy and precision.(Kundu, 2024)	Classification
Mean Absolute Error (MAE)	The average of the differences between predicted and actual values.	Regression
R^2 (R-Squared)	Also known as the coefficient of determination, shows how well the predicted data fits the actual data (CFI, 2024).	Regression

Table 2.8: Descriptions of various metrics to grade ML models.

2.6.2 In this project

This pipeline aims to produce a binary classification model, so metrics such as F1-score will be useful in the evaluation of each iteration of the model. To do so, MLFlow will be used to store each iteration's parameters and performance metrics.

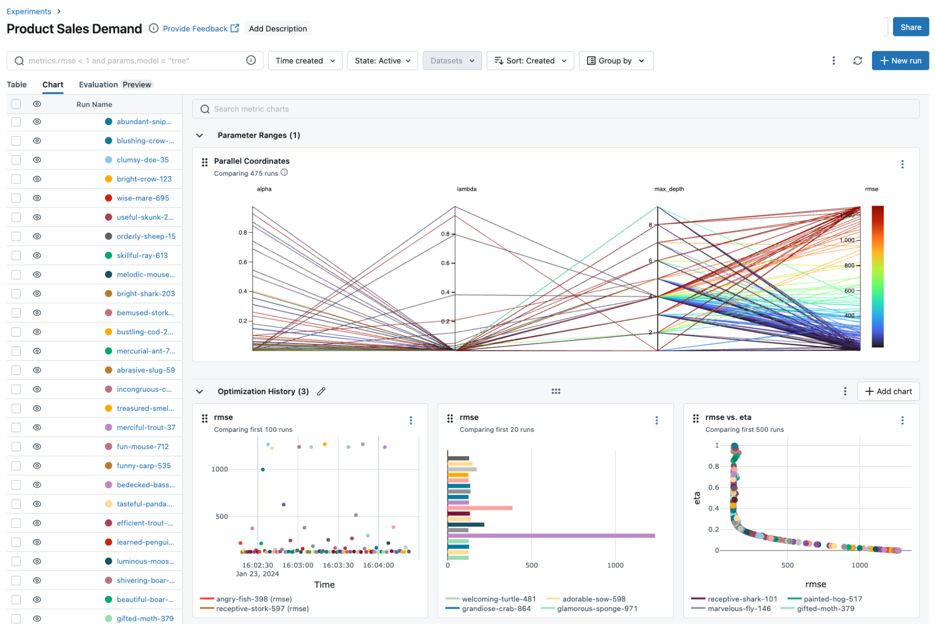


Figure 2.10: An example of the MLFlow monitoring dashboard (MLFlow, 2024).

2.7 Data terminology

This project must take into account four key elements of data handling these being data privacy, security, ethics and data protection laws.

Term	Description
Data privacy	An individual's ability to govern what happens to their personal data (Cloudflare, 2024). Personal data refers to data that could be used to singularly identify someone (Yang, 2021).
Data security	The protection of personal/sensitive information from unauthorized access, use or manipulation (IBM, 2021a). This can be the physical protection of material such as hard drives, and software protections like encryption.
Data ethics	A blanket term encompassing the moral obligations of data collection, use and protection (Harvard Business School, 2021). Ethical data storage would mean that the data subjects willingly gave their data and can ask for its deletion at any point. Mandated by data protection legislation.
Big data ethics	The application of data ethics to massive datasets, concerning the implications of their collection and use on society as a whole (Richards and King, 2014).
General Data Protection Regulations (GDPR)	Legislation introduced in the EU in 2016, giving individuals significantly more rights over their personal data. Data subjects most notably may request a copy of their data, the removal of their data, and to object to their data's collection (ICO, 2024a). Additionally, data must be kept secure using all available means. Companies who fail to adhere to these policies face significant sanctions of €10,000,000 or 2% of the company's annual turnover for minor violations, or double this (€20,000,000, 4% turnover) for major violations, whichever figure is higher.
Data Protection Act 2018	When the UK left the EU, the GDPR no longer applied. Therefore, the UK adopted the GDPR as an amendment to its own Data Protection Act. There are minimal differences between the two, only that the UK's fines are in GBP rather than Euros, meaning fines go up to £17,500,000 or 4% turnover (ICO, 2024b).

Table 2.9: A brief overview of relevant data terminology.

Implementing the MLOps Pipeline

3.1 Ethical and Legal Considerations

The dataset was shared under an Apache 2.0 open-source licence, permitting access and modification with no restrictions except crediting the original author (Apache, 2024a), meaning that there are not any ethical or legal considerations of note with this dataset's usage. Given that the chosen dataset also consists of synthetic data, there are fewer necessary considerations for its use.

3.1.1 Synthetic data

This term has been frequently used throughout the report to describe the Loan Approval Classification Dataset, and refers to data that has been generated algorithmically rather than collected from a real source. However, the generation of the synthetic data does often depend on real data to identify patterns and trends, meaning that legislation such as GDPR is still relevant in those scenarios because the generated data must be impossible to link back to the original subject (Lopez and Elbi, 2022). An example of synthetic data generation is SMOTE, which was used in the generation of the loan dataset (Zoppelleto, 2024), which creates synthetic data in a dataset based on the other rows as previously mentioned in Section 1.3. SMOTE will also be used in Section 3.4 to balance the dataset.

3.1.2 Model drift

Model drift is a phenomenon that affects deployed production machine learning models where changes in their environment over time affects their performance (Nigenda et al., 2022). Model drift is not an immediate occurrence, and can take a long time to occur, emphasising the importance of continuous integration and deployment. Table 3.1 details the different kinds of model drift.

Drift type	Description	Dataset example
Data drift	Changes in the distributions of the ingested data over time that eventually significantly differ from the model's training data (DataCamp, 2024).	In this dataset, interest rates and people's yearly income may change as time passes or economic inflation occurs.
Concept drift	Changes in the relationship between the input and target variables (Nigenda et al., 2022).	In this dataset, this is somewhat unlikely. However, it could occur if there is a significant housing or job market crash that would cause more people to need loans, which would change the original relationships.

Table 3.1: The common types of model drift.

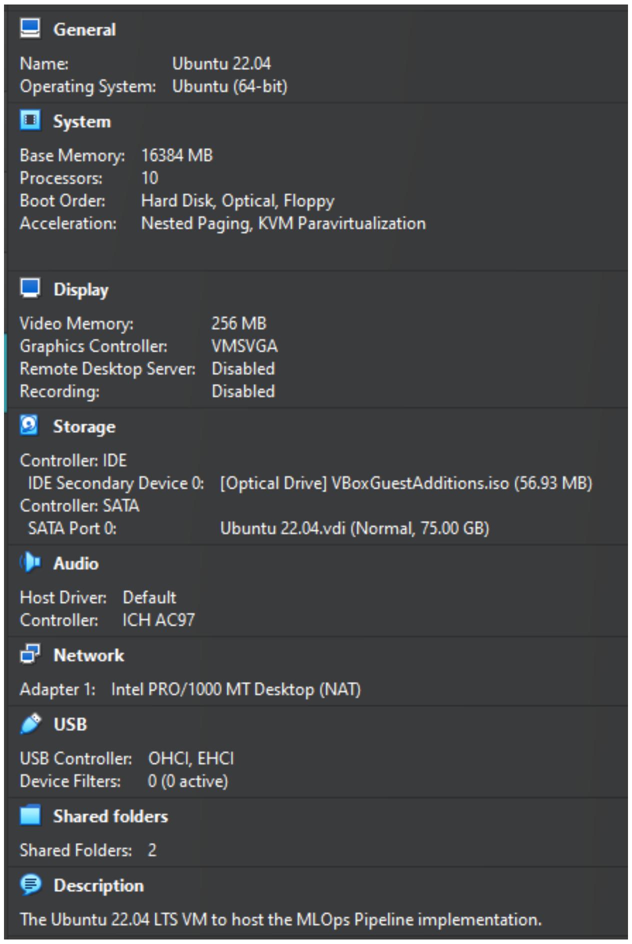
Model drift can be a significant issue that greatly impacts the performance of ML models if it is not quickly noticed, and can also be an ethical concern if the model is being used to influence decisions such as granting loans in this project. Therefore, combatting model drift through methods such as continuous monitoring is necessary to ensure that the model remains accurate and usable. This can be accomplished in this project through MLFlow's monitoring dashboard.

3.2 Initial Software Setup

This section details the configuration of the key software and packages within the pipeline. The installation processes of Miniconda and Docker are very lengthy and can therefore be found in Appendix A.

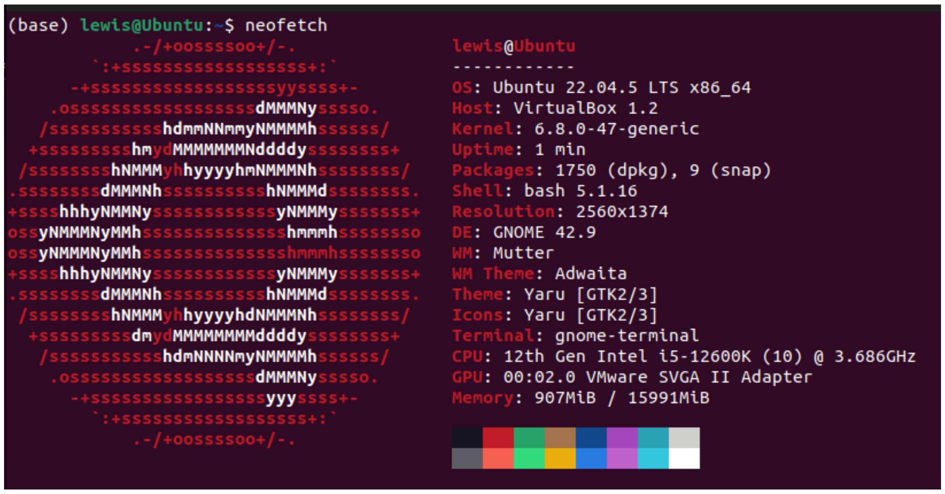
3.2.1 VirtualBox & Ubuntu

The majority of data scientists utilise Linux distributions for their projects due to its open-source nature and the availability of many tools and packages. Therefore, VirtualBox, software to create a virtual machine which runs inside the host machine (Oracle, 2024), will be used to virtualise an Ubuntu 22.04 LTS system on the Windows host machine. This particular OS was chosen because of its relative recency and frequent software & security updates. Virtual machines also allow for "snapshots", which store the current state of the machine and its files at that time to be restored at any point in the event of unforeseen errors. Should a catastrophic error that would damage the system occur, the host machine would be unaffected as the virtual machine is isolated.



```
General
Name:               Ubuntu 22.04
Operating System:   Ubuntu (64-bit)

System
Base Memory:   16384 MB
Processors:    10
Boot Order:    Hard Disk, Optical, Floppy
Acceleration:  Nested Paging, KVM Paravirtualization

Display
Video Memory:            256 MB
Graphics Controller:     VMSVGA
Remote Desktop Server:   Disabled
Recording:               Disabled

Storage
Controller: IDE
 IDE Secondary Device 0:   [Optical Drive] VBoxGuestAdditions.iso (56.93 MB)
Controller: SATA
 SATA Port 0:              Ubuntu 22.04.vdi (Normal, 75.00 GB)

Audio
Host Driver:  Default
Controller:   ICH AC97

Network
Adapter 1:  Intel PRO/1000 MT Desktop (NAT)

USB
USB Controller:  OHCI, EHCI
Device Filters:  0 (0 active)

Shared folders
Shared Folders:  2

Description
The Ubuntu 22.04 LTS VM to host the MLOps Pipeline implementation.
```

Figure 3.1: The configuration for the pipeline's VM.

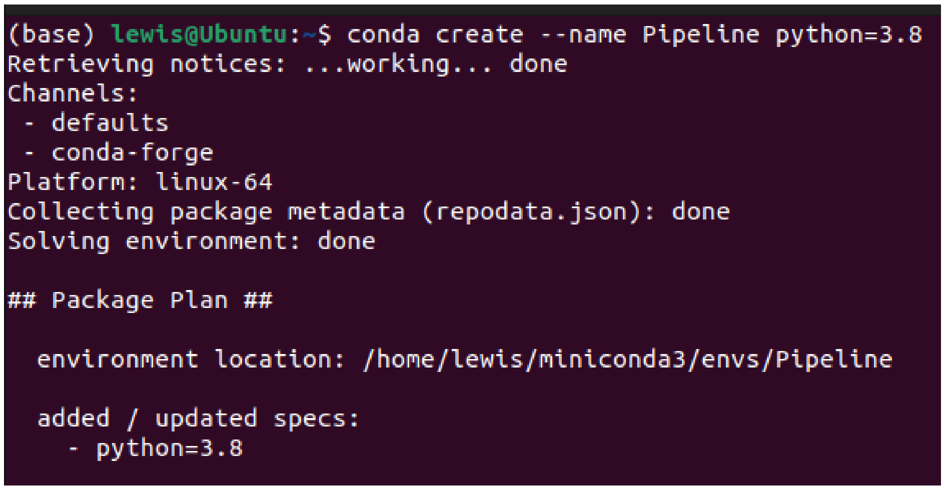


```
(base) lewis@Ubuntu:~$ neofetch
            .-/+oossssoo+/-.               lewis@Ubuntu
        `:+ssssssssssssssssss+:`           ------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 22.04.5 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: VirtualBox 1.2
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 6.8.0-47-generic
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 1 min
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 1750 (dpkg), 9 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.1.16
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Resolution: 2560x1374
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   DE: GNOME 42.9
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   WM: Mutter
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   WM Theme: Adwaita
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Theme: Yaru [GTK2/3]
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/    Icons: Yaru [GTK2/3]
  +sssssssssdmydMMMMMMMMddddyssssssss+     Terminal: gnome-terminal
   /ssssssssssshdmNNNNmyNMMMMhssssss/      CPU: 12th Gen Intel i5-12600K (10) @ 3.686GHz
    .ossssssssssssssssssdMMMNysssso.       GPU: 00:02.0 VMware SVGA II Adapter
      -+sssssssssssssssssyyyssss+-         Memory: 907MiB / 15991MiB
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.
```

Figure 3.2: The VM's Neofetch display.

3.2.2 Conda environment and packages

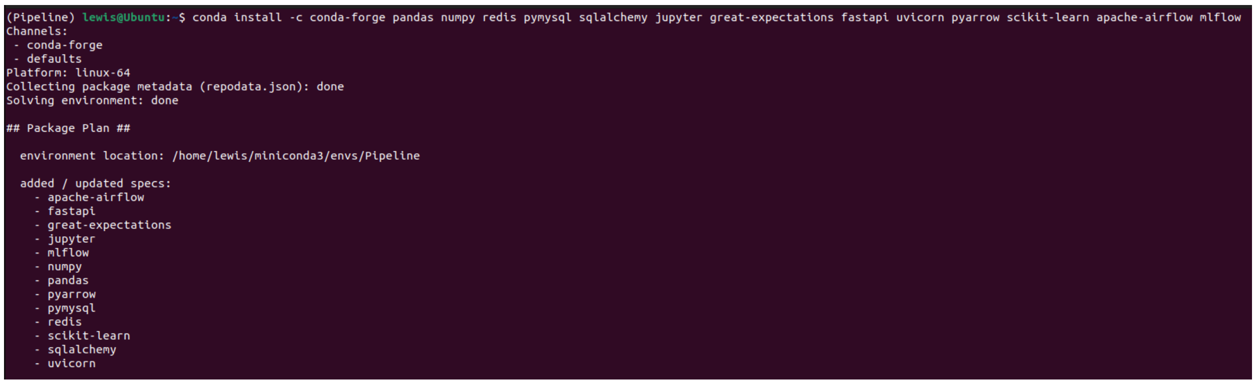
This section details the installation of packages to the Pipeline environment.



```
(base) lewis@Ubuntu:~$ conda create --name Pipeline python=3.8
Retrieving notices: ...working... done
Channels:
 - defaults
 - conda-forge
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/lewis/miniconda3/envs/Pipeline

  added / updated specs:
    - python=3.8
```

Figure 3.3: Creating the "Pipeline" Conda environment.

Python 3.8 is used due to package compatibility issues with later versions of Python.

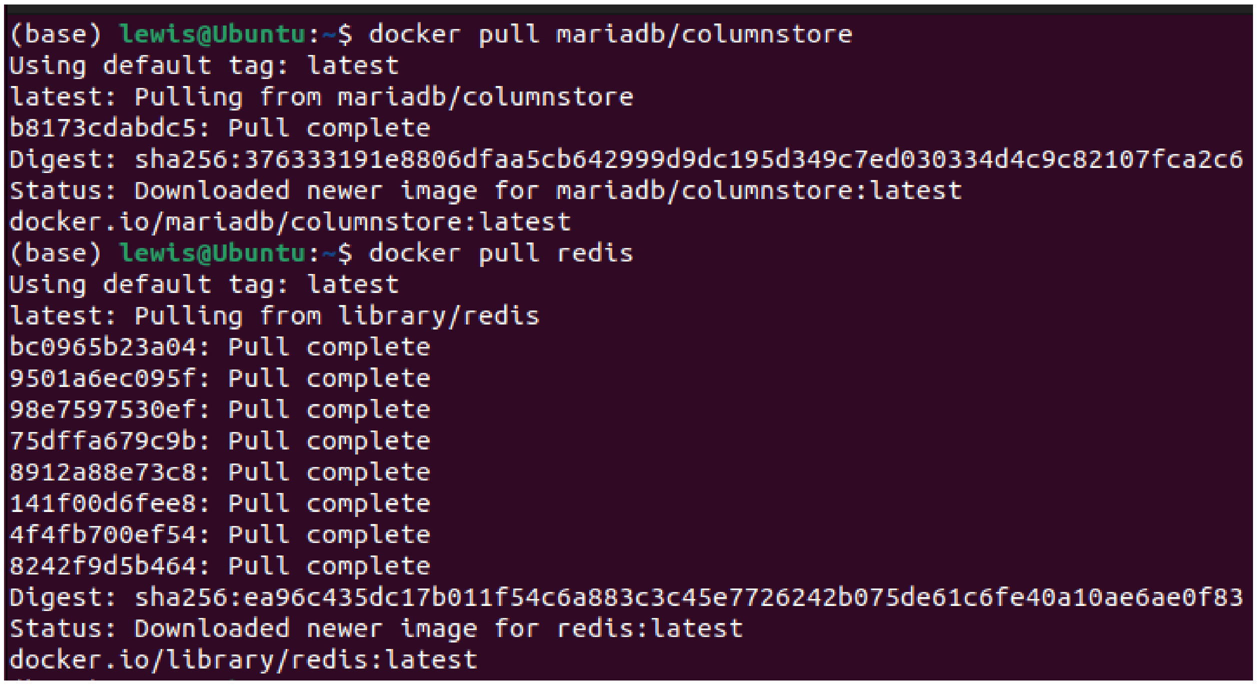


```
(Pipeline) lewis@Ubuntu:~$ conda install -c conda-forge pandas numpy redis pymysql sqlalchemy jupyter great-expectations fastapi uvicorn pyarrow scikit-learn apache-airflow mlflow
Channels:
 - conda-forge
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /home/lewis/miniconda3/envs/Pipeline

  added / updated specs:
    - apache-airflow
    - fastapi
    - great-expectations
    - jupyter
    - mlflow
    - numpy
    - pandas
    - pyarrow
    - pymysql
    - redis
    - scikit-learn
    - sqlalchemy
    - uvicorn
```

Figure 3.4: Installing packages to the environment via Conda.

3.2.3 Docker and Docker images

This section covers the downloading of the two necessary Docker images, MariaDB Columnstore and Redis. Their usage is further elaborated upon later in this chapter. The images for the containers will first need to be downloaded from Docker's repository, shown in Figure 3.5.

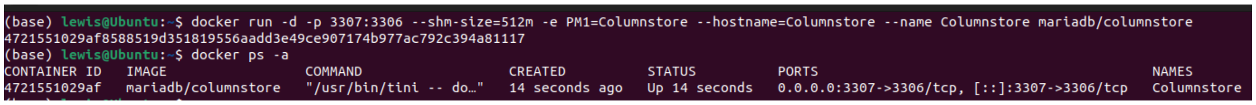


```
(base) lewis@Ubuntu:~$ docker pull mariadb/columnstore
Using default tag: latest
latest: Pulling from mariadb/columnstore
b8173cdabdc5: Pull complete
Digest: sha256:376333191e8806dfaa5cb642999d9dc195d349c7ed030334d4c9c82107fca2c6
Status: Downloaded newer image for mariadb/columnstore:latest
docker.io/mariadb/columnstore:latest
(base) lewis@Ubuntu:~$ docker pull redis
Using default tag: latest
latest: Pulling from library/redis
bc0965b23a04: Pull complete
9501a6ec095f: Pull complete
98e7597530ef: Pull complete
75dffa679c9b: Pull complete
8912a88e73c8: Pull complete
141f00d6fee8: Pull complete
4f4fb700ef54: Pull complete
8242f9d5b464: Pull complete
Digest: sha256:ea96c435dc17b011f54c6a883c3c45e7726242b075de61c6fe40a10ae6ae0f83
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

Figure 3.5: Pulling the Docker images.

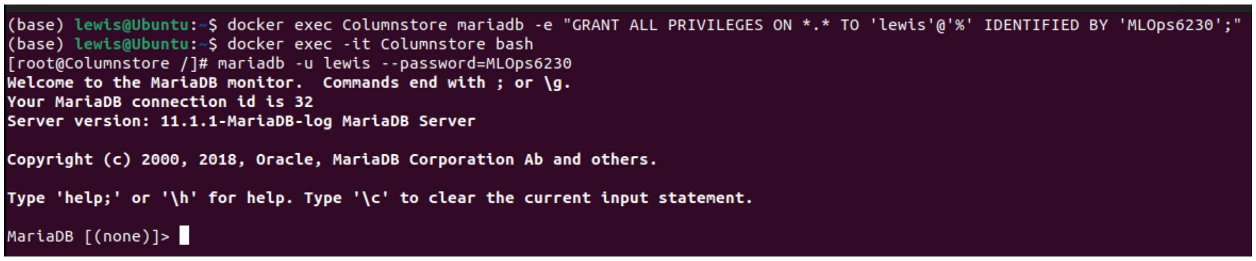
MariaDB Columnstore

MariaDB Columnstore runs on port 3306, but is accessed through port 3307.



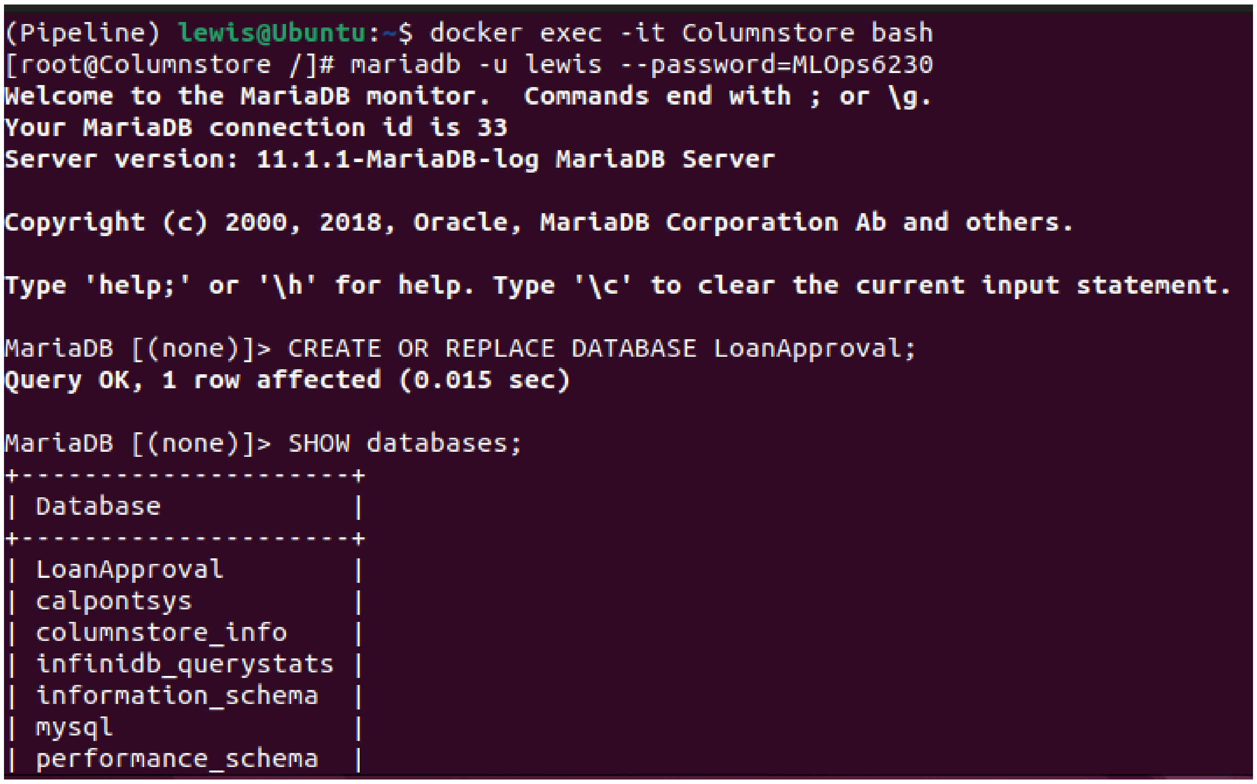
```
(base) lewis@Ubuntu:~$ docker run -d -p 3307:3306 --shm-size=512m -e PM1=Columnstore --hostname=Columnstore --name Columnstore mariadb/columnstore
4721551029af8588519d351819556aadd3e49ce907174b977ac792c394a81117
(base) lewis@Ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS                                         NAMES
4721551029af   mariadb/columnstore   "/usr/bin/tini -- do…"   14 seconds ago   Up 14 seconds   0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp   Columnstore
```

Figure 3.6: Creating the MariaDB Columnstore container.



```
(base) lewis@Ubuntu:~$ docker exec Columnstore mariadb -e "GRANT ALL PRIVILEGES ON *.* TO 'lewis'@'%' IDENTIFIED BY 'MLOps6230';"
(base) lewis@Ubuntu:~$ docker exec -it Columnstore bash
[root@Columnstore /]# mariadb -u lewis --password=MLOps6230
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 32
Server version: 11.1.1-MariaDB-log MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Figure 3.7: Creating the user account for MariaDB.

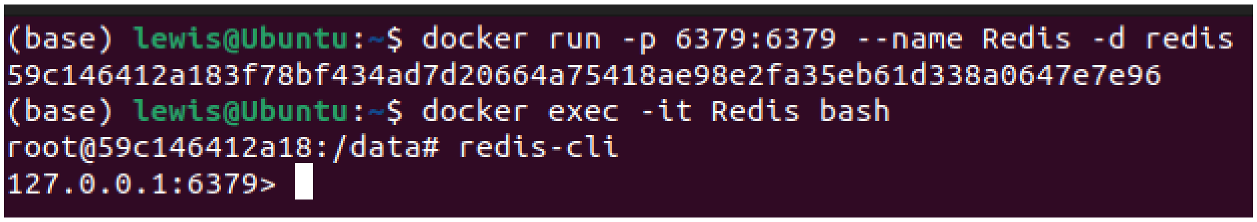


```
(Pipeline) lewis@Ubuntu:~$ docker exec -it Columnstore bash
[root@Columnstore /]# mariadb -u lewis --password=MLOps6230
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 33
Server version: 11.1.1-MariaDB-log MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE OR REPLACE DATABASE LoanApproval;
Query OK, 1 row affected (0.015 sec)

MariaDB [(none)]> SHOW databases;
+----------------------+
| Database             |
+----------------------+
| LoanApproval         |
| calpontsys           |
| columnstore_info     |
| infinidb_querystats  |
| information_schema   |
| mysql                |
| performance_schema   |
```

Figure 3.8: Creating the database for later use.

Redis

Redis runs on port 6379.



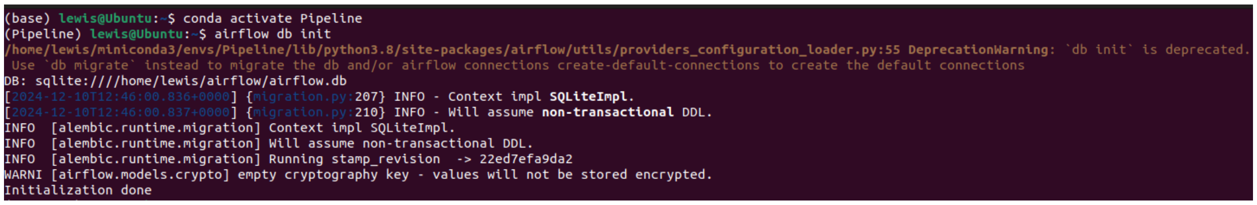
```
(base) lewis@Ubuntu:~$ docker run -p 6379:6379 --name Redis -d redis
59c146412a183f78bf434ad7d20664a75418ae98e2fa35eb61d338a0647e7e96
(base) lewis@Ubuntu:~$ docker exec -it Redis bash
root@59c146412a18:/data# redis-cli
127.0.0.1:6379>
```

Figure 3.9: Creating the Redis container.

3.2.4 Airflow and MLFlow initialisation

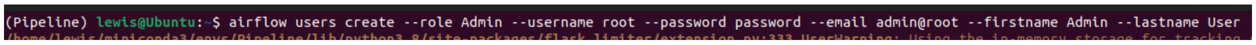
Airflow and MLFlow do not immediately work upon install, and must first be initialised.

Airflow



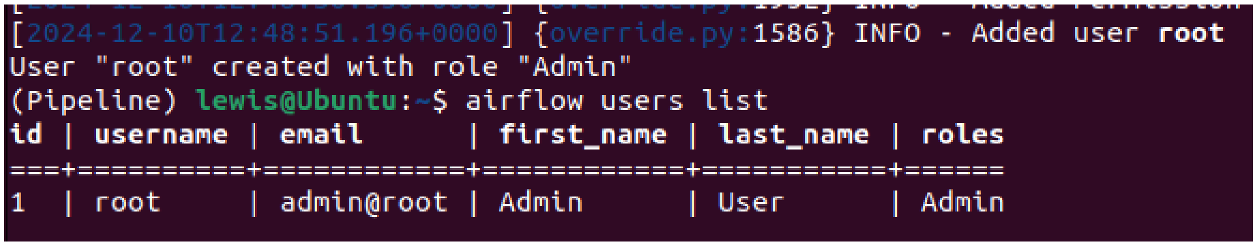
```
(base) lewis@Ubuntu:~$ conda activate Pipeline
(Pipeline) lewis@Ubuntu:~$ airflow db init
/home/lewis/miniconda3/envs/Pipeline/lib/python3.8/site-packages/airflow/utils/providers_configuration_loader.py:55 DeprecationWarning: `db init` is deprecated.
 Use `db migrate` instead to migrate the db and/or airflow connections create-default-connections to create the default connections
DB: sqlite:////home/lewis/airflow/airflow.db
[2024-12-10T12:46:00.836+0000] {migration.py:207} INFO - Context impl SQLiteImpl.
[2024-12-10T12:46:00.837+0000] {migration.py:210} INFO - Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running stamp_revision  -> 22ed7efa9da2
WARNI [airflow.models.crypto] empty cryptography key - values will not be stored encrypted.
Initialization done
```

Figure 3.10: Initialising Airflow's database.



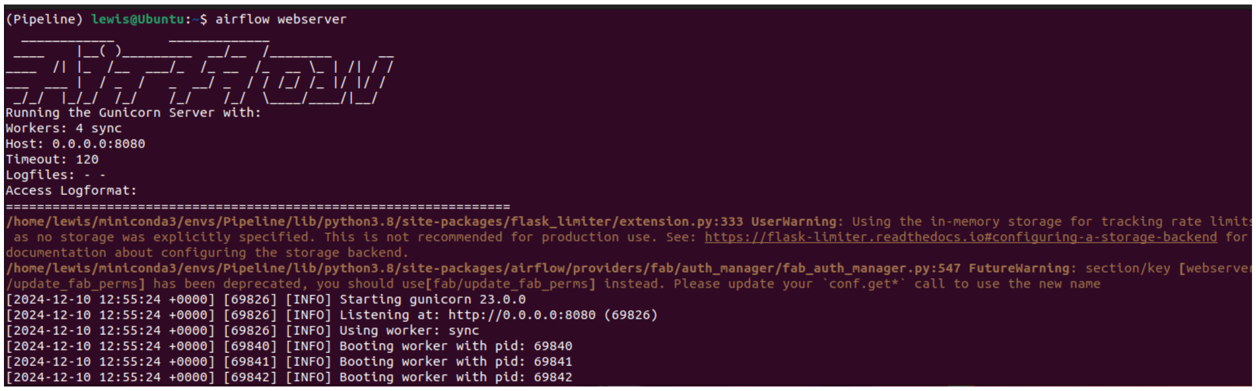
```
(Pipeline) lewis@Ubuntu:~$ airflow users create --role Admin --username root --password password --email admin@root --firstname Admin --lastname User
```

Figure 3.11: Creating an administrative Airflow user.



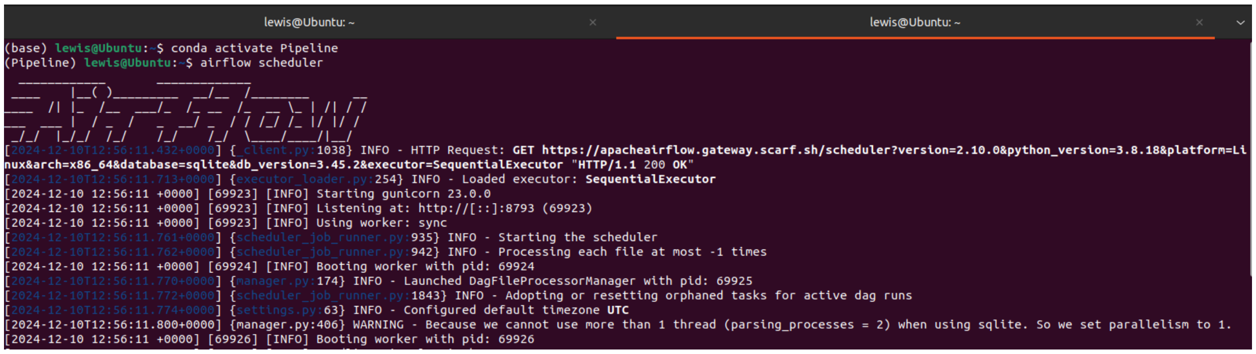
```
[2024-12-10T12:48:51.196+0000] {override.py:1586} INFO - Added user root
User "root" created with role "Admin"
(Pipeline) lewis@Ubuntu:~$ airflow users list
id | username | email      | first_name | last_name | roles
===+==========+============+============+===========+======
1  | root     | admin@root | Admin      | User      | Admin
```

Figure 3.12: Verifying that the user was created.



```
(Pipeline) lewis@Ubuntu:~$ airflow webserver
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
Running the Gunicorn Server with:
Workers: 4 sync
Host: 0.0.0.0:8080
Timeout: 120
Logfiles: - -
Access Logformat:
=================================================================
/home/lewis/miniconda3/envs/Pipeline/lib/python3.8/site-packages/flask_limiter/extension.py:333 UserWarning: Using the in-memory storage for tracking rate limits
 as no storage was explicitly specified. This is not recommended for production use. See: https://flask-limiter.readthedocs.io#configuring-a-storage-backend for
documentation about configuring the storage backend.
/home/lewis/miniconda3/envs/Pipeline/lib/python3.8/site-packages/airflow/providers/fab/auth_manager/fab_auth_manager.py:547 FutureWarning: section/key [webserver
/update_fab_perms] has been deprecated, you should use[fab/update_fab_perms] instead. Please update your `conf.get*` call to use the new name
[2024-12-10 12:55:24 +0000] [69826] [INFO] Starting gunicorn 23.0.0
[2024-12-10 12:55:24 +0000] [69826] [INFO] Listening at: http://0.0.0.0:8080 (69826)
[2024-12-10 12:55:24 +0000] [69826] [INFO] Using worker: sync
[2024-12-10 12:55:24 +0000] [69840] [INFO] Booting worker with pid: 69840
[2024-12-10 12:55:24 +0000] [69841] [INFO] Booting worker with pid: 69841
[2024-12-10 12:55:24 +0000] [69842] [INFO] Booting worker with pid: 69842
```

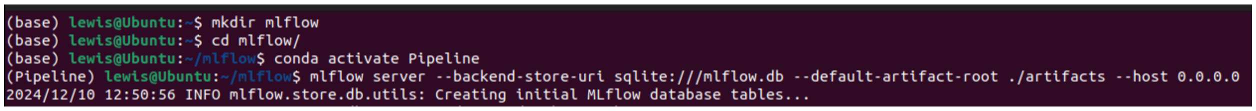
Figure 3.13: Successfully starting Airflow's web server.



```
(base) lewis@Ubuntu:~$ conda activate Pipeline
(Pipeline) lewis@Ubuntu:~$ airflow scheduler
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2024-12-10T12:56:11.432+0000] {_client.py:1038} INFO - HTTP Request: GET https://apacheairflow.gateway.scarf.sh/scheduler?version=2.10.0&python_version=3.8.18&platform=Li
nux&arch=x86_64&database=sqlite&db_version=3.45.2&executor=SequentialExecutor "HTTP/1.1 200 OK"
[2024-12-10T12:56:11.713+0000] {executor_loader.py:254} INFO - Loaded executor: SequentialExecutor
[2024-12-10 12:56:11 +0000] [69923] [INFO] Starting gunicorn 23.0.0
[2024-12-10 12:56:11 +0000] [69923] [INFO] Listening at: http://[::]:8793 (69923)
[2024-12-10 12:56:11 +0000] [69923] [INFO] Using worker: sync
[2024-12-10T12:56:11.761+0000] {scheduler_job_runner.py:935} INFO - Starting the scheduler
[2024-12-10T12:56:11.762+0000] {scheduler_job_runner.py:942} INFO - Processing each file at most -1 times
[2024-12-10 12:56:11 +0000] [69924] [INFO] Booting worker with pid: 69924
[2024-12-10T12:56:11.770+0000] {manager.py:174} INFO - Launched DagFileProcessorManager with pid: 69925
[2024-12-10T12:56:11.772+0000] {scheduler_job_runner.py:1843} INFO - Adopting or resetting orphaned tasks for active dag runs
[2024-12-10T12:56:11.774+0000] {settings.py:63} INFO - Configured default timezone UTC
[2024-12-10T12:56:11.800+0000] {manager.py:406} WARNING - Because we cannot use more than 1 thread (parsing_processes = 2) when using sqlite. So we set parallelism to 1.
[2024-12-10 12:56:11 +0000] [69926] [INFO] Booting worker with pid: 69926
```

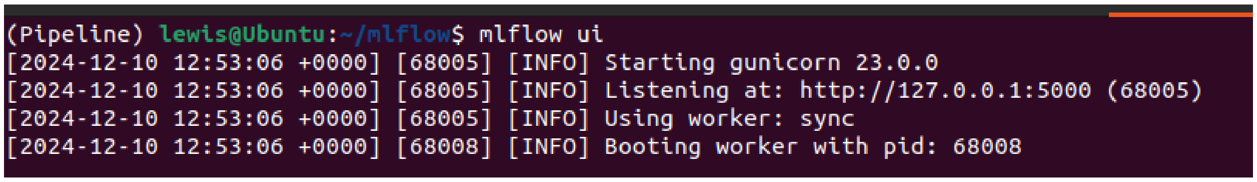
Figure 3.14: Successfully starting Airflow's task scheduler.

MLFlow



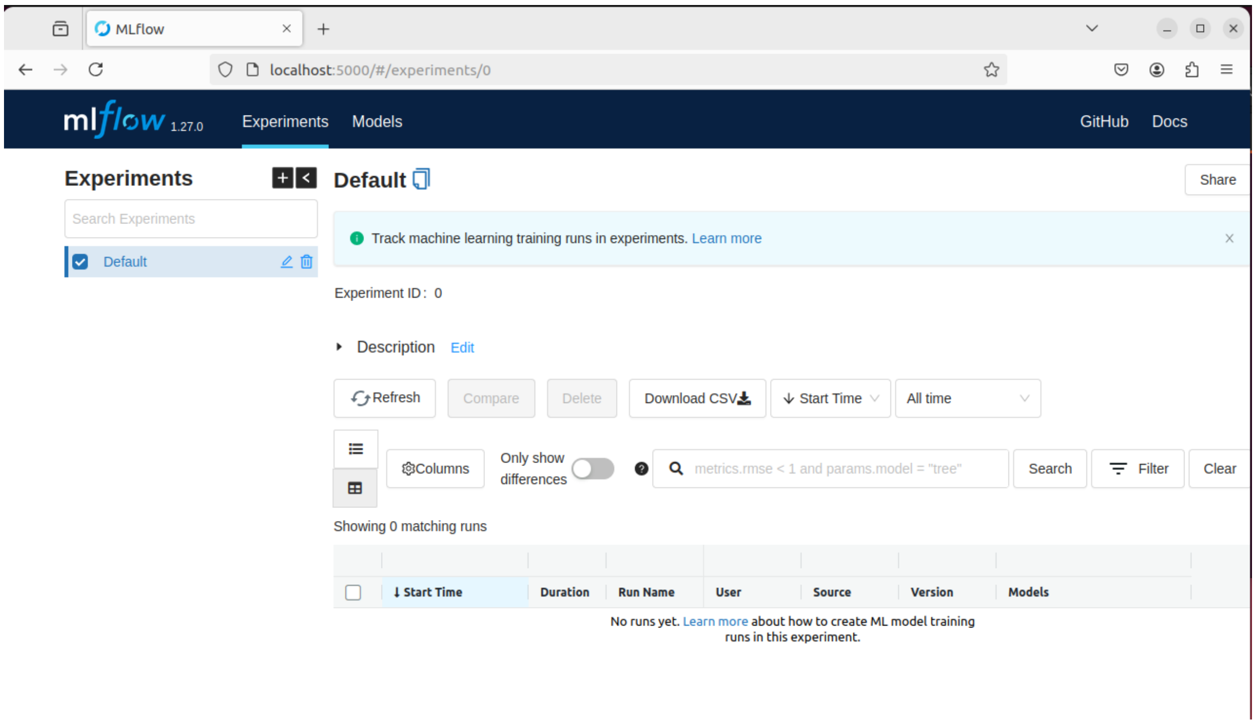
```
(base) lewis@Ubuntu:~$ mkdir mlflow
(base) lewis@Ubuntu:~$ cd mlflow/
(base) lewis@Ubuntu:~/mlflow$ conda activate Pipeline
(Pipeline) lewis@Ubuntu:~/mlflow$ mlflow server --backend-store-uri sqlite:///mlflow.db --default-artifact-root ./artifacts --host 0.0.0.0
2024/12/10 12:50:56 INFO mlflow.store.db.utils: Creating initial MLflow database tables...
```

Figure 3.15: Creating a directory for MLFlow and initialising its database.



```
(Pipeline) lewis@Ubuntu:~/mlflow$ mlflow ui
[2024-12-10 12:53:06 +0000] [68005] [INFO] Starting gunicorn 23.0.0
[2024-12-10 12:53:06 +0000] [68005] [INFO] Listening at: http://127.0.0.1:5000 (68005)
[2024-12-10 12:53:06 +0000] [68005] [INFO] Using worker: sync
[2024-12-10 12:53:06 +0000] [68008] [INFO] Booting worker with pid: 68008
```

Figure 3.16: Running MLFlow's frontend web UI.



Track machine learning training runs in experiments. Learn more

Experiment ID : 0

Showing 0 matching runs

	Start Time	Duration	Run Name	User	Source	Version	Models

No runs yet. Learn more about how to create ML model training runs in this experiment.

Figure 3.17: MLFlow's web UI.

3.3 Data Ingestion

3.4 Data Preprocessing

3.5 Model Development

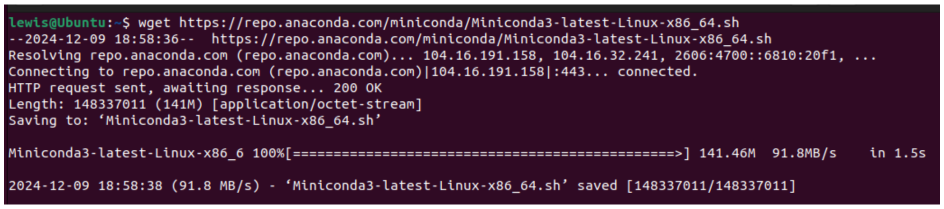
3.6 Model Deployment

3.7 Model Evaluation

Conclusion

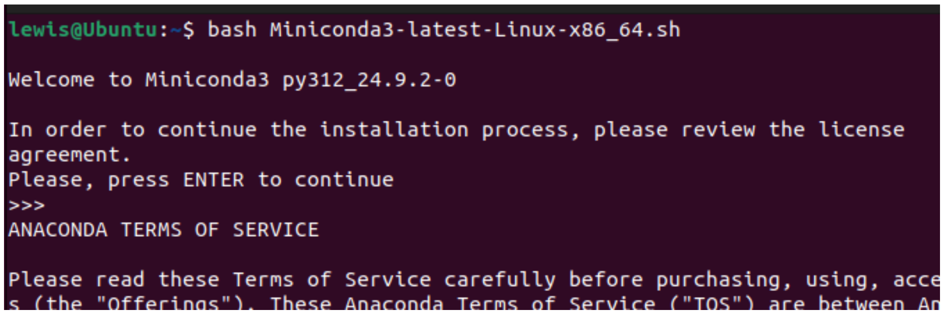
Appendix A - Software Installation

Miniconda



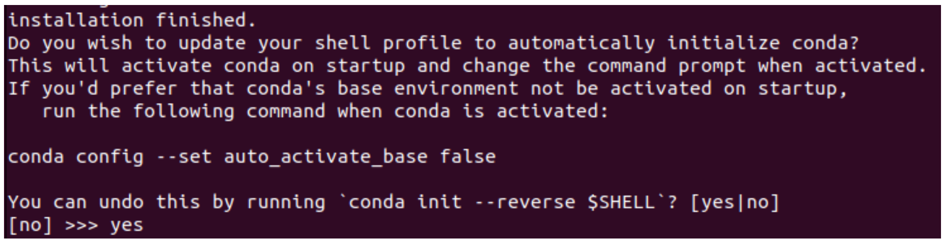
```
lewis@Ubuntu:~$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
--2024-12-09 18:58:36--  https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.191.158, 104.16.32.241, 2606:4700::6810:20f1, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.191.158|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 148337011 (141M) [application/octet-stream]
Saving to: 'Miniconda3-latest-Linux-x86_64.sh'

Miniconda3-latest-Linux-x86_6 100%[============================================>] 141.46M  91.8MB/s    in 1.5s

2024-12-09 18:58:38 (91.8 MB/s) - 'Miniconda3-latest-Linux-x86_64.sh' saved [148337011/148337011]
```

Figure 3.18: Getting the latest Miniconda installation script.



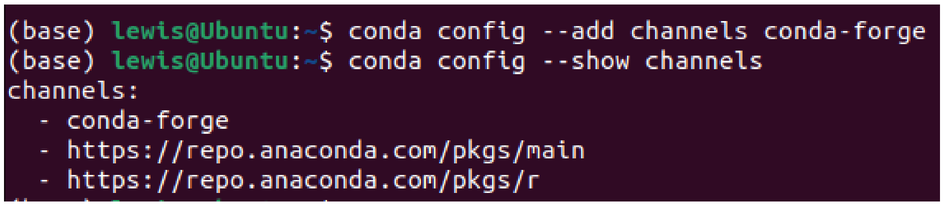
```
lewis@Ubuntu:~$ bash Miniconda3-latest-Linux-x86_64.sh

Welcome to Miniconda3 py312_24.9.2-0

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
ANACONDA TERMS OF SERVICE

Please read these Terms of Service carefully before purchasing, using, acce
s (the "Offerings"). These Anaconda Terms of Service ("TOS") are between An
```

Figure 3.19: Executing the Miniconda installation script.



```
installation finished.
Do you wish to update your shell profile to automatically initialize conda?
This will activate conda on startup and change the command prompt when activated.
If you'd prefer that conda's base environment not be activated on startup,
   run the following command when conda is activated:

conda config --set auto_activate_base false

You can undo this by running `conda init --reverse $SHELL`? [yes|no]
[no] >>> yes
```

Figure 3.20: Making Conda run automatically in all shells.

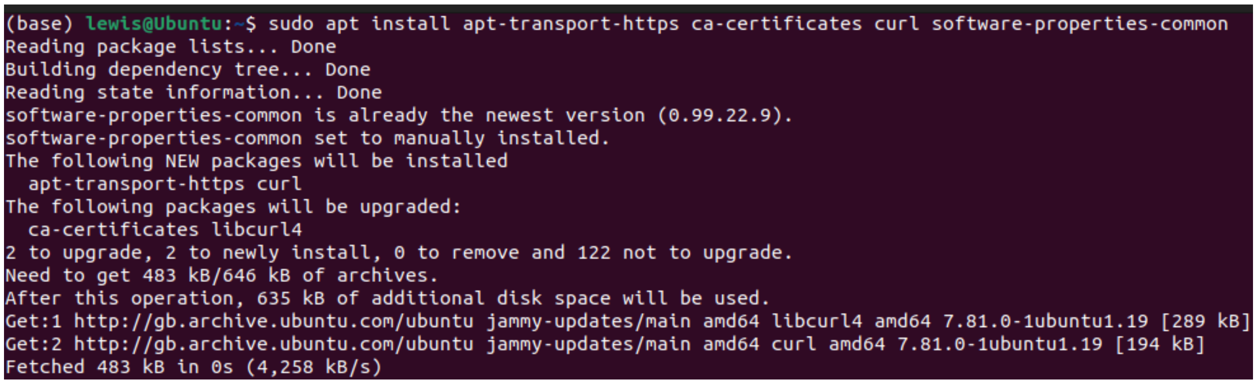


```
(base) lewis@Ubuntu:~$ conda config --add channels conda-forge
(base) lewis@Ubuntu:~$ conda config --show channels
channels:
  - conda-forge
  - https://repo.anaconda.com/pkgs/main
  - https://repo.anaconda.com/pkgs/r
```

Figure 3.21: Adding conda-forge as a channel for package retrieval.

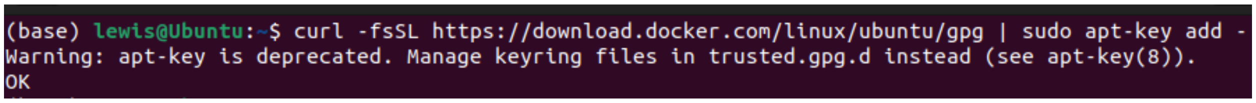
Docker

Docker's install process on a fresh Ubuntu system is quite lengthy due to a lot of system configuration being required. This installation process follows DigitalOcean (2024)'s guide.



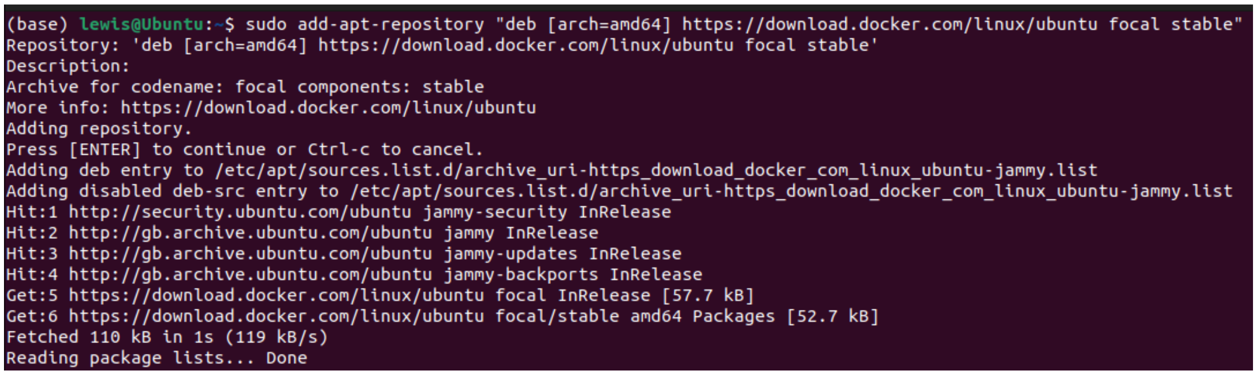
```
(base) lewis@Ubuntu:~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
software-properties-common is already the newest version (0.99.22.9).
software-properties-common set to manually installed.
The following NEW packages will be installed
  apt-transport-https curl
The following packages will be upgraded:
  ca-certificates libcurl4
2 to upgrade, 2 to newly install, 0 to remove and 122 not to upgrade.
Need to get 483 kB/646 kB of archives.
After this operation, 635 kB of additional disk space will be used.
Get:1 http://gb.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libcurl4 amd64 7.81.0-1ubuntu1.19 [289 kB]
Get:2 http://gb.archive.ubuntu.com/ubuntu jammy-updates/main amd64 curl amd64 7.81.0-1ubuntu1.19 [194 kB]
Fetched 483 kB in 0s (4,258 kB/s)
```

Figure 3.22: Installing HTTPS certificates.



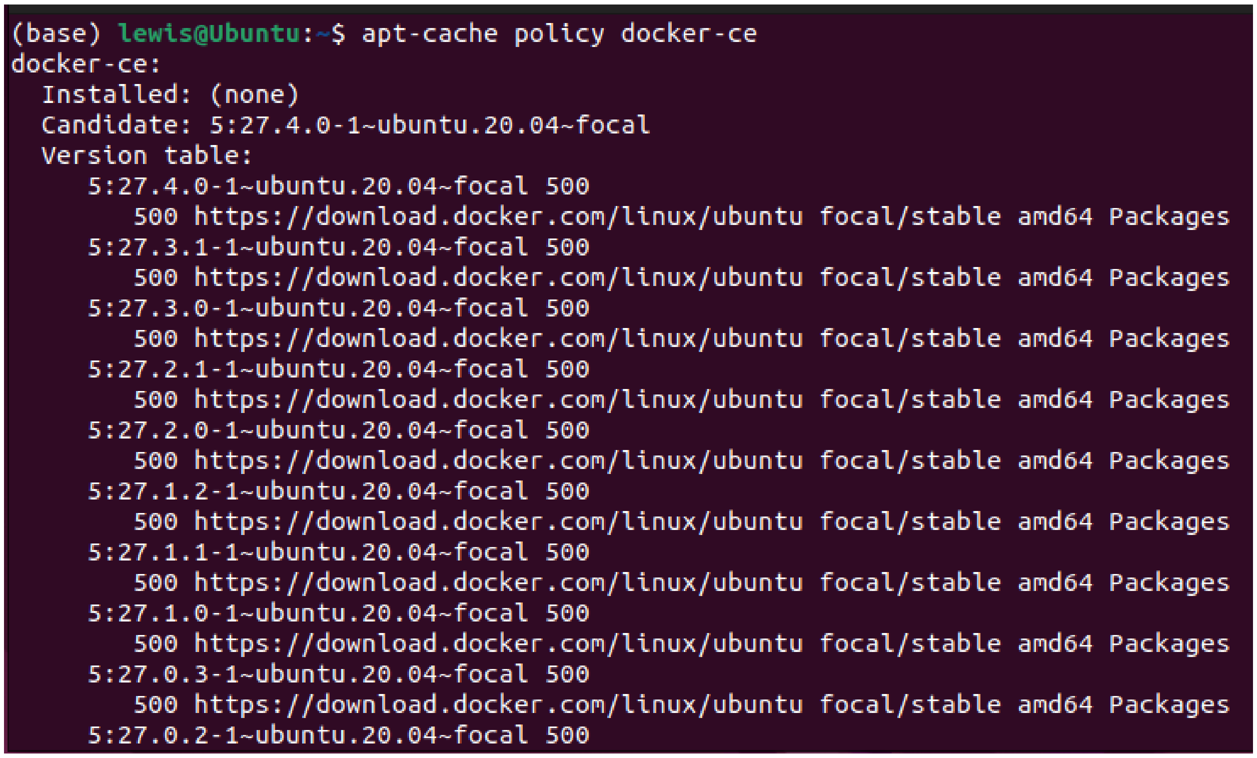
```
(base) lewis@Ubuntu:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
```

Figure 3.23: Adding Docker's public GPG key for checksum validation.



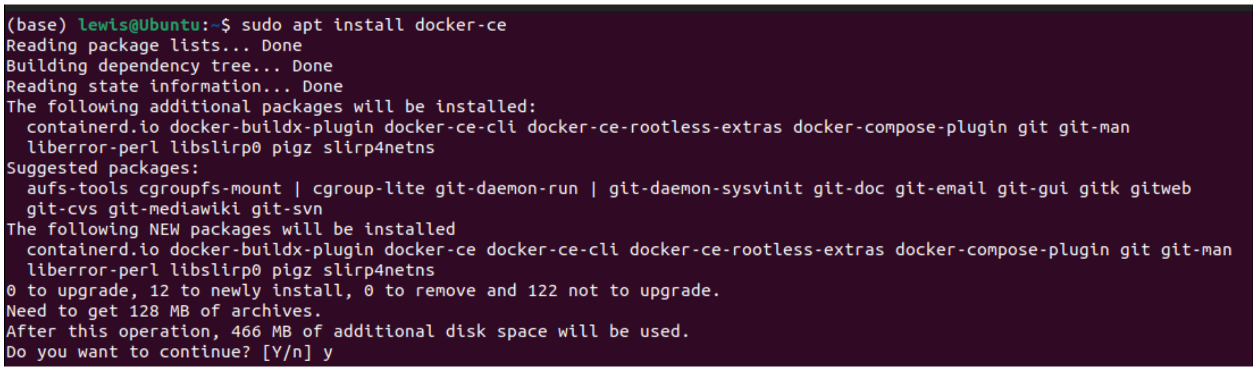
```
(base) lewis@Ubuntu:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable'
Description:
Archive for codename: focal components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:2 http://gb.archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 http://gb.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:4 http://gb.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:5 https://download.docker.com/linux/ubuntu focal InRelease [57.7 kB]
Get:6 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [52.7 kB]
Fetched 110 kB in 1s (119 kB/s)
Reading package lists... Done
```

Figure 3.24: Adding Docker's download page to Ubuntu's list of package repositories.



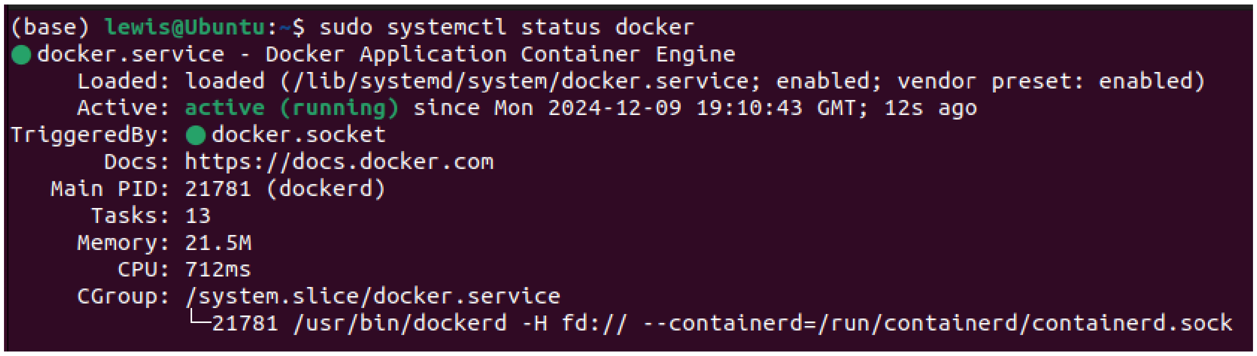
```
(base) lewis@Ubuntu:~$ apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 5:27.4.0-1~ubuntu.20.04~focal
  Version table:
     5:27.4.0-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.3.1-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.3.0-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.2.1-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.2.0-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.1.2-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.1.1-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.1.0-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.0.3-1~ubuntu.20.04~focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:27.0.2-1~ubuntu.20.04~focal 500
```

Figure 3.25: Checking if Docker is installed, which it is not.



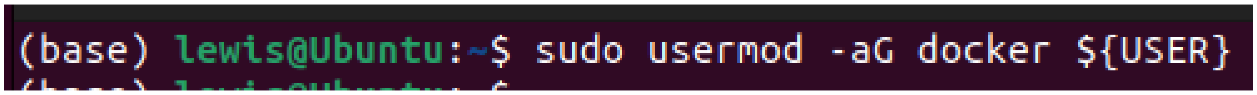
```
(base) lewis@Ubuntu:~$ sudo apt install docker-ce
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin git git-man
  liberror-perl libslirp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb
  git-cvs git-mediawiki git-svn
The following NEW packages will be installed
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin git git-man
  liberror-perl libslirp0 pigz slirp4netns
0 to upgrade, 12 to newly install, 0 to remove and 122 not to upgrade.
Need to get 128 MB of archives.
After this operation, 466 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Figure 3.26: Installing Docker.



```
(base) lewis@Ubuntu:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2024-12-09 19:10:43 GMT; 12s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 21781 (dockerd)
      Tasks: 13
     Memory: 21.5M
        CPU: 712ms
     CGroup: /system.slice/docker.service
             └─21781 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Figure 3.27: Confirming that the Docker service is running.



```
(base) lewis@Ubuntu:~$ sudo usermod -aG docker ${USER}
```

Figure 3.28: Adding the VM user to the Docker group so they can execute commands.

Bibliography

Anaconda (2024). *About Anaconda*. Anaconda. URL: https://www.anaconda.com/about-us (visited on 11/22/2024).

Apache (2024a). *Apache License, Version 2.0*. URL: https://www.apache.org/licenses/LICENSE-2.0 (visited on 12/05/2024).

Apache (2024b). *Streaming, Serialization, and IPC — Apache Arrow v18.0.0*. URL: https://arrow.apache.org/docs/python/ipc.html (visited on 11/18/2024).

Apache (2024c). *Use Cases*. Apache Airflow. URL: https://airflow.apache.org/use-cases/ (visited on 11/18/2024).

AWS (2024a). *ETL vs ELT - Difference Between Data-Processing Approaches - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/compare/the-difference-between-etl-and-elt/ (visited on 11/17/2024).

AWS (2024b). *OLTP vs OLAP - Difference Between Data Processing Systems - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/compare/the-difference-between-olap-and-oltp/ (visited on 11/16/2024).

AWS (2024c). *What is Docker? | AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/docker/ (visited on 11/18/2024).

Bartley, Kevin (Oct. 30, 2024). *ETL vs ELT: Key Differences, Comparisons, & Use Cases*. Rivery. URL: https://rivery.io/blog/etl-vs-elt/ (visited on 11/17/2024).

Bendi Ramana and N. Venkateswarlu (2022). *ILPD (Indian Liver Patient Dataset)*. DOI: 10.24432/C5D02C.

Butterfield, Sean (2024). *22b Lesson - Pitch-class integer notation*. Inquiry-Based Music Theory. URL: https://smbutterfield.github.io/ibmt17-18/22-intro-to-non-diatonic-materials/b2-tx-pcintnotation.html (visited on 11/12/2024).

CFI (2024). *R-Squared - Definition, Interpretation, Formula, How to Calculate*. URL: https://corporatefinanceinstitute.com/resources/data-science/r-squared/ (visited on 11/18/2024).

Cleveland Clinic (2024). *Alkaline Phosphatase (ALP): What It Is, Causes & Treatment*. Cleveland Clinic. URL: https://my.clevelandclinic.org/health/diagnostics/22029-alkaline-phosphatase-alp (visited on 11/12/2024).

Cloudflare (2024). *What is data privacy? | Privacy definition*. URL: https://www.cloudflare.com/learning/privacy/what-is-data-privacy/ (visited on 12/06/2024).

Creative Commons (2024). *Deed - CC0 1.0 Universal - Creative Commons*. URL: https://creativecommons.org/publicdomain/zero/1.0/ (visited on 11/21/2024).

DataCamp (2024). *Understanding Data Drift and Model Drift: Drift Detection in Python*. URL: https://www.datacamp.com/tutorial/understanding-data-drift-model-drift (visited on 12/10/2024).

DigitalOcean (2024). *How To Install and Use Docker on Ubuntu 20.04 | DigitalOcean*. URL: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04 (visited on 12/09/2024).

Docker (May 10, 2022). *Docker: Accelerated Container Application Development*. URL: https://www.docker.com/ (visited on 11/18/2024).

Docker Hub (2024). *mariadb/columnstore - Docker Image | Docker Hub*. URL: https://hub.docker.com/r/mariadb/columnstore (visited on 11/18/2024).

FastAPI (2024). *FastAPI*. URL: https://fastapi.tiangolo.com/ (visited on 11/18/2024).

GeeksForGeeks (June 10, 2023). *Snowflake Schema in Data Warehouse Model*. GeeksforGeeks. URL: https://www.geeksforgeeks.org/snowflake-schema-in-data-warehouse-model/ (visited on 11/21/2024).

Gupta, Kanishk, Binayak Chakrabarti, Aseer Ahmad Ansari, Siddharth Swarup Rautaray, and Manjusha Pandey (Apr. 24, 2021). *Loanification - Loan Approval Classification using Machine Learning Algorithms*. Rochester, NY. DOI: 10.2139/ssrn.3833303.

GX (2024). *GX Core: a powerful, flexible data quality solution*. URL: https://www.greatexpectations.io/gx-core (visited on 11/18/2024).

Harvard Business School (Mar. 16, 2021). *5 Principles of Data Ethics for Business*. Business Insights Blog. URL: https://online.hbs.edu/blog/post/data-ethics (visited on 12/07/2024).

IBM (June 11, 2021a). *What Is Data Security? | IBM*. URL: https://www.ibm.com/topics/data-security (visited on 12/07/2024).

IBM (Oct. 4, 2021b). *What is the k-nearest neighbors algorithm? | IBM*. URL: https://www.ibm.com/topics/knn (visited on 11/17/2024).

ICO (Nov. 13, 2024a). *A guide to individual rights*. Publisher: ICO. URL: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/individual-rights/individual-rights/ (visited on 12/07/2024).

ICO (Nov. 27, 2024b). *Enforcement of this code*. Publisher: ICO. URL: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/data-sharing/data-sharing-a-code-of-practice/enforcement-of-this-code/ (visited on 12/07/2024).

InCycle Software (2024). *MLOps ENTERPRISE ACCELERATOR*. URL: https://www.incyclesoftware.com/azure-machine-learning-enterprise-accelerator (visited on 11/13/2024).

Kaggle (2024). *Kaggle: Your Home for Data Science*. URL: https://www.kaggle.com/ (visited on 11/15/2024).

Kaminsky, Michael (2024). *Star Schema vs. OBT for Data Warehouse Performance | Blog | Fivetran*. URL: https://www.fivetran.com/blog/star-schema-vs-obt (visited on 11/21/2024).

Kundu, Rohit (2024). *F1 Score in Machine Learning: Intro & Calculation*. URL: https://www.v7labs.com/blog/f1-score-guide (visited on 11/17/2024).

Lo, Ta-Wei (2024). *Loan Approval Classification Dataset*. URL: https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data (visited on 11/15/2024).

Lopez, Cesar and Abdullah Elbi (2022). "On the legal nature of synthetic data". In.

MariaDB (2024). *MariaDB ColumnStore*. MariaDB KnowledgeBase. URL: https://mariadb.com/kb/en/mariadb-columnstore/ (visited on 11/18/2024).

Mayo Clinic (2024). *Bilirubin test - Mayo Clinic*. URL: https://www.mayoclinic.org/tests-procedures/bilirubin/about/pac-20393041 (visited on 11/12/2024).

MIcrosoft (June 6, 2024). *Database normalization description - Microsoft 365 Apps*. URL: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description (visited on 11/21/2024).

Microsoft (Aug. 28, 2024). *SMOTE - Azure Machine Learning*. URL: https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/smote?view=azureml-api-2 (visited on 11/22/2024).

MLFlow (2024). *MLFlow*. URL: mlflow.org (visited on 11/18/2024).

Nigenda, David, Zohar Karnin, Muhammad Bilal Zafar, Raghu Ramesha, Alan Tan, Michele Donini, and Krishnaram Kenthapadi (Aug. 5, 2022). *Amazon SageMaker Model Monitor: A System for Real-Time Insights into Deployed Machine Learning Models*. DOI: 10.48550/arXiv.2111.13657. arXiv: 2111.13657[cs].

Oracle (2024). *Oracle VirtualBox*. URL: https://www.virtualbox.org/ (visited on 12/07/2024).

pandas (2024). *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/about/ (visited on 11/18/2024).

Python (2024). *pickle — Python object serialization*. Python documentation. URL: https://docs.python.org/3/library/pickle.html (visited on 11/21/2024).

RedHat (2024). *What is a REST API?* URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (visited on 11/17/2024).

RestfulAPI (Dec. 12, 2023). *What is REST?* REST API Tutorial. URL: https://restfulapi.net/ (visited on 11/17/2024).

Richards, Neil M. and Jonathan King (May 19, 2014). *Big Data Ethics*. Rochester, NY.

scikit-learn (2024). *scikit-learn: machine learning in Python — scikit-learn 1.5.2 documentation*. URL: https://scikit-learn.org/stable/ (visited on 11/18/2024).

Sky News (2024). *World's oldest man dies aged 112, Guinness World Records announces*. Sky News. URL: https://news.sky.com/video/worlds-oldest-man-dies-aged-112-guinness-world-records-announces-13260926 (visited on 12/08/2024).

Smallcombe, Mark (2024). *ETL vs ELT: 5 Critical Differences*. Integrate.io. URL: https://www.integrate.io/blog/etl-vs-elt/ (visited on 11/17/2024).

Spotify (2024). *Web API Reference | Spotify for Developers*. URL: https://developer.spotify.com/documentation/web-api/reference/get-audio-features (visited on 11/12/2024).

SQLAlchemy (2024). *SQLAlchemy*. URL: https://www.sqlalchemy.org (visited on 11/18/2024).

Straw, Isabel and Honghan Wu (Apr. 25, 2022). "Investigating for bias in healthcare algorithms: a sex-stratified analysis of supervised machine learning models in liver disease prediction". In: *BMJ Health & Care Informatics* 29 (1). Publisher: BMJ Publishing Group Ltd. ISSN: 2632-1009. DOI: 10.1136/bmjhci-2021-100457.

TowardsDataScience (May 13, 2021). *Getting Started with Conda. Just the basics. What is Conda? Why... | by...* archive.ph. URL: https://archive.ph/OTuGh (visited on 11/18/2024).

UCI Machine Learning Repository (2024). *About - UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu/about (visited on 11/12/2024).

Uvicorn (2024). *Uvicorn*. URL: https://www.uvicorn.org/ (visited on 11/18/2024).

Vergnou, Brice (July 27, 2021). *Brice-Vergnou/spotify_recommendation*. URL: https://github.com/Brice-Vergnou/spotify_recommendation (visited on 11/16/2024).

Vergnou, Brice (2024). *Spotify Recommendation*. URL: https://www.kaggle.com/datasets/bricevergnou/spotify-recommendation (visited on 11/21/2024).

Yang, Jing (Sept. 2021). "Big Data Privacy Protection Technology". In: *Journal of Physics: Conference Series* 2037 (1). Publisher: IOP Publishing, p. 012136. ISSN: 1742-6596. DOI: 10.1088/1742-6596/2037/1/012136.

Zoppelleto, Lorenzo (2024). *Financial Risk for Loan Approval*. URL: https://www.kaggle.com/datasets/lorenzozoppelletto/financial-risk-for-loan-approval (visited on 12/07/2024).