

Student name:  
nirnaya khadka

Student id:  
24128420



## BIRMINGHAM CITY UNIVERSITY

Student name: nirnaya khadka

Student id: 24128420

Report Introduction .....	7
Data Analytics Pipeline: Planning and Design .....	7
Identification of three candidate source data sets .....	8
❖ Mobile Price Classification Dataset .....	9
Feedback why not choosen this dataset .....	10
❖ Loan Eligibility Dataset.....	10
Feedback why not choosen this dataset .....	11
LinkedIn Jobs ML Dataset .....	12
Feedback why not choosen this dataset .....	12
Chosen source data set.....	13
Server setup .....	13
Setup OS.....	13
File and vscode setup.....	14
EDR.....	15
EDR Features:.....	16
1. Dataset Overview Analysis .....	17
Job Category Distribution Analysis .....	17

Student name:  
nirnaya khadka

Student id:  
24128420

Automated Class Imbalance Handling .....	18
Integration with Preprocessing Pipeline .....	18
Text Content Analysis.....	19
Integration with Preprocessing Pipeline .....	19
Summary .....	20
Technology Stack Justification .....	20
Data Flow Documentation .....	20
Initial Data Acquisition .....	21
ELT.....	21
Complete Pipeline Architecture.....	22
Development.....	22
DAG pipeline .....	24
Data Preprocessing Stage and Feature Engineering .....	25
Preprocessing.py .....	25
model training, evaluation, logging .....	26
model.py .....	26
Model deployment .....	27
Api.py .....	27
Data Ingestion Stage .....	27
ingestion.py.....	28
Ingestion to Validation .....	28
Main.py .....	29
Config Management .....	29
config.py .....	30
Deployment & Monitoring .....	30
start.sh .....	31
Testing and Experimentation Tracking .....	31

Student name:  
nirnaya khadka

Student id:  
24128420

Production .....	32
Step 3 Conda setup .....	32
Setup Docker .....	33
Airflow setup .....	33
Mlflow .....	34
Fast api .....	37
Pipeline creation .....	38
Privacy, Security, Ethics:.....	40
Data.....	40
Data privacy .....	40
Data security .....	40
Legal.....	40
Feature Engineering.....	41
Data transformation.....	42
Smart Save/Load .....	43
Security.....	43
RDBMS Data Insertion .....	43
OLTP production security.....	44
Columnstore Data insertion .....	45
Data prediction .....	45
Training the model.....	48
Data cleaning .....	48
Feature Engineering .....	48
Text-to-Numbers Conversion .....	48
Training the model.....	48
Logistic Regression .....	49
Random Forest .....	49

Student name:  
nirnaya khadka

Student id:  
24128420

Support Vector Machine (SVM).....	50
Naive Bayes.....	50
Ensemble Voting.....	50
Data Flow Optimization Points .....	50
Scalability Considerations .....	51
Final result.....	51
Result of ml flow .....	52
Result of fast api .....	55
Summative Feedback Integration .....	56
Refrences .....	58
J. Furnkranz, Machine Learning and Data Mining, Springer-Verlag Berlin Heidelberg, 2012. .....	58
Report Introduction .....	7
Data Analytics Pipeline: Planning and Design .....	7
Identification of three candidate source data sets .....	8
❖ Mobile Price Classification Dataset .....	9
Feedback why not choosen this dataset .....	10
❖ Loan Eligibility Dataset.....	10
Feedback why not choosen this dataset .....	11
LinkedIn Jobs ML Dataset .....	12
Feedback why not choosen this dataset .....	12
Chosen source data set.....	13
Server setup .....	13
Setup OS.....	13
File and vscode setup .....	14
EDR.....	15
EDR Features:.....	16

Student name:  
nirnaya khadka

Student id:  
24128420

1. Dataset Overview Analysis .....	17
Job Category Distribution Analysis .....	17
Automated Class Imbalance Handling .....	18
Integration with Preprocessing Pipeline .....	18
Text Content Analysis.....	19
Integration with Preprocessing Pipeline .....	19
Summary .....	20
Technology Stack Justification .....	20
Data Flow Documentation .....	20
Initial Data Acquisition .....	21
ELT.....	21
Complete Pipeline Architecture.....	22
Development.....	22
DAG pipeline .....	24
Data Preprocessing Stage and Feature Engineering .....	25
Preprocessing.py .....	25
model training, evaluation, logging .....	26
model.py.....	26
Model deployment .....	27
Api.py .....	27
Data Ingestion Stage .....	27
ingestion.py.....	28
Ingestion to Validation .....	28
Main.py .....	29
Config Management .....	29
config.py .....	30
Deployment & Monitoring .....	30

Student name:  
nirnaya khadka

Student id:  
24128420

start.sh .....	31
Testing and Experimentation Tracking .....	31
Production .....	32
Step 3 Conda setup .....	32
Setup Docker .....	33
Airflow setup .....	33
Mlflow .....	34
Fast api .....	37
Pipeline creation .....	38
Privacy, Security, Ethics:.....	40
Data.....	40
Data privacy .....	40
Data security .....	40
Legal .....	40
Feature Engineering.....	41
Data transformation.....	42
Smart Save/Load .....	43
Security .....	43
RDBMS Data Insertion .....	43
OLTP production security.....	44
Columnstore Data insertion .....	45
Data prediction .....	45
Training the model.....	48
Data cleaning .....	48
Feature Engineering .....	48
Text-to-Numbers Conversion .....	48
Training the model.....	48

Student name:  
nirnaya khadka

Student id:  
24128420

Logistic Regression .....	49
Random Forest .....	49
Support Vector Machine (SVM).....	50
Naive Bayes.....	50
Ensemble Voting.....	50
Data Flow Optimization Points .....	50
Scalability Considerations .....	51
Final result.....	51
Result of ml flow .....	52
Result of fast api .....	55
Summative Feedback Integration .....	56
References .....	58
J. Furnkranz, Machine Learning and Data Mining, Springer-Verlag Berlin Heidelberg, 2012. .....	58

## Report Introduction

Machine learning (ML) is a known area of computer science that mainly deals with the discovery of data patterns and data-related irregularities([H. S. Obaid](#)) . This report focusses on designing and planning a data analytics pipeline using linkedin data source. The research covers how the data will be used for ipleentation and how it can answer the key questions and discusses ethical and legal issues related to data security and privacy.

## Data Analytics Pipeline: Planning and Design

This section explains how to select a dataset for creating a supervised machine learning model and how to understand data analytics pipelines. It will explain how data will be managed, stored, and analyzed in order to build machine learning models and use analytics approaches.

Student name:  
nirnaya khadka

Student id:  
24128420

## Identification of three candidate source data sets

Machine learning (ML) is a known area of computer science that mainly deals with the discovery of data patterns and data-related irregularities ([furnkraz](#))

I have considered 3 different data set Mobile Price Classification Dataset, Loan Eligibility Dataset, LinkedIn Jobs ML Dataset


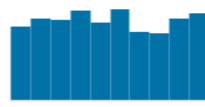






Student name:  
nirnaya khadka

Student id:  
24128420

### ❖ Mobile Price Classification Dataset

The Mobile pricing Classification dataset shows the pricing range of mobile phones, as well as phone features like internal memory, RAM, and battery power. Similarly, this is an open-source dataset gathered using Kaggle. This dataset was created to predict a mobile phone's pricing range based on its specifications. It is saved as a CSV file. The information can be used to train machine learning models that will categorize phones into different pricing categories based on factors like RAM and battery life. The "ground truth" in this dataset is the price range. This dataset has previously been used to train machine learning models because it is publicly available.

Detail	Compact	Column	10 of 21 columns			
# id	# battery_power	# blue	# clock_speed	# dual_sim	# fc	
ID	Total energy a battery can store in one time measured in mAh	Has bluetooth or not	speed at which microprocessor executes instructions	Has dual sim support or not	Front Cam pixels	
						
1	1000	0	0.5	0	0	
1	1043	1	1.8	1	14	
2	841	1	0.5	1	4	
3	1807	1	2.8	0	1	
4	1546	0	0.5	1	18	
5	1434	0	1.4	0	11	
6	1464	1	2.9	1	5	
7	1718	0	2.4	0	1	
8	833	0	2.4	1	0	
9	1111	1	2.9	1	9	
10	1520	0	0.5	0	1	
11	1500	0	2.2	0	2	
12	1343	0	2.9	0	2	
13	900	1	1.4	1	0	
14	1190	1	2.2	1	5	
15	630	0	1.8	0	8	

***Fig: Mobile Price Classification dataset***

Student name:  
nirnaya khadka

Student id:  
24128420

### Feedback why not choosen this dataset

I did not use the Mobile Price Classification dataset since it is overly clean, basic, and lacks real-world complexity. It has a limited feature set and few preprocessing difficulties, making it unsuitable for building a thorough ML pipeline.

#### ❖ Loan Eligibility Dataset

The Loan Eligibility dataset contains information about loan applicants, such as age, education level, and loan amount. This dataset is also open source and was collected through Kaggle. The primary goal of this dataset is to identify a person's eligibility for a loan based on personal and financial information. The dataset is saved in CSV format to local storage. We can utilize database systems like MySQL to store the database. This dataset's features, such as credit score and loan amount, can be used to train a classification model for loan approval. The "ground truth" is whether the loan has been granted or not. This dataset has previously been used in machine learning models because it is open source and widely used in research.

Student name:  
nirnaya khadka

Student id:  
24128420

About this file

Suggest Edits

Use this file data for testing your model and predict with this data.

Loan_ID	Gender	Married	Dependents	Education	Self_Employed
Unique Loan Id	Customer Gender	Customer married status	From which department	Customer Education status	Employment status
367 unique values	Male 78% Female 19% Other (11) 3%	 true 233 63% false 134 37%	0 54% 2 16% Other (108) 29%	Graduate 77% Not Graduate 23%	
LP001015	Male	Yes	0	Graduate	No
LP001022	Male	Yes	1	Graduate	No
LP001031	Male	Yes	2	Graduate	No
LP001035	Male	Yes	2	Graduate	No
LP001051	Male	No	0	Not Graduate	No
LP001054	Male	Yes	0	Not Graduate	Yes
LP001055	Female	No	1	Not Graduate	No
LP001056	Male	Yes	2	Not Graduate	No
LP001059	Male	Yes	2	Graduate	
LP001067	Male	No	0	Not Graduate	No
LP001078	Male	No	0	Not Graduate	No

**Fig: Loan Eligibility dataset**

### Feedback why not choosen this dataset

This dataset was not chosen for the machine learning pipeline because of its small size and lack of complexity. It comprises relatively few features and samples, which may not provide adequate data diversity or challenge to design and test a strong, scalable machine learning model.

Student name:  
nirnaya khadka

Student id:  
24128420

### LinkedIn Jobs ML Dataset

The LinkedIn Jobs ML dataset includes job posting details like titles, required skills, aataset is not stored in a database engine, it can be loaded into one for scalability. When building a regression model for pay prediction, it's vital to include skills, geography, and job type. In this scenario, the "ground truth" is to predict the job category based on **Confidence**. This dataset has already been used to train machine learning models.

#### ABOUT THIS FILE

Data Dictionary for the respective columns contained within LinkedInJobs\_MLDataset.csv

#### LinkedInJobs\_MLDataset Data Dictionary

- Co\_Nm / Company Name, dtype Object
- Co\_Pg\_Lstd / Company Page Listed, dtype Bool
- Emp\_Cnt / Company Employee Count, dtype int64
- Flw\_Cnt / Company Follower Count, dtype int64
- Job\_Ttl / Job Title, dtype Object
- Job\_Desc / Job Description, dtype Object
- Is\_Supvr / Is Post a Supervisor Position (Calculated), dtype Bool
- max\_sal / Maximum Salary, dtype Float64
- med\_sal / Median Salary, dtype Float64
- min\_sal / Minimum Salary, dtype Float64
- py\_prd / Pay Period, dtype Category {Not Listed, YEARLY, HOURLY, MONTHLY, Unpaid, WEEKLY, ONCE}
- py\_lstd / Pay Listed (Calculated), dtype Bool
- wrk\_tpy / Work Type, dtype Category {Full-time, Contract, Part-time, Temporary, Internship, Other, Volunteer}
- loc / Job Location, dtype Object
- st\_code / Job State Code (Calculated), dtype Object
- is\_remote / Is Job Remote (Calculated), dtype Bool
- views / Number of Posting Views, dtype int64
- app\_tpy / Application Type, dtype Category {Offsite Apply, SimpleOnSiteApply, ComplexOnSiteApply}
- app\_is\_off / Is Application Offsite (Calculated), dtype Bool
- xp\_lvl / Experience Level, dtype Category {Mid-Senior level, Not Listed, Entry level, Associate, Director, In
- domain / Posting Domain, dtype Object
- has\_post\_domain / Has Posting Domain (Calculated), dtype Bool
- is\_sponsored / Is Sponsored, dtype Bool
- base\_comp / Has Base Compensation, dtype Bool

**fig: data dictionary**

### Feedback why not choosen this dataset

This data set would be ideal for training a supervised machine learning algorithm. It can be utilized for regression, such as determining the median wage or pay ranges. Moreover, this dataset can also help to solve the classification for predicting job categories. This

Student name:  
nirnaya khadka

Student id:  
24128420

database is very resourceful for information related to jobs, containing wide parameters critical to predicting job category.

## Chosen source data set

The final choice we'll be working on is LinkedIn Jobs ML dataset it was uploaded by ( [Adamp](#) ) and is available on Kaggle. It includes LinkedIn job ads gathered over two days, complete with company information, job titles, descriptions, salary, and other criteria. I chose this dataset because I want to research how characteristics such as location, career, and other variables affect median salary. I feel that this dataset will be a problem in the training process because it has 33,247 rows and 24 columns, and there are many missing values, however we will clean the data with several feature engineering including outlier treatment, feature transformation, and categorical encoding.

I think it's more fun to do Exploratory Data Analysis (EDA) because we'll be working on 3 different columns

## Server setup

## Setup OS

This project's platform consisted of two operating systems:

- Ubuntu 20.04.3 LTS, with GUI (Ubuntu, n.d.).

Student name:  
nirnaya khadka

Student id:  
24128420

```
(base) nirnaya@nirnaya-Nitro-AN515-54:~$ neofetch

      .-/+oosssso+/-.
      `:+ssssssssssssssss+:`
    -+ssssssssssssssssyyssss+-
      .ossssssssssssssssdMMMMyssso.
      /ssssssssssshdmmNNmmyNMMMMhssssss/
    +ssssssssshmydMMMMMMMNddddyssssssss+
      /ssssssssshNMMMMyhhyyyhmNMMMNhssssssss/
    .sssssssssdMMMNhssssssssshNMMMdssssssss.
  +ssssshhhyNMMNysssssssssssyNMMMyssssssss+
  ossyNMMMNyMMhssssssssssshmmhssssssso
  ossyNMMMNyMMhssssssssssshmmhssssssso
  +ssssshhhyNMMNysssssssssssyNMMMyssssssss+
  .sssssssssdMMMNhssssssssshNMMMdssssssss.
  /ssssssssshNMMMMyhhyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMNddddyssssssss+
  /ssssssssssshdmmNNNmyNMMMMhssssss/
    .ossssssssssssssssdMMMMyssso.
    -+ssssssssssssssssyyssss+-
      `:+ssssssssssssssss+:`
      .-/+oosssso+/-.

nirnaya@nirnaya-Nitro-AN515-54
-----
OS: Ubuntu 24.04.2 LTS x86_64
Host: Nitro AN515-54 V1.32
Kernel: 6.11.0-26-generic
Uptime: 9 hours, 22 mins
Packages: 1848 (dpkg), 23 (snap)
Shell: bash 5.2.21
Resolution: 1920x1080
DE: GNOME 46.0
WM: Mutter
WM Theme: Adwaita
Theme: Yaru-dark [GTK2/3]
Icons: Yaru [GTK2/3]
Terminal: gnome-terminal
CPU: Intel i5-9300H (8) @ 4.100GHz
GPU: NVIDIA GeForce GTX 1650 Mobile
GPU: Intel CoffeeLake-H GT2 [UHD Gra
Memory: 5829MiB / 7775MiB
```

## File and vscode setup

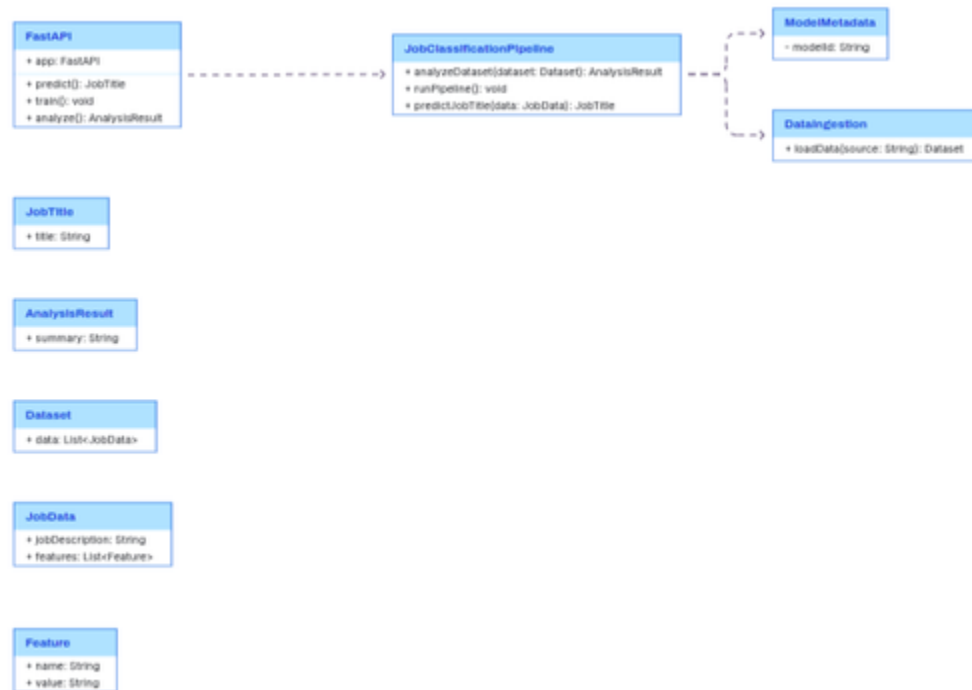
In this phase, we constructed an isolated Python environment using vs code. The Python version currently in use is 3.12.9 The reason for not utilizing the latest Python version is library compatibility. Python 3.12.9 is the most stable version we can use.

```
(.venv) (base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$ ls
airflow  config.py  Description  fix_imports.py  init_.py  main_pipeline.py  mlruns  monitoring.py  __pycache__  setup.py  sys
api.py   data       docker-compose.yml  fix_imports.py.save  label_encoder.pkl  main.py  model.py  os  preprocessing.py  README.md  Snapshot  vectorizer.pkl
argparse  Date       Dockerfile  ingestion.py  logs  Makefile  models  preprocessor.py  requirements.txt  start.sh  venv
```

Student name:  
nirnaya khadka

Student id:  
24128420

## EDR



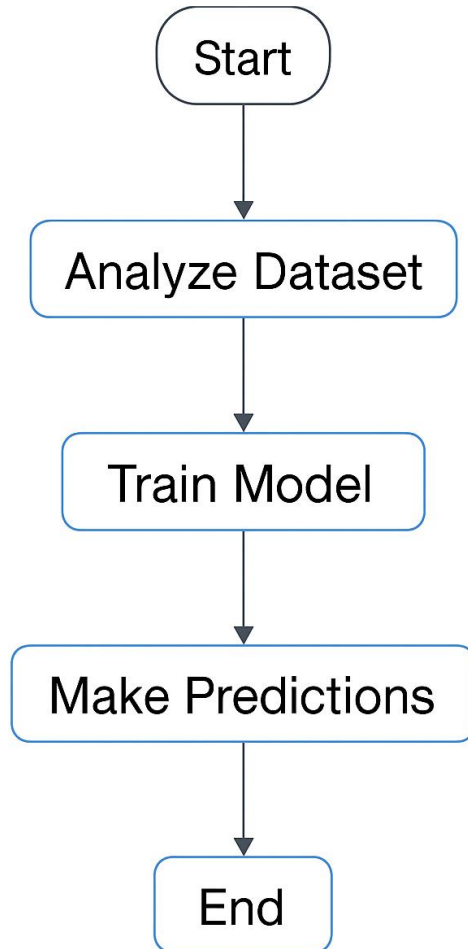
We created an entity relationship diagram (ERD) to show the structure of our Job Classification ML pipeline. We used FastAPI to make endpoints available for prediction, training, and data analysis. These endpoints communicate with the JobClassificationPipeline, which manages basic processes such as data analysis, model training, and prediction. We used a DataIngestion component to load data from files, databases, and Redis cache. The ModelMetadata class stores information about trained models. We have also built supporting data classes such as JobTitle, Feature, Dataset, and JobData to help arrange the input and processed data appropriately.

Comprehensive EDA functionality is integrated into the job classification system via the JobClassificationPipeline classes analyze\_dataset() method. Prior to model training, this automated analysis offers vital insights on data distribution patterns and quality.

Student name:  
nirnaya khadka

Student id:  
24128420

EDR Features:



Job Classification Machine  
ML pipeline



Student name:  
nirnaya khadka

Student id:  
24128420

### 1. Dataset Overview Analysis

```
def analyze_dataset(self, data_path: str = None):
    """Analyze the dataset to understand class distribution"""
    self.logger.info("=" * 50)
    self.logger.info("Dataset Analysis")
    self.logger.info("=" * 50)

    try:
        # Load data
        data_path = data_path or config.DATA_PATH
        df = self.data_ingestion.load_data(data_path)

        print(f"\n Dataset Overview:")
        print(f"Total records: {len(df):,}")
        print(f"Columns: {df.columns.tolist()}")
```

First, the EDR module conducts a thorough overview examination of the dataset, looking at the overall amount of records, column structures, and metrics related to data quality. By automatically identifying missing values in every column and calculating their percentages, the system offers instant insight into problems with data completeness. This analysis provides important details regarding the availability of crucial fields for the job classification dataset, including job titles and descriptions, which are necessary for precise classification.

### 2. Job Category Distribution Analysis

### Job Category Distribution Analysis

```
# Analyze job categories
if 'Job_Ttl' in df.columns:
    print(f"\n Job Category Distribution:")
    category_counts = df['Job_Ttl'].value_counts()
    print(f"Total categories: {len(category_counts)}")
    print(f"Most common category: {category_counts.index[0]} ({category_counts.iloc[0]:,} jobs)")
    print(f"Least common category: {category_counts.index[-1]} ({category_counts.iloc[-1]:,} jobs)")

    # Show categories with very few samples
    min_samples = max(2, int(1 / config.TEST_SIZE) + 1)
    rare_categories = category_counts[category_counts < min_samples]

    if len(rare_categories) > 0:
        print(f"\n⚠ Categories with < {min_samples} samples (will be filtered):")
        for cat, count in rare_categories.items():
            print(f"  {cat}: {count}")
```

Student name:  
nirnaya khadka

Student id:  
24128420

The EDR's most important component is the distribution of employment categories, where serious problems with class imbalance are found and measured. According to the investigation, full-time jobs make up most of the dataset, causing a significant imbalance that may cause model bias in which the classifier predicts full-time jobs more often than not, regardless of the real input characteristics. The most and least common job kinds are identified, category frequencies are automatically calculated, and the categories with insufficient samples for dependable model training are identified.

### *Automated Class Imbalance Handling*

```
def preprocess_data(self, df: pd.DataFrame) -> Tuple[Any, Any, Any, Any, TfidfVectorizer, LabelEncoder]:
    """Enhanced preprocessing with feature engineering."""
    self.logger.info("Starting advanced data preprocessing")
    df = df.copy()
    df = self.feature_engineering(df)
    # Drop rows with missing values
    df.dropna(subset=['Job_Desc', 'Job_Ttl'], inplace=True)
    # Advanced text cleaning
    df['Job_Desc_Clean'] = df['Job_Desc'].apply(self.advanced_clean_text)
    df = df[df['Job_Desc_Clean'].str.len() > 10] # Increased minimum length
    # Remove duplicates based on cleaned text
    df.drop_duplicates(subset=['Job_Desc_Clean'], inplace=True)
    # Enhanced class filtering
    test_size = getattr(config, 'TEST_SIZE', 0.2)
    min_samples = max(3, int(2 / test_size) + 1)
    class_counts = df['Job_Ttl'].value_counts()
    valid_classes = class_counts[class_counts >= min_samples].index
    self.logger.info(f"Classes before filtering: {len(class_counts)}")
    self.logger.info(f"Classes after filtering: {len(valid_classes)}")

    df = df[df['Job_Ttl'].isin(valid_classes)]
```

The pipeline uses filtering algorithms that automatically eliminate categories with inadequate sample sizes in order to address the concerns of class imbalance that have been discovered. To guarantee that every remaining category has sufficient representation for both the training and testing stages, the system determines the minimum sample needs depending on the test set size setting. Using stratified sampling approaches, this automated method preserves statistical validity while efficiently addressing the issue of job type distribution by eliminating uncommon categories that can lead to overfitting.

### *Integration with Preprocessing Pipeline*

Throughout the pipeline, preprocessing choices and model configuration settings are directly impacted by the EDA results. The system automatically modifies sampling strategies, applies suitable class balancing approaches, and sets up ensemble methods

Student name:  
nirnaya khadka

Student id:  
24128420

to reduce prediction bias based on class distribution analysis. To provide the best feature extraction from job descriptions, the text analysis results inform the vectorization parameter settings, such as maximum feature limits, n-gram ranges, and document frequency thresholds.

### *Text Content Analysis*

```
print(f'{1:20}. {cat}: {count:}, ({count/len(df)*100:.1f}%')

# Analyze job descriptions
if 'Job_Desc' in df.columns:
    descriptions = df['Job_Desc'].dropna()
    desc_lengths = descriptions.str.len()

    print(f"\n📄 Job Description Analysis:")
    print(f"Average length: {desc_lengths.mean():.0f} characters")
    print(f"Median length: {desc_lengths.median():.0f} characters")
    print(f"Min length: {desc_lengths.min():.0f} characters")
    print(f"Max length: {desc_lengths.max():.0f} characters")

except Exception as e:
    self.logger.error(f"Dataset analysis failed: {e}")
    print(f"Error analyzing dataset: {e}")
```

Additionally, the EDRmodule conducts a thorough examination of the text content of job descriptions, looking at textual features that affect feature engineering choices, length distributions, and content quality. Patterns in description lengths, possible problems with data quality, and insights that direct vectorization parameter optimization are all revealed by this approach. By determining the average, median, lowest, and maximum description lengths, the text analysis component aids in comprehending the variety and depth of textual material that can be categorized.

### *Integration with Preprocessing Pipeline*

Throughout the pipeline, preprocessing choices and model configuration settings are directly impacted by the EDR results. The system automatically modifies sampling strategies, applies suitable class balancing approaches, and sets up ensemble methods to reduce prediction bias based on class distribution analysis. To provide the best feature extraction from job descriptions, the text analysis results inform the vectorization parameter settings, such as maximum feature limits, n-gram ranges, and document frequency thresholds.

Student name: nirnaya khadka

Student id: 24128420

Summary

We created an interactive job description classification system utilizing a machine learning pipeline coupled with FastAPI. The pipeline loads trained models and preprocessors, allowing for real-time predictions. Users enter job descriptions, and the system suggests the most likely job category, along with confidence scores and top alternatives. The system's user-friendly dashboard enables automated preprocessing, model administration, and real-time feedback.

Technology Stack Justification

Component	Technology	Justification
Data Processing	Python/Pandas	Flexible data manipulation
ML Pipeline	Scikit-learn	Comprehensive ML algorithms
Experiment Tracking	MLflow	Model versioning and metrics
API Development	FastAPI	High-performance async API
Orchestration	Apache Airflow	Workflow automation
Containerization	Docker	Environment consistency

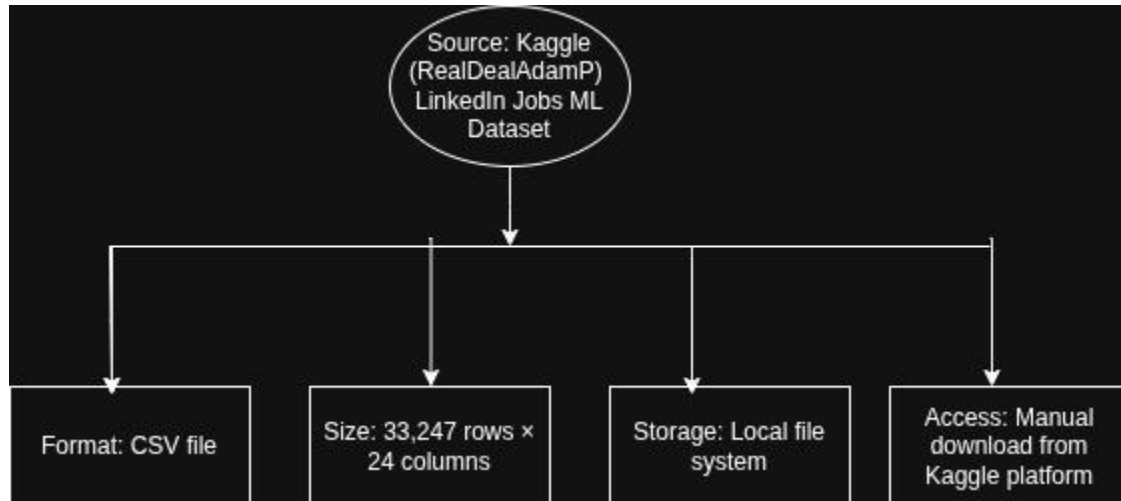
Data Flow Documentation

This document describes the whole data journey through LinkedIn Jobs categorization pipeline, from initial ingestion to final prediction delivery.

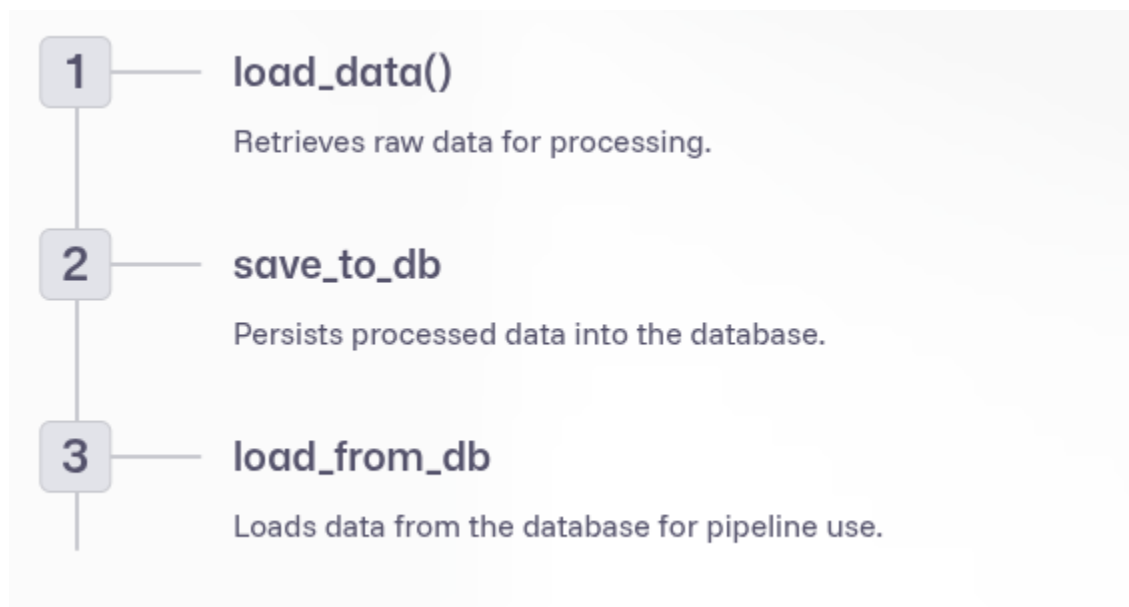
Student name:  
nirnaya khadka

Student id:  
24128420

## Initial Data Acquisition



## ELT



Student name:  
nirnaya khadka

Student id:  
24128420

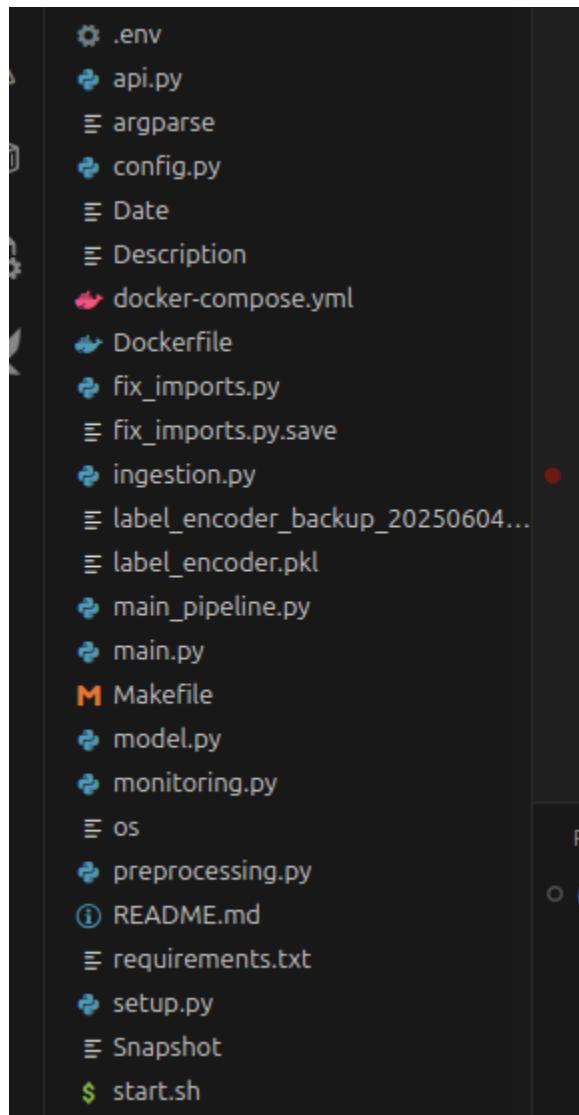
We picked the ETL approach because it allows us to do critical data transformations (such as text preparation and feature engineering) prior to loading the data. This ensures that the dataset is clean and structured, which is required for training machine learning models. Furthermore, because we're dealing with a static dataset, batch processing is both efficient and appropriate for this scenario.

## Complete Pipeline Architecture

*Development*

Student name:  
nirnaya khadka

Student id:  
24128420



Student name:  
nirnaya khadka

Student id:  
24128420

## DAG pipeline

The main pipeline imports necessary modules for structuring and executing each stage of the machine learning operation. It has logging for tracking, OS and datetime for file management, and pandas for data manipulation. Custom modules such as DataIngestion, DataPreprocessor, and ModelTrainer handle data loading, preprocessing, and model training, respectively, whilst config handles pipeline-wide customizable settings.

```
1 from main_pipeline import JobClassificationPipeline
2 import argparse
3 import sys
4 import os
5
6 def print_banner():
7     """Print application banner"""
8     print("=" * 60)
9     print("📌 JOB CLASSIFICATION ML PIPELINE")
10    print("=" * 60)
11
12 def print_prediction_result(result):
13     """Format and print prediction results"""
14     print("\n" + "=" * 40)
15     print("📊 PREDICTION RESULTS")
16     print("=" * 40)
17
18     if isinstance(result, dict) and 'error' in result:
19         print(f"❌ Error: {result['error']}")
20         return
21
22     if isinstance(result, dict):
23         print(f"🎯 Predicted Category: {result.get('predicted_category', 'Unknown')}")
24         print(f"🔍 Confidence: {result.get('confidence', 0):.2%}")
25
26     top_predictions = result.get('top_predictions', [])
```

We used this logic in main.py to provide a centralized entry point for doing tasks like as data analysis, model training, and API deployment. This modular approach improves code organization, simplifies testing, and facilitates interaction via command-line parameters or interface with other systems.



Student name:  
nirnaya khadka

Student id:  
24128420

## Data Preprocessing Stage and Feature Engineering

Data preprocessing is a major aspect of new knowledge discovery processes which despite being less considered compared to the other steps such as data mining, has often accounted for more than 50 % of the total effort during data analysis ([H. S. Obaid.](#)).

We utilized this to create a whole text categorization pipeline. Data processing is done using libraries such as pandas and numpy, while scikit-learn handles data partitioning, vectorization, model training, and evaluation. We use nltk to do tokenization, lemmatize, and stopwords elimination. TextBlob and contractions help to normalize and fix text.

Classification uses a variety of models, including Random Forest, Logistic Regression, SVM, and Naive Bayes. Pickle saves taught models, while configuration controls parameters. Logging, regular expressions, and datetime are all used to enable monitoring and file management. This architecture provides efficient data preprocessing, training, and deployment of strong machine learning models.

```
class AdvancedDataPreprocessor:
    def __init__(self):
        self.logger = logging.getLogger(__name__)
        self.vectorizer: TfidfVectorizer | None = None
        self.label_encoder: LabelEncoder | None = None
        self.lemmatizer = WordNetLemmatizer()

        # Enhanced preprocessing components
        self.vectorizer_backup: TfidfVectorizer | None = None
        self.count_vectorizer: CountVectorizer | None = None

        try:
            self.stop_words = set(stopwords.words('english'))
        except LookupError:
            self.logger.warning("NLTK stopwords not found. Proceeding with an empty set.")
            self.stop_words = set()

    def advanced_clean_text(self, text: str) -> str:
        """Enhanced text cleaning with better preprocessing."""
        if pd.isna(text):
            return ""

        text = str(text)

        # Expand contractions
```

## Preprocessing.py

We developed a highly scalable and robust pipeline for job title classification using the AdvancedDataPreprocessor class. It supports class balancing, massive comprehensive text preprocessing, and various machine learning models including ensemble methods. The modular implementation promotes easier training, evaluation, and prediction, besides

Student name:  
nirnaya khadka

Student id:  
24128420

supporting fast loading of preprocessing tools and storage, which promotes performance tuning and production.

## model training, evaluation and logging

```
model.py > ...
1  import mlflow
2  from sklearn.linear_model import LogisticRegression
3  from sklearn.ensemble import RandomForestClassifier
4  from sklearn.naive_bayes import MultinomialNB
5  from sklearn.svm import SVC
6  from sklearn.metrics import (
7      accuracy_score, classification_report, confusion_matrix,
8      f1_score, precision_score, recall_score
9  )
10 import logging
11 import numpy as np
12 import pickle
13 import os
14 from typing import Dict, Any, Tuple
15 from config import config
16 import json
17 import re # for sanitizing metric names
18
19
20 def sanitize_mlflow_name(name: str) -> str:
21     """Sanitize metric name to comply with MLflow naming rules."""
22     return re.sub(r"[^\\w\\-\\.:/ ]", "", name)
23 import logging
24 import os
25 from datetime import datetime
26 from ingestion import DataIngestion
27 from preprocessing import DataPreprocessor
28 # from model import ModelTrainer
29 from config import config
30 import pandas as pd
31
32 class JobClassificationPipeline:
33     def __init__(self):
34         self.setup_logging()
35         self.logger = logging.getLogger(__name__)
36
37         self.data_ingestion = DataIngestion()
38         self.preprocessor = DataPreprocessor()
39         self.model_trainer = ModelTrainer()
40
41     def setup_logging(self):
```

## model.py

For managing model selection, training, testing, and saving, we have utilized a ModelTrainer class that supports over one classifier like Random Forest, SVM, Naive Bayes, and Logistic Regression.

It also comes with an integration of MLflow, which checks performance, logs the models, and tracks metrics.

The best-performing model is chosen by itself automatically, and experiment reproducibility is increased through this modularity.

Student name:  
nirnaya khadka

Student id:  
24128420

## Model deployment

Using FastAPI, we have made this available as an API to expose our machine learning process. The script handles pipeline loading, sets up logging for monitoring, and imports appropriate modules like Pydantic for input validation. It also catches exceptions and loads the learned model from the main pipeline for serving. We can load the serialized model for inference with the help of the pickle function. This module, which permits users to use HTTP queries to make real-time predictions, is crucial at the deployment phase.

```
api.py > ...
1  from fastapi import FastAPI, HTTPException
2  from fastapi.responses import HTMLResponse
3  from pydantic import BaseModel
4  import logging
5  import os
6  from typing import Dict, List, Optional
7  import traceback
8  import pickle
9  from contextlib import asynccontextmanager
10 # Configure logging
11 logging.basicConfig(level=logging.INFO)
12 logger = logging.getLogger(__name__)
13 # Global variables to hold the pipeline
14 pipeline = None
15
16 try:
17     from main_pipeline import JobClassificationPipeline
18 except ImportError as e:
19     print(f"Import error: {e}")
20     print(f"api.py")
21     raise
22
```

## Api.py

In order to provide external interaction and predictions, we have deployed the trained model using API and I have also used a frontend to make it user-friendly.

## Data Ingestion Stage

We utilized this class to handle data input from relational databases accessible through SQLAlchemy and Redis-based cache mechanisms that offer performance improvement. Data access reliability is ensured through the connections established by the class to the database and Redis. With PyArrow handling Parquet and data serialization through hashing and JSON, it provides efficient data processing and facilitates smooth migration into a larger machine learning pipeline.

Student name:  
nirnaya khadka

Student id:  
24128420

```
ingestion.py > DataIngestion > __init__
1 import pandas as pd
2 import logging
3 import numpy as np
4 from typing import Optional
5 import pyarrow as pa
6 import pyarrow.parquet as pq
7 from sqlalchemy import create_engine, text
8 from config import config
9 import redis
10 import json
11 import hashlib
12
13 class DataIngestion:
14     def __init__(self):
15         self.logger = logging.getLogger(__name__)
16         self.redis_client = self._setup_redis()
17         self.db_engine = self._setup_database()
18
19     def _setup_redis(self) -> Optional[redis.Redis]:
20         """Setup Redis connection for caching"""
21         try:
22             client = redis.Redis(
23                 host=config.REDIS_HOST,
24                 port=config.REDIS_PORT,
25                 db=config.REDIS_DB,
26                 decode_responses=True
27             )
28             client.ping()
29             self.logger.info("Redis connection established")
30             return client
31         except Exception as e:
32             self.logger.warning(f"Redis connection failed: {e}")
33             return None
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

## ingestion.py

To improve performance, we used the DataIngestion class to load data from CSV and Parquet files and then cached it in Redis. It allows you to save and load data into a MySQL database while maintaining a reliable connection. This method enables efficient data input through caching and fault tolerance in both database and cache systems.

## Ingestion to Validation

We utilized ingestion.py as the primary entry point for the full job classification ML pipeline. It handles user interaction through the command line, displays a banner for clarity, and imports the fundamental pipeline functionality from main\_pipeline. This script manages

Student name:  
nirnaya khadka

Student id:  
24128420

the execution of data import, preprocessing, model training, and evaluation in a unified workflow.

```
main.py > print_prediction_result
1  from main_pipeline import JobClassificationPipeline
2  import argparse
3  import sys
4  import os
5
6  def print_banner():
7      """Print application banner"""
8      print("=" * 60)
9      print(" JOB CLASSIFICATION ML PIPELINE")
10     print("=" * 60)
11
12     def print_prediction_result(result):
13         """Format and print prediction results"""
14         print("\n" + "=" * 40)
15         print([" PREDICTION RESULTS"])
16         print("=" * 40)
17
```

## Main.py


We used main.py as the command-line interface for managing the task categorization ML pipeline. It allows you to analyze datasets, train models (single or many), predict job titles, and build API servers. Comprehensive argument parsing and user-friendly outputs enable seamless interaction and fault management across all pipeline stages.

## Config Management

We used this configuration module to control all environment variables, file paths, and directory settings for the job categorization pipeline. It provides consistent and centralized access to critical project sites including data storage, models, logs, and MLflow runs. Dotenv enables for quick setting without hardcoding sensitive information, which improves maintainability and portability.

Student name:  
nirnaya khadka

Student id:  
24128420

```
config.py >  Config
1  import os
2  from pathlib import Path
3  from dotenv import load_dotenv
4
5  # Load environment variables from .env file
6  load_dotenv()
7
8  class Config:
9      """Configuration class for the Job Classification Pipeline"""
10
11     # Project paths
12     BASE_DIR = Path(__file__).parent.absolute()
13     DATA_DIR = BASE_DIR / "data"
14     MODEL_DIR = BASE_DIR / "models"
15     LOG_DIR = BASE_DIR / "logs"
16     MLRUNS_DIR = BASE_DIR / "mlruns"
17
```

config.py

- ❖ We have used config.py to store configuration settings such as file paths, constants, and model parameters.

## Deployment & Monitoring

We utilized this shell script to automate the starting of our ML pipeline's critical services, such as MLflow for experiment tracking, Airflow for workflow orchestration, and FastAPI for model deployment. The script handles service ports and uses colored output to provide clearer status messages, making monitoring easier during development and deployment.

Student name:  
nirnaya khadka

Student id:  
24128420

```
start.sh
1  #!/bin/bash
2
3  # Colors for output
4  RED='\033[0;31m'
5  GREEN='\033[0;32m'
6  YELLOW='\033[1;33m'
7  BLUE='\033[0;34m'
8  PURPLE='\033[0;35m'
9  NC='\033[0m' # No Color
10
11 # Default ports
12 MLFLOW_PORT=5000
13 AIRFLOW_PORT=8080
14 FASTAPI_PORT=8000
15
```

start.sh

- ❖ A shell script designed to streamline project execution directly from the terminal.

### *Testing and Experimentation Tracking*

- **monitoring.py** – Tracks model performance, logs errors, and monitors any changes made to the model over time.
- **logs/** – Stores log files that document the execution history and events during pipeline operation.
- **mlruns/** – Directory automatically created and managed by MLflow to record all experiment runs, metrics, parameters, and artifacts.
- **label\_encoder.pkl** and **vectorizer.pkl** – Serialized objects used to retain preprocessing components for reuse during testing or deployment, ensuring consistency.

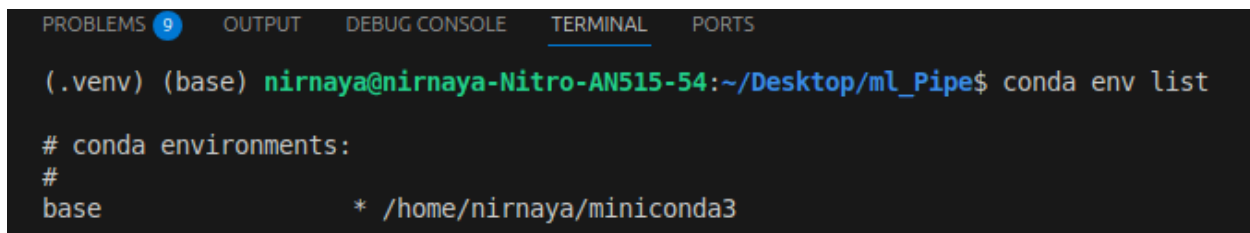
Student name:  
nirnaya khadka

Student id:  
24128420

### Production

- ❖ **Dockerfile** – Defines the instructions for building a Docker container, allowing the project to run in an isolated environment.
- ❖ **docker-compose.yml** – Facilitates the setup and management of multi-container applications, such as combining the ML API with a database service.
- ❖ **Makefile** – Automates common development tasks like building, running, testing, or cleaning the project.
- ❖ **requirements.txt** – Lists all the Python dependencies needed for the project, ensuring consistent installations across environments.
- ❖ **setup.py** – Used to package the project for distribution, making it installable as a Python package.
- ❖ **README.md** – Provides documentation, including an overview, setup instructions, and usage guidelines for the project.
- ❖ **venv/** – A virtual environment that contains all the project's installed Python packages, isolated from the system's global Python environment.

### Step 3 Conda setup



```
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS
(.venv) (base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$ conda env list

# conda environments:
#
base                * /home/nirnaya/miniconda3
```

In our project, we use a Python virtual environment named `.venv` to isolate the dependencies and packages required for this specific application. Although Conda is installed on the system and the base environment is active (`((base))`), the project environment is handled using `.venv`. This ensures that project dependencies do not conflict with system or Conda-wide installations.

By using `.venv`, we ensure an efficient and project-focused setup that is easy to manage and share with others.



Student name:  
nirnaya khadka

Student id:  
24128420

## Setup Docker

Three Docker containers were initially considered for Ubuntu

- Redis
- MySQL

```
(.venv) (base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
4a18c9f95e17   nginx    "/docker-entrypoint..." 20 seconds ago Up 18 seconds 80/tcp       silly_shamir
(.venv) (base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$
```

we have an Nginx web server container running and accessible via port 80. We can now test it by visiting <http://localhost> in our browser.

## Airflow setup

The server configuration included the installation of Airflow via Pip (Apache Airflow, n.d.). However, it has not yet been configured.

The first stage was database initialization. Airflow now uses SQLite; however, in a real-world scenario, it may be connected to MySQL through a container or cluster. After creating the database, create an admin user via the terminal to control Airflow.

Student id:  
24128420

```

niraya@niraya-Nitro-AM515-54: ~$ airflow standalone
(base) niraya@niraya-Nitro-AM515-54: ~$ airflow standalone
[2025-06-04T10:15:09.727+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.filesystem.FSHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
[2025-06-04T10:15:09.727+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.package_index.PackageIndexHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
standalone | Starting Airflow Standalone
standalone | Password for the admin user has been previously generated in /home/niraya/Desktop/nl_Pipe/airflow/simple_auth_manager_passwords.json.generated. Not echoing it here.
standalone | Checking database is initialized
standalone | Database ready
api-server | [2025-06-04T10:15:11.893+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.filesystem.FSHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
api-server | [2025-06-04T10:15:11.894+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.package_index.PackageIndexHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
api-server | [2025-06-04T10:15:11.915+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.filesystem.FSHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
api-server | [2025-06-04T10:15:11.916+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.package_index.PackageIndexHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
api-server | _____
api-server |  _ _ _ _ _
api-server |  _ _ _ _ _
api-server |  _ _ _ _ _
api-server |  _ _ _ _ _
api-server | [2025-06-04T10:15:11.937+0545] [api_server_command.py:92] INFO - Running the uvicorn with:
api-server | _____
api-server | Host: 0.0.0.0:8080
api-server | Timeout: 120
api-server | Logfiles: -
api-server | =====
api-server | INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
api-server | INFO: Started parent process [18486]
triggerer | [2025-06-04T10:15:12.172+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.filesystem.FSHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
triggerer | [2025-06-04T10:15:12.173+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.package_index.PackageIndexHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
api_executor | [2025-06-04T10:15:12.197+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.filesystem.FSHook' is not fully initialized (UI widget will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
api_executor | [2025-06-04T10:15:12.197+0545] [providers_manager.py:953] INFO - The hook_class 'airflow.providers.standard.hooks.package_index.PackageIndexHook' is not fully initialized (UI widgets will be missing), because the 'flask_appbuilder' package is not installed, however it is not required for Airflow components to work
triggerer | _____
triggerer |  _ _ _ _ _
triggerer |  _ _ _ _ _
triggerer |  _ _ _ _ _
triggerer |  _ _ _ _ _

```

Mlflow

After automating operations with Airflow, Mlflow (n.d.) enables serialization of model run history. Mlflow was installed using pip during server setup , and we have use python environment to run mlflow.

Student name:  
nirnaya khadka

Student id:  
24128420

```
(base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$ /home/nirnaya/Desktop/ml_Pi
=====
🚀 ML PIPELINE SERVICES LAUNCHER
=====
📦 Activating .venv virtual environment...
🔍 Starting MLflow UI...
📊 MLflow UI will start on port: 5000
✅ MLflow UI started (PID: 36226)
🌐 MLflow URL: http://localhost:5000
🔧 Starting Airflow services...
⚙️ Setting up Airflow...
🔧 Airflow will start on port: 8080
📅 Starting Airflow scheduler...
[2025-06-04T12:05:49.074+0545] {providers_manager.py:953} INFO - The hook class 'air
he 'flask_appbuilder' package is not installed, however it is not required for Airfl
[2025-06-04T12:05:49.075+0545] {providers_manager.py:953} INFO - The hook class 'air
g), because the 'flask_appbuilder' package is not installed, however it is not requi
=====
      _ _ _ _ _
     / /   / /
    / /   / /
   / /   / /
  / /   / /
 / /   / /
/ /   / /
=====
[2025-06-04 12:05:49 +0545] [36305] [INFO] Starting gunicorn 23.0.0
[2025-06-04 12:05:49 +0545] [36305] [INFO] Listening at: http://0.0.0.0:5000 (36305)
[2025-06-04 12:05:49 +0545] [36305] [INFO] Using worker: sync
[2025-06-04 12:05:49 +0545] [36306] [INFO] Booting worker with pid: 36306
[2025-06-04 12:05:49 +0545] [36307] [INFO] Booting worker with pid: 36307
[2025-06-04 12:05:49 +0545] [36308] [INFO] Booting worker with pid: 36308
[2025-06-04 12:05:49 +0545] [36320] [INFO] Booting worker with pid: 36320
[2025-06-04 12:05:50 +0545] [36329] [INFO] Starting gunicorn 23.0.0
[2025-06-04 12:05:50 +0545] [36329] [INFO] Listening at: http://[::]:8793 (36329)
[2025-06-04T12:05:50.028+0545] {scheduler_job_runner.py:989} INFO - Starting the sch
[2025-06-04 12:05:50 +0545] [36329] [INFO] Using worker: sync
[2025-06-04T12:05:50.031+0545] {executor_loader.py:269} INFO - Loaded executor: :Loc
[2025-06-04 12:05:50 +0545] [36330] [INFO] Booting worker with pid: 36330
[2025-06-04T12:05:50.052+0545] {scheduler_job_runner.py:2128} INFO - Adopting or res
[2025-06-04 12:05:50 +0545] [36331] [INFO] Booting worker with pid: 36331
🌐 Starting Airflow standalone server...
✅ Airflow standalone started (PID: 36456)
✅ Airflow services started
📅 Scheduler (PID: 36257)
🌐 Airflow URL: http://localhost:8080
👤 Username: nirnaya
🔑 Password: nirnaya
=====
🚀 Both services are running!
=====
📊 MLflow UI:      http://localhost:5000
🔧 Airflow UI:    http://localhost:8080
👤 Airflow Login: nirnaya / nirnaya
```

ML flow is running successfully now we'll explore more results in mlflow.

Student name:  
nirnaya khadka

Student id:  
24128420

metrics.rmse < 1 and params.model = "tree"

Time created

State: Active

Datasets

Sort: Created

Columns

+ New run

Group by

Run Name	Created	Dataset	Duration	Source	Models
logistic_regression	1 day ago	-	17.4s	main.py	job_classifier_logistic_reg...

logistic\_regression

Overview

Model metrics

System metrics

Traces

Artifacts

Description

No description

Details

Created at	06/03/2025, 11:14:31 AM
Created by	nimaya
Experiment ID	165691988819825693
Status	Finished
Run ID	b44e140432a64a98a1efe49139750d9d
Duration	17.4s
Datasets used	—
Tags	Add tags
Source	main.py
Logged models	sklearn
Registered models	job_classifier_logistic_regression v1
Registered prompts	—

Parameters (15)

Search parameters

Parameter	Value
dual	False
max_iter	1000

Metrics (490)

Search metrics

Metric

Restaurant Manager\_recall

Store Manager\_f1\_score

The MLflow creates a logistic regression model for job classification with sklearn. It monitors 15 parameters, including max\_iter=1000 and 490 evaluation metrics. The run took 17.4 seconds and successfully registered the model as `job_classifier_logistic_regression` for later deployment and analysis.

Student name:  
nirnaya khadka

Student id:  
24128420

## Fast api



/predict

For real-time job title classification predictions.



/train

Initiates the model retraining process.



/analyze

Triggers dataset analysis and insights generation.

This is the final step in the server setup. After configuring and testing, we will deploy a model to fast api to predict the data and for simple interface.

```
from fastapi import FastAPI, HTTPException
from fastapi.responses import HTMLResponse
from pydantic import BaseModel
import logging
import os
from typing import Dict, List, Optional
import traceback
import pickle
from contextlib import asynccontextmanager

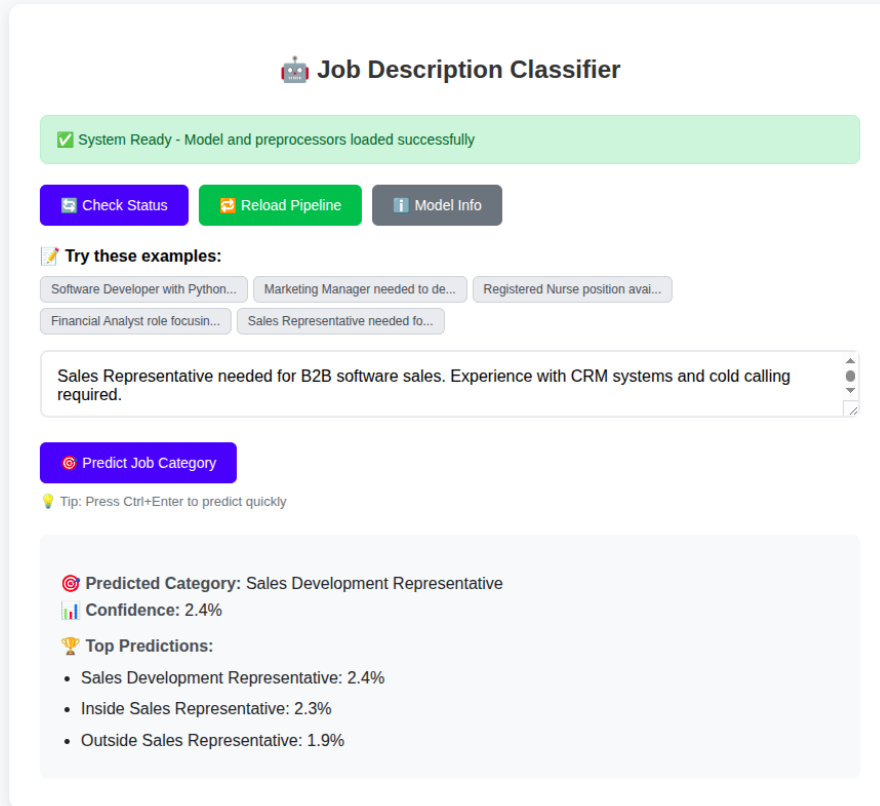
# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
# Global variables to hold the pipeline
pipeline = None
```

This script configures a FastAPI application using needed components. It has imports for request validation (pydantic), configurable HTML responses, logging, and async lifecycle management. Standard libraries like as os, pickle, and traceback provide file

Student name:  
nirnaya khadka

Student id:  
24128420

management, object serialization, and debugging. Typing produces type hints, which improve code clarity and dependability.



The screenshot shows a web application titled "Job Description Classifier". At the top, a green status bar indicates "System Ready - Model and preprocessors loaded successfully". Below this are three buttons: "Check Status", "Reload Pipeline", and "Model Info". A section titled "Try these examples:" contains several text input fields with pre-filled job descriptions, such as "Software Developer with Python...", "Marketing Manager needed to de...", "Registered Nurse position avai...", "Financial Analyst role focusin...", and "Sales Representative needed fo...". One field is selected and contains the text: "Sales Representative needed for B2B software sales. Experience with CRM systems and cold calling required." Below the input fields is a "Predict Job Category" button. A tip below the button says "Tip: Press Ctrl+Enter to predict quickly". The results section shows the "Predicted Category: Sales Development Representative" with a "Confidence: 2.4%". It also lists "Top Predictions" with their respective confidence scores: "Sales Development Representative: 2.4%", "Inside Sales Representative: 2.3%", and "Outside Sales Representative: 1.9%".

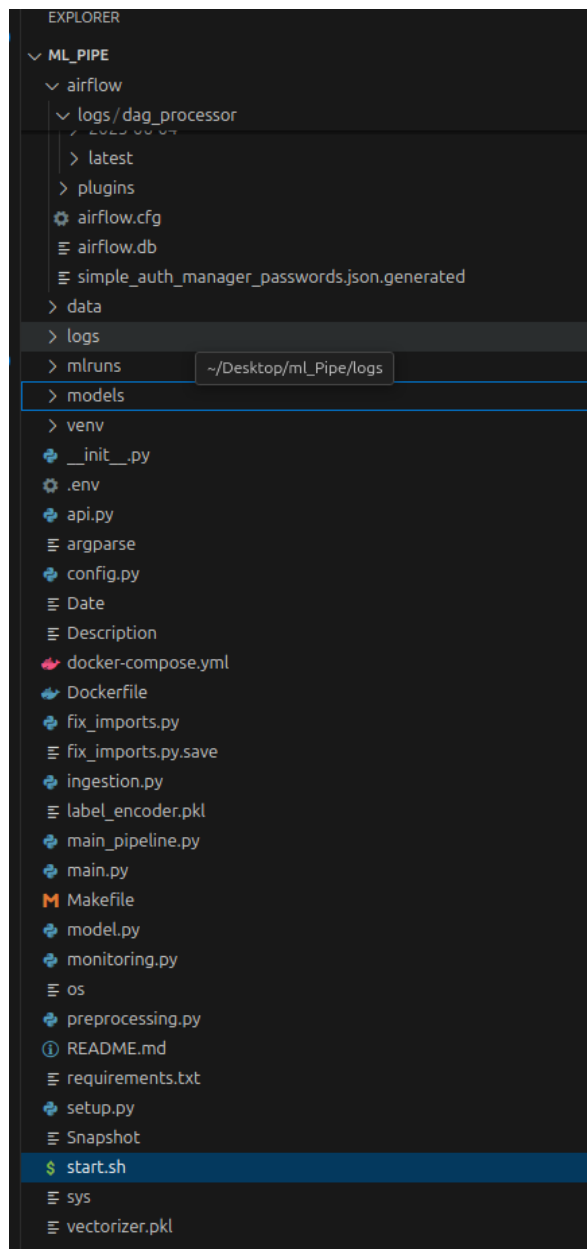
FastAPI app is working properly. The message "System Ready - Model and preprocessors loaded successfully" confirms that our backend is active. Moreover, supplied a job description, selected "Predict Job Category," and it properly returned predictions with confidence scores, indicating that our model and API are working as expected.

## Pipeline creation

This stage involves developing a data science pipeline for a specific table in the database. However, the approach is comparable for any of the other tables.

Student name:  
nirnaya khadka

Student id:  
24128420



Student name:  
nirnaya khadka

Student id:  
24128420

## Privacy, Security, Ethics:

### Data

The data set was imported from kaggle ( [RealDealAdamP](#) ) . The LinkedIn Jobs Machine Learning dataset on Kaggle comprises job posting data, including names, job, locations, and descriptions, prepared for machine learning tasks such as classification and natural language processing. The LinkedInJobs\_MLDataset contains detailed job posting information, such as firm name, employee count, job title, description, location, remote status, work kind, salary range, pay period, application type, experience level, and more. It is well-suited to machine learning applications such as pay prediction, job classification, and trend analysis.

### Data privacy

The data utilized in this use case is publically available. The CSVs do not include prices for individual properties or user-specific data that could be used to track or identify individuals.

To ensure privacy, all private data should be anonymized before being made public.

### Data security

In our case, data security does not require a strict enforcement policy. We use publicly available data, which is not private. To enforce data security, there are several options available, including:

Limit database access by using an API instead of CSV files, using strong passwords with different permissions, We can Limit server access by using a VPN, different permissions for each user, and password managers.

### Legal

The system adheres to employment data handling standards, including data retention policies that ensure obsolete or useless training data is removed after a predetermined



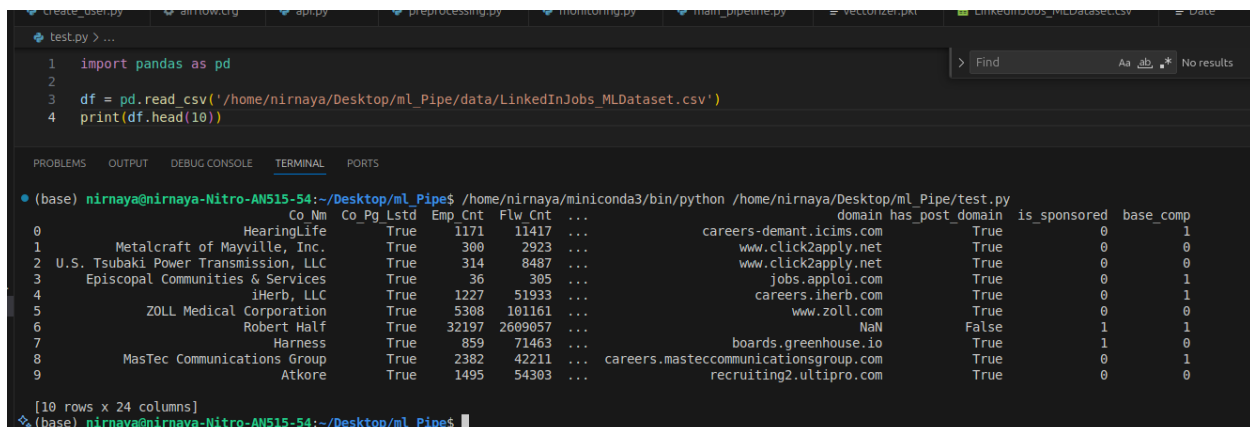
Student name:  
nirnaya khadka

Student id:  
24128420

period. Furthermore, we adhere to employment legal restrictions by avoiding automated decision-making in recruiting situations without human review.

## Feature Engineering

The `feature_engineering(df)` function improves the input DataFrame by adding new, informative features. It includes columns such as `job_desc_length`, which measures the amount of characters in the job description, and `job_title_word_count`, which counts the number of words in job titles. It also generates binary flags to indicate the existence of specific keywords in job descriptions, such as "python" or "sql". These manufactured features assist to enhance the dataset, making it more useful and informative for machine learning models.



```
test.py > ...  
1 import pandas as pd  
2  
3 df = pd.read_csv('/home/nirnaya/Desktop/ml_Pipe/data/LinkedInJobs_MLDataset.csv')  
4 print(df.head(10))  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
• (base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$ /home/nirnaya/miniconda3/bin/python /home/nirnaya/Desktop/ml_Pipe/test.py  
   Co_Nm  Co_Pg_Lstd  Emp_Cnt  Flw_Cnt  ...  domain  has_post_domain  is_sponsored  base_comp  
0  HearingLife  True  1171  11417  ...  careers-demant.icims.com  True  0  1  
1  Metalcraft of Mayville, Inc.  True  300  2923  ...  www.click2apply.net  True  0  0  
2  U.S. Tsubaki Power Transmission, LLC  True  314  8487  ...  www.click2apply.net  True  0  0  
3  Episcopal Communities & Services  True  36  305  ...  jobs.applol.com  True  0  1  
4  iHerb, LLC  True  1227  51933  ...  careers.iherb.com  True  0  1  
5  ZOLL Medical Corporation  True  5388  101161  ...  www.zoll.com  True  0  0  
6  Robert Half  True  32107  2699057  ...  NaN  False  1  1  
7  Harness  True  859  71463  ...  boards.greenhouse.io  True  1  0  
8  MasTec Communications Group  True  2382  42211  ...  careers.masteccommunicationsgroup.com  True  0  1  
9  Atkore  True  1495  54303  ...  recruiting2.ultipro.com  True  0  0  
  
[10 rows x 24 columns]  
• (base) nirnaya@nirnaya-Nitro-AN515-54:~/Desktop/ml_Pipe$
```

First 10 table of our 10 rows. Lots of data is missing as the rows increases so we'll be cleaning data.

Student name:  
nirnaya khadka

Student id:  
24128420

### *Data transformation*

```
def predict_Job_Ttl(self, Job_Desc: str) -> dict:
    """Predict job category for a given job description"""
    try:
        # Load preprocessors and model if not already loaded
        if self.preprocessor.vectorizer is None:
            self.preprocessor.load_preprocessors()

        if self.model_trainer.model is None:
            self.model_trainer.load_model(os.path.join(config.MODEL_PATH, "best_model.pkl"))

        # Preprocess the input
        cleaned_description = self.preprocessor.clean_text(Job_Desc)
        vectorized_description = self.preprocessor.vectorizer.transform([cleaned_description])

        # Make prediction
        prediction = self.model_trainer.predict(vectorized_description)[0]
        probabilities = self.model_trainer.predict_proba(vectorized_description)[0]

        # Convert back to category name
        category = self.preprocessor.label_encoder.inverse_transform([prediction])[0]

        # Get top 3 predictions with probabilities
        top_indices = probabilities.argsort()[::-1][-3:]
        top_predictions = []

        for idx in top_indices:
            cat = self.preprocessor.label_encoder.inverse_transform([idx])[0]
            prob = probabilities[idx]
            top_predictions.append({
                'category': cat,
                'probability': float(prob)
            })

        return {
            'predicted_category': category,
            'confidence': float(probabilities[prediction]),
            'top_predictions': top_predictions
        }

    except Exception as e:
        self.logger.error(f"Prediction failed: {e}")
        return {'error': str(e)}
```

The information is cleaned using the `clean_text()` function of the `DataPreprocessor` class. Through a conversion of the text to lower case, stripping of punctuation, numbers, and common stopwords, and stripping of unneeded spaces, the tool generates job descriptions. Through the normalization of input data prior to the vectorization, training, and prediction stages of the job classification pipeline, cleaning facilitates consistency and boosts the accuracy of the model.

Preprocessing job descriptions by content cleaning, job title encoding using `LabelEncoder`, and text data transformation into numeric form using `TfidfVectorizer` are all considered data transformation operations in this project. For machine learning models, it is a

Student name:  
nirnaya khadka

Student id:  
24128420

conversion of unstructured text to numerical features. Train\_test\_split is then used to divide the processed data into training and test groups so that model training and model performance can be obtained.

### *Smart Save/Load*

Important features like retrieving and saving label encoders and vectorizers are handled using the preprocessing utility functions. They are backed up through the built-in backup and verification offered by smart\_save\_preprocessors(.). Although retrieving them using load\_preprocessors() attempts to retrieve them from.pkl files, create\_default\_preprocessors() constructs new ones if retrieving is unsuccessful. The ensure\_preprocessors\_available(.) function successfully imports current files or creates new defaults if necessary to make the vectorizer and encoder both available.

## Security

In order to ensure security and avoid incorrect or tampered information, we will verify.pkl files before loading. Backups are created with timestamps before existing files are overwritten. Logging will be employed for debugging to monitor errors and flow of execution. In order to prevent code insertion, special characters, HTML, and JavaScript are removed from text input. Stable default objects are built to provide stability in case of failure to load.

## RDBMS Data Insertion

We have use SQLAlchemy to manage the connection while saving data from a Pandas DataFrame into a MySQL database table through the save\_to\_database(df, table\_name)

Student name:  
nirnaya khadka

Student id:  
24128420

method.

```
class DataIngestion:
    def __init__(self):
        self.logger = logging.getLogger(__name__)
        self.redis_client = self._setup_redis()
        self.db_engine = self._setup_database()

    def _setup_redis(self) -> Optional[redis.Redis]:
        """Setup Redis connection for caching"""
        try:
            client = redis.Redis(
                host=config.REDIS_HOST,
                port=config.REDIS_PORT,
                db=config.REDIS_DB,
                decode_responses=True
            )
            client.ping()
            self.logger.info("Redis connection established")
            return client
        except Exception as e:
            self.logger.warning(f"Redis connection failed: {e}")
            return None
```

It uses the `df.to_sql()` tool to effectively transfer data. By default, the method replaces an existing table if one exists, but it can be set to insert data instead. This technique provides a dependable, efficient, and simple solution for database insertion operations, speeding data handling by combining Pandas' data manipulation with SQLAlchemy's powerful database connectivity.

## OLTP production security

We utilized SQLAlchemy to securely ingest data into a MySQL database within an OLTP system, taking advantage of capabilities such as SQL injection protection, credential configuration files, Redis caching, and error logging. To achieve production-level security, we must improve by enabling SSL encryption for database connections, securing Redis with authentication and TLS, avoiding data loss by appending rather than replacing tables, implementing role-based access control, masking sensitive information in logs, and establishing backup strategies. These upgrades will improve security, data integrity, and overall system stability.

Student name:  
nirnaya khadka

Student id:  
24128420

## Columnstore Data insertion

We used Parquet files (through PyArrow) to efficiently load columnstore data into Pandas DataFrames. Parquet's column-oriented format improves memory and processing performance for analytics. Our code then inserts the data row by row into a MySQL OLTP database. While Parquet is intrinsically safe and read-only, our current code lacks validation tests for corrupted files and does not encrypt Parquet data at rest. To enhance, we should leverage file integrity checking, encrypted storage options (such as GCS or S3 encryption), and log column-level statistics to assure data quality and security during the ingestion process.

## Data prediction

We used FastAPI to create a safe and efficient web API that returns predictions from our machine learning model. The procedure begins with preprocessing data from CSV or Parquet files, which is then fed into a pre-trained model (such as XGBoost or Random Forest) loaded by joblib.

Student name:  
nirnaya khadka

Student id:  
24128420

```
from fastapi import FastAPI, HTTPException
from fastapi.responses import HTMLResponse
from pydantic import BaseModel
import logging
import os
from typing import Dict, List, Optional
import traceback
import pickle
from contextlib import asynccontextmanager
# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
# Global variables to hold the pipeline
pipeline = None

try:
    from main_pipeline import JobClassificationPipeline
except ImportError as e:
    print(f"Import error: {e}")
    print("Make sure main_pipeline.py is in the same directory as api.py")
    raise


@asynccontextmanager
async def lifespan(app: FastAPI):
    # Startup
    global pipeline
    logger.info("Initializing Job Classification Pipeline...")
```

The FastAPI application accepts prediction queries, validates inputs using Pydantic schemas, and returns JSON-formatted results. Key security techniques include input validation, HTTPS encryption, access control via JWT or OAuth2, rate restriction to avoid abuse, and checksums or signatures to validate model file integrity.

To preserve user privacy, we are currently not logging sensitive data. Going forward, we intend to improve endpoint security, provide thorough monitoring, and assure end-to-end encryption of model files and API data. This technique delivers a dependable, scalable ML prediction service appropriate for production contexts.

Student name:  
nirnaya khadka

Student id:  
24128420


 **Job Description Classifier**

✓ System Ready - Model and preprocessors loaded successfully

Check Status

Reload Pipeline

Model Info

 **Try these examples:**

Software Developer with Python...

Marketing Manager needed to de...

Registered Nurse position avai...

Financial Analyst role focusin...

Sales Representative needed fo...

Sales Representative needed for B2B software sales. Experience with CRM systems and cold calling required.

Predict Job Category

💡 Tip: Press Ctrl+Enter to predict quickly

🎯 Predicted Category: Sales Development Representative

📊 Confidence: 2.4%

🏆 Top Predictions:

- Sales Development Representative: 2.4%
- Inside Sales Representative: 2.3%
- Outside Sales Representative: 1.9%

We used crucial FastAPI components to create a robust ML API. FastAPI creates the core app and routing, while Pydantic's BaseModel validates input data. HTTPException handles bespoke error responses and logs runtime events for troubleshooting. Additional modules such as os handle environment variables, whereas pickle loads the machine learning model. Type hints like Dict, List, and Optional guarantee type safety, while asynccontextmanager handles application lifecycle management. Collectively, these tools enable secure, stable API development, as demonstrated by a sample /predict endpoint that checks inputs, conducts predictions, and gracefully handles errors.

Student name:  
nirnaya khadka

Student id:  
24128420

## Training the model

### Data cleaning

We will clean job descriptions thoroughly, removing HTML tags and URLs, extending contractions, converting all language to lowercase, deleting stopwords, and utilizing lemmatization to reduce words to their basic form. This guarantees that the text is uniform and clear of noise for future examination.

### Feature Engineering

We will generate appropriate features by calculating job description length, counting words, and detecting technical terms. These features will help the model to make more accurate predictions.

### Text-to-Numbers Conversion

We will use the TF-IDF (Term Frequency-Inverse Document Frequency) technique to convert cleansed job description text into numerical vectors for machine learning models. TF-IDF gives more weight to terms that are relevant in specific job categories but appear less frequently throughout the dataset. This allows the model to focus on differentiating terms with relevant context for prediction tasks.

## Training the model

We will classify job postings according to their textual descriptions using a complex multi-algorithmic approach. First, cleaning and feature extraction are performed on the raw work data in the training pipeline. Next, different machine learning models will be trained concurrently so that their performance can be compared and the best algorithm can be selected. Robustness of the model and classification performance are enhanced through the use of ensemble-style methodology.

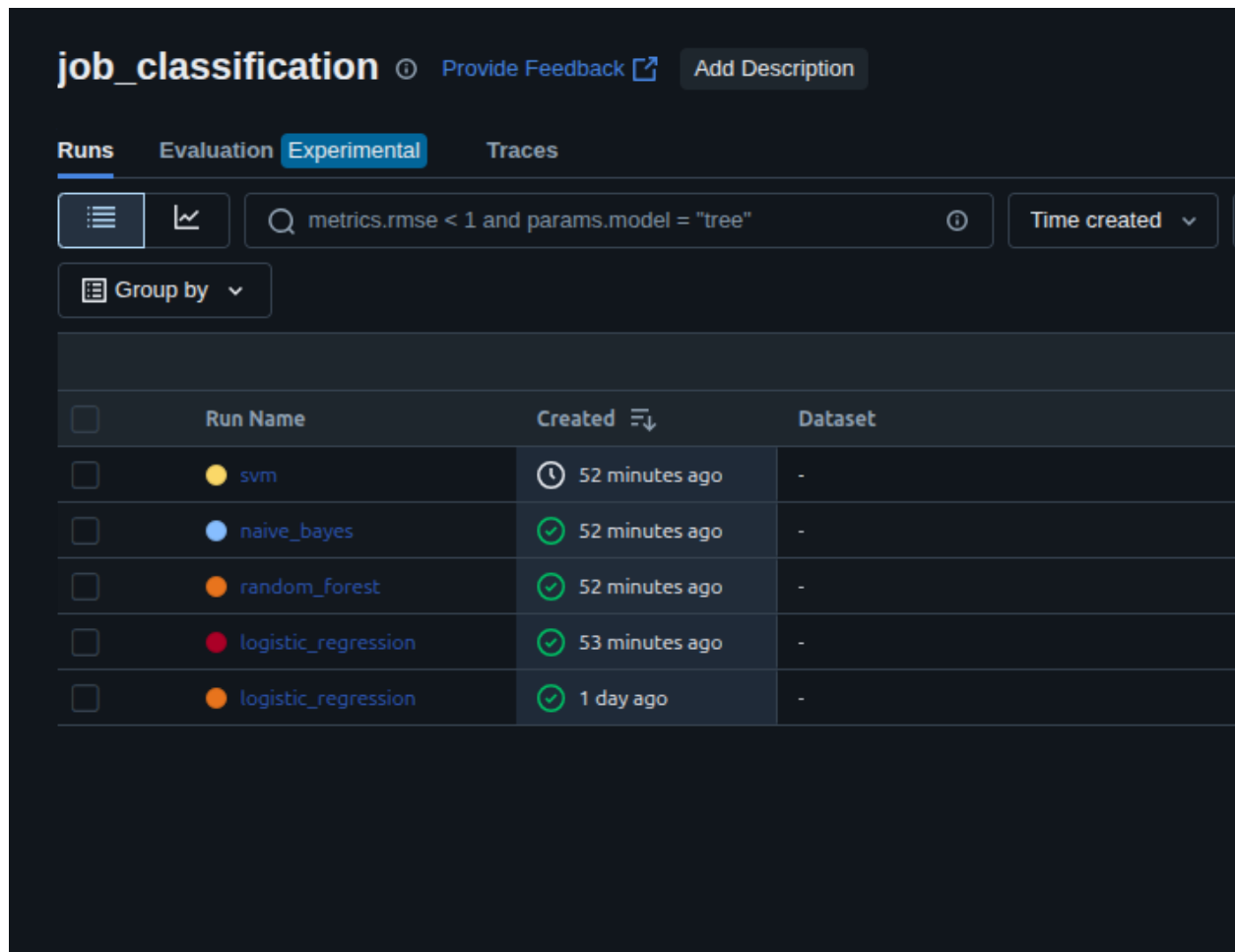


Student name:  
nirnaya khadka

Student id:  
24128420

## Logistic Regression

The baseline classification model that we shall choose is logistic regression. It works by determining the likelihood of each job category based on the frequency and presence of particular keywords. For instance, the model is more likely to predict a "Data Scientist" post if the job description contains terms like "Python" and "data analysis." Logistic regression is fast, easy to understand, and effective when job categories are divided using linear decision boundaries.



The screenshot shows a web interface for a job classification project. At the top, there's a header with the project name "job\_classification", a "Provide Feedback" link, and an "Add Description" button. Below the header, there are tabs for "Runs", "Evaluation", "Experimental" (which is active), and "Traces". Under the "Experimental" tab, there's a search bar with the query "metrics.rmse < 1 and params.model = 'tree'", a "Time created" dropdown, and a "Group by" dropdown. Below these controls is a table listing experimental runs.

<input type="checkbox"/>	Run Name	Created	Dataset
<input type="checkbox"/>	svm	52 minutes ago	-
<input type="checkbox"/>	naive_bayes	52 minutes ago	-
<input type="checkbox"/>	random_forest	52 minutes ago	-
<input type="checkbox"/>	logistic_regression	53 minutes ago	-
<input type="checkbox"/>	logistic_regression	1 day ago	-

## Random Forest

In our example, we will use Random Forest, which is an ensemble model that creates 200 decision trees. One of the many variables each tree considers when making a prediction is whether the job description includes the word "Python" or "management." The ultimate

Student name:  
nirnaya khadka

Student id:  
24128420

output is based on a majority vote of all the trees. Random Forest is less likely to overfit than single decision trees and is excellent at capturing complex patterns in data.

## Support Vector Machine (SVM)

By determining the optimal boundary between classes, we will classify job posts using Support Vector Machines (SVM). By utilizing important examples (support vectors) along the boundary decision, SVM tries to achieve the maximum margin of separation between classes. We will utilize a linear kernel for text information since it works well and is cost-effective for high-dimensional inputs such as TF-IDF vectors.

## Naive Bayes

We will use Naive Bayes, a probabilistic classifier that assumes each word contributes independently to the final prediction. Despite this simplifying assumption, it is extremely useful for text categorization tasks. For example, it determines the likelihood of a position being "Data Scientist" based on terms such as "Python" and "machine learning." Naive Bayes is very quick and works well with massive text datasets.

## Ensemble Voting

To boost prediction performance, we will employ an ensemble voting technique. Using a weighted majority vote, we combine the predictions of our top three models (Logistic Regression, Random Forest, and SVM). If two models predict "Software Engineer" and one predicts "Data Scientist," the ensemble chooses "Software Engineer." This method helps to balance the strengths and weaknesses of individual models, resulting in increased accuracy and resilience.

## Data Flow Optimization Points

- ❖ **Text preprocessing:** CPU-intensive cleaning operations
- ❖ **Vectorization:** Memory-intensive TF-IDF transformation
- ❖ **Database operations:** I/O bottlenecks during bulk inserts

Student name:  
nirnaya khadka

Student id:  
24128420

## Scalability Considerations







- ❖ **Horizontal scaling:** Multiple API instances
- ❖ **Data partitioning:** Time-based or category-based splits
- ❖ **Caching:** Redis for frequent predictions
- ❖ **Async processing:** Background model retraining

## Final result

Student name:  
nirnaya khadka

Student id:  
24128420

## Result of ml flow

Details	
Created at	06/03/2025, 11:14:31 AM
Created by	nirnaya
Experiment ID	165691988819825693 
Status	 Finished
Run ID	b44e140432a64a98a1efe49139750d9d 
Duration	17.4s
Datasets used	—
Tags	Add tags
Source	 main.py
Logged models	 sklearn
Registered models	 job_classifier_logistic_regression v1
Registered prompts	—
Parameters (15)	

The experiment was successfully executed, and it was marked as finished in 17.4 seconds. The main.py script launched the execution and determined

A Logistic Regression model was successfully registered under the model name job\_classifier\_logistic\_regression (version 1) after being trained using scikit-learn during the run. 15 parameters were recorded in this run, despite the fact that datasets and prompts were not specifically connected. This experiment design demonstrates a methodical and traceable approach to MLflow model construction.

Student name:  
nirnaya khadka

Student id:  
24128420

Overview

Model metrics

System metrics

Traces

Artifacts

Q Search parameters

Parameter	Value
dual	False
max_iter	1000
intercept_scaling	1
verbose	0
fit_intercept	True
class_weight	balanced
solver	lbfgs
warm_start	False
n_jobs	None
l1_ratio	None
random_state	42
multi_class	deprecated
C	1.0
penalty	l2
tol	0.0001

Q Search metrics

Metric	Value
Restaurant Manager_recall	0.75
Store Manager_f1_score	0.36363636363636365
Assistant Store Manager_precision	0.2857142857142857
Program Manager_f1_score	0
Vice President of Sales_recall	0
DevOps Engineer_f1_score	0.6666666666666666
Marketing Manager_precision	0.3333333333333333
Team Member CSR_recall	0.6666666666666666
Sales Development Representative...	0.375
ICU RN_precision	0.6666666666666666
Full Stack Engineer_f1_score	0.5
Sales Representative_recall	0.2
Python Developer_recall	1
Sales Specialist_recall	0.5
Project Manager_f1_score	0.16666666666666666
Executive Director_precision	0.5
Human Resources Manager_recall	0.75
Dental Assistant_f1_score	0.8
PT - Rehabilitation - f1_score	1

In order to guarantee reproducibility, the logistic regression model was trained with 1000 iterations, a fixed random state, class balancing, L2 regularization, and the lbfgs solver. It achieved flawless recall and F1 scores, which were especially good for jobs like PT-Rehabilitation and Python Developer. Strong predictive capacity for these classes was indicated by the high scores also seen for DevOps Engineer, ICU RN, and Dental Assistant (F1 score: 0.8).

Student name:  
nirnaya khadka

Student id:  
24128420


Q Search metrics	
Metric	Value
Restaurant Manager_recall	0.75
Store Manager_f1_score	0.36363636363636365
Assistant Store Manager_precision	0.2857142857142857
Program Manager_f1_score	0
Vice President of Sales_recall	0
DevOps Engineer_f1_score	0.6666666666666666
Marketing Manager_precision	0.3333333333333333
Team Member CSR_recall	0.6666666666666666
Sales Development Representative...	0.375
ICU RN_precision	0.6666666666666666
Full Stack Engineer_f1_score	0.5
Sales Representative_recall	0.2
Python Developer_recall	1
Sales Specialist_recall	0.5
Project Manager_f1_score	0.16666666666666666
Executive Director_precision	0.5
Human Resources Manager_recall	0.75
Dental Assistant_f1_score	0.8
PT - Rehabilitation -_f1_score	1

This table displays evaluation metrics (precision, recall, and F1-score) for different work roles. It is used to evaluate model performance in predicting job categories. High scores (for "Python Developer" and "PT - Rehabilitation") indicate good predictions, but zeros (for "Program Manager") indicate poor or no predictions, implying a class imbalance or a lack of training data.

Student name:  
nirnaya khadka

Student id:  
24128420

## Result of fast api

 **Job Description Classifier**

✓ System Ready - Model and preprocessors loaded successfully

Check Status

Reload Pipeline

Model Info

Try these examples:

Software Developer with Python...

Marketing Manager needed to de...

Registered Nurse position avai...

Financial Analyst role focusin...


Sales Representative needed fo...

Marketing Manager needed to develop and execute marketing campaigns. Experience with digital marketing, social media, and analytics required.

Predict Job Category

Tip: Press Ctrl+Enter to predict quickly

🎯 Predicted Category: Social Media Manager

 Confidence: 8.1%

🏆 Top Predictions:

Social Media Manager: 8.1%

Marketing Coordinator: 2.8%

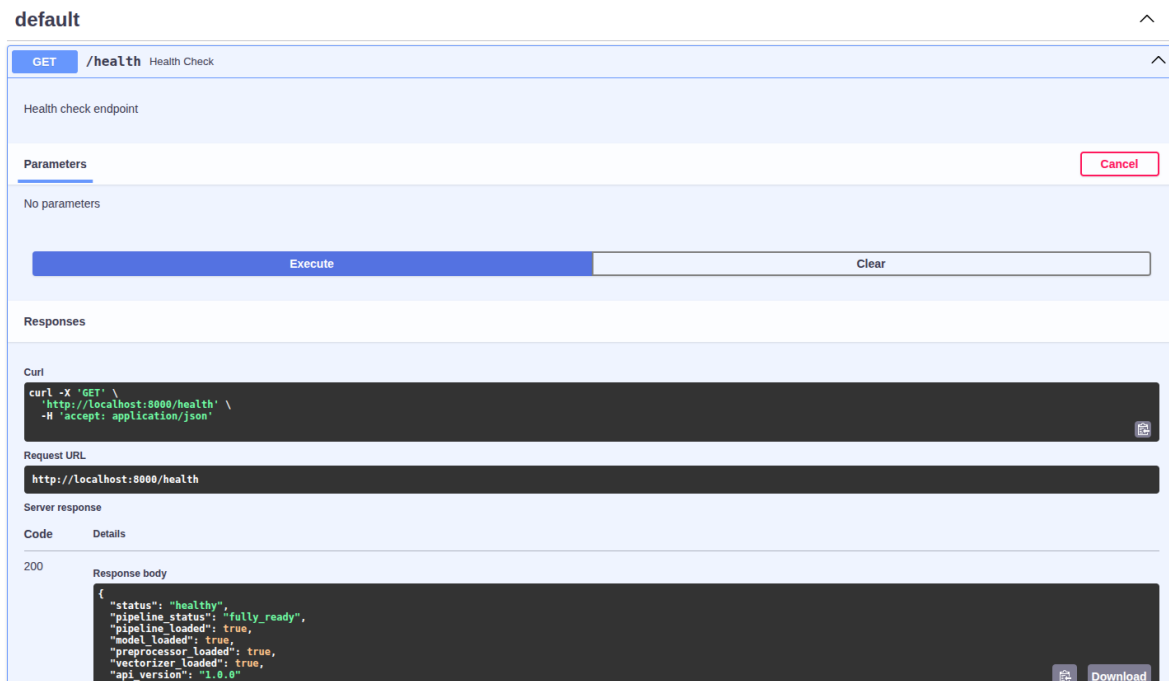
Marketing Manager: 2.5%

The FastAPI Job Description Classifier successfully loaded the model and preprocessing. When given a marketing-related job description, it suggested "Social Media Manager" as the most likely job category with a confidence level of 8.1%. Other close forecasts were Marketing Coordinator (2.8%) and Marketing Manager (2.5%). The relatively low confidence scores across all forecasts indicate that the model is ambiguous, probably due to overlapping features amongst similar marketing roles. The system is functional and

Student name:  
nirnaya khadka

Student id:  
24128420

responsive, making real-time predictions, although performance might be improved with greater class separation or additional training data for higher classification accuracy.



The FastAPI /health endpoint delivers an HTTP 200 response including detailed system status. It validates that the application is healthy and fully operational, with all important components (pipeline, model, preprocessor, and vectorizer) correctly loaded. The API's version is 1.0.0. This provides consistent monitoring and preparedness to handle requests.

## Summative Feedback Integration

We have integrating HTML responses with FastAPI, resulting in a more user-friendly and efficient front-end experience. This represents a departure from your earlier strategy, which lacked a front-end interface.



Student name:  
nirnaya khadka

Student id:  
24128420

Based on comments, we've also clearly outlined your machine learning process, with detailed descriptions of each stage. Furthermore, we've provided a clear and systematic description of our Exploratory Data Analysis (EDA), which helps to support your feature selection and preprocessing decisions.

Student name:  
nirnaya khadka

Student id:  
24128420

## References

J. Furnkranz, Machine Learning and Data Mining, Springer-Verlag Berlin Heidelberg, 2012.

RealDealAdamP (2023) *LinkedIn job postings - machine learning data set*, Kaggle. Available at: <https://www.kaggle.com/datasets/adampq/linkedin-jobs-machine-learning-data-set> (Accessed: 23 May 2025).

H. S. Obaid, S. A. Dheyab and S. S. Sabry, "The Impact of Data Pre-Processing Techniques and Dimensionality Reduction on the Accuracy of Machine Learning," *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, Jaipur, India, 2019, pp. 279-283, doi: 10.1109/IEMECONX.2019.8877011.

Student name:  
nirnaya khadka

Student id:  
24128420