SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week9 - Thursday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:17.4(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 17 – AI for Data Processing: Data Cleaning and Preprocessing Scripts** **Lab Objectives:** • Learn how to clean raw datasets using AI-assisted Python scripting.	Week9 - Thursday

	• Apply preprocessing techniques such as handling missing values, encoding categorical data, and normalization. • Automate repetitive data-cleaning tasks with AI-generated code. • Understand how preprocessing impacts model performance.	
	Task 1 – Employee Data Preprocessing **Task:** Use AI to generate a Python script for cleaning an employee dataset. **Instructions:** • Handle missing values in columns (salary, department, joining_date). • Convert the "joining_date" column into proper datetime format. • Standardize department names (e.g., "HR", "hr", "Human Resources" → "HR"). • Encode categorical variables (department, job_role). **Expected Output:** • A cleaned Pandas DataFrame with consistent departments, proper dates, and encoded features.	
	Task 2 – Sales Transaction Data Preprocessing **Task:** Use AI to generate a script for preprocessing a sales transaction dataset. **Instructions:** • Convert transaction dates to proper datetime format. • Create a new column for "Month-Year" from the transaction date. • Remove rows with negative or zero transaction amounts. • Normalize the "transaction_amount" column using Min-Max scaling. **Expected Output:** • A preprocessed DataFrame with valid dates, normalized amounts, and no invalid records.	
	Task 3 – Healthcare Patient Records Cleaning **Task:** Use AI to generate a script for cleaning healthcare patient records**.** **Instructions:** • Fill missing values in numeric columns (e.g., blood_pressure, heart_rate) with column mean. • Standardize units (convert height from cm to meters). • Correct inconsistent categorical labels (e.g., "M", "Male", "male"	

	→ "Male"). • Drop irrelevant columns such as patient_id after cleaning. **Expected Output:** • A cleaned healthcare dataset suitable for ML model training**.**	
	Task 4 – Social Media Sentiment Dataset Preparation **Task:** Use AI to write a script to preprocess a social media text dataset. **Instructions:** • Remove special characters, URLs, and emojis from text. • Convert all text to lowercase. • Tokenize and remove stopwords. • Apply lemmatization for standardizing words. **Expected Output:** • A processed dataset with clean text, ready for NLP sentiment analysis.	
	Task 5 – Financial Dataset Feature Engineering **Task:** Use AI to create a preprocessing script for a financial dataset. **Instructions:** • Handle missing values in stock price and volume. • Create new features such as moving average (7-day, 30-day). • Normalize continuous variables using StandardScaler. • Encode categorical columns (sector, company_name). **Expected Output:** • A feature-engineered DataFrame with new indicators and normalized values for ML tasks. ✅ Deliverables (For All Tasks) 1. AI-generated prompts for code and test case generation. 2. At least 3 assert test cases for each task. 3. AI-generated initial code and execution screenshots. 4. Analysis of whether code passes all tests. 5. Improved final version with inline comments and explanation. 6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.	

Task-1:

Code:

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('employee_data.csv')

# Handle missing values
df['salary'].fillna(df['salary'].mean(), inplace=True)
df['department'].fillna('Unknown', inplace=True)
df['joining_date'].fillna(df['joining_date'].mode()[0], inplace=True)

# Convert 'joining_date' to datetime format
df['joining_date'] = pd.to_datetime(df['joining_date'], errors='coerce')

# Standardize department names
department_mapping = {
    'HR': 'HR',
    'hr': 'HR',
    'Human Resources': 'HR',
    'IT': 'IT',
    'it': 'IT',
    'Information Technology': 'IT',
    # Add more mappings as needed
}

df['department'] = df['department'].replace(department_mapping)

# Encode categorical variables
df = pd.get_dummies(df, columns=['department', 'job_role'], drop_first=True)

# Display the cleaned DataFrame
print(df.head())
```

Original data:

	A	B	C	D	E	F	G	H	I
1	employee_	name	salary	departmer	job_role	joining_date			
2	101	Alice	55000	HR	Manager	########			
3	102	Bob		human res	Analyst	########			
4	103	Charlie	72000	IT	Developer	March 10 2019			
5	104	David	61000	Finance	Accountant				
6	105	Eva	58000	finance	Analyst	########			
7	106	Frank		Hr	Manager	########			
8	107	Grace	75000		Clerk	########			
9	108	Helen	50000	it	Developer	########			
10									
11									
12									
13									

Cleaned data:

```
employee_id,name,salary,department,job_role,joining_date
101,Alice,55000.0,HR,Ma[illegible]15
102,Bob,,human resource[illegible]4/2020
103,Charlie,72000.0,IT,Developer,March 10 2019
104,David,61000.0,Finance,Accountant,
105,Eva,58000.0,finance,Analyst,2020/06/01
106,Frank,,Hr,Manager,2022-07-20
107,Grace,75000.0,,Clerk,2018-12-05
108,Helen,50000.0,it,Developer,13-01-2023
```

Col 5: job_role

Task-2:

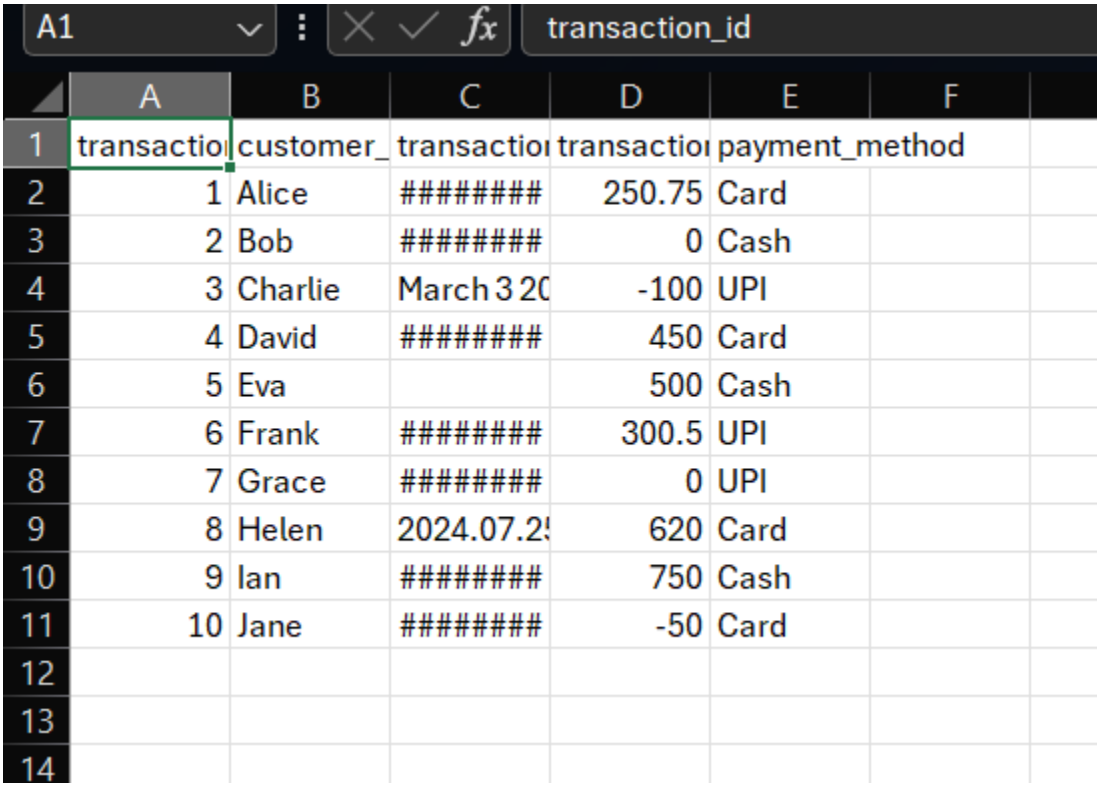
Code:

```
sk-2.py > ...
import pandas as pd

# Load the dataset
df = pd.read_csv('sales_transactions.csv')

# Convert transaction dates to proper datetime format
df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors='coerce')

# Create a new column for "Month-Year"
df['Month-Year'] = df['transaction_date'].dt.to_period('M')

# Remove rows with negative or zero transaction amounts
df = df[df['transaction_amount'] > 0]

# Normalize the "transaction_amount" column using Min-Max scaling
df['normalized_transaction_amount'] = (df['transaction_amount'] - df['transaction_amount'].min()) / (df['transaction_amount'].max() - df['transactio

# Output the preprocessed DataFrame
print(df)
```

Output:

```
[5 rows x 14 columns]
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-17.4> & C:/Users/Namitha/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Namitha/OneDrive/Desktop/AIAC/Lab-17.4/task-2.p
y
   transaction_id customer_name transaction_date  transaction_amount payment_method Month-Year  normalized_transaction_amount
0               1         Alice       2024-01-15              250.75           Card    2024-01                       0.000000
3               4         David              NaT              450.00           Card        NaT                       0.399099
4               5           Eva              NaT              500.00           Cash        NaT                       0.499249
5               6         Frank       2024-05-20              300.50            UPI    2024-05                       0.099649
7               8         Helen              NaT              620.00           Card        NaT                       0.739609
8               9           Ian       2024-08-10              750.00           Cash    2024-08                       1.000000
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-17.4>
```

Original data:



Cleaned data:

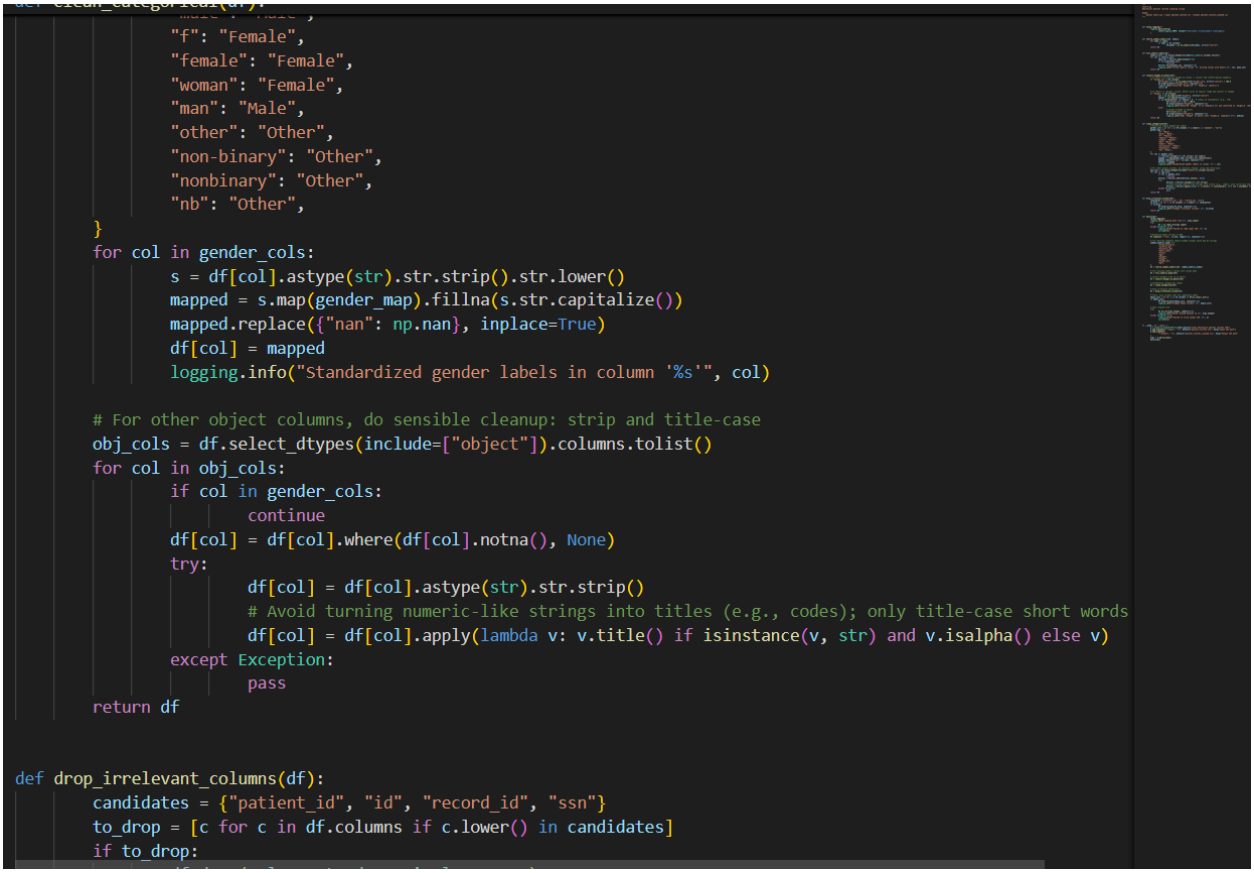
```
s_transactions.csv > data
transaction_id,customer_name,transaction_date,transaction_amount,payment_method
1,Alice,2024-01-15,250.75,Card
2,Bob,15/02/2024,0.0,Cash
3,Charlie,March 3 2024,-100.0,UPI
4,David,2024/04/10,450.0,Card
5,Eva,,500.0,Cash
6,Frank,2024-05-20,300.5,UPI
7,Grace,06-06-2024,0.0,UPI
8,Helen,2024.07.25,620.0,Card
9,Ian,2024-08-10,750.0,Cash
10,Jane,2024/09/05,-50.0,Card
```

Task-3:

Code:

```python
import argparse
import logging
import sys
import numpy as np
import pandas as pd

#!/usr/bin/env python3
"""
task-3.py
Healthcare patient records cleaning script.

Usage:
    python task-3.py --input patient_records.csv --output patient_records_cleaned.csv
"""




def setup_logging():
        logging.basicConfig(
                level=logging.INFO, format="%(asctime)s %(levelname)s %(message)s"
        )


def coerce_common_numeric(df, names):
        for name in names:
                if name in df.columns:
                        df[name] = pd.to_numeric(df[name], errors="coerce")
        return df


def fill_numeric_means(df):
        numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
        for col in numeric_cols:
                mean_val = df[col].mean(skipna=True)
                if np.isnan(mean_val):
                        continue
```

```
                df[col].fillna(mean_val, inplace=True)
                logging.info("Filled numeric column '%s' missing values with mean=%.4f", col, mean_val)
        return df


def convert_height_to_meters(df):
        # If there's an explicit height_cm column -> convert and create/replace height_m
        if "height_cm" in df.columns:
                df["height_m"] = pd.to_numeric(df["height_cm"], errors="coerce") / 100.0
                df.drop(columns=["height_cm"], inplace=True)
                logging.info("Converted 'height_cm' -> 'height_m' (meters)")
                return df

        # If there's a 'height' column, detect units by typical range and convert if needed
        if "height" in df.columns:
                col = pd.to_numeric(df["height"], errors="coerce")
                median = col.median(skipna=True)
                if pd.notna(median) and median > 3:  # likely in centimeters (e.g., 170)
                        df["height_m"] = col / 100.0
                        df.drop(columns=["height"], inplace=True)
                        logging.info("Detected 'height' in cm (median=%.2f) and converted to 'height_m' (meters)", m
                else:
                        # assume already in meters
                        df["height_m"] = col
                        df.drop(columns=["height"], inplace=True)
                        logging.info("Kept 'height' as meters into 'height_m' (median=%.2f)", median)
        return df


def clean_categorical(df):
        # Normalize common gender/sex labels
        gender_cols = [c for c in df.columns if c.lower() in ("gender", "sex")]
        gender_map = {
                "m": "Male",
                "male": "Male"
```



```
                "f": "Female",
                "female": "Female",
                "woman": "Female",
                "man": "Male",
                "other": "Other",
                "non-binary": "Other",
                "nonbinary": "Other",
                "nb": "Other",
        }
        for col in gender_cols:
                s = df[col].astype(str).str.strip().str.lower()
                mapped = s.map(gender_map).fillna(s.str.capitalize())
                mapped.replace({"nan": np.nan}, inplace=True)
                df[col] = mapped
                logging.info("Standardized gender labels in column '%s'", col)

        # For other object columns, do sensible cleanup: strip and title-case
        obj_cols = df.select_dtypes(include=["object"]).columns.tolist()
        for col in obj_cols:
                if col in gender_cols:
                        continue
                df[col] = df[col].where(df[col].notna(), None)
                try:
                        df[col] = df[col].astype(str).str.strip()
                        # Avoid turning numeric-like strings into titles (e.g., codes); only title-case short words
                        df[col] = df[col].apply(lambda v: v.title() if isinstance(v, str) and v.isalpha() else v)
                except Exception:
                        pass
        return df


def drop_irrelevant_columns(df):
        candidates = {"patient_id", "id", "record_id", "ssn"}
        to_drop = [c for c in df.columns if c.lower() in candidates]
        if to_drop:
```

```
            df.drop(columns=to_drop, inplace=True)
            logging.info("Dropped irrelevant columns: %s", to_drop)
    return df


def main(args):
    setup_logging()
    logging.info("Loading data from %s", args.input)
    try:
        df = pd.read_csv(args.input)
    except Exception as e:
        logging.error("Failed to read input CSV: %s", e)
        sys.exit(1)

    # Normalize empty strings to NaN
    df.replace(r"^\s*$", np.nan, regex=True, inplace=True)

    # Try coercing commonly numeric-named columns which may be strings
    common_numeric_names = [
        "blood_pressure",
        "systolic_bp",
        "diastolic_bp",
        "heart_rate",
        "pulse",
        "age",
        "weight",
        "height",
        "height_cm",
        "bmi",
    ]
    df = coerce_common_numeric(df, common_numeric_names)

    # Fill missing numeric values with column mean
    df = fill_numeric_means(df)
```

```
        df = convert_height_to_meters(df)

        # Standardize categorical labels
        df = clean_categorical(df)

        # Drop irrelevant identifiers
        df = drop_irrelevant_columns(df)

        # Final: drop columns that are completely empty
        empty_cols = [c for c in df.columns if df[c].isna().all()]
        if empty_cols:
                df.drop(columns=empty_cols, inplace=True)
                logging.info("Dropped empty columns: %s", empty_cols)

        # Save cleaned file
        try:
                df.to_csv(args.output, index=False)
                logging.info("Saved cleaned dataset to %s", args.output)
        except Exception as e:
                logging.error("Failed to write output CSV: %s", e)
                sys.exit(1)


f __name__ == "__main__":
        p = argparse.ArgumentParser(description="Clean healthcare patient records CSV")
        p.add_argument("--input", "-i", default="patient_records.csv", help="Input CSV path")
        p.add_argument(
                "--output", "-o", default="patient_records_cleaned.csv", help="Output CSV path"
        )
        args = p.parse_args()
        main(args)
```

Output:

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(mean_val, inplace=True)
2025-11-07 15:47:47,095 INFO Filled numeric column 'patient_id' missing values with mean=3.0000
2025-11-07 15:47:47,096 INFO Filled numeric column 'blood_pressure' missing values with mean=130.0000
2025-11-07 15:47:47,096 INFO Filled numeric column 'heart_rate' missing values with mean=82.5000
2025-11-07 15:47:47,097 INFO Filled numeric column 'height' missing values with mean=129.1750
2025-11-07 15:47:47,098 INFO Filled numeric column 'weight' missing values with mean=67.5000
2025-11-07 15:47:47,102 INFO Detected 'height' in cm (median=165.00) and converted to 'height_m' (meters)
2025-11-07 15:47:47,104 INFO Standardized gender labels in column 'gender'
2025-11-07 15:47:47,106 INFO Dropped irrelevant columns: ['patient_id']
2025-11-07 15:47:47,112 INFO Saved cleaned dataset to patient_records_cleaned.csv
```

Original data:

	A	B	C	D	E	F	G	H
1	patient_id	blood_pres	heart_rate	height	weight	gender		
2	1	120	80	170	65	M		
3	2		75	165	70	male		
4	3	140		1.7	80	Female		
5	4	130	90			f		
6	5		85	180	55	MALE		
7								
8								
9								
10								

Cleaned data:

patient_records_cleaned.csv > data

```
1   blood_pressure,heart_rate,weight,gender,height_m
2   120.0,80.0,65.0,Male,1.7
3   130.0,75.0,70.0,Male,1.65
4   140.0,82.5,80.0,Female,
5   130.0,90.0,67.5,Female,1.2917500000000002
6   130.0,85.0,55.0,Male,1.8
7
```

Col 5: height_m

Task-4:

Code:

```python
import pandas as pd
import re
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Load the dataset
df = pd.read_csv('social_media_raw_dataset.csv')

# Initialize lemmatizer and stopwords
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Remove special characters and emojis
    text = re.sub(r'[^\w\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return ' '.join(tokens)

# Apply preprocessing to the text column
df['cleaned_text'] = df['text'].apply(preprocess_text)

# Save the processed dataset
df.to_csv('processed_social_media_dataset.csv', index=False)
```

Original data:

	A	B	C	D	E	F	G	H
		blood_pres	heart_rate	height	weight	gender		
1		120	80	170	65	M		
2			75	165	70	male		
3		140		1.7	80	Female		
4		130	90			f		
5			85	180	55	MALE		

Cleaned data:

social_media_raw_dataset.csv > data

```
post_id,text
1,I loooove this product!!! 😍😍 Visit https://example.com for details!!!
2,Ughhh this is the WORST 😡😡 #disappointed
3,New update just dropped 🎉 Check it out: http://update.com
4,Totally neutral... I guess?? 🤷
5,WOW!!! Amazing service @brand 👏🔥🔥
```

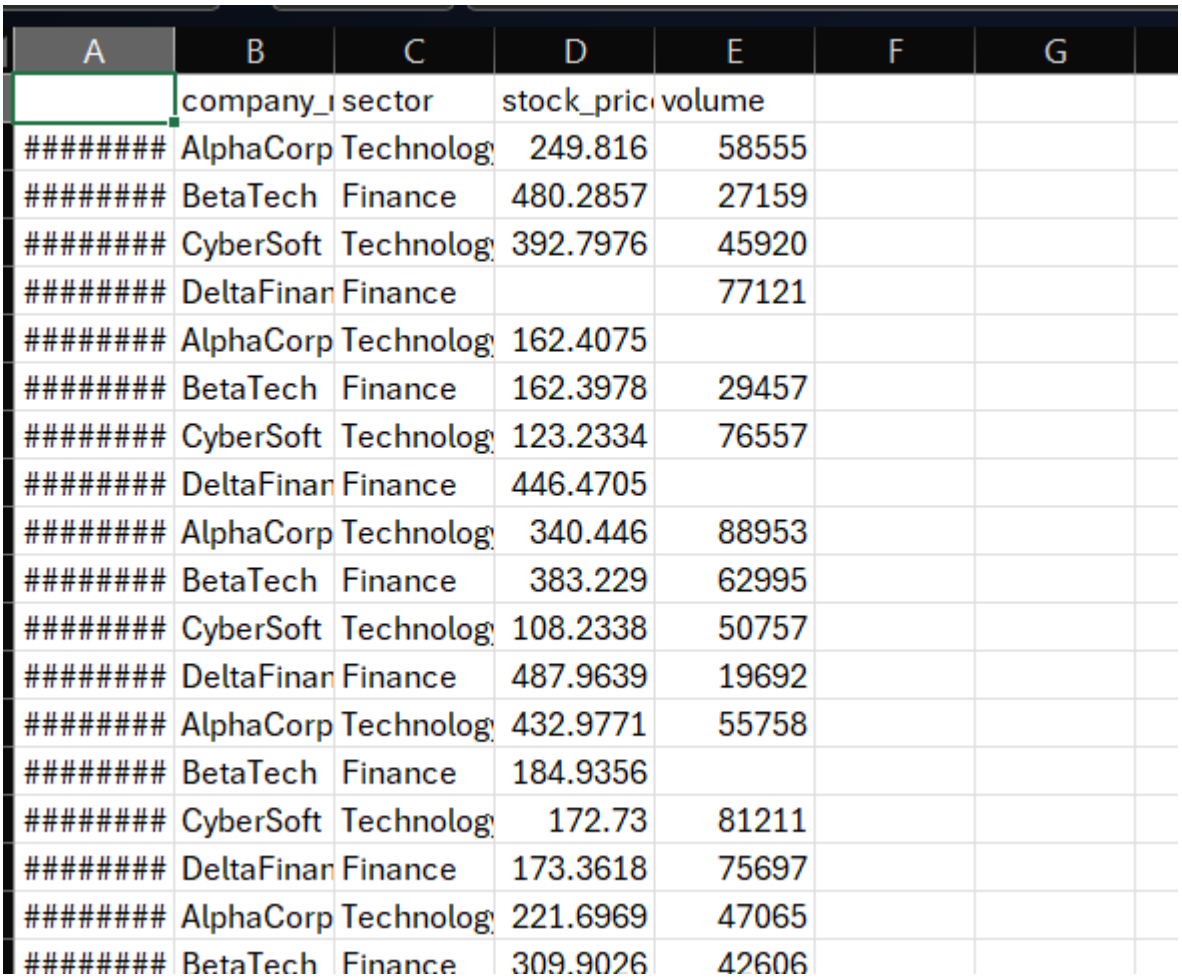
Task-5:

Code:

task-5.py > ...

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer

# Read the dataset
def preprocess_financial_data(file_path='financial_raw_dataset.csv'):
    try:
        # Read the CSV file
        df = pd.read_csv(file_path)

        # Handle missing values
        numeric_imputer = SimpleImputer(strategy='mean')
        numeric_columns = ['price', 'volume']
        df[numeric_columns] = numeric_imputer.fit_transform(df[numeric_columns])

        # Create moving averages
        df['7_day_ma'] = df.groupby('company_name')['price'].rolling(window=7).mean().reset_index(0, drop=True)
        df['30_day_ma'] = df.groupby('company_name')['price'].rolling(window=30).mean().reset_index(0, drop=True)

        # Create additional technical indicators
        df['price_change'] = df.groupby('company_name')['price'].pct_change()
        df['volume_change'] = df.groupby('company_name')['volume'].pct_change()

        # Normalize continuous variables
        scaler = StandardScaler()
        continuous_cols = ['price', 'volume', '7_day_ma', '30_day_ma', 'price_change', 'volume_change']
        df[continuous_cols] = scaler.fit_transform(df[continuous_cols])

        # Encode categorical variables
        le = LabelEncoder()
        categorical_cols = ['sector', 'company_name']
        for col in categorical_cols:
            df[f'{col}_encoded'] = le.fit_transform(df[col])
```

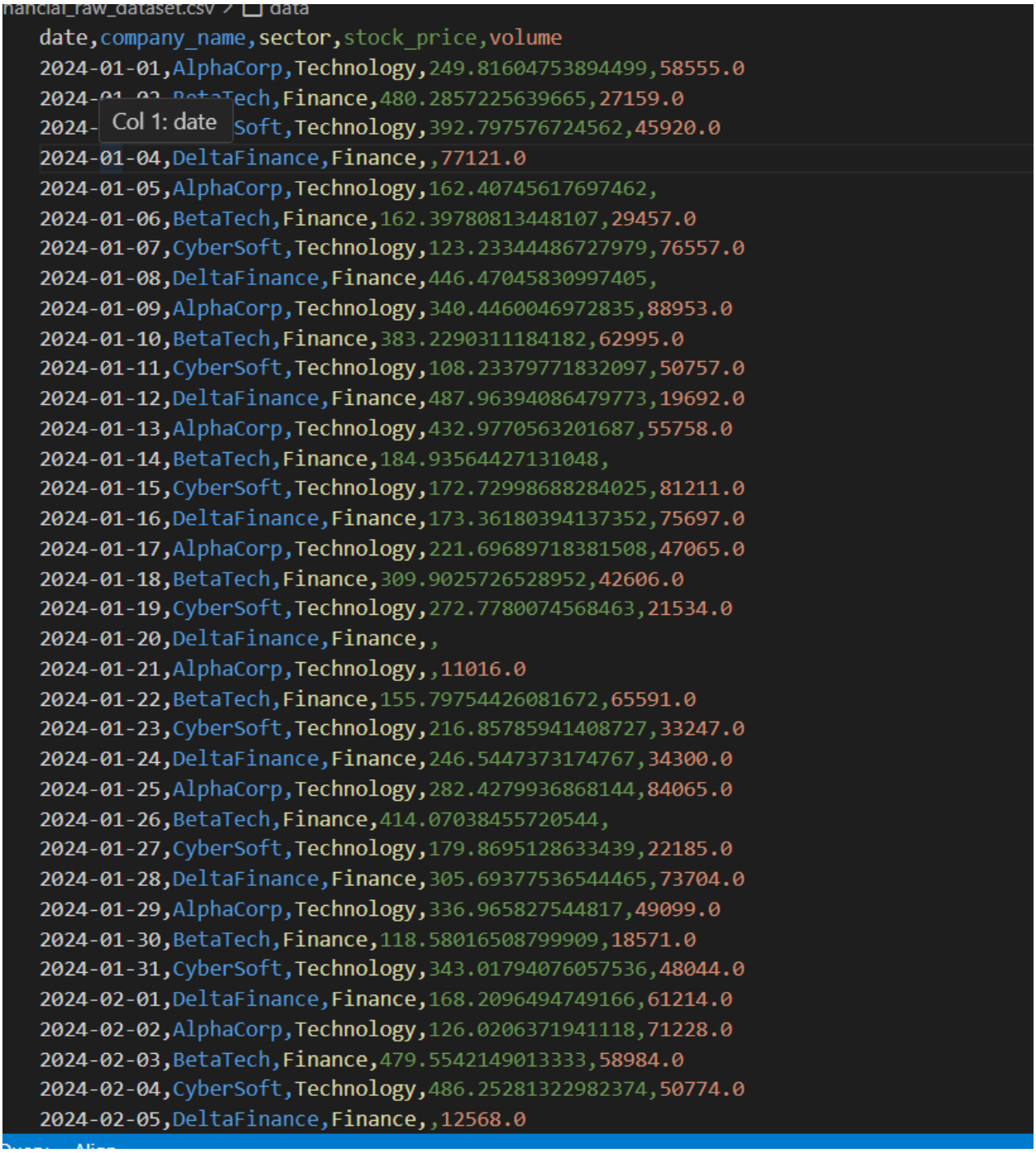
Oridinal data:



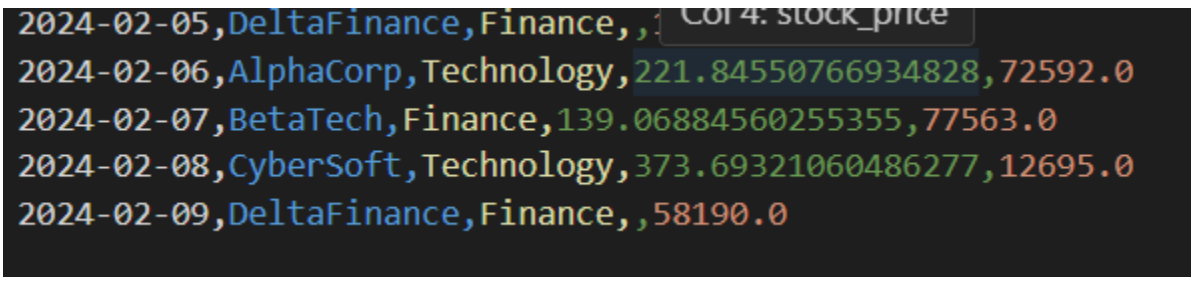
	A	B	C	D	E	F	G
		company_	sector	stock_pric	volume		
2	########	AlphaCorp	Technolog	249.816	58555		
3	########	BetaTech	Finance	480.2857	27159		
4	########	CyberSoft	Technolog	392.7976	45920		
5	########	DeltaFinan	Finance		77121		
6	########	AlphaCorp	Technolog	162.4075			
7	########	BetaTech	Finance	162.3978	29457		
8	########	CyberSoft	Technolog	123.2334	76557		
9	########	DeltaFinan	Finance	446.4705			
10	########	AlphaCorp	Technolog	340.446	88953		
11	########	BetaTech	Finance	383.229	62995		
12	########	CyberSoft	Technolog	108.2338	50757		
13	########	DeltaFinan	Finance	487.9639	19692		
14	########	AlphaCorp	Technolog	432.9771	55758		
15	########	BetaTech	Finance	184.9356			
16	########	CyberSoft	Technolog	172.73	81211		
17	########	DeltaFinan	Finance	173.3618	75697		
18	########	AlphaCorp	Technolog	221.6969	47065		
19	########	BetaTech	Finance	309.9026	42606		

A	B	C	D	E	F	G	
########	CyberSoft	Technolog	216.8579	33247			
########	DeltaFinar	Finance	246.5447	34300			
########	AlphaCorp	Technolog	282.428	84065			
########	BetaTech	Finance	414.0704				
########	CyberSoft	Technolog	179.8695	22185			
########	DeltaFinar	Finance	305.6938	73704			
########	AlphaCorp	Technolog	336.9658	49099			
########	BetaTech	Finance	118.5802	18571			
########	CyberSoft	Technolog	343.0179	48044			
########	DeltaFinar	Finance	168.2096	61214			
########	AlphaCorp	Technolog	126.0206	71228			
########	BetaTech	Finance	479.5542	58984			
########	CyberSoft	Technolog	486.2528	50774			
########	DeltaFinar	Finance		12568			
########	AlphaCorp	Technolog	221.8455	72592			
########	BetaTech	Finance	139.0688	77563			
########	CyberSoft	Technolog	373.6932	12695			
########	DeltaFinar	Finance		58190			

Cleaned data:




```
date,company_name,sector,stock_price,volume
2024-01-01,AlphaCorp,Technology,249.81604753894499,58555.0
2024-01-02,BetaTech,Finance,480.2857225639665,27159.0
2024-01-03,CyberSoft,Technology,392.797576724562,45920.0
2024-01-04,DeltaFinance,Finance,,77121.0
2024-01-05,AlphaCorp,Technology,162.40745617697462,
2024-01-06,BetaTech,Finance,162.39780813448107,29457.0
2024-01-07,CyberSoft,Technology,123.23344486727979,76557.0
2024-01-08,DeltaFinance,Finance,446.47045830997405,
2024-01-09,AlphaCorp,Technology,340.4460046972835,88953.0
2024-01-10,BetaTech,Finance,383.2290311184182,62995.0
2024-01-11,CyberSoft,Technology,108.23379771832097,50757.0
2024-01-12,DeltaFinance,Finance,487.96394086479773,19692.0
2024-01-13,AlphaCorp,Technology,432.9770563201687,55758.0
2024-01-14,BetaTech,Finance,184.93564427131048,
2024-01-15,CyberSoft,Technology,172.72998688284025,81211.0
2024-01-16,DeltaFinance,Finance,173.36180394137352,75697.0
2024-01-17,AlphaCorp,Technology,221.69689718381508,47065.0
2024-01-18,BetaTech,Finance,309.9025726528952,42606.0
2024-01-19,CyberSoft,Technology,272.7780074568463,21534.0
2024-01-20,DeltaFinance,Finance,,
2024-01-21,AlphaCorp,Technology,,11016.0
2024-01-22,BetaTech,Finance,155.79754426081672,65591.0
2024-01-23,CyberSoft,Technology,216.85785941408727,33247.0
2024-01-24,DeltaFinance,Finance,246.5447373174767,34300.0
2024-01-25,AlphaCorp,Technology,282.4279936868144,84065.0
2024-01-26,BetaTech,Finance,414.07038455720544,
2024-01-27,CyberSoft,Technology,179.8695128633439,22185.0
2024-01-28,DeltaFinance,Finance,305.69377536544465,73704.0
2024-01-29,AlphaCorp,Technology,336.965827544817,49099.0
2024-01-30,BetaTech,Finance,118.58016508799909,18571.0
2024-01-31,CyberSoft,Technology,343.01794076057536,48044.0
2024-02-01,DeltaFinance,Finance,168.2096494749166,61214.0
2024-02-02,AlphaCorp,Technology,126.0206371941118,71228.0
2024-02-03,BetaTech,Finance,479.5542149013333,58984.0
2024-02-04,CyberSoft,Technology,486.25281322982374,50774.0
2024-02-05,DeltaFinance,Finance,,12568.0
```




```
2024-02-05,DeltaFinance,Finance,,
2024-02-06,AlphaCorp,Technology,221.84550766934828,72592.0
2024-02-07,BetaTech,Finance,139.06884560255355,77563.0
2024-02-08,CyberSoft,Technology,373.69321060486277,12695.0
2024-02-09,DeltaFinance,Finance,,58190.0
```