AI ASSISSSTED CODING

NAME:- GUGGILLA ANUJA

HALLTICKET NO:2403A51101

BATCH NO:-06

COURSE NAME:- AI ASSISSTED CODING

FINAL SEM LAB EXAM

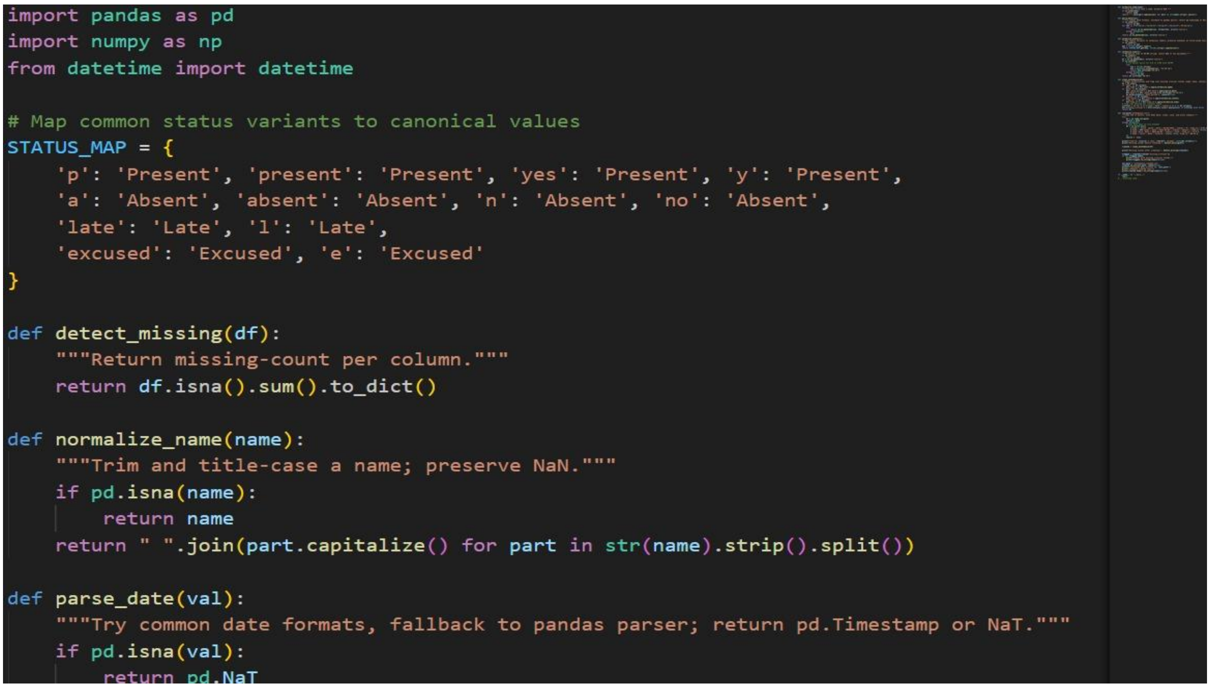
Q1: Clean attendance dataset

- Task 1: AI detects missing values.
- Task 2: Normalize data formatting.

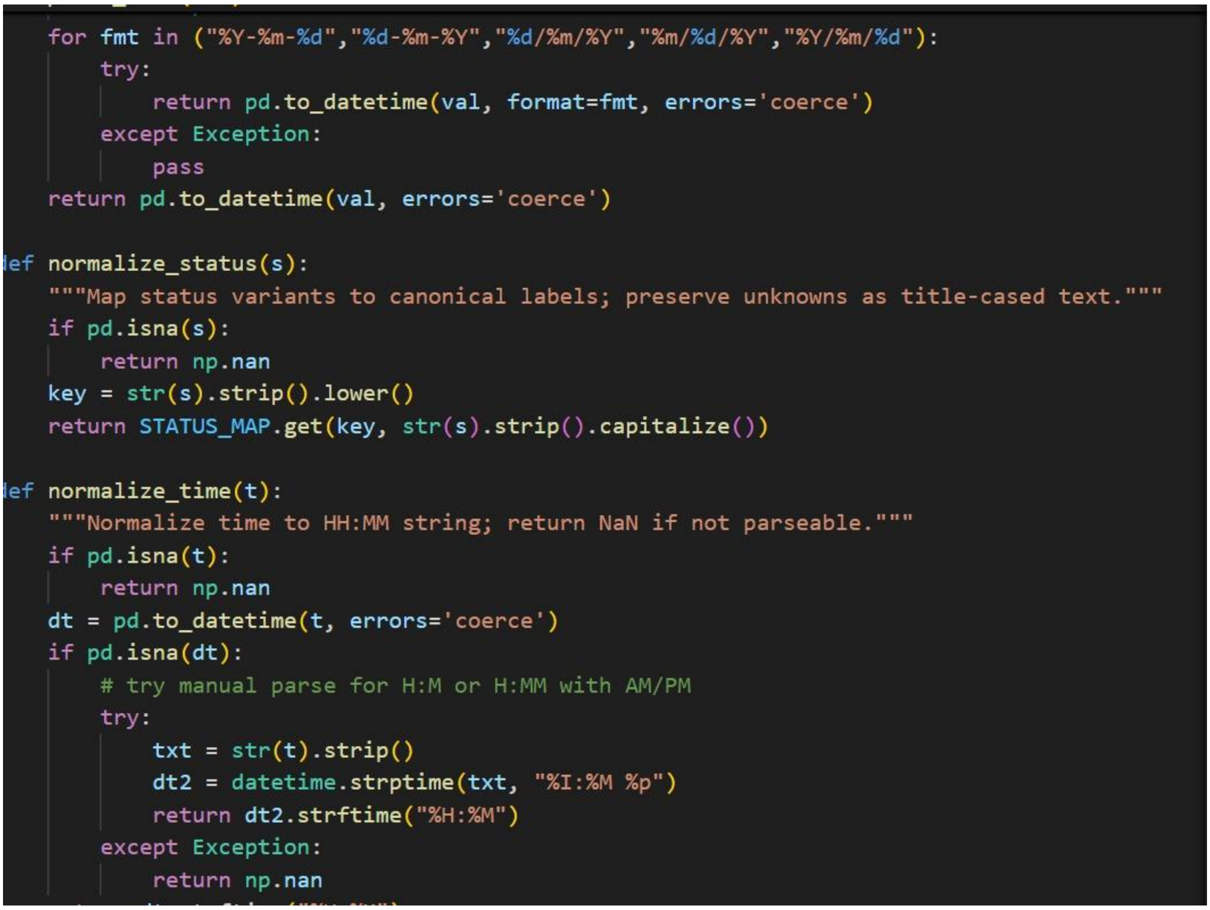
PROMPT:-

"Write a Python script to clean an attendance CSV: detect missing values, normalize names/dates/status/times, flag rows missing critical fields, save cleaned CSV and print a short summary. Add brief comments."

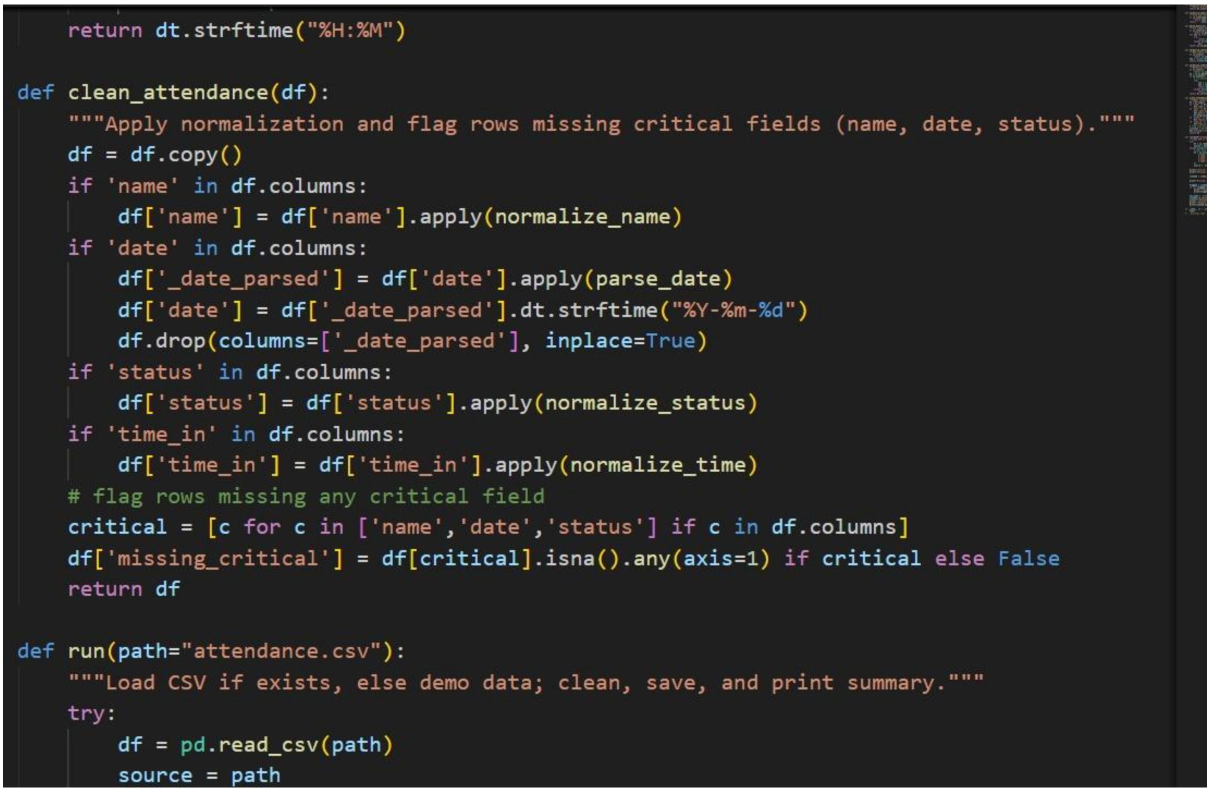
CODE:-



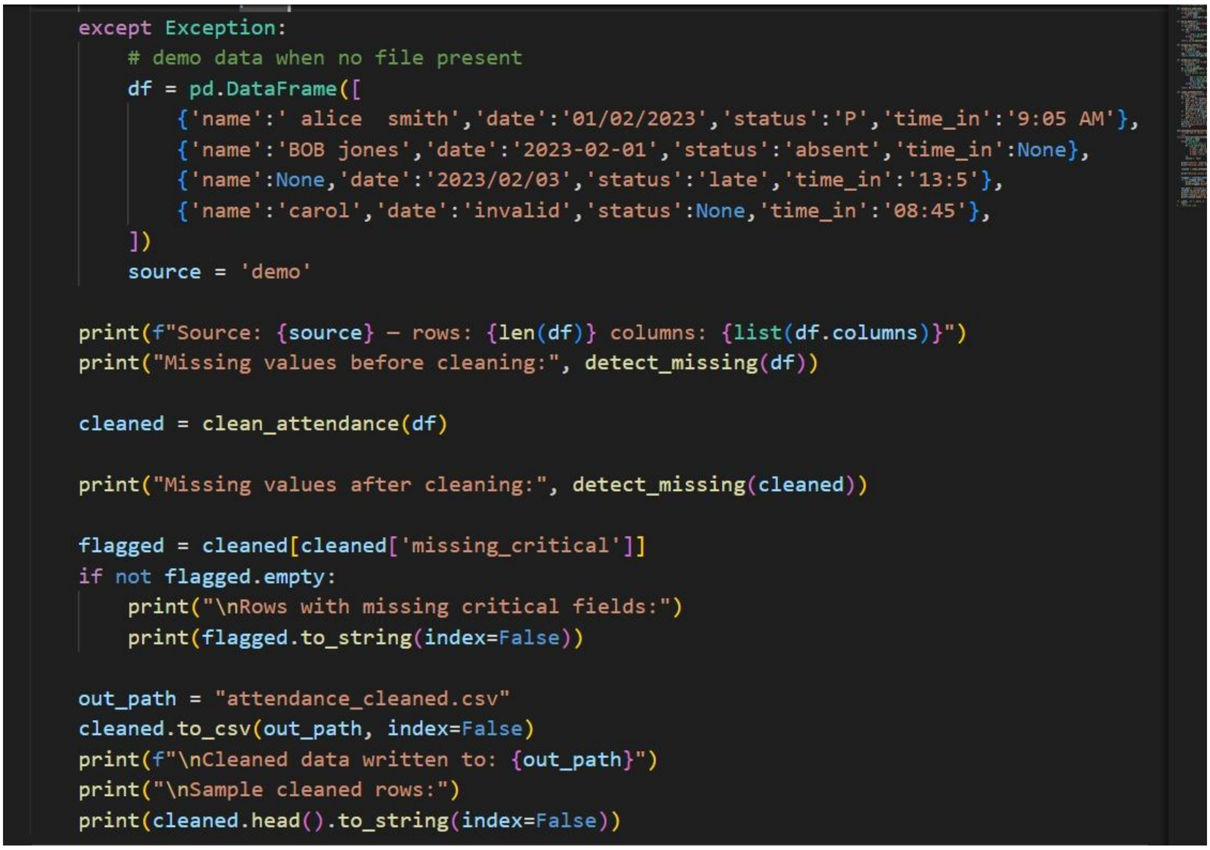
```python
import pandas as pd
import numpy as np
from datetime import datetime

# Map common status variants to canonical values
STATUS_MAP = {
    'p': 'Present', 'present': 'Present', 'yes': 'Present', 'y': 'Present',
    'a': 'Absent', 'absent': 'Absent', 'n': 'Absent', 'no': 'Absent',
    'late': 'Late', 'l': 'Late',
    'excused': 'Excused', 'e': 'Excused'
}

def detect_missing(df):
    """Return missing-count per column."""
    return df.isna().sum().to_dict()

def normalize_name(name):
    """Trim and title-case a name; preserve NaN."""
    if pd.isna(name):
        return name
    return " ".join(part.capitalize() for part in str(name).strip().split())

def parse_date(val):
    """Try common date formats, fallback to pandas parser; return pd.Timestamp or NaT."""
    if pd.isna(val):
        return pd.NaT
```



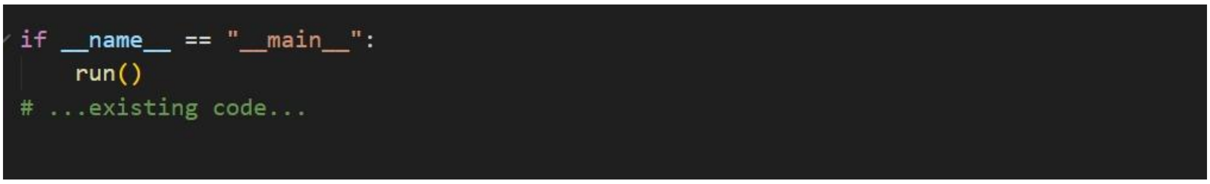
```python
    for fmt in ("%Y-%m-%d","%d-%m-%Y","%d/%m/%Y","%m/%d/%Y","%Y/%m/%d"):
        try:
            return pd.to_datetime(val, format=fmt, errors='coerce')
        except Exception:
            pass
    return pd.to_datetime(val, errors='coerce')

def normalize_status(s):
    """Map status variants to canonical labels; preserve unknowns as title-cased text."""
    if pd.isna(s):
        return np.nan
    key = str(s).strip().lower()
    return STATUS_MAP.get(key, str(s).strip().capitalize())

def normalize_time(t):
    """Normalize time to HH:MM string; return NaN if not parseable."""
    if pd.isna(t):
        return np.nan
    dt = pd.to_datetime(t, errors='coerce')
    if pd.isna(dt):
        # try manual parse for H:M or H:MM with AM/PM
        try:
            txt = str(t).strip()
            dt2 = datetime.strptime(txt, "%I:%M %p")
            return dt2.strftime("%H:%M")
        except Exception:
            return np.nan
```



```python
    return dt.strftime("%H:%M")

def clean_attendance(df):
    """Apply normalization and flag rows missing critical fields (name, date, status)."""
    df = df.copy()
    if 'name' in df.columns:
        df['name'] = df['name'].apply(normalize_name)
    if 'date' in df.columns:
        df['_date_parsed'] = df['date'].apply(parse_date)
        df['date'] = df['_date_parsed'].dt.strftime("%Y-%m-%d")
        df.drop(columns=['_date_parsed'], inplace=True)
    if 'status' in df.columns:
        df['status'] = df['status'].apply(normalize_status)
    if 'time_in' in df.columns:
        df['time_in'] = df['time_in'].apply(normalize_time)
    # flag rows missing any critical field
    critical = [c for c in ['name','date','status'] if c in df.columns]
    df['missing_critical'] = df[critical].isna().any(axis=1) if critical else False
    return df

def run(path="attendance.csv"):
    """Load CSV if exists, else demo data; clean, save, and print summary."""
    try:
        df = pd.read_csv(path)
        source = path
```

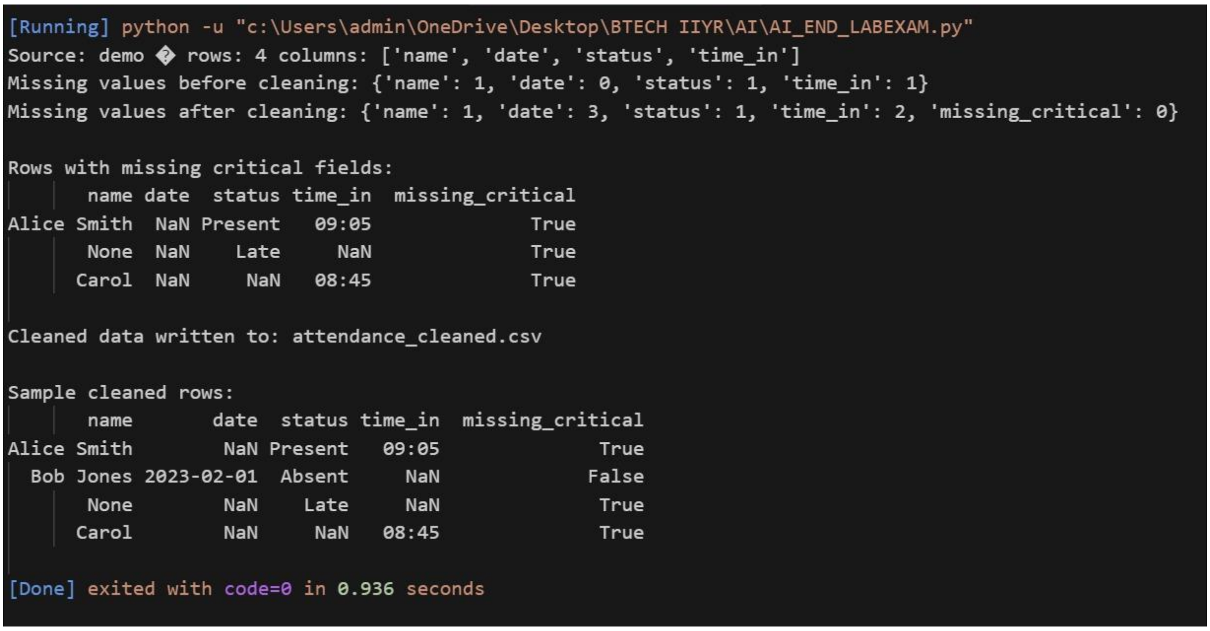


```python
    except Exception:
        # demo data when no file present
        df = pd.DataFrame([
            {'name':' alice  smith','date':'01/02/2023','status':'P','time_in':'9:05 AM'},
            {'name':'BOB jones','date':'2023-02-01','status':'absent','time_in':None},
            {'name':None,'date':'2023/02/03','status':'late','time_in':'13:5'},
            {'name':'carol','date':'invalid','status':None,'time_in':'08:45'},
        ])
        source = 'demo'

    print(f"Source: {source} — rows: {len(df)} columns: {list(df.columns)}")
    print("Missing values before cleaning:", detect_missing(df))

    cleaned = clean_attendance(df)

    print("Missing values after cleaning:", detect_missing(cleaned))

    flagged = cleaned[cleaned['missing_critical']]
    if not flagged.empty:
        print("\nRows with missing critical fields:")
        print(flagged.to_string(index=False))

    out_path = "attendance_cleaned.csv"
    cleaned.to_csv(out_path, index=False)
    print(f"\nCleaned data written to: {out_path}")
    print("\nSample cleaned rows:")
    print(cleaned.head().to_string(index=False))
```



```python
if __name__ == "__main__":
    run()
# ...existing code...
```

OUTPUT:-



```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\BTECH IIYR\AI\AI_END_LABEXAM.py"
Source: demo � rows: 4 columns: ['name', 'date', 'status', 'time_in']
Missing values before cleaning: {'name': 1, 'date': 0, 'status': 1, 'time_in': 1}
Missing values after cleaning: {'name': 1, 'date': 3, 'status': 1, 'time_in': 2, 'missing_critical': 0}

Rows with missing critical fields:
       name date  status time_in  missing_critical
Alice Smith  NaN Present   09:05              True
       None  NaN    Late     NaN              True
      Carol  NaN     NaN   08:45              True

Cleaned data written to: attendance_cleaned.csv

Sample cleaned rows:
       name       date  status time_in  missing_critical
Alice Smith        NaN Present   09:05              True
  Bob Jones 2023-02-01  Absent     NaN             False
       None        NaN    Late     NaN              True
      Carol        NaN     NaN   08:45              True

[Done] exited with code=0 in 0.936 seconds
```

OBSERVATION:-

Detects missing values per column and normalizes fields: name → Title Case, date → ISO YYYY-MM-DD (best-effort parsing), status → Present/Absent/Late/Excused mapping, time_in → HH:MM where parseable.

Flags rows missing any critical field (name, date, status). Writes cleaned CSV and prints a concise summary.

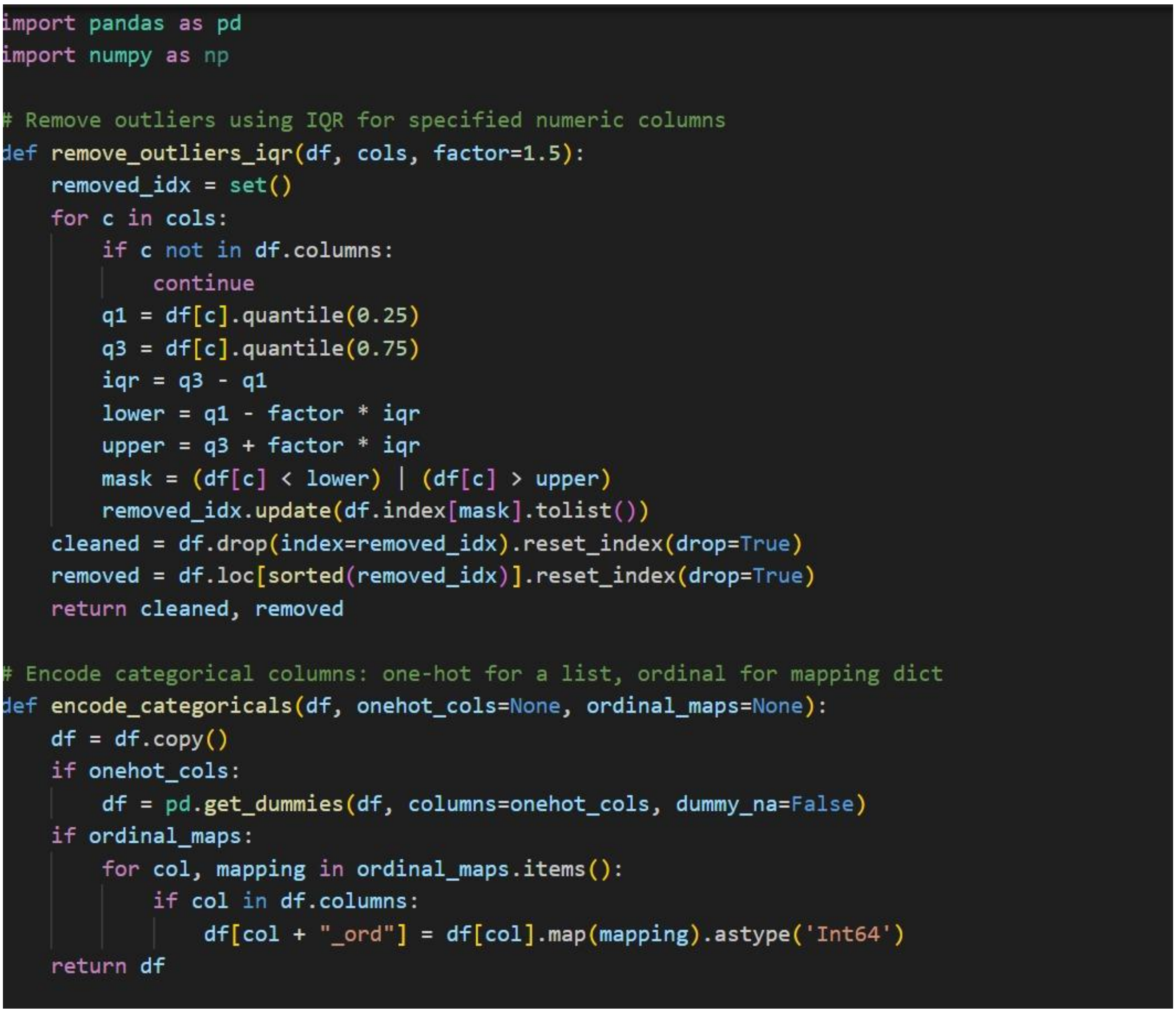
Q2: Preprocess grades

- Task 1: AI removes outliers.
- Task 2: Encode categorical values

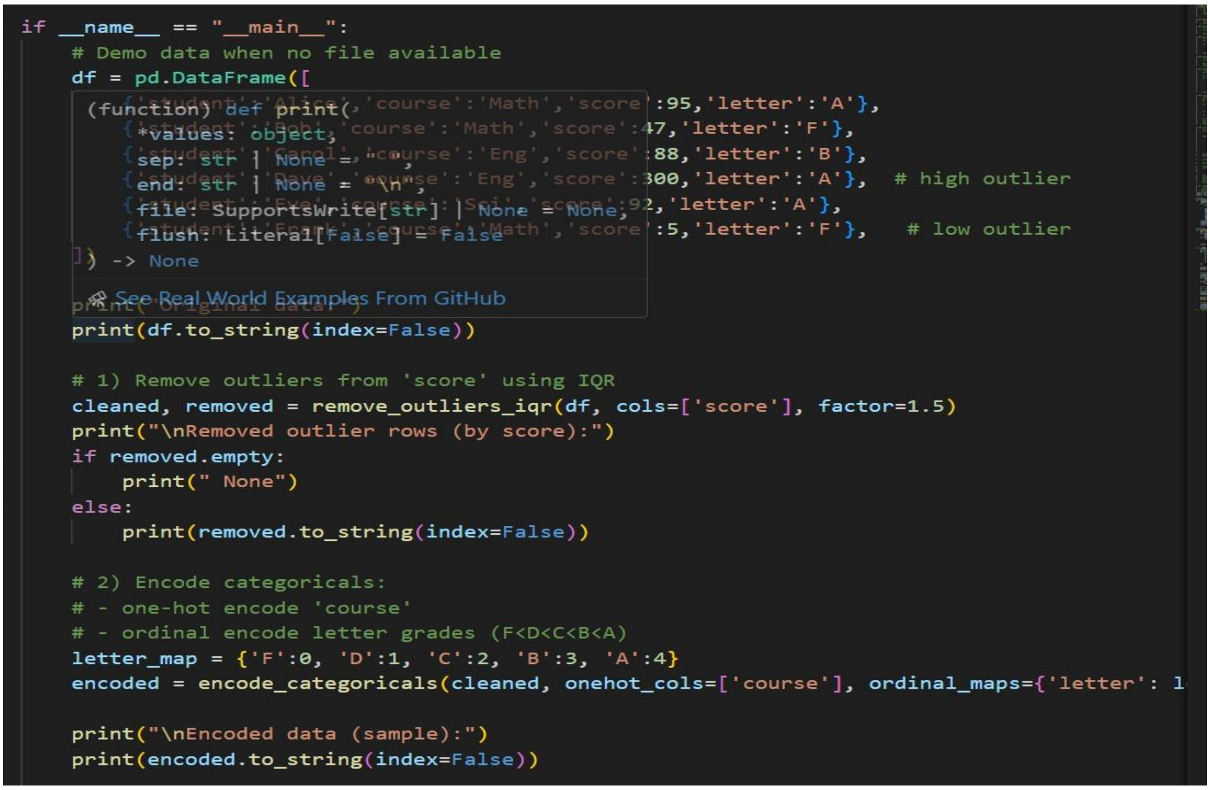
PROMPT:-

"Create a Python script to preprocess a grades dataset: detect & remove numeric outliers (IQR), encode categorical columns (one-hot for course, ordinal for letter grades). Print removed rows and show encoded output."

CODE:-

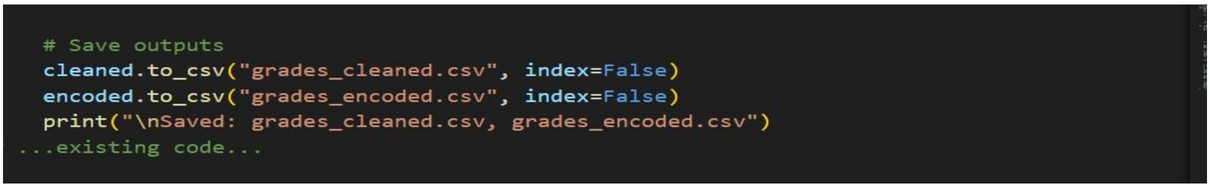


```
import pandas as pd
import numpy as np

# Remove outliers using IQR for specified numeric columns
def remove_outliers_iqr(df, cols, factor=1.5):
    removed_idx = set()
    for c in cols:
        if c not in df.columns:
            continue
        q1 = df[c].quantile(0.25)
        q3 = df[c].quantile(0.75)
        iqr = q3 - q1
        lower = q1 - factor * iqr
        upper = q3 + factor * iqr
        mask = (df[c] < lower) | (df[c] > upper)
        removed_idx.update(df.index[mask].tolist())
    cleaned = df.drop(index=removed_idx).reset_index(drop=True)
    removed = df.loc[sorted(removed_idx)].reset_index(drop=True)
    return cleaned, removed

# Encode categorical columns: one-hot for a list, ordinal for mapping dict
def encode_categoricals(df, onehot_cols=None, ordinal_maps=None):
    df = df.copy()
    if onehot_cols:
        df = pd.get_dummies(df, columns=onehot_cols, dummy_na=False)
    if ordinal_maps:
        for col, mapping in ordinal_maps.items():
            if col in df.columns:
                df[col + "_ord"] = df[col].map(mapping).astype('Int64')
    return df
```



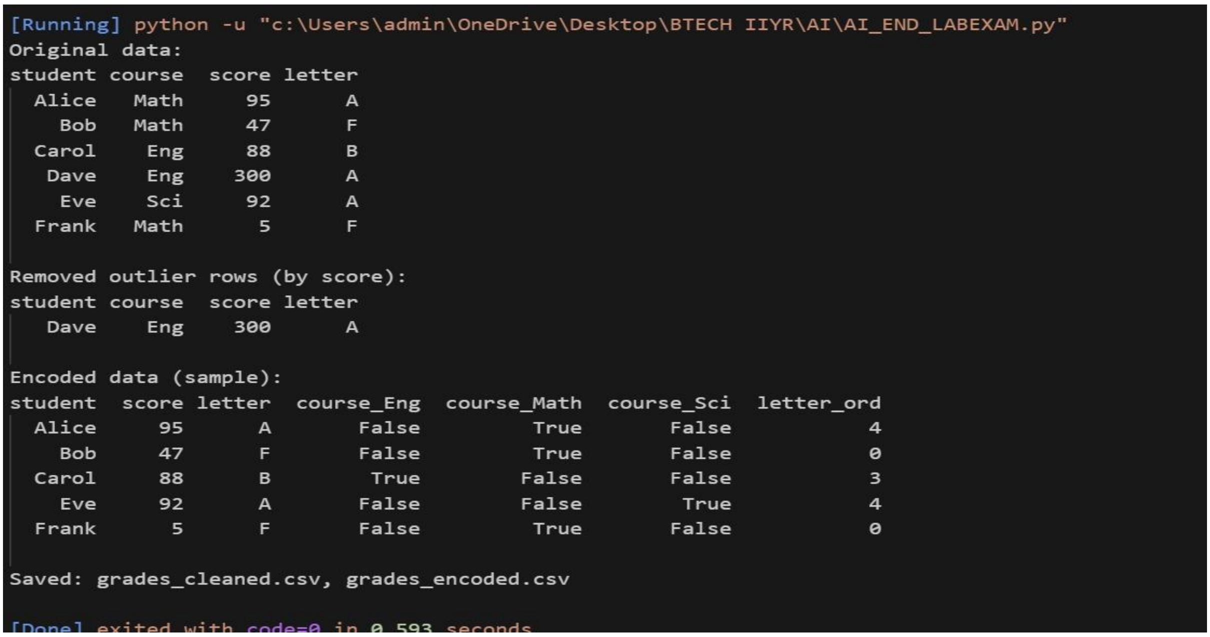
```
if __name__ == "__main__":
    # Demo data when no file available
    df = pd.DataFrame([
        {'student':'Alice','course':'Math','score':95,'letter':'A'},
        {'student':'Bob','course':'Math','score':47,'letter':'F'},
        {'student':'Carol','course':'Eng','score':88,'letter':'B'},
        {'student':'Dave','course':'Eng','score':300,'letter':'A'},   # high outlier
        {'student':'Eve','course':'Sci','score':92,'letter':'A'},
        {'student':'Frank','course':'Math','score':5,'letter':'F'},    # low outlier
    ])
    print("Original data:")
    print(df.to_string(index=False))

    # 1) Remove outliers from 'score' using IQR
    cleaned, removed = remove_outliers_iqr(df, cols=['score'], factor=1.5)
    print("\nRemoved outlier rows (by score):")
    if removed.empty:
        print(" None")
    else:
        print(removed.to_string(index=False))

    # 2) Encode categoricals:
    # - one-hot encode 'course'
    # - ordinal encode letter grades (F<D<C<B<A)
    letter_map = {'F':0, 'D':1, 'C':2, 'B':3, 'A':4}
    encoded = encode_categoricals(cleaned, onehot_cols=['course'], ordinal_maps={'letter': l

    print("\nEncoded data (sample):")
    print(encoded.to_string(index=False))
```

```
(function) def print(
    *values: object,
    sep: str | None = " ",
    end: str | None = "\n",
    file: SupportsWrite[str] | None = None,
    flush: Literal[False] = False
) -> None

See Real World Examples From GitHub
```



```
    # Save outputs
    cleaned.to_csv("grades_cleaned.csv", index=False)
    encoded.to_csv("grades_encoded.csv", index=False)
    print("\nSaved: grades_cleaned.csv, grades_encoded.csv")
...existing code...
```

OUTPUT:-



```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\BTECH IIYR\AI\AI_END_LABEXAM.py"
Original data:
student course  score letter
  Alice   Math     95      A
    Bob   Math     47      F
  Carol    Eng     88      B
   Dave    Eng    300      A
    Eve    Sci     92      A
  Frank   Math      5      F

Removed outlier rows (by score):
student course  score letter
   Dave    Eng    300      A

Encoded data (sample):
student  score letter  course_Eng  course_Math  course_Sci  letter_ord
  Alice     95      A       False         True       False           4
    Bob     47      F       False         True       False           0
  Carol     88      B        True        False       False           3
    Eve     92      A       False        False        True           4
  Frank      5      F       False         True       False           0

Saved: grades_cleaned.csv, grades_encoded.csv

[Done] exited with code=0 in 0.593 seconds
```

OBSERVATION:-

Outliers removed by IQR per numeric column; removed rows returned separately for review.

Categorical encoding: one-hot for nominal columns, ordinal mapping for ordered categories (letter grades).

Script writes cleaned and encoded CSVs and prints removed rows and final encoded table.