NAME:-PEDDAPELLI ANUSHA

HALL.TICKET.NO:-2403A51103

COURSE:-AI ASSISSTED CODING

DATE:19/11/2025

LAB TEST-04

Q1. Training dataset contains categorical, numeric, and timestamp columns.

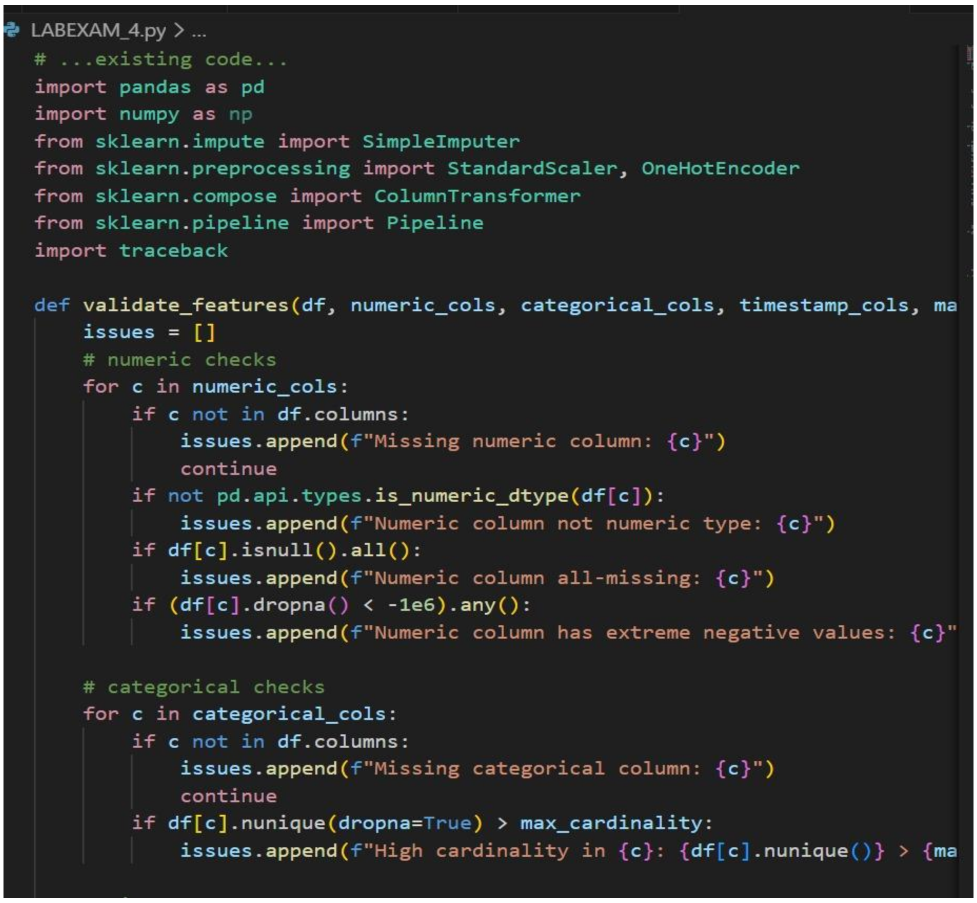
a) Create AI-assisted mixed-type preprocessing script.

b) Explain feature validation rules.

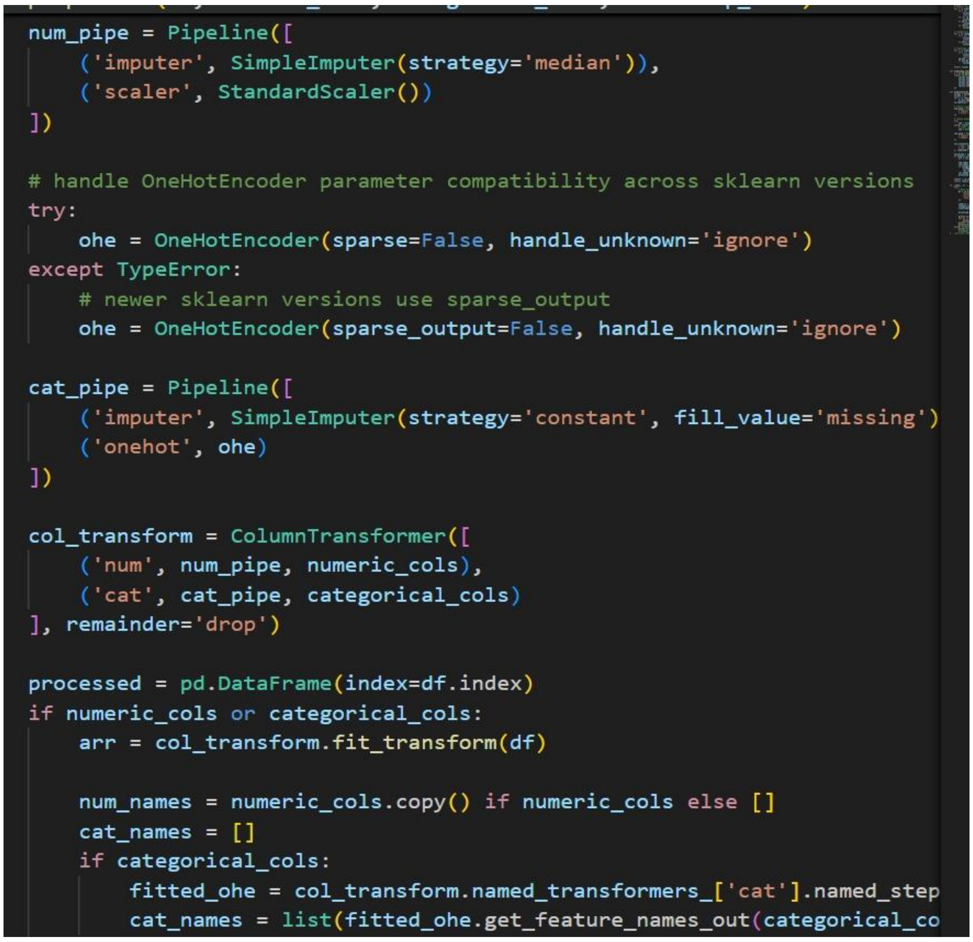
PROMPT:-

"Create a Python script that AI-assists mixed-type preprocessing for a dataset with categorical, numeric, and timestamp columns. Include feature validation rules and a small demo."

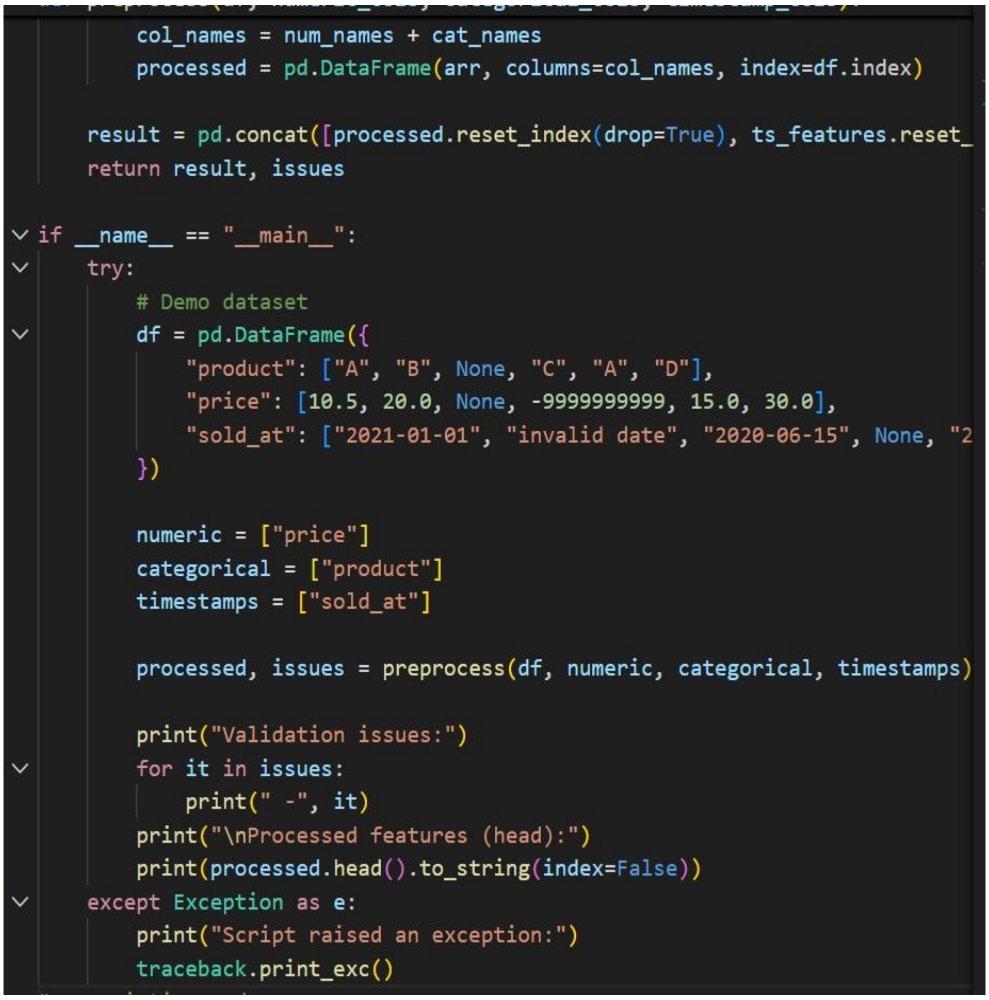
CODE:-


LABEXAM_4.py > ...

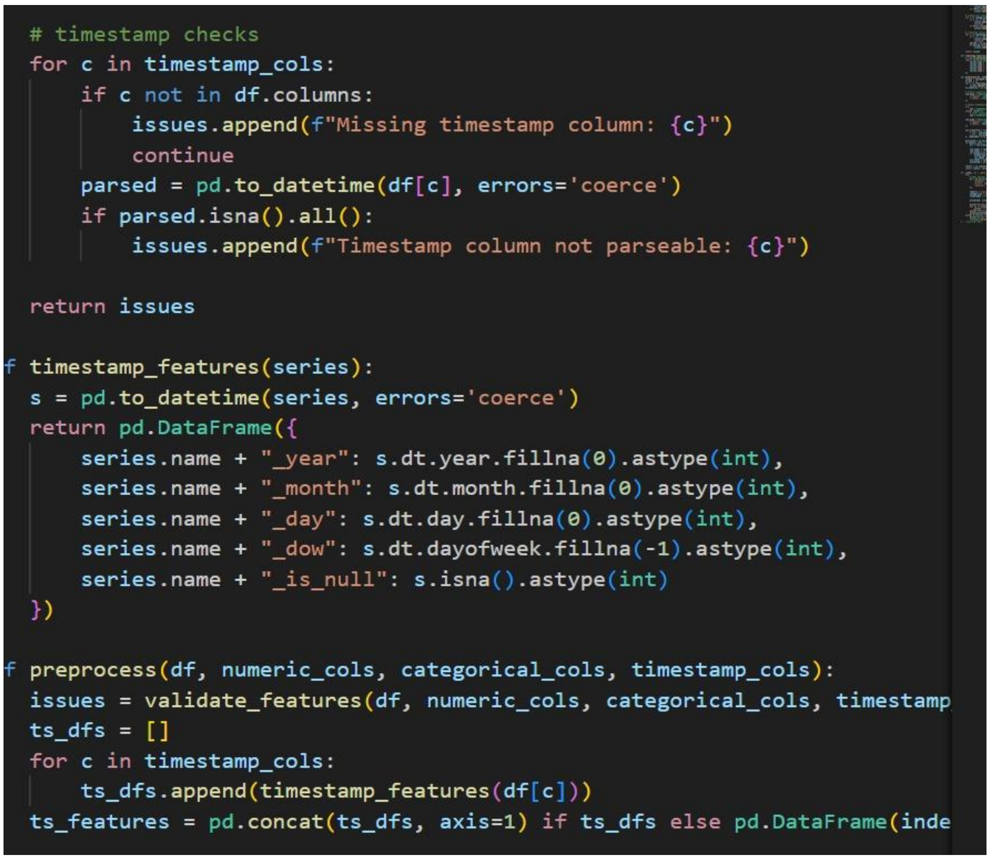
```python
# ...existing code...
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import traceback

def validate_features(df, numeric_cols, categorical_cols, timestamp_cols, ma
    issues = []
    # numeric checks
    for c in numeric_cols:
        if c not in df.columns:
            issues.append(f"Missing numeric column: {c}")
            continue
        if not pd.api.types.is_numeric_dtype(df[c]):
            issues.append(f"Numeric column not numeric type: {c}")
        if df[c].isnull().all():
            issues.append(f"Numeric column all-missing: {c}")
        if (df[c].dropna() < -1e6).any():
            issues.append(f"Numeric column has extreme negative values: {c}"

    # categorical checks
    for c in categorical_cols:
        if c not in df.columns:
            issues.append(f"Missing categorical column: {c}")
            continue
        if df[c].nunique(dropna=True) > max_cardinality:
            issues.append(f"High cardinality in {c}: {df[c].nunique()} > {ma
```



```python
num_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# handle OneHotEncoder parameter compatibility across sklearn versions
try:
    ohe = OneHotEncoder(sparse=False, handle_unknown='ignore')
except TypeError:
    # newer sklearn versions use sparse_output
    ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

cat_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing'))
    ('onehot', ohe)
])

col_transform = ColumnTransformer([
    ('num', num_pipe, numeric_cols),
    ('cat', cat_pipe, categorical_cols)
], remainder='drop')

processed = pd.DataFrame(index=df.index)
if numeric_cols or categorical_cols:
    arr = col_transform.fit_transform(df)

    num_names = numeric_cols.copy() if numeric_cols else []
    cat_names = []
    if categorical_cols:
        fitted_ohe = col_transform.named_transformers_['cat'].named_step
        cat_names = list(fitted_ohe.get_feature_names_out(categorical_co
```

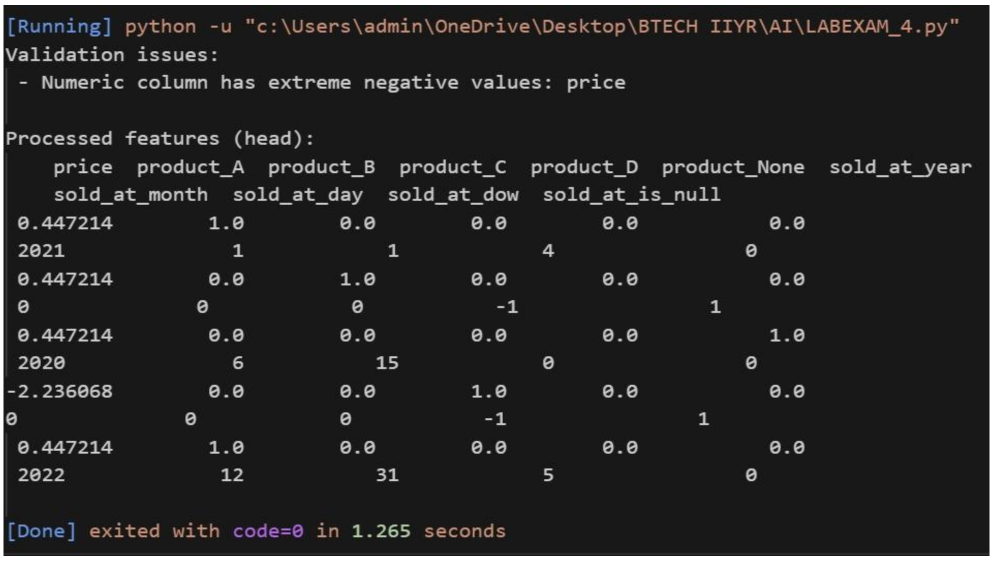


```
        col_names = num_names + cat_names
        processed = pd.DataFrame(arr, columns=col_names, index=df.index)

    result = pd.concat([processed.reset_index(drop=True), ts_features.reset_
    return result, issues

if __name__ == "__main__":
    try:
        # Demo dataset
        df = pd.DataFrame({
            "product": ["A", "B", None, "C", "A", "D"],
            "price": [10.5, 20.0, None, -9999999999, 15.0, 30.0],
            "sold_at": ["2021-01-01", "invalid date", "2020-06-15", None, "2
        })

        numeric = ["price"]
        categorical = ["product"]
        timestamps = ["sold_at"]

        processed, issues = preprocess(df, numeric, categorical, timestamps)

        print("Validation issues:")
        for it in issues:
            print(" -", it)
        print("\nProcessed features (head):")
        print(processed.head().to_string(index=False))
    except Exception as e:
        print("Script raised an exception:")
        traceback.print_exc()
```



```
    # timestamp checks
    for c in timestamp_cols:
        if c not in df.columns:
            issues.append(f"Missing timestamp column: {c}")
            continue
        parsed = pd.to_datetime(df[c], errors='coerce')
        if parsed.isna().all():
            issues.append(f"Timestamp column not parseable: {c}")

    return issues

f timestamp_features(series):
    s = pd.to_datetime(series, errors='coerce')
    return pd.DataFrame({
        series.name + "_year": s.dt.year.fillna(0).astype(int),
        series.name + "_month": s.dt.month.fillna(0).astype(int),
        series.name + "_day": s.dt.day.fillna(0).astype(int),
        series.name + "_dow": s.dt.dayofweek.fillna(-1).astype(int),
        series.name + "_is_null": s.isna().astype(int)
    })

f preprocess(df, numeric_cols, categorical_cols, timestamp_cols):
    issues = validate_features(df, numeric_cols, categorical_cols, timestamp
    ts_dfs = []
    for c in timestamp_cols:
        ts_dfs.append(timestamp_features(df[c]))
    ts_features = pd.concat(ts_dfs, axis=1) if ts_dfs else pd.DataFrame(inde
```

OUTPUT:-



```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\BTECH IIYR\AI\LABEXAM_4.py"
Validation issues:
 - Numeric column has extreme negative values: price

Processed features (head):
    price  product_A  product_B  product_C  product_D  product_None  sold_at_year
    sold_at_month  sold_at_day  sold_at_dow  sold_at_is_null
 0.447214        1.0        0.0        0.0        0.0           0.0
 2021              1            1            4                0
 0.447214        0.0        1.0        0.0        0.0           0.0
 0             0            0           -1                1
 0.447214        0.0        0.0        0.0        0.0           1.0
 2020              6           15            0                0
-2.236068        0.0        0.0        1.0        0.0           0.0
0              0            0           -1                1
 0.447214        1.0        0.0        0.0        0.0           0.0
 2022             12           31            5                0

[Done] exited with code=0 in 1.265 seconds
```

OBSERVATION:-

Script validates numeric/categorical/timestamp columns (presence, types, cardinality, parseability).

Numeric: median imputation + standard scaling. Categorical: missing→'missing' then one-hot (unknowns ignored).

Timestamp: parsed and expanded to year/month/day/dow and null flag.

Returns processed feature DataFrame and list of validation issues.

Q2. AI suggests using AutoML transformation pipeline.

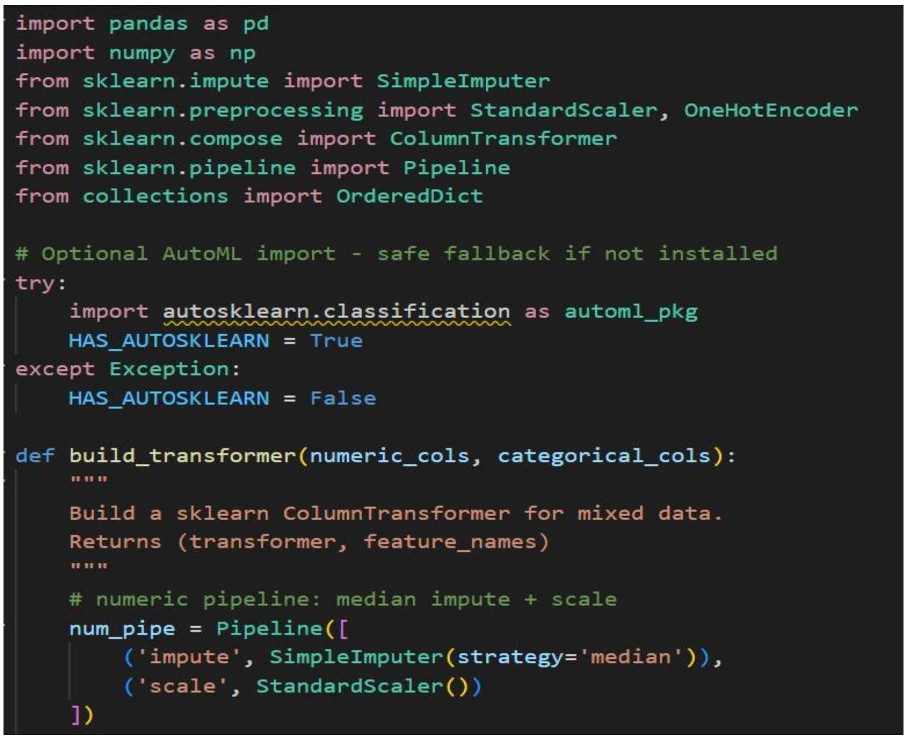
a) Define integration steps.

b) State when custom logic is required.

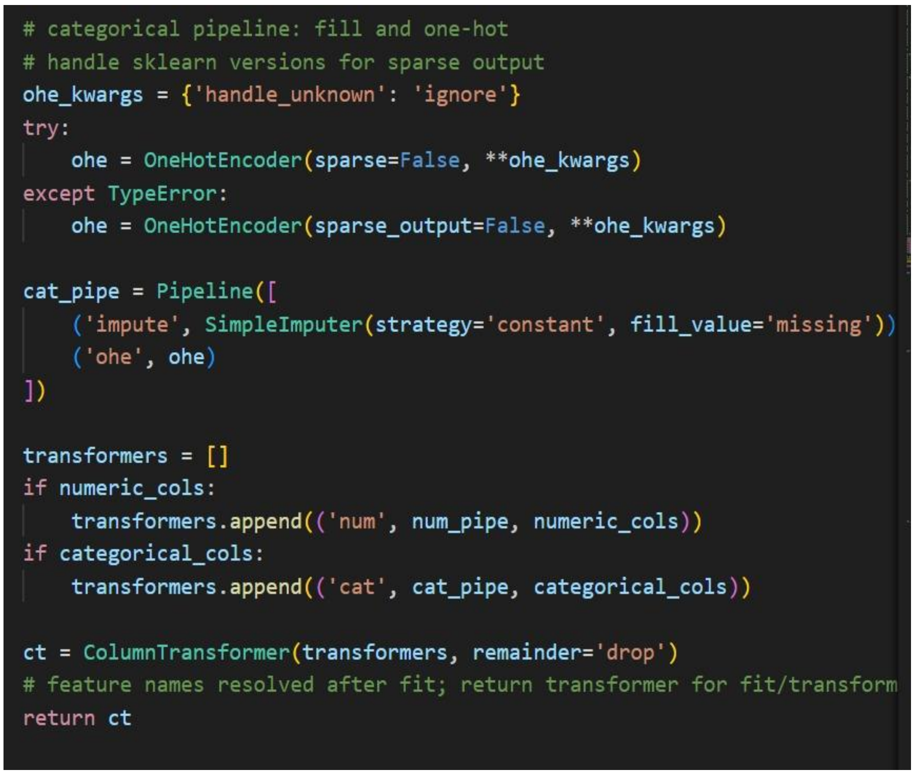
PROMPT:-

"Show AutoML transformation pipeline integration steps for mixed-type data, when to add custom logic, and produce a small Python script that builds a ColumnTransformer pipeline and optionally attempts to wire an AutoML estimator if available."

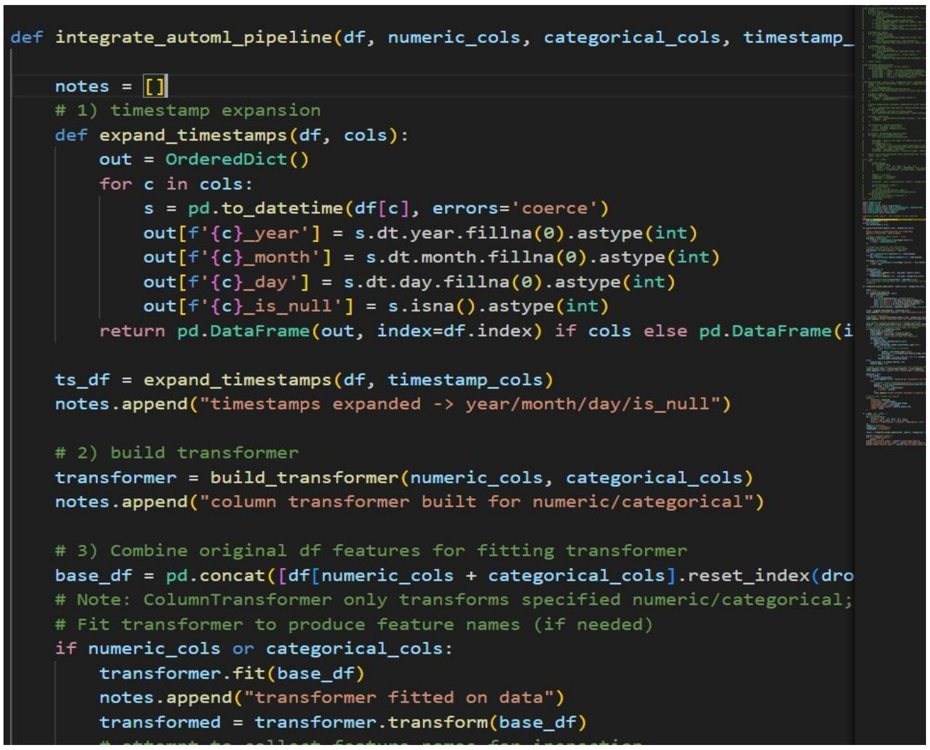
CODE:-



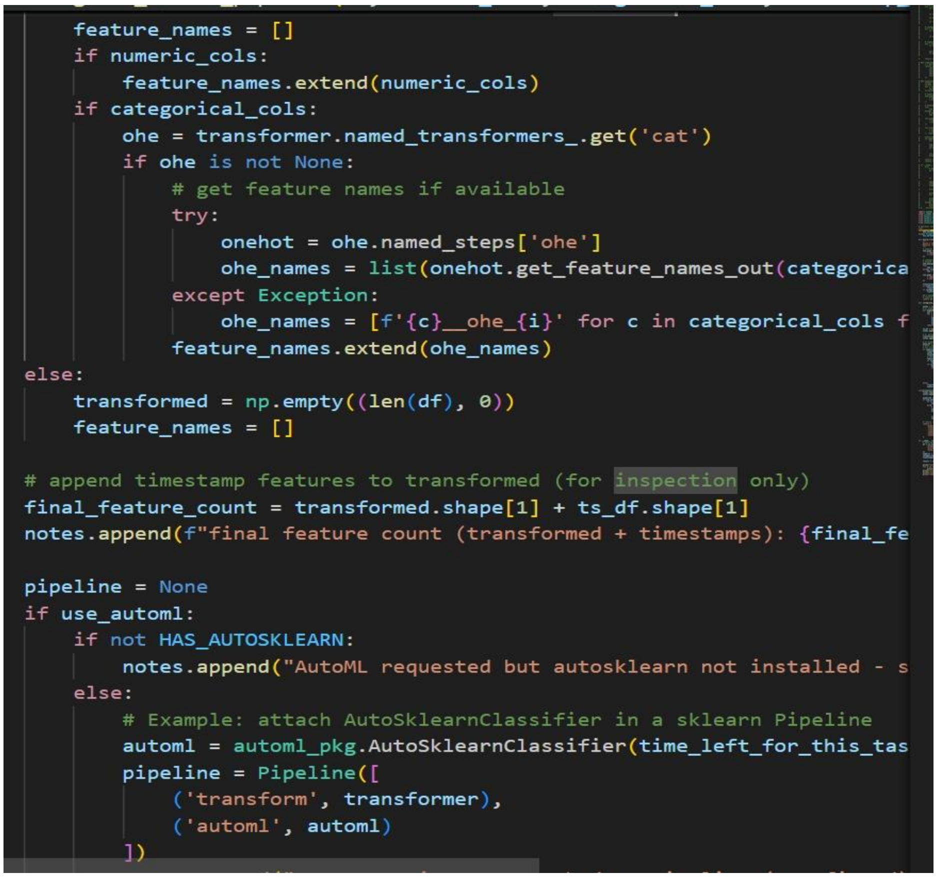
```python
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from collections import OrderedDict

# Optional AutoML import - safe fallback if not installed
try:
    import autosklearn.classification as automl_pkg
    HAS_AUTOSKLEARN = True
except Exception:
    HAS_AUTOSKLEARN = False

def build_transformer(numeric_cols, categorical_cols):
    """
    Build a sklearn ColumnTransformer for mixed data.
    Returns (transformer, feature_names)
    """
    # numeric pipeline: median impute + scale
    num_pipe = Pipeline([
        ('impute', SimpleImputer(strategy='median')),
        ('scale', StandardScaler())
    ])
```



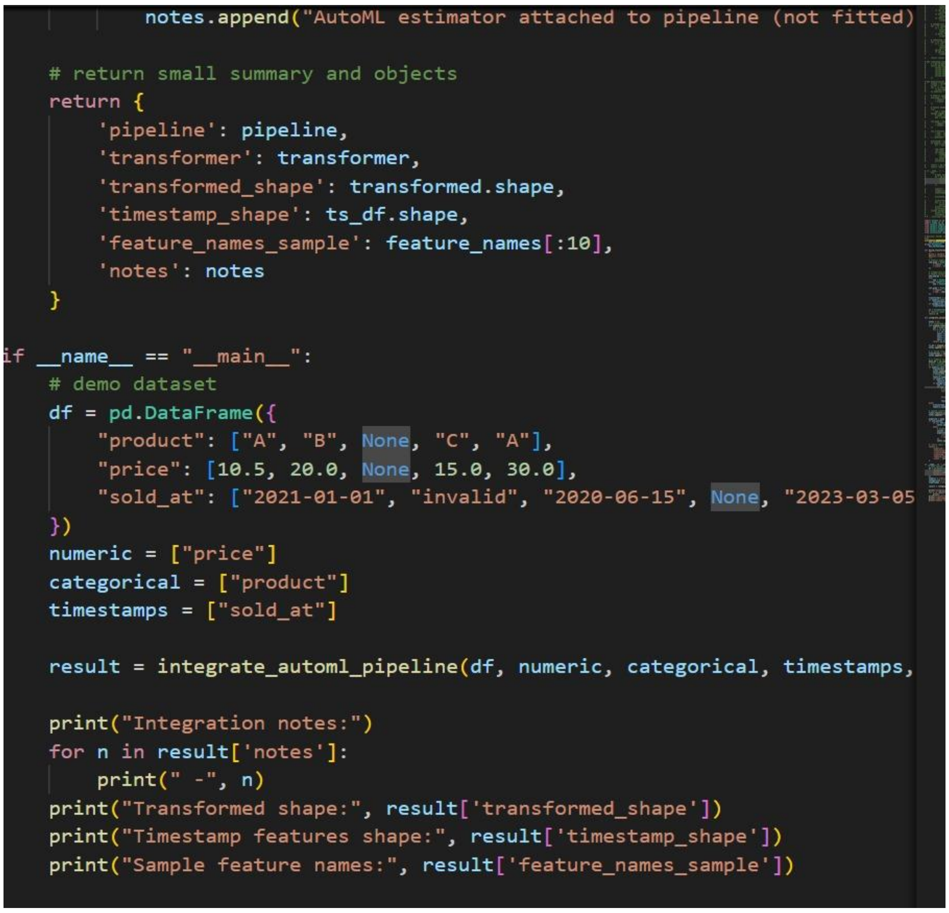
```python
# categorical pipeline: fill and one-hot
# handle sklearn versions for sparse output
ohe_kwargs = {'handle_unknown': 'ignore'}
try:
    ohe = OneHotEncoder(sparse=False, **ohe_kwargs)
except TypeError:
    ohe = OneHotEncoder(sparse_output=False, **ohe_kwargs)

cat_pipe = Pipeline([
    ('impute', SimpleImputer(strategy='constant', fill_value='missing'))
    ('ohe', ohe)
])

transformers = []
if numeric_cols:
    transformers.append(('num', num_pipe, numeric_cols))
if categorical_cols:
    transformers.append(('cat', cat_pipe, categorical_cols))

ct = ColumnTransformer(transformers, remainder='drop')
# feature names resolved after fit; return transformer for fit/transform
return ct
```



```python
def integrate_automl_pipeline(df, numeric_cols, categorical_cols, timestamp_

    notes = []
    # 1) timestamp expansion
    def expand_timestamps(df, cols):
        out = OrderedDict()
        for c in cols:
            s = pd.to_datetime(df[c], errors='coerce')
            out[f'{c}_year'] = s.dt.year.fillna(0).astype(int)
            out[f'{c}_month'] = s.dt.month.fillna(0).astype(int)
            out[f'{c}_day'] = s.dt.day.fillna(0).astype(int)
            out[f'{c}_is_null'] = s.isna().astype(int)
        return pd.DataFrame(out, index=df.index) if cols else pd.DataFrame(i

    ts_df = expand_timestamps(df, timestamp_cols)
    notes.append("timestamps expanded -> year/month/day/is_null")

    # 2) build transformer
    transformer = build_transformer(numeric_cols, categorical_cols)
    notes.append("column transformer built for numeric/categorical")

    # 3) Combine original df features for fitting transformer
    base_df = pd.concat([df[numeric_cols + categorical_cols].reset_index(dro
    # Note: ColumnTransformer only transforms specified numeric/categorical;
    # Fit transformer to produce feature names (if needed)
    if numeric_cols or categorical_cols:
        transformer.fit(base_df)
        notes.append("transformer fitted on data")
        transformed = transformer.transform(base_df)
        # attempt to collect feature names for inspection
```

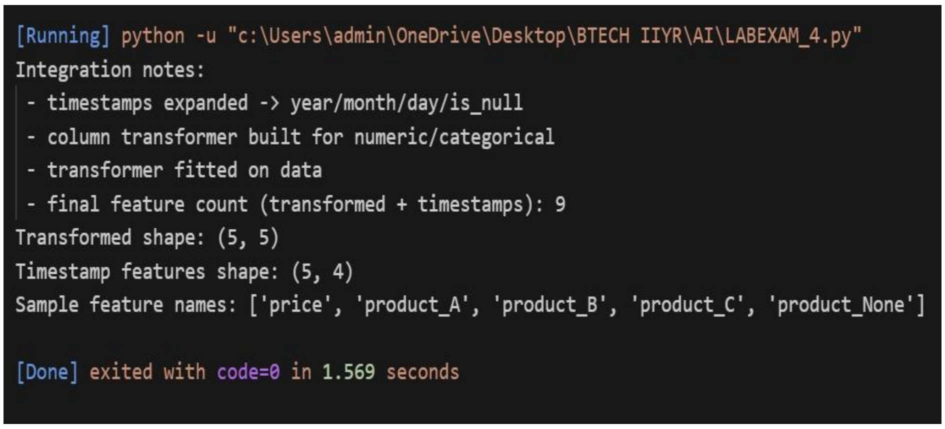


```
        feature_names = []
        if numeric_cols:
            feature_names.extend(numeric_cols)
        if categorical_cols:
            ohe = transformer.named_transformers_.get('cat')
            if ohe is not None:
                # get feature names if available
                try:
                    onehot = ohe.named_steps['ohe']
                    ohe_names = list(onehot.get_feature_names_out(categorica
                except Exception:
                    ohe_names = [f'{c}__ohe_{i}' for c in categorical_cols f
                feature_names.extend(ohe_names)
    else:
        transformed = np.empty((len(df), 0))
        feature_names = []

    # append timestamp features to transformed (for inspection only)
    final_feature_count = transformed.shape[1] + ts_df.shape[1]
    notes.append(f"final feature count (transformed + timestamps): {final_fe

    pipeline = None
    if use_automl:
        if not HAS_AUTOSKLEARN:
            notes.append("AutoML requested but autosklearn not installed - s
        else:
            # Example: attach AutoSklearnClassifier in a sklearn Pipeline
            automl = automl_pkg.AutoSklearnClassifier(time_left_for_this_tas
            pipeline = Pipeline([
                ('transform', transformer),
                ('automl', automl)
            ])
```



```
            notes.append("AutoML estimator attached to pipeline (not fitted)

    # return small summary and objects
    return {
        'pipeline': pipeline,
        'transformer': transformer,
        'transformed_shape': transformed.shape,
        'timestamp_shape': ts_df.shape,
        'feature_names_sample': feature_names[:10],
        'notes': notes
    }

if __name__ == "__main__":
    # demo dataset
    df = pd.DataFrame({
        "product": ["A", "B", None, "C", "A"],
        "price": [10.5, 20.0, None, 15.0, 30.0],
        "sold_at": ["2021-01-01", "invalid", "2020-06-15", None, "2023-03-05
    })
    numeric = ["price"]
    categorical = ["product"]
    timestamps = ["sold_at"]

    result = integrate_automl_pipeline(df, numeric, categorical, timestamps,

    print("Integration notes:")
    for n in result['notes']:
        print(" -", n)
    print("Transformed shape:", result['transformed_shape'])
    print("Timestamp features shape:", result['timestamp_shape'])
    print("Sample feature names:", result['feature_names_sample'])
```

OUTPUT:-



```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\BTECH IIYR\AI\LABEXAM_4.py"
Integration notes:
 - timestamps expanded -> year/month/day/is_null
 - column transformer built for numeric/categorical
 - transformer fitted on data
 - final feature count (transformed + timestamps): 9
Transformed shape: (5, 5)
Timestamp features shape: (5, 4)
Sample feature names: ['price', 'product_A', 'product_B', 'product_C', 'product_None']

[Done] exited with code=0 in 1.569 seconds
```

OBSERVATION:-

Integration steps: expand timestamps, build ColumnTransformer for numeric/categorical, fit transformer, optionally attach AutoML estimator, export pipeline for inference.

Use AutoML when you want end-to-end model search; require labeled data and compute budget.

Custom logic required for: high-cardinality categorical encoding, domain-specific timestamp features, target leakage prevention, advanced imputation, and business-rule validation.