model impotion

Medical Data Grouping for Smart Imputation

Feature Map per Group

Group	Target Columns	Also Train On
1. Body Metrics	AGE_YEARS, HEIGHT, WEIGHT, BMI	GENDER_NAME, CITY_NAME, COLLECTYEAR
2. Lipid Profile	LDL, HDL, Triglycerides, Cholesterol	AGE_YEARS, GENDER_NAME, BMI, Diabetes, CITY_NA
3. Blood Panel	RBC, Hemoglobin, Platelets, WBC, RDW	AGE_YEARS, GENDER_NAME, Iron, Vitamin B12, CRP
4. Liver Function	ALT, AST, Bilirubin, Albumin, ALP	AGE_YEARS, GENDER_NAME, BMI, Medication, Alcoho
5. Renal Function	Creatinine, eGFR, Urea, Potassium	AGE_YEARS, GENDER_NAME, Blood Pressure, Diabete
6. Thyroid Hormones	TSH, Free T4, FT3	GENDER_NAME, AGE_YEARS, Pregnancy, Family Histo
7. Glucose & HbA1c	Fasting Glucose, HbA1c	AGE_YEARS, BMI, GENDER_NAME, CITY_NAME, Famil
8. Urinalysis	pH, Protein, RBCs/HPF, Crystals	GENDER_NAME, AGE_YEARS, eGFR, Creatinine, UTI his
9. Vitamins & Minerals	Vitamin D, Iron, Ferritin, Calcium	AGE_YEARS, GENDER_NAME, CITY_NAME, Sun exposu
10. Inflammation Markers	CRP, ESR, Rheumatoid Factor, Anti-CCP	AGE_YEARS, GENDER_NAME, Autoimmune flag, Infect

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np
import os
import warnings
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
import plotly.express as px
from tqdm import tqdm
warnings.filterwarnings("ignore")
```

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', '{:.2f}'.format)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df=pd.read_csv("/content/drive/MyDrive/XXXX/filled_final_by_rid_batches.csv")
```

```
df.shape
```

```
(900000, 104)
```

```
df.isnull().sum().sum()
```

```
np.int64(55841104)
```

```
df.isnull().sum()
```

	0
Unnamed: 0	0
RESEARCH_ID	0
SAMPLE_ID	0
COLLECTYEAR	0
REGN_DATE	0
...	...
Systolic Pressure	381332
Diastolic Pressure	381332
Amorphous_Numeric	897127
Bilirubin_Numeric	897127
T. Cholesterol/HDL_Numeric	898838

104 rows × 1 columns

dtype: int64

```
df.columns
```

```
Index(['Unnamed: 0', 'RESEARCH_ID', 'SAMPLE_ID', 'COLLECTYEAR',
'REGN_DATE',
       'AGE_YEARS', 'AGE_DAYS', 'AGE_MONTHS', 'HEIGHT', 'WEIGHT',
       ...
       'RESEARCH_ID_int', 'GENDER_BINARY', 'CITY_NAME_ENCODED', 'BDL',
       'Florescence Pattern ', 'Systolic Pressure', 'Diastolic Pressure',
       'Amorphous_Numeric', 'Bilirubin_Numeric', 'T.
Cholesterol/HDL_Numeric'],
      dtype='object', length=104)
```

```
df.dtypes
```

	0
Unnamed: 0	int64
RESEARCH_ID	object
SAMPLE_ID	object
COLLECTYEAR	int64
REGN_DATE	object
AGE_YEARS	float64
AGE_DAYS	int64
AGE_MONTHS	int64
HEIGHT	int64
WEIGHT	float64
BMI	float64
Thyroid Stimulating Hormone (TSH)	float64
Uric Acid in Serum	float64
Alanine Aminotransferase (ALT)	float64
Ferritin In Serum	float64
Blood Urea Nitrogen (BUN)	float64
Lymphocytes absolute count	float64
R. B. Cs / HPFs	float64
Aspect(Urine Physical Examination) Ordinal Encoding	float64
Eosinophils absolute count	float64

Vitamin D (25 OH-Vit D -Total)	float64
C-Reactive Protein (CRP) quantitative	float64
Transferrin	float64
Red cell count	float64
Basophils absolute count	float64
Crystals(Urine Microscopic Examination :)	float64
Protein(Urine Physical Examination)	float64
Colour(Urine Physical Examination)	float64
Nitrite	float64
LDL Cholesterol	float64
LDL / HDL	float64
24 Hour Urine Volume (263)	float64
Hemoglobin	float64
Total Leucocytic Count	float64
Hematocrit	float64
MCV	float64
Glucose(Urine Physical Examination)	float64
Urea in Serum	float64
Prostatic Specific Antigen (PSA) Total	float64
Testosterone (Total)	float64
Alkaline Phosphatase	float64
Total Protein in Serum	float64
Estimated Glomerular Filtration Rate(eGFR)	float64
Anti CCP Abs	float64
BUN/Creatinine Ratio	float64
Ketones	float64
MCHC	float64
pH(Urine Physical Examination)	float64
Amorphous Elements	float64
Blood and Haemoglobin	float64

Blood and Haemoglobin	float64
Epithelial Cells / HPF	float64
Casts(Urine Microscopic Examination :)	float64
Chloride in Serum	float64
Cholesterol	float64
T. Cholesterol/HDL	float64
Urobilinogen	float64
R.B.Cs / HPF	float64
Erythrocyte Sedimentation Rate(ESR)	float64
Glucose in Plasma (Fasting)	float64
Hb A1c %	float64
Mean of blood glucose	float64
Microalbuminuria (24 h urine)	float64
Bilirubin (Total)	float64
Florescence Pattern	int64
Lead in blood	float64
Monocytes absolute count	float64
Consistancy	float64
Neutrophils absolute count	float64
Specific Gravity	float64
W. B. Cs / HPF	float64
Aspartate Aminotransferase (AST)	float64
Calcium in Serum (Total)	float64
Free T4	float64
Potassium (K) in Serum	float64
Albumin in Serum	float64
Iron (Fe) in Serum	float64
CRP H.S	float64
Triglycerides (TG) in Serum	float64
Rheumatoid Factor (quantitative)	float64

Platelet Count	float64
Albumin in Urine (263)	float64
MCH	float64
RDW	float64
W.B.Cs / HPF	float64
Leucocyte esterase	float64
Concentration	float64
Creatinine in Serum	float64
Sodium (Na) in Serum	float64
Bilirubin (Direct)	float64
Magnesium (Mg) in Serum	float64
Titre on Hep2 cells	float64
HDL Cholesterol	float64
Globulin in Serum	float64
Cystatin C	float64
RESEARCH_ID_int	int64
GENDER_BINARY	int64
CITY_NAME_ENCODED	int64
BDL	int64
Florescence Pattern	int64
Systolic Pressure	float64
Diastolic Pressure	float64
Amorphous_Numeric	float64
Bilirubin_Numeric	float64
T. Cholesterol/HDL_Numeric	float64

dtype: object

AI Imputation for Body Metrics

```
df.isnull().sum().sum()
```

```
df[["AGE_YEARS", "AGE_MONTHS","AGE_DAYS", "HEIGHT", "WEIGHT", "BMI"]].isnull().
```

	0
AGE_YEARS	10089
AGE_MONTHS	0
AGE_DAYS	0
HEIGHT	0
WEIGHT	0
BMI	0

dtype: int64

```
df[["AGE_YEARS", "AGE_MONTHS","AGE_DAYS", "HEIGHT", "WEIGHT", "BMI"]].isnull().
```

```
np.int64(10089)
```

```
#df_original_long = df[["AGE_YEARS", "AGE_MONTHS", "HEIGHT", "WEIGHT", "BMI"]].
#df_imputed_long = df[["AGE_YEARS", "AGE_MONTHS", "HEIGHT", "WEIGHT", "BMI"]].m
#df_original_long["Imputation Status"] = "Before Imputation"
#df_imputed_long["Imputation Status"] = "After Imputation"
#df_combined = pd.concat([df_original_long, df_imputed_long])
#fig = px.box(df_combined, x="Column", y="Value", color="Imputation Status",
#             title="Comparison of Missing Values Before and After Imputation",
#fig.show()
```

```
df_max=pd.read_csv("/content/drive/My Drive/FEasmaa/final_electrolytes_BP_grour
```

```
zero_as_nan_cols = ["AGE_YEARS", "AGE_MONTHS", "AGE_DAYS", "HEIGHT", "WEIGHT",
df3[zero_as_nan_cols] = df3[zero_as_nan_cols].replace(0, np.nan)
```

```
target_cols = ["AGE_YEARS", "AGE_MONTHS", "AGE_DAYS", "HEIGHT", "WEIGHT", "BMI'


supporting_features = ["GENDER_NAME", "CITY_NAME", "COLLECTYEAR"]


available_features = [col for col in supporting_features if col in df.columns]
```

```
all_cols = target_cols + available_features
sub_df = df[all_cols].copy()


label_encoders = {}
for col in sub_df.select_dtypes(include="object").columns:
    le = LabelEncoder()
    sub_df[col] = le.fit_transform(sub_df[col].astype(str))
    label_encoders[col] = le


output_folder = "/content/drive/MyDrive/FEasmaa/final_1"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/body_metrics_imputed.csv"


total_filled = 0
best_models_log = {}


for target in tqdm(target_cols, desc=" Imputing Progress", ncols=80):
    if sub_df[target].isna().sum() == 0:
        continue


    train_data = sub_df[sub_df[target].notna()]
    test_data = sub_df[sub_df[target].isna()]

    if len(train_data) < 50:
        continue

    X_train = train_data.drop(columns=[target])
    y_train = train_data[target]
    X_test = test_data.drop(columns=[target])


    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_me
    xgb.fit(X_train, y_train)


    predictions = xgb.predict(X_test)


    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)
    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)


print("\n AI Imputation Completed")
print(f" Total filled values: {total_filled}")
print(f" File saved at: {output_path}")
print(" Model Summary:")
for col, info in best_models_log.items():
    print(f" - {col}: {info['model']} (Filled: {info['filled']}, MSE: {info['ms
```

Show hidden output

```
df1=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups_Body Metrics/body_metric
```

```
df1[["AGE_YEARS", "AGE_MONTHS", "HEIGHT", "WEIGHT", "BMI"]].isnull().sum()
```

	0
AGE_YEARS	0
AGE_MONTHS	0
HEIGHT	0
WEIGHT	0
BMI	0

dtype: int64

```
df1[["AGE_YEARS", "AGE_MONTHS", "HEIGHT", "WEIGHT", "BMI"]].isnull().sum().sum(
```

```
np.int64(0)
```

```
df1.shape
```

```
(900000, 104)
```

```
df1.isnull().sum().sum()
```

```
np.int64(55831015)
```

```
df_original_long = df1[["AGE_YEARS", "AGE_MONTHS", "HEIGHT", "WEIGHT", "BMI"]].
df_imputed_long = df1[["AGE_YEARS", "AGE_MONTHS", "HEIGHT", "WEIGHT", "BMI"]].m
df_original_long["Imputation Status"] = "Before Imputation"
df_imputed_long["Imputation Status"] = "After Imputation"
df_combined = pd.concat([df_original_long, df_imputed_long])
fig = px.box(df_combined, x="Column", y="Value", color="Imputation Status",
             title="Comparison of Missing Values Before and After Imputation",
fig.show()
```

AI Imputation for Lipid Profile

```
df1[[
    "LDL Cholesterol",
    "HDL Cholesterol",
    "T. Cholesterol/HDL",
    "LDL / HDL",
    "Cholesterol",
    "Triglycerides (TG) in Serum"
]].isnull().sum()
```

	0
LDL Cholesterol	368057
HDL Cholesterol	369199
T. Cholesterol/HDL	898838
LDL / HDL	898835
Cholesterol	311826
Triglycerides (TG) in Serum	317623

dtype: int64

```
df1[[
    "LDL Cholesterol",
    "HDL Cholesterol",
    "T. Cholesterol/HDL",
    "LDL / HDL",
    "Cholesterol",
    "Triglycerides (TG) in Serum"
]].isnull().sum().sum()
```

```
np.int64(3164378)
```

```
df=df1
target_cols = [
    "LDL Cholesterol",
    "HDL Cholesterol",
    "T. Cholesterol/HDL",
    "LDL / HDL",
    "Cholesterol",
    "Triglycerides (TG) in Serum"
]


supporting_features = ["AGE_YEARS", "GENDER_NAME", "CITY_NAME", "WEIGHT", "HEI(
available_features = [col for col in supporting_features if col in df.columns]
all_cols = target_cols + available_features
sub_df = df[all_cols].copy()


label_encoders = {}
for col in sub_df.select_dtypes(include="object").columns:
    le = LabelEncoder()
    sub_df[col] = le.fit_transform(sub_df[col].astype(str))
    label_encoders[col] = le


output_folder = "/content/drive/MyDrive/XXXX/ai_groups_Lipid Profile"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/lipid_profile_imputed.csv"


total_filled = 0
best_models_log = {}

for target in tqdm(target_cols, desc=" Imputing Lipid Profile", ncols=80):
    if sub_df[target].isna().sum() == 0:
        continue
    train_data = sub_df[sub_df[target].notna()]
    test_data = sub_df[sub_df[target].isna()]

    if len(train_data) < 50:
        continue

    X_train = train_data.drop(columns=[target])
    y_train = train_data[target]
    X_test = test_data.drop(columns=[target])

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_me
    xgb.fit(X_train, y_train)
    predictions = xgb.predict(X_test)

    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)

    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)



print("\n AI Imputation for Lipid Profile Completed")
print(f" Total filled values: {total_filled}")
print(f"File saved at: {output_path}")
print(" Model Summary:")
for col, info in best_models_log.items():
    print(f" - {col}: {info['model']} (Filled: {info['filled']}, MSE: {info['ms
```

Show hidden output

```
df2=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups_Lipid Profile/lipid_prof
df2[[
    "LDL Cholesterol",
    "HDL Cholesterol",
    "T. Cholesterol/HDL",
    "LDL / HDL",
    "Cholesterol",
    "Triglycerides (TG) in Serum"
]].isnull().sum()
```

	0
LDL Cholesterol	0
HDL Cholesterol	0
T. Cholesterol/HDL	0
LDL / HDL	0
Cholesterol	0
Triglycerides (TG) in Serum	0

dtype: int64

```
df2[[
    "LDL Cholesterol",
    "HDL Cholesterol",
    "T. Cholesterol/HDL",
    "LDL / HDL",
    "Cholesterol",
    "Triglycerides (TG) in Serum"
]].isnull().sum().sum()
```

```
np.int64(0)
```

```
df2.shape
```

```
(900000, 104)
```

```
df.isnull().sum().sum()
```

```
np.int64(52666637)
```

AI Imputation for Kidney Function

```
df2.columns = df2.columns.str.strip()
```

```
df2[[
    "Creatinine in Serum",
    "Urea in Serum",
    "Estimated Glomerular Filtration Rate(eGFR)",
    "Blood Urea Nitrogen (BUN)",
    "BUN/Creatinine Ratio"
]].isnull().sum()
```

	0
Creatinine in Serum	47889
Urea in Serum	492425
Estimated Glomerular Filtration Rate(eGFR)	373001
Blood Urea Nitrogen (BUN)	289485
BUN/Creatinine Ratio	405587

dtype: int64

```
df2[[
    "Creatinine in Serum",
    "Urea in Serum",
    "Estimated Glomerular Filtration Rate(eGFR)",
    "Blood Urea Nitrogen (BUN)",
    "BUN/Creatinine Ratio"
]].isnull().sum().sum()
```

```
np.int64(1608387)
```

```
df = df2
df.columns = df.columns.str.strip()
df_backup = df.copy()
```

```
target_cols = [
    "Creatinine in Serum", "Urea in Serum",
    "Estimated Glomerular Filtration Rate(eGFR)",
    "Blood Urea Nitrogen (BUN)", "BUN/Creatinine Ratio"
]
supporting_features = ["AGE_YEARS", "GENDER_NAME", "CITY_NAME", "WEIGHT", "HEIG
available_features = [col for col in supporting_features if col in df.columns]
all_cols = [col for col in (target_cols + available_features) if col in df.colu


missing_targets = [col for col in target_cols if col not in df.columns]
if missing_targets:




sub_df = df[all_cols].copy()
label_encoders = {}
for col in sub_df.select_dtypes(include="object").columns:
    le = LabelEncoder()
    sub_df[col] = le.fit_transform(sub_df[col].astype(str))
    label_encoders[col] = le


output_folder = "/content/drive/MyDrive/XXXX/ai_groups_ Kidney Function"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/kidney_function_imputed.csv"


total_filled = 0
best_models_log = {}

for target in tqdm(target_cols, desc=" Imputing Kidney Function", ncols=80):
    if target not in sub_df.columns or sub_df[target].isna().sum() == 0:
        continue

    train_data = sub_df[sub_df[target].notna()]
    test_data = sub_df[sub_df[target].isna()]
    if len(train_data) < 50:

        continue

    X_test = test_data.drop(columns=[target])


    train_combined = pd.concat([train_data.drop(columns=[target]), train_data[t
    train_combined.replace([np.inf, -np.inf], np.nan, inplace=True)
    train_combined.dropna(inplace=True)
    X_train = train_combined.drop(columns=[target])
    y_train = train_combined[target]


    y_train = y_train[(~y_train.isna()) & (~np.isinf(y_train)) & (np.abs(y_trai
    X_train = X_train.loc[y_train.index]


    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    X_train.dropna(inplace=True)
    X_train = X_train[(np.abs(X_train) < 1e10).all(axis=1)]
    y_train = y_train.loc[X_train.index]

    if X_train.empty or y_train.empty or y_train.isna().any():
        continue

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_me
    xgb.fit(X_train, y_train)
    predictions = xgb.predict(X_test)

    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)
    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)


print("AI Imputation for Kidney Function Completed")
print(f" Total filled values: {total_filled}")
print(f" File saved at: {output_path}")
print(" Model Summary:")
for col, info in best_models_log.items():
    print(f" - {col}: {info['model']} (Filled: {info['filled']}, MSE: {info['ms
```

Show hidden output

```
df3=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups_ Kidney Function/kidney_
df3[[
    "Creatinine in Serum",
    "Urea in Serum",
    "Estimated Glomerular Filtration Rate(eGFR)",
    "Blood Urea Nitrogen (BUN)",
    "BUN/Creatinine Ratio"
]].isnull().sum()
```

	0
Creatinine in Serum	0
Urea in Serum	0
Estimated Glomerular Filtration Rate(eGFR)	0
Blood Urea Nitrogen (BUN)	0
BUN/Creatinine Ratio	0

dtype: int64

```
df3.shape
```

```
(900000, 104)
```

```
df3.isnull().sum().sum()
```

```
np.int64(51058250)
```

```
df3[[
    "Creatinine in Serum",
    "Urea in Serum",
    "Estimated Glomerular Filtration Rate(eGFR)",
    "Blood Urea Nitrogen (BUN)",
    "BUN/Creatinine Ratio"
]].isnull().sum().sum()
```

```
np.int64(0)
```

Liver Function

```
df3[[

    'Alkaline Phosphatase',
    'Bilirubin (Total)',
    'Bilirubin (Direct)'
]].isnull().sum()
```

	0
Alkaline Phosphatase	386750
Bilirubin (Total)	345572
Bilirubin (Direct)	346604

dtype: int64

```
df3[[

    'Alkaline Phosphatase',
    'Bilirubin (Total)',
    'Bilirubin (Direct)'
]].isnull().sum().sum()
```

```
np.int64(1078926)
```

```
df = df3
df.columns = df.columns.str.strip()
df_backup = df.copy()


target_cols = [
    'Alkaline Phosphatase',
    'Bilirubin (Total)',
    'Bilirubin (Direct)'
]


supporting_features = ["AGE_YEARS", "GENDER_NAME", "CITY_NAME", "HEIGHT", "WEI(
available_features = [col for col in supporting_features if col in df.columns]
all_cols = [col for col in (target_cols + available_features) if col in df.colu


sub_df = df[all_cols].copy()
label_encoders = {}
for col in sub_df.select_dtypes(include="object").columns:
    le = LabelEncoder()
    sub_df[col] = le.fit_transform(sub_df[col].astype(str))
    label_encoders[col] = le


output_folder = "/content/drive/MyDrive/XXXX/ai_groups_Liver Function"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/liver_function_imputed.csv"

total_filled = 0
best_models_log = {}

for target in tqdm(target_cols, desc=" Imputing Liver Function", ncols=80):
    if sub_df[target].isna().sum() == 0:
        continue

    train_data = sub_df[sub_df[target].notna()]
    test_data = sub_df[sub_df[target].isna()]
    if len(train_data) < 50:
        continue

    X_test = test_data.drop(columns=[target])
    train_combined = pd.concat([train_data.drop(columns=[target]), train_data[t
    train_combined.replace([np.inf, -np.inf], np.nan, inplace=True)
    train_combined.dropna(inplace=True)

    X_train = train_combined.drop(columns=[target])
    y_train = train_combined[target]


    y_train = y_train[(~y_train.isna()) & (~np.isinf(y_train)) & (np.abs(y_trai
    X_train = X_train.loc[y_train.index]


    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    X_train.dropna(inplace=True)
    X_train = X_train[(np.abs(X_train) < 1e10).all(axis=1)]
    y_train = y_train.loc[X_train.index]

    if X_train.empty or y_train.empty:
        continue

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_me
    xgb.fit(X_train, y_train)
    predictions = xgb.predict(X_test)

    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)
    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)

print("\n AI Imputation for Liver Function Completed")
print(f" Total filled values: {total_filled}")
print(f" File saved at: {output_path}")
print(" Model Summary:")
for col, info in best_models_log.items():
    print(f" - {col}: {info['model']} (Filled: {info['filled']}, MSE: {info['ms
```

Show hidden output

```
df4=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups_Liver Function/liver_fur
```

```
df4[[

    "Alkaline Phosphatase",
    "Bilirubin (Total)",
    "Bilirubin (Direct)"
]].isnull().sum()
```

	0
Alkaline Phosphatase	0
Bilirubin (Total)	0
Bilirubin (Direct)	0

dtype: int64

```
df4[[

    "Alkaline Phosphatase",
    "Bilirubin (Total)",
    "Bilirubin (Direct)"
]].isnull().sum().sum()
```

```
np.int64(0)
```

Blood Profile

```
df4[[
    "Hemoglobin",
    "Platelet Count",
    "MCV",
    "MCHC",
    "RDW"
]].isnull().sum()
```

	0
Hemoglobin	896934
Platelet Count	899706
MCV	896940
MCHC	896940
RDW	896990

dtype: int64

```
df = df4
df.columns = df.columns.str.strip()
df_backup = df.copy()


target_cols = [
    "Hemoglobin",
    "Platelet Count",
    "MCV",
    "MCHC",
    "RDW"
]
```

```
supporting_features = ["AGE_YEARS", "GENDER_NAME", "CITY_NAME", "HEIGHT", "WEIG
available_features = [col for col in supporting_features if col in df.columns]
all_cols = [col for col in (target_cols + available_features) if col in df.colu


label_encoders = {}
for col in df.select_dtypes(include="object").columns:
    if col in all_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))
        label_encoders[col] = le


output_folder = "/content/drive/MyDrive/XXXX/ai_groups_Blood Profile"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/blood_profile_imputed.csv"

total_filled = 0
best_models_log = {}


for target in tqdm(target_cols, desc=" Imputing Blood Profile", ncols=80):
    if df[target].isna().sum() == 0:
        continue

    train_data = df[df[target].notna()]
    test_data = df[df[target].isna()]
    if len(train_data) < 50:

        continue

    X_test = test_data[available_features]
    train_combined = pd.concat([train_data[available_features], train_data[targ
    train_combined.replace([np.inf, -np.inf], np.nan, inplace=True)
    train_combined.dropna(inplace=True)

    X_train = train_combined[available_features]
    y_train = train_combined[target]

    y_train = y_train[(~y_train.isna()) & (~np.isinf(y_train)) & (np.abs(y_trai
    X_train = X_train.loc[y_train.index]


    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    X_train.dropna(inplace=True)
    X_train = X_train[(np.abs(X_train) < 1e10).all(axis=1)]
    y_train = y_train.loc[X_train.index]

    if X_train.empty or y_train.empty:

        continue

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_me
    xgb.fit(X_train, y_train)
    predictions = xgb.predict(X_test)

    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)
    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)



print("\n AI Imputation for Blood Profile Completed")
print(f" Total filled values: {total_filled}")
print(f" File saved at: {output_path}")
print(" Model Summary:")
for col, info in best_models_log.items():
    print(f" - {col}: {info['model']} (Filled: {info['filled']}, MSE: {info['ms
```

Show hidden output

```
df5=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups_Blood Profile/blood_prof
```

```
df5.shape
```

```
(900000, 104)
```

```
df5.isnull().sum().sum()
```

```
np.int64(45491814)
```

```
df5[[
    "Hemoglobin",
    "Platelet Count",
    "MCV",
    "MCHC",
    "RDW"
]].isnull().sum()
```

	0
Hemoglobin	0
Platelet Count	0
MCV	0
MCHC	0
RDW	0

dtype: int64

```
df5 [[
    "Hemoglobin",
    "Platelet Count",
    "MCV",
    "MCHC",
    "RDW"
]].isnull().sum().sum()
```

```
np.int64(0)
```

Glucose Control

```
df5[[
    'Hb A1c %',
    "Glucose in Plasma (Fasting)",
    "Mean of blood glucose",

]].isnull().sum()
```

```
                                  0
           Hb A1c %          311653
Glucose in Plasma (Fasting)  131872
     Mean of blood glucose   312316

dtype: int64
```

```
df5[[

    'Hb A1c %',
    "Glucose in Plasma (Fasting)",
    "Mean of blood glucose",

]].isnull().sum().sum()
```

```
np.int64(755841)
```

```
df = df5
df.columns = df.columns.str.strip()
df_backup = df.copy()



target_cols = [
    'Hb A1c %',
    "Glucose in Plasma (Fasting)",
    "Mean of blood glucose",

]



supporting_features = ["AGE_YEARS", "GENDER_NAME", "CITY_NAME", "HEIGHT", "WEIGH
available_features = [col for col in supporting_features if col in df.columns]
all_cols = [col for col in (target_cols + available_features) if col in df.colum



label_encoders = {}
```

```
for col in df.select_dtypes(include="object").columns:
    if col in all_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))
        label_encoders[col] = le


output_folder = "/content/drive/MyDrive/XXXX/ai_groups"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/glucose_control_imputed.csv"

total_filled = 0
best_models_log = {}


for target in tqdm(target_cols, desc=" Imputing Glucose Control", ncols=80):
    if df[target].isna().sum() == 0:
        continue

    train_data = df[df[target].notna()]
    test_data = df[df[target].isna()]
    if len(train_data) < 50:

        continue

    X_test = test_data[available_features]
    train_combined = pd.concat([train_data[available_features], train_data[targe
    train_combined.replace([np.inf, -np.inf], np.nan, inplace=True)
    train_combined.dropna(inplace=True)

    X_train = train_combined[available_features]
    y_train = train_combined[target]


    y_train = y_train[(~y_train.isna()) & (~np.isinf(y_train)) & (np.abs(y_train
    X_train = X_train.loc[y_train.index]


    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    X_train.dropna(inplace=True)
    X_train = X_train[(np.abs(X_train) < 1e10).all(axis=1)]
    y_train = y_train.loc[X_train.index]

    if X_train.empty or y_train.empty:
        continue

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_mea
    xgb.fit(X_train, y_train)
    xgb.fit(X_train, y_train)
    predictions = xgb.predict(X_test)

    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)
    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)

print("AI Imputation for Glucose Control Completed")
print(f"Total filled values: {total_filled}")
print(f" File saved at: {output_path}")
print("Model Summary:")
for col, info in best_models_log.items():
    print(f" - {col}: {info['model']} (Filled: {info['filled']}, MSE: {info['mse
```

Show hidden output

```
df6=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups/glucose_control_imputed.
```

```
df6[[
   'Hb A1c %',
    "Glucose in Plasma (Fasting)",
    "Mean of blood glucose",

]].isnull().sum()
```

	0
Hb A1c %	0
Glucose in Plasma (Fasting)	0
Mean of blood glucose	0

dtype: int64

```
df6[[
    'Hb A1c %',
    "Glucose in Plasma (Fasting)",
    "Mean of blood glucose",

]].isnull().sum().sum()
```

```
np.int64(0)
```

```
df6.shape
```

```
(900000, 104)
```

```
df6.isnull().sum().sum()
```

```
np.int64(44735973)
```

```
df6.duplicated().sum().sum()
```

```
np.int64(0)
```

```
df6[[
    "Alanine Aminotransferase (ALT)",
    "Aspartate Aminotransferase (AST)",
    "Alkaline Phosphatase",
    "Bilirubin (Total)",
    "Bilirubin (Direct)"
]].isnull().sum()
```

	0
Alanine Aminotransferase (ALT)	63483
Aspartate Aminotransferase (AST)	67206
Alkaline Phosphatase	0
Bilirubin (Total)	0
Bilirubin (Direct)	0

dtype: int64

Liver Function 2

```
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder


df = df6.copy()
df.columns = df.columns.str.strip()
df_backup = df.copy()

target_cols = [
    "Alanine Aminotransferase (ALT)",
    "Aspartate Aminotransferase (AST)",
    "Alkaline Phosphatase",
    "Bilirubin (Total)",
    "Bilirubin (Direct)"
]

supporting_features = [
    "AGE_YEARS", "GENDER_BINARY", "CITY_NAME_ENCODED", "HEIGHT", "WEIGHT", "BMI
    "Systolic Pressure", "Diastolic Pressure"
]

available_features = [col for col in supporting_features if col in df.columns]
all_cols = [col for col in (target_cols + available_features) if col in df.colu


label_encoders = {}
for col in df.select_dtypes(include="object").columns:
    if col in all_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))
        label_encoders[col] = le


output_folder = "/content/drive/MyDrive/XXXX/ai_groups_Liver Function_22222"
os.makedirs(output_folder, exist_ok=True)
output_path = f"{output_folder}/liver_function_imputed.csv"
```

```
total_filled = 0
best_models_log = {}

for target in tqdm(target_cols, desc="Imputing Liver Function", ncols=80):
    if df[target].isna().sum() == 0:
        continue

    train_data = df[df[target].notna()]
    test_data = df[df[target].isna()]
    if len(train_data) < 50:
        continue

    X_test = test_data[available_features]
    train_combined = pd.concat([train_data[available_features], train_data[targ
    train_combined.replace([np.inf, -np.inf], np.nan, inplace=True)
    train_combined.dropna(inplace=True)

    X_train = train_combined[available_features]
    y_train = train_combined[target]

    y_train = y_train[(~y_train.isna()) & (~np.isinf(y_train)) & (np.abs(y_trai
    X_train = X_train.loc[y_train.index]

    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    X_train.dropna(inplace=True)
    X_train = X_train[(np.abs(X_train) < 1e10).all(axis=1)]
    y_train = y_train.loc[X_train.index]

    if X_train.empty or y_train.empty:
        continue

    xgb = XGBRegressor(n_estimators=100, random_state=42)
    xgb_score = -np.mean(cross_val_score(xgb, X_train, y_train, scoring="neg_me
    xgb.fit(X_train, y_train)
    predictions = xgb.predict(X_test)

    df.loc[X_test.index, target] = predictions
    total_filled += len(predictions)
    best_models_log[target] = {
        "model": "XGBoost",
        "filled": len(predictions),
        "mse": round(xgb_score, 4)
    }


df.to_csv(output_path, index=False)
```

```
summary_log = {
    "Total Filled": total_filled,
    "Saved To": output_path,
    "Columns Imputed": best_models_log
}
summary_log
```

```
🔄 Imputing Liver Function: 100%|████████████████| 5/5 [00:07<00:00,  1.57
{'Total Filled': 130689,
 'Saved To': '/content/drive/MyDrive/XXXX/ai_groups_Liver
Function_22222/liver_function_imputed.csv',
 'Columns Imputed': {'Alanine Aminotransferase (ALT)': {'model':
'XGBoost',
   'filled': 63483,
   'mse': np.float64(67707.569)},
  'Aspartate Aminotransferase (AST)': {'model': 'XGBoost',
   'filled': 67206,
   'mse': np.float64(1723.06)}}}
```

```
df7=pd.read_csv("/content/drive/MyDrive/XXXX/ai_groups_Liver Function_22222/li
```

```
df7[[
    "Alanine Aminotransferase (ALT)",
    "Aspartate Aminotransferase (AST)",
    "Alkaline Phosphatase",
    "Bilirubin (Total)",
    "Bilirubin (Direct)"
]].isnull().sum()
```

	0
Alanine Aminotransferase (ALT)	0
Aspartate Aminotransferase (AST)	0
Alkaline Phosphatase	0
Bilirubin (Total)	0
Bilirubin (Direct)	0

dtype: int64

```
df7[[
    "Creatinine in Serum",
    "Urea in Serum",
    "Estimated Glomerular Filtration Rate(eGFR)",
    "Blood Urea Nitrogen (BUN)",
    "BUN/Creatinine Ratio"
]].isnull().sum()
```

	0
Creatinine in Serum	0
Urea in Serum	0
Estimated Glomerular Filtration Rate(eGFR)	0
Blood Urea Nitrogen (BUN)	0
BUN/Creatinine Ratio	0

dtype: int64

```
missing_report = df7.isnull().sum().to_frame(name='Missing Count')
missing_report['Total Rows'] = len(df7)
missing_report['Missing %'] = (missing_report['Missing Count'] / missing_report

missing_report = missing_report[missing_report['Missing Count'] > 0]
missing_report = missing_report.sort_values(by='Missing %', ascending=False)

pd.set_option('display.float_format', lambda x: '%.2f' % x)
missing_report
```

	Missing Count	Total Rows	Missing %
Blood and Haemoglobin	900000	900000	100.00
Ketones	900000	900000	100.00
Amorphous Elements	900000	900000	100.00
Casts(Urine Microscopic Examination :)	900000	900000	100.00
Urobilinogen	900000	900000	100.00
W.B.Cs / HPF	900000	900000	100.00
Cystatin C	899972	900000	100.00
24 Hour Urine Volume (263)	899966	900000	100.00
Albumin in Urine (263)	899965	900000	100.00

Albumin in Urine (203)	899965	900000	100.00
Concentration	899951	900000	99.99
R.B.Cs / HPF	899950	900000	99.99
Consistancy	899950	900000	99.99
W. B. Cs / HPF	899296	900000	99.92
Rheumatoid Factor (quantitative)	898971	900000	99.89
T. Cholesterol/HDL_Numeric	898838	900000	99.87
Lead in blood	898285	900000	99.81
Epithelial Cells / HPF	897127	900000	99.68
Leucocyte esterase	897127	900000	99.68
Bilirubin_Numeric	897127	900000	99.68
Specific Gravity	897127	900000	99.68
Amorphous_Numeric	897127	900000	99.68
Colour(Urine Physical Examination)	897127	900000	99.68
Glucose(Urine Physical Examination)	897127	900000	99.68
Nitrite	897127	900000	99.68
Aspect(Urine Physical Examination) Ordinal Encoding	897127	900000	99.68
pH(Urine Physical Examination)	897127	900000	99.68
Protein(Urine Physical Examination)	897127	900000	99.68
Red cell count	897015	900000	99.67
Lymphocytes absolute count	896940	900000	99.66
Neutrophils absolute count	896940	900000	99.66
MCH	896940	900000	99.66
Total Leucocytic Count	896940	900000	99.66
Monocytes absolute count	896940	900000	99.66
Basophils absolute count	896940	900000	99.66
Eosinophils absolute count	896940	900000	99.66
Hematocrit	896934	900000	99.66
Transferrin	894646	900000	99.41
Titre on Hep2 cells	891124	900000	99.01

Titre on Hep2 cells	891124	900000	99.01
Anti CCP Abs	879183	900000	97.69
Microalbuminuria (24 h urine)	870421	900000	96.71
C-Reactive Protein (CRP) quantitative	861733	900000	95.75
Magnesium (Mg) in Serum	842978	900000	93.66
Prostatic Specific Antigen (PSA) Total	681556	900000	75.73
Testosterone (Total)	669736	900000	74.42
Ferritin In Serum	620550	900000	68.95
Iron (Fe) in Serum	610339	900000	67.82
Erythrocyte Sedimentation Rate(ESR)	575728	900000	63.97
Globulin in Serum	404908	900000	44.99
Total Protein in Serum	396129	900000	44.01
Chloride in Serum	381543	900000	42.39
Diastolic Pressure	381332	900000	42.37
Systolic Pressure	381332	900000	42.37
Sodium (Na) in Serum	370012	900000	41.11
Potassium (K) in Serum	366659	900000	40.74
CRP H.S	353305	900000	39.26
Free T4	254496	900000	28.28
Albumin in Serum	203570	900000	22.62
Thyroid Stimulating Hormone (TSH)	105898	900000	11.77
Calcium in Serum (Total)	97279	900000	10.81
Uric Acid in Serum	89232	900000	9.91

```
df8=pd.read_csv('/content/drive/My Drive/XXXX/final marg.csv')
```

```
df9.isnull().sum().sum()
```

```
np.int64(37576155)
```

```
Start coding or generate with AI.
```