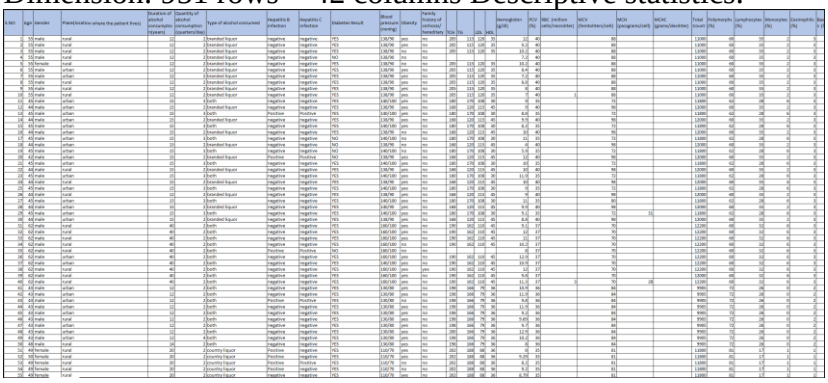
Data Collection and Preprocessing Phase

Date	2 July
Team ID	SWTID1720110187
Project Title	Liver Cirrhosis Prediction Dataset
Maximum Marks	6 Marks

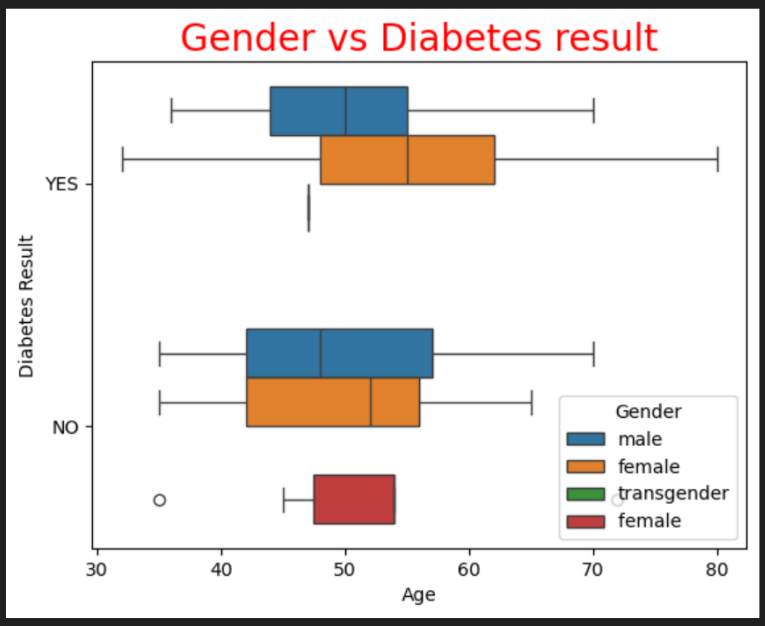
Data Exploration and Preprocessing Template

Identifies data sources, assesses quality issues like missing values and duplicates, and implements resolution plans to ensure accurate and reliable analysis.

Section	Description
Data Overview	Dimension: 951 rows × 42 columns Descriptive statistics: 

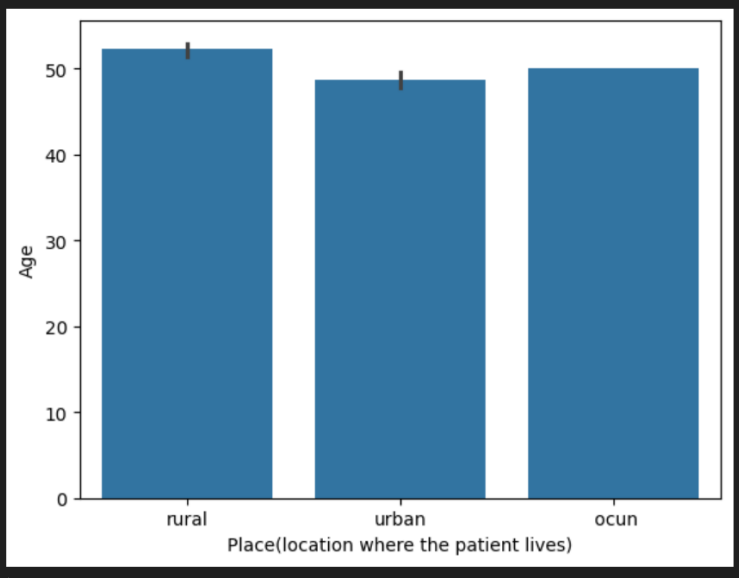
Univariate Analysis

```
sns.boxplot(x='Age',y='Diabetes Result',data=dt,hue='Gender')
plt.title('Gender vs Diabetes result',color='red',size=20)
plt.show()
```



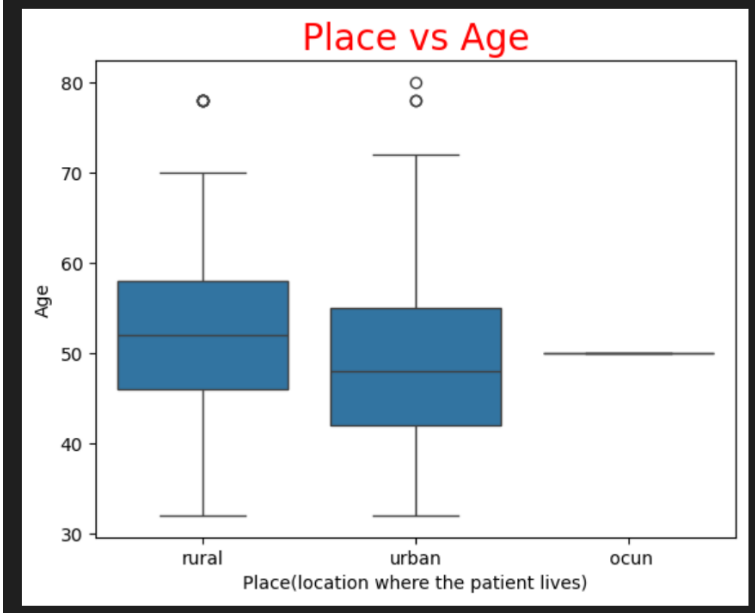

```
sns.barplot(x=dt['Place(location where the patient lives)'],y=dt['Age'])
```

```
<Axes: xlabel='Place(location where the patient lives)', ylabel='Age'>
```

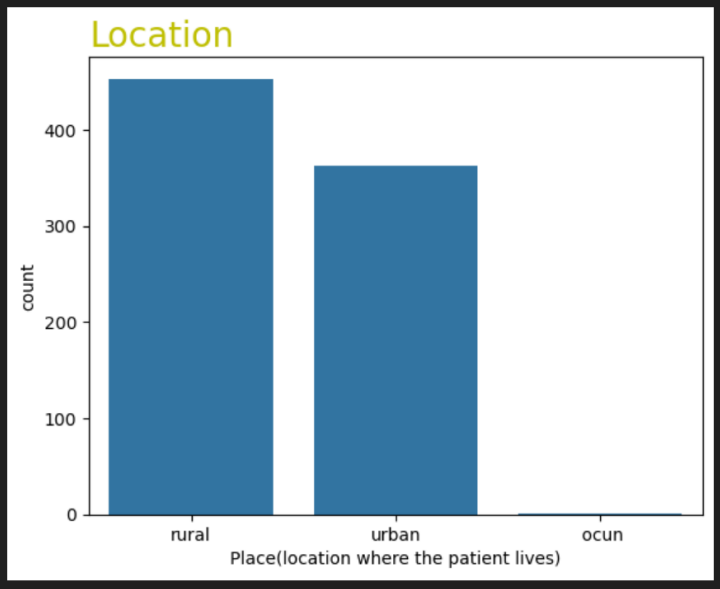


```
sns.boxplot(x='Place(location where the patient lives)',y='Age',data=dt)
plt.title('Place vs Age',color='red',size=20)
```

```
Text(0.5, 1.0, 'Place vs Age')
```

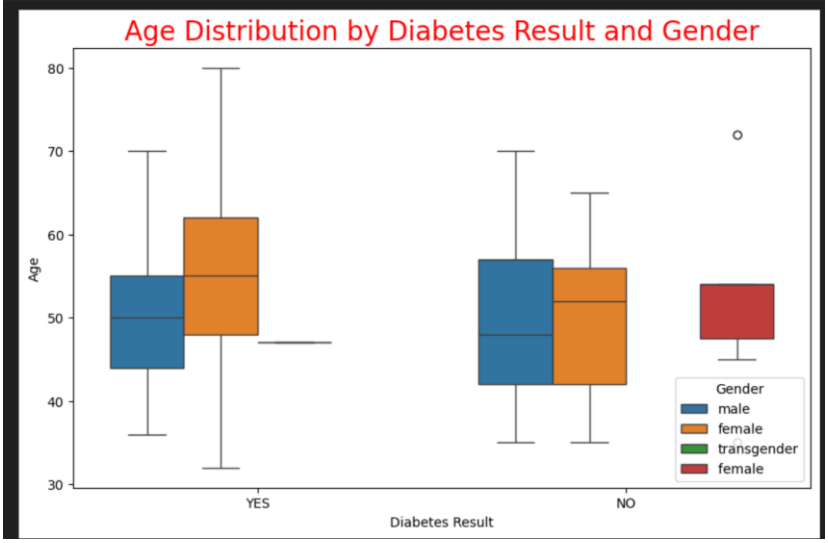



```
sns.countplot(data=dt,x='Place(location where the patient lives)')
plt.title("Location",color='y',size=20,loc="left")
plt.show()
```

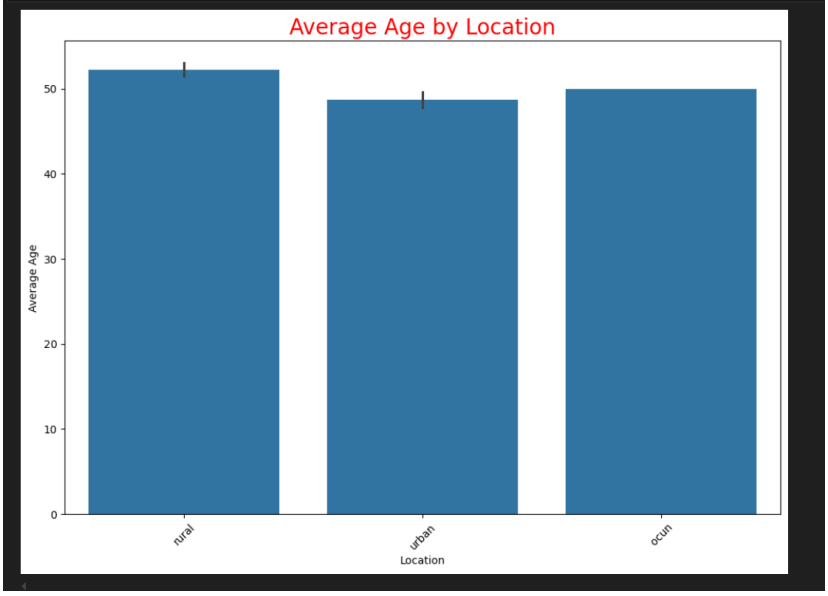


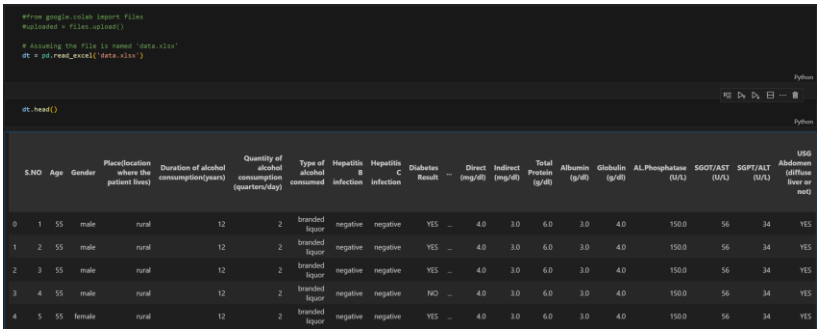
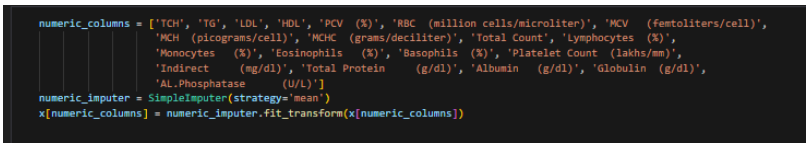
Bivariate Analysis

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Diabetes Result', y='Age', data=dt, hue='Gender')
plt.title('Age Distribution by Diabetes Result and Gender', color='red', size=20)
plt.xlabel('Diabetes Result')
plt.ylabel('Age')
plt.show()
```




```
plt.figure(figsize=(12, 8))
sns.barplot(x='Place(location where the patient lives)', y='Age', data=dt)
plt.title('Average Age by Location', color='red', size=20)
plt.xlabel('Location')
plt.ylabel('Average Age')
plt.xticks(rotation=45)
plt.show()
```



Multivariate Analysis	
Outliers and Anomalies	-
Data Preprocessing Code Screenshots	
Loading Data	
Handling Missing Data	

```
#from google.colab import files
#uploaded = files.upload()

# Assuming the file is named 'data.xlsx'
dt = pd.read_excel('data.xlsx')
```

```
dt.head()
```

	S.NO	Age	Gender	Place(location where the patient lives)	Duration of alcohol consumption(years)	Quantity of alcohol consumption (quarters/day)	Type of alcohol consumed	Hepatitis B infection	Hepatitis C infection	Diabetes Result	...	Direct (mg/dl)	Indirect (mg/dl)	Total Protein (g/dl)	Albumin (g/dl)	Globulin (g/dl)	AL.Phosphatase (U/L)	SGOT/AST (U/L)	SGPT/ALT (U/L)	USG Abdomen (diffuse liver or not)
0	1	55	male	rural	12	2	branded liquor	negative	negative	YES	...	4.0	3.0	6.0	3.0	4.0	150.0	56	34	YES
1	2	55	male	rural	12	2	branded liquor	negative	negative	YES	...	4.0	3.0	6.0	3.0	4.0	150.0	56	34	YES
2	3	55	male	rural	12	2	branded liquor	negative	negative	YES	...	4.0	3.0	6.0	3.0	4.0	150.0	56	34	YES
3	4	55	male	rural	12	2	branded liquor	negative	negative	NO	...	4.0	3.0	6.0	3.0	4.0	150.0	56	34	YES
4	5	55	female	rural	12	2	branded liquor	negative	negative	YES	...	4.0	3.0	6.0	3.0	4.0	150.0	56	34	YES

```
numeric_columns = ['TCH', 'TG', 'LDL', 'HDL', 'PCV  (%)', 'RBC  (million cells/microliter)', 'MCV   (femtoliters/cell)',
                   'MCH  (picograms/cell)', 'MCHC  (grams/deciliter)', 'Total Count', 'Lymphocytes  (%)',
                   'Monocytes   (%)', 'Eosinophils   (%)', 'Basophils  (%)', 'Platelet Count  (lakhs/mm)',
                   'Indirect     (mg/dl)', 'Total Protein     (g/dl)', 'Albumin   (g/dl)', 'Globulin  (g/dl)',
                   'AL.Phosphatase      (U/L)']
numeric_imputer = SimpleImputer(strategy='mean')
x[numeric_columns] = numeric_imputer.fit_transform(x[numeric_columns])
```

Data Transformation	ENCODING CATEGORICAL DATA

```
bp_split = dt['Blood pressure (mmhg)'].str.split('/', expand=True)
x['Systolic'] = pd.to_numeric(bp_split[0], errors='coerce')
x['Diastolic'] = pd.to_numeric(bp_split[1], errors='coerce')
x = x.drop(columns=['Blood pressure (mmhg)','S.NO'])

print("Columns in x before transformation:")
print(x.columns)
x = pd.DataFrame(x)
```

```
Columns in x before transformation:
Index(['Age', 'Gender', 'Place(location where the patient lives)',
       'Duration of alcohol consumption(years)',
       'Quantity of alcohol consumption (quarters/day)',
       'Type of alcohol consumed', 'Hepatitis B infection',
       'Hepatitis C infection', 'Diabetes Result', 'Obesity',
       'Family history of cirrhosis/ hereditary', 'TCH', 'TG', 'LDL', 'HDL',
       'Hemoglobin  (g/dl)', 'PCV  (%)', 'RBC  (million cells/microliter)',
       'MCV   (femtoliters/cell)', 'MCH  (picograms/cell)',
       'MCHC  (grams/deciliter)', 'Total Count', 'Polymorphs  (%) ',
       'Lymphocytes  (%)', 'Monocytes   (%)', 'Eosinophils   (%)',
       'Basophils  (%)', 'Platelet Count  (lakhs/mm)',
       'Total Bilirubin    (mg/dl)', 'Direct    (mg/dl)',
       'Indirect     (mg/dl)', 'Total Protein     (g/dl)', 'Albumin   (g/dl)',
       'Globulin  (g/dl)', 'AL.Phosphatase      (U/L)', 'SGOT/AST      (U/L)',
       'SGPT/ALT (U/L)', 'USG Abdomen (diffuse liver or  not)', 'Systolic',
       'Diastolic'],
      dtype='object')
```

```
cat_columns = ['Gender', 'Place(location where the patient lives)', 'Type of alcohol consumed',
               'Hepatitis B Infection', 'Hepatitis C Infection', 'Diabetes Result', 'Obesity',
               'Family history of cirrhosis/ hereditary', 'USG Abdomen (diffuse liver or  not)']
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), cat_columns)], remainder='passthrough')
#ct = make_column_transformer((OneHotEncoder(), cat_columns), remainder='passthrough')
x_encoded = ct.fit_transform(x)
x_encoded = pd.DataFrame(x_encoded, columns=ct.get_feature_names_out())
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Get feature names from one-hot encoder
onehot_features = ct.named_transformers_['encoder'].get_feature_names_out(cat_columns)

# Get remaining feature names that were not one-hot encoded
passthrough_features = [column for column in x.columns if column not in cat_columns]

# Combine the two lists
all_features = list(onehot_features) + passthrough_features

# Print all feature names
print(all_features)

# Assuming x is already defined after preprocessing steps like dropping columns
#print("Columns in x:")
#print(x.columns)
# Check if all columns in cat_columns exist in x.columns
#missing_columns = [col for col in cat_columns if col not in x.columns]

#if missing_columns:
#    print("Error: The following categorical columns are missing in x:", missing_columns)
#else:
#    print("All categorical columns are correctly specified and present in x.")
#print(ct)



#transformed_columns = ct.get_feature_names_out()
#print("Transformed columns:")
#print(transformed_columns)
```

Python

```
['Gender_female', 'Gender_female ', 'Gender_male', 'Gender_transgender', 'Place(location where the patient lives)_ ocun', 'Place(location where the patient lives)_rural', 'Place(loca
```

Feature Engineering	PREPROCESSED DATA
	`preprocessed_data = pd.DataFrame(x_encoded, columns=ct.get_feature_names_out())`
	`preprocessed_data['Predicted Value'] = y_encoded`
	`# Define preprocessing pipeline`
	`numeric_transformer = Pipeline(steps=[`
	`('imputer', SimpleImputer(strategy='mean')),`
	`('scaler', StandardScaler())`
	`])`
	`categorical_transformer = Pipeline(steps=[`
	`('imputer', SimpleImputer(strategy='constant', fill_value='missing')),`
	`('onehot', OneHotEncoder(handle_unknown='ignore'))`
	`])`
	`preprocessor = ColumnTransformer(`
	`transformers=[`
	`('num', numeric_transformer, numeric_columns),`
	`('cat', categorical_transformer, cat_columns)`
	`])`
	`preprocessed_data.isnull().sum()`
	encoder__Gender_female 0
	encoder__Gender_female 0
	encoder__Gender_male 0
	encoder__Gender_transgender 0
	encoder__Place(location where the patient lives)_ ocun 0
	encoder__Place(location where the patient lives)_rural 0
	encoder__Place(location where the patient lives)_urban 0
	encoder__Place(location where the patient lives)_nan 0
	encoder__Type of alcohol consumed_ branded liquor 0
	encoder__Type of alcohol consumed_both 0
	encoder__Type of alcohol consumed_branded liquor 0
	encoder__Type of alcohol consumed_country liquor 0
	encoder__Hepatitis B infection_Positive 0
	encoder__Hepatitis B infection_negative 0
	encoder__Hepatitis B infection_positive 0
	encoder__Hepatitis C infection_Positive 0
	encoder__Hepatitis C infection_negative 0
	encoder__Hepatitis C infection_positive 0
	encoder__Diabetes Result_NO 0
	encoder__Diabetes Result_YES 0
	encoder__Obesity_no 0
	encoder__Obesity_yes 0
	encoder__Family history of cirrhosis/ hereditary_no 0
	encoder__Family history of cirrhosis/ hereditary_yes 0
	encoder__USG Abdomen (diffuse liver or not)_YES 0
	...
	remainder__SGPT/ALT (U/L) 0
	remainder__Systolic 0
	remainder__Diastolic 0
	Predicted Value 0
	dtype: int64
	Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

	FEATURE SCALING
	`scaler = StandardScaler()`
	`normalized_features = scaler.fit_transform(preprocessed_data.drop(columns=['Predicted Value']))`
	`normalized_data = pd.DataFrame(normalized_features, columns=preprocessed_data.columns[:-1])`
	`normalized_data['Predicted Value'] = preprocessed_data['Predicted Value']`
	`print(normalized_data.head())`
	`#fit and transform the data`
	`'''`
	`X = dt.drop(columns=['Predicted Value'])`
	`y = dt['Predicted Value']`
	`X_preprocessed = preprocessor.fit_transform(X)`
	`# Save the preprocessor`
	`joblib.dump(preprocessor, 'preprocessor.pkl')`
	`'''`
	encoder__Gender_female encoder__Gender_female encoder__Gender_male \
	0 -0.508211 -0.108234 0.534342
	1 -0.508211 -0.108234 0.534342
	2 -0.508211 -0.108234 0.534342
	3 -0.508211 -0.108234 0.534342
	4 1.967688 -0.108234 -1.871462
	encoder__Gender_transgender \
	0 -0.072739
	1 -0.072739
	2 -0.072739
	3 -0.072739
	4 -0.072739
	encoder__Place(location where the patient lives)_ ocun \
	0 -0.032461
	1 -0.032461
	2 -0.032461
	3 -0.032461
	4 -0.032461
	encoder__Place(location where the patient lives)_rural \
	0 1.04744
	1 1.04744
	2 1.04744
	...
	3 0
	4 0
	[5 rows x 58 columns]
	Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
	`"\nX = dt.drop(columns=['Predicted Value'])\ny = dt['Predicted Value']\nX_preprocessed = preprocessor.fit_transform(X)\n\n# Save the preprocessor\njoblib.dump(preprocessor, 'preprocessor.pkl')\n"`

Save Processed Data	-