LINEAR REGRESSION (OLS) & XGBOOST COMBINED WITH EXPLANATION

#CODE

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
import xgboost as xgb
import openpyxl

#LoadOriginalDataset
# List of continuous and categorical variables
continuous_vars = ['Cont1', 'Cont2', 'Cont3', 'Cont4', 'Cont5']
categorical_vars = ['Cat1', 'Cat2', 'Cat3', 'Cat4', 'Cat5']

# Outlier detection and treatment for continuous variables using percentile method
for var in continuous_vars:
    lower_bound = leaf[var].quantile(0.01)
    upper_bound = leaf[var].quantile(0.99)
    median = leaf[var].median()
    leaf[var] = np.where((leaf[var] < lower_bound) | (leaf[var] > upper_bound), median, leaf[var])

# Missing value detection and treatment
# Fill missing values in continuous variables with the median
for var in continuous_vars:
    median = leaf[var].median()
    leaf[var].fillna(median, inplace=True)
# Fill missing values in categorical variables with the mode
for var in categorical_vars:
    mode = leaf[var].mode()[0]
    leaf[var].fillna(mode, inplace=True)
```

```
# One-hot encode categorical variables
leaf_encoded = pd.get_dummies(leaf[categorical_vars], dtype=int)
leaf = leaf.drop(columns=categorical_vars)
leaf = pd.concat([leaf, leaf_encoded], axis=1)

# Create interaction variables for continuous variables
for i in range(len(continuous_vars)):
    for j in range(i+1, len(continuous_vars)):
        col_name = f"{continuous_vars[i]}_{continuous_vars[j]}"
        leaf[col_name] = leaf[continuous_vars[i]] * leaf[continuous_vars[j]]

# Create interaction variables for continuous and encoded categorical variables
encoded_categorical_vars = leaf_encoded.columns.tolist()
for cont in continuous_vars:
    for cat in encoded_categorical_vars:
        col_name = f"{cont}_{cat}"
        leaf[col_name] = leaf[cont] * leaf[cat]

# Create interaction variables for encoded categorical variables
for i in range(len(encoded_categorical_vars)):
    for j in range(i+1, len(encoded_categorical_vars)):
        col_name = f"{encoded_categorical_vars[i]}_{encoded_categorical_vars[j]}"
        leaf[col_name] = leaf[encoded_categorical_vars[i]] * leaf[encoded_categorical_vars[j]]

# Separate the dependent variable
y = leaf['DNBScore']
X = leaf.drop(columns=['DNBScore'])

# Check for multicollinearity using VIF
def calculate_vif(X):
```

```
    vif = pd.DataFrame()
    vif['Variable'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif

vif_df = calculate_vif(X)
print("VIF before removing multicollinear features:")
print(vif_df)

# Remove features with VIF greater than 10
features_to_remove = vif_df[vif_df['VIF'] > 10]['Variable'].tolist()
X = X.drop(columns=features_to_remove)

# Check VIF again after removal
vif_df_after = calculate_vif(X)
print("VIF after removing multicollinear features:")
print(vif_df_after)

# Train Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(X, y)
y_pred_linear = linear_model.predict(X)
linear_mse = mean_squared_error(y, y_pred_linear)

# Print Linear Regression MSE
print(f"Linear Regression MSE: {linear_mse}")

# Get Linear Regression coefficients
linear_coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': linear_model.coef_
```

```
}).sort_values(by='Coefficient', key=abs, ascending=False)

# Print Linear Regression Coefficients
print("Linear Regression Coefficients:")
print(linear_coefficients)

# Convert the DataFrame to DMatrix
data_dmatrix = xgb.DMatrix(data=X, label=y)

# Train the XGBoost model
params = {
    'objective': 'reg:squarederror',
    'max_depth': 3,
    'learning_rate': 0.1,
    'n_estimators': 100,
}

# Train the model on the entire dataset
model = xgb.train(params, data_dmatrix, num_boost_round=100)

# Get feature importance scores based on gain
importance_scores = model.get_score(importance_type='gain')
sorted_importance = {k: v for k, v in sorted(importance_scores.items(), key=lambda item: item[1], reverse=True)}

# Print the feature importance scores
print("XGBoost Feature Importance Scores (Gain):", sorted_importance)

# Convert the feature importance dictionary to a DataFrame
importance_df = pd.DataFrame(list(sorted_importance.items()), columns=['Feature', 'Importance'])
```

 # Combine Linear Regression coefficients and XGBoost feature importance

```
combined_df = pd.merge(linear_coefficients, importance_df, on='Feature', how='outer').fillna(0)
combined_df = combined_df.sort_values(by='Importance', ascending=False)
```

 # Export the DataFrame to an Excel file

```
with pd.ExcelWriter('feature_importance_scores.xlsx', engine='openpyxl') as writer:
    linear_coefficients.to_excel(writer, sheet_name='Linear Regression Coeffs', index=False)
    importance_df.to_excel(writer, sheet_name='XGBoost Feature Importance', index=False)
    combined_df.to_excel(writer, sheet_name='Combined Importance', index=False)

print("Feature importance scores have been exported to 'feature_importance_scores.xlsx'.")
```

Explanation

1. **Outlier Detection and Treatment**:
    - For each continuous variable, calculate the 1st and 99th percentiles.
    - Replace values below the 1st percentile or above the 99th percentile with the median of the respective variable.
2. **Missing Value Detection and Treatment**:
    - For continuous variables, fill missing values with the median.
    - For categorical variables, fill missing values with the mode.
3. **One-Hot Encoding**:
    - Use pd.get_dummies to convert categorical variables into a binary matrix.
    - dtype=int ensures the columns are of integer type.
4. **Combining Data**:
    - Drop the original categorical columns from the dataset.
    - Concatenate the one-hot encoded columns with the original dataset.
5. **Interaction Variables**:
    - Generate interaction terms for continuous variables.
    - Generate interaction terms for continuous variables with the one-hot encoded categorical variables.
    - Generate interaction terms between different one-hot encoded categorical variables.

6. **Multicollinearity Check Using VIF**:
    - Calculate VIF for all features.
    - Remove features with VIF greater than 10.
    - Recalculate VIF to confirm multicollinearity has been addressed.
7. **Linear Regression**:
    - Train a linear regression model on the dataset after removing multicollinear features.
    - Calculate the mean squared error (MSE) for the linear regression model.
    - Extract and print the coefficients of the linear regression model.
8. **XGBoost**:
    - Prepare the data and train the XGBoost model.
    - Print feature importance

Potential Number of Interaction Terms

If you have 20 variables and you create pairwise interaction terms, the number of unique interaction terms can be calculated using the combination formula $C(n, 2) = \frac{n(n-1)}{2}$:

For 20 variables, the number of pairwise interactions would be: $\frac{20*19}{2} = 190$

This assumes interactions between all variables (both continuous and categorical, after one-hot encoding).

Reasons for Fewer Interaction Terms in the Output

1. **Feature Selection in XGBoost**:
    - XGBoost may not use all interaction terms if some are not significant in reducing the loss function.
    - During training, XGBoost performs feature selection by including only those features (or interactions) that contribute to a better model.
2. **Regularization**:
    - XGBoost applies regularization techniques (e.g., L1 and L2 regularization) to prevent overfitting, which can lead to some features being assigned zero importance or being pruned out during training.
3. **Initial Dataset After Encoding**:
    - If you have categorical variables that are one-hot encoded, the actual number of initial features can be larger than 20. For example, a categorical variable with 3 levels will be split into 3 binary variables.

- The code example provided earlier does handle one-hot encoding for categorical variables. If your dataset includes these encoded variables, the total number of initial features will increase before interaction terms are created.

4. **Creation of Interaction Terms**:
    - The provided code creates interaction terms for continuous variables, continuous with encoded categorical, and between categorical variables.
    - If you are seeing fewer interaction terms, ensure that all interaction terms are being correctly generated and included in the dataset used for training.