CS613 Project -Francis Kurian

Monte Carlo Simulation of Divergence detection in Data Distributions using Kullback-Leibler (KL) measure

Monte Carlo simulation is a statistical technique used to model and analyze complex systems or processes by generating random samples or scenarios to estimate outcomes and assess uncertainty.

In a Monte Carlo simulation, a mathematical model is created to represent the system under study. Random input values, often following known probability distributions, are then generated for the model's variables. The model is run with these random inputs, and the process is repeated numerous times to create a distribution of possible outcomes. This distribution allows analysts to make probabilistic predictions and assess the likelihood of different scenarios or events occurring.

This simulation address the problem of drifts in the data distributions and how to detect the divergence within the same data fields (FICO Score) processed from two different sample populations. A significant drift in data distributions between samples is a surprise and such divergence has been a subject matter of several studies in the field of statistics.

Kullback–Leibler (KL) divergence in mathematical statistics is a distance and relative entropy measure. It quantifies the relative difference between two probability distributions with a relative entropy of zero suggests two distributions are identical. We are using Population Stability Index (PSI), a variant of KL in this project. PSI is described in detail below.

Population Stability Index as a variant of Kullback–Leibler divergence

Given two discrete probability distributions P (actual), and Q (expected), KL divergence is defined as:

$$D_{KL}(P(x)|Q(x)) = \sum_{i=1}^{B} P(x_i).\, ln\, \frac{P(x_i)}{Q(x_i)}$$

An interpretation of KL divergence is that it measures the expected excess surprise in using the actual distribution versus the expected distribution as a divergence of the actual from the expected. B is the number of buckets(discrete) of the distribution.

DK_L measures divergence however, researchers note that it's not a true distance measure as its definition is not symmetric. $That\ is, D_{KL}(Q(x)|P(x) \neq D_{KL}(P(x)|Q(x)$

A symmetric measure is obtained by defining:

$$D(P,Q) = D_{KL}(Q|P) = D_{KL}(P|Q)$$

$$= \sum P(x_i) ln \frac{P(x_i)}{Q(x_i)} + \sum Q(x_i) ln \frac{Q(x_i)}{P(x_i)}$$

$$= \sum P(x_i) ln \frac{P(x_i)}{Q(x_i)} - \sum Q(x_i) ln \frac{P(x_i)}{Q(x_i)}$$

$$= \sum (P(x_i) - Q(x_i)) ln \frac{P(x_i)}{Q(x_i)}$$

This variant of K-L divergence is known as Population Stability Index (PSI) and is widely used in machine learning and model validations a divergence measure. The following steps will show how to compute PSI using the $ sales data we discussed in the problem statement.

From the derivation above,

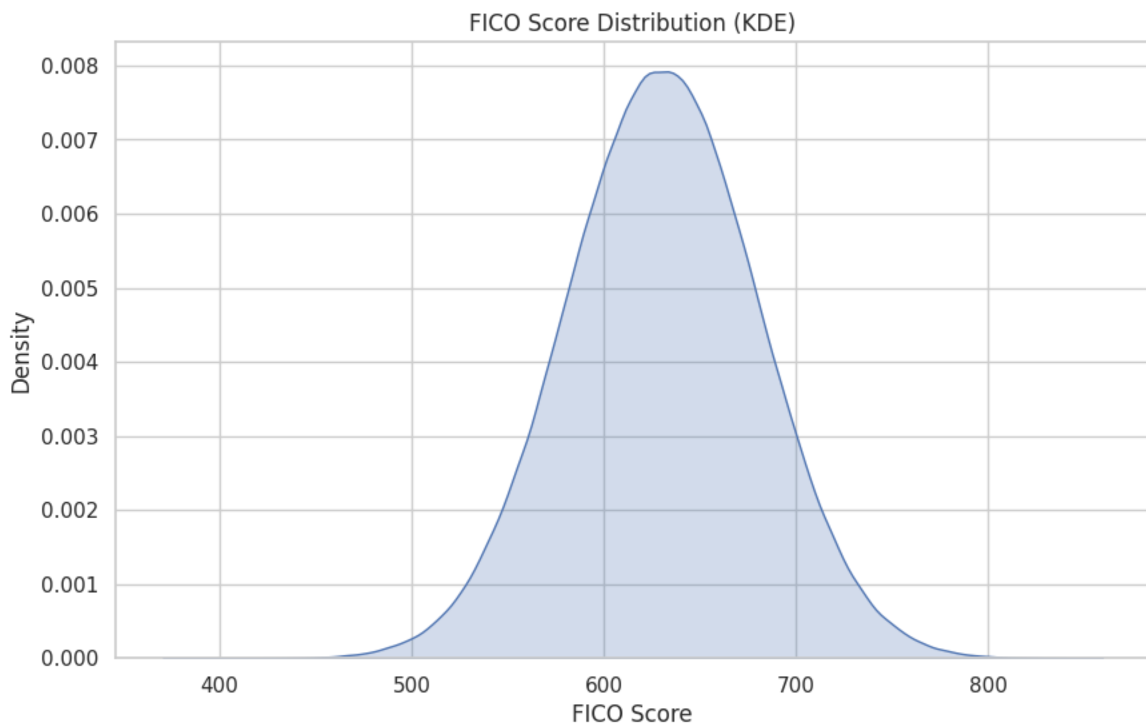
$$PSI = \sum_{i=1}^{B} [P(x_i) - Q(x_i)] \times ln \frac{P(x_i)}{Q(x_i)}$$

Step 1: Baseline Distribution Data Preperation -1 Million records normal distribution

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Set the random seed for reproducibility
np.random.seed(0)

# Define the parameters
sample_size = 1000000
mean_score = 630
```

```python
std_deviation = 50

# Generate FICO scores using a normal distribution
fico_scores = np.random.normal(mean_score, std_deviation, sample_size)

# Round the FICO scores to integers between 1 and 850
fico_scores = np.round(fico_scores)
fico_scores = np.clip(fico_scores, 1, 850)  # Clip values to the desired range

# Create a DataFrame
data = pd.DataFrame({'PersonID': range(1, sample_size + 1),
                     'FICO_Score': fico_scores})

# Define the bins and labels for 6 FICO_Score segments
bins = [0, 600, 640, 680, 720, 760, 850]
labels = ['<600', '600-640', '641-680', '681-720', '721-760', '>760']

# Create a new column 'new_count' to store the grouping information
data['base_count'] = pd.cut(data['FICO_Score'], bins=bins, labels=labels, right=False)


# Create summary dataset data2
data2 = data['base_count'].value_counts().reset_index()
data2.columns = ['FICO_Range', 'base_count']
```

```python
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# Create a KDE plot
sns.set(style="whitegrid")  # Set the style for the plot
plt.figure(figsize=(10, 6))  # Set the figure size

sns.kdeplot(data['FICO_Score'], shade=True, color="b")  # Create the KDE plot

plt.title("FICO Score Baseline Distribution (KDE)")
plt.xlabel("FICO Score")
plt.ylabel("Density")
plt.show()
```



Step 2: Monte Carlo Simulation: Real life Scenarios to show drifts from diffrent samples

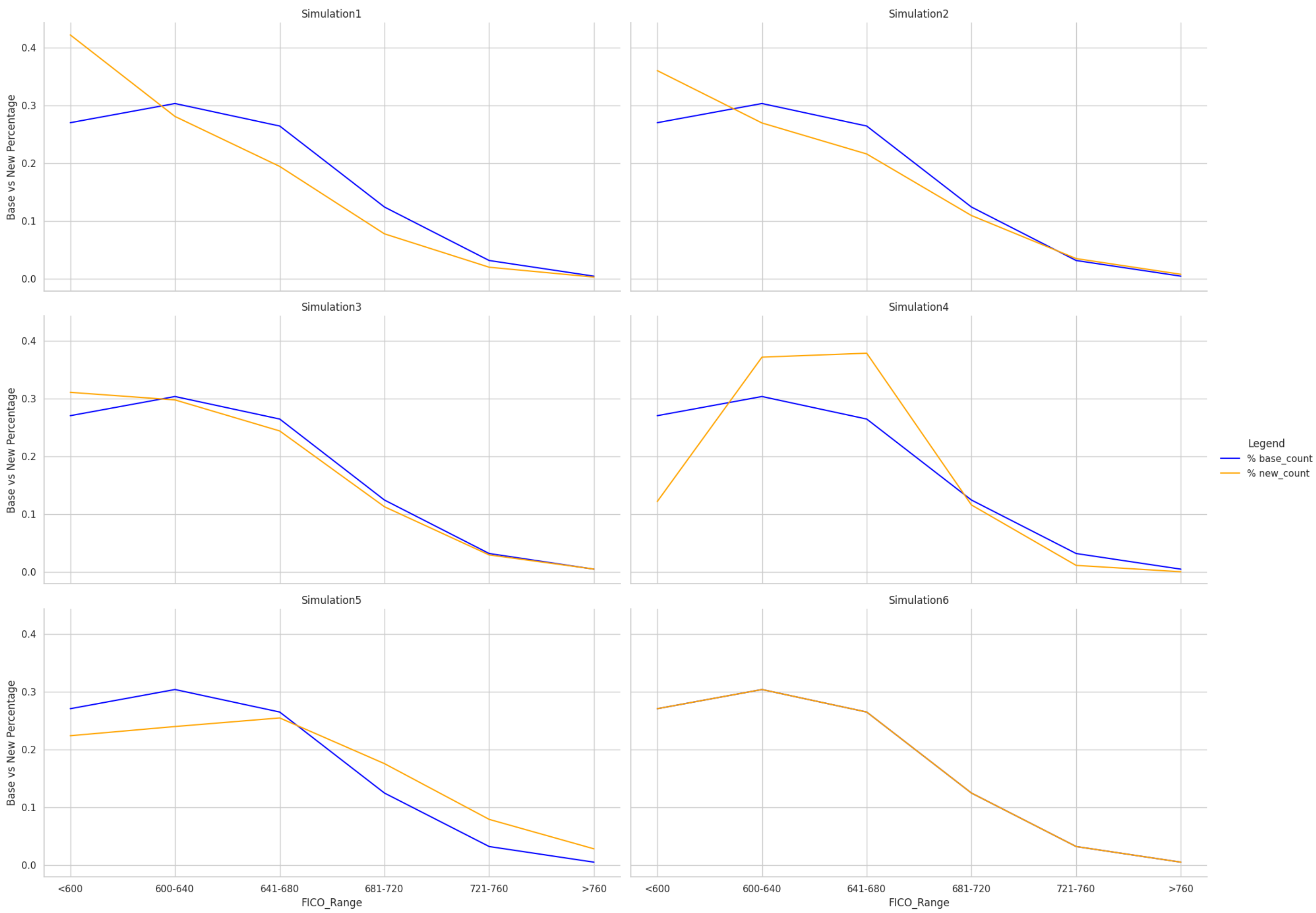
Step 3: Compute PSI for each sample

```
def simulate_fico_data(sample_size, mean_score, std_deviation, simulation_name):
    # Set the random seed for reproducibility
    np.random.seed(0)

    # Generate FICO scores using a normal distribution
    fico_scores = np.random.normal(mean_score, std_deviation, sample_size)

    # Round the FICO scores to integers between 1 and 850
    fico_scores = np.round(fico_scores)
    fico_scores = np.clip(fico_scores, 1, 850)  # Clip values to the desired range

    # Create a DataFrame
    data_new = pd.DataFrame({'PersonID': range(1, sample_size + 1),
                            'FICO_Score': fico_scores})

    # Define the bins and labels for 6 FICO_Score segments
    bins = [0, 600, 640, 680, 720, 760, 850]
    labels = ['<600', '600-640', '641-680', '681-720', '721-760', '>760']

    # Create a new column 'new_count' to store the grouping information
    data_new['new_count'] = pd.cut(data_new['FICO_Score'], bins=bins, labels=labels, right=False)

    # Create summary dataset data2
    data_new2 = data_new['new_count'].value_counts().reset_index()
    data_new2.columns = ['FICO_Range', 'new_count']

    # Merge data2 and data_new2 using the 'FICO_Range' column
    merged_data = pd.merge(data2, data_new2, on='FICO_Range', how='inner')

    # Step 1: Calculate '% base_count'
    merged_data['% base_count'] = merged_data['base_count'] / merged_data['base_count'].sum()

    # Step 2: Calculate '% new_count'
    merged_data['% new_count'] = merged_data['new_count'] / merged_data['new_count'].sum()

    # Step 3: Calculate PSI for each row
    merged_data['PSI'] = (merged_data['% new_count'] - merged_data['% base_count']) * np.log(merged_data['% new_count'] / merg

    # Step 4: Calculate the total PSI
    total_psi = merged_data['PSI'].sum()

    # Add a 'simulation' column specified by the user
    merged_data['simulation'] = simulation_name

    return merged_data, total_psi

# Example usage with user-specified simulation name


# Create an empty DataFrame to store results
all_results = pd.DataFrame()

# Perform multiple function calls and append results
sample_sizes = [100000, 500000, 200000,100000,300000, 1000000]
mean_scores = [610, 620, 625, 640,645,630]
std_deviations = [55, 58, 52,35,60,50]

for i in range(len(sample_sizes)):
    sample_size = sample_sizes[i]
    mean_score = mean_scores[i]
    std_deviation = std_deviations[i]
    simulation_name = f'Simulation{i + 1}'  # Generate simulation name

    result, total_psi = simulate_fico_data(sample_size, mean_score, std_deviation, simulation_name)

    # Append the result to the all_results DataFrame
    all_results = pd.concat([all_results, result], ignore_index=True)


# Select and print specific columns in all_results
print(all_results[['simulation','FICO_Range', '% base_count','new_count', '% new_count', 'PSI']])
```

	simulation	FICO_Range	% base_count	new_count	% new_count	PSI
0	Simulation1	600-640	0.303673	28125	0.281250	0.001720
1	Simulation1	<600	0.270458	42242	0.422420	0.067757
2	Simulation1	641-680	0.264681	19475	0.194750	0.021456
3	Simulation1	681-720	0.124479	7806	0.078060	0.021662
4	Simulation1	721-760	0.031898	2029	0.020290	0.005252
5	Simulation1	>760	0.004811	323	0.003230	0.000630
6	Simulation2	600-640	0.303673	134980	0.269968	0.003965
7	Simulation2	<600	0.270458	180322	0.360654	0.025958
8	Simulation2	641-680	0.264681	108155	0.216316	0.009759
9	Simulation2	681-720	0.124479	54878	0.109759	0.001852
10	Simulation2	721-760	0.031898	17627	0.035255	0.000336
11	Simulation2	>760	0.004811	4024	0.008048	0.001666
12	Simulation3	600-640	0.303673	59558	0.297790	0.000115
13	Simulation3	<600	0.270458	62177	0.310885	0.005632
14	Simulation3	641-680	0.264681	48811	0.244055	0.001673
15	Simulation3	681-720	0.124479	22574	0.112870	0.001136
16	Simulation3	721-760	0.031898	5890	0.029450	0.000195
17	Simulation3	>760	0.004811	990	0.004950	0.000004
18	Simulation4	600-640	0.303673	37182	0.371820	0.013797
19	Simulation4	<600	0.270458	12188	0.121880	0.118429
20	Simulation4	641-680	0.264681	37863	0.378630	0.040797
21	Simulation4	681-720	0.124479	11597	0.115970	0.000602
22	Simulation4	721-760	0.031898	1135	0.011350	0.021233
23	Simulation4	>760	0.004811	35	0.000350	0.011691
24	Simulation5	600-640	0.303673	71856	0.239600	0.015184
25	Simulation5	<600	0.270458	67083	0.223685	0.008881
26	Simulation5	641-680	0.264681	76317	0.254475	0.000401
27	Simulation5	681-720	0.124479	52601	0.175395	0.017460
28	Simulation5	721-760	0.031898	23664	0.078906	0.042576
29	Simulation5	>760	0.004811	8379	0.027939	0.040686
30	Simulation6	600-640	0.303673	303671	0.303673	0.000000
31	Simulation6	<600	0.270458	270457	0.270458	0.000000
32	Simulation6	641-680	0.264681	264680	0.264681	0.000000
33	Simulation6	681-720	0.124479	124478	0.124479	0.000000
34	Simulation6	721-760	0.031898	31898	0.031898	0.000000
35	Simulation6	>760	0.004811	4811	0.004811	0.000000

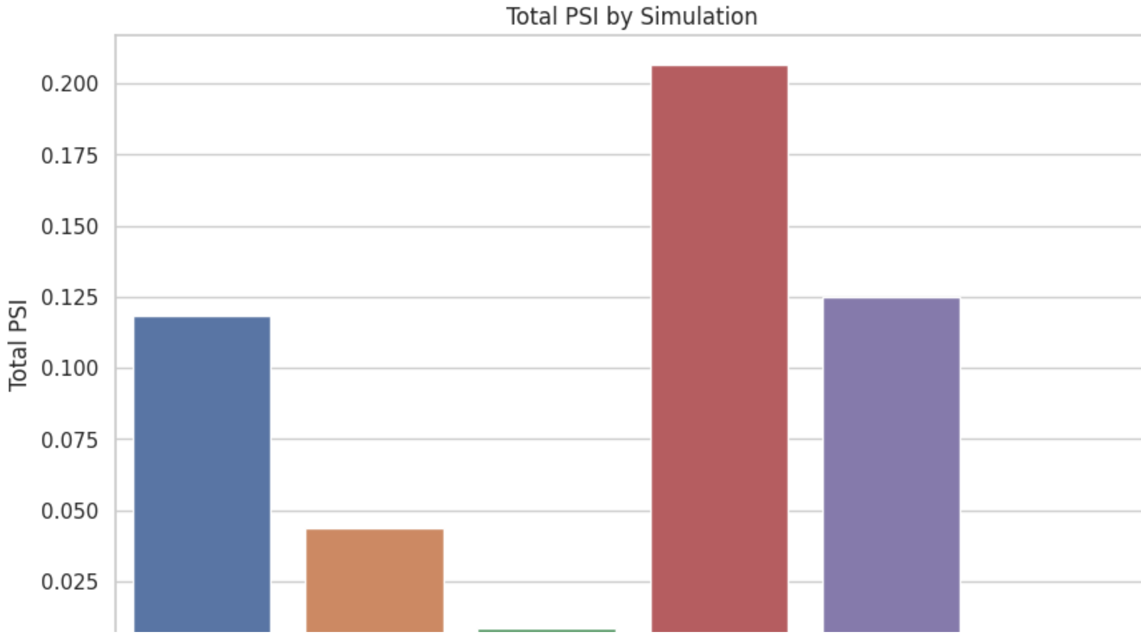
Step 4: Demonstrate the divergence in samples visually

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style for the plots
sns.set_style("whitegrid")

# Create a grid for 2x3 subplots, one for each simulation
g = sns.FacetGrid(all_results, col="simulation", col_wrap=2, height=5, aspect=2)

# Plot % base_count and % new_count for each simulation with different colors
g.map(sns.lineplot, "FICO_Range", "% base_count", color="blue", label="% base_count")
g.map(sns.lineplot, "FICO_Range", "% new_count", color="orange", label="% new_count")

# Set labels and title
g.set_axis_labels("FICO_Range", "Base vs New Percentage")
g.set_titles(col_template="{col_name}")

# Add a customized legend for each subplot
g.add_legend(title="Legend")

# Show the plots
plt.show()

```



Step 5: Compute PSI to establish the drifts in samples - Sample 6 is same as baseline

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate total PSI by Simulation
total_psi_by_simulation = all_results.groupby('simulation')['PSI'].sum().reset_index()

# Set the style for the plots
sns.set_style("whitegrid")

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='simulation', y='PSI', data=total_psi_by_simulation)

# Set labels and title
plt.xlabel("Simulation")
plt.ylabel("Total PSI")
plt.title("Total PSI by Simulation")

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Show the bar plot
plt.show()

```



Summary and Conclusion:

Rules of thumb followed is that if PSI < 0.1 then distributions are similar or ‘little drift’. PSI between 0.1 and 0.25 shows a ‘moderate drift’ and demands a review. Finally, PSI >0.25 means significant divergence or ‘significant drift’ from baseline distribution that needs immediate attention. As expected, PSI was able to detect the distortions introduced into data fields through the simulations with Simulations 1, 4 and 5 showing above the threshold PIS values.