# Data Scientist Professional Practical Exam Submission

**Use this template to write up your summary for submission. Code in Python or R needs to be included.**

## 📝 Task List

Your written report should include both code, output and written text summaries of the following:

- Data Validation:
  - Describe validation and cleaning steps for every column in the data
- Exploratory Analysis:
  - Include two different graphics showing single variables only to demonstrate the characteristics of data
  - Include at least one graphic showing two or more variables to represent the relationship between features
  - Describe your findings
- Model Development
  - Include your reasons for selecting the models you use as well as a statement of the problem type
  - Code to fit the baseline and comparison models
- Model Evaluation
  - Describe the performance of the two models based on an appropriate metric
- Business Metrics
  - Define a way to compare your model performance to the business
  - Describe how your models perform using this approach
- Final summary including recommendations that the business should undertake

*Start writing report here..*

# Data Validation

## Data Validation Summary

This data set has 947 rows, 8 columns. I have validated all variables. I have made some changes after validation:

- recipe: 947 index of recipes without missing values, same as the description. No cleaning is needed.
- calories: numeric values with 52 missing values. Missing values are replaced by the multiindex mean values.
- carbohydrate: numeric values with 52 missing values. Missing values are replaced by the multiindex mean values.
- sugar: numeric values with 52 missing values. Missing values are replaced by the multiindex mean values.
- protein: numeric values with 52 missing values. Missing values are replaced by the multiindex mean values.
- category: 11 categories without missing values. No cleaning is needed.
- servings: 6 categories without missing values. I replaced some values with format category number values.
- high_traffic: 1 categories with 373 missing values. There is only one kind of value, 'High'. So I replaced the 'nan' with 'Low'.

```
# Start coding here...
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from scipy.stats import kruskal
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

```
df=pd.read_csv("recipe_site_traffic_2212.csv")
```

```
df.info()
```

```
df.head(10)
```

```
df['high_traffic'].unique()
```

```
df['category'].value_counts()
```

```
df['servings'].value_counts()
```

```
df.isna().sum()
```

```
df.nunique()
```

```
df['high_traffic']= df['high_traffic'].fillna('Low')
#用 df['high_traffic']或者 df.high_traffic 一样
```

```
df['servings'] = df['servings'].replace(
    {
    '4 as a snack': 4,
    '6 as a snack': 6
    }
)
df['servings'] = df['servings'].astype(int)
```

```
52/len(df)*100
```

The missing values from calories / carbohydrate / sugar / protein is only 5.5% in this data set. However, the the pattern or rule of the missing values is not figured out. Hence, I will not delete the missing values untill furthur explory.

```
# deal with the missing values of the four nutrients - 用分类的multiindex的均值替换了nan

means_multi = df.pivot_table(index=['category', 'servings'], values=['calories', 'carbohydrate', 'sugar', 'protein']) #pivot_table 默认使用均值（mean）作为聚合函数

for index, row in df[df['calories'].isna()].iterrows():
    repl_value = means_multi.loc[(row['category'], row['servings']), ['calories', 'carbohydrate', 'sugar', 'protein']]
    df.loc[index, ['calories', 'carbohydrate', 'sugar', 'protein']] = repl_value
```

```
df.info()
```

```
df.head(10)
```

# EDA

## EDA Summary

From the following graphics we can tell that ...

1. The different types of graphics are showed.
2. Several graphics shwoing more than variables are presented.
3. From these graphics we can tell that:
    1. Breakfast is the category with most recipes.
    2. The count of 'High' traffic is more than the one of 'Low'.
    3. The value distributions of the four nutrients are left-skewness.
    4. The four nutrients are either very weak positive correlation or very weak negative correlation.
    5. The amount of certain nutrients are related to certain category dishes, which align with our commen sense.
4. No single nutrient shows a strong correlation with high traffic, I need to explore combinations of features rather than looking at them individually in next steps.

## Two different types of graphic showing single variables only

```
df['category'].value_counts().plot(kind='bar')
plt.xlabel('Category')
plt.ylabel('Count')
plt.title('Bar Chart of Categories')
plt.show()
```

```
df['high_traffic'].value_counts().plot(kind='bar')
plt.xlabel('Traffic')
plt.ylabel('Count')
plt.title('Bar Chart of Traffic')
plt.show()
```

```
# Assess skewness
def plot_skewness(dataframe):
    fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(8, 6))
    sns.histplot(data=dataframe, x='calories', kde=True, ax=axs[0,0])
    sns.histplot(data=dataframe, x='carbohydrate', kde=True, ax=axs[0,1])
    sns.histplot(data=dataframe, x='sugar', kde=True, ax=axs[1,0])
    sns.histplot(data=dataframe, x='protein', kde=True, ax=axs[1,1])
    plt.tight_layout()
    plt.show()

plot_skewness(df)
```

### Conclusion from these four histgrams

By using distplot to visualize the distribution of calories, carbohydrates, sugars, and proteins, I found that they were all skewed to the left. This means that most values are concentrated on the left side of the distribution, while the right side has fewer values.

This could be important since left-skewness can affect the performance of statistical models, which assumes normally distributed data.

## At least one graphic showing two or more variables

```
# the correlation of these four nutrients
correlation = df[['calories', 'carbohydrate', 'sugar', 'protein']].corr()
correlation
```

```
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Calories, Carbohydrate, Sugar, and Protein')
plt.show()
```

### Conclusion from the nutrients heatmap

The low correlation suggests that these four features do not have strong linear relationships.

```
plt.figure(figsize=(15, 10))
for i, col in enumerate(['calories', 'carbohydrate', 'sugar', 'protein'], 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x='category', y=col, data=df, palette='Set3')
    plt.xticks(rotation=45)
    plt.title(f'{col.capitalize()} by Category')
plt.tight_layout()
plt.show()
```

### conclusion from these box plots

1. Protein-heavy categories (Meat, Chicken) have low carbohydrate and sugar levels, while carbohydrate-heavy categories (Potato, Desserts) have low protein levels.
2. There is a high variability in calorie content across all categories, but especially in Pork, One Dish Meal, and Meat.
3. Desserts and Beverages appear to be major contributors to high sugar intake, while Meat and Chicken are major protein sources.
4. There are significant variations in the nutritional values of individual items.

```
# 将 category 转换为 one-hot 编码
#df = pd.get_dummies(df, columns=['category'], drop_first=True)
df['category_original'] = df['category']
# 进行 one-hot 编码，并添加前缀
df = pd.get_dummies(df, columns=['category'], drop_first=True)

# 将 high_traffic 转换为二进制
df['high_traffic_bin'] = df['high_traffic'].apply(lambda x: 1 if x == 'High' else
0)
```

```
df.head(10)
```

```
corr = df.corr() # high_traffic_bin就是把原本的high和low变成了1和0

# 绘制热图
plt.figure(figsize=(15, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap: Nutritional Features vs Traffic')
plt.show()
```

### insight from the heatmap

There's no single variable show great correlation to the high traffic. Thus, we need to look into the nutrients combined and its relations with category and high traffic.

```
# 使用营养特征进行聚类
X = df[['calories', 'carbohydrate', 'sugar', 'protein']]
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(X)

# 计算每个 cluster 中 High 和 Low 的比例
proportions = df.groupby('cluster')
['high_traffic'].value_counts(normalize=True).unstack()
proportions['High_to_Low_Ratio'] = proportions['High'] / proportions['Low']
print(proportions)  # 打印比例表格

# 画出聚类与 high_traffic 的关系
plt.figure(figsize=(8,6))
ax = sns.countplot(x='cluster', hue='high_traffic', data=df, palette='Set2')

# 在每个 cluster 柱子上显示 High_to_Low Ratio
for cluster in range(len(proportions)):
    high_count = df[(df['cluster'] == cluster) & (df['high_traffic'] ==
'High')].shape[0]
    low_count = df[(df['cluster'] == cluster) & (df['high_traffic'] ==
'Low')].shape[0]
    ratio = proportions.loc[cluster, 'High_to_Low_Ratio']

    # 获取当前柱子的位置
    x_pos = cluster  # x 轴上的位置
    y_pos = max(high_count, low_count) + 10  # 让文字位于最高的柱子上方

    # 显示比例
    ax.text(x_pos, y_pos, f'Ratio: {ratio:.2f}', ha='right', fontsize=12,
 color='black')

# 设置标题
plt.title('Nutrients Cluster Distribution by Traffic (with High/Low Ratio)')
plt.show()
```

Since it's not obvious to find the relation between a single variable and high_traffic. Just tried to cluster the nutrients.

```
# Group by cluster and calculate summary statistics
cluster_nutrient_profiles = df.groupby('cluster')[['calories', 'carbohydrate',
'sugar', 'protein']].describe()

# Display the summary statistics for each cluster
print(cluster_nutrient_profiles)
```

```
# Melt the data for easier plotting
melted = df.melt(id_vars=['cluster'], value_vars=['calories', 'carbohydrate',
'sugar', 'protein'])

# Plot
plt.figure(figsize=(10, 6))
sns.boxplot(x='cluster', y='value', hue='variable', data=melted)
plt.title("Nutrient Profiles by Cluster")
plt.show()
```

### Insights from the cluster box plot

From the nutrients profile and the cluster box plot we can get that:

- Cluster 0: Likely represents everyday, balanced recipes; could target health-conscious users.
- Cluster 1: Represents rich, indulgent recipes; might attract users seeking treats or special occasions.
- Cluster 2: Suggests protein-focused or main meals; could appeal to those seeking substantial or protein-heavy options.

From previous plot, the proportion of each cluster relating to high_traffic, we know that cluster1 and cluster2 are more likely to related to high_traffic. We also need to align each cluster to certain category.

```
# Cross-tabulation to see how clusters align with recipe categories
category_cluster_dist = pd.crosstab(df['category_original'], df['cluster'],
normalize='index')
print(category_cluster_dist)

# Optional: Visualize it
category_cluster_dist.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Recipe Category Distribution by Cluster')
plt.ylabel('Proportion')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

### insights

For this step, we mainly learnt that the relation of cluster and category. Conclusions:

- Cluster 0: Dominated by Beverages, Breakfast, and Vegetables
- Cluster 1: Composed largely of Chicken Breast, Lunch/Snacks, and One Dish Meal.
- Cluster 2: Contains a mix but is notably associated with Meat, Pork, and Potato.

This suggests that certain recipe types naturally group into nutrient-based clusters, nutrient-based clustering captures category-specific nutritional patterns.

```
summary = df.groupby(['cluster', 'category_original',
'high_traffic_bin']).size().unstack(fill_value=0)
summary['total'] = summary.sum(axis=1)
summary['high_traffic_ratio'] = summary[1] / summary['total']
print(summary)
```

```
# Visualize the proportion of high traffic recipes within each cluster and
category
summary['high_traffic_ratio'].unstack().plot(kind='bar', figsize=(12, 6))
plt.title('Proportion of High Traffic Recipes by Cluster and Category')
plt.ylabel('High Traffic Ratio')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

### insights

This step mainly explored the Nutrient-Cluster-Traffic Interactions. Like, which combinations of nutrient profile (cluster) and category are driving high traffic.

- Cluster 2 consistently has a high proportion of high-traffic recipes across almost all categories, indicating that recipes with nutrient profiles characteristic of Cluster 2 tend to be more popular.
- Cluster 1 also shows relatively high traffic ratios, particularly for Vegetables, Potato, and One Dish Meal.
- Cluster 0, by contrast, exhibits lower traffic ratios, especially in categories like Beverages and Breakfast.

Key Insight: Cluster 2 appears to contain the most promising nutrient profiles for high-traffic recipes, while Cluster 1 shows potential, and Cluster 0 seems less likely to drive traffic.

```
category_columns = ['category_Breakfast', 'category_Chicken', 'category_Chicken
Breast', 'category_Dessert', 'category_Lunch/Snacks', 'category_Meat',
'category_One Dish Meal', 'category_Pork', 'category_Potato',
'category_Vegetable']
for col in category_columns:
    contingency_category_traffic = pd.crosstab(df[col], df['high_traffic_bin'])
    # Perform Chi-Squared Test
    chi2, p, dof, expected = chi2_contingency(contingency_category_traffic)
    print(f"{col}  Chi-Squared: {chi2}, p-value: {p}")
```

### conclusion

Chi-Squared Value: 6.50, p-value: 0.0388. Interpretation: The p-value is below 0.05, indicating a **statistically significant relationship between clusters and traffic performance.**

This statistically supports the visual observation that nutrient-based clusters differ in their likelihood of driving traffic.

For the categories, the most statistically significant categories relating to high_traffic are (in order):

1. Vegetable
2. Potato
3. Breakfast
4. Pork
5. Chicken
6. Meat
7. Chicken breast
8. One dish meal

# Model Development

## The type of problem --

This is a binary classification problem.

The objective is to predict whether a recipe will lead to high traffic (high_traffic_bin = 1) or low traffic (high_traffic_bin = 0), based on features such as calories, nutrients, recipe category, and cluster assignments.

Classification is appropriate here because the outcome is categorical with two classes (high or low traffic).

Clustering was used as an unsupervised learning step to group recipes based on their nutritional profiles.

However, the core problem remains a binary classification task, where the goal is to predict recipe traffic performance.

## Analysis of Cluster Characteristics

Investigate what differentiates Cluster 2 and Cluster 1 from Cluster 0 for 'High' traffic.

```
high_traffic_nutrients = df[df['high_traffic_bin'] == 1].groupby('cluster')
[['calories', 'carbohydrate', 'sugar', 'protein']].describe()
print(high_traffic_nutrients)
```

## Fitting a baseline model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

X = df[['calories', 'carbohydrate', 'sugar', 'protein', 'cluster']]
y = df['high_traffic_bin']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

The inital one does not meet expectation.

## Refined a baseline model

```
# Features including clusters, category dummies, and engineered features
X = df[['calories', 'carbohydrate', 'sugar', 'protein', 'cluster']]
X = pd.get_dummies(X, columns=['cluster'], drop_first=True)  # Optional: if
cluster is categorical

# Add categories as dummies
X = X.join(pd.get_dummies(df['category_original'], drop_first=True))

y = df['high_traffic']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model_baseline = LogisticRegression(max_iter=500)
model_baseline.fit(X_train, y_train)

y_pred = model_baseline.predict(X_test)

print(classification_report(y_test, y_pred))
```

### Insights

The goal, mentioned in the report is that the recall of the model for the "high traffic" class (label 1 or 'High') is greater than 80%.

Recall is the proportion of actual high traffic recipes that your model correctly predicts as high traffic.

The refined model meets the goal.

## Fitting a comparison model

The choice of Logistic Regression, Decision Tree, and Random Forest is quite common in binary classification problems

- Logistic Regression → Baseline model. Check if a simple linear model is enough. → Linear relationships, interpretability.
- Decision Tree → Check for non-linear patterns & feature interactions. → Threshold effects, category-based splits.
- Random Forest → Check if ensemble can improve accuracy & robustness. → Final push for performance, reduces overfitting. *If Decision Tree works well, Random Forest is likely to perform even better.*

```
# A decision tree classifier

from sklearn.tree import DecisionTreeClassifier

model_tree = DecisionTreeClassifier(max_depth=4, random_state=42)
model_tree.fit(X_train, y_train)

y_pred_tree = model_tree.predict(X_test)

print(classification_report(y_test, y_pred_tree))
```

```
# Random forest classifier

from sklearn.ensemble import RandomForestClassifier

model_rf = RandomForestClassifier(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train)

y_pred_rf = model_rf.predict(X_test)

print(classification_report(y_test, y_pred_rf))
```

## Model Evaluation

**1. Compare Model Performance**

Based on the results of the three models for high traffic recipe, all three models show satisfied recall value, all above 80%, while RF and LR are better ones.

But LR and DT are better on precision. Besides, LR has the highest f1-score and accuracy. LR is also a simpler model.

In summary, LR is good enough.

**2. Confusion Matrices**

- Logistic Regression (First Confusion Matrix) Recall for High Traffic Recipes is quite good: It correctly predicts 96 out of 113 popular recipes (Recall = 96 / (96 + 17) ≈ 0.85). False Negatives (17 missed popular recipes) are relatively low, which aligns with your goal. False Positives (26 unpopular recipes wrongly shown) are somewhat high, but less critical for the business than missing popular ones. Takeaway: This matrix supports why Logistic Regression is your best model—it minimizes missed popular recipes while keeping a reasonable false positive rate.
- Decision Tree (Second Confusion Matrix) Recall for High Traffic Recipes is slightly worse than Logistic Regression: 94/113 → Recall ≈ 0.83. False Negatives (19 missed popular recipes) are higher than Logistic Regression. False Positives (26 unpopular recipes wrongly shown) is the same as Logistic Regression. Takeaway: Performance is slightly worse than Logistic Regression in terms of Recall, making it a bit less reliable for the business goal.
- Random Forest (Third Confusion Matrix) Recall for High Traffic Recipes is the best: 97/113 → Recall ≈ 0.86. False Negatives (16 missed popular recipes) are the lowest among the models. False Positives (38 unpopular recipes wrongly shown) are the highest, meaning more unpopular recipes will be shown. Takeaway: This model prioritizes catching popular recipes (high Recall), but sacrifices Precision, meaning more unpopular recipes would be shown. If the business were solely Recall-focused, this might be preferred. However, Logistic Regression offers a better balance between catching popular recipes and avoiding unpopular ones.

**Conclusion:** Logistic Regression is still the best overall choice. High Recall (0.85) means few popular recipes are missed. Moderate False Positives mean unpopular recipes are not shown excessively. Business Balance: It achieves the business goal (high Recall) without excessively showing poor recipes.

```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(model_baseline, X_test, y_test)
ConfusionMatrixDisplay.from_estimator(model_tree, X_test, y_test)
ConfusionMatrixDisplay.from_estimator(model_rf, X_test, y_test)
```

## Definition of a metric for the business to monitor

## How should the business monitor what they want to achieve?

**Business Goal** The product team aims to display popular recipes on the homepage, which increases website traffic and subscriptions. The team has requested a model capable of correctly identifying popular (high-traffic) recipes at least 80% of the time, while minimizing the display of unpopular recipes.

**Recommended Performance Metric**

Recall for High Traffic Recipes ---

Definition: The proportion of actual high-traffic recipes that are correctly identified by the model.

Reasoning: This metric is most aligned with the business goal, as missing a popular recipe could lead to a significant drop in website traffic. While precision is also important, the cost of missing a popular recipe outweighs the inconvenience of occasionally showing an unpopular one.

## Estimatetheinitial value(s) for the metric based on the current data

Using the evaluation results from the best-performing model (**Logistic Regression**), we estimate the initial value of this metric:

Recall for High Traffic Recipes: 85% This exceeds the 80% requirement set by the product team and indicates that the model can reliably identify popular recipes for the homepage

# Final Summary & Recommendations

## Summary of Model Evaluation

Three models were developed and compared:

- Logistic Regression (Baseline model)
- 
- Decision Tree Classifier
- 
- Random Forest Classifier

Performance evaluation focused on metrics relevant to the business goal, particularly recall for high-traffic recipes. The confusion matrices and classification reports highlighted the following key insights:

- Logistic Regression achieved the best balance of recall (85%), precision, and overall accuracy.
- 
- Random Forest had the highest recall (86%) but resulted in a higher number of false positives, increasing the risk of displaying unpopular recipes.
- 
- Decision Tree performance was slightly lower than Logistic Regression across all key metrics.

## Recommendations

1. **Logistic Regression** is recommended for deployment. It meets the product team's requirement by achieving 85% recall for high-traffic recipes while maintaining a reasonable precision level, reducing the risk of displaying unpopular recipes.
2. Prioritize High-Traffic Recipe Types:

- Focus on Meat, Chicken, and One Dish Meals.
- Prefer substantial, high-calorie, and high-protein recipes.
- Select family-sized (4 or more servings) recipes over single-serving ones.

3. Monitor Performance: Track Recall for High Traffic Recipes weekly.
4. Enhance Model Features: Incorporate user ratings, preparation time, and ingredient complexity into the model.
5. Retrain the Model Periodically: Every 3 to 6 months.

## Next steps

1. Deploy Logistic Regression Model: Integrate the model into the recipe selection process for the homepage.
2. Monitor Performance: Track the chosen business metric (Recall for High Traffic Recipes) on a weekly basis to ensure the model maintains performance.

3. Data Collection & Feature Enhancement: Consider collecting additional recipe features (e.g., user ratings, preparation time, ingredient complexity) to enhance model accuracy over time.
4. Periodic Model Retraining: Retrain the model every 3 to 6 months as more data becomes available and user preferences evolve.

## ✅ When you have finished...

- Publish your Workspace using the option on the left
- Check the published version of your report:
    - Can you see everything you want us to grade?
    - Are all the graphics visible?
- Review the grading rubric. Have you included everything that will be graded?
- Head back to the **Certification Dashboard** to submit your practical exam report and record your presentation