Supervisor
Assist.prof. Dr.Suhad F.Shihan

Group Members
1. Member One – **Rusul Hayder Abd Zaid**
2. Member Two – **Mustafa Yarub Abdul-Hussein**
3. Member Three **–Mustafa Maytham Mahmood**
4. Member Four – **Noor Abd Alkareem Hadi Salman**

1. Seminar Title

Data Splitting

2. Objective

1. What is Data Splitting? Why Do We Need It?

What is Data Splitting?
Data splitting is the process of dividing your collected dataset into distinct subsets for the specific purposes of **training**, **validating**, and **testing** a machine learning model.

Analogy:
Think of it like studying for a final exam:

- The **textbook** is your **training set** (you learn from it).
- The **practice problems** are your **validation set** (you test your understanding as you learn).
- The **final exam** itself is the **test set** (the ultimate, unbiased test of your knowledge).

3. Abstract

The Core Problem: Overfitting

- **Overfitting** occurs when a model learns the detail and noise in the training data to the extent that it negatively impacts performance on new data. It's **memorizing** instead of **learning**.
- **The Goal:** We want a model that performs well on **unseen data**. This is called **generalization**.

.

4. Introduction

Why We Need Data Splitting:

1. **Model Training (The Training Set):** To allow the model to learn patterns and relationships.
2. **Model Selection & Tuning (The Validation Set):** To evaluate different models or hyperparameters during development without cheating.
3. **Unbiased Evaluation (The Test Set):** For a final, honest assessment of the model's performance on new data. This is the best estimate of real-world performance.
4. **To Prevent Data Leakage:** Ensuring information from the test set does not influence the training process.

5. Literature Review / Background

Types of Data Splitting / Common Methods

The Hold-Out Method

- **Description:** The simplest method. The dataset is randomly split into two or three parts.
- **Common Ratios:** 80% Train / 20% Test, or 70% Train / 15% Validation / 15% Test.
- **Pros:** Simple, fast, and computationally cheap.
- **Cons:** Evaluation can have **high variance**; results depend heavily on the random split. Not ideal for small datasets.

.

6. Methodology

K-Fold Cross-Validation

- **Description:** The dataset is partitioned into *K* equal-sized folds.

- The model is trained *K* times, each time using a different fold as the validation set and the remaining *K-1* folds as the training set.
- The final performance is the **average** of the performance on each of the *K* validation folds.
- **Pros:** Much more reliable and stable estimate of model performance. Makes excellent use of limited data.

- . **Cons:** Computationally expensive (model is trained *K* times).

Stratified K-Fold Cross-Validation

- **Description:** A variation of K-Fold that preserves the original percentage of samples for each **class** in every fold.
- **Why it's important:** Essential for imbalanced datasets. Prevents a scenario where a random fold contains only one class, which would skew the results.

Other Methods:

- **Leave-One-Out Cross-Validation (LOOCV):** K-Fold where *K* = number of samples. Very computationally expensive.
- **Time Series Split:** For time-dependent data. The training set is always from periods before the validation/test set to avoid **look-ahead bias**.

Common Problems during Data Splitting

Problem #1 - Data Leakage

- **The Issue:** Allowing information from the test or validation set to "leak" into the training process.
- **How it happens:** Performing preprocessing (e.g., scaling, imputation) on the entire dataset before splitting. The test data influences the training process.
- **The Result:** Over-optimistic, completely invalid performance metrics. The model will fail in production.

Problem #2 - Improper Splitting Ratio

- **Too little training data:** The model hasn't seen enough examples to learn properly (underfitting).
- **Too little test/validation data:** The evaluation score is not reliable or stable; it has high variance.

Problem #3 - Non-Random and Non-Stratified Splitting

- **Non-Random Splitting:** If data is sorted (e.g., by date) and split without shuffling, it introduces time-based bias.

- **Non-Stratified Splitting:** For classification, can lead to folds with unrepresentative class distributions, misleading your evaluation.

7. Applications

1. Model Training and Learning

- **Application:** This is the most direct use. The **training set** is the textbook from which the algorithm learns patterns, relationships, and correlations between the input features and the target variable.
- **Example:** A neural network learns to identify cats in images by adjusting its weights based on millions of labeled images in the training set.

2. Hyperparameter Tuning and Model Selection

- **Application:** This is a critical application for the **validation set**. You can't use the test set for this, as it would lead to overfitting.
  - **Hyperparameter Tuning:** Finding the optimal settings for an algorithm (e.g., the best max_depth for a Decision Tree, the best learning rate for a neural network). Different hyperparameters are tested, and the model's performance on the validation set determines the winner.
  - **Model Selection:** Comparing different types of algorithms (e.g., Logistic Regression vs. Random Forest vs. SVM) to see which one performs best on your specific problem. The model with the best validation score is chosen.
- **Example:** You train 100 different Random Forest models, each with a different number of trees. The model that performs best on the validation set is selected as your final candidate.

3. unbiased Model Evaluation and Performance Estimation

- **Application:** The **test set** provides a final, honest grade for your model. Because the model has never seen this data during training or tuning, its performance on the test set is the best estimate of how it will perform on new, real-world data.
- **Example:** After developing a credit scoring model, you evaluate it on the held-out test set. An accuracy of 92% on this set is a reliable metric to report to stakeholders, giving them confidence in its deployment.

5. **Preventing Overfitting and Ensuring Generalization**

- **Application:** This is the overarching goal of all the applications above. Data splitting is the primary tool for detecting and preventing overfitting.
  - A large gap between training accuracy (e.g., 99%) and validation/test accuracy (e.g., 80%) is a classic sign of overfitting. The model has memorized the training data but failed to learn generalizable patterns.
- **Example:** A medical diagnosis model performs perfectly on its training data of patient records but fails miserably on the validation set. This signals that the model is overfitting and is not safe to deploy in a hospital. The data split revealed this critical flaw.

5. Mitigating Data-Specific Biases

- **Application:** Techniques like **Stratified Splitting** ensure that your splits are representative of the whole dataset, which is crucial for dealing with imbalanced data.
- **Example:** In a dataset for fraud detection where only 1% of transactions are fraudulent, a random split could accidentally place most fraud cases in the test set. Stratified splitting ensures the 1:99 ratio is maintained in all splits, leading to a fair evaluation.

6. Time Series Forecasting

- **Application:** For time-dependent data, a special type of splitting (e.g., **Time Series Split**) is used where the test set always comes after the training set in time. This simulates the real-world scenario of making predictions for the future based on past data and avoids look-ahead bias.
- **Example:** Predicting next week's stock prices using data from the past three years. You would train on data from January 2020 to December 2022 and test on data from January 2023.

7. Ensemble Methods and Meta-Learning

- **Application:** Advanced techniques like **Bagging** (e.g., Random Forests) inherently use a form of data splitting. They train multiple models on different random subsets (bootstraps) of the training data and then aggregate their predictions, which reduces variance and improves accuracy.

Real-World Industry Applications by Domain:

Domain	Application of Data Splitting
Healthcare	Developing a model to predict patient readmission. The test set performance determines if the model is reliable enough for clinical trials.
Finance	Building a fraud detection system. The validation set is used to tune the model's sensitivity, and the test set provides the final false positive rate.
E-Commerce	Creating a recommendation engine. Splitting user interaction data chronologically to test if the model can accurately predict future purchases.
Automotive	Training a self-driving car's vision system. The test set contains never-before-seen road scenarios to rigorously evaluate safety.
Manufacturing	Predicting machine failure (predictive maintenance). The model is evaluated on a test set of recent failure data to estimate its real-world cost savings.

.

8. Practical application

[1]:
```
#"Importing libraries."
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[2]:
```
df=pd.read_csv("insurance.csv") #I read the file insurance.csv using pandas.
```

[3]:
```
df.head() # I want to check that the file was loaded correctly and display the first 5 rows.
```

[3]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

[4]:
```
df.shape #"I want to check the dimensions of the DataFrame and see the size of the dataset."
```

[4]:
```
(1338, 7)
```

[5]:
```
df.info() #I want to display a summary of the dataset to see column names, data types, and non-missing values."
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

[6]:
```
df.isna().sum() #"I want to check if there are any missing values in the dataset."
```

[6]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

[7]:
```
df.duplicated().sum() #"I want to check for any duplicate rows in the dataset."
```

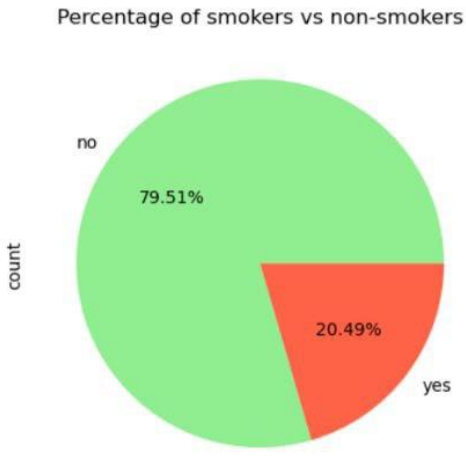
[7]:
```
np.int64(1)
```

[8]:
```
df.drop_duplicates(inplace=True) #Removing duplicate rows is part of data preprocessing
```

[9]:
```
print("Min of age :",df['age'].min())  #"I want to find the minimum and maximum values in the 'age' column."
print("Max of age :",df['age'].max())
```

```
Min of age : 18
Max of age : 64
```

[10]:
```
df['smoker'].value_counts().plot(kind='pie',autopct='%.2f%%',colors=['lightgreen', 'tomato'])
colors=['lightgreen', 'tomato']
plt.title("Percentage of smokers vs non-smokers")   #"To visualize the percentage of smokers and non-smokers in the dataset."
plt.show()
```



[120]:
```
numeric_features = ["age", "bmi", "children"]       #numeric_features
categorical_features = ["sex", "smoker", "region"]  #categorical_features
target = "charges"

# Features and Target
X = df[numeric_features + categorical_features]
y = df[target]
```

[121]:
```
X = df.drop('charges', axis=1)    #"Splitting the data into features and target."
y = df['charges']
```

*[122]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=2)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=2)
print(len(X_train), len(X_val), len(X_test))
```

```
935 201 201
```

*[123]:
```
#"The code performs data preprocessing before training the model, converting numeric and categorical columns into a format suitable for the model."
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(drop="first"), categorical_features)
    ]
)
pipeline = Pipeline(steps=[("preprocessor", preprocessor)])
X_train_processed = pipeline.fit_transform(X_train)

X_test_processed = pipeline.transform(X_test)
print("train", X_train_processed.shape)
print("test", X_test_processed.shape)
```

```
train (935, 8)
test (201, 8)
```

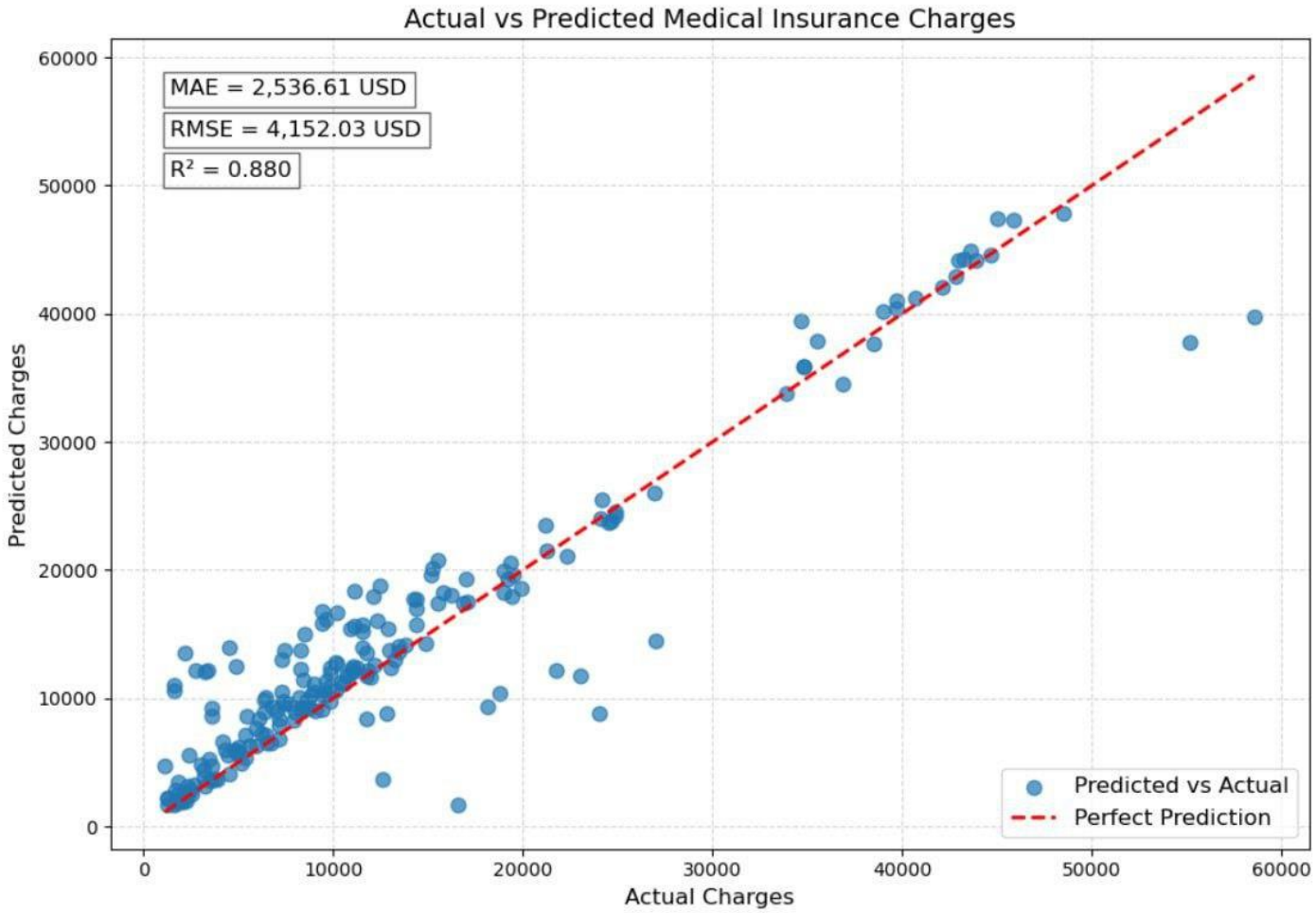
[15]:
```
#"Creating a machine learning model."
rf_model = RandomForestRegressor(
    n_estimators=100,
    random_state=42
)
```

[16]:
```
#"Training the model."
rf_model.fit(X_train_processed, y_train)
```

[16]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

[17]:
```
#"Testing the model."
y_pred = rf_model.predict(X_test_processed)
```

[18]:
```
#Mean (Absolute Error)

#MSE (Mean Squared Error)

#RMSE (Root Mean Squared Error)

#R² (Coefficient of Determination)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

plt.figure(figsize=(10,7))

plt.scatter(y_test, y_pred, alpha=0.7, color='#1f77b4', s=60, label='Predicted vs Actual')

plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         'r--', linewidth=2, label='Perfect Prediction')

plt.text(0.05, 0.95, f'MAE = {mae:,.2f} USD', transform=plt.gca().transAxes, fontsize=12, verticalalignment='top', bbox=dict(facecolor='white', alpha=0.6
plt.text(0.05, 0.90, f'RMSE = {rmse:,.2f} USD', transform=plt.gca().transAxes, fontsize=12, verticalalignment='top', bbox=dict(facecolor='white', alpha=0
plt.text(0.05, 0.85, f'R² = {r2:.3f}', transform=plt.gca().transAxes, fontsize=12, verticalalignment='top', bbox=dict(facecolor='white', alpha=0.6))

plt.xlabel("Actual Charges", fontsize=12)
plt.ylabel("Predicted Charges", fontsize=12)
plt.title("Actual vs Predicted Medical Insurance Charges", fontsize=14)
plt.legend(fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



9. Results & Discussion

. Best Way to Partition Data?

There Is No Single "Best" Way
The best method depends on your **dataset size**, **problem type**, and **computational resources**.

A Practical Guide and Best Practices

1. **Isolate a Hold-Out Test Set First:** The first step is to isolate a portion of your data (**~20%**) as the final **test set**. Lock it away and **do not look at it again** until you are completely done with model development.
2. **Shuffle Your Data:** Always shuffle your data randomly before splitting to avoid order-based biases.
3. **Choose Your Method:**

- **For Most Problems:** Use **Stratified K-Fold Cross-Validation** (for classification) or standard **K-Fold** (for regression) on the remaining data (*K=5* or *K=10* is common).

- **For Very Large Datasets:** A simple **hold-out validation** (e.g., 98% train, 2% validation) is sufficient and faster.
- **For Time Series Data:** Use **Time Series Split**. Never use a random split.

4. **The Golden Rule: Prevent Data Leakage!**

- Any preprocessing must be **fit** on the training fold and then **transformed** on the validation/test fold. Use pipelines to automate this and avoid mistakes.

Cheat Sheet:

- **Large Dataset?** -> Hold-Out
- **Small Dataset?** -> K-Fold Cross-Validation
- **Imbalanced Classes?** -> Stratified K-Fold
- **Time Series Data?** -> Time Series Split
- **GOLDEN RULE:** Isolate Test Set First -> No Data Leakage

10. Conclusion

1. Data splitting is **non-negotiable** for building generalized models.
2. The **Test Set** is sacred—it's for final evaluation only.
3. The **Validation Set** (or CV process) is your workbench for model tuning.
4. **K-Fold Cross-Validation** provides a more robust performance estimate.
5. **Data Leakage** is the most common and dangerous pitfall. Always use pipelines.

11. References

1. Foundational Textbooks & Academic Sources:

- **Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.**

- **Relevance:** This is a canonical text that provides the rigorous statistical foundation for why we need training, validation, and test sets. It offers detailed explanations of the bias-variance tradeoff, overfitting, and cross-validation techniques (including K-Fold). It is considered essential reading for understanding the theory behind model evaluation.

- **James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer.**

- **Relevance:** A more accessible companion to *The Elements of Statistical Learning*. It clearly explains the holdout method, cross-validation, and the importance of separating data for model validation in a way that is perfect for students and practitioners.

- **Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.**
  - **Relevance:** The book's chapters on fundamentals (e.g., Capacity, Overfitting and Underfitting, Hyperparameters and Validation Sets) provide a strong theoretical basis for the concepts of generalization and model evaluation, which are the core reasons for data splitting.
- **Kaufman, S., Rosset, S., Perlich, C., & Stitelman, O. (2012). Leakage in Data Mining: Formulation, Detection, and Avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(4), 1-21.**
  - **Relevance:** This is a seminal paper dedicated entirely to the problem of data leakage. It provides a formal definition, categorizes different types of leakage, and offers strategies for detecting and preventing it, making it the definitive reference for the "Common Problems" section.

2. Practical Guides & Authoritative Courses:

- **Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media.**
  - **Relevance:** A highly practical guide. The book has excellent sections on data preparation, the critical importance of creating a test set immediately, and implementing stratified sampling and cross-validation using scikit-learn. It explains data leakage pitfalls clearly with code examples.
- **Andrew Ng's Machine Learning Course (Coursera / Stanford CS229).**
  - **Relevance:** One of the most popular introductory courses. It famously emphasizes the importance of separating data into training, validation, and test sets for model development and algorithm selection. The concepts of overfitting and the need for a hold-out set for unbiased evaluation are explained with great clarity.

3. Software Documentation & Industry Best Practices:

- **scikit-learn Documentation: Model evaluation: quantifying the quality of predictions**
  - **Relevance:** The official documentation for one of the most widely used ML libraries is a primary source for practical implementation. It provides detailed explanations and code for train_test_split, cross_val_score, KFold, StratifiedKFold, and TimeSeriesSplit. It explicitly warns against data leakage in preprocessing.
- **Google's Rules of Machine Learning (Part of Google's ML Best Practices Guide).**
  - **Relevance:** This guide, born from industry experience, strongly advocates for holding out an evaluation set (test set) from the very beginning. It treats this as a fundamental rule for launching successful ML projects and avoiding the pitfalls of overfitting to automated metrics.

.