Lab Manual

Subject:

Machine Learning (AI-414)

Submitted by:

Yumna Irfan

Registration No:

2023-BS-AI-021

Submitted To:

Sir Saeed

Department:

Artificial Intelligence

Table of Contents

Project 1: Insurance Prediction Using Linear Regression

Project 2: Student Performance Using Classification

Project 1: Insurance Prediction Using Regression

Summary:

The insurance prediction using regression project focuses on building a machine learning model to estimate insurance charges based on various customer attributes such as age, sex, body mass index (BMI), number of children, smoking status, and region. The primary objective is to develop a reliable regression model—typically using algorithms like Linear Regression that can accurately predict insurance costs. The project begins with data preprocessing, including handling missing values, encoding categorical variables, and normalizing numerical features. Exploratory data analysis (EDA) is conducted to understand relationships between variables and their impact on insurance charges. After training and testing multiple regression models, the one with the best performance evaluated using metrics like Mean Squared Error (MSE), and R^2 score is selected. This project not only demonstrates the practical application of regression in the insurance domain but also highlights the importance of data-driven decision-making in risk assessment and pricing strategies.

Main Aim:

- **Predict Insurance Costs**: Develop a model to predict insurance premiums based on customer attributes.
- **Data-Driven Pricing**: Assist insurance companies in setting fair and accurate premiums based on risk factors.
- **Risk Assessment**: Use customer demographics and health information to assess individual risk levels.
- **Improve Profitability**: Help insurance firms make data-backed decisions to optimize their pricing models and ensure profitability.
- **Enhance Customer Satisfaction**: Provide more personalized and fair pricing for customers based on their specific characteristics.
- **Machine Learning Application**: Demonstrate how regression techniques can be applied to real-world business problems, specifically in the insurance industry.

Abstract:

The insurance prediction project uses regression models to predict insurance premiums based on customer attributes like age, gender, BMI, and smoking status. The goal is to build an accurate model that assists insurance companies in setting fair, data-driven premiums. The process involves data preprocessing, exploratory data analysis, and applying various regression algorithms such as Linear Regression. Model performance is evaluated using metrics like MSE, and R^2 score. Ultimately, the project aims to improve pricing strategies, risk assessment, and customer satisfaction through predictive insights.

Explanation of Steps:

1. Load insurance dataset and explore structure.
2. Explore Dataset
3. Identifying Numerical and Categorical Columns
4. Feature Scaling
5. Testing and Training
6. Building and evaluating Linear Regression Model
7. Visualize model predictions.

Explanation of Libraries:

Numpy:

NumPy is a powerful library in Python for numerical computing. It provides support for arrays, matrices, and a large collection of mathematical functions to operate on these arrays. It's one of the foundational libraries in Python, especially in data science, machine learning, and scientific computing.

NumPy arrays are faster and more memory-efficient than traditional Python lists. This is because they are implemented in C and allow for element-wise operations

directly on the data. You can do things like addition, subtraction, multiplication, etc., across entire arrays without using explicit loops.

Broadcasting is a feature in NumPy that allows you to perform operations on arrays of different shapes. The smaller array is "broadcast" across the larger array to match the dimensions.

Pandas:

Pandas is another fundamental library in Python, especially in data science and machine learning. It provides powerful data structures to handle, manipulate, and analyze data.

DataFrame (2D array-like):

A DataFrame is a two-dimensional table, and it's the core data structure in Pandas. It's similar to a spreadsheet or a SQL table, where data is organized in rows and columns.

Basic Operations:

- **Accessing Columns**: You can access a column using its name as an attribute or as a key.
- **Accessing Rows**: You can access rows using .iloc[] (integer-based indexing) or .loc[] (label-based indexing).

Data Manipulation:

Pandas provides powerful tools to manipulate data, including:

- **Filtering**: You can filter rows based on conditions.
- **Sorting**: You can sort your DataFrame by column values.
- **Adding/Removing Columns**: You can easily add or drop columns.

Missing Data:

Pandas has built-in methods to handle missing data.

Scikit-Learn:

Scikit-learn is one of the most popular and widely-used libraries for machine learning in Python. It provides simple and efficient tools for data mining, statistical modeling, and predictive data analysis. It's built on top of NumPy, SciPy, and matplotlib, which allows it to integrate seamlessly into the broader scientific Python ecosystem.

Key Features of Scikit-learn:

1. **Supervised Learning**: Scikit-learn includes algorithms for classification and regression, where you have labeled data.
2. **Unsupervised Learning**: It also includes tools for clustering, dimensionality reduction, and anomaly detection.
3. **Model Selection**: Functions for splitting data into training/testing sets, cross-validation, and hyperparameter tuning.
4. **Preprocessing**: It provides utilities for feature scaling, encoding categorical variables, handling missing data, and more.
5. **Evaluation**: Includes metrics to evaluate model performance like accuracy, precision, recall, confusion matrix, etc.
6. **Pipeline**: Support for combining multiple steps of a machine learning process (like preprocessing, training, etc.) into a single object, which makes model building more efficient and easy to reproduce.

Model Evaluation and Selection:

Scikit-learn provides a range of evaluation metrics and tools for model validation, such as:

- **Train-test split**: Split data into training and testing subsets using train_test_split.
- **Cross-validation**: Evaluate the model on different subsets of data using cross_val_score.
- **Metrics**: Use different metrics to assess model performance, like accuracy_score, confusion_matrix, mean_squared_error, etc.

Preprocessing:

Scikit-learn also includes various preprocessing tools to clean and prepare your data:

- **Scaling**: Standardize or normalize the features (e.g., StandardScaler, MinMaxScaler).
- **Encoding**: Convert categorical variables to numerical values (e.g., LabelEncoder, OneHotEncoder).
- **Imputation**: Handle missing values (e.g., SimpleImputer).

Project Code:

Importing Libraries:

DATA PREPROCESSING

1. Import necessary libraries

```
[5]: import pandas as pd
     import numpy as np
     from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHotEncoder
     from sklearn.model_selection import train_test_split
     from sklearn.impute import SimpleImputer
     import matplotlib.pyplot as plt
```

Reading Data:

2. Load dataset and Showing Dataset Dimesnsion

```
[7]: df = pd.read_csv('insurance.csv')
     df.shape

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

Data Exploration:

3. Explore the data

[9]:
```
df.sample()
```

[9]:

	age	sex	bmi	children	smoker	region	charges
731	53	male	21.4	1	no	southwest	10065.413

[10]:
```
df.head()
```

[10]:

	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

[11]:
```
df.tail()
```

[11]:

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

Information and Datatype Check:

Checking Information and Datatypes

[13]:
```
df.info()
```

```
<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
```

[14]:
```
df.dtypes
```

[14]:
```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

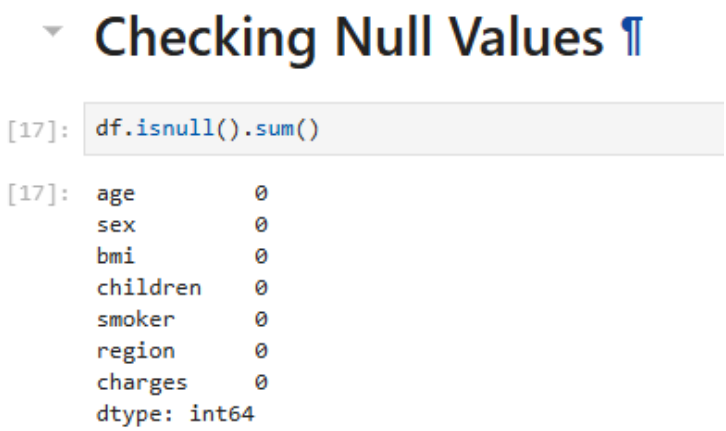
[15]:
```
df.describe()
```

[15]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

Data Cleaning:

Data cleaning involves identifying and correcting errors, inconsistencies, and missing values in a dataset to improve its quality for analysis.



Outliers:

Outliers are data points that significantly differ from the rest of the data in a dataset. They are values that are much higher or lower than most of the other values and can distort statistical analyses and machine learning models.

Boxplot:

A **boxplot** (also known as a **box-and-whisker plot**) is a graphical representation of the distribution of a dataset that provides a visual summary of key statistical measures. It helps in identifying outliers and understanding the spread of data, central tendency, and variability.

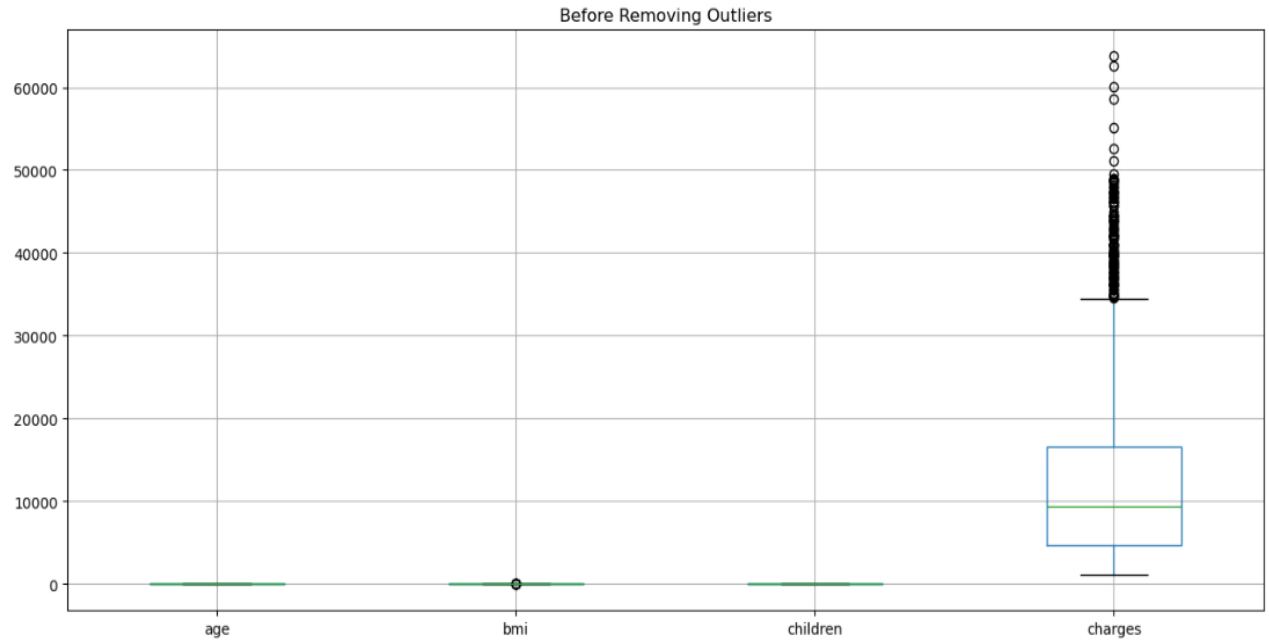
4. Identifying Numerical and Categorical Columns

Checking Outliers

[19]:
```
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
mask = pd.Series(True, index=df.index)

for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    mask = mask & df[col].between(lower_bound, upper_bound)

data_cleaned = df[mask]
```

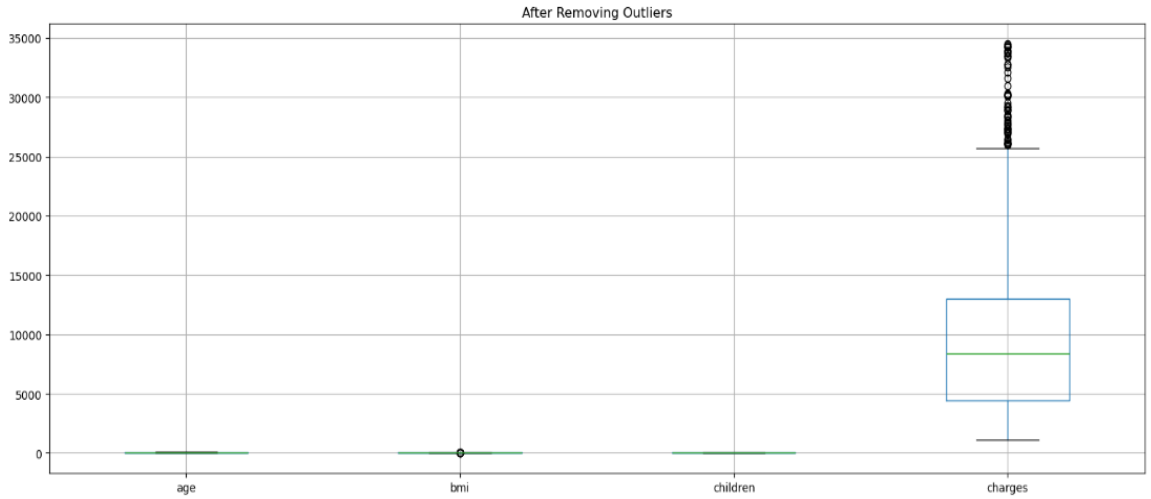
[20]:
```
plt.figure(figsize=(25,6))
plt.subplot(1,2,1)
df.boxplot()
plt.title("Before Removing Outliers")
plt.tight_layout()
plt.show()
```

Before Removing Outliers



After Removing Outliers:

```
[21]: plt.figure(figsize=(30,6))
      plt.subplot(1,2,1)
      data_cleaned.boxplot()
      plt.title("After Removing Outliers")
      plt.tight_layout()
      plt.show()
```




```
[22]: cat_cols = df.select_dtypes(include=['object']).columns
```

Imputing Categorical and Numerical Columns:

Imputing Numerical and Categorical Columns refers to the process of filling in missing values in your dataset, which is crucial for maintaining the integrity and quality of the data. There are various strategies for imputing missing values depending on whether the column is **numerical** or **categorical**.

Impute numerical columns with mean

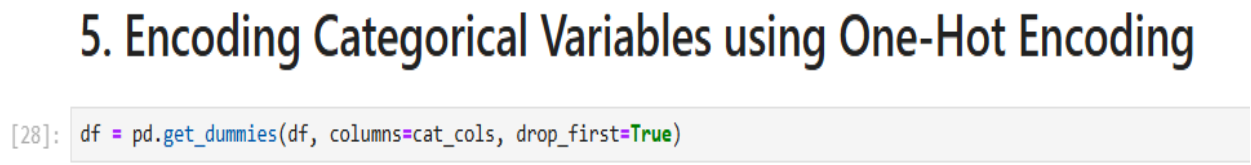
```
[24]: num_imputer = SimpleImputer(strategy='mean')
      df[num_cols] = num_imputer.fit_transform(df[num_cols])
```

Impute categorical columns with most frequent

```
[26]: cat_imputer = SimpleImputer(strategy='most_frequent')
      df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])
```

One-Hot Encoding:

One-Hot Encoding is a method of converting categorical variables (i.e., variables that contain labels or categories) into a binary format. This is done by creating new columns that represent each category as a binary (0 or 1) value, making it easier for machine learning algorithms to process categorical data.



Feature Scaling:

Feature scaling is the process of normalizing or standardizing numerical features in a dataset to ensure they are on a comparable scale. This step is essential for many machine learning algorithms that rely on the magnitude of the input features, such as gradient descent-based algorithms (like linear regression, logistic regression, and neural networks), distance-based algorithms (like k-nearest neighbors and k-means clustering), and principal component analysis (PCA).

Standardization:

Standardization is a technique used to scale the features of your data such that each feature has a mean of 0 and a standard deviation of 1. It is often used when the features in your dataset have different units or scales, or when you're working with algorithms that assume the data follows a Gaussian distribution (such as Linear Regression, Logistic Regression, and PCA).

How Does Standardization Work?

Standardization transforms the data using the following formula:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

6. Feature Scaling

Standardization

[30]:
```
scaler = StandardScaler()
df[num_cols] = scaler.fit_transform(df[num_cols])
```

Splitting Features:

7. Splitting Features and Target Variable

[32]:
```
X = df.drop('charges', axis=1)
y = df['charges']
```

Testing and Training:

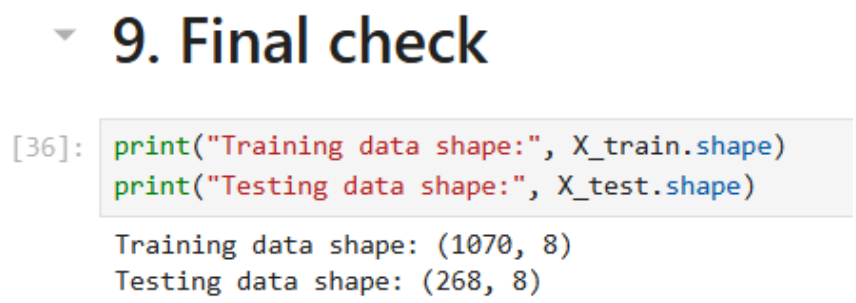
The **train-test split** is a fundamental concept in machine learning, used to evaluate the performance of your model. It involves dividing your data into two subsets:

1. **Training Set**: The portion of the data used to train the model.
2. **Testing Set**: The portion of the data used to evaluate the model's performance.

8. Train-test split

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Final Checking:



Linear Regression:

Linear Regression is one of the simplest and most widely used techniques in statistical modeling and machine learning. It models the relationship between a dependent variable (target) and one or more independent variables (features) by fitting a linear equation to observed data.

Key Concept:

Linear regression aims to find the best-fitting straight line (in 2D) or hyperplane (in higher dimensions) that represents the relationship between the target variable (denoted as **Y**) and the feature(s) (denoted as **X**).

Formula:

For a **simple linear regression** (one feature), the relationship is modeled as:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Types of Linear Regression:

1. **Simple Linear Regression**: Models the relationship between one independent variable and one dependent variable.
2. **Multiple Linear Regression**: Models the relationship between two or more independent variables and one dependent variable.

Working:

1. Initialization**:**
2. Calculate Error:
3. Adjust the Coefficients**:**
4. Gradient Descent**:**

Building Linear Regression Model ¶

[38]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Initialize model and Training model

[40]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

[40]:
```
LinearRegression
LinearRegression()
```

Model Prediction

[42]:
```
y_pred = model.predict(X_test)
```

Model Evaluation Metrics

[44]:
```
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 0.22926355667538661
R² Score: 0.7835929767120722
```

Data Visualization:

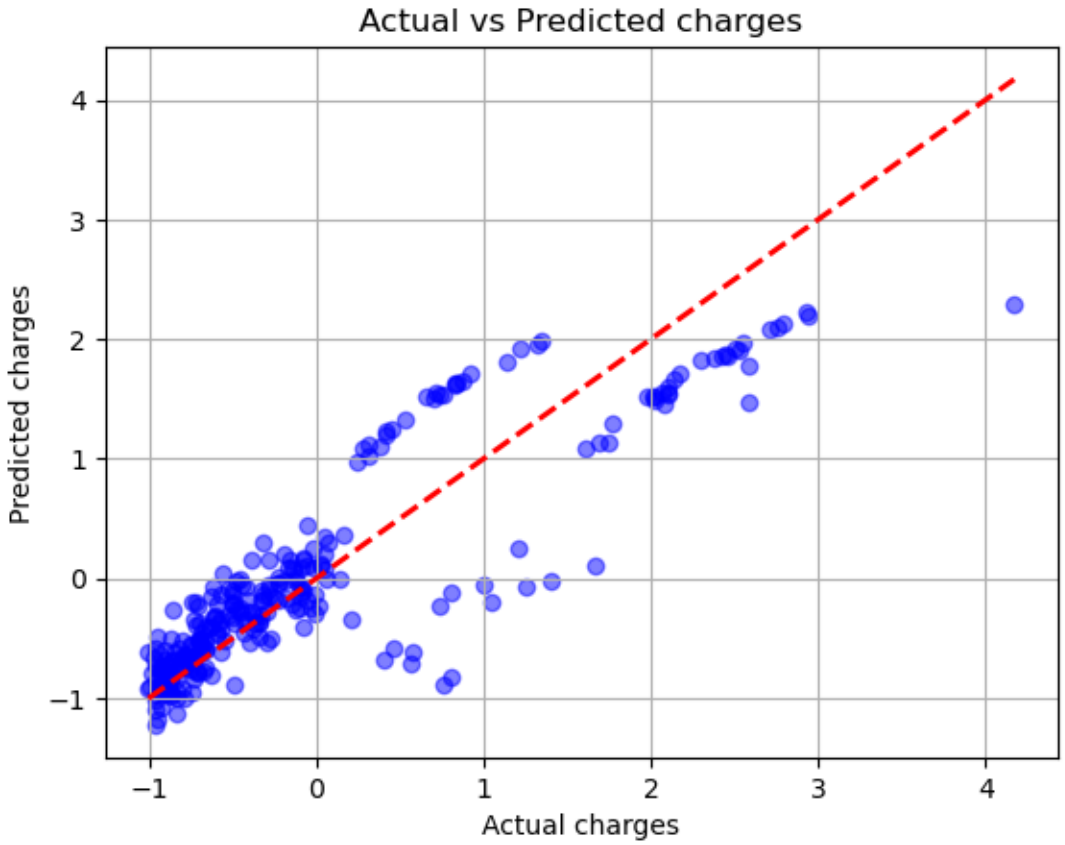
Showing actual vs predicted values of a multiple regression model.

Data Visualization Setup

[46]:
```
import matplotlib.pyplot as plt
```

Actual vs Predicted Plot

[48]:
```
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.ylabel('Predicted charges')
plt.xlabel('Actual charges')
plt.title('Actual vs Predicted charges')
plt.grid(True)
plt.show()
```



Conclusion:

The analysis of the insurance dataset reveals that insurance charges, the dependent variable, are significantly influenced by several independent variables. Age is a critical factor, with older individuals generally facing higher charges due to increased health risks. BMI also shows a strong correlation, as higher values often lead to greater medical costs. Smoking status has a dramatic impact, with smokers paying substantially more than non-smokers, reflecting the associated health risks.

Gender plays a role, with males typically incurring higher charges, possibly due to riskier behaviors or biological factors. The number of children affects costs, as more dependents may require additional coverage. Regional differences further influence charges, with areas like the Southeast showing higher premiums, likely due to varying healthcare costs and disease prevalence.

The dataset highlights the complexity of predicting insurance costs, as interactions between variables—such as age, BMI, and smoking—create nonlinear relationships. Machine learning models like regression or decision trees could effectively capture these patterns. Feature engineering, such as combining age and BMI, might improve accuracy, while handling outliers ensures robust predictions.

Categorical variables like region and smoking status require proper encoding, while numerical features like BMI benefit from normalization. The dataset also suggests potential biases, such as uneven regional representation, which must be addressed to avoid skewed results.

In short, insurance charges are shaped by multiple factors, and predictive models must account for these relationships. Future research could incorporate additional variables, such as medical history, to refine predictions further. This analysis provides a foundation for developing accurate pricing models, helping insurers and policymakers make informed decisions.

Project 2: Student Performance Using Classification

Summary:

This dataset contains 500 student records with features like study hours, attendance, past scores, parental education, and internet access, used to predict pass/fail outcomes. Key predictors likely include study time, attendance, and prior academic performance. The final exam score may need exclusion to prevent data leakage. Suitable for binary classification models (e.g., Random Forest) to identify at-risk students and improve academic interventions. Analysis should check for class balance and feature correlations before modeling.

Main Purpose:

1. **Predict Student Performance**: Classify whether students will pass or fail based on their academic and demographic data.
2. **Identify Key Influencing Factors**: Determine which variables (study hours, attendance, past scores, etc.) most impact academic success.
3. **Support Early Intervention:** Help educators identify at-risk students for timely support and improved outcomes.
4. **Evaluate Educational Policies**: Assess how factors like parental education and internet access affect performance.
5. **Build a Classification Model**: Train machine learning models (e.g., Logistic Regression, Random Forest) to automate pass/fail predictions.

Abstract:

This research develops a machine learning model to predict student pass/fail outcomes using academic and demographic data from 500 records. Key features include study hours, attendance, past performance, parental education, and internet access. The study identifies critical success factors while addressing potential data leakage from final exam scores. Using classification algorithms like Random Forest

the model aims to support early intervention for at-risk students. Analysis reveals how socioeconomic factors influence academic achievement alongside traditional metrics. The findings offer actionable insights for educational institutions to improve student outcomes through targeted support strategies, contributing to evidence-based educational policymaking.

Steps:

1. Importing Libraries and Reading Data.
2. Data Exploration.
3. Defining Target Column and Checking Class Distribution.
4. Numerical and Categorical Column Identification.
5. One-Hot Encoding.
6. Scaling Features.
7. Testing and Training.
8. Random Forest Classifier.
9. Data Visualization and Random Forest Confusion Matrix.
10. Building and Training ANN Model.
11. Model Prediction and Evaluation.
12. ANN Confusion Matrix.

Project Code:

Importing Libraries:

DATA PREPROCESSING

1. Import necessary libraries

[5]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
```

Reading Dataset and Showing Dimension:

2. Load dataset

```
[7]: df = pd.read_csv('student_performance_dataset.csv')
     df.shape
```

```
[7]: (708, 10)
```

Data Exploration:

3. Explore the data

```
[9]: df.head()
```

[9]:

	Student_ID	Gender	Study_Hours_per_Week	Attendance_Rate	Past_Exam_Scores	Parental_E
0	S147	Male	31	68.267841	86	
1	S136	Male	16	78.222927	73	
2	S209	Female	21	87.525096	74	
3	S458	Female	27	92.076483	99	
4	S078	Female	37	98.655517	63	

```
[10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708 entries, 0 to 707
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Student_ID                 708 non-null    object
 1   Gender                     708 non-null    object
 2   Study_Hours_per_Week       708 non-null    int64
 3   Attendance_Rate            708 non-null    float64
 4   Past_Exam_Scores           708 non-null    int64
 5   Parental_Education_Level   708 non-null    object
 6   Internet_Access_at_Home    708 non-null    object
 7   Extracurricular_Activities 708 non-null    object
 8   Final_Exam_Score           708 non-null    int64
 9   Pass_Fail                  708 non-null    object
dtypes: float64(1), int64(3), object(6)
memory usage: 55.4+ KB
```

```
[11]: df.describe()
```

[11]:

	Study_Hours_per_Week	Attendance_Rate	Past_Exam_Scores	Final_Exam_Score
count	708.000000	708.000000	708.000000	708.000000
mean	26.132768	78.107722	77.871469	58.771186
std	8.877727	13.802802	14.402739	6.705877
min	10.000000	50.116970	50.000000	50.000000
25%	19.000000	67.550094	65.000000	52.000000
50%	27.000000	79.363046	79.000000	59.500000
75%	34.000000	89.504232	91.000000	64.000000
max	39.000000	99.967675	100.000000	77.000000

Data Cleaning:

```
[12]: df.isnull().sum()
```

```
[12]: Student_ID                  0
      Gender                      0
      Study_Hours_per_Week        0
      Attendance_Rate             0
      Past_Exam_Scores            0
      Parental_Education_Level    0
      Internet_Access_at_Home     0
      Extracurricular_Activities  0
      Final_Exam_Score            0
      Pass_Fail                   0
      dtype: int64
```

```
[13]: df.columns
```

```
[13]: Index(['Student_ID', 'Gender', 'Study_Hours_per_Week', 'Attendance_Rate',
             'Past_Exam_Scores', 'Parental_Education_Level',
             'Internet_Access_at_Home', 'Extracurricular_Activities',
             'Final_Exam_Score', 'Pass_Fail'],
            dtype='object')
```

Target Column for Prediction:

4. Defining the Target Column for Prediction

```
[15]: target_col=df['Pass_Fail']
```

Checking Class Distribution in the Target Column

```
[17]: target_col.value_counts()
```

```
[17]: Pass_Fail
      Pass    354
      Fail    354
      Name: count, dtype: int64
```

Dropping Columns:

Dropping Unnecessary Columns from the DataFrame

```
[19]: df = df.drop(['Student_ID', 'Gender', 'Pass_Fail'], axis=1)
```

Simple Imputer:

A SimpleImputer is a class from the sklearn.impute module in Python, used to handle missing data in datasets. When working with real-world data, it's common to encounter missing values. The SimpleImputer allows you to fill or "impute" these missing values using a variety of strategies.

5. Identifying Numerical and Categorical Columns in the DataFrame

```
[21]: num_cols = df.select_dtypes(include=['int64', 'float64']).columns
      cat_cols = df.select_dtypes(include=['object']).columns
```

Impute numerical columns with mean

```
[23]: num_imputer = SimpleImputer(strategy='mean')
      df[num_cols] = num_imputer.fit_transform(df[num_cols])
```

Impute categorical columns with most frequent

```
[25]: cat_imputer = SimpleImputer(strategy='most_frequent')
      df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])
```

One-Hot Encoding:

6. One-Hot Encoding Categorical Columns and Previewing Transformed Data

[27]:
```
df = pd.get_dummies(df, columns=cat_cols, drop_first=True)
df.head(2)
```

[27]:

	Study_Hours_per_Week	Attendance_Rate	Past_Exam_Scores	Final_Exam_Score	Parental_Education_Level_High School	Parental_Education_Level_Masters	Parental_Education_L
0	31.0	68.267841	86.0	63.0	True	False	
1	16.0	78.222927	73.0	50.0	False	False	

Defining Variables:

Defining Features (X) and Target Variable (y)

[33]:
```
X = df
y = target_col
```

[34]:
```
X
```

[34]:

	Study_Hours_per_Week	Attendance_Rate	Past_Exam_Scores	Final_Exam_Score	Parental_Education_Level_High School	Parental_Education_Level_Masters
0	0.548640	-0.713394	0.564773	0.631059	True	False
1	-1.142177	0.008352	-0.338471	-1.308910	False	False
2	-0.578571	0.682762	-0.268991	-0.562768	False	False
3	0.097755	1.012739	1.468017	0.929516	False	False
4	1.224966	1.489721	-1.033274	1.675657	False	True
...	...	...	...	...	...	...
703	-1.367619	0.474952	0.008930	-1.308910	False	False
704	0.999524	-1.292588	0.356332	0.481831	False	True
705	-0.127687	1.470106	-0.199511	-0.264311	False	False
706	-0.578571	1.307926	0.425812	0.929516	False	False
707	-0.465850	0.166509	1.051135	-0.562768	False	False

708 rows × 9 columns

[35]:
```
y
```

[35]:
```
0      Pass
1      Fail
2      Fail
3      Pass
4      Pass
       ...
703    Fail
704    Pass
705    Fail
706    Pass
707    Fail
Name: Pass_Fail, Length: 708, dtype: object
```

Splitting and Final Check:

8. Train-test split

```
[37]: X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size=0.2, random_state=42
      )
```

9. Final check

```
[39]: print("Training data shape:", X_train.shape)
      print("Testing data shape:", X_test.shape)
```

```
Training data shape: (566, 9)
Testing data shape: (142, 9)
```

Random Forest Classifier and Model Evaluation:

Importing Libraries for Model Training and Evaluation

```
[41]: from sklearn.model_selection import train_test_split
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import accuracy_score, classification_report
```

Training the Random Forest Classifier Model

```
[44]: model = RandomForestClassifier(random_state=42)
      model.fit(X_train, y_train)
```

```
[44]: RandomForestClassifier
      RandomForestClassifier(random_state=42)
```

Model Performance and Accuracy:

Evaluating Model Performance on Test Data

[46]:
```
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

        Fail       1.00      1.00      1.00        71
        Pass       1.00      1.00      1.00        71

    accuracy                           1.00       142
   macro avg       1.00      1.00      1.00       142
weighted avg       1.00      1.00      1.00       142
```

Displaying Model Accuracy

[48]:
```
print(X_train.shape, X_test.shape)
print("Train Accuracy:", model.score(X_train, y_train))
print("Test Accuracy:", model.score(X_test, y_test))
```

```
(566, 9) (142, 9)
Train Accuracy: 1.0
Test Accuracy: 1.0
```

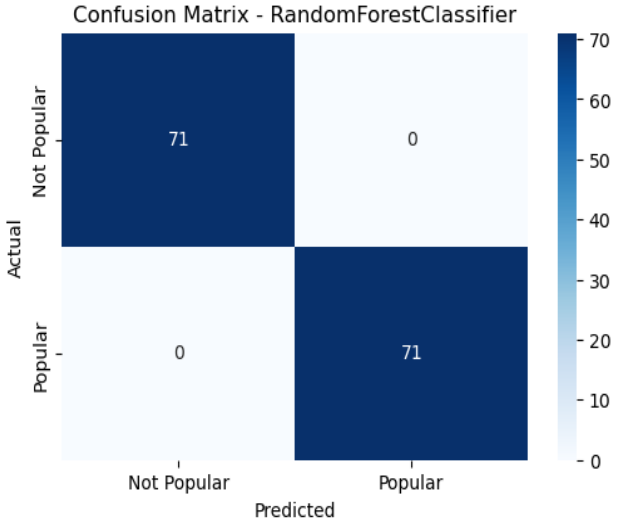
Importing Libraries for Data Visualization:

[50]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

Random Forest Confusion Matrix:

Random Forest: Confusion Matrix

[52]:
```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Popular', 'Popular'], yticklabels=['Not Popular', 'Popular'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - RandomForestClassifier')
plt.show()
```



Libraries for Neural Network:

Libraries for Building and Evaluating a Neural Network Model

[53]:
```
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
```

Label Encoding and Identification Of Categorical Columns:

Applying Label Encoding to the Target Variable

```
[55]: le = LabelEncoder()
      y = le.fit_transform(y)
```

Identifying Categorical Columns in the Feature Set

```
[57]: categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
```

Applying One-Hot Encoding to Categorical Features

```
[59]: preprocessor = ColumnTransformer(
          transformers=[('cat', OneHotEncoder(drop='first', handle_unknown='ignore'), categorical_cols)],
          remainder='passthrough'
      )
```

Data Transformation

Data transformation applies preprocessing steps like scaling, encoding, and handling missing values to prepare data for model training.

Data Transformation

```
[61]: X_processed = preprocessor.fit_transform(X)
```

Scaling and Feature Splitting:

Scaling Features Using StandardScaler

```
[63]: scaler = StandardScaler()
      X_scaled = scaler.fit_transform(X_processed)
```

Train-test split

```
[65]: X_train, X_test, y_train, y_test = train_test_split(
          X_scaled, y, test_size=0.2, random_state=42
      )
```

Artificial Neural Network:

An **Artificial Neural Network (ANN)** is a computational model inspired by the human brain, consisting of layers of interconnected nodes (neurons). It typically has three types of layers: an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to neurons in the previous and next layers, with each connection having a weight. During training, the network learns by adjusting these weights based on the error between predicted and actual outputs, using techniques like backpropagation. ANNs are used for tasks such as classification, regression, and pattern recognition.

Building ANN

```
[67]: model = Sequential([
          Input(shape=(X_train.shape[1],)),
          Dense(64, activation='relu'),
          Dense(32, activation='relu'),
          Dense(1, activation='sigmoid')
      ])
```

Model Compilation:

Model Compilation in neural networks refers to configuring the model for training by specifying the optimizer, loss function, and evaluation metrics.

Model Compilation

```
[69]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Model Training:

Training model

```
[71]: model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=1)
Epoch 1/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7877 - loss: 0.5924
Epoch 2/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9079 - loss: 0.4295
Epoch 3/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9099 - loss: 0.3240
Epoch 4/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9458 - loss: 0.2150
Epoch 5/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9295 - loss: 0.1967
Epoch 6/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9502 - loss: 0.1597
Epoch 7/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9559 - loss: 0.1224
Epoch 8/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9565 - loss: 0.1129
Epoch 9/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9793 - loss: 0.0912
Epoch 10/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9844 - loss: 0.0790
Epoch 11/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9833 - loss: 0.0697
Epoch 12/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9933 - loss: 0.0550
Epoch 13/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9857 - loss: 0.0593
Epoch 14/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9896 - loss: 0.0516
Epoch 15/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9885 - loss: 0.0500
Epoch 16/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9944 - loss: 0.0405
Epoch 17/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9975 - loss: 0.0357
Epoch 18/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9946 - loss: 0.0391
Epoch 19/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9984 - loss: 0.0296
Epoch 20/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0295
[71]: <keras.src.callbacks.history.History at 0x17218699790>
```

Model Prediction and Evaluation:

Predictions and Evaluation

```
[73]: y_pred_prob = model.predict(X_test)
      y_pred = (y_pred_prob > 0.5).astype(int).flatten()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

Evaluating Model Performance

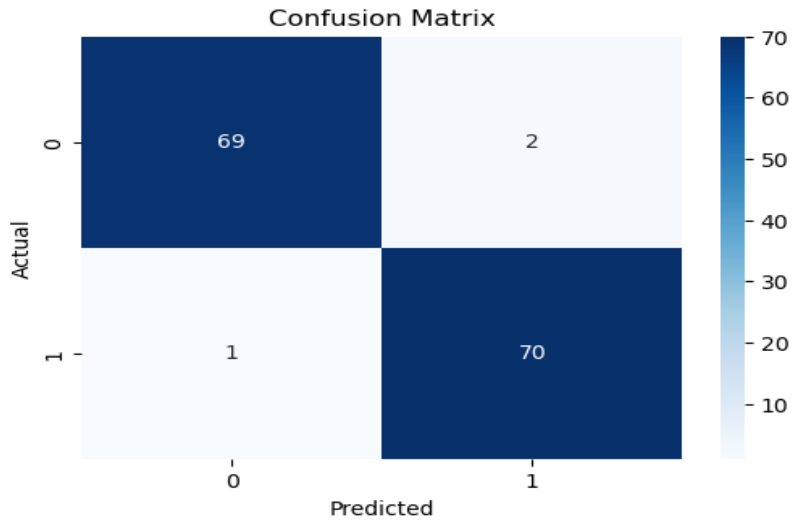
```
[75]: print("Accuracy:", accuracy_score(y_test, y_pred))
      print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9788732394366197
Classification Report:
                precision    recall  f1-score   support

           0       0.99      0.97      0.98        71
           1       0.97      0.99      0.98        71

    accuracy                           0.98       142
   macro avg       0.98      0.98      0.98       142
weighted avg       0.98      0.98      0.98       142
```

ANN Confusion Matrix:

ANN Confusion Matrix

```
[77]: cm = confusion_matrix(y_test, y_pred)

      plt.figure(figsize=(6, 4))
      sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
      plt.xlabel('Predicted')
      plt.ylabel('Actual')
      plt.title('Confusion Matrix')
      plt.show()
```



Conclusion:

The dataset offers a comprehensive view of factors influencing student performance, with Pass-Fail as the dependent variable and various attributes as independent variables. Key findings show that study hours, attendance rate, past exam scores, parental education, internet access, and extracurricular activities significantly impact student outcomes.

Students who study more (30+ hours per week) tend to perform better, though there are exceptions, indicating study hours alone aren't enough. Higher attendance rates (above 80%) correlate with better performance, while lower attendance leads to higher failure rates. Past exam scores are also strong predictors of success.

Parental education level impacts performance, with students of highly educated parents generally performing better. Internet access at home provides an advantage, though it’s not universally true. Extracurricular activities show mixed results, with both successes and failures depending on the type and intensity. Interestingly, students involved in certain activities like sports or arts seem to be more engaged, but others may be overburdened, leading to poor outcomes.

Gender doesn’t show a clear pattern in determining outcomes. The dataset reveals outliers, such as students with high study hours or past exam scores who still fail, potentially due to unmeasured factors like stress or teaching quality.

In short, student performance is shaped by a combination of study habits, attendance, past performance, and support systems. No single factor guarantees success, but a holistic approach addressing both academic and environmental factors is crucial for improving outcomes. Schools should consider these insights to design interventions that promote better attendance, home study resources, and balanced extracurricular engagement. Further analysis could examine the potential role of social support networks, peer influences, or mental health factors in student success. Addressing these elements could lead to more personalized and effective educational strategies.