Data Analysis

Data analysis inspects, cleans, transforms, and models data to discover useful information, inform conclusions, and support decision-making. It can be classified into two main categories:

- **Descriptive analysis**: Summarizes historical data to understand what has happened.
- **Predictive analysis**: Uses past data to predict future events.
- **Prescriptive analysis**: Suggests actions to achieve desired outcomes.

Ex:

A retail company analyzes its historical sales data to understand the seasonal trends and predict the next quarter's sales.

Exploratory Data Analysis (EDA)

EDA is a preliminary step in data analysis used to summarize the main characteristics of the dataset, often with visual tools. It helps in understanding data distribution, relationships between variables, and identifying patterns.

Key Components of EDA:

- **Introduction**: EDA helps you to see what the data can tell you beyond the formal modeling or hypothesis testing task. It includes understanding the data types and summarizing data distributions.
- **Exploring Relationships and Patterns**: EDA includes discovering correlations between variables, outliers, and patterns such as trends or seasonality.

Suppose you have a dataset of house prices and want to explore relationships between features like square footage, location, and price.

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Load dataset

df = pd.read_csv('/content/Housing.csv')

df.describe()
```

```
# Scatter plot to explore relationship between square footage and price
sns.scatterplot(x='area', y='price', data=df)
plt.show()

# Correlation heatmap
# Include only numerical features for correlation analysis
numerical_df = df.select_dtypes(include=['number'])
sns.heatmap(numerical_df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

The scatter plot and heatmap help you see relationships between the size of the house and its price and identify which features are most strongly correlated.

Feature Engineering and Selection

Feature Engineering is creating or modifying new features to improve model performance. **Feature Selection** helps select the most relevant features to train a model effectively, removing irrelevant or redundant data.

Example: Predicting House Prices

We will walk through an example where we perform feature engineering and selection on a dataset of house prices to predict the cost of a house using Python and machine learning libraries like pandas, scikit-learn, and matplotlib.

A simplified version of a housing dataset containing features such as area, bedrooms, and bathrooms.

```
import pandas as pd

# Loading the dataset
data = {'area': [1500, 2000, 2500, 3000, 3500],
        'bedrooms': [3, 4, 3, 5, 4],
```

```
    'bathrooms': [2, 3, 2, 4, 3],
    'year_built': [1990, 2005, 2010, 2015, 2020],
    'price': [300000, 400000, 350000, 500000, 450000]}

df = pd.DataFrame(data)
print(df)
```

Feature Engineering

1. **Creating New Features**: One helpful feature might be the **age** of the house (derived from the year_built feature).

```
import datetime

current_year = datetime.datetime.now().year
df['age'] = current_year - df['year_built']
print(df[['year_built', 'age']])
```

Now, we have a new feature, age, which might be more relevant to predicting price than the raw year_built.

Combining Features: You can combine features like bedrooms and bathrooms into a new feature called total_rooms.

```
df['total_rooms'] = df['bedrooms'] + df['bathrooms']
print(df[['bedrooms', 'bathrooms', 'total_rooms']])
```

Feature Selection

Now that we have engineered new features, let's proceed with feature selection. We will select features based on their correlation with the target variable (price).

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

```
# Correlation matrix
correlation_matrix = df.corr()

# Plotting the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

From the correlation matrix, we can observe which features are highly correlated with price. Features with high positive or negative correlation might be retained, while those with low correlation can be dropped.

Feature Selection Using a Model

We can also use a machine learning model like RandomForestRegressor to estimate the importance of features.

```
from sklearn.ensemble import RandomForestRegressor

# Defining features and target
X = df[['area', 'age', 'total_rooms']]  # Selected features
y = df['price']  # Target variable

# Training a Random Forest model
model = RandomForestRegressor()
model.fit(X, y)

# Get feature importance
feature_importance = model.feature_importances_
feature_names = X.columns
for name, importance in zip(feature_names, feature_importance):
```

```
print(f'{name}: {importance:.4f}')
```

This output will show the importance of each feature for predicting house prices. You can retain or drop features based on the importance scores to improve model performance.

 Feature Engineering allows you to create more informative attributes, such as age and total_rooms, which provide better input to the model.
 Feature Selection helps by reducing irrelevant or redundant features, ensuring the model focuses on the most predictive attributes.

Predictive vs. Descriptive Analytics: Complete Example

Descriptive Analytics focuses on summarizing historical data to understand what happened in the past. It provides insights into trends and patterns through statistical measures like mean, median, variance, and visualizations.

Predictive Analytics uses statistical models and machine learning to forecast future outcomes based on historical data. It predicts trends, outcomes, and the likelihood of future events.

Example: Sales Data Analysis

We will use a simplified sales dataset in Python to demonstrate descriptive and predictive analytics.

```
import pandas as pd


# Sales dataset: Year, Product, Units Sold, Revenue

data = {'year': [2019, 2020, 2021, 2022],

    'product_A_units': [150, 200, 250, 300],

    'product_B_units': [100, 150, 200, 250],

    'product_A_revenue': [45000, 60000, 75000, 90000],

    'product_B_revenue': [30000, 45000, 60000, 75000]}
```

```
df = pd.DataFrame(data)

print(df)
```

This dataset contains sales data for two products (Product A and Product B) over four years, with information on units sold and total revenue generated.

Descriptive Analytics

1. **Summarizing the Data**: Descriptive analytics involves summarizing historical data to identify trends.

```
# Descriptive statistics for units sold

product_summary = df[['product_A_units', 'product_B_units']].describe()

print(product_summary)
```

This output shows the descriptive statistics for units sold, including the mean (average), standard deviation, and quartile ranges.

2. **Visualization**: Visualization helps us understand trends and patterns visually. Here, we plot the sales of Product A and Product B over time.

```
import matplotlib.pyplot as plt


# Plotting sales trend over time

plt.plot(df['year'], df['product_A_units'], label='Product A')

plt.plot(df['year'], df['product_B_units'], label='Product B')

plt.xlabel('Year')

plt.ylabel('Units Sold')

plt.title('Units Sold Over Time')

plt.legend()

plt.show()
```

This plot shows the sales trend for both products over time. Descriptive analytics helps you summarize and visualize this historical data.

Predictive Analytics

Now, let's move to predictive analytics by using historical data to predict future sales.

1. **Fitting a Model**: Based on past sales data, we will use a simple linear regression model to predict sales for the next year (2023).

```
from sklearn.linear_model import LinearRegression


# Prepare the data for predictive modeling
X = df[['year']]  # Independent variable (Year)
y_A = df['product_A_units']  # Dependent variable (Product A units sold)


# Initialize and train the model for Product A
model_A = LinearRegression()
model_A.fit(X, y_A)


# Predicting future sales for Product A for 2023
year_to_predict = [[2023]]
predicted_units_A = model_A.predict(year_to_predict)
print(f"Predicted units sold for Product A in 2023: {predicted_units_A[0]:.0f}")
```

Prediction for Product B:

```
# Dependent variable for Product B
y_B = df['product_B_units']


# Initialize and train the model for Product B
model_B = LinearRegression()
model_B.fit(X, y_B)


# Predicting future sales for Product B for 2023
```

predicted_units_B = model_B.predict(year_to_predict)

print(f"Predicted units sold for Product B in 2023: {predicted_units_B[0]:.0f}")

 Descriptive Analytics helped us summarize and visualize historical sales data. We found that both products had a steady increase in sales over the years.

 Predictive Analytics used historical sales data to forecast future sales. By training a linear regression model, we predicted the sales for 2023 based on past trends.

Statistics for Data Analysis

Statistics play a crucial role in data analysis by providing techniques to describe, analyze, and interpret data.

Descriptive Statistics

Descriptive statistics summarize the data distribution's central tendency, variability, and shape.

- **Central Tendency**: Mean, median, mode.
- **Variability**: Range, variance, standard deviation.

We will use a hypothetical dataset of exam scores to demonstrate these concepts.

import pandas as pd

Dataset of student exam scores

data = {'student': ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],

'score': [82, 90, 76, 85, 92, 88, 75, 95, 80, 70]}

df = pd.DataFrame(data)

print(df)

Descriptive Statistics

Descriptive statistics summarize the basic features of the data, including measures of central tendency (mean, median, mode) and variability (standard deviation, variance).

a) Measures of Central Tendency

1. **Mean**: The average score.
2. **Median**: The middle score when sorted.
3. **Mode**: The most frequently occurring score.

```
# Mean

mean_score = df['score'].mean()

print(f"Mean Score: {mean_score}")



# Median

median_score = df['score'].median()

print(f"Median Score: {median_score}")



# Mode

mode_score = df['score'].mode()[0]  # Mode can return multiple values, take the first one

print(f"Mode Score: {mode_score}")
```

Measures of Variability

1. **Variance**: The spread of scores.
2. **Standard Deviation**: A measure of how much the scores deviate from the mean

```
# Variance

variance_score = df['score'].var()

print(f"Variance: {variance_score}")



# Standard Deviation

std_dev_score = df['score'].std()
```

```
print(f"Standard Deviation: {std_dev_score}")
```

Visualization of Data Distribution

```
import matplotlib.pyplot as plt

import seaborn as sns


# Plotting the distribution of scores

plt.figure(figsize=(8, 4))

sns.histplot(df['score'], bins=5, kde=True)

plt.title("Distribution of Exam Scores")

plt.xlabel("Score")

plt.ylabel("Frequency")

plt.show()
```