Task Start Here

In [ ]:
```
#importing libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
def print_bold(text):
    print("\033[1m" + text + "\033[0m")
```

Step 1: Extracting Data from source

Extraction involves gathering data from diverse source systems and various tools and technologies facilitate this process. ETL tools such as Apache Nifi, Talend, and Informatica offer visual interfaces for designing data extraction workflows.

Database management systems (DBMS) like Oracle, SQL Server, and MySQL often provide built-in features or tools for data extraction. Additionally, cloud-based services such as AWS Glue and Google Cloud Dataflow enable scalable and efficient extraction from cloud-based sources.

In [ ]:
```
data = pd.read_csv('insurance.csv') #data loading from local source
```

In [ ]:
```
df=data.copy() #keeping original data for future comparison and using df for processing
```

Exporing Data

In [ ]:
```
print_bold("Original Data Table :")
df.head()
```

In [ ]:

```
print_bold("Original Data Info:\n")
print(df.info()) #data info - original
```

In [ ]:

```
#checking for unique value in sex column with count of value.
print("Total Unique Sex Present:", df['sex'].nunique(),'\n')
print(df['sex'].value_counts())
```

In [ ]:

```
#value count of each unique value in sex column
plt.figure(figsize=(10,7))
sns.countplot(x='sex', data=df, palette='pastel')
plt.title('Gender Count')
plt.xlabel('Gender')
plt.ylabel('Count')

ax = sns.countplot(x='sex', data=df, palette='pastel')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')
plt.show()
```

In [ ]:

```
# age distribution
plt.figure(figsize=(10, 6))
ax = sns.histplot(df['age'], bins=20, kde=True, color='skyblue')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')

for i in ax.patches:
    ax.annotate(f'{int(i.get_height())}', (i.get_x() + i.get_width() / 2., i.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')

plt.show()
```

In [ ]:

```
#checking for count of each value in smoker column.
print(df['smoker'].value_counts())
```

In [ ]:
```
#value count of each unique value in smoker
plt.figure(figsize=(8, 5))
sns.countplot(x='smoker', data=df, palette='Set2')
plt.title('Count of Smokers and Non-Smokers')
plt.xlabel('Smoker')
plt.ylabel('Count')
ax = sns.countplot(x='smoker', data=df, palette='pastel')
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')
plt.show()
```

In [ ]:
```
#boxplot for charges based on smoker status
plt.figure(figsize=(10, 6))
sns.boxplot(x='smoker', y='charges', data=df, palette='viridis')
plt.title('Distribution of Charges for Smokers and Non-Smokers')
plt.xlabel('Smoker')
plt.ylabel('Charges')
plt.show()
```

In [ ]:
```
print_bold("Original Data Description:\n")
print(df.describe())
```

In [ ]:
```
print(df['region'].value_counts()) #count of each unique value present on the region column
```

In [ ]:
```
#Cheking for null value present in each column of the data.
print(df.isnull().sum())
```

Step 2: Transformation on Data

Transformation encompasses diverse operations on the extracted data, and numerous tools and technologies are available for this phase. ETL tools like Apache Spark, Microsoft SSIS, and IBM InfoSphere offer powerful transformation capabilities, supporting data cleansing, enrichment, and aggregation. Programming languages such as Python and libraries like Pandas provide flexibility for custom transformations.

Data quality tools like Trifacta and Talend Data Quality assist in ensuring high-quality transformations. Machine learning frameworks like TensorFlow or scikit-learn can be integrated for advanced data transformations. Cloud-based platforms such as Azure Data Factory and Google Cloud Dataprep also offer transformation services.

In [ ]:
```
#mapping Sex column i.e. is Male with 0 and Female with 1 for making it numerical.
df['sex'] = df['sex'].map({'male': 0, 'female': 1})
```

In [ ]:
```
#mapping Smoker column i.e. is no with 0 and yes with 1 for making it numerical.
df['smoker'] = df['smoker'].map({'no': 0, 'yes': 1})
```

In [ ]:
```
#finding unique value present in the region column.
print("Total Unique Regions:", df['region'].nunique())
#getting the list of unique region name.
print("Region Present in the Data:",df['region'].unique().tolist())
```

In [ ]:
```
#using label encoder to assign numerical value to the element present in the region for better modelling.
label_encoder = LabelEncoder()
df['region_encoded'] = label_encoder.fit_transform(df['region'])
```

In [ ]:
```
#one hot encoding on region column for future analysis and better ML model performance
df = pd.get_dummies(df, columns=['region'], prefix='region')
df['region_encoded'] = df['region_encoded'].astype(np.int64)
```

In [ ]:
```
#getting the numericaly assigned value for each region present in the region column.
region_mapping = list(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
print_bold("Mapping done on original region values to encoded values:")
print(region_mapping)
```

In [ ]:
```
print_bold("Transformed Data Table:\n")
df.head()
```

Step 2B : Transformation Data Quality Check.

In [ ]:
```
#expected data types for the transformed Data
expected_data_types = {
    'age': np.int64,
    'sex': np.int64,
    'bmi': np.float64,
    'children': np.int64,
    'smoker': np.int64,
    'charges': np.float64,
    'region_encoded': np.int64,
    'region_northeast': np.uint8,
    'region_northwest': np.uint8,
    'region_southeast': np.uint8,
    'region_southwest': np.uint8
}

data_types_check = df.dtypes.to_dict() == expected_data_types #comparison

sex_smoker_check = (df['sex'].isin([0, 1]).all()) and (df['smoker'].isin([0, 1]).all())

# data quality check
if data_types_check and sex_smoker_check:
    print_bold("\nData Quality Check: Transformation logic verified. Data types and values are as expected.")
    print_bold("\nTransformed Data Info:\n")
    print(df.info())

else:
    print_bold("\nData Quality Check: Transformation logic verification failed. Check data types and values.")
```

Step 3: Loading data into desired storage.

Loading transformed data into target systems involves various tools and technologies. ETL tools like Apache NiFi and Talend provide functionalities for designing efficient loading workflows. Relational database management systems (RDBMS) such as PostgreSQL, MySQL, and SQL Server offer mechanisms for bulk loading and transaction management. NoSQL databases like MongoDB and cloud-based data warehouses like Amazon Redshift and Google BigQuery support optimized loading strategies.

In [ ]:
```
#Storing value into the assingned local storage.
```

```
#uncomment below line to save data in the local storage.
#df.to_csv('transformed_insurance.csv', index=False)
print_bold('The transformed file with the name "transformed_insurance.csv" is being saved successfully on desired location.')
```

Creating Server Connection to load transformed data into local SQL server for future desired operation.

In [ ]:

```
import sqlalchemy
```

In [ ]:

```
#providing neccesary information required to connect with server
server_name = 'LAPTOP-SAMIR\SQLEXPRESS'
database_name = 'Wednesday'
trans_table = "transformed_insurance"  #table name of the data

connection_string = f'mssql+pyodbc://@{server_name}/{database_name}?driver=ODBC+Driver+17+for+SQL+Server&trusted_connection=yes'

print_bold("Server Connection Established")
```

In [ ]:

```
df.to_sql(name= trans_table, con=connection_string, if_exists='replace', index=False) #sending request to server

print_bold(f'Action Completed: The {trans_table} data table is being added to the SQL server under the database {database_name}.\n
```

In [ ]:

```
#sending test query for the uploaded data on the server
query = "SELECT age, sex, bmi, smoker, charges FROM transformed_insurance"
query_result = pd.read_sql(query, con=connection_string)
print(query_result.head())
```

Note: Server connection needs to be closed after a successful operation to avoid any data loss, to protect overwrite, etc. based on the server used.

ETL Ends Here

Task - Data Analysis Using SQL.

Using pandas data frame to get insight from the data using SQL, SQL query is being supported on pandas dataframe using pandasql library.

We can also transfer this data into SQL database using several libraries to perform analysis in SQL studio in this method server configuration is required which is supported by the library.

In [ ]:
```
import pandasql as ps
```

In [ ]:
```
data.head() #first 5 rows of the data
```

In [ ]:
```
data.info() #data info - orginal
```

Using orginal data for analysis using SQL purposely.

Query - 1 :

In [ ]:
```
#view table for the data df
table = """
    SELECT * FROM data;
"""

result_table = ps.sqldf(table, locals())
print("Output:\n\n",result_table.head())
```

Query - 2 :

In [ ]:
```
#top 5 age which have maximum individual.
query_age_distribution = """
    SELECT
        age,
        COUNT(*) AS count
    FROM
        data
    GROUP BY
        age
```

```
    ORDER BY
        count DESC
    LIMIT 5;
"""

result_query_age_distribution = ps.sqldf(query_age_distribution, locals())
print(result_query_age_distribution)
```

Query - 3 :

In [ ]:

```
#Find the average charges for each region:
query_avg_charges_region = """
    SELECT
        region,
        AVG(charges) AS avg_charges
    FROM
        data
    GROUP BY
        region;
"""

result_avg_charges_region = ps.sqldf(query_avg_charges_region, locals())
print("Output:\n\n",result_avg_charges_region)
```

Query - 4 :

In [ ]:

```
#Find the average charges for smokers and non-smokers:
query_avg_charges_smoker = """
    SELECT
        smoker,
        AVG(charges) AS avg_charges
    FROM
        data
    GROUP BY
        smoker;
"""
```

```
result_avg_charges_smoker = ps.sqldf(query_avg_charges_smoker, locals())
print("Output:\n\n",result_avg_charges_smoker)
```

Query - 5 :

In [ ]:

```
#Average Charges by Smoking Status and Gender:
query_avg_charges_by_smoking_gender = """
    SELECT
        smoker,
        sex,
        AVG(charges) AS avg_charges
    FROM
        data
    GROUP BY
        smoker, sex;
"""
result_query_avg_charges_by_smoking_gender = ps.sqldf(query_avg_charges_by_smoking_gender, locals())
print("Output:\n\n",result_query_avg_charges_by_smoking_gender)
```

Query - 6 :

In [ ]:

```
#Find the total number of smokers and non-smokers in each region:
query_smoker_count_region = """
    SELECT
        region,
        smoker,
        COUNT(*) AS count
    FROM
        data
    GROUP BY
        region, smoker;
"""

result_smoker_count_region = ps.sqldf(query_smoker_count_region, locals())
print("Output:\n\n",result_smoker_count_region)
```

Query - 7 :

In [ ]:

```
#Find the average BMI for each sex:
query_avg_bmi_sex = """
    SELECT
        sex,
        AVG(bmi) AS avg_bmi
    FROM
        data
    GROUP BY
        sex;
"""

result_avg_bmi_sex = ps.sqldf(query_avg_bmi_sex, locals())
print("Output:\n\n",result_avg_bmi_sex)
```

Query - 8 :

In [ ]:

```
#Find the average charges for individuals with and without children:
query_avg_charges_children = """
    SELECT
        children,
        AVG(charges) AS avg_charges
    FROM
        data
    GROUP BY
        children;
"""

result_avg_charges_children = ps.sqldf(query_avg_charges_children, locals())
print("Output:\n\n",result_avg_charges_children)
```

Query - 9 :

In [ ]:

```
#Find the average BMI and charges for each combination of sex and smoker status:
query_avg_bmi_charges_sex_smoker = """
    SELECT
        sex,
        smoker,
        AVG(bmi) AS avg_bmi,
        AVG(charges) AS avg_charges,
        AVG(age) as avg_age
    FROM
        data
    GROUP BY
        sex, smoker;
"""

result_avg_bmi_charges_sex_smoker = ps.sqldf(query_avg_bmi_charges_sex_smoker, locals())
print("Output:\n\n",result_avg_bmi_charges_sex_smoker)
```

Query - 10 :

In [ ]:

```
#Determine the average charges for individuals with different numbers of children, grouped by smoker status:
query_avg_charges_children_smoker = """
    SELECT
        children,
        smoker,
        AVG(charges) AS avg_charges
    FROM
        data
    GROUP BY
        children, smoker;
"""

result_avg_charges_children_smoker = ps.sqldf(query_avg_charges_children_smoker, locals())
print("Output:\n\n",result_avg_charges_children_smoker)
```

Query - 11 :

In [ ]:
```
#Find the top 5 individuals with the highest charges:
query_top5_highest_charges = """
    SELECT
        *
    FROM
        data
    ORDER BY
        charges DESC
    LIMIT 5;
"""

result_top5_highest_charges = ps.sqldf(query_top5_highest_charges, locals())
print("Output:\n\n",result_top5_highest_charges)
```

Query - 12 :

In [ ]:

```
#top 2 individuals from each region with the highest charges
query_top2_highest_charges_per_region = """
    SELECT region,
        age,
        sex,
        bmi,
        children,
        smoker,
        charges
    FROM (
        SELECT * ,
            ROW_NUMBER() OVER (PARTITION BY region ORDER BY charges DESC) AS row_num
        FROM
            data
    ) ranked
    WHERE
        row_num <= 2
    ORDER BY
        region, row_num;
"""

result_top2_highest_charges_per_region = ps.sqldf(query_top2_highest_charges_per_region, locals())
print("Output:\n\n",result_top2_highest_charges_per_region)
```

Task Ends Here