unemployment-in-india

October 30, 2024

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```python
data = '/content/drive/MyDrive/Unemployment_Rate_upto_11_2020.csv'
df = pd.read_csv(data)
```

[ ]:
```python
# Display the first few rows of the dataframe
df.head()
```

[ ]:
```
           Region         Date  Frequency   Estimated Unemployment Rate (%)  \
0  Andhra Pradesh   31-01-2020          M                              5.48
1  Andhra Pradesh   29-02-2020          M                              5.83
2  Andhra Pradesh   31-03-2020          M                              5.79
3  Andhra Pradesh   30-04-2020          M                             20.51
4  Andhra Pradesh   31-05-2020          M                             17.43

    Estimated Employed   Estimated Labour Participation Rate (%) Region.1  \
0             16635535                                     41.02    South
1             16545652                                     40.90    South
2             15881197                                     39.18    South
3             11336911                                     33.10    South
4             12988845                                     36.46    South

   longitude  latitude
0    15.9129     79.74
1    15.9129     79.74
2    15.9129     79.74
3    15.9129     79.74
4    15.9129     79.74
```

[ ]:
```python
'''Step 2: Data Cleaning and Preprocessing Now that we have a better
 ↪understanding of the dataset, we'll clean and preprocess the data to ensure
 ↪it's ready for analysis.'''
# Print and Clean Column Names

# Print column names to identify any issues
print("Original Column Names:")
print(df.columns)

# Remove any leading or trailing spaces from column names
df.columns = df.columns.str.strip()

# Print column names to confirm the changes
print("\nCleaned Column Names:")
print(df.columns)
```

```
Original Column Names:
Index(['Region', ' Date', ' Frequency', ' Estimated Unemployment Rate (%)',
       ' Estimated Employed', ' Estimated Labour Participation Rate (%)',
       'Region.1', 'longitude', 'latitude'],
      dtype='object')

Cleaned Column Names:
Index(['Region', 'Date', 'Frequency', 'Estimated Unemployment Rate (%)',
       'Estimated Employed', 'Estimated Labour Participation Rate (%)',
       'Region.1', 'longitude', 'latitude'],
      dtype='object')
```

[ ]:
```python
# Convert 'Date' to Datetime Format

# Verify the presence of the 'Date' column
if 'Date' in df.columns:
    # Remove any leading or trailing spaces from the 'Date' column values
    df['Date'] = df['Date'].str.strip()

    # Convert 'Date' to datetime format
    df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')

    # Display the data types to confirm the changes
    print("\nData types after conversion:")
    print(df.dtypes)

    # Display the first few rows of the dataframe
    df.head()
else:
    print("The 'Date' column was not found. Please check the dataset for any
 ↪discrepancies.")
```

```
Data types after conversion:
Region                                               object
Date                                         datetime64[ns]
Frequency                                            object
Estimated Unemployment Rate (%)                     float64
Estimated Employed                                    int64
Estimated Labour Participation Rate (%)             float64
Region.1                                             object
longitude                                           float64
latitude                                            float64
dtype: object
```

[ ]:
```python
# Check for Missing Values and Outliers

# Check for missing values in the dataset
missing_values = df.isnull().sum()
print("Missing values in each column:")
print(missing_values)

# Basic statistics to identify any outliers
print("\nSummary statistics:")
print(df.describe())
```

```
Missing values in each column:
Region                                     0
Date                                       0
Frequency                                  0
Estimated Unemployment Rate (%)            0
Estimated Employed                         0
Estimated Labour Participation Rate (%)    0
Region.1                                   0
longitude                                  0
latitude                                   0
dtype: int64

Summary statistics:
                                Date  Estimated Unemployment Rate (%)  \
count                            267                       267.000000
mean   2020-06-16 09:15:30.337078528                        12.236929
min              2020-01-31 00:00:00                         0.500000
25%              2020-03-31 00:00:00                         4.845000
50%              2020-06-30 00:00:00                         9.650000
75%              2020-08-31 00:00:00                        16.755000
max              2020-10-31 00:00:00                        75.850000
std                              NaN                        10.803283

       Estimated Employed  Estimated Labour Participation Rate (%)  \
```

```
count      2.670000e+02                               267.000000
mean       1.396211e+07                                41.681573
min        1.175420e+05                                16.770000
25%        2.838930e+06                                37.265000
50%        9.732417e+06                                40.390000
75%        2.187869e+07                                44.055000
max        5.943376e+07                                69.690000
std        1.336632e+07                                 7.845419

        longitude    latitude
count  267.000000  267.000000
mean    22.826048   80.532425
min     10.850500   71.192400
25%     18.112400   76.085600
50%     23.610200   79.019300
75%     27.278400   85.279900
max     33.778200   92.937600
std      6.270731    5.831738
```

[ ]:
```
# Rename Columns (Optional)

# Rename columns for easier reference
df.rename(columns={
    'Estimated Unemployment Rate (%)': 'Unemployment_Rate',
    'Estimated Employed': 'Employed',
    'Estimated Labour Participation Rate (%)': 'Labour_Participation_Rate',
    'Region.1': 'Region_Category'}, inplace=True)
```

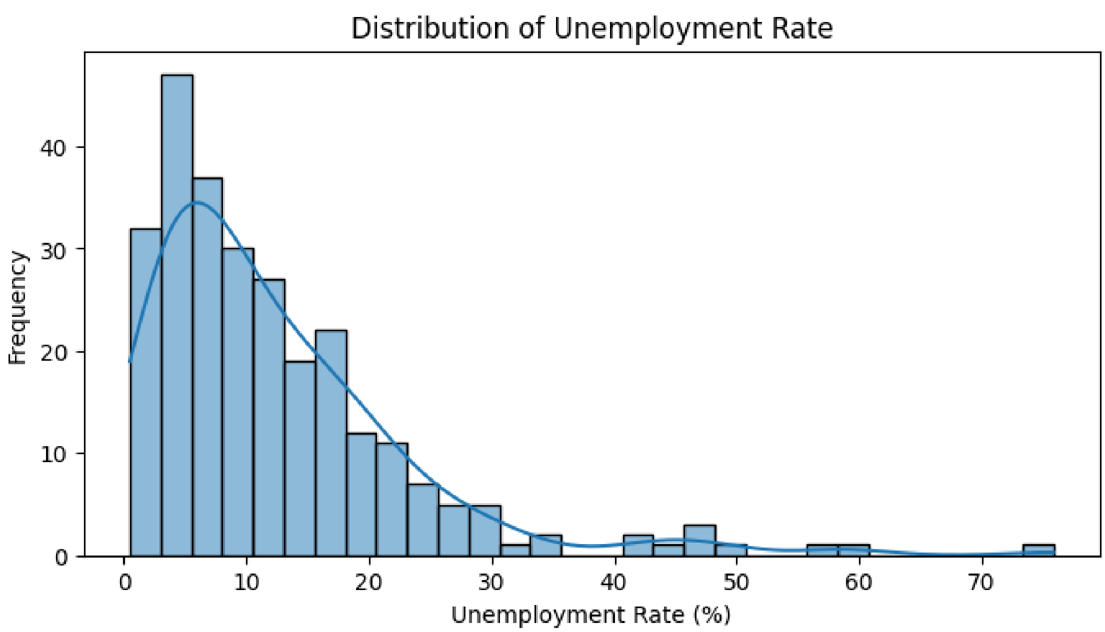
[ ]:
```
# Display the first few rows after preprocessing
df.head()
```

[ ]:
```
           Region       Date Frequency  Unemployment_Rate  Employed  \
0  Andhra Pradesh 2020-01-31         M               5.48  16635535
1  Andhra Pradesh 2020-02-29         M               5.83  16545652
2  Andhra Pradesh 2020-03-31         M               5.79  15881197
3  Andhra Pradesh 2020-04-30         M              20.51  11336911
4  Andhra Pradesh 2020-05-31         M              17.43  12988845

   Labour_Participation_Rate Region_Category  longitude  latitude
0                      41.02           South    15.9129     79.74
1                      40.90           South    15.9129     79.74
2                      39.18           South    15.9129     79.74
3                      33.10           South    15.9129     79.74
4                      36.46           South    15.9129     79.74
```

[ ]:
```
'''Step 3: Exploratory Data Analysis (EDA) After cleaning and preprocessing the
↪data, the next step is to perform Exploratory Data Analysis (EDA).'''
```

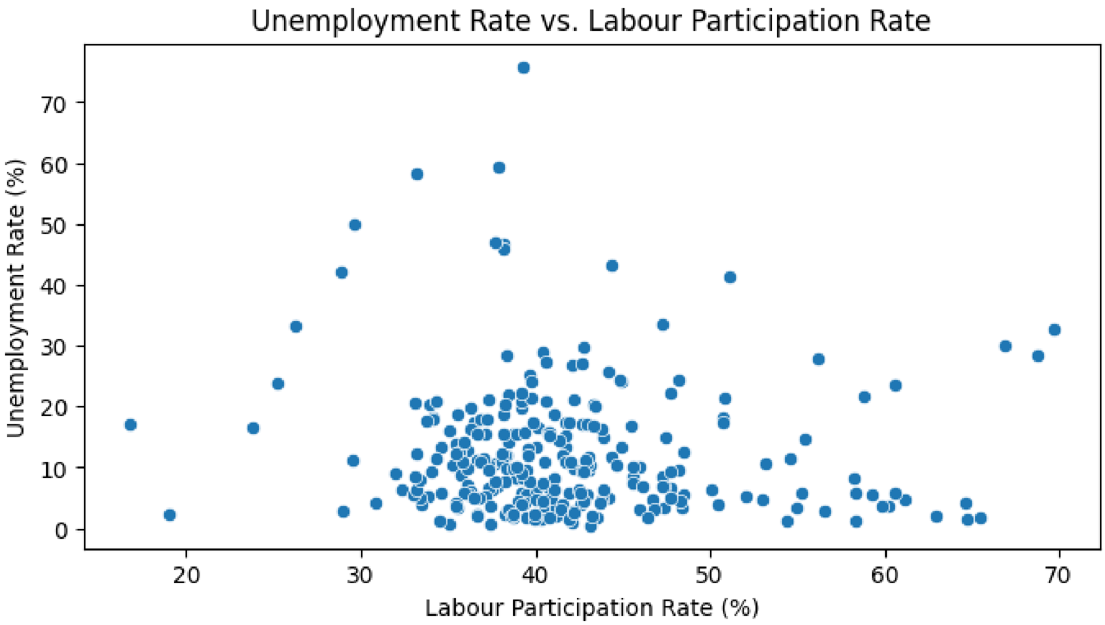
```
'''This step will help us understand the patterns, relationships, and key␣
 ↪statistics in the data.'''
```

[ ]:
```
# Univariate Analysis

# Distribution of Unemployment Rate
plt.figure(figsize=(8, 4))
sns.histplot(df['Unemployment_Rate'], bins=30, kde=True)
plt.title('Distribution of Unemployment Rate')
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Frequency')
plt.show()
```



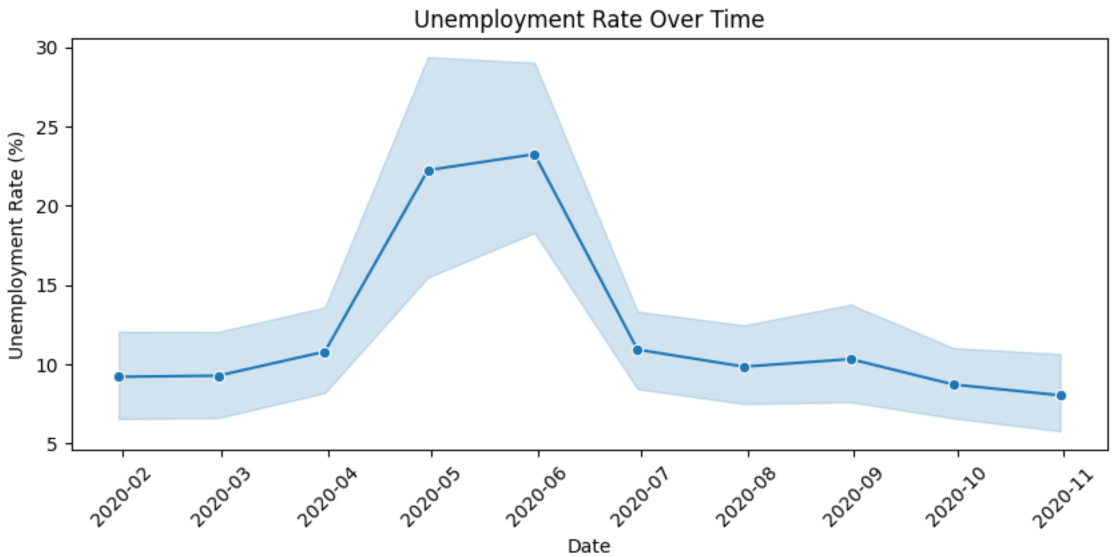
[ ]:
```
# Bivariate Analysis

# Unemployment Rate vs. Labour Participation Rate
plt.figure(figsize=(8, 4))
sns.scatterplot(x='Labour_Participation_Rate', y='Unemployment_Rate', data=df)
plt.title('Unemployment Rate vs. Labour Participation Rate')
plt.xlabel('Labour Participation Rate (%)')
plt.ylabel('Unemployment Rate (%)')
plt.show()

# Calculate and display the correlation between these variables
correlation = df['Labour_Participation_Rate'].corr(df['Unemployment_Rate'])
```

```
print(f"Correlation between Labour Participation Rate and Unemployment Rate:␣
↪{correlation:.2f}")
```

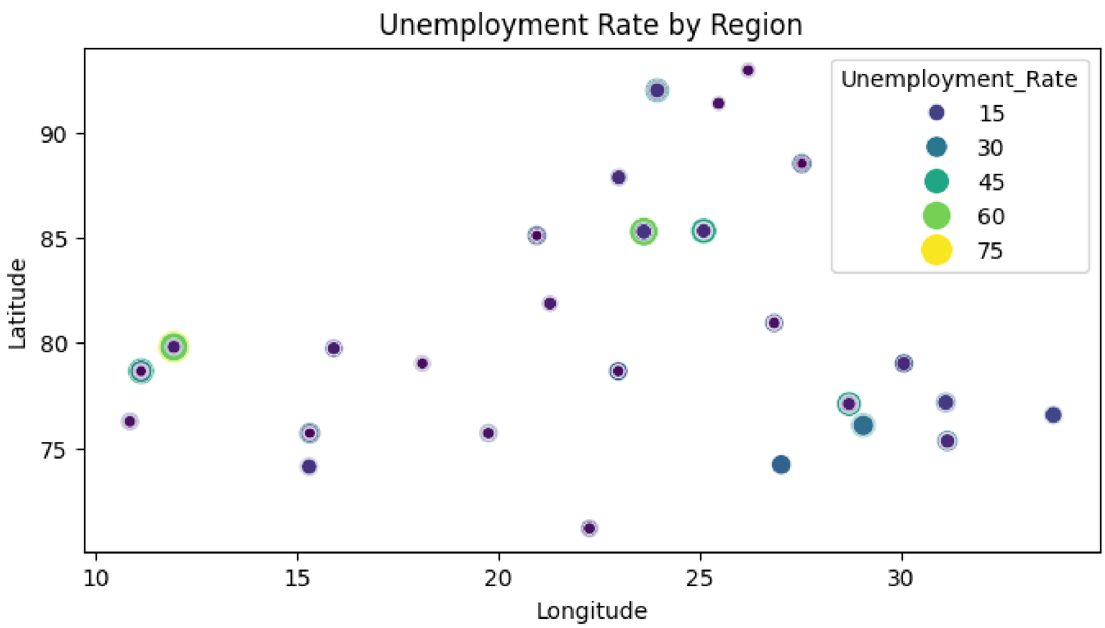



```
Correlation between Labour Participation Rate and Unemployment Rate: -0.07
```

[ ]:
```
# Time Series Analysis

# Trend of Unemployment Rate Over Time
plt.figure(figsize=(10,4))
sns.lineplot(x='Date', y='Unemployment_Rate', data=df, marker='o')
plt.title('Unemployment Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Unemployment Rate (%)')
plt.xticks(rotation=45)
plt.show()
```



[ ]:
```
# Geographical Analysis

# Unemployment Rate by Region
plt.figure(figsize=(8, 4))
sns.scatterplot(x='longitude', y='latitude', hue='Unemployment_Rate',␣
↪size='Unemployment_Rate', data=df, palette='viridis', sizes=(20, 200))
plt.title('Unemployment Rate by Region')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



[ ]:
```
# Hypothesis Testing Once we have explored the data through EDA, we can move on to hypothesis testing to validate any assumptions or insights derived from the data.

# 1) Formulate Hypotheses Null Hypothesis (H0): There is no significant difference in the unemployment rate across different regions.
# Alternative Hypothesis (H1): There is a significant difference in the unemployment rate across different regions.
# 2) Perform the Test
```

[ ]:
```
from scipy.stats import f_oneway

# Extracting data for ANOVA
regions = df['Region_Category'].unique()
anova_data = [df[df['Region_Category'] == region]['Unemployment_Rate'] for region in regions]

# Performing the ANOVA test
anova_result = f_oneway(*anova_data)
print(f"ANOVA test result: F-statistic = {anova_result.statistic:.2f}, p-value = {anova_result.pvalue:.4f}")

# Interpretation
if anova_result.pvalue < 0.05:
    print("Result: Reject the null hypothesis. There is a significant difference in unemployment rates across regions.")
```

```
else:
    print("Result: Fail to reject the null hypothesis. No significant␣
↪difference in unemployment rates across regions.")
```

```
ANOVA test result: F-statistic = 5.04, p-value = 0.0006
Result: Reject the null hypothesis. There is a significant difference in
unemployment rates across regions.
```

[ ]:
```
# Model Building (Revised) Now that we have a good understanding of the data␣
↪through our exploratory data analysis (EDA),
# it's time to build models that can help us predict future unemployment rates.
# We'll begin by setting up a baseline model and then refine our approach.
```

[ ]:
```
# Splitting the Data First, we need to split the data into training and testing␣
↪sets. This will help us evaluate the performance of our model on unseen data.
```

[ ]:
```
from sklearn.model_selection import train_test_split

# Features and target variable
X = df[['Labour_Participation_Rate', 'Employed', 'longitude', 'latitude']]
y = df['Unemployment_Rate']

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

print(f"Training set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Training set size: 213
Test set size: 54
```

[ ]:
```
# Baseline Model: Linear Regression We'll start with a simple Linear Regression␣
↪model as a baseline.
# This will give us an initial understanding of the relationship between the␣
↪variables.
```

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

# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Squared Error (MSE): 88.80
R-squared (R2): 0.06
```

[ ]:
```
# Model Refinement: Feature Engineering If the baseline model's performance is␣
↪not satisfactory, we can refine it by engineering new features or trying␣
↪different models.
# For instance, we could include interactions between features or try␣
↪polynomial regression
```

[ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# Create a pipeline with polynomial features and linear regression
poly_model = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('linear', LinearRegression())
])

# Train the refined model
poly_model.fit(X_train, y_train)

# Make predictions
y_pred_poly = poly_model.predict(X_test)

# Evaluate the refined model
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print(f"Polynomial Model - Mean Squared Error (MSE): {mse_poly:.2f}")
print(f"Polynomial Model - R-squared (R2): {r2_poly:.2f}")
```

```
Polynomial Model - Mean Squared Error (MSE): 75.38
Polynomial Model - R-squared (R2): 0.20
```

[ ]:
```
# Model Selection Depending on the results from the baseline and refined␣
↪models, we can choose the best-performing model.
# If necessary, we might explore other algorithms such as Decision Trees,␣
↪Random Forests, or Gradient Boosting Machines.
```

[ ]:
```
# Model Evaluation Finally, we will assess the selected model on the test set␣
↪using metrics such as MSE and R-squared.
# If the model is satisfactory, we can move forward with deploying it or using␣
↪it for predictive analysis.
```

[ ]:
```
# Final model evaluation on the test set
final_model = model  # or poly_model, depending on performance

y_final_pred = final_model.predict(X_test)
final_mse = mean_squared_error(y_test, y_final_pred)
final_r2 = r2_score(y_test, y_final_pred)

print(f"Final Model - Mean Squared Error (MSE): {final_mse:.2f}")
print(f"Final Model - R-squared (R2): {final_r2:.2f}")
```

```
Final Model - Mean Squared Error (MSE): 88.80
Final Model - R-squared (R2): 0.06
```