# Air_transport_notebook

July 31, 2024

## 0.1 Air Transport Analysis for selected European Countries with Ireland

Analysis of air transport data is the main objective of this research project

## 0.2 Importing & Joining data

[1]:
```
pip install eurostat
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: eurostat in
/home/44d0387b-deba-4fb3-a3ba-f192c2622970/.local/lib/python3.9/site-packages
(1.0.4)
Requirement already satisfied: requests in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
eurostat) (2.27.1)
Requirement already satisfied: pandas in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
eurostat) (1.4.2)
Requirement already satisfied: python-dateutil>=2.8.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
pandas->eurostat) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
pandas->eurostat) (2021.3)
Requirement already satisfied: numpy>=1.18.5 in
/home/44d0387b-deba-4fb3-a3ba-f192c2622970/.local/lib/python3.9/site-packages
(from pandas->eurostat) (1.26.2)
Requirement already satisfied: six>=1.5 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from python-
dateutil>=2.8.1->pandas->eurostat) (1.16.0)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests->eurostat) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests->eurostat) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
```

```
requests->eurostat) (1.26.9)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests->eurostat) (2021.10.8)
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
pip install openpyxl
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: openpyxl in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (3.0.9)
Requirement already satisfied: et-xmlfile in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[1]:
```
import eurostat
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-
packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and
<1.23.0 is required for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

# 1 Import datasets

[3]:
```
#Import and melt the first dataset
df1 = pd.read_excel("avia_air_transport__custom.xlsx", sheet_name='Sheet 1')
# Melt the DataFrame to convert it to a long format
melted_df1 = pd.melt(df1, id_vars=['Country'], var_name='Year', value_name='Air␣
 ↪Passenger Transport')
melted_df1['Year'] = melted_df1['Year'].astype(float)
melted_df1['Air Passenger Transport']=melted_df1['Air Passenger Transport'].
 ↪astype(float)
#Saving the melted df
melted_df1.to_csv('air_passengers_melted.csv',index=False)
# Display the info of the resulting DataFrame
melted_df1.dtypes
melted_df1.head()
```

[3]:

| | Country | Year | Air Passenger Transport |
|---|---|---|---|
| 0 | Denmark | 2011.0 | 2369405.0 |
| 1 | Ireland | 2011.0 | 113912.0 |

```
2   Norway  2011.0               14116104.0
3  Denmark  2012.0                1924590.0
4  Ireland  2012.0                  53660.0
```

[5]:
```
##Import and melt the second dataset
df2 = pd.read_excel("At_risk_pov_threshold.xlsx", sheet_name='Sheet 1')
# Melt the DataFrame to convert it to a long format
melted_df2 = pd.melt(df2, id_vars=['Country'], var_name='Year',␣
 ↪value_name='Purchasing Power')
# Drop rows where 'Pop Distribution' is not specified (indicated by ':')
melted_df2 = melted_df2[melted_df2['Purchasing Power']!=':']
melted_df2['Year'] = melted_df2['Year'].astype(float)
melted_df2['Purchasing Power']=melted_df2['Purchasing Power'].astype(float)
selected_countries=['Denmark','Ireland','Norway']
melted_df2=melted_df2[melted_df2['Country'].isin(selected_countries)]
#Saving the melted df
melted_df2.to_csv('purchasing_power_melted.csv',index=False)
# Display the info of the resulting DataFrame
melted_df2.dtypes
melted_df2.head()
```

[5]:
```
    Country    Year  Purchasing Power
3   Denmark  2011.0           11510.0
6   Ireland  2011.0            9977.0
28   Norway  2011.0           14519.0
40  Denmark  2012.0           11537.0
43  Ireland  2012.0            9912.0
```

[7]:
```
##Import and melt the third dataset
df3 = pd.read_excel("air_goods_transport.xlsx",sheet_name='Sheet 1')
# Melt the DataFrame to convert it to a long format
melted_df3 = pd.melt(df3, id_vars=['Country'], var_name='Year', value_name='Air␣
 ↪Goods Transport')
melted_df3['Year'] = melted_df3['Year'].astype(float)
melted_df3['Air Goods Transport']=melted_df3['Air Goods Transport'].
 ↪astype(float)
#Saving the melted df
melted_df3.to_csv('air_goods_transport_melted.csv',index=False)
# Display the info of the resulting DataFrame
melted_df3.dtypes
melted_df3.head()
```

[7]:
```
    Country    Year  Air Goods Transport
0  Denmark  2011.0             155662.3
1  Ireland  2011.0             113408.7
2   Norway  2011.0              65742.0
3  Denmark  2012.0             166283.0
```

```
4   Ireland  2012.0              126833.6
```

### 1.1 Combining Data

This step focuses on joining the melted dataset by using outer joins. The summary of the final dataset will be displayed.

[9]:
```
# Merge the result with melted_df1, melted_df2 and melted_df3 on common columns␣
 ↪'Country' and 'Year'
merged_df1 = pd.merge(melted_df1, melted_df2, on=['Country', 'Year'],␣
 ↪how='outer')
df = pd.merge(merged_df1, melted_df3, on=['Country', 'Year'], how='outer')
df.to_csv('merged_df.csv',index=False)
```

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

[15]:
```
    Country    Year  Air Passenger Transport  Purchasing Power  \
0  Denmark  2011.0                2369405.0           11510.0
1  Ireland  2011.0                 113912.0            9977.0
2   Norway  2011.0               14116104.0           14519.0
3  Denmark  2012.0                1924590.0           11537.0
4  Ireland  2012.0                  53660.0            9912.0

   Air Goods Transport
0             155662.3
1             113408.7
2              65742.0
3             166283.0
4             126833.6
```

### 1.2 Data Summary

[17]:
```
df.describe()
```

[17]:
```
              Year  Air Passenger Transport  Purchasing Power  \
count    36.00000             3.600000e+01         36.000000
mean   2016.50000             5.273855e+06      13233.694444
std       3.50102             6.456175e+06       2276.235687
min    2011.00000             3.178300e+04       9912.000000
25%    2013.75000             9.840625e+04      11643.500000
50%    2016.50000             1.927521e+06      12693.000000
75%    2019.25000             1.411696e+07      15764.250000
max    2022.00000             1.592829e+07      16994.000000

       Air Goods Transport
count            36.000000
mean         164412.130556
```

```
std           46600.681687
min           65742.000000
25%          134538.200000
50%          156603.000000
75%          189143.000000
max          265889.000000
```

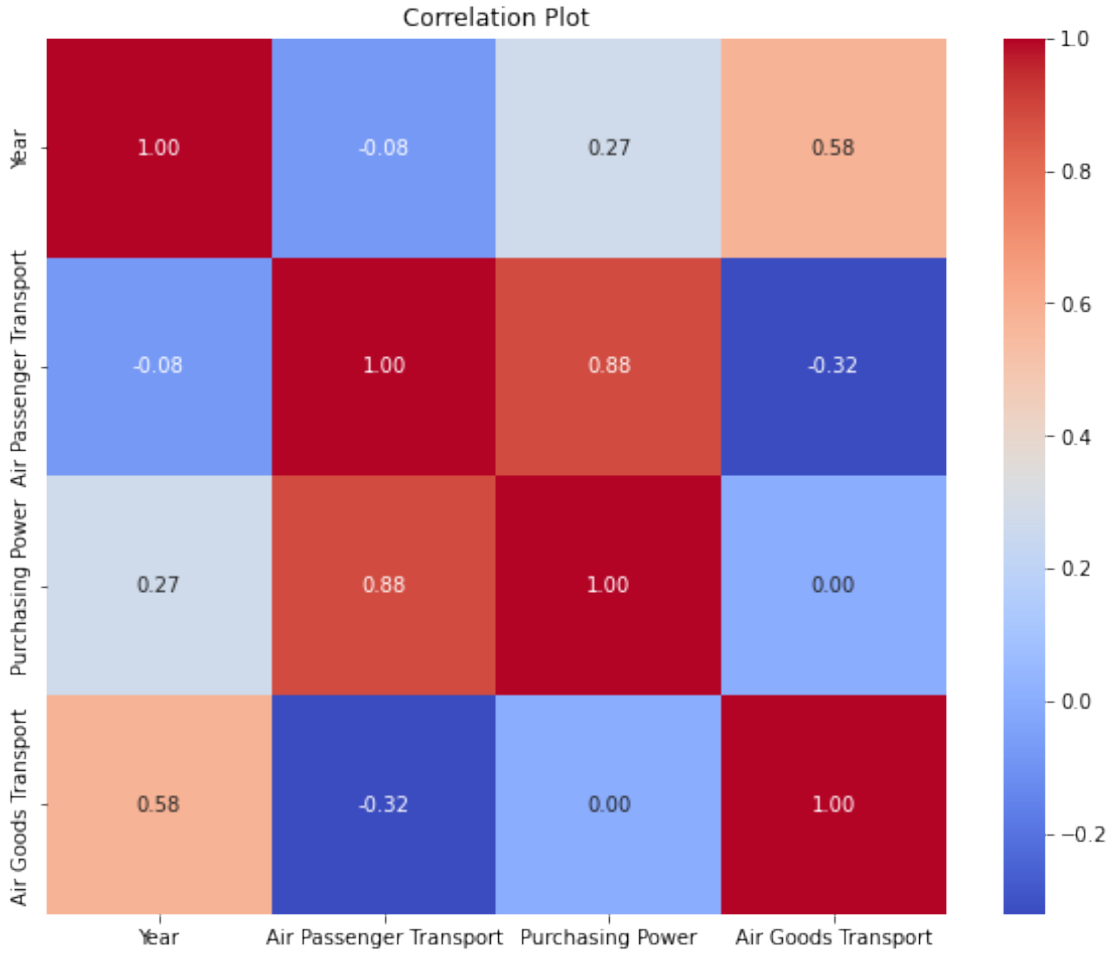
[19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 36 entries, 0 to 35
Data columns (total 5 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Country                  36 non-null     object
 1   Year                     36 non-null     float64
 2   Air Passenger Transport  36 non-null     float64
 3   Purchasing Power         36 non-null     float64
 4   Air Goods Transport      36 non-null     float64
dtypes: float64(4), object(1)
memory usage: 1.7+ KB
```

### 1.3 Correlation Analysis

In order to analyze the relationships between numeric variables, the correlation matrix and the pearson correlation has been found.

[21]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate the correlation matrix for the merged data frame 'final_merged_df'
correlation_df=df.drop(['Country'],axis=1)
correlation_matrix = correlation_df.dropna().corr()
correlation_matrix
correlation_matrix.to_csv('correlation_matrix.csv',index=True)
# Create a heatmap for the correlation matrix
plt.figure(figsize=(10, 8))  # Set the figure size
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Plot")
plt.show()
```




[23]:
```
correlation_matrix
```

[23]:
```
                            Year  Air Passenger Transport  Purchasing Power  \
Year                    1.000000                -0.076470          0.272571
Air Passenger Transport -0.076470                 1.000000          0.882492
Purchasing Power         0.272571                 0.882492          1.000000
Air Goods Transport      0.577949                -0.322073          0.000482

                        Air Goods Transport
Year                               0.577949
Air Passenger Transport           -0.322073
Purchasing Power                   0.000482
Air Goods Transport                1.000000
```

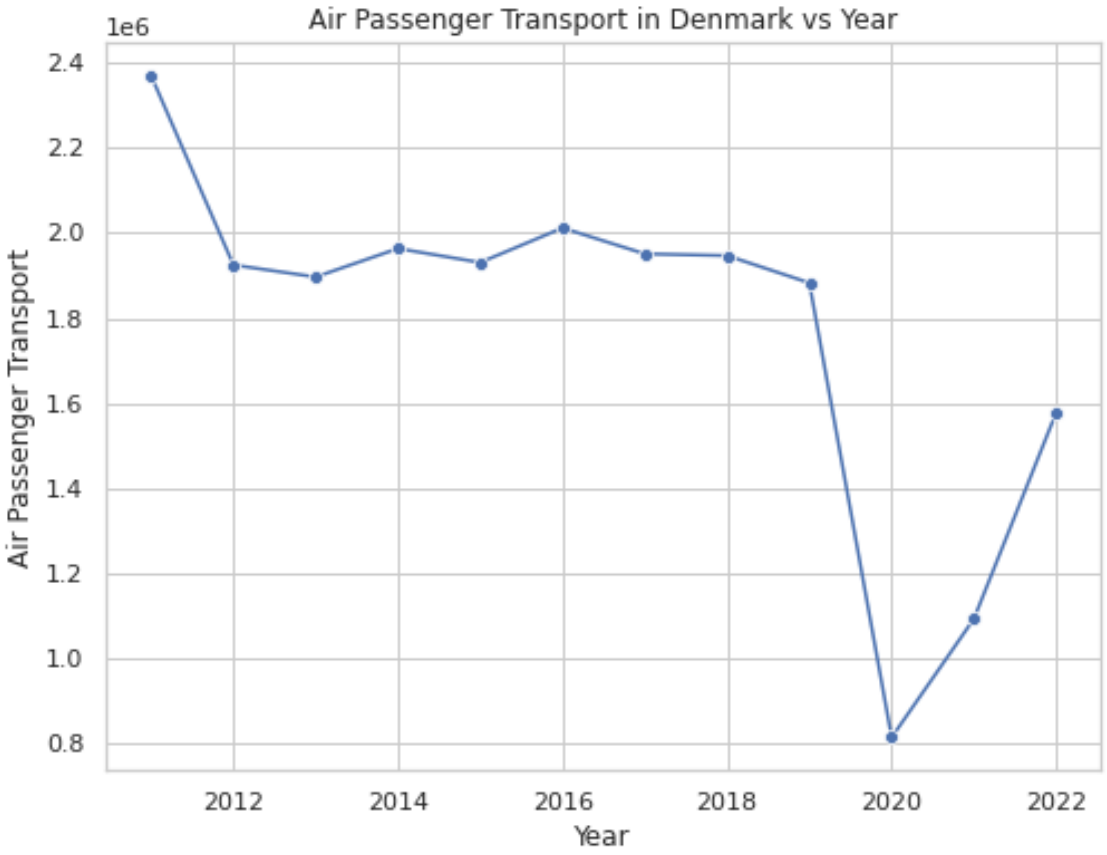
## 1.4 EDA

Responsible for Exploratory Data Analysis of various datasets.

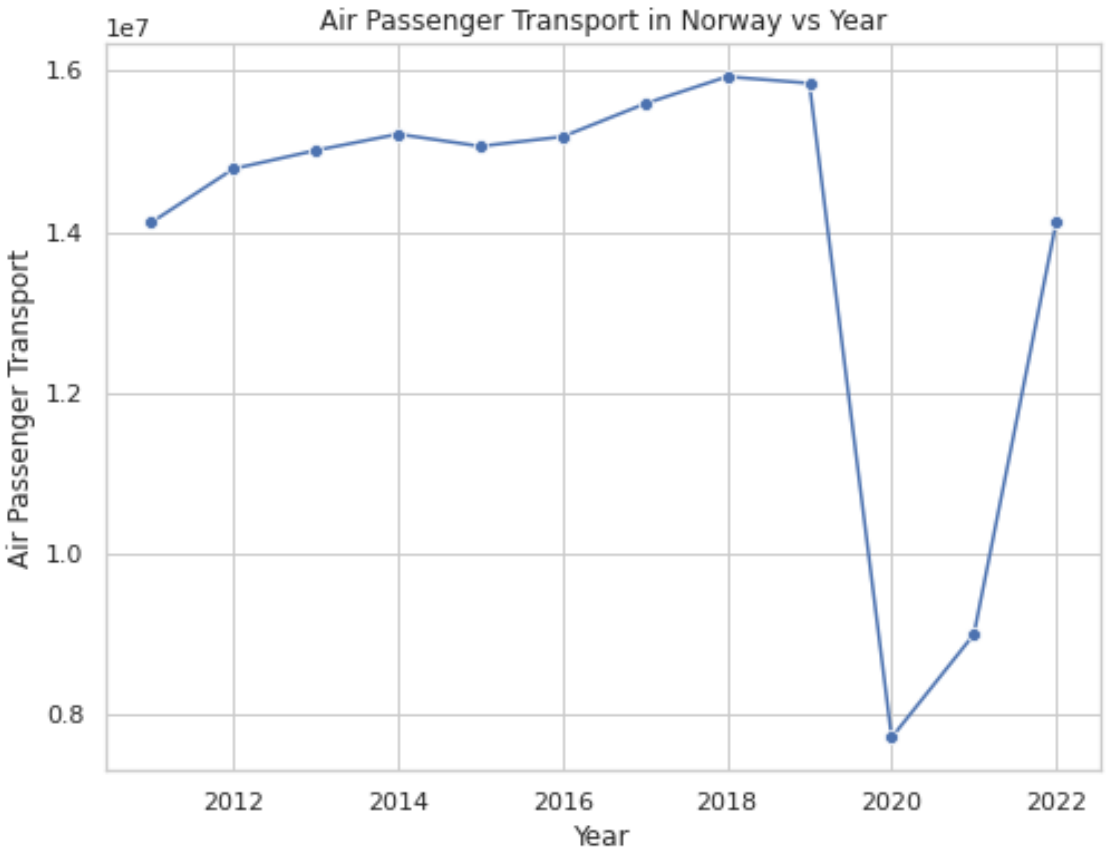
### 1.5 Air Passenger Transport by Country

Separate plots have been created for each country since Ireland's values were pretty close so it's better to visualizae the line plots for each country separately in order to better visualize the trends.

[25]:
```
# Get unique countries in the DataFrame
unique_countries = df['Country'].unique()

# Set the style of seaborn
sns.set(style="whitegrid")

# Create a separate line plot for each country
for country in unique_countries:
    plt.figure(figsize=(8, 6))
    country_data = df[df['Country'] == country]

    sns.lineplot(x="Year", y="Air Passenger Transport", data=country_data,␣
 ↪marker="o")
    plt.title(f"Air Passenger Transport in {country} vs Year")
    plt.xlabel("Year")
    plt.ylabel("Air Passenger Transport")
    plt.show()
```




Air Passenger Transport in Ireland vs Year
Air Passenger Transport
120000
100000
80000
60000
40000
2012
2014
2016
2018
2020
2022
Year

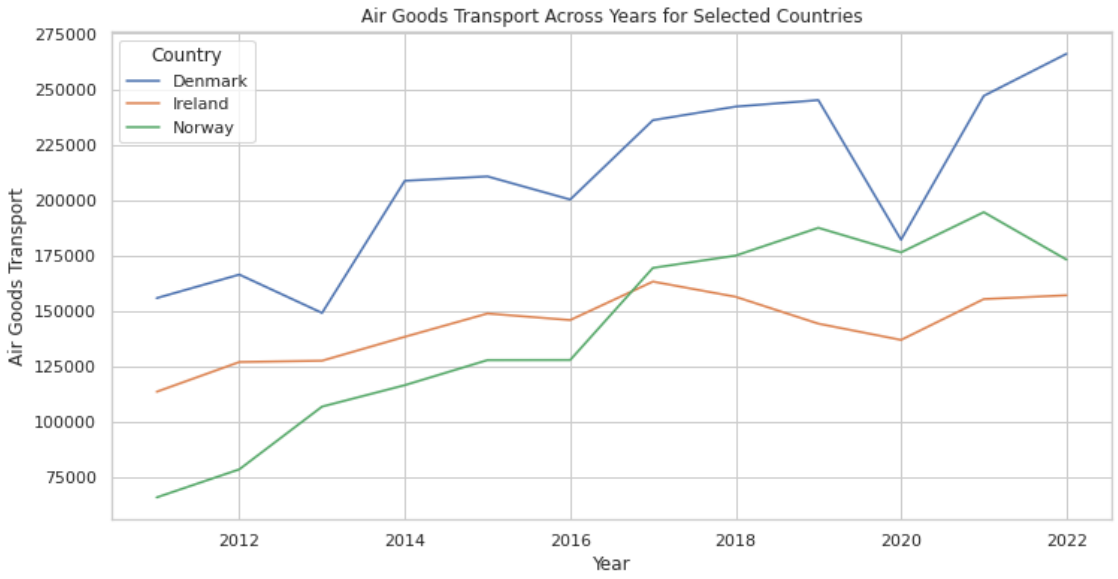



### 1.6 Air Goods Transport across the years

This script generates a Seaborn line plot to depict the trends in goods transport (measured in kilotons) for specified countries across different years. The dataset is filtered to include only the designated countries, and rows containing null values in the 'Good Transport(kt)' column are omitted. The line plot visually portrays the variations in goods transport values throughout the years, assigning a unique color to each country. The x-axis denotes the years, the y-axis illustrates the goods transport values in kilotons, and each line corresponds to a particular country. This setup facilitates a comparative examination of goods transport trends among the selected countries over time. The plot is configured with a title, x and y-axis labels, and a legend indicating the country associated with each line. The figure size is specified as (12, 6) to optimize the visual presentation.

[27]:
```
# Drop rows with null values in the 'Passenger Transport' column
filtered_df = df.dropna(subset=['Air Goods Transport'])
# Line plot for 'Good Transport (kt)' across the years for selected countries
plt.figure(figsize=(12, 6))
sns.lineplot(x='Year', y='Air Goods Transport', hue='Country', data=filtered_df)
plt.title('Air Goods Transport Across Years for Selected Countries')
plt.xlabel('Year')
plt.ylabel('Air Goods Transport')
```

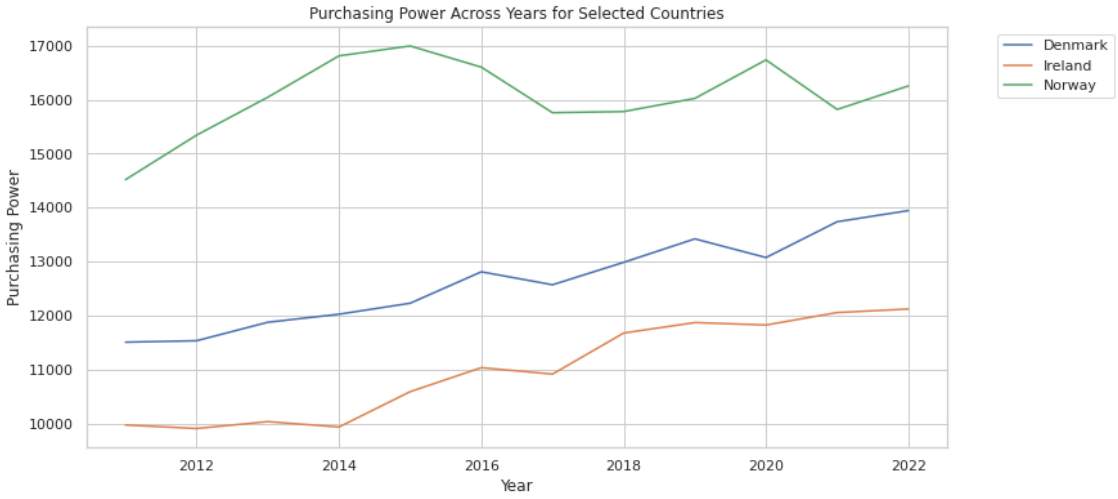
```
plt.show()
```




### 1.7 Purchasing Power across years for below 60% median equalized income

This visual representation illustrates the trajectory of Purchasing Power over the years for Denmark, Norway and Ireland with incomes below 60% of the median equalized income. The dataset has been refined to exclude entries with missing values in the 'Purchasing Power' category. The line plot showcases how Purchasing Power values evolve annually, distinguishing each country with a unique color. The x-axis denotes the respective years, while the y-axis represents the Purchasing Power values. This visualization provides a comparative analysis of how the Purchasing Power of selected countries, falling below 60% of the median equalized income, changes over time. The legend, positioned outside the plot for clarity, identifies each country's representation by color.

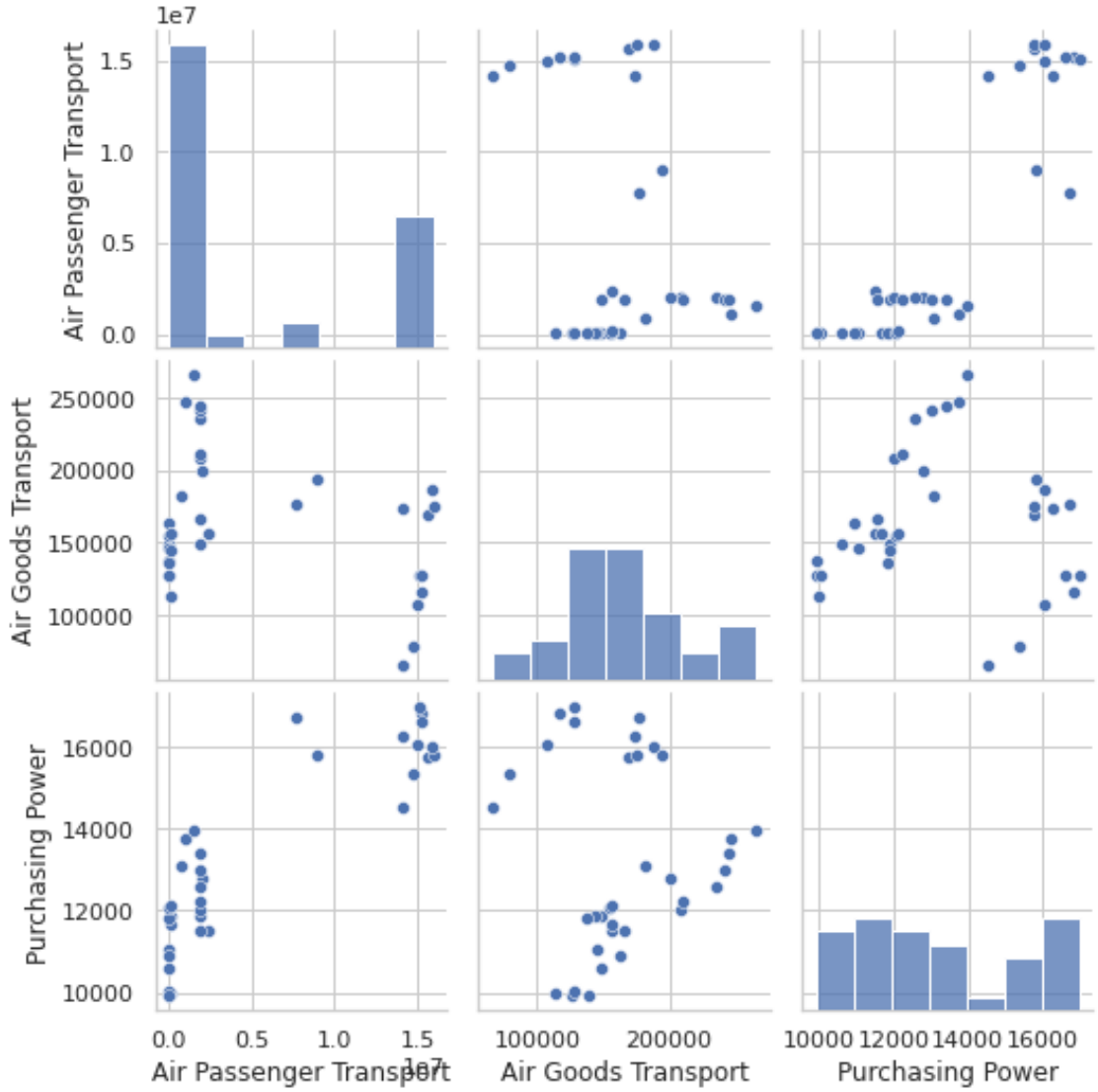
[29]:
```
filtered_df = df.dropna(subset=['Purchasing Power'])
# Line plot for 'Purchasing Power' across the years for selected countries
plt.figure(figsize=(12, 6))
sns.lineplot(x='Year', y='Purchasing Power', hue='Country', data=filtered_df)
plt.title('Purchasing Power Across Years for Selected Countries')
plt.xlabel('Year')
plt.ylabel('Purchasing Power')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')  # Move legend outside␣
↪the plot
plt.show()
```




## 1.8 Non-Linearity Determination

This code snippet aims to explore the potential non-linear relationships among the variables of interest, namely 'Air Passenger Transport,' 'Air Goods Transport,' and 'Purchasing Power.' The seaborn pair plot is employed for this purpose, generating scatter plots for each variable combination. The resulting visual display allows for the observation of patterns and trends in the data, aiding in the identification of potential non-linear correlations between the chosen variables. Each scatter plot represents the relationship between two variables, providing insights into their interactions and dependencies. The pair plot serves as a valuable tool for assessing the linearity or non-linearity of the selected variables in the dataset.

[31]:
```
# Select variables of interest
variables_of_interest= ['Air Passenger Transport','Air Goods Transport',␣
 ↪'Purchasing Power']
# Pair plot for scatter plots
sns.pairplot(df[variables_of_interest])
plt.show()
```




## 1.9 Hypothesis testing using ML model

*Hypothesis testing has been done in order to test the hypothesis:*

H0:There is no significant relationship between Passenger Transport, Good Transport, and the socio-economic factor (Purchasing Power).

H1: There is a significant relationship between Passenger Transport, Good Transport, and the socio-economic factor (Purchasing Power).

### 1.9.1 Regression Analysis Using Random Forest Models:

For the regression analysis, the Random Forest algorithm is employed to predict two crucial variables: Passenger Transport and Air Goods Transport. The comprehensive process is outlined below:

1. **Data Pre-processing:**
   - Missing values are handled, ensuring a clean dataset for analysis.
2. **Feature Standardization:**
   - Numerical features, including Air Passenger Transport, Air Goods Tribution, and Purchasig Power, are selected for standardization.
   - A StandardScaler is instantiated and fitted to transform the selected numerical features, ensuring uniform scales.
3. **Data Splitting:**
   - The dataset is divided into features (X) and target variables (y1 for Passenger Transport, y2 for Air Goods Transport).
4. **Train-Test Split:**
   - The data is further split into training and testing sets, allocating 20% for testing purposes.
5. **Random Forest Model Creation:**
   - Two Random Forest models, rf_model1 and rf_model2, are initialized with 100 decision trees each.
6. **Model Training:**
   - The models are trained on the designated training data (X_train, y1_train, y2_train).
7. **Prediction:**
   - Predictions are generated for both target variables (Passenger Transport and Air Goods Transport) using the test data (X_test).
8. **Evaluation:**
   - The Mean Squared Error (MSE) is calculated to assess the predictive performance of the models.
9. **Results Display:**
   - The MSE values for Passenger Transport and Air Goods Transport are printed, providing a quantitative measure of the models' accuracy. This regression analysis leverages the capabilities of the Random Forest algorithm to capture intricate relationships between the standardized numerical features and the target variables. Adjusting parameters and refining features further enhances the model's suitability for precise analytical requirements.

[63]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
df1=df.dropna()
# Select numerical features to standardize and use in the model
numerical_features = ['Air Passenger Transport', 'Air Goods Transport',␣
 ↪'Purchasing Power']
target_variable1 = 'Air Passenger Transport'
target_variable2 = 'Air Goods Transport'

# Create a StandardScaler object
scaler = StandardScaler()
# Fit the scaler on the numerical features and transform the data
df1[numerical_features] = scaler.fit_transform(df1[numerical_features])
```

```
# Split the data into features and target variables
X = df1[numerical_features]
y1 = df1[target_variable1]
y2 = df1[target_variable2]

# Split the data into training and testing sets
X_train, X_test, y1_train, y1_test, y2_train, y2_test = train_test_split(X, y1,␣
 ↪y2, test_size=0.2, random_state=42)

# Create Random Forest models
rf_model1 = RandomForestRegressor(random_state=42)
rf_model2 = RandomForestRegressor(random_state=42)

# Train the models
rf_model1.fit(X_train, y1_train)
rf_model2.fit(X_train, y2_train)

# Make predictions
y1_pred = rf_model1.predict(X_test)
y2_pred = rf_model2.predict(X_test)

# Evaluate the models
mse1 = mean_squared_error(y1_test, y1_pred)
mse2 = mean_squared_error(y2_test, y2_pred)

# Display the MSE
print(f'{target_variable1} MSE: {mse1}')
print(f'{target_variable2} MSE: {mse2}')
```

```
Air Passenger Transport MSE: 0.010477336284448708
Air Goods Transport MSE: 0.0029428281101532526
```

### 1.10 Ridge Regression with Polynomial Features

I have done Ridge Regression with polynomial features of degree 2 in order to evaluate and test the model.

[41]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import make_pipeline
import statsmodels.api as sm
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score
# Subset data and drop NaN values
df1 = df.dropna()
```

```
# Select numerical features to standardize and use in the model
independent_variables = ['Year', 'Air Passenger Transport', 'Air Goods␣
 ↪Transport', 'Purchasing Power']
target_variable1 = 'Air Passenger Transport'
target_variable2 = 'Air Goods Transport'
# Split the data into features and target variables
X = df1[independent_variables]
Y1 = df1[target_variable1]
Y2 = df1[target_variable2]
# Split the data into training and testing sets
x_train, x_test, y1_train, y1_test, y2_train, y2_test = train_test_split(X, Y1,␣
 ↪Y2, test_size=0.2, random_state=42)
# Create Ridge regression models with Polynomial Features
pr=PolynomialFeatures(degree=2)
x_test_pr=pr.fit_transform(x_test)
x_train_pr=pr.fit_transform(x_train)
ridge_model1=Ridge(alpha=0.1)
ridge_model2=Ridge(alpha=0.1)
# Train the models
ridge_model1.fit(x_train_pr, y1_train)
ridge_model2.fit(x_train_pr, y2_train)
# Make Predictions
y1_hat=ridge_model1.predict(x_test_pr)
y2_hat=ridge_model2.predict(x_test_pr)
# Print R^2 value
print('The R^2 value for', f'{target_variable1} is:',r2_score(y1_test,y1_hat))
print('The R^2 value for', f'{target_variable2} is:',r2_score(y2_test,y2_hat))

# Evaluate the models
mse1 = mean_squared_error(y1_test, y1_hat)
mse2 = mean_squared_error(y2_test, y2_hat)
# Print the mse
print('The Mean squared error for',f'{target_variable1} is:',mse1)
print('The Mean squared error for',f'{target_variable2} is:',mse2)
```

```
The R^2 value for Air Passenger Transport is: 1.0
The R^2 value for Air Goods Transport is: 0.9999999999999999
The Mean squared error for Air Passenger Transport is: 8.589406426569685e-10
The Mean squared error for Air Goods Transport is: 1.8988795547101237e-07
```

```
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-
packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning: Ill-conditioned
matrix (rcond=3.45197e-31): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-
packages/sklearn/linear_model/_ridge.py:157: LinAlgWarning: Ill-conditioned
matrix (rcond=3.45197e-31): result may not be accurate.
  return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

### 1.11 Time Series Forecasting for Denmark

[55]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
# Subset data for selected countries
selected_df = df[df['Country']=='Denmark']

# Extract relevant columns
data = selected_df[['Year','Air Passenger Transport', 'Air Goods Transport',␣
 ↪'Purchasing Power']]

# Use 'Year' as the index
data.set_index('Year', inplace=True)

# Normalize the data using StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Prepare data for LSTM
def create_dataset(data, time_steps=1):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

# Set the time steps for LSTM
time_steps = 5  # Increase the number of time steps

# Create LSTM datasets
X, y = create_dataset(scaled_data, time_steps)

# Reshape input data to be 3D [samples, time steps, features]
X = np.reshape(X, (X.shape[0], X.shape[1], data.shape[1]))

# Build the LSTM model with Dropout layers
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], X.
 ↪shape[2])))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(LSTM(units=50))
model.add(Dropout(0.2))  # Add Dropout layer
```

```
model.add(Dense(units=data.shape[1]))  # Output layer with the same number of␣
 ↪features

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X, y, epochs=100, batch_size=32)  # Increase the number of epochs

# Make predictions for multiple future time steps
num_forecast_steps = 10  # Set the number of forecast steps
forecast = []

for step in range(num_forecast_steps):
    last_data = scaled_data[-time_steps:]
    last_data = np.reshape(last_data, (1, time_steps, data.shape[1]))
    step_forecast = model.predict(last_data)
    scaled_data = np.vstack([scaled_data, step_forecast])
    forecast.append(step_forecast)

# Invert the scaling to get actual values
forecast = scaler.inverse_transform(np.array(forecast).
 ↪reshape(num_forecast_steps, data.shape[1]))

# Create a dataframe with the forecasted values
forecast_df = pd.DataFrame(forecast, columns=data.columns)
# Save to csv
forecast_df.to_csv('Denmark_forecasting.csv',index=False)
# Print or use forecast_df for further analysis
forecast_df
```

```
Epoch 1/100
1/1 [==============================] - 4s 4s/step - loss: 1.0231
Epoch 2/100
1/1 [==============================] - 0s 11ms/step - loss: 1.0129
Epoch 3/100
1/1 [==============================] - 0s 9ms/step - loss: 1.0006
Epoch 4/100
1/1 [==============================] - 0s 8ms/step - loss: 0.9873
Epoch 5/100
1/1 [==============================] - 0s 9ms/step - loss: 0.9661
Epoch 6/100
1/1 [==============================] - 0s 9ms/step - loss: 0.9546
Epoch 7/100
1/1 [==============================] - 0s 8ms/step - loss: 0.9383
Epoch 8/100
1/1 [==============================] - 0s 8ms/step - loss: 0.9269
```

```
Epoch 9/100
1/1 [==============================] - 0s 7ms/step - loss: 0.9050
Epoch 10/100
1/1 [==============================] - 0s 8ms/step - loss: 0.8739
Epoch 11/100
1/1 [==============================] - 0s 7ms/step - loss: 0.8722
Epoch 12/100
1/1 [==============================] - 0s 8ms/step - loss: 0.8473
Epoch 13/100
1/1 [==============================] - 0s 9ms/step - loss: 0.7980
Epoch 14/100
1/1 [==============================] - 0s 9ms/step - loss: 0.7943
Epoch 15/100
1/1 [==============================] - 0s 7ms/step - loss: 0.7399
Epoch 16/100
1/1 [==============================] - 0s 7ms/step - loss: 0.7244
Epoch 17/100
1/1 [==============================] - 0s 7ms/step - loss: 0.6951
Epoch 18/100
1/1 [==============================] - 0s 7ms/step - loss: 0.6830
Epoch 19/100
1/1 [==============================] - 0s 10ms/step - loss: 0.6114
Epoch 20/100
1/1 [==============================] - 0s 7ms/step - loss: 0.5726
Epoch 21/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4864
Epoch 22/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4789
Epoch 23/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4769
Epoch 24/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4140
Epoch 25/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4212
Epoch 26/100
1/1 [==============================] - 0s 9ms/step - loss: 0.3650
Epoch 27/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4362
Epoch 28/100
1/1 [==============================] - 0s 9ms/step - loss: 0.4621
Epoch 29/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4567
Epoch 30/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4488
Epoch 31/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3880
Epoch 32/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4042
```

```
Epoch 33/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3686
Epoch 34/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3863
Epoch 35/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3516
Epoch 36/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3621
Epoch 37/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3442
Epoch 38/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4472
Epoch 39/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4012
Epoch 40/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3467
Epoch 41/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3842
Epoch 42/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3904
Epoch 43/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3986
Epoch 44/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3839
Epoch 45/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3424
Epoch 46/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3752
Epoch 47/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3765
Epoch 48/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3795
Epoch 49/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3622
Epoch 50/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3724
Epoch 51/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3826
Epoch 52/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3663
Epoch 53/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3350
Epoch 54/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3755
Epoch 55/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3661
Epoch 56/100
1/1 [==============================] - 0s 10ms/step - loss: 0.3257
```

```
Epoch 57/100
1/1 [==============================] - 0s 10ms/step - loss: 0.3481
Epoch 58/100
1/1 [==============================] - 0s 9ms/step - loss: 0.3699
Epoch 59/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3713
Epoch 60/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3590
Epoch 61/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3287
Epoch 62/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3277
Epoch 63/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3080
Epoch 64/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3938
Epoch 65/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4405
Epoch 66/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3033
Epoch 67/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3205
Epoch 68/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3542
Epoch 69/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3666
Epoch 70/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3304
Epoch 71/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3678
Epoch 72/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3126
Epoch 73/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3383
Epoch 74/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3413
Epoch 75/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3383
Epoch 76/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3297
Epoch 77/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3463
Epoch 78/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3501
Epoch 79/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3760
Epoch 80/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2856
```

```
Epoch 81/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2854
Epoch 82/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3233
Epoch 83/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3555
Epoch 84/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3643
Epoch 85/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3395
Epoch 86/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3391
Epoch 87/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3408
Epoch 88/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2894
Epoch 89/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3134
Epoch 90/100
1/1 [==============================] - 0s 14ms/step - loss: 0.3078
Epoch 91/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3271
Epoch 92/100
1/1 [==============================] - 0s 8ms/step - loss: 0.2709
Epoch 93/100
1/1 [==============================] - 0s 8ms/step - loss: 0.2736
Epoch 94/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3540
Epoch 95/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3130
Epoch 96/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3120
Epoch 97/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3243
Epoch 98/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3010
Epoch 99/100
1/1 [==============================] - 0s 8ms/step - loss: 0.2648
Epoch 100/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2712
1/1 [==============================] - 1s 800ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
```

```
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 17ms/step
```

[55]:

| | Air Passenger Transport | Air Goods Transport | Purchasing Power |
|---|---|---|---|
| 0 | 1418820.875 | 258700.281250 | 13896.488281 |
| 1 | 1565012.625 | 257126.484375 | 13695.178711 |
| 2 | 1725798.750 | 249141.656250 | 13347.365234 |
| 3 | 1409601.750 | 270106.781250 | 14078.376953 |
| 4 | 1223558.500 | 277144.781250 | 14406.267578 |
| 5 | 1264029.625 | 273830.812500 | 14301.406250 |
| 6 | 1246801.375 | 275553.500000 | 14353.556641 |
| 7 | 1280804.750 | 279554.218750 | 14394.494141 |
| 8 | 1332367.625 | 286820.281250 | 14469.216797 |
| 9 | 1346957.375 | 287961.375000 | 14471.683594 |

### 1.12 Time Series Forecasting for Norway

[57]:
```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
# Subset data for selected countries
selected_df = df[df['Country']=='Norway']

# Extract relevant columns
data = selected_df[['Year','Air Passenger Transport', 'Air Goods Transport',␣
 ↪'Purchasing Power']]

# Use 'Year' as the index
data.set_index('Year', inplace=True)

# Normalize the data using StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Prepare data for LSTM
def create_dataset(data, time_steps=1):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

# Set the time steps for LSTM
time_steps = 5  # Increase the number of time steps
```

```
# Create LSTM datasets
X, y = create_dataset(scaled_data, time_steps)

# Reshape input data to be 3D [samples, time steps, features]
X = np.reshape(X, (X.shape[0], X.shape[1], data.shape[1]))

# Build the LSTM model with Dropout layers
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], X.
 ↪shape[2])))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(LSTM(units=50))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(Dense(units=data.shape[1]))  # Output layer with the same number of␣
 ↪features

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X, y, epochs=100, batch_size=32)  # Increase the number of epochs

# Make predictions for multiple future time steps
num_forecast_steps = 10  # Set the number of forecast steps
forecast = []

for step in range(num_forecast_steps):
    last_data = scaled_data[-time_steps:]
    last_data = np.reshape(last_data, (1, time_steps, data.shape[1]))
    step_forecast = model.predict(last_data)
    scaled_data = np.vstack([scaled_data, step_forecast])
    forecast.append(step_forecast)

# Invert the scaling to get actual values
forecast = scaler.inverse_transform(np.array(forecast).
 ↪reshape(num_forecast_steps, data.shape[1]))

# Create a dataframe with the forecasted values
forecast_df = pd.DataFrame(forecast, columns=data.columns)
# Save to csv
forecast_df.to_csv('Norway_forecasting.csv',index=False)
# Print or use forecast_df for further analysis
forecast_df
```

```
Epoch 1/100
```

```
1/1 [==============================] - 4s 4s/step - loss: 0.8924
Epoch 2/100
1/1 [==============================] - 0s 14ms/step - loss: 0.8950
Epoch 3/100
1/1 [==============================] - 0s 9ms/step - loss: 0.8811
Epoch 4/100
1/1 [==============================] - 0s 9ms/step - loss: 0.8718
Epoch 5/100
1/1 [==============================] - 0s 9ms/step - loss: 0.8612
Epoch 6/100
1/1 [==============================] - 0s 10ms/step - loss: 0.8428
Epoch 7/100
1/1 [==============================] - 0s 9ms/step - loss: 0.8365
Epoch 8/100
1/1 [==============================] - 0s 9ms/step - loss: 0.8274
Epoch 9/100
1/1 [==============================] - 0s 12ms/step - loss: 0.8137
Epoch 10/100
1/1 [==============================] - 0s 10ms/step - loss: 0.7889
Epoch 11/100
1/1 [==============================] - 0s 11ms/step - loss: 0.7760
Epoch 12/100
1/1 [==============================] - 0s 10ms/step - loss: 0.7584
Epoch 13/100
1/1 [==============================] - 0s 11ms/step - loss: 0.7523
Epoch 14/100
1/1 [==============================] - 0s 12ms/step - loss: 0.7465
Epoch 15/100
1/1 [==============================] - 0s 12ms/step - loss: 0.7114
Epoch 16/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6976
Epoch 17/100
1/1 [==============================] - 0s 11ms/step - loss: 0.7278
Epoch 18/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6542
Epoch 19/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6485
Epoch 20/100
1/1 [==============================] - 0s 10ms/step - loss: 0.6276
Epoch 21/100
1/1 [==============================] - 0s 10ms/step - loss: 0.5977
Epoch 22/100
1/1 [==============================] - 0s 10ms/step - loss: 0.5838
Epoch 23/100
1/1 [==============================] - 0s 10ms/step - loss: 0.5719
Epoch 24/100
1/1 [==============================] - 0s 10ms/step - loss: 0.5452
Epoch 25/100
```

```
1/1 [==============================] - 0s 11ms/step - loss: 0.5707
Epoch 26/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4577
Epoch 27/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4867
Epoch 28/100
1/1 [==============================] - 0s 10ms/step - loss: 0.5986
Epoch 29/100
1/1 [==============================] - 0s 11ms/step - loss: 0.4825
Epoch 30/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4497
Epoch 31/100
1/1 [==============================] - 0s 11ms/step - loss: 0.5070
Epoch 32/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4665
Epoch 33/100
1/1 [==============================] - 0s 15ms/step - loss: 0.4184
Epoch 34/100
1/1 [==============================] - 0s 14ms/step - loss: 0.4621
Epoch 35/100
1/1 [==============================] - 0s 11ms/step - loss: 0.4680
Epoch 36/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4283
Epoch 37/100
1/1 [==============================] - 0s 11ms/step - loss: 0.4292
Epoch 38/100
1/1 [==============================] - 0s 15ms/step - loss: 0.4201
Epoch 39/100
1/1 [==============================] - 0s 11ms/step - loss: 0.4817
Epoch 40/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4220
Epoch 41/100
1/1 [==============================] - 0s 11ms/step - loss: 0.3659
Epoch 42/100
1/1 [==============================] - 0s 16ms/step - loss: 0.4447
Epoch 43/100
1/1 [==============================] - 0s 16ms/step - loss: 0.3899
Epoch 44/100
1/1 [==============================] - 0s 10ms/step - loss: 0.4797
Epoch 45/100
1/1 [==============================] - 0s 9ms/step - loss: 0.3860
Epoch 46/100
1/1 [==============================] - 0s 9ms/step - loss: 0.3620
Epoch 47/100
1/1 [==============================] - 0s 11ms/step - loss: 0.3603
Epoch 48/100
1/1 [==============================] - 0s 14ms/step - loss: 0.3855
Epoch 49/100
```

```
1/1 [==============================] - 0s 11ms/step - loss: 0.3414
Epoch 50/100
1/1 [==============================] - 0s 9ms/step - loss: 0.3503
Epoch 51/100
1/1 [==============================] - 0s 16ms/step - loss: 0.3553
Epoch 52/100
1/1 [==============================] - 0s 12ms/step - loss: 0.3427
Epoch 53/100
1/1 [==============================] - 0s 11ms/step - loss: 0.2954
Epoch 54/100
1/1 [==============================] - 0s 29ms/step - loss: 0.3268
Epoch 55/100
1/1 [==============================] - 0s 22ms/step - loss: 0.3611
Epoch 56/100
1/1 [==============================] - 0s 17ms/step - loss: 0.3690
Epoch 57/100
1/1 [==============================] - 0s 12ms/step - loss: 0.2249
Epoch 58/100
1/1 [==============================] - 0s 17ms/step - loss: 0.3140
Epoch 59/100
1/1 [==============================] - 0s 19ms/step - loss: 0.3062
Epoch 60/100
1/1 [==============================] - 0s 12ms/step - loss: 0.3055
Epoch 61/100
1/1 [==============================] - 0s 15ms/step - loss: 0.2348
Epoch 62/100
1/1 [==============================] - 0s 15ms/step - loss: 0.3374
Epoch 63/100
1/1 [==============================] - 0s 11ms/step - loss: 0.2693
Epoch 64/100
1/1 [==============================] - 0s 17ms/step - loss: 0.2970
Epoch 65/100
1/1 [==============================] - 0s 11ms/step - loss: 0.1904
Epoch 66/100
1/1 [==============================] - 0s 14ms/step - loss: 0.2536
Epoch 67/100
1/1 [==============================] - 0s 11ms/step - loss: 0.1534
Epoch 68/100
1/1 [==============================] - 0s 11ms/step - loss: 0.2218
Epoch 69/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1877
Epoch 70/100
1/1 [==============================] - 0s 9ms/step - loss: 0.2208
Epoch 71/100
1/1 [==============================] - 0s 10ms/step - loss: 0.2395
Epoch 72/100
1/1 [==============================] - 0s 14ms/step - loss: 0.1354
Epoch 73/100
```

```
1/1 [==============================] - 0s 10ms/step - loss: 0.2301
Epoch 74/100
1/1 [==============================] - 0s 10ms/step - loss: 0.2078
Epoch 75/100
1/1 [==============================] - 0s 11ms/step - loss: 0.2033
Epoch 76/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1967
Epoch 77/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1845
Epoch 78/100
1/1 [==============================] - 0s 11ms/step - loss: 0.1542
Epoch 79/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1677
Epoch 80/100
1/1 [==============================] - 0s 13ms/step - loss: 0.1460
Epoch 81/100
1/1 [==============================] - 0s 13ms/step - loss: 0.0961
Epoch 82/100
1/1 [==============================] - 0s 23ms/step - loss: 0.2038
Epoch 83/100
1/1 [==============================] - 0s 11ms/step - loss: 0.0912
Epoch 84/100
1/1 [==============================] - 0s 14ms/step - loss: 0.1661
Epoch 85/100
1/1 [==============================] - 0s 10ms/step - loss: 0.0858
Epoch 86/100
1/1 [==============================] - 0s 12ms/step - loss: 0.0960
Epoch 87/100
1/1 [==============================] - 0s 11ms/step - loss: 0.0833
Epoch 88/100
1/1 [==============================] - 0s 10ms/step - loss: 0.1334
Epoch 89/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0892
Epoch 90/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0890
Epoch 91/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1233
Epoch 92/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1035
Epoch 93/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0831
Epoch 94/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1392
Epoch 95/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1338
Epoch 96/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0807
Epoch 97/100
```

```
1/1 [==============================] - 0s 13ms/step - loss: 0.0474
Epoch 98/100
1/1 [==============================] - 0s 11ms/step - loss: 0.2463
Epoch 99/100
1/1 [==============================] - 0s 13ms/step - loss: 0.0574
Epoch 100/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1147
1/1 [==============================] - 1s 854ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 26ms/step
```

[57]:

| | Air Passenger Transport | Air Goods Transport | Purchasing Power |
|---|---|---|---|
| 0 | 14004339.0 | 155740.203125 | 16092.397461 |
| 1 | 14566069.0 | 147992.468750 | 16070.138672 |
| 2 | 15010255.0 | 140918.859375 | 16069.227539 |
| 3 | 14545103.0 | 141532.390625 | 16165.734375 |
| 4 | 12132781.0 | 168806.046875 | 16050.342773 |
| 5 | 12640071.0 | 165559.000000 | 16042.933594 |
| 6 | 12744108.0 | 166119.406250 | 16029.267578 |
| 7 | 13196537.0 | 162829.078125 | 16018.729492 |
| 8 | 13834903.0 | 155529.515625 | 16036.041992 |
| 9 | 14391537.0 | 146063.781250 | 16115.030273 |

### 1.13 Time Series Forecasting for Ireland

[65]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Subset data for selected countries
selected_df = df[df['Country']=='Ireland']

# Extract relevant columns
data = selected_df[['Year','Air Passenger Transport', 'Air Goods Transport',␣
 ↪'Purchasing Power']]

# Use 'Year' as the index
data.set_index('Year', inplace=True)
```

```
# Normalize the data using MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# Prepare data for LSTM
def create_dataset(data, time_steps=1):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

# Set the time steps for LSTM
time_steps = 5  # Increase the number of time steps

# Create LSTM datasets
X, y = create_dataset(scaled_data, time_steps)

# Reshape input data to be 3D [samples, time steps, features]
X = np.reshape(X, (X.shape[0], X.shape[1], data.shape[1]))

# Build the LSTM model with Dropout layers
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], X.
 ↪shape[2])))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(LSTM(units=50))
model.add(Dropout(0.2))  # Add Dropout layer
model.add(Dense(units=data.shape[1]))  # Output layer with the same number of␣
 ↪features

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X, y, epochs=100, batch_size=32)  # Increase the number of epochs

# Make predictions for multiple future time steps
num_forecast_steps = 10  # Set the number of forecast steps
forecast = []

for step in range(num_forecast_steps):
    last_data = scaled_data[-time_steps:]
    last_data = np.reshape(last_data, (1, time_steps, data.shape[1]))
```

```
    step_forecast = model.predict(last_data)
    scaled_data = np.vstack([scaled_data, step_forecast])
    forecast.append(step_forecast)

# Invert the scaling to get actual values
forecast = scaler.inverse_transform(np.array(forecast).
 ↪reshape(num_forecast_steps, data.shape[1]))

# Create a dataframe with the forecasted values
forecast_df = pd.DataFrame(forecast, columns=data.columns)
# Save to csv
forecast_df.to_csv('Ireland_forecasting.csv', index=False)
# Print or use forecast_df for further analysis
forecast_df
```

```
Epoch 1/100
1/1 [==============================] - 4s 4s/step - loss: 0.5494
Epoch 2/100
1/1 [==============================] - 0s 10ms/step - loss: 0.5403
Epoch 3/100
1/1 [==============================] - 0s 9ms/step - loss: 0.5206
Epoch 4/100
1/1 [==============================] - 0s 9ms/step - loss: 0.5058
Epoch 5/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4853
Epoch 6/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4751
Epoch 7/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4516
Epoch 8/100
1/1 [==============================] - 0s 8ms/step - loss: 0.4351
Epoch 9/100
1/1 [==============================] - 0s 7ms/step - loss: 0.4122
Epoch 10/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3902
Epoch 11/100
1/1 [==============================] - 0s 7ms/step - loss: 0.3559
Epoch 12/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3452
Epoch 13/100
1/1 [==============================] - 0s 8ms/step - loss: 0.3280
Epoch 14/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2936
Epoch 15/100
1/1 [==============================] - 0s 7ms/step - loss: 0.2476
Epoch 16/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1817
```

```
Epoch 17/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1598
Epoch 18/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1158
Epoch 19/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0998
Epoch 20/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1000
Epoch 21/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0875
Epoch 22/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0910
Epoch 23/100
1/1 [==============================] - 0s 9ms/step - loss: 0.1376
Epoch 24/100
1/1 [==============================] - 0s 8ms/step - loss: 0.1525
Epoch 25/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1032
Epoch 26/100
1/1 [==============================] - 0s 8ms/step - loss: 0.1441
Epoch 27/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0837
Epoch 28/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0902
Epoch 29/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0786
Epoch 30/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0653
Epoch 31/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0866
Epoch 32/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0698
Epoch 33/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0839
Epoch 34/100
1/1 [==============================] - 0s 12ms/step - loss: 0.0852
Epoch 35/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0897
Epoch 36/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0974
Epoch 37/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0647
Epoch 38/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0636
Epoch 39/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0735
Epoch 40/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0817
```

```
Epoch 41/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0697
Epoch 42/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0884
Epoch 43/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0925
Epoch 44/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0924
Epoch 45/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0713
Epoch 46/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0905
Epoch 47/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0645
Epoch 48/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0673
Epoch 49/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0438
Epoch 50/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1000
Epoch 51/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0852
Epoch 52/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0687
Epoch 53/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0569
Epoch 54/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0534
Epoch 55/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0714
Epoch 56/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0616
Epoch 57/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0830
Epoch 58/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0655
Epoch 59/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0715
Epoch 60/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0798
Epoch 61/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0571
Epoch 62/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0681
Epoch 63/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0818
Epoch 64/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0600
```

```
Epoch 65/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0820
Epoch 66/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0909
Epoch 67/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0471
Epoch 68/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0822
Epoch 69/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0944
Epoch 70/100
1/1 [==============================] - 0s 9ms/step - loss: 0.0846
Epoch 71/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0670
Epoch 72/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0653
Epoch 73/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0843
Epoch 74/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0760
Epoch 75/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0735
Epoch 76/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1018
Epoch 77/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0837
Epoch 78/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0743
Epoch 79/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0737
Epoch 80/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0762
Epoch 81/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0859
Epoch 82/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0644
Epoch 83/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0654
Epoch 84/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0572
Epoch 85/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0619
Epoch 86/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0644
Epoch 87/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0715
Epoch 88/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0695
```

```
Epoch 89/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0678
Epoch 90/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0595
Epoch 91/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0553
Epoch 92/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0561
Epoch 93/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0615
Epoch 94/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0799
Epoch 95/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0692
Epoch 96/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0663
Epoch 97/100
1/1 [==============================] - 0s 8ms/step - loss: 0.0623
Epoch 98/100
1/1 [==============================] - 0s 15ms/step - loss: 0.0560
Epoch 99/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0571
Epoch 100/100
1/1 [==============================] - 0s 7ms/step - loss: 0.0616
1/1 [==============================] - 1s 770ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
```

[65]:

| | Air Passenger Transport | Air Goods Transport | Purchasing Power |
|---|---|---|---|
| 0 | 77714.460938 | 150907.625000 | 11861.805664 |
| 1 | 74657.679688 | 148065.062500 | 11729.854492 |
| 2 | 72288.656250 | 146263.375000 | 11655.519531 |
| 3 | 76424.898438 | 150481.750000 | 11879.756836 |
| 4 | 79843.054688 | 153317.406250 | 12000.525391 |
| 5 | 76836.835938 | 150155.890625 | 11820.367188 |
| 6 | 76588.343750 | 149872.734375 | 11803.055664 |
| 7 | 76626.335938 | 150037.906250 | 11819.701172 |
| 8 | 77021.859375 | 150647.015625 | 11865.952148 |
| 9 | 77182.609375 | 150785.093750 | 11871.676758 |

## 2 Sentimental Analysis

[43]:
```
pip install praw
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Collecting praw
  Downloading praw-7.7.1-py3-none-any.whl (191 kB)
     |                  | 191 kB 20.5 MB/s eta 0:00:01
Collecting update-checker>=0.18
  Downloading update_checker-0.18.0-py3-none-any.whl (7.0 kB)
Requirement already satisfied: websocket-client>=0.54.0 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from praw)
(0.58.0)
Collecting prawcore<3,>=2.1
  Downloading prawcore-2.4.0-py3-none-any.whl (17 kB)
Requirement already satisfied: requests<3.0,>=2.6.0 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
prawcore<3,>=2.1->praw) (2.27.1)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests<3.0,>=2.6.0->prawcore<3,>=2.1->praw) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests<3.0,>=2.6.0->prawcore<3,>=2.1->praw) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests<3.0,>=2.6.0->prawcore<3,>=2.1->praw) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
requests<3.0,>=2.6.0->prawcore<3,>=2.1->praw) (1.26.9)
Requirement already satisfied: six in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
websocket-client>=0.54.0->praw) (1.16.0)
Installing collected packages: update-checker, prawcore, praw
Successfully installed praw-7.7.1 prawcore-2.4.0 update-checker-0.18.0
Note: you may need to restart the kernel to use updated packages.
```

[45]:
```
import praw
user_agent = 'Reddit_Scrapper 1.0 by /u//FeatureChoice5036'
reddit = praw.Reddit(
 client_id= 'VPVTJwFmCORKXfSHiTVZLw',
 client_secret='rLWIgS6xLk13CHuo3jscWlbPnu6JJg',
 user_agent=user_agent
)
```

[47]:
```
import pandas as pd
import numpy as np
import re #RegEx : Regular expression
```

[85]:
```
import praw
from nltk.sentiment import SentimentIntensityAnalyzer

# Set up your Reddit API credentials and user agent
user_agent = 'Reddit_Scrapper 1.0 by /u//FeatureChoice5036'
reddit = praw.Reddit(
    client_id='VPVTJwFmC0RKXfSHiTVZLw' ,
    client_secret = 'rLWIgS6xLk13CHuo3jscWlbPnu6JJg' ,

    user_agent=user_agent
)

# List of airports to search for
airports = ['Dublin Airport', 'Cork Airport', 'Shannon Airport', 'Knock␣
 ↪Airport', 'Kerry Airport', 'Donegal Airport']


# Initialize the SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# Function to get the sentiment score for a given text
def get_sentiment_score(text):
    return sia.polarity_scores(text)['compound']

# Search for sentiments related to each airport
for airport in airports:
    print(f"Sentiments for {airport}:")
    for submission in reddit.subreddit('all').search(airport,␣
 ↪time_filter='month', sort='relevance', limit=5):
        sentiment_score = get_sentiment_score(submission.title)
        print(f"Title: {submission.title}")
        print(f"Sentiment Score: {sentiment_score}")
        print("---")

# Note: Replace 'YOUR_CLIENT_ID' and 'YOUR_CLIENT_SECRET' with your actual␣
 ↪Reddit API credentials.
```

```
Sentiments for Dublin Airport:
Title: Bad taste in my mouth in Dublin Airport
Sentiment Score: -0.5423
---
Title: Traveling from and to Dublin Airport
Sentiment Score: 0.0
```

---
Title: EU will be ensuring we've an IE rail connection to Dublin airport.
Sentiment Score: 0.2732
---
Title: Dublin airport loonies..
Sentiment Score: 0.0
---
Title: Spotted this in Dublin Airport today
Sentiment Score: 0.0
---
Sentiments for Cork Airport:
Title: Cork Airport
Sentiment Score: 0.0
---
Title: Cork Airport at 6 AM?
Sentiment Score: 0.0
---
Title: Did Cork Airport ever fly to these places?
Sentiment Score: 0.0
---
Title: Cork Airport - Can I just sleep there overnight?
Sentiment Score: 0.0
---
Title: If you could add some direct routes from Cork Airport where would you choose?
Sentiment Score: 0.0
---
Sentiments for Shannon Airport:
Title: Shannon airport
Sentiment Score: 0.0
---
Title: Shannon Airport Bus
Sentiment Score: 0.0
---
Title: Getting to Shannon Airport
Sentiment Score: 0.0
---
Title: Shannon Airport in Ireland, runway is imaged like a Christmas tree.
Sentiment Score: 0.3612
---
Title: Traveling from Ireland to London to Miami- Preclearance??
Sentiment Score: 0.0
---
Sentiments for Knock Airport:
Title: Knock Airport in Ireland has given me a…concerning welcome..
Sentiment Score: 0.0
---
Title: Joe Rogan knocks Biden over Trump’s claim of airports during

```
Revolutionary War
Sentiment Score: -0.5994
---
Title: Joe Rogan knocks Biden over Trump's claim of airports during
Revolutionary War
Sentiment Score: -0.5994
---
Title: What happens if you knock off your door on the way to the airport
Sentiment Score: 0.0
---
Title: Joe Rogan knocks Biden over Trump's claim of airports during
Revolutionary War
Sentiment Score: -0.5994
---
Sentiments for Kerry Airport:
Title: Kerry Airport parking; is it safe?
Sentiment Score: 0.4404
---
Title: Help! Central coastal city between Kenmare and Cork
Sentiment Score: 0.4574
---
Title: Killarney without a car?
Sentiment Score: 0.0
---
Title: 22 year old American looking for a rental car in/near Killarney
Sentiment Score: 0.0
---
Title: 6 nights in early March: are we doing too much?
Sentiment Score: 0.0
---
Sentiments for Donegal Airport:
Title: Me coming up with a fun idea for a rail expansion across Ireland
Sentiment Score: 0.5106
---
Title: Several more High Power sites live today
Sentiment Score: 0.0
---
Title: How much would it cost to upgrade just the train track from Cork to
Dublin to high speed similar to Japan? 320km/h so the journey would be less than
1 hour.
Sentiment Score: 0.0
---
Title: Let me rant to yall for a sec about the Chicago show
Sentiment Score: -0.34
---
Title: Our (completed) 3 week road trip itinerary
Sentiment Score: 0.0
---
```

### 2.1 Create a Sentiment Dataset for non-zero Sentiments

First individual datasets for each airport have been created and then it has been combined into one

[157]:
```
import pandas as pd

# Sample data
data_dublin = [
    {'Airport': 'Dublin Airport', 'Title': 'Bad taste in my mouth in Dublin␣
 ↪Airport', 'Sentiment Score': -0.5423},
    {'Airport': 'Dublin Airport', 'Title': "EU will be ensuring we've an IE␣
 ↪rail connection to Dublin airport", 'Sentiment Score': 0.2732}
]

data_kerry = [
    {'Airport': 'Kerry Airport', 'Title': 'Kerry Airport parking; is it safe?',␣
 ↪'Sentiment Score': 0.4404},
    {'Airport': 'Kerry Airport', 'Title': 'Help! Central coastal city between␣
 ↪Kenmare and Cork', 'Sentiment Score': 0.4574}
]
data_knock = [
    {'Airport': 'Knock Airport', 'Title': 'Joe Rogan knocks Biden over Trump's␣
 ↪claim of airports during Revolutionary War',
     'Sentiment Score': -0.5994}
]
data_donegal=  [
    {'Airport': 'Donegal Airport', 'Title': 'Me coming up with a fun idea for a␣
 ↪rail expansion across Ireland', 'Sentiment Score': 0.5106},
    {'Airport': 'Donegal Airport', 'Title': 'Let me rant to yall for a sec␣
 ↪about the Chicago show', 'Sentiment Score': -0.34}
]
data_shannon=[ {'Airport':'Shannon Airport','Title':'Shannon Airport in␣
 ↪Ireland, runway is imaged like a Christmas tree.',
                'Sentiment Score': 0.3612}]
# Create DataFrames for each airport
df_dublin = pd.DataFrame(data_dublin)
df_kerry = pd.DataFrame(data_kerry)
df_knock=pd.DataFrame(data_knock)
df_donegal=pd.DataFrame(data_donegal)
df_shannon=pd.DataFrame(data_shannon)
# Concatenate DataFrames
df_combined = pd.concat([df_dublin, df_kerry, df_knock, df_donegal, df_shannon],
                        ignore_index=True)

# Display the combined DataFrame
df_combined.to_csv('Airports_Sentiment_table.csv',index=False)
```

```
df_combined
```

[157]:

| | Airport | Title | \ |
|---|---|---|---|
| 0 | Dublin Airport | Bad taste in my mouth in Dublin Airport | |
| 1 | Dublin Airport | EU will be ensuring we've an IE rail connectio… | |
| 2 | Kerry Airport | Kerry Airport parking; is it safe? | |
| 3 | Kerry Airport | Help! Central coastal city between Kenmare and… | |
| 4 | Knock Airport | Joe Rogan knocks Biden over Trump's claim of a… | |
| 5 | Donegal Airport | Me coming up with a fun idea for a rail expans… | |
| 6 | Donegal Airport | Let me rant to yall for a sec about the Chicag… | |
| 7 | Shannon Airport | Shannon Airport in Ireland, runway is imaged l… | |

| | Sentiment Score |
|---|---|
| 0 | -0.5423 |
| 1 | 0.2732 |
| 2 | 0.4404 |
| 3 | 0.4574 |
| 4 | -0.5994 |
| 5 | 0.5106 |
| 6 | -0.3400 |
| 7 | 0.3612 |

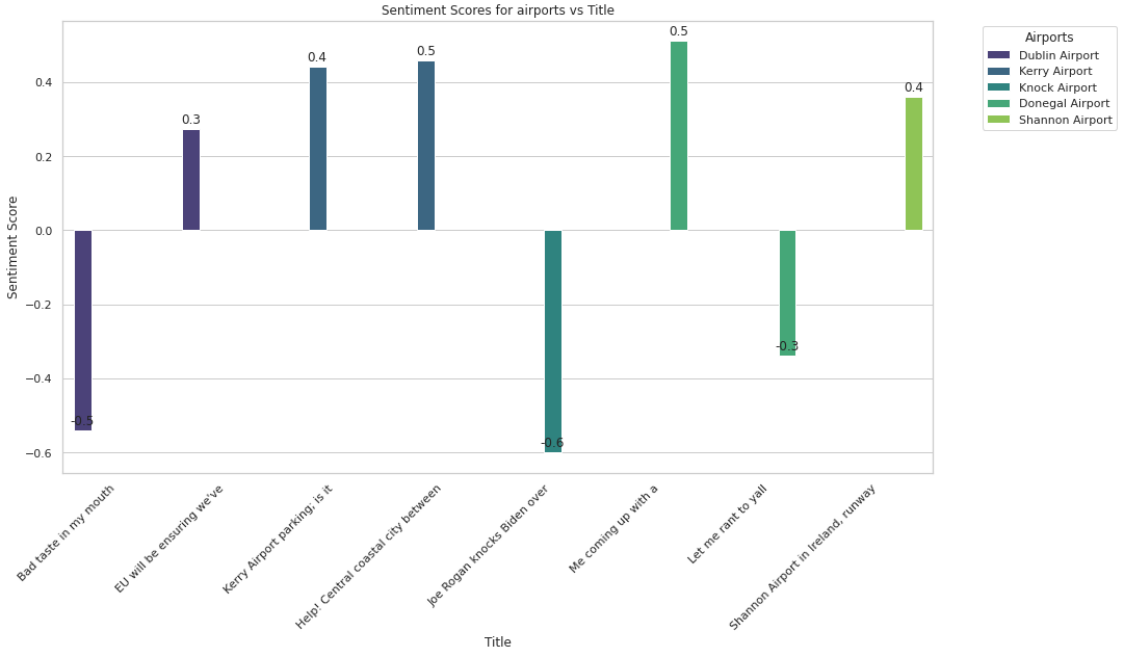
[105]:
```
df_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airport          8 non-null      object
 1   Title            8 non-null      object
 2   Sentiment Score  8 non-null      float64
dtypes: float64(1), object(2)
memory usage: 320.0+ bytes
```

### 2.2 Plot the non-zero sentiment Scores for all the Airports

[159]:
```
# Limit the number of words in the Title column
df=df_combined
df['Title'] = df['Title'].str.split().str[:5].str.join(' ')
# Set the style of seaborn
sns.set(style="whitegrid")
# Bar Plot - Comparison of Ireland with top 5 Countries with Longest Rail Line␣
 ↪Lengths
plt.figure(figsize=(15, 8))
bar_plot = sns.barplot(x='Title', y='Sentiment Score', hue="Airport", data=df,␣
 ↪ci=None, palette='viridis')
```

```
# Annotate each bar with the total rail line length
for p in bar_plot.patches:
    height = p.get_height()
    bar_plot.annotate(f'{height:.1f}',
                      (p.get_x() + p.get_width() / 2., height),
                      ha='center', va='center', xytext=(0, 9),␣
 ↪textcoords='offset points')

plt.title(f"Sentiment Scores for airports vs Title")
plt.xlabel("Title")
plt.ylabel("Sentiment Score")
plt.legend(title="Airports", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
 ↪readability
plt.show()
```




## 3 Get Subjectivity and Polarity of Title

[137]:
```
!pip install textblob
!pip install wordcloud
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: textblob in
/home/44d0387b-deba-4fb3-a3ba-f192c2622970/.local/lib/python3.9/site-packages
```

```
(0.17.1)
Requirement already satisfied: nltk>=3.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
textblob) (3.7)
Requirement already satisfied: click in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
nltk>=3.1->textblob) (8.0.4)
Requirement already satisfied: regex>=2021.8.3 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
nltk>=3.1->textblob) (2022.3.15)
Requirement already satisfied: tqdm in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
nltk>=3.1->textblob) (4.64.0)
Requirement already satisfied: joblib in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
nltk>=3.1->textblob) (1.1.0)
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Collecting wordcloud
  Downloading
wordcloud-1.9.3-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (513
kB)
     |                  | 513 kB 20.0 MB/s eta 0:00:01
Requirement already satisfied: pillow in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
wordcloud) (9.0.1)
Requirement already satisfied: matplotlib in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
wordcloud) (3.5.1)
Requirement already satisfied: numpy>=1.6.1 in
/home/44d0387b-deba-4fb3-a3ba-f192c2622970/.local/lib/python3.9/site-packages
(from wordcloud) (1.26.2)
Requirement already satisfied: cycler>=0.10 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
matplotlib->wordcloud) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
matplotlib->wordcloud) (3.0.4)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
matplotlib->wordcloud) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in
```

```
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from
matplotlib->wordcloud) (4.25.0)
Requirement already satisfied: six>=1.5 in
/opt/conda/envs/anaconda-2022.05-py39/lib/python3.9/site-packages (from python-
dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Installing collected packages: wordcloud
  WARNING: The script wordcloud_cli is installed in
'/home/44d0387b-deba-4fb3-a3ba-f192c2622970/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
warning, use --no-warn-script-location.
Successfully installed wordcloud-1.9.3
```

[139]:
```
from textblob import TextBlob
from wordcloud import WordCloud, STOPWORDS
```

[161]:
```
from textblob import TextBlob

# Create a function to get the subjectivity
def getSubjectivity(text):
    return TextBlob(text).sentiment.subjectivity

# Create a function to get Polarity
def getPolarity(text):
    return TextBlob(text).sentiment.polarity
df_reddit=df_combined
# Now we create a new column for what we just did and add it to the df_reddit␣
 ↪dataframe
df_reddit['Subjectivity'] = df_reddit['Title'].apply(getSubjectivity)
df_reddit['Polarity'] = df_reddit['Title'].apply(getPolarity)

# Now display data
df_reddit.head()
```

[161]:

| | Airport | Title | Sentiment Score | Subjectivity | Polarity |
|---|---|---|---|---|---|
| 0 | Dublin Airport | Bad taste in my mouth | -0.5423 | 0.666667 | -0.7 |
| 1 | Dublin Airport | EU will be ensuring we've | 0.2732 | 0.000000 | 0.0 |
| 2 | Kerry Airport | Kerry Airport parking; is it | 0.4404 | 0.000000 | 0.0 |
| 3 | Kerry Airport | Help! Central coastal city between | 0.4574 | 0.250000 | 0.0 |
| 4 | Knock Airport | Joe Rogan knocks Biden over | -0.5994 | 0.000000 | 0.0 |

Group Polarity

[163]:
```
# Group the range of Polarity into different categories
def getInsight(score):
    if score < 0:
        return "Negative"
    elif score == 0:
        return "Neutral"
    else:
        return "Positive"

df_reddit["Insight"] = df_reddit["Sentiment Score"].apply(getInsight)
df_reddit.head(50)
```

[163]:

| | Airport | Title | Sentiment Score | Subjectivity | Polarity | Insight |
|---|---|---|---|---|---|---|
| 0 | Dublin Airport | Bad taste in my mouth | -0.5423 | 0.666667 | -0.7 | Negative |
| 1 | Dublin Airport | EU will be ensuring we've | 0.2732 | 0.000000 | 0.0 | Positive |
| 2 | Kerry Airport | Kerry Airport parking; is it | 0.4404 | 0.000000 | 0.0 | Positive |
| 3 | Kerry Airport | Help! Central coastal city between | 0.4574 | 0.250000 | 0.0 | Positive |
| 4 | Knock Airport | Joe Rogan knocks Biden over | -0.5994 | 0.000000 | 0.0 | Negative |
| 5 | Donegal Airport | Me coming up with a | 0.5106 | 0.000000 | 0.0 | Positive |
| 6 | Donegal Airport | Let me rant to yall | -0.3400 | 0.000000 | 0.0 | Negative |
| 7 | Shannon Airport | Shannon Airport in Ireland, runway | 0.3612 | 0.000000 | 0.0 | Positive |

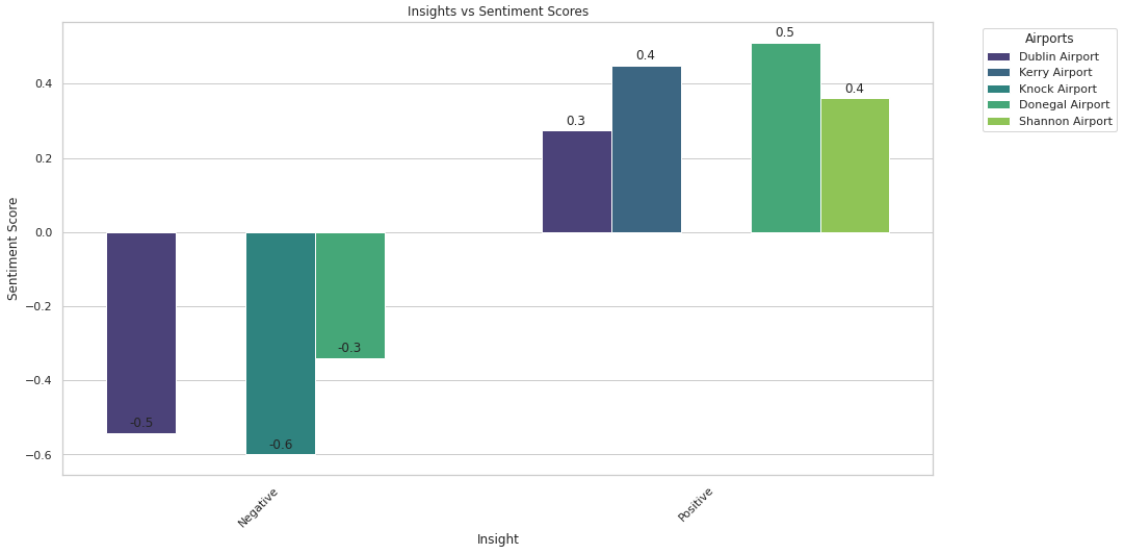
### 3.0.1 Data Visualization of Insights vs Sentiment Score

[102]:
```
plt.style.use('fivethirtyeight')
```

**Plot the Sentiment Scores vs Insights**

[131]:
```
#Plot the va# Bar Plot - Comparison of Ireland with top 5 Countries with␣
 ↪Longest Rail Line Lengths
plt.figure(figsize=(15, 8))
bar_plot = sns.barplot(x='Insight', y='Sentiment Score', hue="Airport",␣
 ↪data=df_reddit, ci=None, palette='viridis')
# Annotate each bar with the total rail line length
for p in bar_plot.patches:
```

```
    height = p.get_height()
    bar_plot.annotate(f'{height:.1f}',
                      (p.get_x() + p.get_width() / 2., height),
                      ha='center', va='center', xytext=(0, 9),␣
 ↪textcoords='offset points')

plt.title(f"Insights vs Sentiment Scores")
plt.xlabel("Insight")
plt.ylabel("Sentiment Score")
plt.legend(title="Airports", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
 ↪readability
plt.show()
```




[ ]: