Jiesen Zhang, Jieru Lin, Yu Xu

A16853508, A16635527, A16247845

COGS 109

Professor Eran Mukamel

Final Project Report

Correlation and Predictive Power of Provincial Unemployment Rates on National Unemployment in Canada

Introduction

Recently, we have witnessed significant economic challenges worldwide that are reshaping the trajectories of both developed and developing nations. The pandemic, in particular, has exacerbated pre-existing inequalities in economies worldwide, triggering an unprecedented global recession and leaving a lasting impact on the global economy. Unemployment is a significant economic indicator that reflects the overall health and functioning of an economy. Particularly in a country like Canada, with its diverse industries and demographics, the unemployment rate serves as a valuable tool for understanding economic health and policy effectiveness. By examining the period from 1976 to the present, we expect to gain some valuable insights into historical patterns and assess the relationship between unemployment rates and related factors over time.

The data in this project provides information on unemployment numbers and percentages from 1976 to the present, and it comes from Labor Force Statistics in Canada, which is a national statistical office that reports monthly about Canada's economy. The data consists of Canada and its provinces, with both sexes and different age groups. There are a total of 38985 observations and 13 predictors: the reference period (by year, month), the geographic area, the sex being investigated, the age group of the economic measure, employment, full-time employment, labor force, part-time employment, population, unemployment, employment rate, participation rate, unemployment rate. After we import the data, we wrangle the data for exploratory data analysis to find potential hypotheses we could test.

Explorative Data Analysis

Data Cleaning

In [5]:
```
# Get all the columns of the df
columns = unemployment_df.columns

# Find out how many unique values each column has
for column in columns:
    unique_val = unemployment_df[column].nunique()
    print('The ' + column + ' column has ' + str(unique_val) + ' values')
```

```
The REF_DATE column has 565 values
The GEO column has 11 values
The Sex column has 1 values
The Age group column has 9 values
The Employment column has 18716 values
The Full-time employment column has 16223 values
The Labour force column has 19173 values
The Part-time employment  column has 9151 values
The Population column has 22431 values
The Unemployment column has 6565 values
The Employment rate column has 758 values
The Participation rate column has 770 values
The Unemployment rate column has 328 values
```

In [6]:
```
# Check the unique value that the `Sex` column has
unemployment_df['Sex'].iloc[0]
```

Out[6]:
```
'Both sexes'
```

As shown above, the Sex column has only one unique value, which is 'Both sexes'. Because it is the same for all rows, it becomes useless for both doing inferences and doing predictions. Therefore we could safely remove it.

In [7]:
```
# Drop the `Sex` column because it won't be useful for either inference or prediction
unemployment_df = unemployment_df.drop('Sex', axis = 1)
```

In [8]:
```
# Check the number of null values in each column
unemployment_df.isnull().sum(axis = 0)
```

Out[8]:
```
REF_DATE                   0
GEO                        0
Age group                  0
Employment                 0
Full-time employment    1695
Labour force               0
Part-time employment    1696
Population                 0
Unemployment               6
Employment rate            0
Participation rate         0
Unemployment rate          6
dtype: int64
```

We won't be using the Full-time employment and Part-time employment columns because it highly relates to the employment column, which may cause multicollinearity.

In [36]:
```
unemployment_df.drop(unemployment_df.columns[[4, 6]], axis=1, inplace=True)
```

In [36]:
```
unemployment_df.drop(unemployment_df.columns[[4, 6]], axis=1, inplace=True)
```

In [37]:
```
# Make a copy of the original data frame
unemployment_filled_df = unemployment_df.copy()

# Fill the unemployment rate using the mean of the column because it is missing at random(MAR)
ur_mar_filled = unemployment_df['Unemployment rate'].fillna(unemployment_df['Unemployment rate'].mean())
unemployment_filled_df['Unemployment rate'] = ur_mar_filled

# Derive the value to fill the `Unemployment` column using the `Unemployment rate` column
unemployment_mar_filled = unemployment_filled_df['Unemployment']\
                    .fillna(unemployment_filled_df['Unemployment rate']/100 * unemployment_filled_df['Labour force'])
unemployment_filled_df['Unemployment'] = unemployment_mar_filled
```

In [38]:
```
# The dataframe before filling the null values in `Unemployment` and `Unemployment rate`
unemployment_df[unemployment_df['Unemployment'].isnull()]
```

Out[38]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Employment rate	Participation rate	Unemployment rate
206	1976-03	Saskatchewan	55 years and over	57800.0	58100.0	180800.0	NaN	32.0	32.1	NaN
344	1976-05	Saskatchewan	55 years and over	61600.0	62000.0	181600.0	NaN	33.9	34.1	NaN
695	1976-11	Alberta	55 years and over	94300.0	95000.0	262400.0	NaN	35.9	36.2	NaN
3317	1980-01	Alberta	55 years and over	99100.0	100500.0	289600.0	NaN	34.2	34.7	NaN
3920	1980-09	Prince Edward Island	55 years and over	6100.0	6300.0	24200.0	NaN	25.2	26.0	NaN
3989	1980-10	Prince Edward Island	55 years and over	6100.0	6200.0	24200.0	NaN	25.2	25.6	NaN

In [39]:
```
# The dataframe after filling the null values in `Unemployment` and `Unemployment rate`
unemployment_filled_df[unemployment_df['Unemployment'].isnull()]
```

Out[39]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Employment rate	Participation rate	Unemployment rate
206	1976-03	Saskatchewan	55 years and over	57800.0	58100.0	180800.0	5503.308197	32.0	32.1	9.472131
344	1976-05	Saskatchewan	55 years and over	61600.0	62000.0	181600.0	5872.721311	33.9	34.1	9.472131
695	1976-11	Alberta	55 years and over	94300.0	95000.0	262400.0	8998.524590	35.9	36.2	9.472131
3317	1980-01	Alberta	55 years and over	99100.0	100500.0	289600.0	9519.491803	34.2	34.7	9.472131
3920	1980-09	Prince Edward Island	55 years and over	6100.0	6300.0	24200.0	596.744262	25.2	26.0	9.472131
3989	1980-10	Prince Edward Island	55 years and over	6100.0	6200.0	24200.0	587.272131	25.2	25.6	9.472131

For this project, we will only be looking at the population of age between 15 to 64 because only a very small percentage of people work over the age of 64. If we don't exclude them, it's going to unnaccessarily boost up unemployment rate.

In [40]:
```
unemployment_filtered_df = unemployment_filled_df[unemployment_filled_df['Age group'] == '15 to 64 years']
unemployment_filtered_df.head()
```

Out[40]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Employment rate	Participation rate	Unemployment rate
1	1976-01	Alberta	15 to 64 years	802400.0	837500.0	1154800.0	35000.0	69.5	72.5	4.2
7	1976-01	British Columbia	15 to 64 years	1015500.0	1108500.0	1628800.0	93000.0	62.3	68.1	8.4
14	1976-01	Canada	15 to 64 years	9465600.0	10185000.0	15015900.0	719400.0	63.0	67.8	7.1
22	1976-01	Manitoba	15 to 64 years	418600.0	442500.0	635700.0	23900.0	65.8	69.6	5.4
28	1976-01	New Brunswick	15 to 64 years	227100.0	255700.0	419800.0	28700.0	54.1	60.9	11.2

During the data wrangling phase, we initially identified that the predictor "both sexes" was not relevant since gender differences would not impact the unemployment rate in this dataset. There’s only one unique value for this column, so it won’t be useful. Subsequently, we narrowed our focus to the age group spanning from 15 to 64 years old, as this age group constitutes the primary labor force in society. To avoid including data from other age groups and prevent overcounting, we retained only the rows with row values of 15 to 64 years old. We also removed the Full-time employment and Part-time employment columns because it highly relates to the employment column, which may cause multicollinearity.

Out of curiosity and for the purpose of exploring the dataset, we conducted a comparison of all Canadian provinces, with a particular emphasis on Ontario, which is the largest province with a robust economy. From the graph, we see that the trend of Ontario's and Canada’s unemployment is consistent, we posited that Ontario might exhibit a close correlation with the national unemployment rate. To put our hypothesis to the test, we made the decision to calculate and compare the mean unemployment rate for each province as follows.

Hypothesis Testing

```
unemployment_filtered_df['Month'] = unemployment_filtered_df['REF_DATE'].str[-2:]
unemployment_filtered_df.groupby('GEO').mean()['Unemployment rate']
```

```
/tmp/ipykernel_107/1957449123.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  unemployment_filtered_df['Month'] = unemployment_filtered_df['REF_DATE'].str[-2:]
/tmp/ipykernel_107/1957449123.py:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is depr
ecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only columns w
hich should be valid for the function.
  unemployment_filtered_df.groupby('GEO').mean()['Unemployment rate']
```

```
GEO
Alberta                       6.622124
British Columbia              8.248673
Canada                        8.215929
Manitoba                      6.426726
New Brunswick                11.078938
Newfoundland and Labrador    15.780885
Nova Scotia                  10.260531
Ontario                       7.512389
Prince Edward Island         12.166726
Quebec                        9.485664
Saskatchewan                  6.047611
Name: Unemployment rate, dtype: float64
```

```
unemployment_filtered_df['REF_DATE'].str[-2:]
```

```
1        01
7        01
14       01
22       01
28       01
         ..
38956    01
38962    01
38968    01
38974    01
38980    01
Name: REF_DATE, Length: 6215, dtype: object
```

However, after we calculated the mean unemployment rate for all provinces in Canada, we found that Saskatchewan has the lowest unemployment rate and decided to change our target province from Ontario to Saskatchewan, as we are intrigued by the potential connection between the nation's overall unemployment rate and this region's exceptionally low rate.

Null Hypothesis: Unemployment rate and provinces are unrelated. The low average unemployment rate of the province of Saskatchewan is due to chance alone.

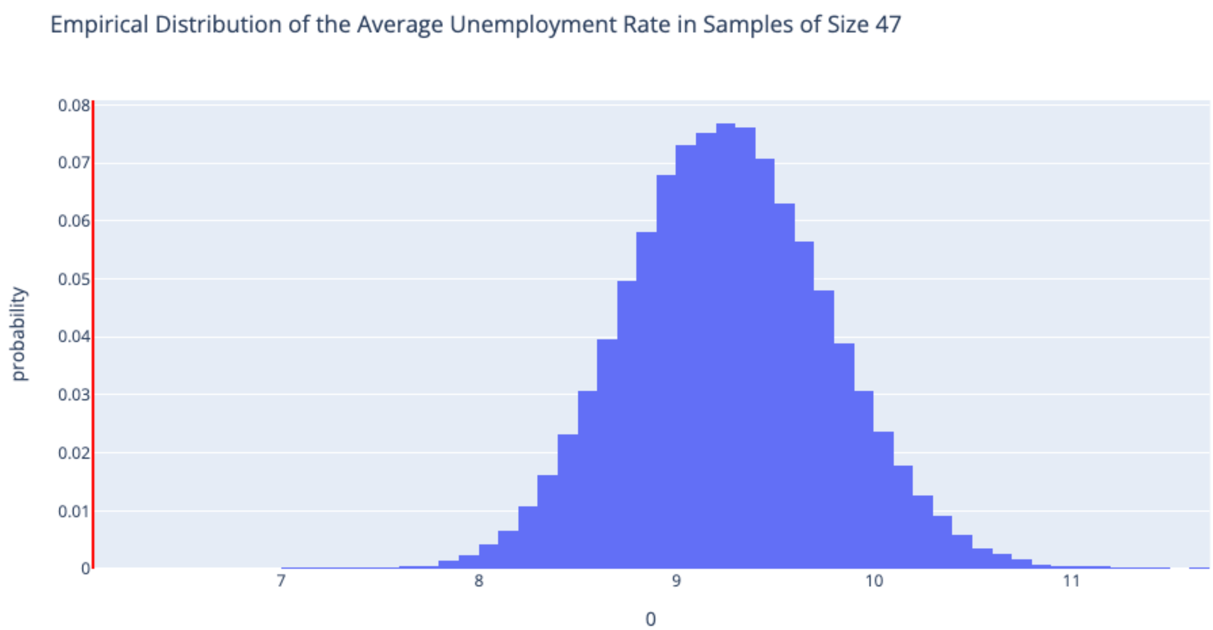
Alternative hypothesis: Unemployment rate and provinces are related. The low average unemployment rate of the province of Saskatchewan is not due to chance alone.

```
# Draws 100000 samples of size 47 from penguins['bill_length_mm'].
num_reps = 100_000
averages = np.random.choice(unemployment_filtered_df['Unemployment rate'], size=(num_reps, 47)).mean(axis=1)
averages
```

```
array([10.1106383 ,  8.45319149,  8.96170213, ...,  8.84255319,
        9.49787234,  9.15531915])
```

Visualizing the empirical distribution of the test statistic

```
fig = px.histogram(pd.DataFrame(averages), x=0, nbins=50, histnorm='probability',
                   title='Empirical Distribution of the Average Unemployment Rate in Samples of Size 47')
fig.add_vline(x=unemployment_filtered_df.loc[unemployment_filtered_df['GEO'] == 'Saskatchewan', 'Unemployment rate'].me
```



It doesn't look like the average unemployment rate of the province of Saskatchewan was due to chance alone. Therefore we reject the null. In the future, we will take a closer look at the province of Saskatchewan to investigate what has led to the low unemployment rate.

Methods-cross validation:

There are in total 13 columns in the dataset provided, in which 5 of them we believe we could use to make a model (Employment, Labour force, Population, Employment rate, Participation rate).

For this project, we plan to do predictive modeling that predicts the future unemployment rate of Canada using historical data. By doing this, we hope to give people an idea of what's to come so they could make better decisions for their future. The two models that we will try are linear regression and KNeighborsRegressor. We chose these two models because the type of prediction we are trying to make is a numerical value.

First of all, we try to use linear regression to predict the unemployment rate of Canada in the future based on the numerical parameters we have in the dataset (Employment, Labour force, Population, Employment rate, and Participation rate). Our goal is to find a linear model with just the best predictive parameters that can best predict the future unemployment rate in Canada. We tested out our practicability of the dataset on a linear model by creating a linear regression model using all the numerical parameters to predict unemployment rate. We performed a 10-fold cross-validation on our model: we split the data into training and testing sets, fit our model in the training data, make predictions with the model we have on the testing data and calculate the RMSE for each fold. We calculate the average of RMSEs in the end. After testing out our practicability of using linear regression modeling, we try to find out the best predictive model that has only the best predictors using 10-fold cross validation on all the possible models in order of forward stepwise selection. After we find the best model, we list out its predictors and calculate its RMSE.

Methods:

In this project, the goal is to predict the future unemployment rate of Canada using historical data. The dataset contains 13 columns, but only 5 columns (Employment, Labour force, Population, Employment rate, and Participation rate) are believed to be relevant for building the model. The choice of linear regression and KNeighborsRegressor models is appropriate because the prediction task involves numerical values.

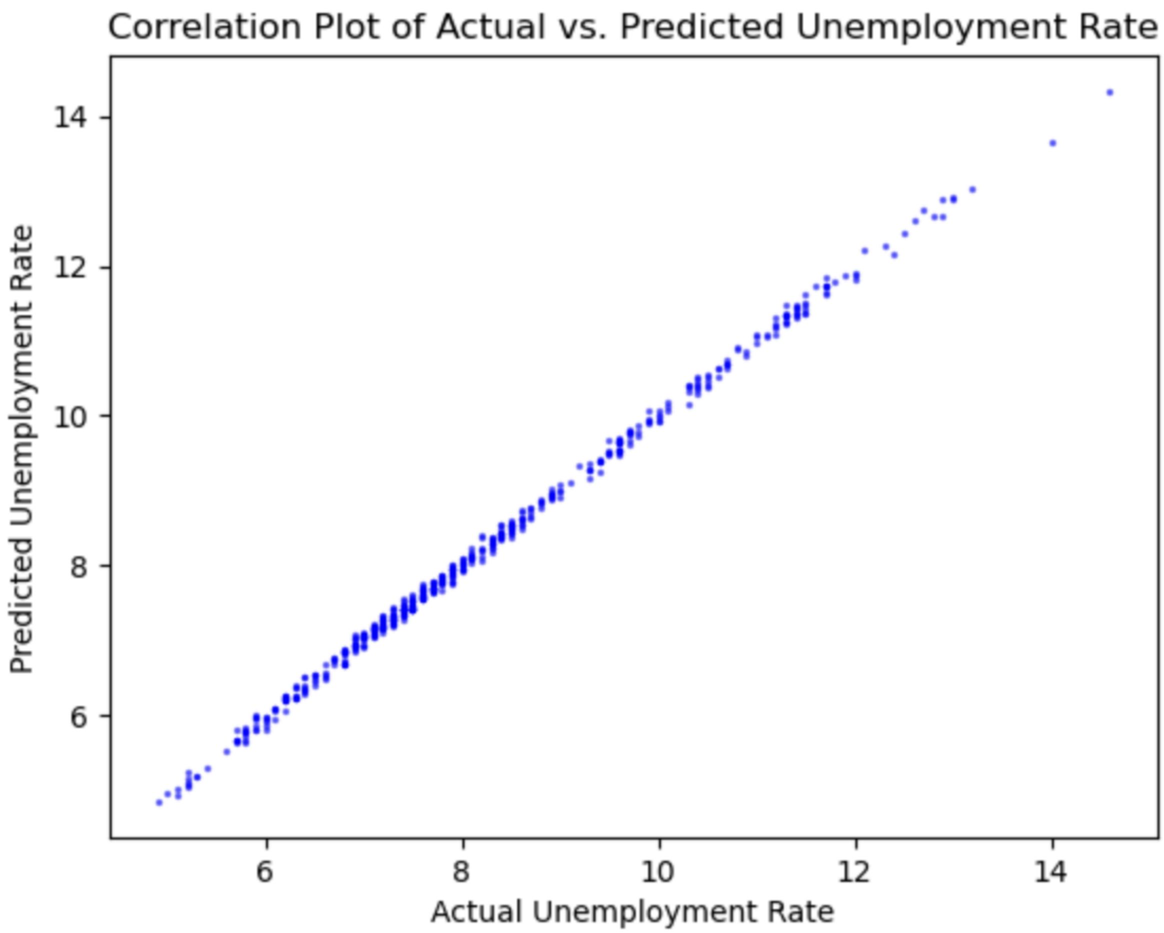
Linear regression is a valid and commonly used method for predictive modeling. It assumes a linear relationship between the dependent variable (unemployment rate in this case) and the independent variables (Employment, Labour force, Population, Employment rate, and Participation rate). The process of linear regression involves fitting a linear equation to the data by minimizing the sum of squared differences between the observed and predicted values.

To implement linear regression, the process begins by creating a linear regression model using all the numerical parameters as predictors for predicting the unemployment rate. To assess the model's performance, a 10-fold cross-validation is performed. This involves splitting the data into 10 subsets, training the model on 9 subsets, and evaluating its performance on the remaining subset. The root mean squared error (RMSE) is calculated for each fold, and the average RMSE is computed at the end. The RMSE is a measure of the model's accuracy in predicting the unemployment rate.

Results - model selection

After evaluating the overall practicability of using linear regression, the next step is to identify the best predictors for the model. This is done through forward stepwise selection, where different models are evaluated by adding one predictor at a time and selecting the one that improves the model's performance the most. The process involves performing 10-fold cross-validation on all possible models in order of forward stepwise selection. Once the best model is identified, its predictors are listed, and the RMSE is calculated to assess its predictive accuracy.

The predictors for the best linear regression model turns out to be 'Employment', 'Labour force', 'Population', 'Employment rate', 'Participation rate'. Despite being a less flexible model, its corresponding RMSE is around 0.07.



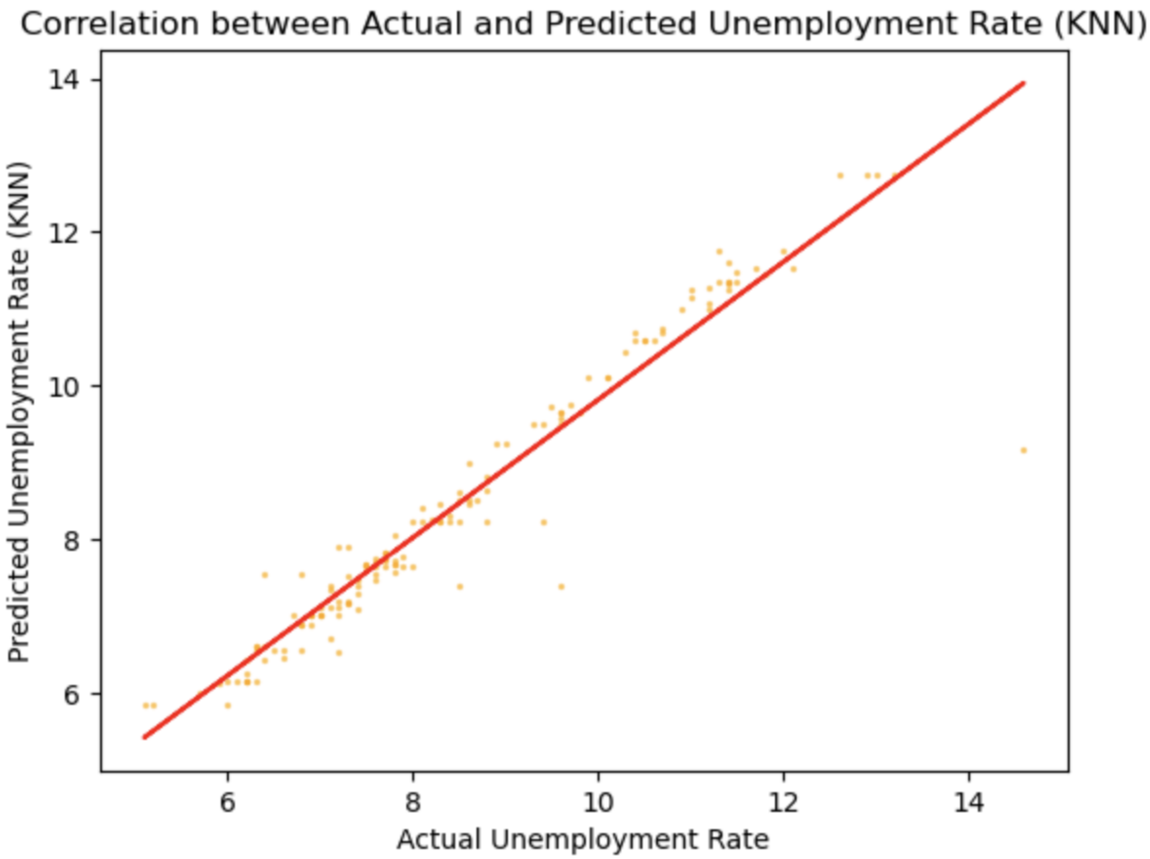
The KNeighborsRegressor method is also a valid approach for predictive modeling, specifically in regression tasks where the objective is to predict a continuous numerical value. It operates by identifying the k nearest neighbors from the training data for each data point in the testing data, and then predicting the value based on the average or weighted average of the neighbors' target values. When compared to linear regression, it is more flexible

To apply the KNeighborsRegressor model, the dataset is divided into feature variables (predictor variables) and the response variable (the variable to be predicted). The irrelevant columns are removed from the dataset to obtain the predictor variables, while the response variable is determined as the 'Unemployment rate' column.

Next, the KNeighborsRegressor model is defined, specifying the necessary numerical parameters for the algorithm. To evaluate the performance of the model, a process of 10-fold cross-validation is employed in a similar way we did when evaluating the linear regression model: During each fold of the cross-validation process, the model is fitted to the training data and then used to make predictions on the testing data. The predictions are compared to the root mean squared error (RMSE), which measures the average difference between the predicted and actual values, providing an indication of the model's accuracy. Additionally, the average RMSE is computed to obtain an overall estimate of the model's predictive accuracy.

To further optimize the model, we applied the forward stepwise selection on the KNeighborsRegressor model. This involves iteratively evaluating different combinations of predictor variables by performing cross-validation. The average RMSE is calculated for each combination, and the model with the lowest RMSE is selected as the best model.

Using the KNeighborsRegressor method, the predictors for the best linear regression model turn out to be Predictors: 'Population', 'Unemployment', 'Employment rate', 'Participation rate'. And its corresponding RMSE is around 0.19.

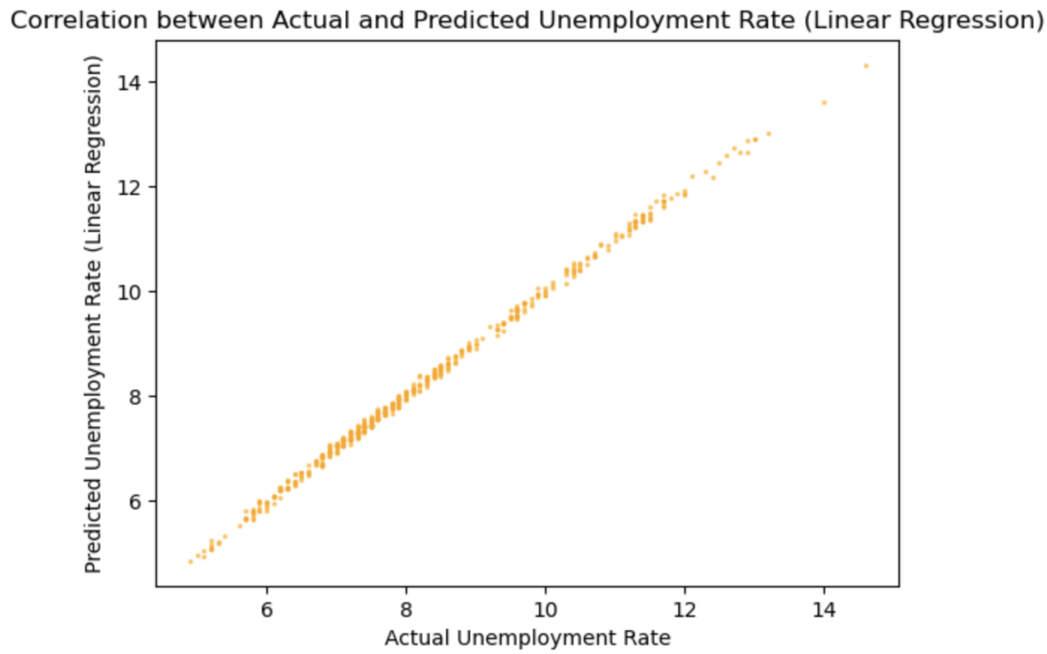


Due to the difference in performance between the two models, we decided to use linear regression. Besides the difference in performance, linear regression is also a better option because it is a less complicated model when compared to KNNRegressor. However, the performance of both models seems to be too high, we suspect that there's overfitting to the data but we are unsure. To determine whether or not there's over-fitting in either model, we will need future investigation.

Results - model estimation

In the end, we fitted the model to the whole dataset and obtained the final parameter estimates. The coefficients of each predictor are 7.26220404e-08, -7.63438245e-07, -1.48126979e+00, and 1.35508167e+00, and the Intercept is 8.418970167545531.

The final R-squared is 0.9982564362806949 and final RMSE is 0.07312501694396009.



Discussion

Based on our final model performance, it seems like the unemployment rate of Canada as a whole is very predictable. However, it is also possible that our model is overfitting. To find out whether or not the result we were able to achieve in the end is due to chance, overfitting, or valid, a deeper study needs to be done. Assuming that our final model is valid, future researchers could conduct further research on studying why the unemployment rate shows the trend that it has. In addition, people could also use it to better prepare for future economic recessions and minimize the damage to individuals and the economy as a whole.

In [1]:
```
import pandas as pd
import numpy as np
import os

from scipy.stats import ks_2samp
import ast
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
pd.options.plotting.backend = 'plotly'

from statsmodels.formula.api import logit
from itertools import combinations

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Initiate the DataFrames

In [2]:
```
# Create path to csv file
interaction_path= os.path.join('data', 'Unemployment_Canada.csv')
```

In [3]:
```
# Import the CSV files as DataFrames
unemployment_df = pd.read_csv(interaction_path)
```

In [4]:
```
unemployment_df.head() # First five rows of the Canadian unemployment DataFrame
```

Out[4]:

	REF_DATE	GEO	Sex	Age group	Employment	Full-time employment	Labour force	Part-time employment	Populati
0	1976-01	Alberta	Both sexes	15 to 24 years	231800.0	174900.0	252300.0	56900.0	36230
1	1976-01	Alberta	Both sexes	15 to 64 years	802400.0	682100.0	837500.0	120300.0	115480
2	1976-01	Alberta	Both sexes	15 years and over	819500.0	693700.0	856500.0	125800.0	127670
3	1976-01	Alberta	Both sexes	25 to 54 years	491400.0	439800.0	505800.0	51600.0	66170
4	1976-01	Alberta	Both sexes	25 years and over	587700.0	518800.0	604200.0	68900.0	91440

Explorative Data Analysis

Data Cleaning

In [5]:
```
# Get all the columns of the df
columns = unemployment_df.columns

# Find out how many unique values each column has
for column in columns:
    unique_val = unemployment_df[column].nunique()
    print('The ' + column + ' column has ' + str(unique_val) + ' values')
```

```
The REF_DATE column has 565 values
The GEO column has 11 values
The Sex column has 1 values
The Age group column has 9 values
The Employment column has 18716 values
The Full-time employment column has 16223 values
The Labour force column has 19173 values
The Part-time employment  column has 9151 values
The Population column has 22431 values
The Unemployment column has 6565 values
The Employment rate column has 758 values
The Participation rate column has 770 values
The Unemployment rate column has 328 values
```

In [6]:
```
# Check the unique value that the `Sex` column has
unemployment_df['Sex'].iloc[0]
```

Out[6]:
```
'Both sexes'
```

As shown above, the Sex column has only one unique value, which is 'Both sexes'. Because it is the same for all rows, it becomes useless for both doing inferences and doing predictions. Therefore we could safely remove it.

In [7]:
```
# Drop the `Sex` column because it won't be useful for either inference or pred
unemployment_df = unemployment_df.drop('Sex', axis = 1)
```

In [8]:
```
# Check the number of null values in each column
unemployment_df.isnull().sum(axis = 0)
```

Out[8]:
```
REF_DATE                    0
GEO                         0
Age group                   0
Employment                  0
Full-time employment     1695
Labour force                0
Part-time employment     1696
Population                  0
Unemployment                6
Employment rate             0
Participation rate          0
Unemployment rate           6
dtype: int64
```

We won't be using the Full-time employment and Part-time employment columns because it highly relates to the employment column, which may cause multicollinearity.

In [9]:
```
unemployment_df.drop(unemployment_df.columns[[4, 6]], axis=1, inplace=True)
```

In [10]:
```
# Make a copy of the original data frame
unemployment_filled_df = unemployment_df.copy()

# Fill the unemployment rate using the mean of the column because it is missing
ur_mar_filled = unemployment_df['Unemployment rate'].fillna(unemployment_df['Ur
unemployment_filled_df['Unemployment rate'] = ur_mar_filled

# Derive the value to fill the `Unemployment` column using the `Unemployment ra
unemployment_mar_filled = unemployment_filled_df['Unemployment']\
                    .fillna(unemployment_filled_df['Unemployment rate']/100 * u
unemployment_filled_df['Unemployment'] = unemployment_mar_filled
```

In [11]:
```
# The dataframe before filling the null values in `Unemployment` and `Unemployn
unemployment_df[unemployment_df['Unemployment'].isnull()]
```

Out[11]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Emp
206	1976-03	Saskatchewan	55 years and over	57800.0	58100.0	180800.0	NaN	
344	1976-05	Saskatchewan	55 years and over	61600.0	62000.0	181600.0	NaN	
695	1976-11	Alberta	55 years and over	94300.0	95000.0	262400.0	NaN	
3317	1980-01	Alberta	55 years and over	99100.0	100500.0	289600.0	NaN	
3920	1980-09	Prince Edward Island	55 years and over	6100.0	6300.0	24200.0	NaN	
3989	1980-10	Prince Edward Island	55 years and over	6100.0	6200.0	24200.0	NaN	

In [12]:
```
# The dataframe after filling the null values in `Unemployment` and `Unemployme
unemployment_filled_df[unemployment_df['Unemployment'].isnull()]
```

Out[12]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Emp
206	1976-03	Saskatchewan	55 years and over	57800.0	58100.0	180800.0	5503.308197	
344	1976-05	Saskatchewan	55 years and over	61600.0	62000.0	181600.0	5872.721311	
695	1976-11	Alberta	55 years and over	94300.0	95000.0	262400.0	8998.524590	
3317	1980-01	Alberta	55 years and over	99100.0	100500.0	289600.0	9519.491803	
3920	1980-09	Prince Edward Island	55 years and over	6100.0	6300.0	24200.0	596.744262	
3989	1980-10	Prince Edward Island	55 years and over	6100.0	6200.0	24200.0	587.272131	

For this project, we will only be looking at the population of age between 15 to 64 because only a very small percentage of people work over the age of 64. If we don't exclude them, it's going to unnaccessarily boost up unemployment rate.

In [13]:

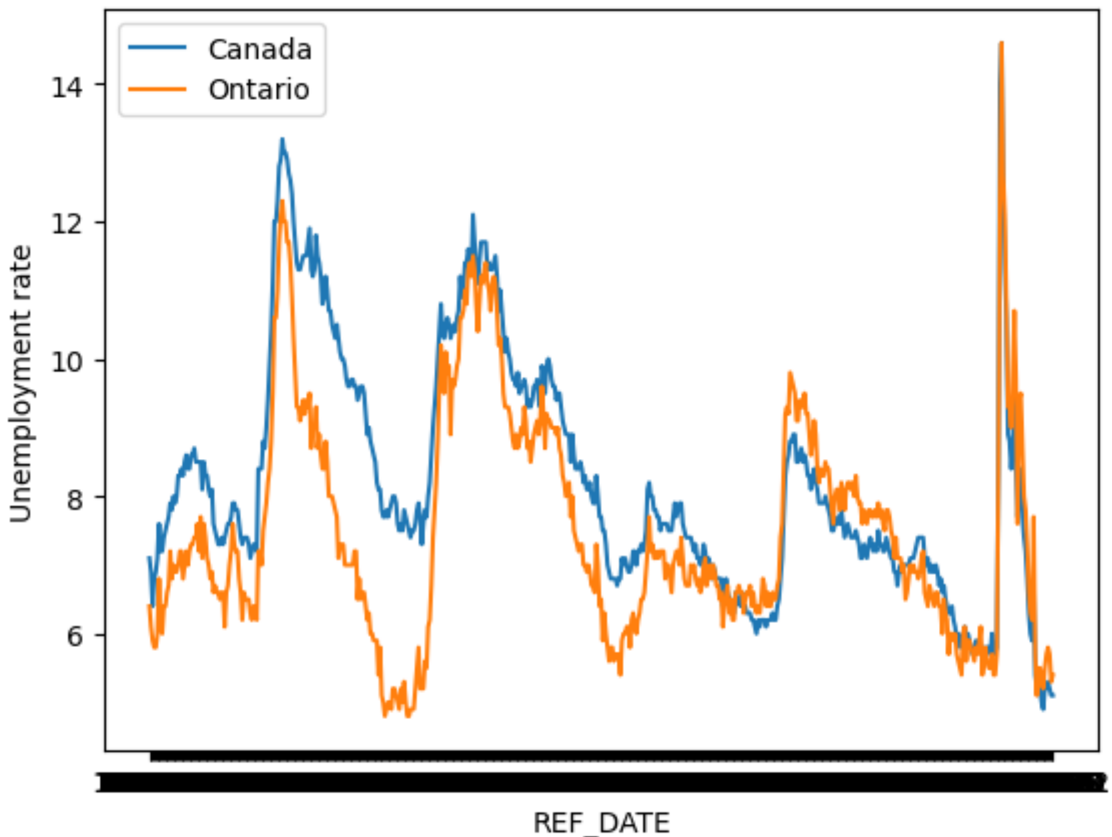
```
unemployment_filtered_df = unemployment_filled_df[unemployment_filled_df['Age g
unemployment_filtered_df.head()
```

Out[13]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Employ
1	1976-01	Alberta	15 to 64 years	802400.0	837500.0	1154800.0	35000.0	
7	1976-01	British Columbia	15 to 64 years	1015500.0	1108500.0	1628800.0	93000.0	
14	1976-01	Canada	15 to 64 years	9465600.0	10185000.0	15015900.0	719400.0	
22	1976-01	Manitoba	15 to 64 years	418600.0	442500.0	635700.0	23900.0	
28	1976-01	New Brunswick	15 to 64 years	227100.0	255700.0	419800.0	28700.0	

In [14]:
```
canada_data = unemployment_filtered_df[unemployment_filtered_df['GEO'] == 'Cana
ontario_data = unemployment_filtered_df[unemployment_filtered_df['GEO'] == 'Ont

plt.plot(canada_data['REF_DATE'], canada_data['Unemployment rate'], label='Cana
plt.plot(ontario_data['REF_DATE'], ontario_data['Unemployment rate'], label='Or
plt.xlabel('REF_DATE')
plt.ylabel('Unemployment rate')
plt.legend()
```

Out[14]:
```
<matplotlib.legend.Legend at 0x7fbf5834aa90>
```



Hypothesis Testing

In [15]:
```
unemployment_filtered_df['Month'] = unemployment_filtered_df['REF_DATE'].str[-2
unemployment_filtered_df.groupby('GEO').mean()['Unemployment rate']
```

```
/var/folders/qb/x2k0b7_x6j928j0nc2w6x7nc0000gn/T/ipykernel_68561/1957449123.p
y:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  unemployment_filtered_df['Month'] = unemployment_filtered_df['REF_DATE'].str
[-2:]
```

Out[15]:

```
GEO
Alberta                       6.622124
British Columbia              8.248673
Canada                        8.215929
Manitoba                      6.426726
New Brunswick                11.078938
Newfoundland and Labrador    15.780885
Nova Scotia                  10.260531
Ontario                       7.512389
Prince Edward Island         12.166726
Quebec                        9.485664
Saskatchewan                  6.047611
Name: Unemployment rate, dtype: float64
```

Null Hypothesis: Unemployment rate and provinces are unrelated. The low average unemployment rate of the province of Saskatchewan is due to chance alone.

Alternative hypothesis: Unemployment rate and provinces are related. The low average unemployment rate of the province of Saskatchewan is not due to chance alone.

In [16]:

```
# Draws 100000 samples of size 47 from penguins['bill_length_mm'].
num_reps = 100_000
averages = np.random.choice(unemployment_filtered_df['Unemployment rate'], size
averages
```

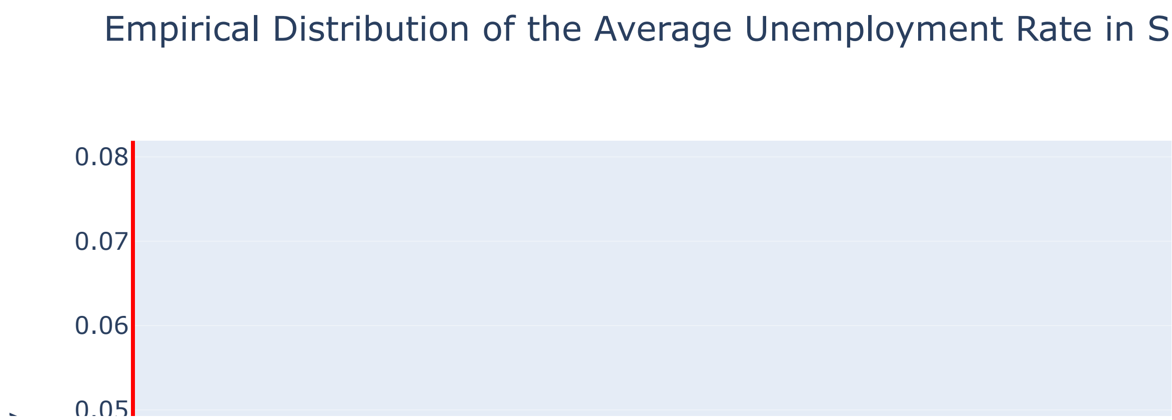
Out[16]:

```
array([8.52978723, 9.29787234, 9.91489362, ..., 8.95319149, 9.1893617 ,
       9.6106383 ])
```

Visualizing the empirical distribution of the test statistic

In [17]:

```
fig = px.histogram(pd.DataFrame(averages), x=0, nbins=50, histnorm='probability
                   title='Empirical Distribution of the Average Unemployment Ra
fig.add_vline(x=unemployment_filtered_df.loc[unemployment_filtered_df['GEO'] ==
```



It doesn't look like the average unemployment rate of the province of Saskatchewan was due to chance alone. Therefore we reject the null. In the future, we will take a closer look at the province of Saskatchewan to investigate what has led to the low unemployment rate.

Model Selection

We would like to use cross validation to perform forward step-wise selection to select the best predictors that we can use in our model to predict the future unemployment rate in Canada based on current data we have.

In [18]:

```
unemployment_filtered_df = unemployment_filtered_df[unemployment_filtered_df['C
unemployment_filtered_df.head()
```

Out[18]:

	REF_DATE	GEO	Age group	Employment	Labour force	Population	Unemployment	Employn
14	1976-01	Canada	15 to 64 years	9465600.0	10185000.0	15015900.0	719400.0	
83	1976-02	Canada	15 to 64 years	9492200.0	10199300.0	15049000.0	707100.0	
152	1976-03	Canada	15 to 64 years	9533200.0	10185500.0	15081200.0	652300.0	
221	1976-04	Canada	15 to 64 years	9572600.0	10268300.0	15113400.0	695700.0	
290	1976-05	Canada	15 to 64 years	9567900.0	10276800.0	15145500.0	708900.0	

First of all, we try to fit a linear regression model with the parameters: Employment, Labour force, Population, Unemployment, Employment rate, Participation rate, and Unemployment rate. And we run a 10-fold cross validation on this model and store its Average RMSE

In [19]:

```
# Define a function that calculates rmse
def rmse(actual, pred):
    return np.sqrt(np.mean((actual - pred) ** 2))
```

In [20]:

```
# Separate the data into feature variables and response variables (x and y)
x = unemployment_filtered_df.drop(['Unemployment rate', 'Unemployment', 'REF_DA
y = unemployment_filtered_df['Unemployment rate']

# Define the model (Linear Regression)
lr_mdl = LinearRegression()

# Number of folds for cross-validation
num_folds = 10

# Initialize an array to store RMSE values for each fold
rmse_scores = []

# Perform k-fold cross-validation
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
for train_index, test_index in kf.split(x):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

    # Fit the model on the training data
    lr_mdl.fit(x_train, y_train)

    # Make predictions on the test data
    y_pred = lr_mdl.predict(x_test)

    # Calculate RMSE for the fold
    cur_rmse = rmse(y_test, y_pred)
    rmse_scores.append(cur_rmse)
```

```
# Print the RMSE for each fold
for fold, it_rmse in enumerate(rmse_scores, start=1):
    print('RMSE for Fold {}: {}'.format(fold, it_rmse))

# Calculate the average RMSE across all folds
average_rmse = np.mean(rmse_scores)
print('Average RMSE: {}'.format(average_rmse))
```

```
RMSE for Fold 1: 0.07596874895431434
RMSE for Fold 2: 0.07552973467871629
RMSE for Fold 3: 0.08311210409287371
RMSE for Fold 4: 0.06685886056858657
RMSE for Fold 5: 0.07063131713998598
RMSE for Fold 6: 0.08308029252072935
RMSE for Fold 7: 0.07703140004914899
RMSE for Fold 8: 0.06801312410372365
RMSE for Fold 9: 0.0734131517249919
RMSE for Fold 10: 0.07179415254964228
Average RMSE: 0.07454328863827131
```

Then we try to find out the best predictive model that has only the best predictors using 10-fold cross validation on all the possible models in order of forward stepwide selection. After we find the best model, we list out its predictors and calculate its RMSE.

In [21]:
```
# Separate the data into feature variables and response variables (x and y)
predictor_variables = unemployment_filtered_df.drop(['Unemployment rate', 'Unem
                                                    axis=1)
y = unemployment_filtered_df['Unemployment rate']


# Initialize the best_rmse, best_model_predictors, and base model
best_rmse = float('inf')
best_model_predictors = None
lr_mdl = LinearRegression()

# Perform cross-validation on different models
for num_predictors in range(1, len(predictor_variables) + 1):
    for predictors in combinations(predictor_variables, num_predictors):
        predictors = list(predictors)
        x = unemployment_filtered_df[predictors]

        kf = KFold(n_splits=10)
        rmse_scores = []

        for train_index, test_index in kf.split(x):
            x_train, x_test, y_train, y_test = train_test_split(x, y, test_size

            lr_mdl.fit(x_train, y_train)
            y_pred = lr_mdl.predict(x_test)
            cur_rmse = rmse(y_test, y_pred)
            rmse_scores.append(cur_rmse)

        avg_rmse = sum(rmse_scores) / len(rmse_scores)

        if avg_rmse < best_rmse:
            best_rmse = avg_rmse
            best_model_predictors = predictors

# Print the best model, its predictors, and R-squared value
```

```
print("Best Model:")
print("Predictors: ", best_model_predictors)
print("RMSE: ", best_rmse)
```

```
Best Model:
Predictors:  ['Employment', 'Labour force', 'Population', 'Employment rate',
'Participation rate']
RMSE:  0.0709421379662861
```

In regards to the linear regression model, the best we are able to achieve is a RMSE around 0.0709421379662861 with the predictors of ['Labour force', 'Unemployment', 'Employment rate', 'Participation rate']

Our second model to try on is KNeighborsRegressor.

In [22]:

```
# Separate the data into feature variables and response variables (x and y)
x = unemployment_filtered_df.drop(['Unemployment rate', 'Unemployment', 'REF_DA
y = unemployment_filtered_df['Unemployment rate']

# Define the model (KNeighborsRegressor)
kn_mdl = KNeighborsRegressor()

# Number of folds for cross-validation
num_folds = 10

# Initialize an array to store RMSE values for each fold
rmse_scores = []

# Perform k-fold cross-validation
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
for train_index, test_index in kf.split(x):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)

    # Fit the model on the training data
    kn_mdl.fit(x_train, y_train)

    # Make predictions on the test data
    y_pred = kn_mdl.predict(x_test)

    # Calculate RMSE for the fold
    cur_rmse = rmse(y_test, y_pred)
    rmse_scores.append(cur_rmse)

# Print the RMSE for each fold
for fold, it_rmse in enumerate(rmse_scores, start=1):
    print('RMSE for Fold {}: {}'.format(fold, it_rmse))

# Calculate the average RMSE across all folds
average_rmse = np.mean(rmse_scores)
print('Average RMSE: {}'.format(average_rmse))
```

```
RMSE for Fold 1: 0.2168273149468081
RMSE for Fold 2: 0.42426406871192845
RMSE for Fold 3: 0.35120717372120885
RMSE for Fold 4: 0.18913442570276687
RMSE for Fold 5: 0.4897685570109823
RMSE for Fold 6: 0.7661574140466114
RMSE for Fold 7: 0.46362320786737937
RMSE for Fold 8: 0.6268746533286502
RMSE for Fold 9: 0.20171099113251603
RMSE for Fold 10: 0.6567590078302672
Average RMSE: 0.4386326814299119
```

In [23]:

```
# Separate the data into feature variables and response variables (x and y)
predictor_variables = unemployment_filtered_df.drop(['Unemployment rate', 'REF_
                                                       axis=1)
y = unemployment_filtered_df['Unemployment rate']

# Perform cross-validation for different models
best_rmse = float('inf')
best_model_predictors = None

# Initialize the base model
kn_mdl = KNeighborsRegressor()

for num_predictors in range(1, len(predictor_variables) + 1):
    for predictors in combinations(predictor_variables, num_predictors):
        predictors = list(predictors)
        x = unemployment_filtered_df[predictors]

        kf = KFold(n_splits=10)
        rmse_scores = []

        for train_index, test_index in kf.split(x):
            x_train, x_test, y_train, y_test = train_test_split(x, y, test_size

            kn_mdl.fit(x_train, y_train)
            y_pred = kn_mdl.predict(x_test)
            cur_rmse = rmse(y_test, y_pred)
            rmse_scores.append(cur_rmse)

        avg_rmse = sum(rmse_scores) / len(rmse_scores)

        if avg_rmse < best_rmse:
            best_rmse = avg_rmse
            best_model_predictors = predictors

# Print the best model, its predictors, and R-squared value
print("Best Model:")
print("Predictors: ", best_model_predictors)
print("RMSE: ", best_rmse)
```

```
Best Model:
Predictors:  ['Population', 'Unemployment', 'Employment rate', 'Participation
rate']
RMSE:  0.1934981272555511
```

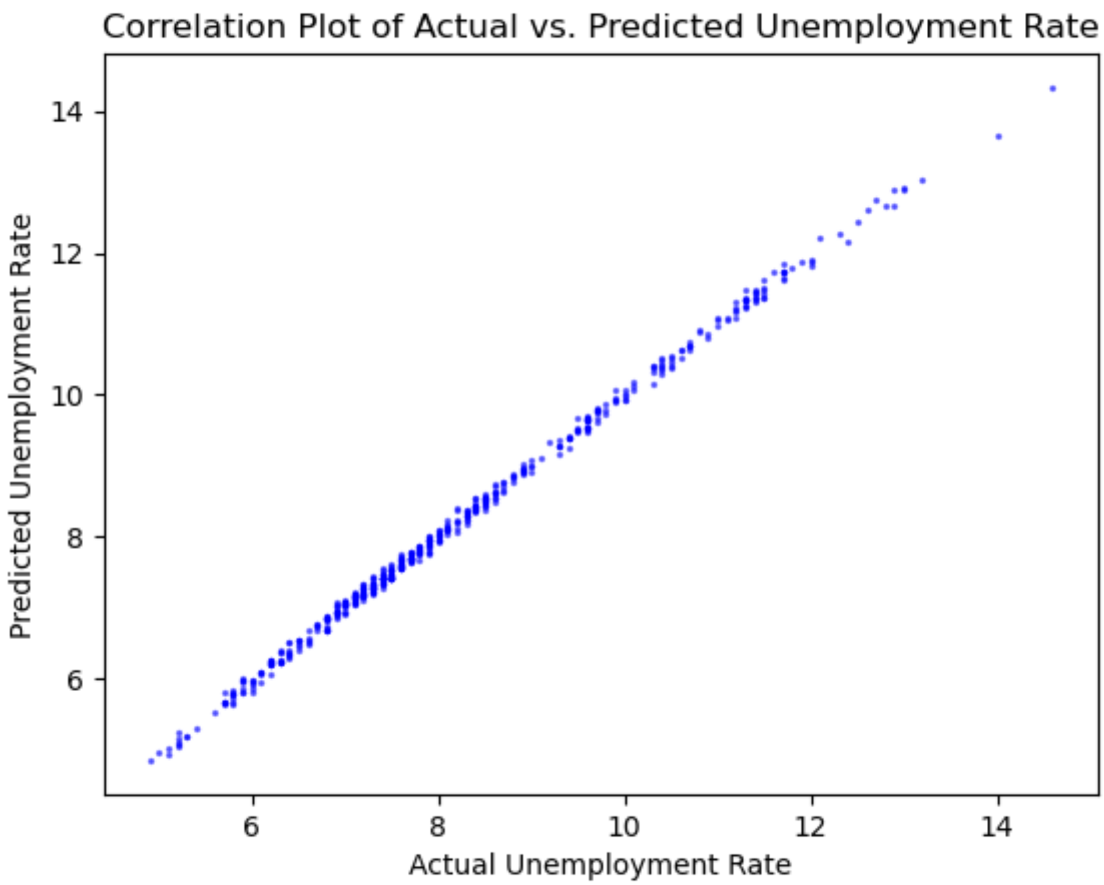
In regards to the KNeighborsRegressor model, the best we are able to achieve is a RMSE around 0.20181145980449777 with the predictors of ['Population', 'Unemployment', 'Employment rate', 'Participation rate']

Model Evaluation

In [24]:
```
x = unemployment_filtered_df[best_model_predictors]
y = unemployment_filtered_df['Unemployment rate']

lr_mdl = LinearRegression()
lr_mdl.fit(x, y)

# Calculate the R-squared value and average RMSE for the best model
y_pred = lr_mdl.predict(x)
r2 = r2_score(y, y_pred)
cur_rmse = rmse(y, y_pred)

# Print the R-squared value and average RMSE for the best model
print("R-squared: ", r2)
print("RMSE: ", cur_rmse)
```

```
R-squared:  0.9982466809509498
RMSE:  0.07332930073554858
```

In [25]:
```
import matplotlib.pyplot as plt

# Create a scatter plot
plt.scatter(y, y_pred, c='blue', alpha=0.5, s=2)

# Add labels and title
plt.xlabel("Actual Unemployment Rate")
plt.ylabel("Predicted Unemployment Rate")
plt.title("Correlation Plot of Actual vs. Predicted Unemployment Rate")

# Add a fitted line (linear regression line)
fit = np.polyfit(y, y_pred, 1)
fit_fn = np.poly1d(fit)

# Display the plot
plt.show()
```

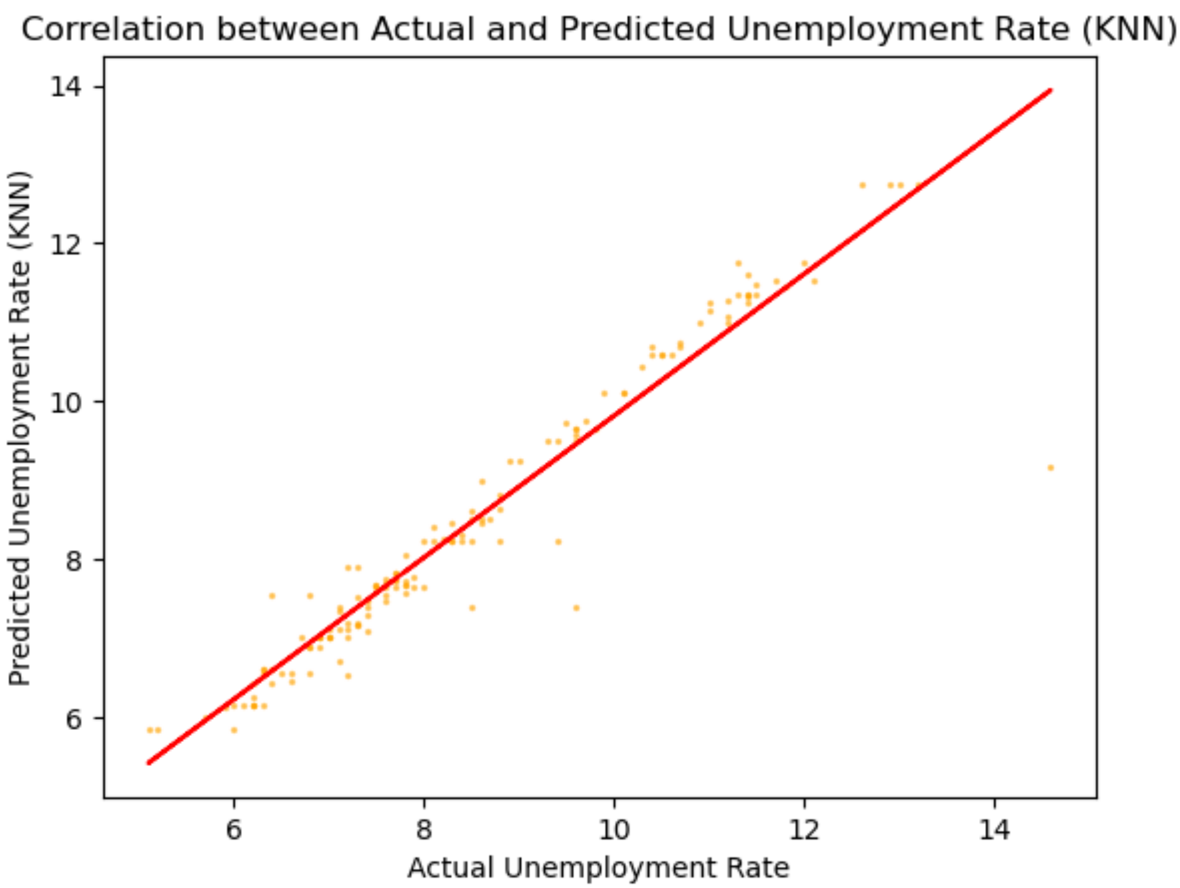


In [26]:
```
# check the knn models and store the performance of this model
knn_rmse = []
knn_r2 = []

for i in range(20):
    knn_mdl = KNeighborsRegressor()

    # We splitted the data to train data and test data to train and test our
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
    knn_mdl.fit(x_test, y_test)
    test_outcome = knn_mdl.predict(x_test)

    # We used x_test and y_test to test the RMSE of the model
    knn_rmse.append(np.sqrt(mean_squared_error(y_test, test_outcome)))
    knn_r2.append(r2_score(y_test, test_outcome))
print("R-squared: ", max(knn_r2))
print("RMSE: ", min(knn_rmse))
```

```
R-squared:  0.9822245824834721
Average RMSE:  0.2426845167040685
```

In [27]:
```
import numpy as np
import matplotlib.pyplot as plt

# Create a scatter plot
plt.scatter(y_test, test_outcome, c='orange', alpha=0.5, s=2)

# Calculate the line of best fit
slope, intercept = np.polyfit(y_test, test_outcome, 1)
best_fit_line = slope * y_test + intercept
```

```
# Plot the fitted line
plt.plot(y_test, best_fit_line, color='red')

# Add labels and title
plt.xlabel("Actual Unemployment Rate")
plt.ylabel("Predicted Unemployment Rate (KNN)")
plt.title("Correlation between Actual and Predicted Unemployment Rate (KNN)")

# Display the plot
plt.show()
```



Final Model Estimation

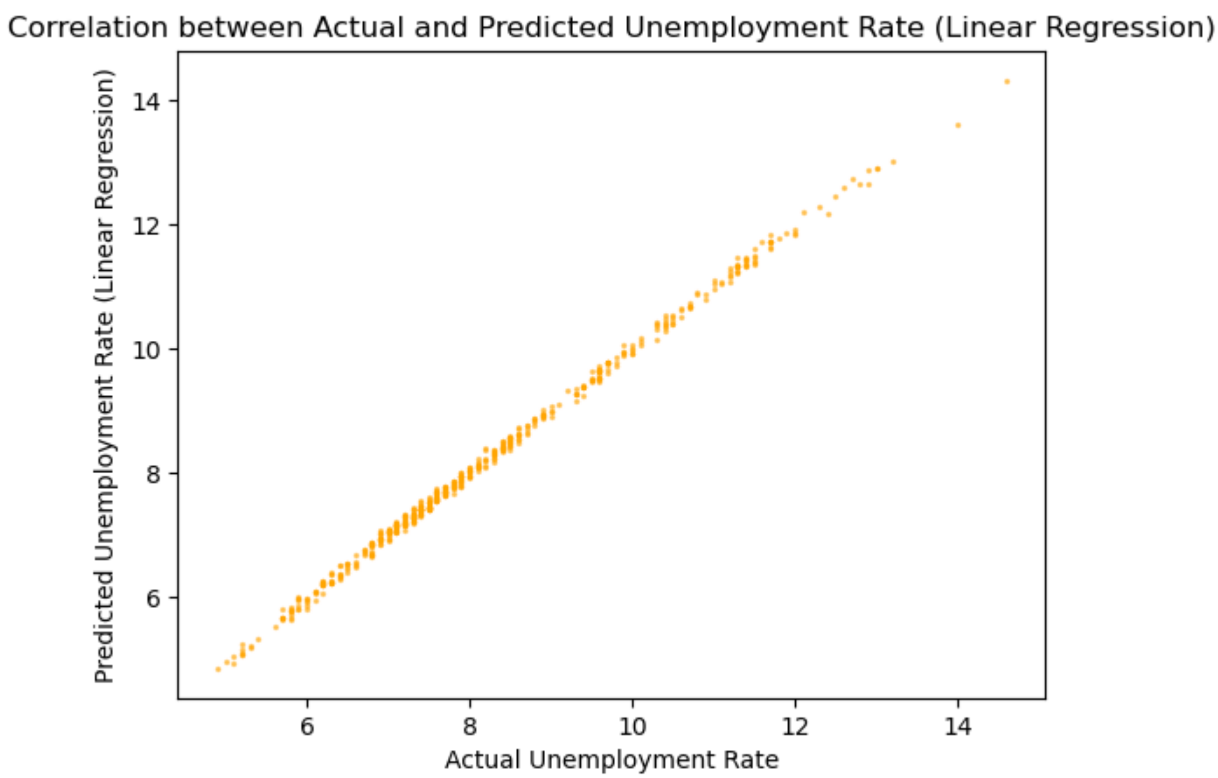
In [28]:

```
x = unemployment_filtered_df[['Labour force', 'Unemployment', 'Employment rate'
y = unemployment_filtered_df['Unemployment rate']

final_mdl = LinearRegression()
final_mdl.fit(x,y)

final_pred = final_mdl.predict(x)
```

In [33]:

```
# Create a scatter plot
# Add labels and title
plt.xlabel("Actual Unemployment Rate")
plt.ylabel("Predicted Unemployment Rate (Linear Regression)")
plt.title("Correlation between Actual and Predicted Unemployment Rate (Linear R

plt.scatter(y, final_pred, c='orange', alpha=0.5, s=2)
```

Out[33]: <matplotlib.collections.PathCollection at 0x7fbf49a53ee0>



In [30]:
```
print("R-squared: ", r2_score(y, final_pred))
print("RMSE: ", rmse(y, final_pred))
```

```
R-squared:  0.9982564362806949
RMSE:  0.07312501694396009
```

In [32]:
```
# Retrieve the coefficients (slope) and intercept
coefficients = final_mdl.coef_
intercept = final_mdl.intercept_

# Print the coefficients and intercept
print("Coefficients:", coefficients)
print("Intercept:", intercept)
```

```
Coefficients: [ 7.26220404e-08 -7.63438245e-07 -1.48126979e+00  1.35508167e+0
0]
Intercept: 8.418970167545531
```