INTRODUCTION

Labor productivity is a crucial metric that reflects the efficiency and output of labor within various sectors of the economy. Understanding and predicting sector-wise labor productivity trends is essential for policymakers, businesses, and researchers to make informed decisions and strategies for economic growth and development.

This project focuses on analysing historical data spanning from 1960 to 1917 to predict sector-wise labor productivity trends in India through various models such as multiple regression model for predicting and ARIMA model for forecasting.

ABOUT THE DATASET

The dataset consists of information related to sector-wise labor productivity indices in India spanning from 1960 to the year 1918. The dataset provides insights into the performance and productivity trends across various sectors of the Indian economy over the specified time period. It enables the analysis of labor productivity dynamics, employment trends, and economic contributions of different sectors. The inclusion of real and nominal value added allows for comparisons between the actual output and the output adjusted for inflation, providing a comprehensive understanding of sectoral productivity.

The data was obtained from https://data.worldbank.org/ and includes the following variables:

1. Sector: Categorization of economic sectors such as agriculture, mining, manufacturing, utilities, construction, trade services, transport services, finance and business services, and other services.
2. Year: The year for which the data was recorded, ranging from 1960 to 1918.
3. Value Added Nominal: The nominal value added within each sector, measured in a currency unit.
4. Value Added Real: The real value added within each sector, adjusted for inflation and measured in a currency unit.
5. Employment: The number of individuals employed within each sector.
6. Labor Productivity Real: The real labor productivity index within each sector, representing the efficiency of labor in generating output, adjusted for inflation.
7. Labor Productivity PPP: The labor productivity index within each sector, adjusted for purchasing power parity (PPP), reflecting the productivity level relative to other countries.

DATA PREPROCESSING

The raw data collected had to be cleaned before it could be used for analysis. The steps undertaken to clean the data are:

1. Below is a screenshot of the Original Data:

```
df= pd.read_csv("/content/CAPSTONES DS.csv")
df.head(10)
```

	sector	year	Value_added_nominal	Value_added_real	Employment	Labor_productivity_real	Labor_productivity_PPP
0	Total	1960	172491.609400	6.679817e+06	162462.625000	41.116024	2.878634
1	1.Agriculture	1960	74004.848410	3.659541e+06	111090.090600	32.942104	1.806162
2	2.Mining	1960	1876.259420	1.937307e+05	1100.909584	175.973267	4.620758
3	3.Manufacturing	1960	24615.863520	6.691691e+05	17624.404550	37.968327	3.786802
4	4.Utilities	1960	1190.584014	3.703341e+04	313.709234	118.050110	10.289747
5	5.Construction	1960	7279.247499	4.226801e+05	4184.319981	101.015243	4.716649
6	6.Trade services	1960	8334.015751	3.839589e+05	8533.829980	44.992565	2.647782
7	7.Transport services	1960	5794.794939	1.328273e+05	2746.315607	48.365635	5.720837
8	8.Finance amd business services	1960	35989.338310	7.908434e+05	562.705184	1405.431274	173.406311
9	9.Other services	1960	13406.660590	3.900329e+05	16306.346560	23.919085	2.229133

2. Checked for data types

```
] df.dtypes

sector                      object
year                         int64
Value_added_nominal        float64
Value_added_real           float64
Employment                 float64
Labor_productivity_real    float64
Labor_productivity_PPP     float64
dtype: object
```

3. Converted datatypes from float to integer

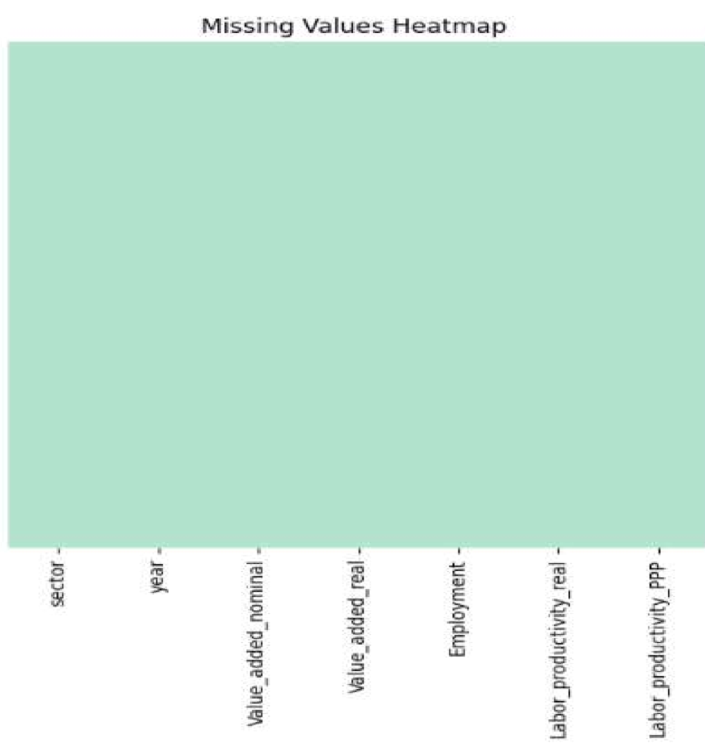
```
[124] df.columns

Index(['sector', 'year', 'Value_added_nominal', 'Value_added_real',
       'Employment', 'Labor_productivity_real', 'Labor_productivity_PPP'],
      dtype='object')
```

```
[125] columns_to_convert = ['Value_added_nominal', 'Value_added_real',
              'Employment', 'Labor_productivity_real', 'Labor_productivity_PPP']

      for column in columns_to_convert:
          df[column] = df[column].fillna(0).astype(int)
```

```
df.dtypes

sector                     object
year                        int64
Value_added_nominal         int64
Value_added_real            int64
Employment                  int64
Labor_productivity_real     int64
Labor_productivity_PPP      int64
dtype: object
```

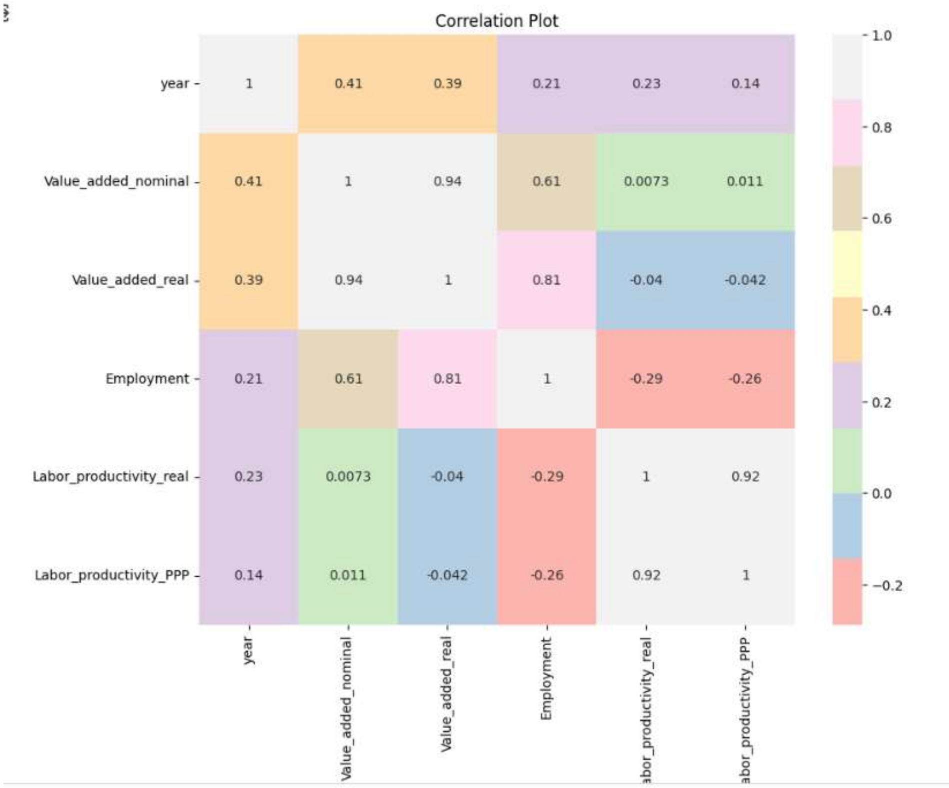
4. Checked for missing values but there were no missing values in the dataset.

```
# Create a heatmap for missing values
plt.figure(figsize=(6, 6))
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='Pastel2')
plt.title('Missing Values Heatmap ')
plt.show()
```



5. Created a correlation matrix and visualize it as a heatmap using the Seaborn library

```python
df2 = df.drop(columns=['sector'])
corr_matrix = df2.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='Pastel1')
plt.title('Correlation Plot')
plt.show()
```



6. Detected outliers:

```
Outliers in 'Value_added_nominal':
330     8072624
340     9403615
350    10959510
360    12698214
370    14102652
         ...
575    12136278
576    18337198
577     9895426
578    32527892
579    22066518
Name: Value_added_nominal, Length: 85, dtype: int64
Outliers in 'Value_added_real':
250     16290140
260     16976306
270     17688428
280     19667692
```

7. Handling outliers:

```
Outlier counts in year:
Z-Score Outliers Count: 0
IQR Outliers Count: 0

Outlier counts in Value_added_nominal:
Z-Score Outliers Count: 10
IQR Outliers Count: 85

Outlier counts in Value_added_real:
Z-Score Outliers Count: 14
IQR Outliers Count: 42

Outlier counts in Employment:
Z-Score Outliers Count: 19
IQR Outliers Count: 108

Outlier counts in Labor_productivity_real:
Z-Score Outliers Count: 12
IQR Outliers Count: 100

Outlier counts in Labor_productivity_PPP:
Z-Score Outliers Count: 9
IQR Outliers Count: 95
```

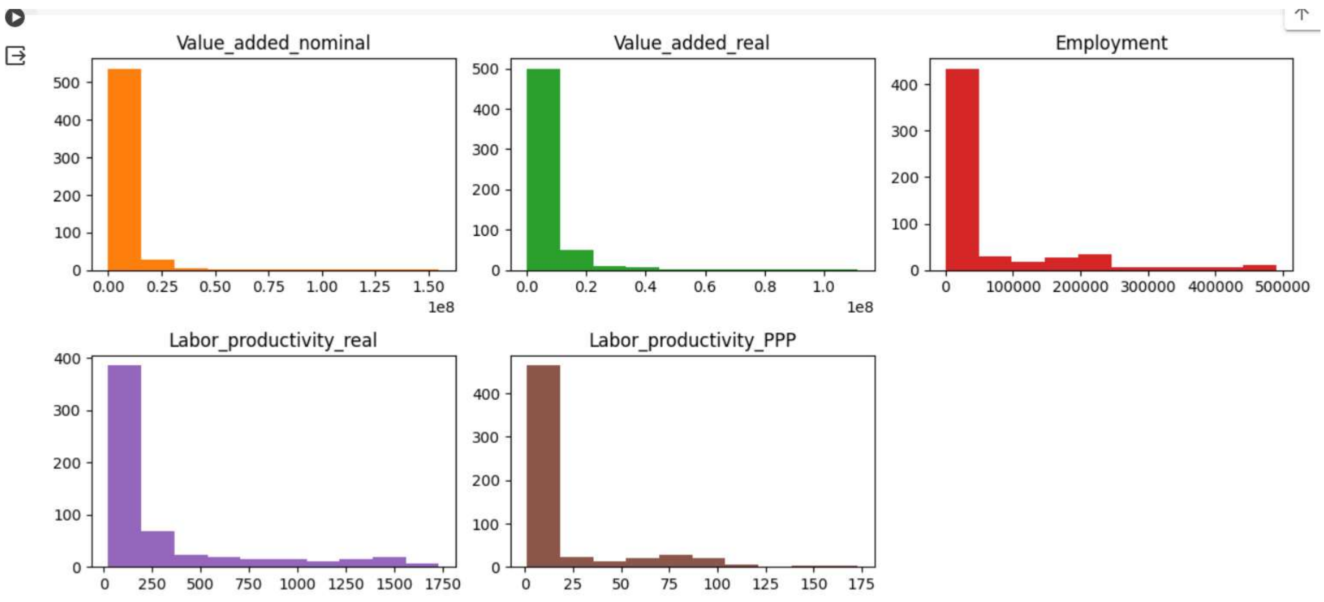
8. Converted categorical values in the 'sector' column of the Data Frame into numerical values using category mapping

```
category_mapping = {
    'Total': 1,
    '1.Agriculture': 2,
    '2.Mining': 3,
    '3.Manufacturing': 4,
    '4.Utilities': 5,
    '5.Construction': 6,
    '6.Trade services': 7,
    '7.Transport services': 8,
    '8.Finance amd business services': 9,
    '9.Other services': 10
}

# Replace categories with numbers
df['sector'] = df['sector'].replace(category_mapping)

print("Updated Sectors:")
print(df['sector'].value_counts())
```

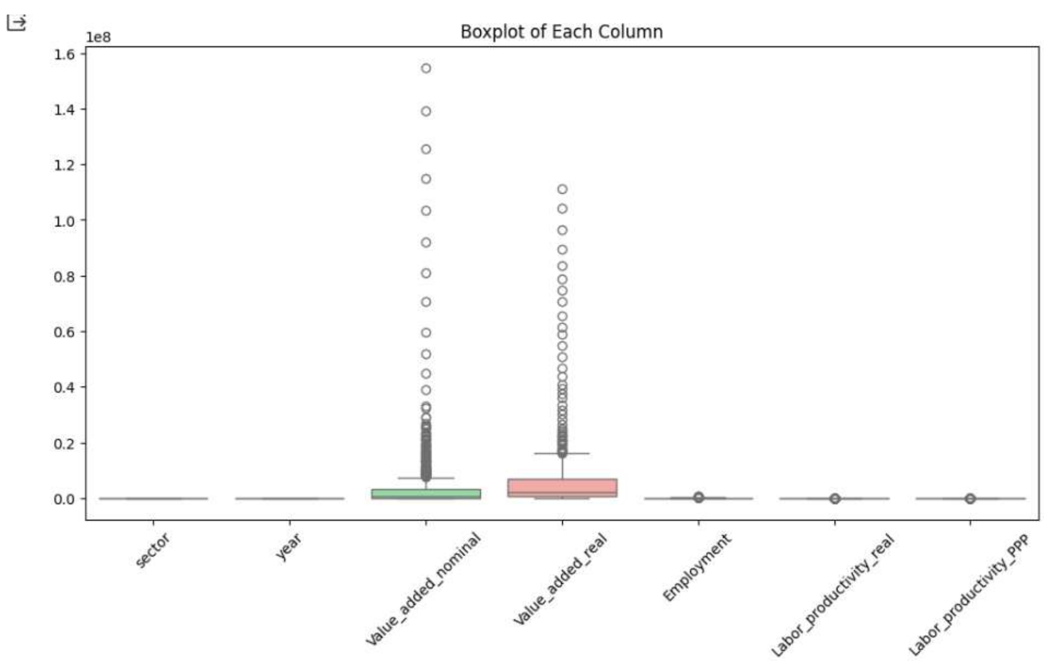
```
Updated Sectors:
sector
1     58
2     58
3     58
4     58
5     58
6     58
```

EXPLORATORY DATA ANALYSIS

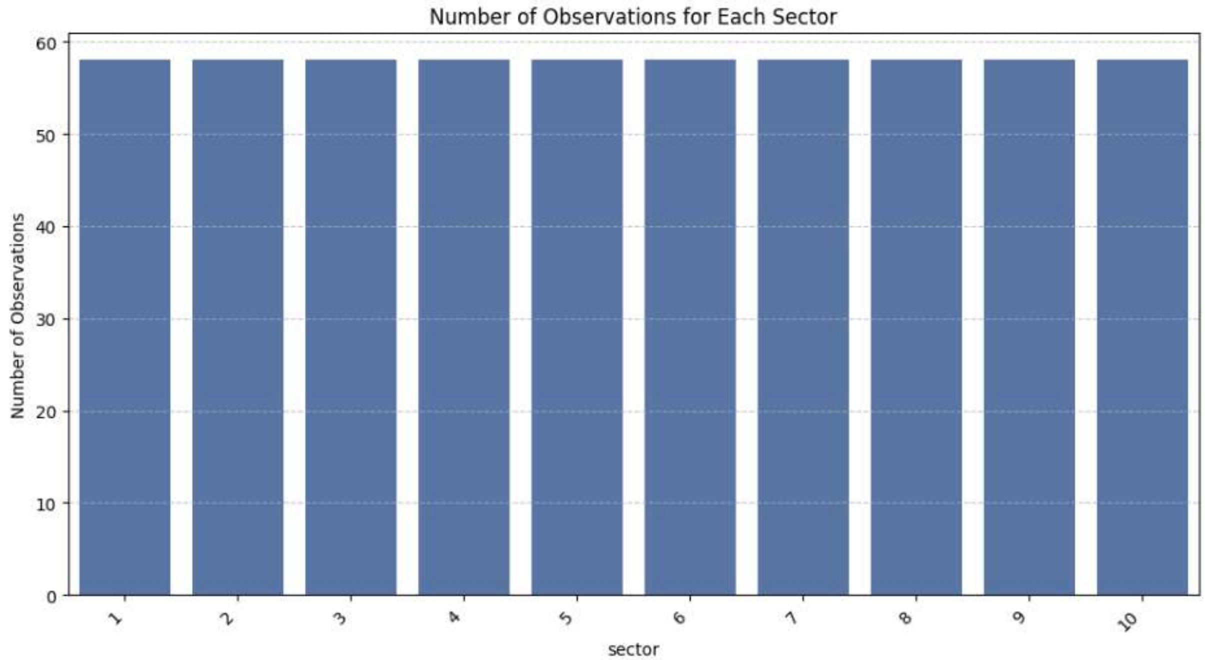
1. Generated histograms for each numerical column in the Data Frame



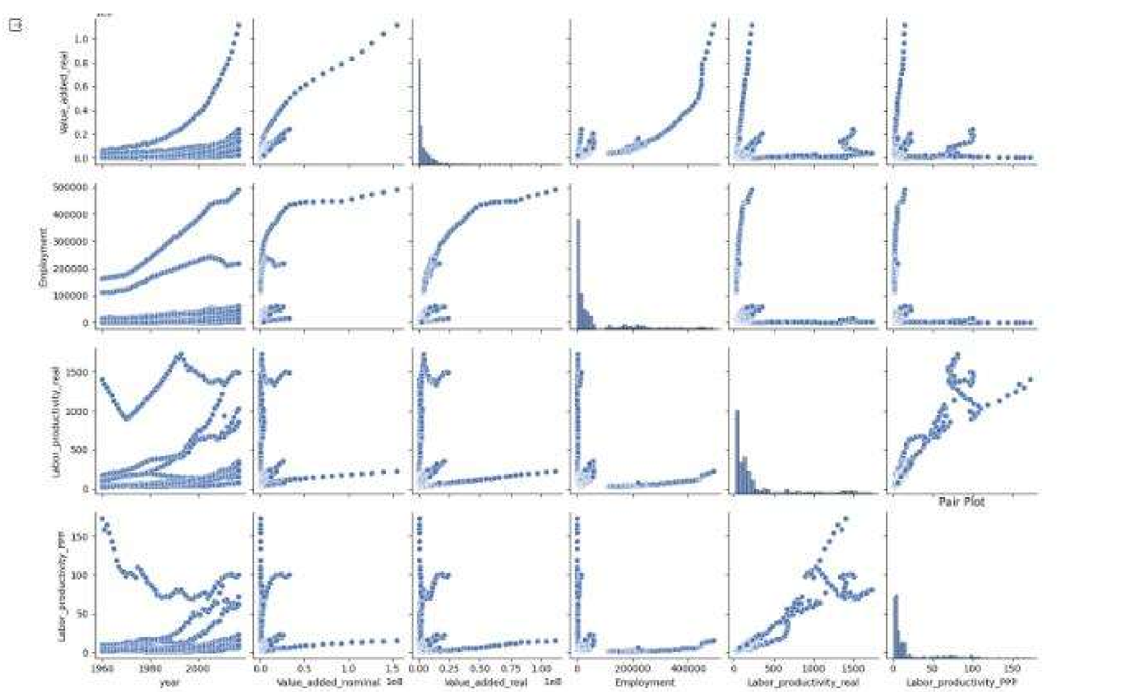
2. Created Box Plot for each column



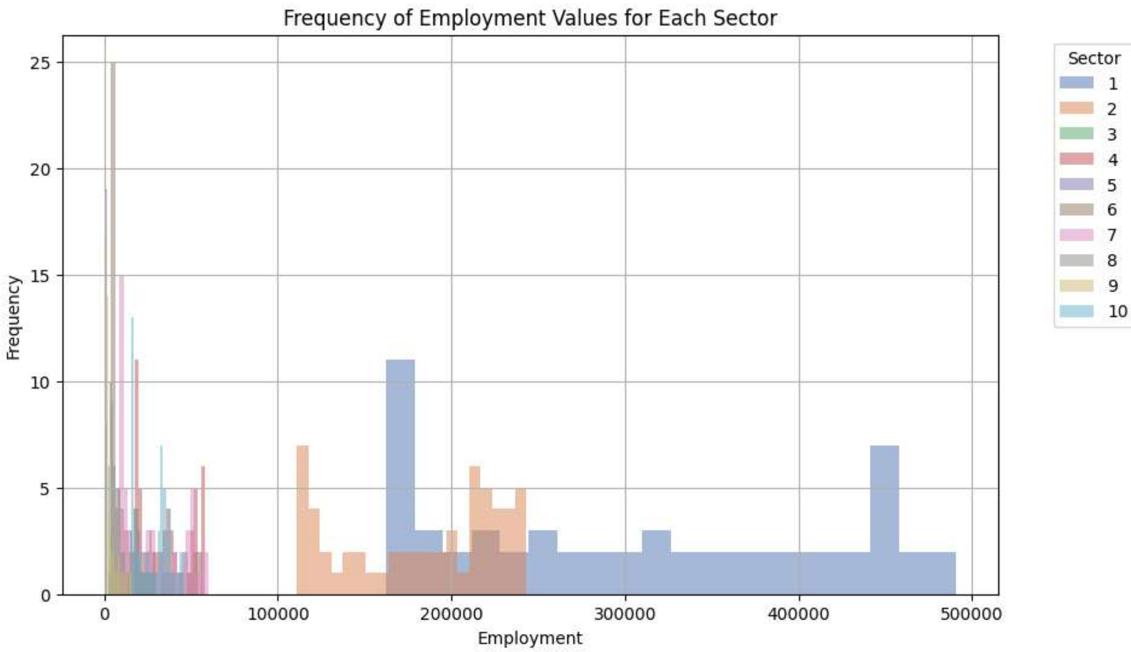
3. Created a bar plot that visualizes the distribution of observations across different sectors



4. Created scatterplots to visually explore the relationships between multiple variables in the dataset



5. Generated the histogram for the distribution of employment values within each sector in the dataset.



AUGUMENTED DICKEY - FULLER TEST

After this I performed the Augmented Dickey-Fuller (ADF) test on each column of a Data Frame to determine whether the time series data in each column is stationary or non-stationary. If a variable is non-stationary then its statistical properties (such as mean and variance) are not constant over time.

```
   Augmented Dickey-Fuller Test on "Employment"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = 0.3918
No. Lags Chosen       = 19
Critical value 1%     = -3.442
Critical value 5%     = -2.867
Critical value 10%    = -2.57
=> P-Value = 0.9812. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.


   Augmented Dickey-Fuller Test on "Labor_productivity_real"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = 1.3054
No. Lags Chosen       = 19
Critical value 1%     = -3.442
Critical value 5%     = -2.867
Critical value 10%    = -2.57
=> P-Value = 0.9966. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.


   Augmented Dickey-Fuller Test on "Labor_productivity_PPP"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = 1.6853
No. Lags Chosen       = 19
Critical value 1%     = -3.442
Critical value 5%     = -2.867
Critical value 10%    = -2.57
=> P-Value = 0.9981. Weak evidence to reject the Null Hypothesis.
=> Series is Non-Stationary.
```

PREDICTION

1. Splitting the dataset into training and testing sets for model training

STANDARIZATION

```
[154] # Split data into training and testing sets
      X = df.drop('sector', axis=1)
      y = df['sector']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[155] scaler = StandardScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)
```

2. Principal Component Analysis: For dimensionality reduction

```
[156] pca=PCA(n_components=2)
      PC=pca.fit_transform(X)
      principalDF=pd.DataFrame(data=PC,columns=['pc1','pc2'])
      df = pd.concat([principalDF, df.reset_index(drop=True)['sector']], axis = 1)
      df
```

	pc1	pc2	sector
0	-3.204659e+06	-3.272375e+06	Total
1	-5.265446e+06	-1.061597e+06	1.Agriculture
2	-7.599629e+06	1.503100e+06	2.Mining
3	-7.269746e+06	1.159698e+06	3.Manufacturing
4	-7.703207e+06	1.620675e+06	4.Utilities
...	...	...	...
575	7.284121e+06	2.905950e+06	5.Construction
576	1.502034e+07	3.473924e+06	6.Trade services
577	4.420068e+06	2.779347e+06	7.Transport services
578	3.275505e+07	4.739484e+06	8.Finance amd business services
579	1.857788e+07	5.069770e+06	9.Other services

LINEAR REGRESSION MODEL

Performed linear regression on the training data, predicted the target variable on the test data using the trained linear regression model.

Calculated the root mean squared error (RMSE) between the predicted values and the actual values in the test set. The output Linear Regression RMSE: 2.5310716847731147e-11 indicates that the root mean squared error (RMSE) between the actual and predicted values is approximately 2.53e-11. This value is very close to zero, which suggests that the linear regression model is performing very well on the test data, with very little error in its predictions.

Checked for the accuracy of the model, an R-squared of 1.0 indicates that the model explains 100% of the variance in the target variable, meaning that the model perfectly predicts the target variable based on the features.

```
[145] #LINEAR REGRESSION
      from sklearn.linear_model import LinearRegression
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import mean_squared_error
```

```
X = df[['Employment']]  # Independent variable
y = df['Employment']  # Dependent variable
```

```
[150] # Assuming X_train, X_test, y_train, y_test are your training and testing data
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
```

```
[151] linear_reg = LinearRegression()
      linear_reg.fit(X_train, y_train)
      y_pred_linear = linear_reg.predict(X_test)
      rmse_linear = mean_squared_error(y_test, y_pred_linear, squared=False)
      print("Linear Regression RMSE:", rmse_linear)

      Linear Regression RMSE: 2.5310716847731147e-11
```

```
#Accuracy
from sklearn.metrics import r2_score
r2_linear = r2_score(y_test, y_pred_linear)
print("Linear Regression R-squared:", r2_linear)

Linear Regression R-squared: 1.0
```

RANDOM FOREST CLASSIFIER

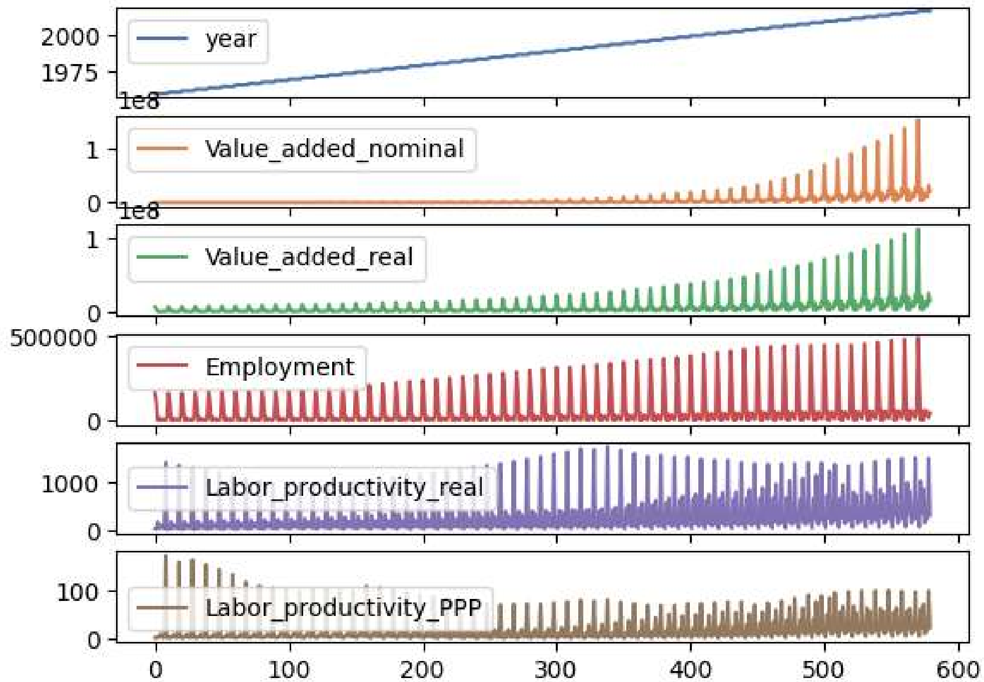
The output indicates that the accuracy of the Random Forest classifier on the test data is 93%.

```
[157] from sklearn.ensemble import RandomForestClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import accuracy_score

      clf = RandomForestClassifier(n_estimators=100, random_state=42)
      clf.fit(X_train, y_train)
      y_pred = clf.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)
      print(f'Accuracy: {accuracy:.2f}')
```

FORECASTING

Visualize the data in the Data Frame by plotting each column separately:

```
array([<Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >, <Axes: >],
      dtype=object)
```



ARIMA (Autoregressive Integrated Moving Average)

```
import pmdarima as pm

model = pm.auto_arima(df['Labor_productivity_PPP'],
                        m=12, seasonal=True,
                      start_p=0, start_q=0, max_order=4, test='adf',error_action='ignore',
                           suppress_warnings=True,
                      stepwise=True, trace=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=5560.658, Time=1.99 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=5568.722, Time=0.14 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=5543.167, Time=2.83 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=5547.917, Time=1.20 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=5778.428, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=5560.326, Time=0.12 sec
```

```
model.summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	580
Model:	SARIMAX(4, 0, 0)x(2, 0, [1, 2], 12)	**Log Likelihood**	-2520.391
Date:	Fri, 03 May 2024	**AIC**	5060.781
Time:	15:31:27	**BIC**	5104.411
Sample:	0 - 580	**HQIC**	5077.791
Covariance Type:	opg		

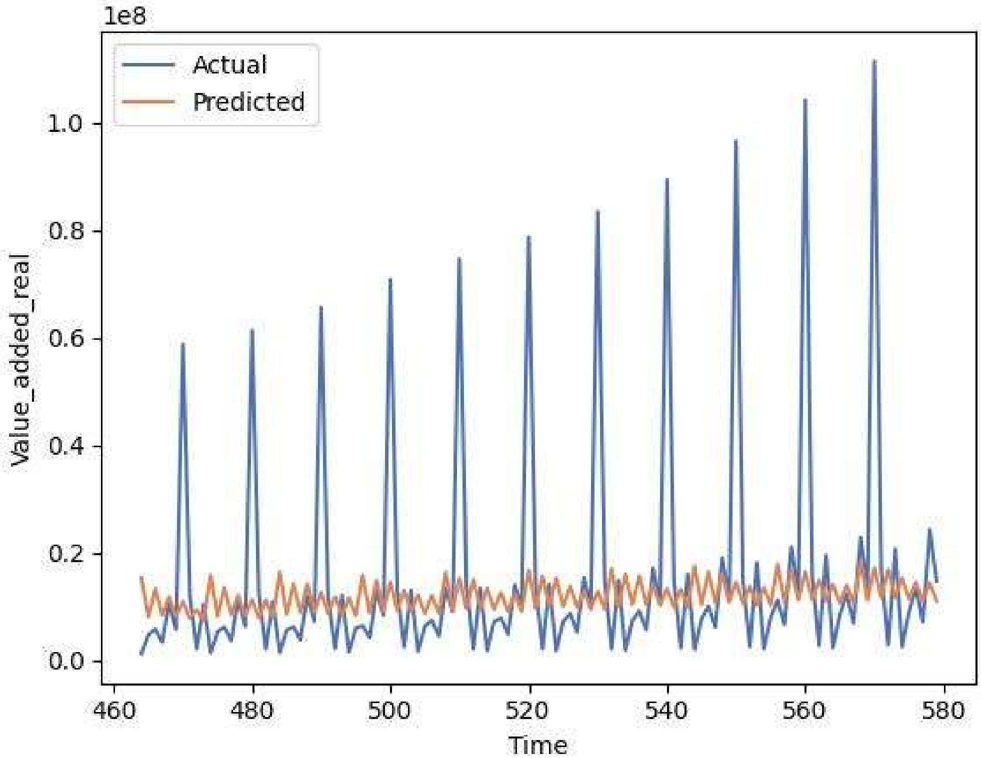
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
intercept	26.7027	6.369	4.193	0.000	14.220	39.185
ar.L1	-0.0180	0.109	-0.165	0.869	-0.232	0.196
ar.L2	0.8715	0.037	23.468	0.000	0.799	0.944
ar.L3	-0.0410	0.123	-0.334	0.739	-0.282	0.200
ar.L4	-0.3021	0.033	-9.292	0.000	-0.366	-0.238
ar.S.L12	-1.6070	0.005	-297.587	0.000	-1.618	-1.596
ar.S.L24	-0.9857	0.003	-291.172	0.000	-0.992	-0.979
ma.S.L12	1.4012	0.045	31.445	0.000	1.314	1.489
ma.S.L24	0.6474	0.045	14.321	0.000	0.559	0.736
sigma2	301.5951	16.628	18.137	0.000	269.004	334.186

Ljung-Box (L1) (Q):	4.67	**Jarque-Bera (JB):**	1133.74
Prob(Q):	0.03	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.34	**Skew:**	1.20
Prob(H) (two-sided):	0.00	**Kurtosis:**	9.41

The model appears to fit the data well, as indicated by the significant coefficients and the low values of information criteria. The residuals seem to be independent and normally distributed based on the diagnostic tests.

SARIMA MODEL

```
Mean Squared Error: 493919156840630.1
```



The plot suggests that the SARIMA model was able to capture the underlying trend and seasonality of the data reasonably well. The code also calculates the mean squared error (MSE) which is 493919156840630.1 a very large value, suggesting significant discrepancies between the actual data points and the corresponding predictions made by the SARIMA model. This indicates that the model's forecasts might not be very accurate for this particular dataset.