AI-DRIVEN EXPLORATION AND PREDICTION OF COMPANY REGISTRATION TRENDS WITH REGISTRAR OF COMPANIES (ROC)

PHASE :4

DEVELOPMENT PHASE PART 2

PROCESS AND DATA:

1. ***Data Collection**: Gather historical stock prices, trading volumes, and relevant financial data. Sources could include financial databases, APIs, or web scraping tools.*
2. ***Data Pre-processing**: Clean the data, handle missing values, and perform feature engineering. This step might involve normalization, scaling, or transforming the data to make it suitable for the chosen model.*
3. ***Feature Selection:** Choose relevant features that might affect stock prices, such as historical prices, trading volumes, news sentiment, economic indicators, and company-specific information.*
4. ***Model Selection:** Select an appropriate machine learning algorithm such as linear regression, decision trees, random forests, or deep learning models like recurrent neural networks (RNNs) or long short-term memory networks (LSTMs).*
5. ***Training the Model:** Use a portion of the data to train the model, adjusting parameters and hyper parameters to optimize performance.*
6. ***Model Evaluation:** Assess the model's performance using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) on a validation dataset.*
7. ***Testing the Model:** Apply the trained model to a separate test dataset to evaluate its predictive power.*
8. ***Iterate and Refine:** Fine-tune the model by iterating on the pre-processing steps, feature selection, and model selection to improve predictive accuracy.*

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from IPython.display import HTML
from sklearn.preprocessing import LabelEncoder
```

```
#Get the data
df = pd.read_csv('/Data_Gov_Tamil_Nadu.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11370 entries, 0 to 11369
Data columns (total 17 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   CORPORATE_IDENTIFICATION_NUMBER          11370 non-null  object
 1   COMPANY_NAME                             11370 non-null  object
 2   COMPANY_STATUS                           11369 non-null  object
 3   COMPANY_CLASS                            11058 non-null  object
 4   COMPANY_CATEGORY                         11058 non-null  object
 5   COMPANY_SUB_CATEGORY                     11058 non-null  object
 6   DATE_OF_REGISTRATION                     11331 non-null  object
 7   REGISTERED_STATE                         11369 non-null  object
 8   AUTHORIZED_CAP                           11369 non-null  float64
 9   PAIDUP_CAPITAL                           11369 non-null  float64
 10  INDUSTRIAL_CLASS                         11059 non-null  float64
 11  PRINCIPAL_BUSINESS_ACTIVITY_AS_PER_CIN   11369 non-null  object
 12  REGISTERED_OFFICE_ADDRESS                11357 non-null  object
 13  REGISTRAR_OF_COMPANIES                   11346 non-null  object
 14  EMAIL_ADDR                               7924 non-null   object
 15  LATEST_YEAR_ANNUAL_RETURN                5554 non-null   object
 16  LATEST_YEAR_FINANCIAL_STATEMENT          5577 non-null   object
dtypes: float64(3), object(14)
memory usage: 1.5+ MB
```

CodeText

```
#check Duplicate Data
print(df.duplicated().sum())

0
```

```
#date and time is converted object in to datetime64
df['DATE_OF_REGISTRATION'] =
pd.to_datetime(df['DATE_OF_REGISTRATION'])
df['day_of_week'] = df['DATE_OF_REGISTRATION'].dt.day_name()
df['month'] = df['DATE_OF_REGISTRATION'].dt.month
print(df['DATE_OF_REGISTRATION'])
```

```
0       1961-01-12
1       2002-02-28
2       1982-01-03
3       2002-02-28
4       2002-02-28
           ...
11365   2004-10-12
11366   2005-05-30
11367   2005-10-06
11368   2005-09-16
11369   2002-02-28
Name: DATE_OF_REGISTRATION, Length: 11370, dtype: datetime64[ns]
```

```
#checking the missing values
print(df.isnull().sum())
```

```
CORPORATE_IDENTIFICATION_NUMBER              0
COMPANY_NAME                                 0
COMPANY_STATUS                               1
COMPANY_CLASS                              312
COMPANY_CATEGORY                           312
COMPANY_SUB_CATEGORY                       312
DATE_OF_REGISTRATION                        39
REGISTERED_STATE                             1
AUTHORIZED_CAP                               1
PAIDUP_CAPITAL                               1
INDUSTRIAL_CLASS                           311
PRINCIPAL_BUSINESS_ACTIVITY_AS_PER_CIN       1
REGISTERED_OFFICE_ADDRESS                   13
REGISTRAR_OF_COMPANIES                      24
EMAIL_ADDR                                3446
LATEST_YEAR_ANNUAL_RETURN                 5816
LATEST_YEAR_FINANCIAL_STATEMENT           5793
day_of_week                                 39
month                                       39
dtype: int64
```

```python
# fill missing values in column with their most repeated
df['COMPANY_CLASS'].fillna(df['COMPANY_CLASS'].mode()[0],
inplace=True)
df['COMPANY_CATEGORY'].fillna(df['COMPANY_CATEGORY'].mode()[0]
, inplace=True)
df['COMPANY_SUB_CATEGORY'].fillna(df['COMPANY_SUB_CATEGORY'].m
ode()[0], inplace=True)
df['DATE_OF_REGISTRATION'].fillna(df['DATE_OF_REGISTRATION'].m
ode()[0], inplace=True)
df['INDUSTRIAL_CLASS'].fillna(df['INDUSTRIAL_CLASS'].mode()[0]
, inplace=True)
df['REGISTERED_OFFICE_ADDRESS'].fillna(df['REGISTERED_OFFICE_A
DDRESS'].mode()[0], inplace=True)
df['REGISTRAR_OF_COMPANIES'].fillna(df['REGISTRAR_OF_COMPANIES
'].mode()[0], inplace=True)
df['EMAIL_ADDR'].fillna(df['EMAIL_ADDR'].mode()[0],
inplace=True)
df['LATEST_YEAR_ANNUAL_RETURN'].fillna(df['LATEST_YEAR_ANNUAL_
RETURN'].mode()[0], inplace=True)
df['LATEST_YEAR_FINANCIAL_STATEMENT'].fillna(df['LATEST_YEAR_F
INANCIAL_STATEMENT'].mode()[0], inplace=True)
```

```python
# interpolate missing values using linear interpolation
df['COMPANY_CLASS'].interpolate(inplace=True)
df['COMPANY_CATEGORY'].interpolate(inplace=True)
df['COMPANY_SUB_CATEGORY'].interpolate(inplace=True)
df['INDUSTRIAL_CLASS'].interpolate(inplace=True)
df['REGISTERED_OFFICE_ADDRESS'].interpolate(inplace=True)
df['REGISTRAR_OF_COMPANIES'].interpolate(inplace=True)
df['EMAIL_ADDR'].interpolate(inplace=True)
df['LATEST_YEAR_ANNUAL_RETURN'].interpolate(inplace=True)
df['LATEST_YEAR_FINANCIAL_STATEMENT'].interpolate(inplace=True
)
```

```python
print(df.isnull().sum())
```

```
CORPORATE_IDENTIFICATION_NUMBER          0
COMPANY_NAME                             0
COMPANY_STATUS                           1
COMPANY_CLASS                            0
COMPANY_CATEGORY                         0
COMPANY_SUB_CATEGORY                     0
DATE_OF_REGISTRATION                     0
REGISTERED_STATE                         1
AUTHORIZED_CAP                           1
PAIDUP_CAPITAL                           1
INDUSTRIAL_CLASS                         0
```

```
PRINCIPAL_BUSINESS_ACTIVITY_AS_PER_CIN     1
REGISTERED_OFFICE_ADDRESS                  0
REGISTRAR_OF_COMPANIES                     0
EMAIL_ADDR                                 0
LATEST_YEAR_ANNUAL_RETURN                  0
LATEST_YEAR_FINANCIAL_STATEMENT            0
day_of_week                                0
month                                      0
```

```python
# standardize the numerical data
encoder = LabelEncoder()
df['COMPANY_CLASS'] =
encoder.fit_transform(df['COMPANY_CLASS'])
df['COMPANY_CLASS'] =
encoder.fit_transform(df['COMPANY_CLASS'])
print(df['COMPANY_CLASS'].to_string())
```

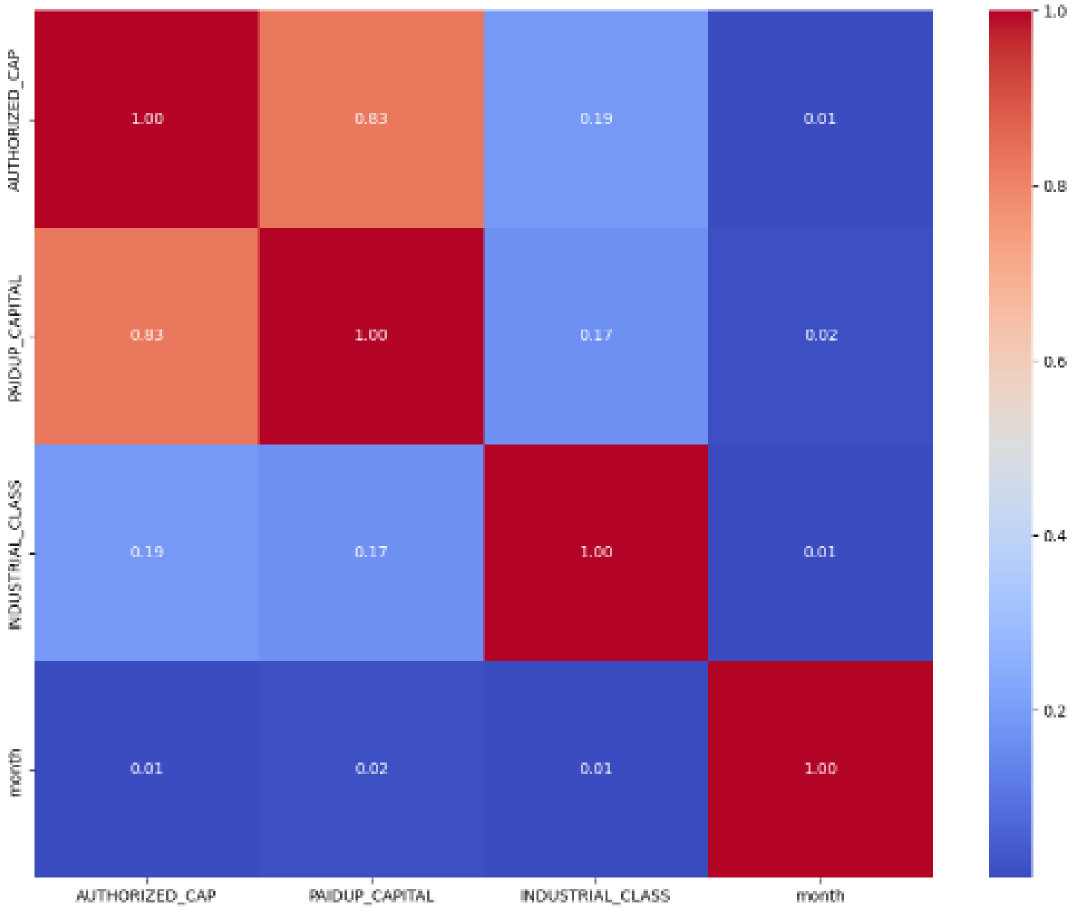
```python
df.describe()
```

	AUTHORIZED_CAP	PAIDUP_CAPITAL	INDUSTRIAL_CLASS	month
count	3.855000e+03	3.855000e+03	3545.000000	3817.000000
mean	1.314988e+08	6.410010e+07	9533.436389	6.510086
std	1.504842e+09	9.001206e+08	19209.718669	3.483183
min	0.000000e+00	0.000000e+00	0.000000	1.000000
25%	2.000000e+05	1.000000e+04	1110.000000	3.000000
50%	1.000000e+06	1.000000e+05	1119.000000	6.000000
75%	5.000000e+06	1.128000e+06	1132.000000	10.000000
max	5.363000e+10	4.789458e+10	99999.000000	12.000000

Exploratory Data Analysis:

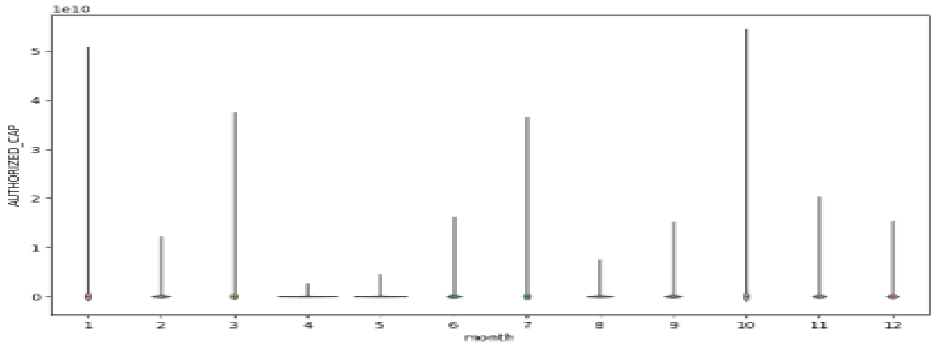
Correlation is **one or more variables are related** to each other. It also helps to find the feature importance and clean the dataset before i start Modeling

```
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")
```




```
# Explore AUTHORIZED_CAP vs month
plt.figure(figsize=(10,6))
sns.violinplot(data=df, x="month", y="AUTHORIZED_CAP",
               split=True, inner="quart", linewidth=1)
```

```
<Axes: xlabel='month', ylabel='AUTHORIZED_CAP'>
```




```python
# Explore INDUSTRIAL_CLASS vs month

plt.figure(figsize=(13,6))
g = sns.kdeplot(df["INDUSTRIAL_CLASS"][df["month"] == 1],
      color="Red", shade = True)
g = sns.kdeplot(df["INDUSTRIAL_CLASS"][df["month"] == 0],
      ax =g, color="Green", shade= True)
g.set_xlabel("INDUSTRIAL_CLASS")
g.set_ylabel("Frequency")
```

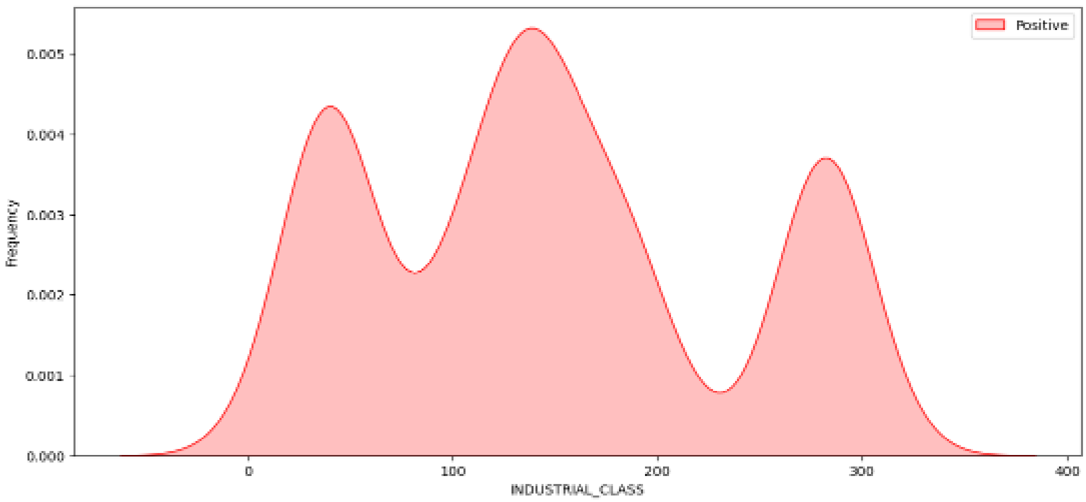
```
<ipython-input-42-98c727c289db>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["INDUSTRIAL_CLASS"][df["month"] == 1],
<ipython-input-42-98c727c289db>:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  g = sns.kdeplot(df["INDUSTRIAL_CLASS"][df["month"] == 0],
<matplotlib.legend.Legend at 0x7a286f08bc10>
```



Feature Engineering

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning.

Outlier Detection:

```
# feature engineering (detect outliers)
```

```
def detect_outliers(df,n,features):
    outlier_indices = []
    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col],75)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col]
> Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of
outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if
v > n )

    return multiple_outliers

# detect outliers from numeric features
outliers_to_drop = detect_outliers(df, 2
,['COMPANY_CLASS','AUTHORIZED_CAP',
'PAIDUP_CAPITAL','INDUSTRIAL_CLASS','month' ])
```

```
#remove outliers
df.drop(df.loc[outliers_to_drop].index, inplace=True)
```

Modeling

In this sections, i tried different models and compare the accuracy for each. Then, i performed Hyperparameter Tuning on Models that has high accuracy.

Before i split the dataset i need to transform the data into quantile using `sklearn.preprocessing`.

```
#  model evaluation
x = df[['year']]
y = df['COMPANY_CLASS']
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
```

```
model= RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(x_train, y_train)
```

RandomForestRegressor

```
RandomForestRegressor(random_state=42)
```

```
#prediction model
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 294077005.65764993
```

```
#future trends
future_years = pd.DataFrame({'year':[2023, 2024, 2025,2026]})
future_registrations = model.predict(future_years)
```

```
print(f"prediction registrations for 2023: {future_registrations[0]}")
print(f"prediction registrations for 2024: {future_registrations[1]}")
print(f"prediction registrations for 2025: {future_registrations[2]}")
```

```
prediction registrations for 2023: 1105.5193812083571

prediction registrations for 2024: 1105.5193812083571

prediction registrations for 2025: 1105.5193812083571
```

Hyperparameter Tuning:

```
def analyze_grid_result(grid_result):

    print("Tuned hyperparameters: (best parameters) ",
grid_result.best_params_)
    print("Accuracy :", grid_result.best_score_)

    means = grid_result.cv_results_["mean_test_score"]
    stds = grid_result.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds,
grid_result.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()
    print("Detailed classification report:")
    y_true, y_pred = y_test, grid_result.predict(x_test)
    print(classification_report(y_true, y_pred))
    print()
```

```
#logistic regression
model = LogisticRegression(solver='liblinear')
solvers = ['newton-cg', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# Define grid search
grid = dict(solver = solvers, penalty = penalty, C = c_values)
cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv =
cv, scoring = 'accuracy', error_score = 0)
logi_result = grid_search.fit(x_train, y_train)
# Logistic Regression Hyperparameter Result
analyze_grid_result(logi_result)
```

```python
# Test predictions
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
                            precision    recall  f1-score   support

                   Private       0.82      0.94      0.88       596
Private(One Person Company)      0.00      0.00      0.00         2
                    Public       0.59      0.28      0.38       173

                  accuracy                           0.79       771
                 macro avg       0.47      0.41      0.42       771
              weighted avg       0.76      0.79      0.76       771
```