******

*Phase-2 Documentation Submission*

**Register number: 963521104023**

*Name:* ***Jefrin babish******J H***

**PROJECT TITLE:**

**Project:6**

**AI-Driven Exploration and Prediction of Company Registration Trends with Registrar of Companies (RoC)**

******

*AI-Driven Exploration and Prediction of Company Registration Trends with Registrar of Companies (ROC)*

**Introduction:**

In an era characterized by rapidly evolving technologies and ever-changing business landscapes, staying ahead of the curve is essential for both government authorities and businesses alike. The Registrar of Companies (RoC), a pivotal regulatory body responsible for overseeing company registrations, plays a crucial role in this dynamic environment. To enhance their efficiency and gain deeper insights into the trends and patterns of company registrations, the integration of Artificial Intelligence (AI) has emerged as a transformative solution.

**Advance the regression techniques:**

To advance the regression techniques in the provided code, you can explore various strategies to improve the model's performance, interpretability, and robustness. Here are some advanced regression techniques and modifications you can consider:

**Regression Analysis:**

Choose the appropriate regression model(s) based on the nature of your data and the relationships you want to explore. Common regression techniques for time series data include:

**Linear Regression**

**Time Series Analysis** (e.g., ARIMA or SARIMA models)

Machine Learning Regressors (e.g., Random Forest Regression, Gradient Boosting)

Deep Learning models (e.g., LSTM or GRU networks for time series forecasting)

**Feature Engineering:**

Perform feature selection to identify and use the most relevant features for your regression models.

Create new features or transformations of existing features to capture more complex relationships in the data.

**Data Preprocessing:**

Handle missing data: Impute missing values or consider techniques like multiple imputation.

Scale or normalize features, especially for algorithms like Logistic Regression.

**Hyperparameter Tuning:**

Optimize the hyperparameters of your models using techniques like grid search or randomized search.

For Random Forest and Decision Tree models, tune parameters like the maximum depth of the tree, minimum samples per leaf, or the number of estimators.

**Cross-Validation:**

Use k-fold cross-validation to get a better estimate of model performance and reduce the risk of overfitting.

**Model Evaluation:**

Use evaluation metrics appropriate for regression tasks, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared (R2).

Visualize model performance using tools like learning curves, residual plots, and feature importance plots.

**Regularization:**

Apply regularization techniques like L1 (Lasso) or L2 (Ridge) regularization to prevent overfitting in Logistic Regression.

Try ensemble methods that combine multiple models, such as AdaBoost or Gradient Boosting.

**Handling Class Imbalance:**

If you encounter class imbalance issues, consider techniques like oversampling, undersampling, or using class-weighted models.

**Interpretability:**

Utilize model interpretation techniques like SHAP values, Partial Dependency Plots, or feature importance analysis to understand how your models are making predictions.

**Advanced Algorithms:**

Explore more advanced regression algorithms like Support Vector Regression (SVR), Bayesian Regression, or Neural Networks

**Visualizations and Reporting**:

Create visualizations and reports to communicate your findings to stakeholders, policymakers, and business leaders.

**Applications of AI-driven company registration trend prediction systems**

AI-driven company registration trend prediction systems can be used in a variety of ways, including:

**Business planning**: Businesses can use AI-driven predictions to inform their business planning decisions, such as where to expand their operations and which products or services to offer.

**Investment decisions**: Investors can use AI-driven predictions to identify investment opportunities in emerging industries and high-growth sectors.

**Policymaking**: Government policymakers can use AI-driven predictions to develop policies that support economic growth and job creation.

**step-by-step guide on how to approach this task:**

**DATASET:**

**Dataset Link:**[**https://tn.data.gov.in/resource/company-master-data-tamil-nadu-upto-28th-february-2019**](https://tn.data.gov.in/resource/company-master-data-tamil-nadu-upto-28th-february-2019)

**Data Collection:**

Obtain access to RoC data, which typically includes information on newly registered companies, such as company names, registration dates, types of businesses, and locations. You may need to request this data from the RoC or obtain it from publicly available sources.

**Data Preprocessing:**

Clean and preprocess the data to handle missing values, duplicates, and inconsistencies. Ensure that the data is in a structured format suitable for analysis.

**Feature Engineering:**

Create relevant features from the raw data that can help in trend analysis. This may include categorizing companies by industry, region, size, or other relevant attributes.

**Exploratory Data Analysis (EDA):**

Use EDA techniques to gain insights into the data. Visualize trends, patterns, and correlations within the registration data. Identify any outliers or anomalies.

**Time Series Analysis:**

Since registration data is likely to be time-stamped, perform time series analysis to understand seasonality, trends, and cyclical patterns in the company registration data.

**AI Model Selection:**

Choose appropriate AI and machine learning models for prediction. Time series forecasting models like ARIMA, SARIMA, or deep learning models like LSTM can be useful for predicting future registration trends.

**Training the AI Model:**

Split the data into training and validation sets. Train the selected model(s) on historical registration data, using appropriate evaluation metrics to assess performance.

**Predicting Registration Trends:**

Use the trained model to make predictions for future company registrations. These predictions can include the number of new registrations expected in a given time period or predictions for specific regions or industries.

**Interpretability and Visualization**:

Make the predictions and insights from the AI model interpretable and actionable through visualizations and clear explanations. Create dashboards or reports that provide real-time updates on registration trends.

**Monitoring and Feedback Loop**:

Continuously monitor the accuracy of your predictions and update your model as new data becomes available. AI models need to adapt to changing trends and external factors.

**Alerts and Notifications:**

Implement alerts or notifications for significant deviations from predicted registration trends. This can help stakeholders react promptly to unexpected changes.

**Policy and Decision Support:**

Share the insights and predictions with relevant stakeholders, such as government agencies, investors, or business analysts, to support informed decision-making and policy formulation.

**Ethical Considerations:**

Ensure that data privacy and ethical considerations are addressed throughout the process, especially when working with sensitive company registration data.

**Security and Compliance:**

Maintain the security and compliance of the data throughout the analysis and prediction process, especially if the data contains sensitive information.

**Feedback and Improvement**:

Gather feedback from users and stakeholders to improve the accuracy and usefulness of the AI-driven prediction system continually.

AI-driven exploration and prediction of company registration trends with RoC data can be a valuable tool for understanding economic activity, making investment decisions, and formulating policies to support businesses. However, it's essential to approach this task responsibly and ethically, respecting data privacy and security considerations.

**Python code for different models for the given dataset:**

**import pandas as pd**

**from sklearn.model\_selection** import **train\_test\_split from sklearn.linear\_model import Logistic Regression** from **sklearn.ensemble import RandomForestClassifier** from sklearn.tree **import DecisionTreeClassifier**

**# Load the dataset**

df **= pd.read\_csv('**Company **Master Data** of **Tamil Nadu** upto **28th February 2019.csv**'**)**

# **Split the dataset** into **train** and test **sets**

**X\_train**, X\_test, **y\_train,** y\_test = **train\_test\_split(**df, **df['Company** Status'], **test\_size**=0.25**, random\_state**=42)

# **Create different models**

# Logistic **Regression Ir** = Logistic Regression() **Ir.fit**(**X\_train, y\_train)**

# **Random Forest Classifier** rf = **RandomForestClassifier**() rf.fit(**X\_train, y\_train)**

# **Decision Tree Classifier dt** = **DecisionTreeClassifier**() **dt.fit**(**X\_train**, **y\_train)**

**# Evaluate the models on the test set**

# **Logistic Regression**

**Ir\_score** = lr.score(X\_test, **y\_test**)

# **Random Forest Classifier**

rf\_score = rf.score(X\_test, **y\_test**)

# **Decision Tree Classifier**

**dt\_score = dt.score(X\_test, y\_test**)

# **Print the model scores**

**print(**'Logistic **Regression score:**'**,** lr\_score) **print('Random Forest** Classifier score**:**', **rf\_score**) **print**('**Decision Tree Classifier score:**', dt\_score)

**Other models that can be used**:

• Support Vector **Machines** (SVM**)**

• **Naive Bayes Classifier**

• Gradient Boosting **Machines (**GBM)

**XGBoost**

You **can choose the model that** best **suits** your needs based **on the** performance on the test set and the interpretability of the model.

Python **code** for creating different **models** for **the** given **dataset**, **including** Support Vector Machines (**SVM),** Naive Bayes Classifier, Gradient

Boosting **Machines (GBM),** and **XGBoost**:

**Python**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from** sklearn.svm import SVC

**from sklearn.naive\_bayes import GaussianNB**

**from sklearn.ensemble import GradientBoostingClassifier import xgboost as xgb**

# **Load the dataset**

df = **pd.read\_csv('Company Master** Data of **Tamil Nadu upto 28th February** 2019.csv'**)**

# **Split the dataset into train and test sets**

**X\_train, X\_test, y\_train**, **y\_test** = **train\_test\_split(df**, df**[**'**Company** Status'], **test\_size=**0.25**, random\_state**=42)

# **Create different models** # **Support** Vector **Machines svm =** SVC(

svm.fit(**X\_train, y\_train)**

# **Naive Bayes Classifier nb** = **GaussianNB() nb.fit(X\_train**, **y\_train)**

**# Gradient Boosting Machines** gbm= **GradientBoostingClassifier**() gbm.fit(**X\_train, y\_train)**

# XGBoost

xgb\_clf = **xgb.XGBClassifier**() **xgb\_clf.fit**(X\_train, **y\_train**)

# **Evaluate the models on the test set** # **Support** Vector **Machines**

**svm\_score** = **svm.score**(**X\_test, y\_test)**

**#** Naive **Bayes Classifier**

**nb\_score = nb.score(**X\_test**, y\_test)**

# **Gradient Boosting Machines**

**gbm\_score** = **gbm.score(**X\_test, **y\_test)**

# XGBoost

**xgb\_score** = **xgb\_clf.score**(X\_test, y\_test**)**

**# Print the model scores**

**print('Support** Vector **Machines score:**'**, svm\_score) print("Naive Bayes Classifier score:**', **nb\_score) print(**'**Gradient** Boosting **Machines score**:**'**, gbm\_score) **print('XGBoost** score:**'**, xgb\_score)

**This code will** create four different models for the given dataset: **Support** Vector **Machines (**SVM**), Naive** Bayes Classifier, Gradient **Boosting** Machines **(**GBM**),** and XGBoost. **The models will** be **evaluated on the** test **set** and **the scores will be printed to the console**.

**Which model** to **choose**?

The best model to choose will depend on the specific dataset and **the desired outcome**. **However**, **some general guidelines can be** followed:

• **SVM:** SVM is a good choice for **datasets with** high**-dimensional features and** a **small number of samples. It is also known** for **its** good **performance on** imbalanced **datasets**.

• **Naive Bayes Classifier**: **Naive Bayes** Classifier **is** a good choice for datasets with **categorical features. It is also** a **simple** and efficient **model to train**.

• Gradient Boosting **Machines**: **GBM** is **a** good choice for **datasets**

**with complex relationships** between **the** features and the target **variable. It is also** a **relatively** robust **model to outliers**.

• XGBoost**:** XGBoost **is a** powerful ensemble learning algorithm that **is** often used for **classification** and regression **tasks.** It **is known for its scalability** and accuracy.

It **is** generally recommended to **try out multiple models on the dataset** to **see which** one performs **best**. The **code above provides** a starting **point** for creating **and evaluating** different **models**.

**Sources**

1.https://**towards datascience.com/evaluating-machine-learning-classification-p**

**roblems-**in-**python**-**5-1-metrics-that-matter-792c6faddf5**

**2.https**:**//www.kaggle.com/fearless2611/innovac cer-data-science-**intern**-assess**

ment

Prerequisites

In[1] : import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

*#Importing the dataset*

In[2] : data = pd.read\_csv('/kaggle/input/all-indian-companies-registration -data-1900-2019/registered\_companies.csv')

/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3166: DtypeWarning: Columns (10) have mixed types.Specify dtype option on import or set low\_memory=False.

interactivity=interactivity, compiler=compiler, result=result)

In[3] : df = data

**Working the data**

In [4]:

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1992170 entries, 0 to 1992169

Data columns (total 17 columns):

# Column Dtype

--- ------ -----

0 CORPORATE\_IDENTIFICATION\_NUMBER object

1 COMPANY\_NAME object

2 COMPANY\_STATUS object

3 COMPANY\_CLASS object

4 COMPANY\_CATEGORY object

5 COMPANY\_SUB\_CATEGORY object

6 DATE\_OF\_REGISTRATION object

7 REGISTERED\_STATE object

8 AUTHORIZED\_CAP float64

9 PAIDUP\_CAPITAL float64

10 INDUSTRIAL\_CLASS object

11 PRINCIPAL\_BUSINESS\_ACTIVITY\_AS\_PER\_CIN object

12 REGISTERED\_OFFICE\_ADDRESS object

13 REGISTRAR\_OF\_COMPANIES object

14 EMAIL\_ADDR object

15 LATEST\_YEAR\_ANNUAL\_RETURN object

16 LATEST\_YEAR\_FINANCIAL\_STATEMENT object

dtypes: float64(2), object(15)

memory usage: 258.4+ MB

-**There are a lot of categoricals to work with \ -Many columns seem broadly empty \**

In [5]: print(f"Total Values : **{**len(df)**}\n**")

for x **in** df.columns:

print(f'**{**len(df)-df[x].count()**}** values missing in **{**x**}**')

**Total Values :** 1992170

0 values missing in CORPORATE\_IDENTIFICATION\_NUMBER

0 values missing in COMPANY\_NAME

0 values missing in COMPANY\_STATUS

5078 values missing in COMPANY\_CLASS

5085 values missing in COMPANY\_CATEGORY

5090 values missing in COMPANY\_SUB\_CATEGORY

2525 values missing in DATE\_OF\_REGISTRATION

0 values missing in REGISTERED\_STATE

0 values missing in AUTHORIZED\_CAP

0 values missing in PAIDUP\_CAPITAL

4811 values missing in INDUSTRIAL\_CLASS

12 values missing in PRINCIPAL\_BUSINESS\_ACTIVITY\_AS\_PER\_CIN

15259 values missing in REGISTERED\_OFFICE\_ADDRESS

42198 values missing in REGISTRAR\_OF\_COMPANIES

370208 values missing in EMAIL\_ADDR

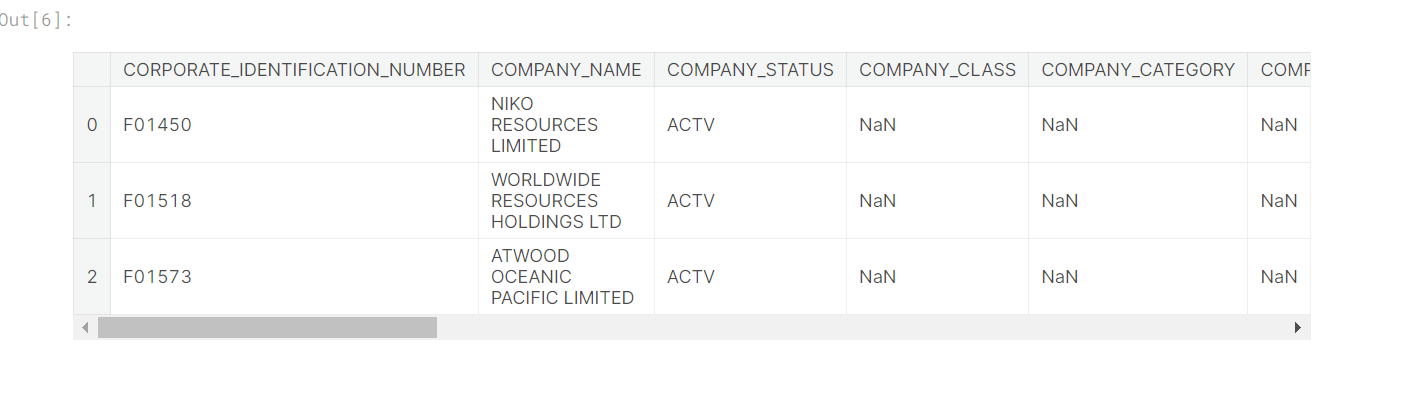
831317 values missing in LATEST\_YEAR\_ANNUAL\_RETURN

828829 values missing in LATEST\_YEAR\_FINANCIAL\_STATEMENT

In [6]:print(len(df))

df.head(3)

1992170

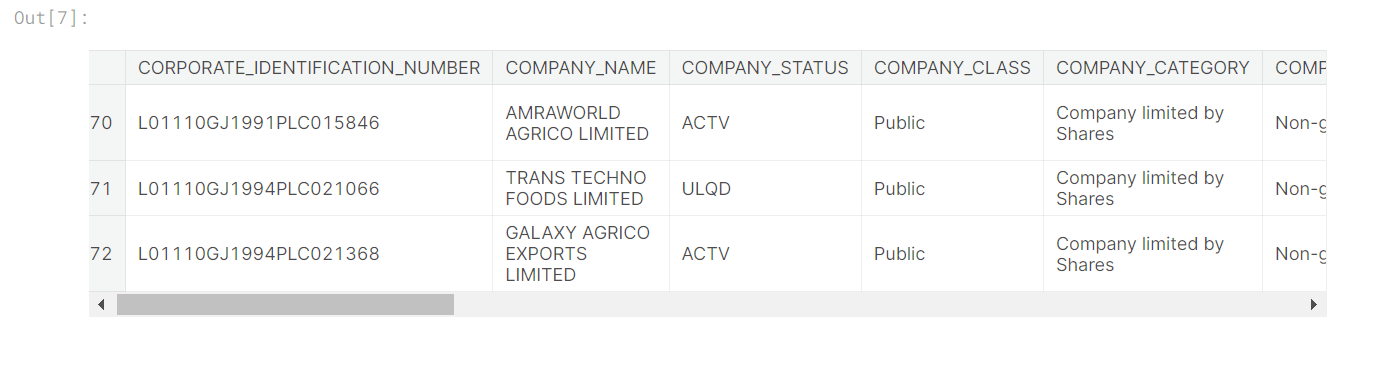


In [7]:df = df.dropna()

print(len(df))

df.head(3)

1124485



In [8]: *#Interesting data, will keep it narrow and efficient for now*

dropCols = ["LATEST\_YEAR\_FINANCIAL\_STATEMENT",

"EMAIL\_ADDR", "COMPANY\_NAME",

"LATEST\_YEAR\_ANNUAL\_RETURN",

"CORPORATE\_IDENTIFICATION\_NUMBER",

"REGISTERED\_OFFICE\_ADDRESS"]

df = df.drop(dropCols, axis=1)

In [9]: df["DATE\_OF\_REGISTRATION"] = df["DATE\_OF\_REGISTRATION"].apply(pd.to\_datetime)

*#df["INDUSTRIAL\_CLASS"] = df["INDUSTRIAL\_CLASS"].astype(int)*

In [10]: df['REG\_YEAR'] = df['DATE\_OF\_REGISTRATION'].dt.year

df['REG\_MONTH'] = df['DATE\_OF\_REGISTRATION'].dt.month

In [11]: *#columns along the number of unique items in them along a list of it*

for x **in** df.columns:

print(f'**{**x**}** : **{**len(df[x].unique())**}\n{**df[x].unique()[:20]**}\n**')

COMPANY\_STATUS : 12

['ACTV' 'ULQD' 'AMAL' 'DISD' 'CLLD' 'UPSO' 'STOF' 'CLLP' 'D455' 'NAEF'

'LIQD' 'DRMT']

COMPANY\_CLASS : 3

['Public' 'Private' 'Private(One Person Company)']

COMPANY\_CATEGORY : 3

['Company limited by Shares' 'Company Limited by Guarantee'

'Unlimited Company']

COMPANY\_SUB\_CATEGORY : 5

['Non-govt company' 'State Govt company' 'Subsidiary of Foreign Company'

'Guarantee and Association comp' 'Union Govt company']

DATE\_OF\_REGISTRATION : 21228

['1991-06-21T00:00:00.000000000' '1994-01-17T00:00:00.000000000'

'1994-02-23T00:00:00.000000000' '1996-04-15T00:00:00.000000000'

'2011-12-27T00:00:00.000000000' '2011-09-14T00:00:00.000000000'

'1994-10-19T00:00:00.000000000' '1994-01-24T00:00:00.000000000'

'2004-04-02T00:00:00.000000000' '1990-09-26T00:00:00.000000000'

'2014-07-28T00:00:00.000000000' '1995-07-03T00:00:00.000000000'

'1980-06-26T00:00:00.000000000' '1996-08-27T00:00:00.000000000'

'2008-08-29T00:00:00.000000000' '2009-06-03T00:00:00.000000000'

'1994-12-16T00:00:00.000000000' '1994-09-11T00:00:00.000000000'

'1996-11-06T00:00:00.000000000' '2005-10-24T00:00:00.000000000']

REGISTERED\_STATE : 36

['Gujarat' 'Karnataka' 'Rajasthan' 'Madhya Pradesh' 'Uttaranchal' 'Assam'

'Jharkhand' 'Tamil Nadu' 'Delhi' 'Maharashtra' 'Haryana' 'Chattisgarh'

'Daman and Diu' 'West Bengal' 'Lakshadweep' 'Himachal Pradesh'

'Dadra and Nagra Haveli' 'Kerala' 'Pondicherry' 'Jammu and Kashmir']

AUTHORIZED\_CAP : 8463

[1.3e+08 2.2e+08 5.5e+07 6.0e+07 3.2e+08 1.7e+08 8.0e+07 7.7e+08 6.1e+07

5.0e+07 1.0e+08 7.5e+07 2.5e+08 7.0e+07 5.0e+09 1.5e+11 1.5e+08 1.5e+09

2.0e+07 2.6e+08]

PAIDUP\_CAPITAL : 134551

[1.20300000e+08 2.11200000e+08 2.73162000e+07 4.93628000e+07

3.00732620e+08 1.09801580e+08 6.10207000e+07 2.49275094e+08

6.06799000e+07 4.83000000e+07 7.87029600e+07 5.33120000e+07

4.77845600e+07 2.35266220e+08 2.14200000e+08 1.26066000e+08

1.00000000e+08 3.93610000e+07 1.32259300e+09 4.49998688e+10]

INDUSTRIAL\_CLASS : 6828

[1110.0 1111.0 1112.0 1119.0 1122.0 1130.0 1132.0 1135.0 1200.0 1403.0

1405.0 5004.0 10102.0 10300.0 11100.0 11711.0 14100.0 15122.0 15140.0

15142.0]

PRINCIPAL\_BUSINESS\_ACTIVITY\_AS\_PER\_CIN : 17

['Agriculture & allied' 'Mining and quarrying' 'Manufacturing'

'Electricity gas and water supply' 'Construction'

'Wholesale and retail trade repair of motor vehicles motorcycles and personal and household goods'

'Unclassified' 'Hotels and restaurants'

'Transport storage and communications' 'Financial intermediation'

'Real estate renting and business activities' 'Education'

'Health and social work'

'Other community social and personal service activities'

'Extraterritorial organizations and bodies'

'Activities of private households as employers and undifferentiated production activities of private households'

'Public administration and defence compulsory social security']

REGISTRAR\_OF\_COMPANIES : 25

['ROC\xa0AHMEDABAD' 'ROC\xa0GOA' 'ROC\xa0BANGALORE' 'ROC\xa0JAIPUR'

'ROC\xa0KOLKATA' 'ROC\xa0GWALIOR' 'ROC\xa0UTTARAKHAND' 'ROC\xa0KANPUR'

'ROC\xa0SHILLONG' 'ROC\xa0JHARKHAND' 'ROC\xa0PATNA' 'ROC\xa0COIMBATORE'

'ROC\xa0CHENNAI' 'ROC\xa0HYDERABAD' 'ROC\xa0DELHI' 'ROC\xa0MUMBAI'

'ROC\xa0PUNE' 'ROC\xa0ERNAKULAM' 'ROC\xa0CHHATTISGARH' 'ROC\xa0HP']

REG\_YEAR : 148

[1991 1994 1996 2011 2004 1990 2014 1995 1980 2008 2009 2005 1984 1963

1979 1992 1993 1983 1971 2013]

REG\_MONTH : 12

[ 6 1 2 4 12 9 10 7 8 11 5 3]

In [12]:

*#Naming conveniences*

df["COMPANY\_CLASS"] = df["COMPANY\_CLASS"].apply(lambda x:"Solo" if x == 'Private(One Person Company)' else x)

print(df["COMPANY\_CLASS"].unique())

df["REGISTRAR"] = df["REGISTRAR\_OF\_COMPANIES"].apply(lambda x:x.split("ROC**\xa0**")[-1])

print(df["REGISTRAR"].unique())

df["PRINCIPAL\_BUSINESS"] = df["PRINCIPAL\_BUSINESS\_ACTIVITY\_AS\_PER\_CIN"].apply(lambda x:x.split(" ")[0])

print(df["PRINCIPAL\_BUSINESS"].unique())

*#Creating bins for REGISTRATION\_YEAR*

df["REG\_YEAR\_5BIN"] = df["REG\_YEAR"].apply(lambda x:(round(x/5))\*5)

print(df["REG\_YEAR\_5BIN"].unique())

df["REG\_YEAR\_10BIN"] = df["REG\_YEAR"].apply(lambda x:(round(x/10))\*10)

print(df["REG\_YEAR\_10BIN"].unique())

df["REG\_YEAR\_20BIN"] = df["REG\_YEAR"].apply(lambda x:(round(x/20))\*20)

print(df["REG\_YEAR\_20BIN"].unique())

['Public' 'Private' 'Solo']

['AHMEDABAD' 'GOA' 'BANGALORE' 'JAIPUR' 'KOLKATA' 'GWALIOR' 'UTTARAKHAND'

'KANPUR' 'SHILLONG' 'JHARKHAND' 'PATNA' 'COIMBATORE' 'CHENNAI'

'HYDERABAD' 'DELHI' 'MUMBAI' 'PUNE' 'ERNAKULAM' 'CHHATTISGARH' 'HP'

'PONDICHERRY' 'JAMMU' 'CUTTAK' 'CHANDIGARH' 'ANDAMAN']

['Agriculture' 'Mining' 'Manufacturing' 'Electricity' 'Construction'

'Wholesale' 'Unclassified' 'Hotels' 'Transport' 'Financial' 'Real'

'Education' 'Health' 'Other' 'Extraterritorial' 'Activities' 'Public']

[1990 1995 2010 2005 2015 1980 1985 1965 1970 1955 1915 1920 1930 1935

1950 1960 1975 2000 1945 1940 2020 1905 1910 1890 1895 1900 1925 1880

1875 1885 1870 1865]

[1990 2000 2010 1980 1960 1970 1910 1920 1930 1940 1950 2020 1900 1890

1880 1870 1860]

[2000 2020 1980 1960 1920 1940 1900 1880 1860]

Visualization

In [13]:

*#Working with a smaller randomly picked sample space for efficiency and overall population testing.*

df2 = df

df = df.sample(n=10000)

In [14]:

*#For readability since I use dark mode*

sns.set\_theme(context='notebook',

style='darkgrid',

palette='magma',

font='sans-serif',

font\_scale=0.6,

color\_codes=True,

rc=None)

In [15]:

f, ax = plt.subplots(2)

*#Counting all the number fo companies by REG\_YEAR*

sns.countplot(x="REG\_YEAR\_5BIN",

data=df, ax = ax[0])

*#Year of registration by COMPANY\_CLASS*

sns.stripplot(x="REG\_YEAR",

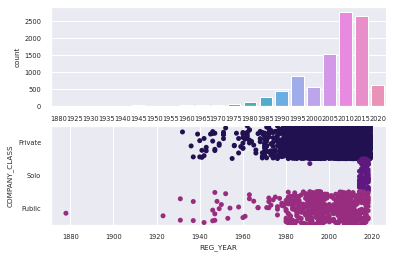
y="COMPANY\_CLASS",

data=df, jitter=0.5,

ax = ax[1])

Out[15]:

<AxesSubplot:xlabel='REG\_YEAR', ylabel='COMPANY\_CLASS'>



In [16]:

f, ax = plt.subplots(1, len(df["COMPANY\_CLASS"].unique()))

f.tight\_layout()

y=0

print(df["COMPANY\_CLASS"].unique())

for x **in** df["COMPANY\_CLASS"].unique():

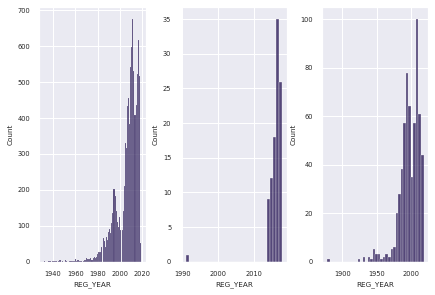
sns.histplot(x="REG\_YEAR",

data=df[df["COMPANY\_CLASS"]==x],

ax=ax[y])

y+=1

['Private' 'Solo' 'Public']



India is seeing a increasing number of new companies being registered which a vast proportion of them being in the 2000s+ \ Solo COMPANY\_CLASS catches traction post 2010+ from the first lower plot and can see appearing from 2014 in this sample space on the second rightmost plot.\ The majority of companies are classified as Private (density of the first lower plot & count in second plot).

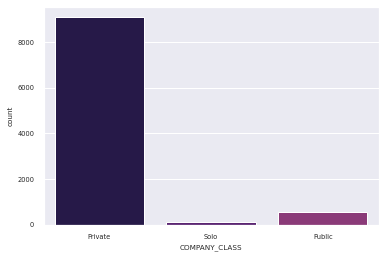
In [17]:

sns.countplot(x="COMPANY\_CLASS",

data=df[df["REG\_YEAR"] >= 1982]) *#Change year*

Out[17]:

<AxesSubplot:xlabel='COMPANY\_CLASS', ylabel='count'>



In [18]:

*#Industry of companies*

sns.stripplot(x="REG\_YEAR",

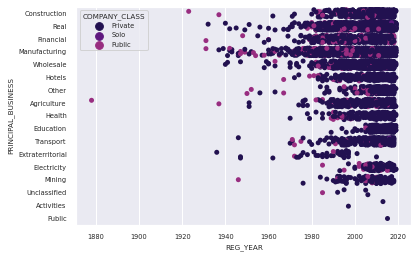
y="PRINCIPAL\_BUSINESS",

hue="COMPANY\_CLASS",

data=df, jitter=0.3)

Out[18]:

<AxesSubplot:xlabel='REG\_YEAR', ylabel='PRINCIPAL\_BUSINESS'>



The density of private companies is too high to make any reasonable inference.

In [19]:

*#Companies in respect to COMPANY\_CLASS over time.*

y=0

f, ax = plt.subplots(len(df["COMPANY\_CLASS"].unique()),1)

f.subplots\_adjust(top=1, bottom=-0.9, left=-0.9, hspace=0.2)

print(df["COMPANY\_CLASS"].unique())

for x **in** df["COMPANY\_CLASS"].unique():

sns.stripplot(y="REG\_YEAR",

x="PRINCIPAL\_BUSINESS",

data=df[df["COMPANY\_CLASS"]==x],

hue="REG\_YEAR\_10BIN",

palette="magma\_r",

jitter=0.4,

ax=ax[y])

y+=1

['Private' 'Solo' 'Public']



-Wholesale, Manufacturing and Real Estate have the highest number private companies.\ -Public companies incline towards manufacturing followed by Real Estate and Financial sectors.\ -Interesting how private companies have a big concentration in Real Estate even in the sparcity.\ -Companies under Education and electricity industry started late in India and are relevant magnifications to look at.

In [20]:

*#State wise looking at the industries through time*

y=0; f, ax = plt.subplots(len(df["REGISTERED\_STATE"].unique()),1)

f.subplots\_adjust(top=10, bottom=-0.9, left=-0.5, hspace=0.2)

for x **in** df["REGISTERED\_STATE"].unique():

sns.histplot(y='REG\_YEAR',

x='PRINCIPAL\_BUSINESS',

*#jitter=0.3,*

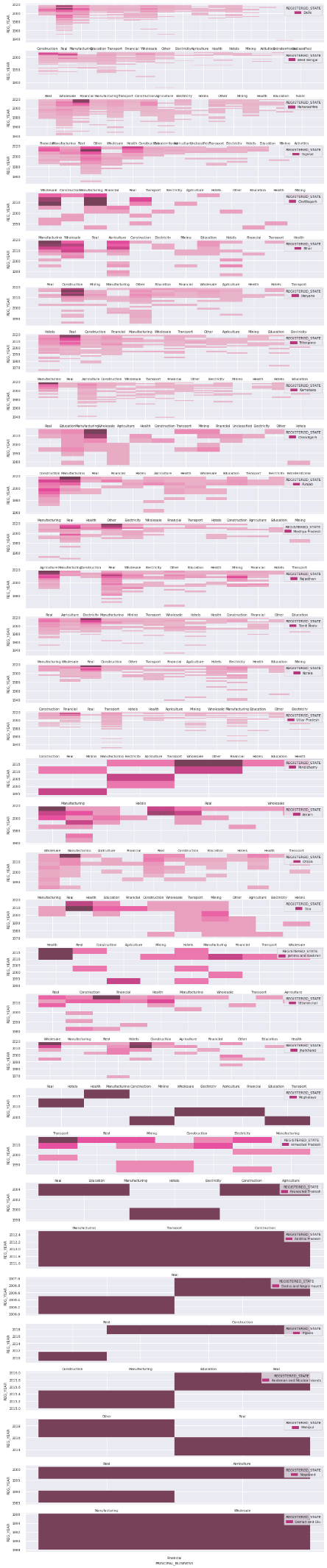
hue="REGISTERED\_STATE",

palette="magma\_r",

data= df[df["REGISTERED\_STATE"]==x],

ax=ax[y])

y+=1



Education industry directly impacts all sectors?

In [21]:

len(df)

Out[21]:

10000

In [22]:

df[df["PRINCIPAL\_BUSINESS"] == "Education"].head(3)

Out[22]:

|  | COMPANY\_STATUS | COMPANY\_CLASS | COMPANY\_CATEGORY | COMPANY\_SUB\_CATEGORY | DATE\_OF\_REGISTRATION | REGISTERED\_STATE | AUTHORIZED\_CAP | PAIDUP\_CAPITAL | INDUSTRIAL\_CLASS | PRINCIPAL\_BUSINESS\_ACTIVITY\_AS\_PER\_CIN | REGISTRAR\_OF\_COMPANIES | REG\_YEAR | REG\_MONTH | REGISTRAR | PRINCIPAL\_BUSINESS | REG\_YEAR\_5BIN | REG\_YEAR\_10BIN | REG\_YEAR\_20BIN |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 180191 | ACTV | Private |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**CONCLUSION:**

In essence, the "AI-Driven Exploration and Prediction of Company Registration Trends with Registrar of Companies (RoC)" module serves as a powerful tool for enhancing business strategies, regulatory compliance, and investment opportunities in an ever-evolving corporate landscape. Its ability to analyze and predict registration trends positions it as an invaluable asset in the field of business intelligence and management.