Problem 1

import libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
        from datetime import datetime, timedelta
```

loading the data

```
In [2]: df=pd.read_parquet(r"C:\Users\Barry\Desktop\projects\akaike assignment\Structured_Data_A
```

```
In [3]: df.head()
```

Out[3]:

	Patient-Uid	Date	Incident
0	a0db1e73-1c7c-11ec-ae39-16262ee38c7f	2019-03-09	PRIMARY_DIAGNOSIS
1	a0dc93f2-1c7c-11ec-9cd2-16262ee38c7f	2015-05-16	PRIMARY_DIAGNOSIS
3	a0dc94c6-1c7c-11ec-a3a0-16262ee38c7f	2018-01-30	SYMPTOM_TYPE_0
4	a0dc950b-1c7c-11ec-b6ec-16262ee38c7f	2015-04-22	DRUG_TYPE_0
8	a0dc9543-1c7c-11ec-bb63-16262ee38c7f	2016-06-18	DRUG_TYPE_1

```
In [4]: df.shape

Out[4]: (3220868, 3)
```

```
In [5]: # checking for null values
        df.isnull().sum()

Out[5]: Patient-Uid    0
        Date           0
        Incident       0
        dtype: int64
```

```
In [6]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        Int64Index: 3220868 entries, 0 to 29080911
        Data columns (total 3 columns):
         #   Column       Dtype
        ---  ------       -----
         0   Patient-Uid  object
         1   Date         datetime64[ns]
         2   Incident     object
        dtypes: datetime64[ns](1), object(2)
        memory usage: 98.3+ MB
```

```
In [7]: #converting date column to pandas datetime type
        df['Date']=pd.to_datetime(df['Date'],format='%Y-%m-%d')
```

Create the positive set (patients who have taken "Target Drug")

In [8]:

```
positive_set=df[df['Incident'] == 'TARGET DRUG']

# Set the current date as a reference point
current_date = positive_set.Date.max()

# Calculate the cutoff date 30 days before the current date
cutoff_date = current_date - timedelta(days=30)

# Filter the data for patients who have taken "Target Drug" within the last 30 days
positive_set = df[(df['Incident'] == 'TARGET DRUG') & (df['Date'] >= cutoff_date)]
```

In [9]:

```
negative_set = df[df['Incident'] != 'TARGET DRUG'].sample(frac=1)[:len(positive_set)]
negative_set.shape
# this code will have the patients who have not taken TARGET DRUG and length same as pos
# to avoid bias and imbalnce in data
```

Out[9]:

```
(2891, 3)
```

In [10]:

```
# Combine the positive and negative sets
model_data = pd.concat([positive_set, negative_set])

# Sort the data by date
model_data = model_data.sort_values(by='Date')

# Create a target variable indicating whether the patient is eligible or not
model_data['Eligible'] = np.where(model_data['Incident'] == 'TARGET DRUG', 1, 0)
```

In [11]:

```
model_data.reset_index(drop=True)
```

Out[11]:

	Patient-Uid	Date	Incident	Eligible
0	a0e92833-1c7c-11ec-bc7e-16262ee38c7f	2015-04-07	DRUG_TYPE_0	0
1	a0e4492e-1c7c-11ec-bcbd-16262ee38c7f	2015-04-07	DRUG_TYPE_8	0
2	a0df8c90-1c7c-11ec-b306-16262ee38c7f	2015-04-09	DRUG_TYPE_1	0
3	a0eba357-1c7c-11ec-9694-16262ee38c7f	2015-04-10	DRUG_TYPE_1	0
4	a0e36140-1c7c-11ec-85b1-16262ee38c7f	2015-04-10	DRUG_TYPE_2	0
...	...	...	...	...
5777	a0edef0d-1c7c-11ec-9dec-16262ee38c7f	2020-09-03	TARGET DRUG	1
5778	a0f0b5fd-1c7c-11ec-8f5d-16262ee38c7f	2020-09-03	TARGET DRUG	1
5779	a0ec6b35-1c7c-11ec-9be6-16262ee38c7f	2020-09-03	TARGET DRUG	1
5780	a0edb933-1c7c-11ec-bee6-16262ee38c7f	2020-09-03	TARGET DRUG	1
5781	a0edd782-1c7c-11ec-bdcb-16262ee38c7f	2020-09-03	TARGET DRUG	1

5782 rows × 4 columns

In [12]:

```
#B. Feature engineering:

# Create frequency-based features
freq_features = model_data.groupby('Patient-Uid').agg({'Incident': 'count'}).reset_index
freq_features.columns = ['Patient-Uid', 'Freq']
```

In [13]:

```
freq_features.head(3)
```

Out[13]:

	Patient-Uid	Freq

0	a0dc950b-1c7c-11ec-b6ec-16262ee38c7f	1
1	a0dc9543-1c7c-11ec-bb63-16262ee38c7f	1
2	a0dc9c2b-1c7c-11ec-ac1e-16262ee38c7f	1

In [14]:

```
# Create time-based features
time_features = model_data.groupby('Patient-Uid').agg({'Date': ['min', 'max']}).reset_in
time_features.columns = ['Patient-Uid', 'Min_Date', 'Max_Date']
time_features['Time_Diff'] = (pd.to_datetime(current_date) - time_features['Max_Date']).
```

In [15]:

```
time_features.head(3)
```

Out[15]:

	Patient-Uid	Min_Date	Max_Date	Time_Diff
0	a0dc950b-1c7c-11ec-b6ec-16262ee38c7f	2019-05-25	2019-05-25	467
1	a0dc9543-1c7c-11ec-bb63-16262ee38c7f	2016-11-01	2016-11-01	1402
2	a0dc9c2b-1c7c-11ec-ac1e-16262ee38c7f	2019-06-18	2019-06-18	443

In [16]:

```
# Merge the features with the target variable
model_data = pd.merge(model_data, freq_features, on='Patient-Uid', how='left')
model_data = pd.merge(model_data, time_features, on='Patient-Uid', how='left')
```

In [17]:

```
model_data
```

Out[17]:

	Patient-Uid	Date	Incident	Eligible	Freq	Min_Date	Max_Date	Time_Diff
0	a0e92833-1c7c-11ec-bc7e-16262ee38c7f	2015-04-07	DRUG_TYPE_0	0	1	2015-04-07	2015-04-07	1976
1	a0e4492e-1c7c-11ec-bcbd-16262ee38c7f	2015-04-07	DRUG_TYPE_8	0	1	2015-04-07	2015-04-07	1976
2	a0df8c90-1c7c-11ec-b306-16262ee38c7f	2015-04-09	DRUG_TYPE_1	0	1	2015-04-09	2015-04-09	1974
3	a0eba357-1c7c-11ec-9694-16262ee38c7f	2015-04-10	DRUG_TYPE_1	0	2	2015-04-10	2020-08-20	14
4	a0e36140-1c7c-11ec-85b1-16262ee38c7f	2015-04-10	DRUG_TYPE_2	0	1	2015-04-10	2015-04-10	1973
...	...	...	...	...	...	...	...	...
5777	a0edef0d-1c7c-11ec-9dec-16262ee38c7f	2020-09-03	TARGET DRUG	1	1	2020-09-03	2020-09-03	0
5778	a0f0b5fd-1c7c-11ec-8f5d-16262ee38c7f	2020-09-03	TARGET DRUG	1	1	2020-09-03	2020-09-03	0
5779	a0ec6b35-1c7c-11ec-9be6-16262ee38c7f	2020-09-03	TARGET DRUG	1	1	2020-09-03	2020-09-03	0
5780	a0edb933-1c7c-11ec-bee6-16262ee38c7f	2020-09-03	TARGET DRUG	1	1	2020-09-03	2020-09-03	0
5781	a0edd782-1c7c-11ec-bdcb-16262ee38c7f	2020-09-03	TARGET DRUG	1	1	2020-09-03	2020-09-03	0

5782 rows × 8 columns

In [18]:

```
model_data.isnull().sum()
model_data.fillna(0,inplace=True)
```

Data splitting

In [19]:

```
features = ['Freq', 'Time_Diff']
target = 'Eligible'
X=model_data[features]
y=model_data[target]
```

In [20]:

```
X.shape,y.shape
```

Out[20]:

```
((5782, 2), (5782,))
```

In [21]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
```

In [22]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [ ]:

```

```

Model Building

In [23]:

```
#  random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train,y_train)

# Make predictions on the validation set
y_pred = rf.predict(X_test)
f1 = f1_score(y_test,y_pred)

print(f"F1 score: {f1:.3f}")
```

```
F1 score: 0.939
```

In [24]:

```
# logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the validation set
y_pred_lg = model.predict(X_test)
f1=f1_score(y_test,y_pred_lg)

print(f"F1 score: {f1:.3f}")
```

```
F1 score: 0.935
```

In [25]:

```
# Gradient Boosting model
GB=GradientBoostingClassifier()
GB.fit(X_train,y_train)

# Make predictions on the validation set
```

```
y_pred_gb=GB.predict(X_test)
f1=f1_score(y_test,y_pred_gb)
print(f"F1 score: {f1:.3f}")
```

```
F1 score: 0.941
```

In [26]:

```
# KNN model
knn = KNeighborsClassifier(4)
knn.fit(X_train,y_train)

# Make predictions on the validation set
y_pred_knn=knn.predict(X_test)
f1=f1_score(y_test,y_pred_knn)
print(f"F1 score: {f1:.3f}")
```

```
F1 score: 0.913
```

Hyperparameter tuning for Gradient Boosting model

In [27]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, f1_score

# Create the Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(random_state=42)

# Define the hyperparameter grid for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': [3, 5, 7]
}

# Define the evaluation metric
scorer = make_scorer(f1_score)

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=gb_classifier, param_grid=param_grid, scoring=score
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluate the best model on the test set
y_pred = best_model.predict(X_test)
f1 = f1_score(y_test, y_pred)

# Print the best hyperparameters and f1 score
print("Best Hyperparameters:", best_params)
print("F1 Score:", f1)
```

```
Best Hyperparameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
F1 Score: 0.9275603663613656
```

Final model

In [ ]:

```

```

Prediction on Test data

In [27]:

```
df1=pd.read_parquet(r"C:\Users\Barry\Desktop\projects\akaike assignment\Structured_Data_
```

In [49]:
```
test_data=df1[:100000]
```

In [50]:
```
test_data.shape
```

Out[50]:
```
(100000, 3)
```

In [54]:
```
test_data.head()
```

Out[54]:

	Patient-Uid	Date	Incident
0	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2016-12-08	SYMPTOM_TYPE_0
1	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2018-10-17	DRUG_TYPE_0
2	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2017-12-01	DRUG_TYPE_2
3	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2018-12-05	DRUG_TYPE_1
4	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2017-11-04	SYMPTOM_TYPE_0

In [55]:
```
current_date_1=df1['Date'].max()
```

In [56]:
```
# Create the features
test_freq_features = test_data.groupby('Patient-Uid').agg({'Incident': 'count'}).reset_i
test_freq_features.columns = ['Patient-Uid', 'Freq']

test_time_features = test_data.groupby('Patient-Uid').agg({'Date': ['min', 'max']}).rese
test_time_features.columns = ['Patient-Uid', 'Min_Date', 'Max_Date']
test_time_features['Time_Diff'] = (pd.to_datetime('2020-08-04') - test_time_features['Ma

# Merge the features
test_data = pd.merge(test_data, test_freq_features, on='Patient-Uid', how='left')
test_data = pd.merge(test_data, test_time_features, on='Patient-Uid', how='left')

# Fill missing values with 0
test_data.fillna(0, inplace=True)
```

In [57]:
```
test_data
```

Out[57]:

	Patient-Uid	Date	Incident	Freq	Min_Date	Max_Date	Time_Diff
0	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2016-12-08	SYMPTOM_TYPE_0	55	2016-06-23	2019-05-21	441
1	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2018-10-17	DRUG_TYPE_0	55	2016-06-23	2019-05-21	441
2	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2017-12-01	DRUG_TYPE_2	55	2016-06-23	2019-05-21	441
3	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2018-12-05	DRUG_TYPE_1	55	2016-06-23	2019-05-21	441
4	a0f9e8a9-1c7c-11ec-8d25-16262ee38c7f	2017-11-04	SYMPTOM_TYPE_0	55	2016-06-23	2019-05-21	441
...	...	...	...	...	...	...	...
99995	a0faa319-1c7c-11ec-bd56-16262ee38c7f	2016-09-13	DRUG_TYPE_1	21	2015-09-26	2019-07-13	388
99996	a0faa319-1c7c-11ec-bd56-16262ee38c7f	2016-05-06	DRUG_TYPE_8	21	2015-09-26	2019-07-13	388
99997	a0faa319-1c7c-11ec-bd56-16262ee38c7f	2015-09-26	DRUG_TYPE_1	21	2015-09-26	2019-07-13	388

99998	a0faa319-1c7c-11ec-bd56-16262ee38c7f	2017-03-02	DRUG_TYPE_8	21	2015-09-26	2019-07-13	388
99999	a0faa319-1c7c-11ec-bd56-16262ee38c7f	2016-02-09	DRUG_TYPE_1	21	2015-09-26	2019-07-13	388

100000 rows × 7 columns

In [59]:

```
test_data.shape
```

Out[59]:

```
(100000, 7)
```

In [60]:

```
# Make predictions on the test data
test_data['label'] = rf.predict(test_data[features])
```

In [61]:

```
# Save the predictions to a CSV file
test_data[['Patient-Uid', 'label']].to_csv('final_submission.csv', index=False)
```

In [ ]:

In [ ]:

In [ ]:

In [88]:

In [89]:

In [ ]: