# *TABLE OF CONTENTS*

# Claim Prediction Model

## Conduct exploratory data analysis on historical claims data.

### Practical Application

One real world Scanario could be say a retail company wanting to optimize its market stratergies by using historical data to understand key factors influencing decisions of the customer puchases and to identify where other opportunities for expansion of revenue. Descriptive statistics will summarize key metrics, while data visualization will explore relationships between demographics and purchase behavior, sales trends over time, and product popularity. This analysis will uncover patterns and generate insights to inform eAective marketing strategies.

### Understanding

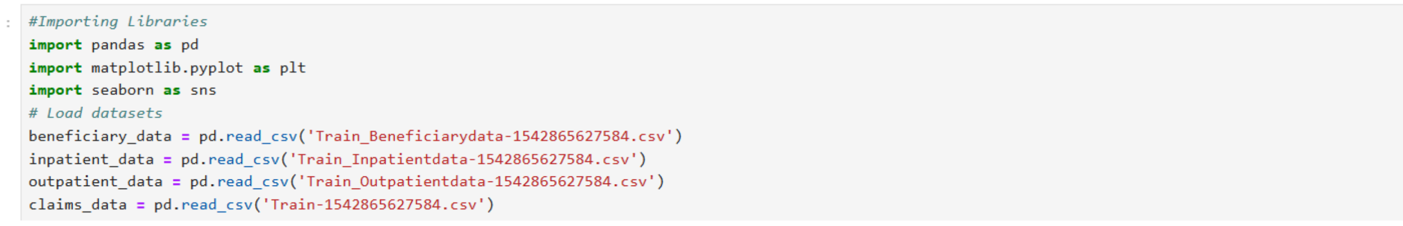
Exploratory Data Analysis (EDA) is a crucial step in the data analysis process, involving the use of statistical tools and visualization techniques to understand the underlying structure of a dataset. EDA aims to uncover patterns, spot anomalies, test hypotheses, and check assumptions through summarizing key metrics and visualizing data distributions. By cleaning the data (handling missing values and outliers) and transforming it as needed, EDA provides insights into relationships between variables, trends over time, and other critical factors, laying the groundwork for more complex analyses and informed decision-making.

Python is extensively used for EDA due to its rich ecosystem of libraries that facilitate data manipulation and visualization. Key Python modules that aid in EDA include

- Pandas for data manipulation and analysis,
- NumPy for numerical operations,
- Matplotlib and Seaborn for creating static, animated, and interactive visualizations,

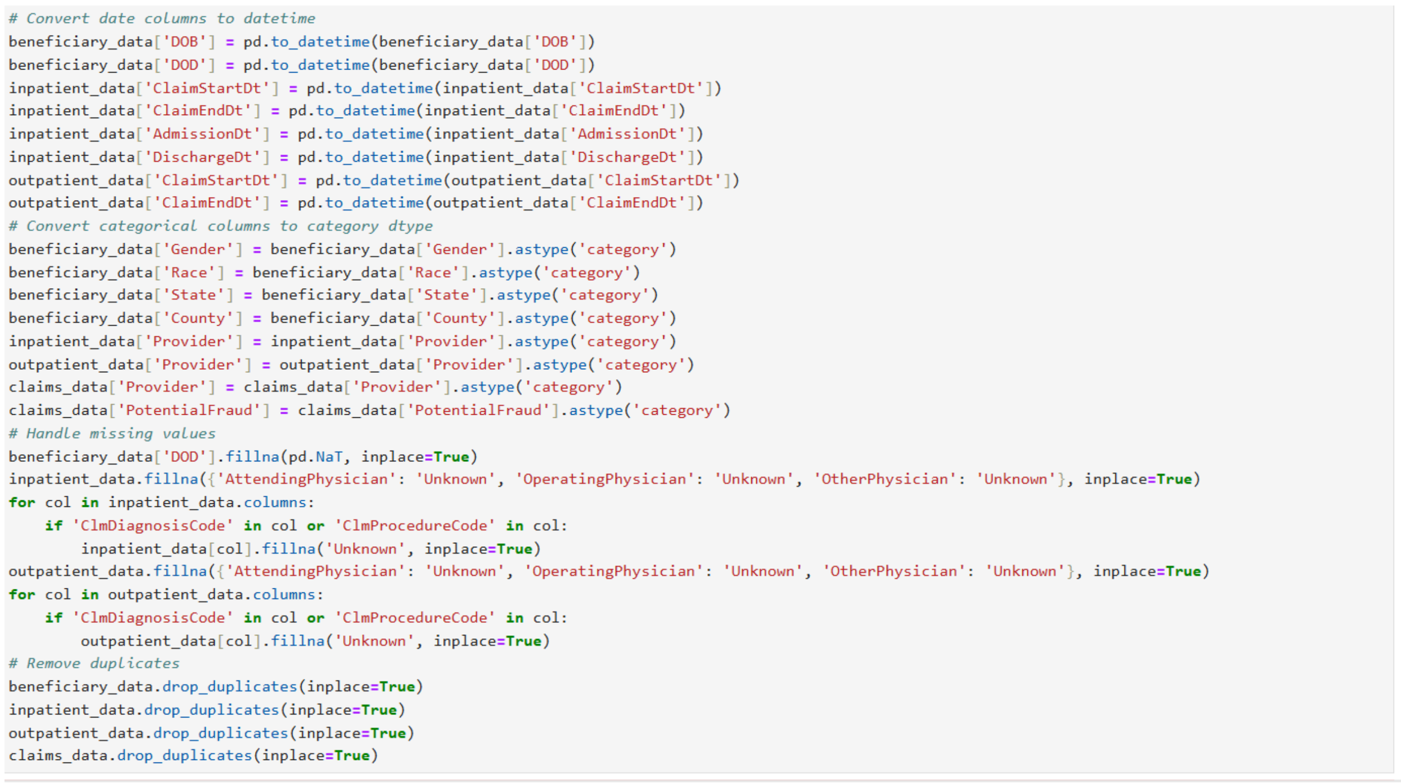
## Python Script

### Importing Libraries and Loading Data Sets



```python
#Importing Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load datasets
beneficiary_data = pd.read_csv('Train_Beneficiarydata-1542865627584.csv')
inpatient_data = pd.read_csv('Train_Inpatientdata-1542865627584.csv')
outpatient_data = pd.read_csv('Train_Outpatientdata-1542865627584.csv')
claims_data = pd.read_csv('Train-1542865627584.csv')
```

### PRE PROCESSING DATA



```python
# Convert date columns to datetime
beneficiary_data['DOB'] = pd.to_datetime(beneficiary_data['DOB'])
beneficiary_data['DOD'] = pd.to_datetime(beneficiary_data['DOD'])
inpatient_data['ClaimStartDt'] = pd.to_datetime(inpatient_data['ClaimStartDt'])
inpatient_data['ClaimEndDt'] = pd.to_datetime(inpatient_data['ClaimEndDt'])
inpatient_data['AdmissionDt'] = pd.to_datetime(inpatient_data['AdmissionDt'])
inpatient_data['DischargeDt'] = pd.to_datetime(inpatient_data['DischargeDt'])
outpatient_data['ClaimStartDt'] = pd.to_datetime(outpatient_data['ClaimStartDt'])
outpatient_data['ClaimEndDt'] = pd.to_datetime(outpatient_data['ClaimEndDt'])
# Convert categorical columns to category dtype
beneficiary_data['Gender'] = beneficiary_data['Gender'].astype('category')
beneficiary_data['Race'] = beneficiary_data['Race'].astype('category')
beneficiary_data['State'] = beneficiary_data['State'].astype('category')
beneficiary_data['County'] = beneficiary_data['County'].astype('category')
inpatient_data['Provider'] = inpatient_data['Provider'].astype('category')
outpatient_data['Provider'] = outpatient_data['Provider'].astype('category')
claims_data['Provider'] = claims_data['Provider'].astype('category')
claims_data['PotentialFraud'] = claims_data['PotentialFraud'].astype('category')
# Handle missing values
beneficiary_data['DOD'].fillna(pd.NaT, inplace=True)
inpatient_data.fillna({'AttendingPhysician': 'Unknown', 'OperatingPhysician': 'Unknown', 'OtherPhysician': 'Unknown'}, inplace=True)
for col in inpatient_data.columns:
    if 'ClmDiagnosisCode' in col or 'ClmProcedureCode' in col:
        inpatient_data[col].fillna('Unknown', inplace=True)
outpatient_data.fillna({'AttendingPhysician': 'Unknown', 'OperatingPhysician': 'Unknown', 'OtherPhysician': 'Unknown'}, inplace=True)
for col in outpatient_data.columns:
    if 'ClmDiagnosisCode' in col or 'ClmProcedureCode' in col:
        outpatient_data[col].fillna('Unknown', inplace=True)
# Remove duplicates
beneficiary_data.drop_duplicates(inplace=True)
inpatient_data.drop_duplicates(inplace=True)
outpatient_data.drop_duplicates(inplace=True)
claims_data.drop_duplicates(inplace=True)
```

- Converted colums to there correct formats
- Filled the missing values
- Removed the duplicate values

### Plotting Graphs to analyse data

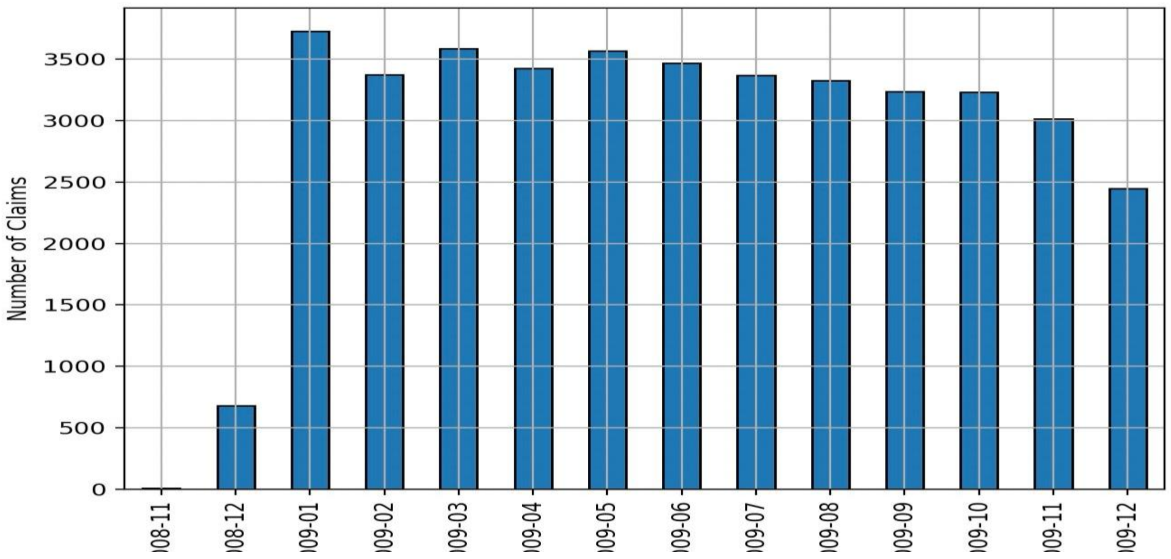
Used Modules like Matplotlib , Seaborn to plot graphs to analyse the data on historical claims data

- Number of claims to Month Histogram
- Age Range wise Reimbursed value
- Gender wise Reimbursed Value
- Distribution of beneficiary ages
- Top Inpatient and Outpatient Diagonosis codes
- Number of claims over time

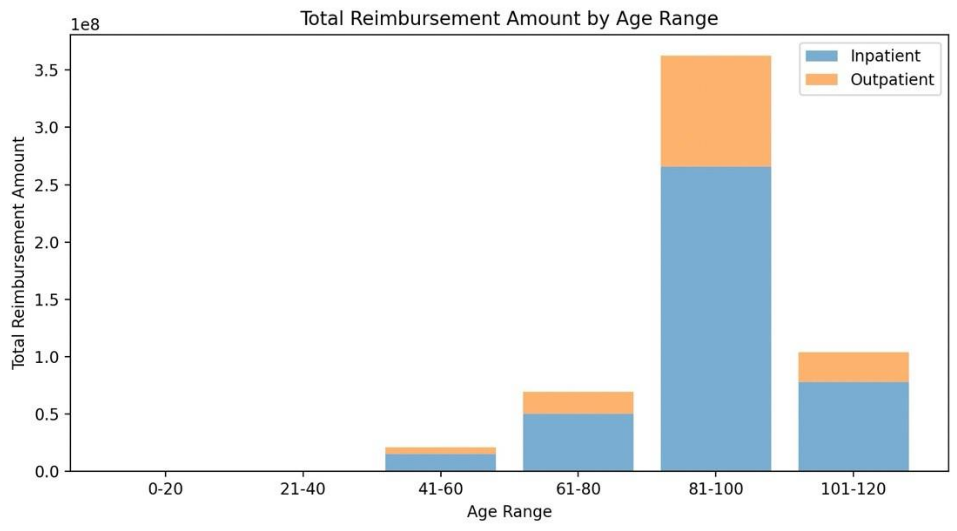
```
inpatient_data['Month-Year'] = inpatient_data['ClaimStartDt'].dt.to_period('M')
month_counts = inpatient_data['Month-Year'].value_counts().sort_index()
month_counts.plot(kind='bar', edgecolor='black')
plt.xlabel('Month-Year')
plt.ylabel('Number of Claims')
plt.grid(True)

# Show the plot
plt.show()
#Find Time
current_year = pd.Timestamp.now().year
beneficiary_data['Age'] = current_year - beneficiary_data['DOB'].dt.year
#Merging data
inpatient_data = inpatient_data.merge(beneficiary_data[['BeneID', 'Age', 'Gender', 'Race']], on='BeneID', how='left')
outpatient_data = outpatient_data.merge(beneficiary_data[['BeneID', 'Age', 'Gender', 'Race']], on='BeneID', how='left')
#Age Range Graph
bins = [0, 20, 40, 60, 80, 100, 120]
labels = ['0-20', '21-40', '41-60', '61-80', '81-100', '101-120']
beneficiary_data['AgeRange'] = pd.cut(beneficiary_data['Age'], bins=bins, labels=labels, right=False)
inpatient_data['AgeRange'] = pd.cut(inpatient_data['Age'], bins=bins, labels=labels, right=False)
outpatient_data['AgeRange'] = pd.cut(outpatient_data['Age'], bins=bins, labels=labels, right=False)
age_reimbursement_inpatient = inpatient_data.groupby('AgeRange')['InscClaimAmtReimbursed'].sum()
age_reimbursement_outpatient = outpatient_data.groupby('AgeRange')['InscClaimAmtReimbursed'].sum()
plt.figure(figsize=(10, 5))
plt.bar(age_reimbursement_inpatient.index.astype(str), age_reimbursement_inpatient, alpha=0.6, label='Inpatient')
plt.bar(age_reimbursement_outpatient.index.astype(str), age_reimbursement_outpatient, alpha=0.6, label='Outpatient', bottom=age_reimbursement_inpatient)
plt.xlabel('Age Range')
plt.ylabel('Total Reimbursement Amount')
plt.title('Total Reimbursement Amount by Age Range')
plt.legend()
plt.show()
#Gender Wise Reimbursement Amount
gender_reimbursement_inpatient = inpatient_data.groupby('Gender')['InscClaimAmtReimbursed'].sum()
gender_reimbursement_outpatient = outpatient_data.groupby('Gender')['InscClaimAmtReimbursed'].sum()
plt.figure(figsize=(10, 5))
plt.bar(gender_reimbursement_inpatient.index.astype(str), gender_reimbursement_inpatient, alpha=0.6, label='Inpatient')
plt.bar(gender_reimbursement_outpatient.index.astype(str), gender_reimbursement_outpatient, alpha=0.6, label='Outpatient', bottom=gender_reimbursement_i
plt.xlabel('Gender')
plt.ylabel('Total Reimbursement Amount')
plt.title('Total Reimbursement Amount by Gender')
plt.legend()
plt.show()
[illegible]
#Gender Wise Reimbursement Amount
gender_reimbursement_inpatient = inpatient_data.groupby('Gender')['InscClaimAmtReimbursed'].sum()
gender_reimbursement_outpatient = outpatient_data.groupby('Gender')['InscClaimAmtReimbursed'].sum()
plt.figure(figsize=(10, 5))
plt.bar(gender_reimbursement_inpatient.index.astype(str), gender_reimbursement_inpatient, alpha=0.6, label='Inpatient')
plt.bar(gender_reimbursement_outpatient.index.astype(str), gender_reimbursement_outpatient, alpha=0.6, label='Outpatient', bottom=gender_reimbursement_in
plt.xlabel('Gender')
plt.ylabel('Total Reimbursement Amount')
plt.title('Total Reimbursement Amount by Gender')
plt.legend()
plt.show()
#Distribution of Beneficiary age
plt.figure(figsize=(10, 5))
sns.histplot(beneficiary_data['Age'], bins=30, kde=True)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Distribution of Beneficiary Ages')
plt.show()
# Top Diagonosis code
top_diagnosis_codes_inpatient = inpatient_data['ClmDiagnosisCode_1'].value_counts().head(10)
top_procedure_codes_inpatient = inpatient_data['ClmProcedureCode_1'].value_counts().head(10)
top_diagnosis_codes_outpatient = outpatient_data['ClmDiagnosisCode_1'].value_counts().head(10)
top_procedure_codes_outpatient = outpatient_data['ClmProcedureCode_1'].value_counts().head(10)
fig, axs = plt.subplots(1, 2, figsize=(15, 5))
# Inpatient Diagnosis Codes
sns.barplot(x=top_diagnosis_codes_inpatient.index, y=top_diagnosis_codes_inpatient.values, ax=axs[0])
axs[0].set_title('Top 10 Inpatient Diagnosis Codes')
axs[0].set_ylabel('Frequency')
axs[0].set_xlabel('Diagnosis Code')
# Outpatient Diagnosis Codes
sns.barplot(x=top_diagnosis_codes_outpatient.index, y=top_diagnosis_codes_outpatient.values, ax=axs[1])
axs[1].set_title('Top 10 Outpatient Diagnosis Codes')
axs[1].set_ylabel('Frequency')
axs[1].set_xlabel('Diagnosis Code')
plt.tight_layout()
plt.show()
```

## Output Graphs

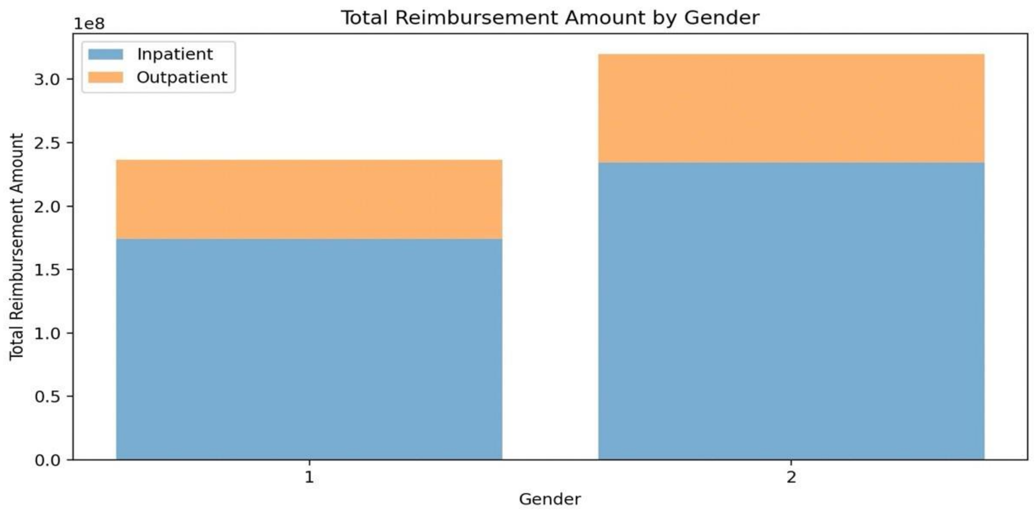
Number of claims to Month Histogram



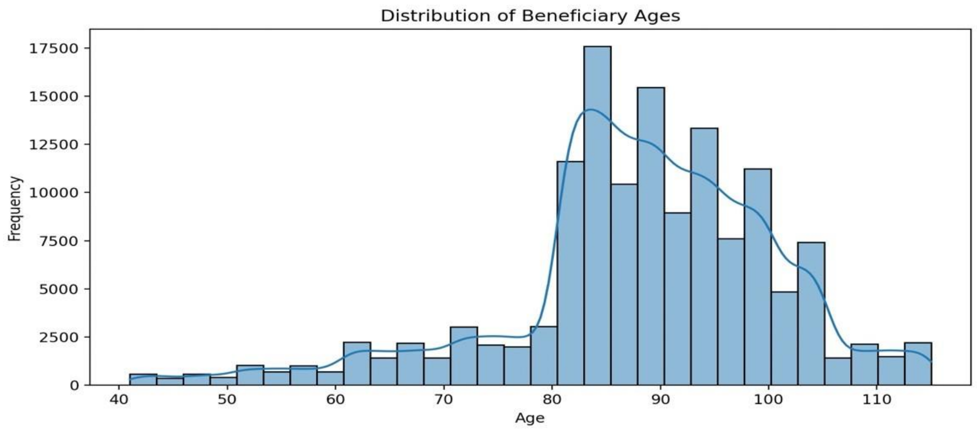
## Age Range wise Reimbursed value



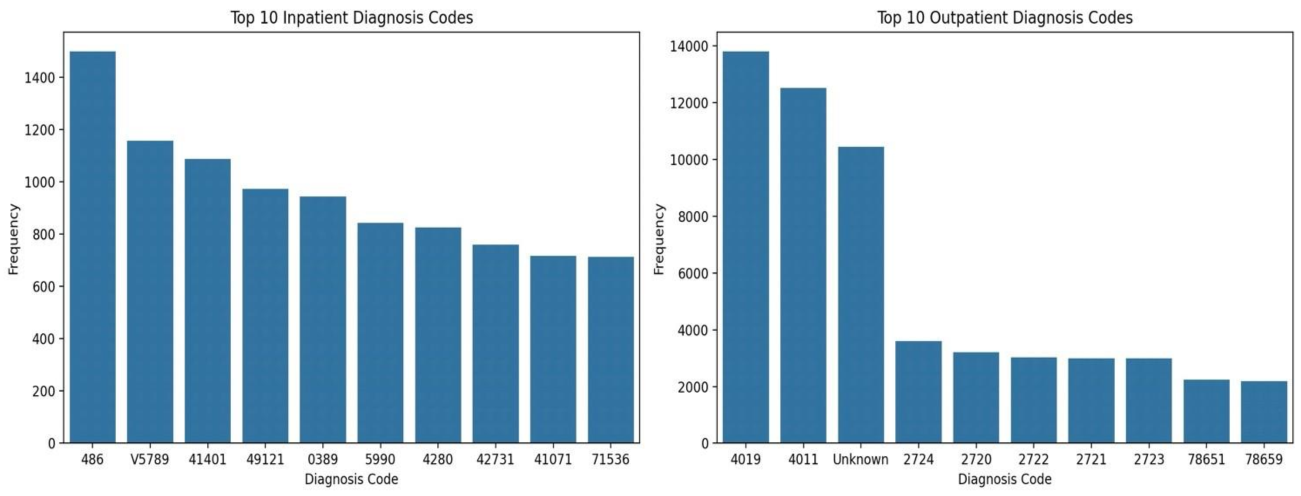
## Gender wise Reimbursed Value



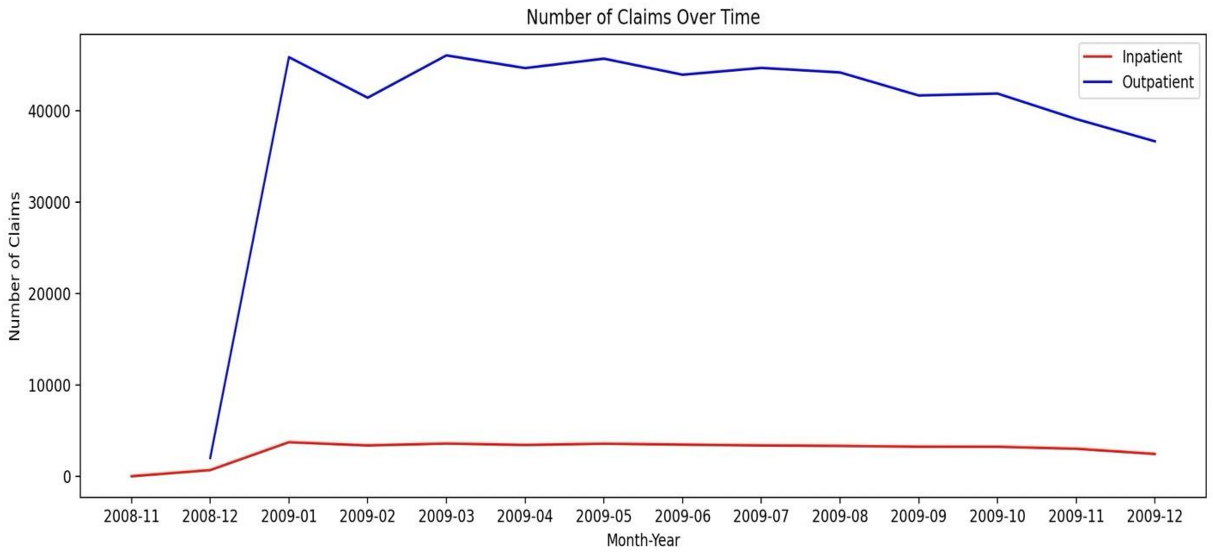
## Distribution of beneficiary ages



## Top Inpatient and Outpatient Diagonosis codes



## Number of claims over time



**Build a predictive model to estimate claim amounts.**

## Practical Applications

One real world practical scenario of prediction models can be to use historical stock prices to predict future stock prices

## Understanding

Prediction models are essential tools in machine learning and data science, used to forecast outcomes based on historical data. These models include various types, each suited to diAerent types of data and prediction tasks

- Linear Regression is used for predicting continuous outcomes.
- Logistic Regression is used for binary classification tasks
- Decision Trees and Random Forests provide interpretable models that can handle both classification and regression tasks by splitting data into branches based on feature values.
- Support Vector Machines (SVM) are powerful for both classification and regression, especially in high-dimensional spaces.
- Neural Networks, including Deep Learning models, are capable of capturing complex patterns in large datasets and are widely used in image and speech recognition tasks.
- Gradient Boosting Machines, such as XGBoost and LightGBM, are highly eAective for structured data.

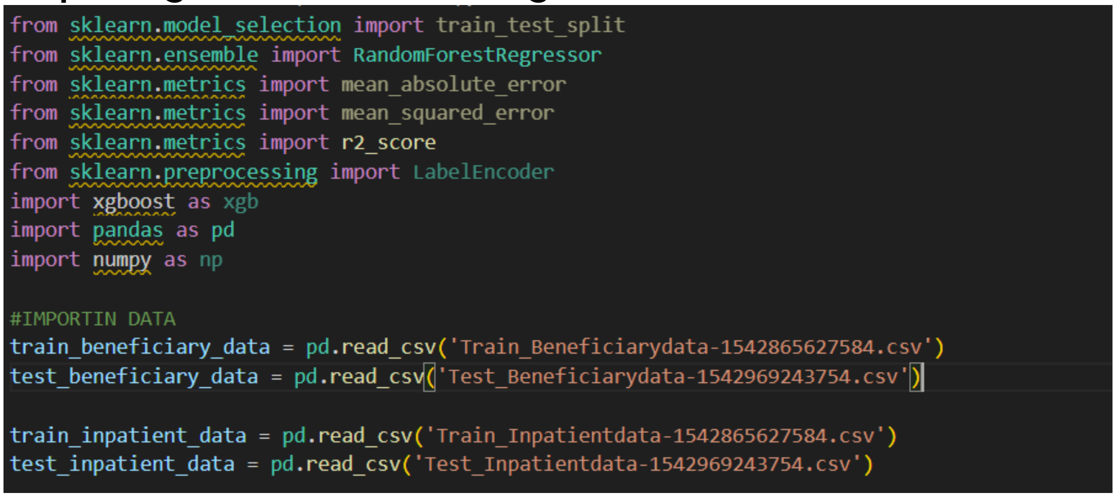
Each of these models has unique strengths and is selected based on the specific requirements of the prediction task at hand.

## Model Used

We chose Random Forest Model as my model as it is an ensemble learning model works on both Numbered and Categorical data well and is highly eAicient

## Script

### Importing Modules and Loading Data

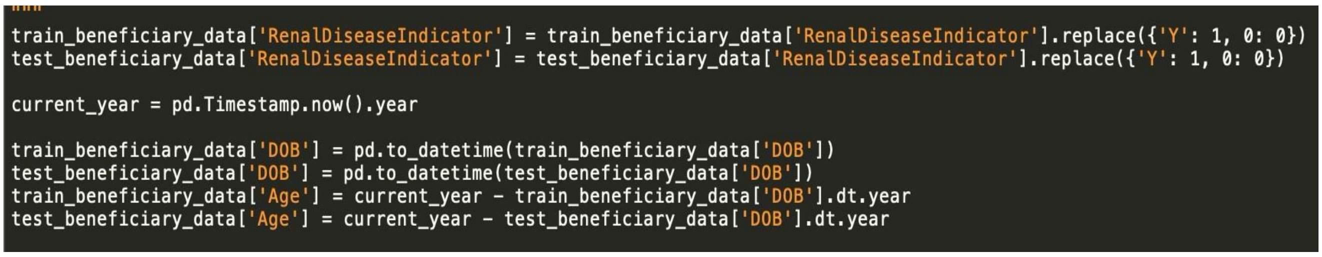


```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb
import pandas as pd
import numpy as np

#IMPORTIN DATA
train_beneficiary_data = pd.read_csv('Train_Beneficiarydata-1542865627584.csv')
test_beneficiary_data = pd.read_csv('Test_Beneficiarydata-1542969243754.csv')

train_inpatient_data = pd.read_csv('Train_Inpatientdata-1542865627584.csv')
test_inpatient_data = pd.read_csv('Test_Inpatientdata-1542969243754.csv')
```

Imported modules :-

- SciKitLearn
- Numpy
- XGBoost
- Pandas
- Numpy

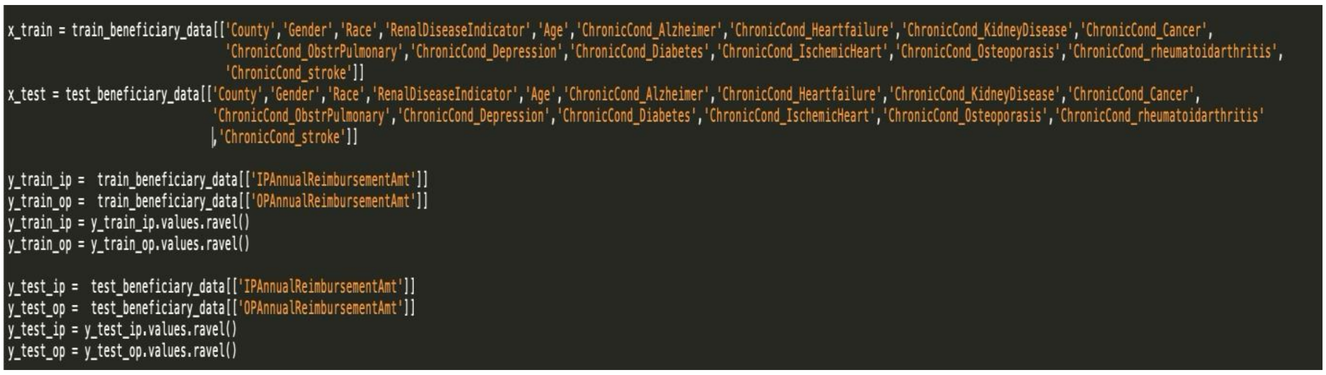
### Preprocessing data



```
train_beneficiary_data['RenalDiseaseIndicator'] = train_beneficiary_data['RenalDiseaseIndicator'].replace({'Y': 1, 0: 0})
test_beneficiary_data['RenalDiseaseIndicator'] = test_beneficiary_data['RenalDiseaseIndicator'].replace({'Y': 1, 0: 0})

current_year = pd.Timestamp.now().year

train_beneficiary_data['DOB'] = pd.to_datetime(train_beneficiary_data['DOB'])
test_beneficiary_data['DOB'] = pd.to_datetime(test_beneficiary_data['DOB'])
train_beneficiary_data['Age'] = current_year - train_beneficiary_data['DOB'].dt.year
test_beneficiary_data['Age'] = current_year - test_beneficiary_data['DOB'].dt.year
```

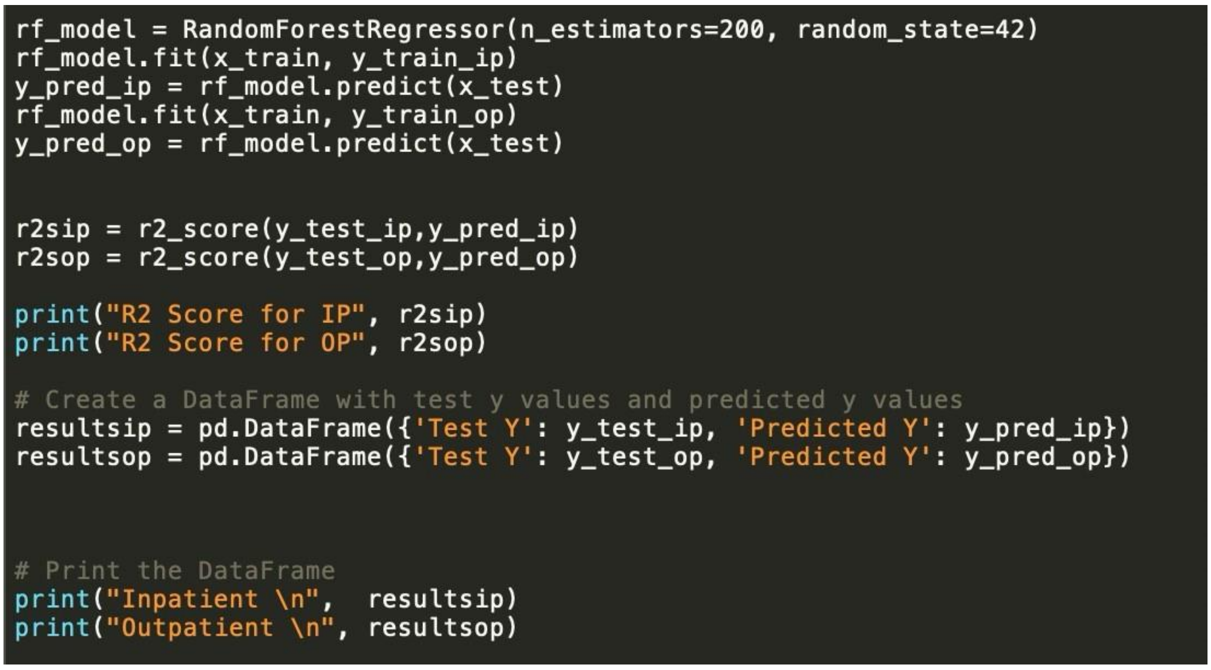
Replaced some entries with 1s and 0s to make it a numeric category
Changed the DOB to datetime format and found age

### Specifying the parameters and the variable to be predicted



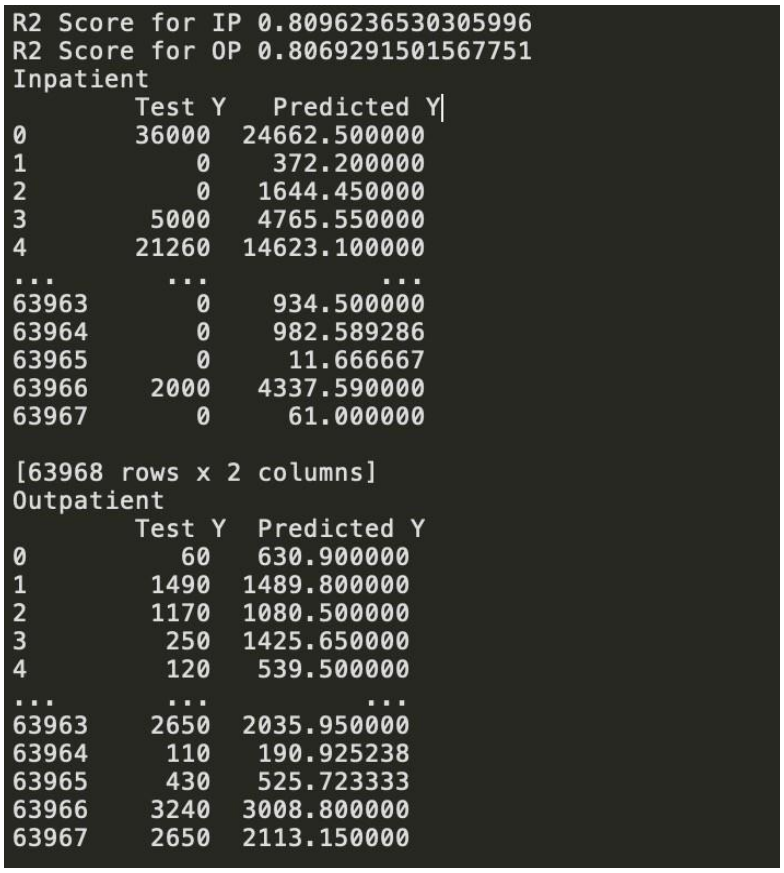
```
x_train = train_beneficiary_data[['County','Gender','Race','RenalDiseaseIndicator','Age','ChronicCond_Alzheimer','ChronicCond_Heartfailure','ChronicCond_KidneyDisease','ChronicCond_Cancer',
                                  'ChronicCond_ObstrPulmonary','ChronicCond_Depression','ChronicCond_Diabetes','ChronicCond_IschemicHeart','ChronicCond_Osteoporasis','ChronicCond_rheumatoidarthritis',
                                  'ChronicCond_stroke']]
x_test = test_beneficiary_data[['County','Gender','Race','RenalDiseaseIndicator','Age','ChronicCond_Alzheimer','ChronicCond_Heartfailure','ChronicCond_KidneyDisease','ChronicCond_Cancer',
                                'ChronicCond_ObstrPulmonary','ChronicCond_Depression','ChronicCond_Diabetes','ChronicCond_IschemicHeart','ChronicCond_Osteoporasis','ChronicCond_rheumatoidarthritis'
                                ,'ChronicCond_stroke']]

y_train_ip =  train_beneficiary_data[['IPAnnualReimbursementAmt']]
y_train_op =  train_beneficiary_data[['OPAnnualReimbursementAmt']]
y_train_ip = y_train_ip.values.ravel()
y_train_op = y_train_op.values.ravel()

y_test_ip =  test_beneficiary_data[['IPAnnualReimbursementAmt']]
y_test_op =  test_beneficiary_data[['OPAnnualReimbursementAmt']]
y_test_ip = y_test_ip.values.ravel()
y_test_op = y_test_op.values.ravel()
```

## Training the Model and outputting the results



```
rf_model = RandomForestRegressor(n_estimators=200, random_state=42)
rf_model.fit(x_train, y_train_ip)
y_pred_ip = rf_model.predict(x_test)
rf_model.fit(x_train, y_train_op)
y_pred_op = rf_model.predict(x_test)


r2sip = r2_score(y_test_ip,y_pred_ip)
r2sop = r2_score(y_test_op,y_pred_op)

print("R2 Score for IP", r2sip)
print("R2 Score for OP", r2sop)

# Create a DataFrame with test y values and predicted y values
resultsip = pd.DataFrame({'Test Y': y_test_ip, 'Predicted Y': y_pred_ip})
resultsop = pd.DataFrame({'Test Y': y_test_op, 'Predicted Y': y_pred_op})



# Print the DataFrame
print("Inpatient \n",  resultsip)
print("Outpatient \n", resultsop)
```

**<u>Output:</u>**



```
R2 Score for IP 0.8096236530305996
R2 Score for OP 0.8069291501567751
Inpatient
        Test Y   Predicted Y
0       36000  24662.500000
1           0    372.200000
2           0   1644.450000
3        5000   4765.550000
4       21260  14623.100000
...       ...           ...
63963       0    934.500000
63964       0    982.589286
63965       0     11.666667
63966    2000   4337.590000
63967       0     61.000000

[63968 rows x 2 columns]
Outpatient
        Test Y  Predicted Y
0          60   630.900000
1        1490  1489.800000
2        1170  1080.500000
3         250  1425.650000
4         120   539.500000
...       ...          ...
63963    2650  2035.950000
63964     110   190.925238
63965     430   525.723333
63966    3240  3008.800000
63967    2650  2113.150000
```