```
import warnings
warnings.filterwarnings("ignore")
```

Importing Libraries

Data Processing and Visualization

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Machine Learning Libraries

Data Splitting

```
from sklearn.model_selection import train_test_split
```

Data Pre-processing

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

Sampling

```
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

ML Algorithms

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Neural Networks

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras import metrics
from keras.initializers import Constant
```

Metrics

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Storing Model

```
import pickle
```

Loading Train and Test Data

```
train_data = pd.read_csv('data/Training Data.csv')
test_data = pd.read_csv('data/Test Data.csv')
```

```
train_data
```

	Id	Income	Age	Experience	Married/Single	House_Ownership	Car_Owner
0	1	1303834	23	3	single	rented	
1	2	7574516	40	10	single	rented	
2	3	3991815	66	4	married	rented	
3	4	6256451	41	2	single	rented	
4	5	5768871	47	11	single	rented	
...	...	...	...	...	...	...	
251995	251996	8154883	43	13	single	rented	
251996	251997	2843572	26	10	single	rented	
251997	251998	4522448	46	7	single	rented	
251998	251999	6507128	45	0	single	rented	
251999	252000	9070230	70	17	single	rented	

252000 rows × 13 columns

```
test_data
```

	ID	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	1	7393090	59	19	single	rented	
1	2	1215004	25	5	single	rented	
2	3	8901342	50	12	single	rented	
3	4	1944421	49	9	married	rented	y
4	5	13429	25	18	single	rented	y
...	...	...	...	...	...	...	
27995	27996	9955481	57	13	single	rented	
27996	27997	2917765	47	9	single	rented	
27997	27998	8082415	24	5	single	rented	
27998	27999	9474180	51	13	single	rented	y
27999	28000	9250350	42	9	single	rented	

28000 rows × 12 columns

Exploratory Data Analysis

Checking Data Type

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252000 entries, 0 to 251999
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   Id                 252000 non-null  int64 
 1   Income             252000 non-null  int64 
 2   Age                252000 non-null  int64 
 3   Experience         252000 non-null  int64 
 4   Married/Single     252000 non-null  object
 5   House_Ownership    252000 non-null  object
 6   Car_Ownership      252000 non-null  object
 7   Profession         252000 non-null  object
 8   CITY               252000 non-null  object
```

```
 9   STATE              252000 non-null  object
 10  CURRENT_JOB_YRS    252000 non-null  int64
 11  CURRENT_HOUSE_YRS  252000 non-null  int64
 12  Risk_Flag          252000 non-null  int64
dtypes: int64(7), object(6)
memory usage: 25.0+ MB
```

We conclude that the dataset has no null values

Data Visualization

```
def plot_hist(data1, data2, feature, labels):
    plt.hist([data1[feature], data2[feature]], bins=10, label=labels, density=True, stacke
    plt.title("Defaults based on {}".format(feature))
    plt.legend()
    plt.xlabel(feature)
    plt.show()

def plot_pie(data, labels, title, axis=None, ax_index=None):
    if axis is not None and ax_index is not None:
        axis[ax_index % 13, ax_index // 13].pie(data, labels=labels, autopct='%1.1f%%')
        axis[ax_index % 13, ax_index // 13].set_title("Defaults based on {}".format(title)
    else:
        plt.pie(data, labels=labels, autopct='%1.1f%%')
        plt.title("Defaults based on {}".format(title))
        plt.show()
```

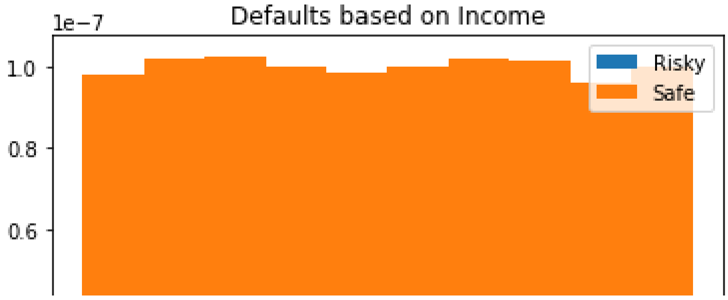
```
def get_defaults(data, feature, value, text=None):
    all_value = data[data[feature] == value]['Risk_Flag'].value_counts()
    print("Proportion of {} who default: {}".format(text or value, all_value[1]/(all_value
    return all_value
```

```
risky_1 = train_data[train_data['Risk_Flag'] == 1].copy()
risky_0 = train_data[train_data['Risk_Flag'] == 0].copy()
```

```
flags = ['Risky', 'Safe']
```

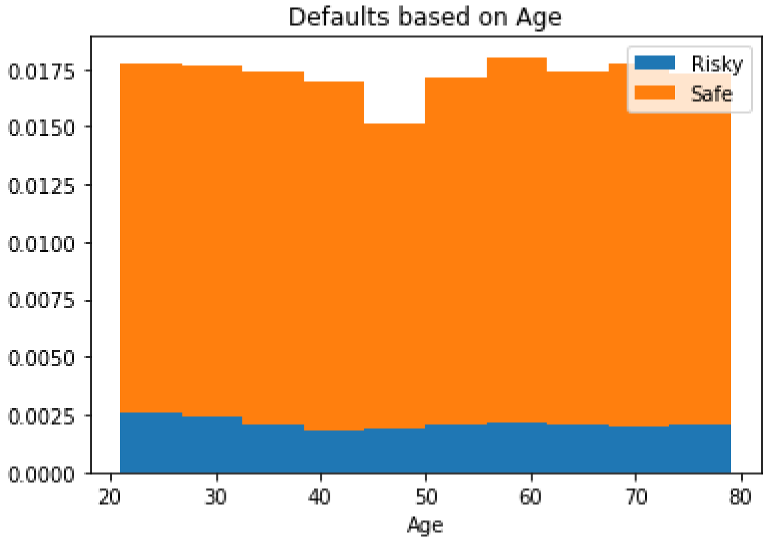
Influence on Income on Risk factor

```
plot_hist(risky_1, risky_0, 'Income', flags)
```



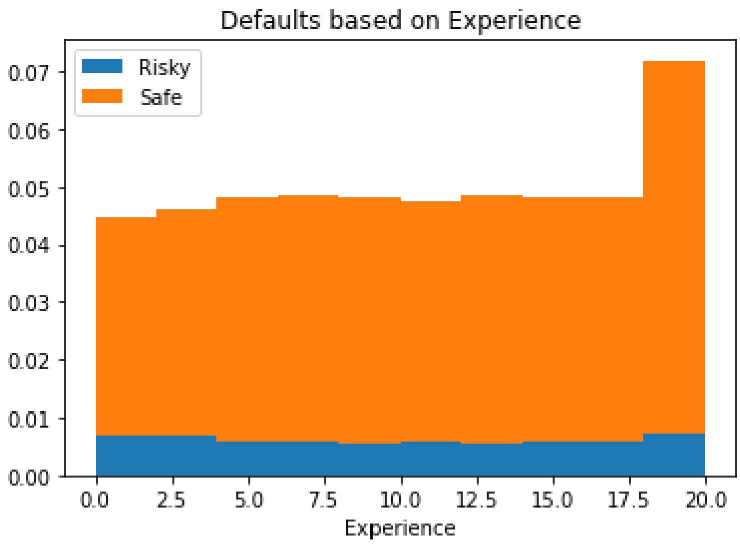
Influence of Age on Risk Factor

```
plot_hist(risky_1, risky_0, 'Age', flags)
```



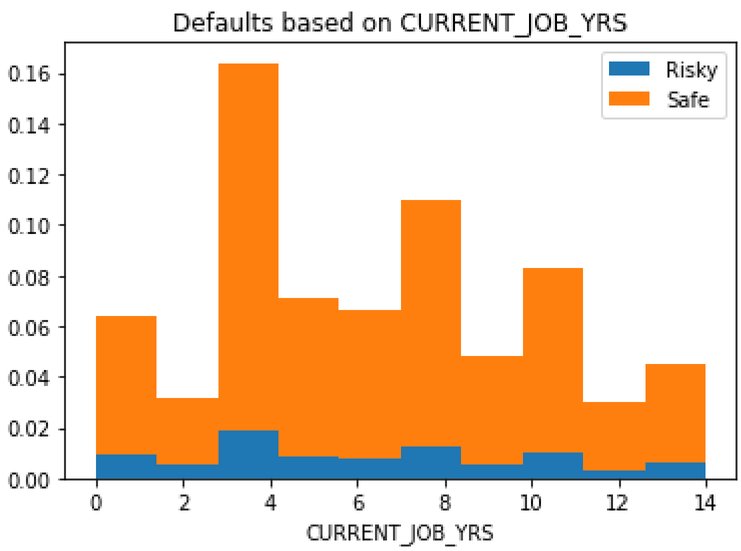
Influence of Experience on Risk Factor

```
plot_hist(risky_1, risky_0, 'Experience', flags)
```



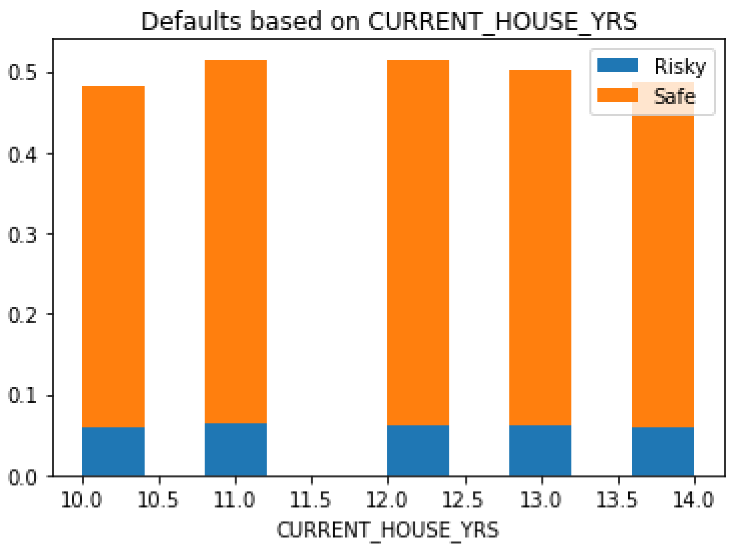
Influence of Job Years on Risk Factor

```
plot_hist(risky_1, risky_0, 'CURRENT_JOB_YRS', flags)
```



Influence of House Years on Risk Factor

```
plot_hist(risky_1, risky_0, 'CURRENT_HOUSE_YRS', flags)
```



Influence of Gender on Risk Factor

```
train_data['Married/Single'].value_counts()
```

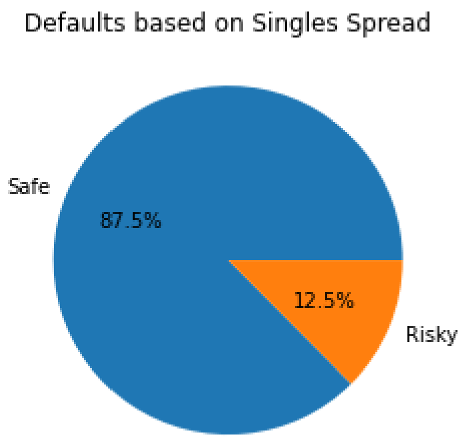
```
single     226272
married     25728
Name: Married/Single, dtype: int64
```

```
all_singles = get_defaults(train_data, 'Married/Single', 'single')
```

```
Proportion of single who default: 0.1253358789421581
```

```
plot_pie(all_singles, flags[::-1], 'Singles Spread')
```

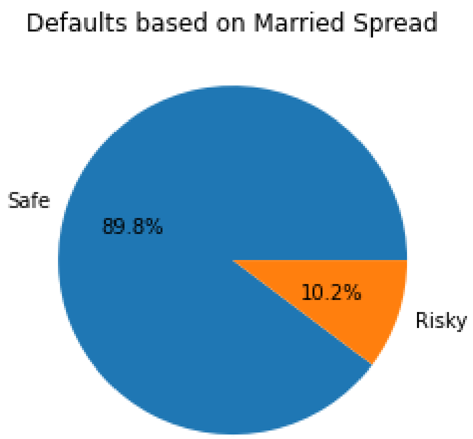
```
plot_pie(all_singles, flags[::-1], 'Singles Spread')
```




```
all_married = get_defaults(train_data, 'Married/Single', 'married')
```

```
Proportion of married who default: 0.10245646766169154
```

```
plot_pie(all_married, flags[::-1], 'Married Spread')
```



Influence of House Ownership on Risk Factor

```
train_data['House_Ownership'].value_counts()
```

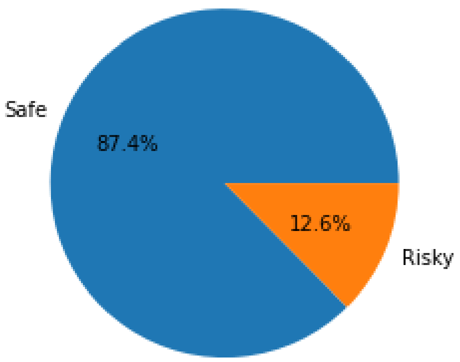
```
rented          231898
owned            12918
norent_noown      7184
Name: House_Ownership, dtype: int64
```

```
all_rented = get_defaults(train_data, 'House_Ownership', 'rented')
```

```
Proportion of rented who default: 0.1255767621971729
```

```
plot_pie(all_rented, flags[::-1], 'Rented Spread')
```

Defaults based on Rented Spread

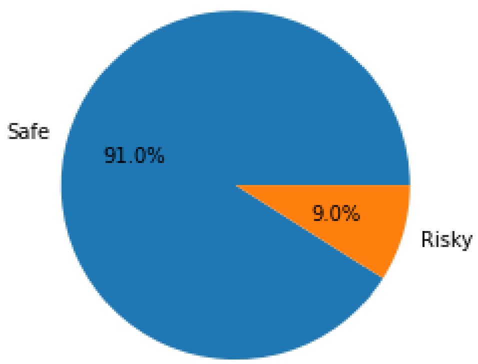



```
all_owned = get_defaults(train_data, 'House_Ownership', 'owned')
```

```
Proportion of owned who default: 0.08979718222635083
```

```
plot_pie(all_owned, flags[::-1], 'House Owned Spread')
```

Defaults based on House Owned Spread




```
all_noown = get_defaults(train_data, 'House_Ownership', 'norent_noown')
```

```
Proportion of norent_noown who default: 0.09952672605790645
```

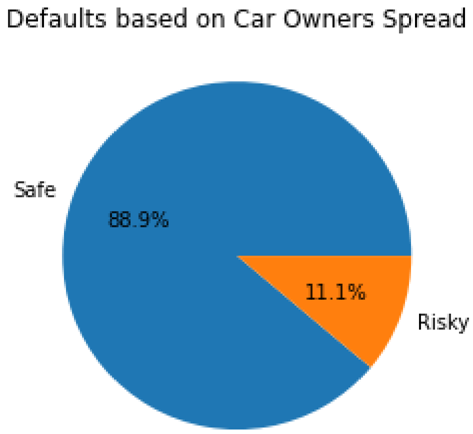
```
plot_pie(all_noown, flags[::-1], 'No Rent No Own Spread')
```

Influence of Car Ownership on Risk Factor

```
all_car = get_defaults(train_data, 'Car_Ownership', 'yes', 'Car Owners')
```

```
Proportion of Car Owners who default: 0.11098684210526316
```

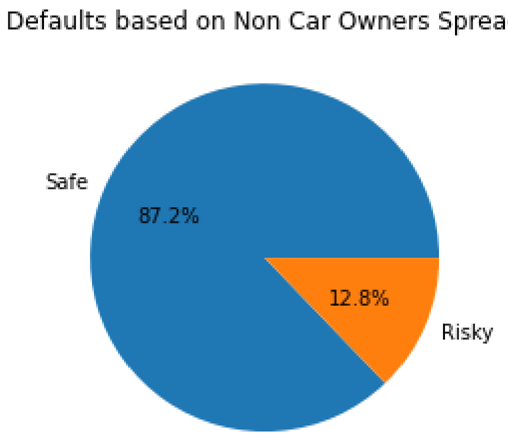
```
plot_pie(all_car, flags[::-1], 'Car Owners Spread')
```




```
no_car = get_defaults(train_data, 'Car_Ownership', 'no', 'Non Car Owners')
```

```
Proportion of Non Car Owners who default: 0.1281875
```

```
plot_pie(no_car, flags[::-1], 'Non Car Owners Spread')
```



Influence of Profession on Risk Factor

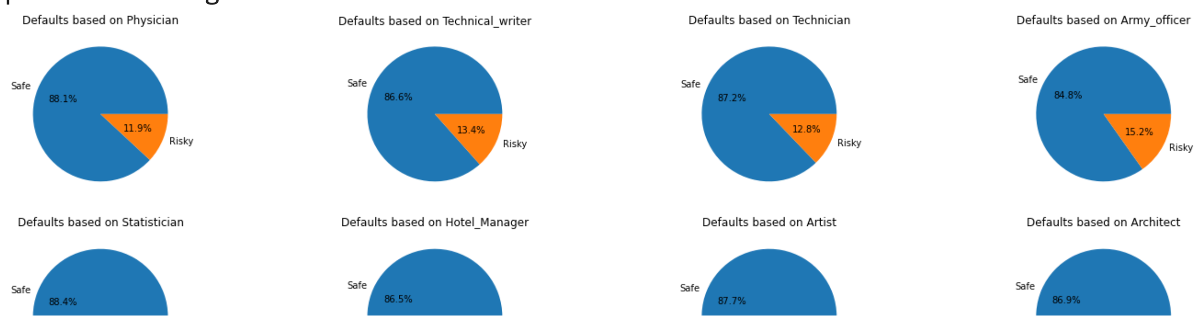
```
train_data['Profession'].value_counts()
```

```
Physician                     5957
Statistician                  5806
Web_designer                  5397
```

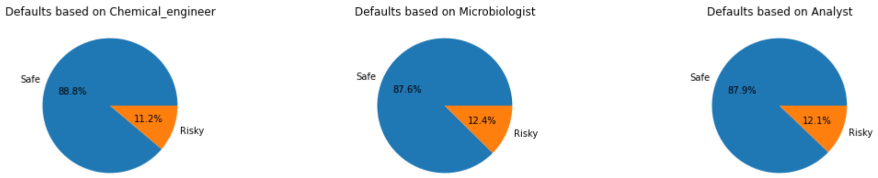
```
Psychologist                  5390
Computer_hardware_engineer    5372
Drafter                       5359
Magistrate                    5357
Fashion_Designer              5304
Air_traffic_controller        5281
Comedian                      5259
Industrial_Engineer           5250
Mechanical_engineer           5217
Chemical_engineer             5205
Technical_writer              5195
Hotel_Manager                 5178
Financial_Analyst             5167
Graphic_Designer              5166
Flight_attendant              5128
Biomedical_Engineer           5127
Secretary                     5061
Software_Developer            5053
Petroleum_Engineer            5041
Police_officer                5035
Computer_operator             4990
Politician                    4944
Microbiologist                4881
Technician                    4864
Artist                        4861
Lawyer                        4818
Consultant                    4808
Dentist                       4782
Scientist                     4781
Surgeon                       4772
Aviator                       4758
Technology_specialist         4737
Design_Engineer               4729
Surveyor                      4714
Geologist                     4672
Analyst                       4668
Army_officer                  4661
Architect                     4657
Chef                          4635
Librarian                     4628
Civil_engineer                4616
Designer                      4598
Economist                     4573
Firefighter                   4507
Chartered_Accountant          4493
Civil_servant                 4413
Official                      4087
Engineer                      4048
Name: Profession, dtype: int64
```

```
professions = list(train_data['Profession'].value_counts().index)
fig, ax = plt.subplots(13, 4, figsize=(25, 52))
ax_index = 0
for profession in professions:
    all_data = get_defaults(train_data, 'Profession', profession)
    plot_pie(all_data, flags[::-1], str(profession), ax, ax_index)
    ax_index += 1
_ = ax[-1, -1].axis('off')
```

```
Proportion of Physician who default: 0.11918751049185831
Proportion of Statistician who default: 0.11557009989665863
Proportion of Web_designer who default: 0.10913470446544377
Proportion of Psychologist who default: 0.12189239332096476
Proportion of Computer_hardware_engineer who default: 0.12844378257632166
Proportion of Drafter who default: 0.1128941966784848
Proportion of Magistrate who default: 0.12002986746313235
Proportion of Fashion_Designer who default: 0.11538461538461539
Proportion of Air_traffic_controller who default: 0.1353910244271918
Proportion of Comedian who default: 0.11960448754516068
Proportion of Industrial_Engineer who default: 0.09866666666666667
Proportion of Mechanical_engineer who default: 0.11155836687751582
Proportion of Chemical_engineer who default: 0.11162343900096061
Proportion of Technical_writer who default: 0.134167468719923
Proportion of Hotel_Manager who default: 0.13538045577443028
Proportion of Financial_Analyst who default: 0.10315463518482679
Proportion of Graphic_Designer who default: 0.11536972512582269
Proportion of Flight_attendant who default: 0.12363494539781592
Proportion of Biomedical_Engineer who default: 0.12755997659449972
Proportion of Secretary who default: 0.13040901007705988
Proportion of Software_Developer who default: 0.1484266772214526
Proportion of Petroleum_Engineer who default: 0.08510216226939099
Proportion of Police_officer who default: 0.16405163853028798
Proportion of Computer_operator who default: 0.12404809619238477
Proportion of Politician who default: 0.11225728155339806
Proportion of Microbiologist who default: 0.12435976234378202
Proportion of Technician who default: 0.12828947368421054
Proportion of Artist who default: 0.1226085167660975
Proportion of Lawyer who default: 0.1295143212951432
Proportion of Consultant who default: 0.1252079866888519
Proportion of Dentist who default: 0.109577582601422
Proportion of Scientist who default: 0.14432127170048106
Proportion of Surgeon who default: 0.11546521374685666
Proportion of Aviator who default: 0.13493064312736444
Proportion of Technology_specialist who default: 0.08148617268313278
Proportion of Design_Engineer who default: 0.1069993656164094
Proportion of Surveyor who default: 0.15146372507424694
Proportion of Geologist who default: 0.144263698630137
Proportion of Analyst who default: 0.12146529562982006
Proportion of Army_officer who default: 0.15211328041192876
Proportion of Architect who default: 0.13120034356882113
Proportion of Chef who default: 0.12146709816612729
Proportion of Librarian who default: 0.11257562662057044
Proportion of Civil_engineer who default: 0.1358318890814558
Proportion of Designer who default: 0.10917790343627665
Proportion of Economist who default: 0.09927837305926088
Proportion of Firefighter who default: 0.13578877301974707
Proportion of Chartered_Accountant who default: 0.15357222345871355
Proportion of Civil_servant who default: 0.11579424427826875
Proportion of Official who default: 0.1357964276975777
Proportion of Engineer who default: 0.11808300395256917
```

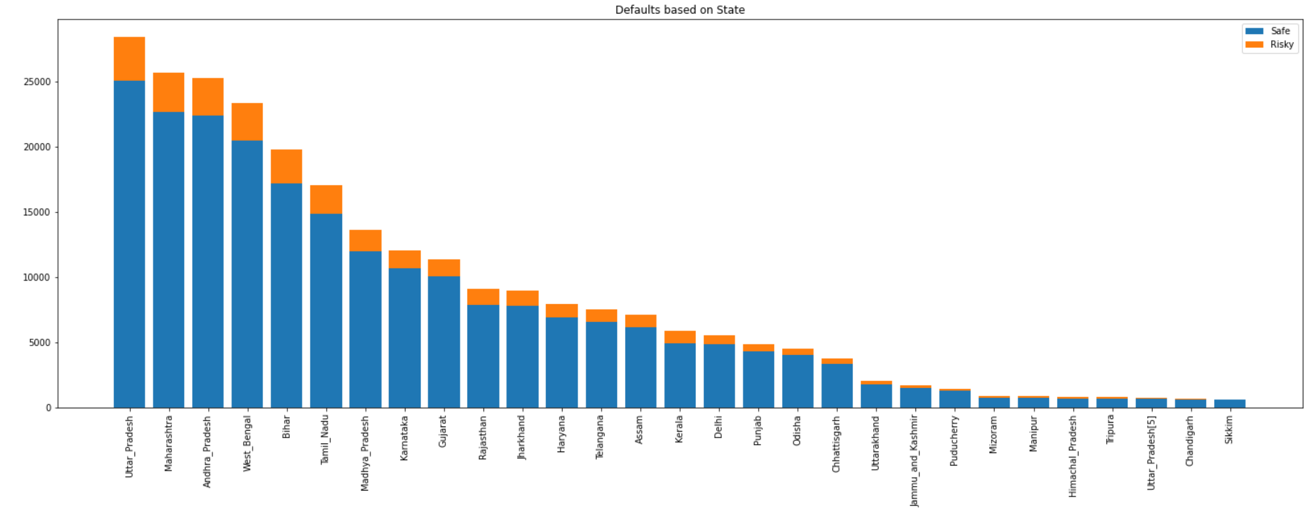



11.6% Risky
13.5% Risky
12.3% Risky
13.1% Risky
Defaults based on Web_designer
Safe 89.1% 10.9% Risky
Defaults based on Financial_Analyst
Safe 89.7% 10.3% Risky
Defaults based on Lawyer
Safe 87.0% 13.0% Risky
Defaults based on Chef
Safe 87.9% 12.1% Risky
Defaults based on Psychologist
Safe 87.8% 12.2% Risky
Defaults based on Graphic_Designer
Safe 88.5% 11.5% Risky
Defaults based on Consultant
Safe 87.5% 12.5% Risky
Defaults based on Librarian
Safe 88.7% 11.3% Risky
Defaults based on Computer_hardware_engineer
Safe 87.2% 12.8% Risky
Defaults based on Flight_attendant
Safe 87.6% 12.4% Risky
Defaults based on Dentist
Safe 89.0% 11.0% Risky
Defaults based on Civil_engineer
Safe 86.4% 13.6% Risky
Defaults based on Drafter
Safe 88.7% 11.3% Risky
Defaults based on Biomedical_Engineer
Safe 87.2% 12.8% Risky
Defaults based on Scientist
Safe 85.6% 14.4% Risky
Defaults based on Designer
Safe 89.1% 10.9% Risky
Defaults based on Magistrate
Safe 88.0% 12.0% Risky
Defaults based on Secretary
Safe 87.0% 13.0% Risky
Defaults based on Surgeon
Safe 88.5% 11.5% Risky
Defaults based on Economist
Safe 90.1% 9.9% Risky
Defaults based on Fashion_Designer
Safe 88.5% 11.5% Risky
Defaults based on Software_Developer
Safe 85.2% 14.8% Risky
Defaults based on Aviator
Safe 86.5% 13.5% Risky
Defaults based on Firefighter
Safe 86.4% 13.6% Risky
Defaults based on Air_traffic_controller
Safe 86.5% 13.5% Risky
Defaults based on Petroleum_Engineer
Safe 91.5% 8.5% Risky
Defaults based on Technology_specialist
Safe 91.9% 8.1% Risky
Defaults based on Chartered_Accountant
Safe 84.6% 15.4% Risky
Defaults based on Comedian
Safe 88.0% 12.0% Risky
Defaults based on Police_officer
Safe 83.6% 16.4% Risky
Defaults based on Design_Engineer
Safe 89.3% 10.7% Risky
Defaults based on Civil_servant
Safe 88.4% 11.6% Risky
Defaults based on Industrial_Engineer
Safe 90.1% 9.9% Risky
Defaults based on Computer_operator
Safe 87.6% 12.4% Risky
Defaults based on Surveyor
Safe 84.9% 15.1% Risky
Defaults based on Official
Safe 86.4% 13.6% Risky
Defaults based on Mechanical_engineer
Safe 88.8% 11.2% Risky
Defaults based on Politician
Safe 88.8% 11.2% Risky
Defaults based on Geologist
Safe 85.6% 14.4% Risky
Defaults based on Engineer
Safe 88.2% 11.8% Risky



Influence of State on Risk Factor

```
states = list(train_data['STATE'].value_counts().index)
fig, ax = plt.subplots(figsize=(25, 8))
ax.bar(states, risky_0['STATE'].value_counts())
ax.bar(states, risky_1['STATE'].value_counts(), bottom=risky_0['STATE'].value_counts())
plt.title("Defaults based on State")
plt.xticks(rotation=90)
_ = plt.legend(flags[::-1])
```



Data Preprocessing

Remove unwanted columns

```
train_data.drop('Id', axis=1, inplace=True)
```

Encoding Categorical Columns

```
categorical_cols = [col for col in train_data.columns if train_data[col].dtype=="O"]

label_encoder = LabelEncoder()

for col in categorical_cols:
    train_data[col] = label_encoder.fit_transform(train_data[col])
```

```
train_data
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownership	Pr
0	1303834	23	3	1	2	0	
1	7574516	40	10	1	2	0	
2	3991815	66	4	0	2	0	
3	6256451	41	2	1	2	1	
4	5768871	47	11	1	2	0	

```
train_data = train_data.drop_duplicates(subset=list(train_data.columns)[:-1])
train_data.shape
```

```
(42007, 12)
```

Splitting Dataset into Training and Validation Set

```
X = train_data.drop('Risk_Flag', axis=1)
y = train_data['Risk_Flag']
```

```
smote = SMOTE(random_state=42)
X_over, y_over = smote.fit_resample(X.values, y)
X_train_over, X_val_over, y_train_over, y_val_over = train_test_split(X_over, y_over, test
```

```
undersampler = RandomUnderSampler(sampling_strategy='majority')
X_under, y_under = undersampler.fit_resample(X.values, y)
X_train_under, X_val_under, y_train_under, y_val_under = train_test_split(X_under, y_under
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=101, s
```

Scaling Data Values in Training Set

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)

X_train_over = scaler.fit_transform(X_train_over)
X_val_over = scaler.transform(X_val_over)

X_train_under = scaler.fit_transform(X_train_under)
X_val_under = scaler.transform(X_val_under)
```

Training Model

```
def model_apply(model, X_training, y_training, X_validation, y_validation, filename):
    model.fit(X_training, y_training)
    y_pred = model.predict(X_validation)

    print("Accuracy: {}".format(accuracy_score(y_validation, y_pred)))

    print("Classification Report: \n{}".format(classification_report(y_validation, y_pred)

    conf_matrix = confusion_matrix(y_validation, y_pred)
    print("Confusion Matrix: ")
    sns.heatmap(conf_matrix, annot=True, fmt='d')

    filename = "model/{}.pkl".format(filename)
    pickle.dump(model, open(filename, 'wb'))
```

```
logistic_regression = LogisticRegression(class_weight='balanced', penalty="l1", solver="li
decision_tree = DecisionTreeClassifier(class_weight='balanced', criterion='gini', random_s
random_forest = RandomForestClassifier(class_weight='balanced', random_state=101, n_estima
neighbours = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
```

Logistic Regression

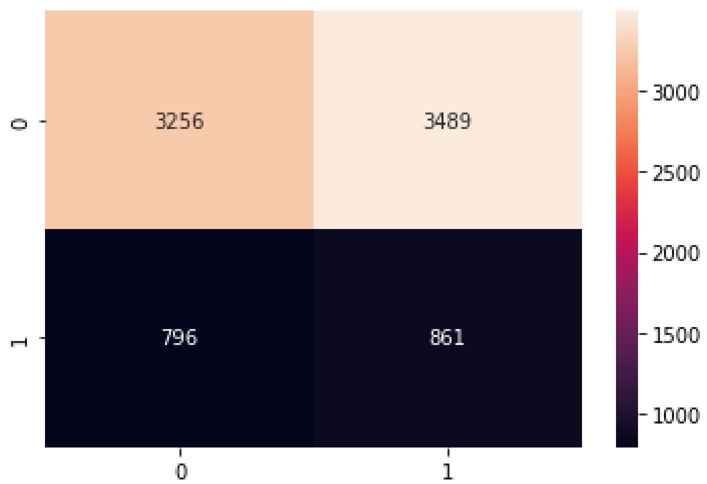
```
print("On entire Data")
model_apply(logistic_regression, X_train, y_train, X_val, y_val, 'LogisticRegression')
```

```
On entire Data
Accuracy: 0.4900023803856225
Classification Report: 
              precision    recall  f1-score   support

           0       0.80      0.48      0.60      6745
           1       0.20      0.52      0.29      1657

    accuracy                           0.49      8402
   macro avg       0.50      0.50      0.44      8402
weighted avg       0.68      0.49      0.54      8402

Confusion Matrix: 
```




```
print("On Random Under Sample")
```

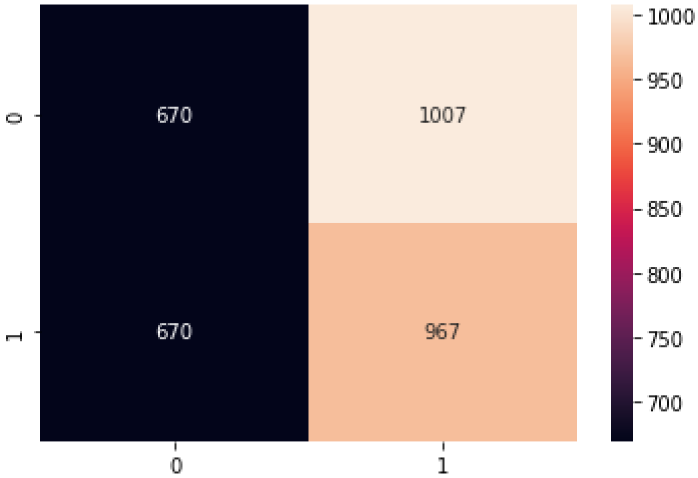
```
model_apply(logistic_regression, X_train_under, y_train_under, X_val_under, y_val_under, '
```

```
On Random Under Sample
Accuracy: 0.4939649969824985
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.40      0.44      1677
           1       0.49      0.59      0.54      1637

    accuracy                           0.49      3314
   macro avg       0.49      0.50      0.49      3314
weighted avg       0.49      0.49      0.49      3314

Confusion Matrix:
```




```
print("On SMOTE Sample")
model_apply(logistic_regression, X_train_over, y_train_over, X_val_over, y_val_over, 'Logi
```

```
On SMOTE Sample
Accuracy: 0.6044184150048187
```

Decision Tree

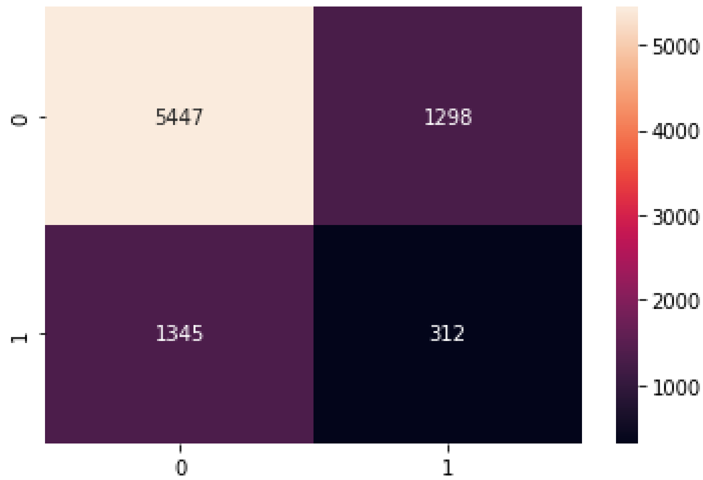
```
print("On entire Data")
model_apply(decision_tree, X_train, y_train, X_val, y_val, 'DecisionTree')
```

```
On entire Data
Accuracy: 0.6854320399904784
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.81      0.80      6745
           1       0.19      0.19      0.19      1657

    accuracy                           0.69      8402
   macro avg       0.50      0.50      0.50      8402
weighted avg       0.68      0.69      0.68      8402

Confusion Matrix:
```




```
print("On Random Under Sample")
model_apply(decision_tree, X_train_under, y_train_under, X_val_under, y_val_under, 'Decisi
```

```
On Random Under Sample
Accuracy: 0.4927579963789982
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.49      0.49      1677
           1       0.49      0.50      0.49      1637

    accuracy                           0.49      3314
   macro avg       0.49      0.49      0.49      3314
weighted avg       0.49      0.49      0.49      3314
```

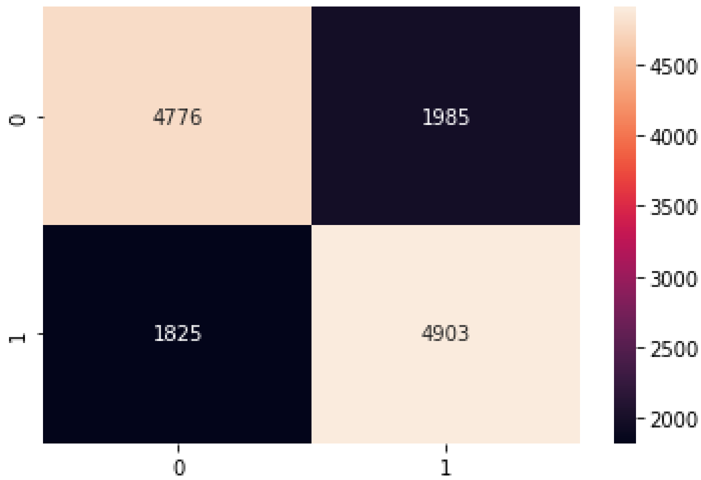
```
print("On SMOTE Sample")
model_apply(decision_tree, X_train_over, y_train_over, X_val_over, y_val_over, 'DecisionTr
```

```
On SMOTE Sample
Accuracy: 0.7175476314033657
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.71      0.71      6761
           1       0.71      0.73      0.72      6728

    accuracy                           0.72     13489
   macro avg       0.72      0.72      0.72     13489
weighted avg       0.72      0.72      0.72     13489

Confusion Matrix:
```



Random Forest

```
print("On entire Data")
model_apply(random_forest, X_train, y_train, X_val, y_val, 'RandomForest')
```

```
On entire Data
Accuracy: 0.8026660318971673
Classification Report:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      6745
           1       0.00      0.00      0.00      1657

    accuracy                           0.80      8402
   macro avg       0.40      0.50      0.45      8402
weighted avg       0.64      0.80      0.71      8402

Confusion Matrix:
```

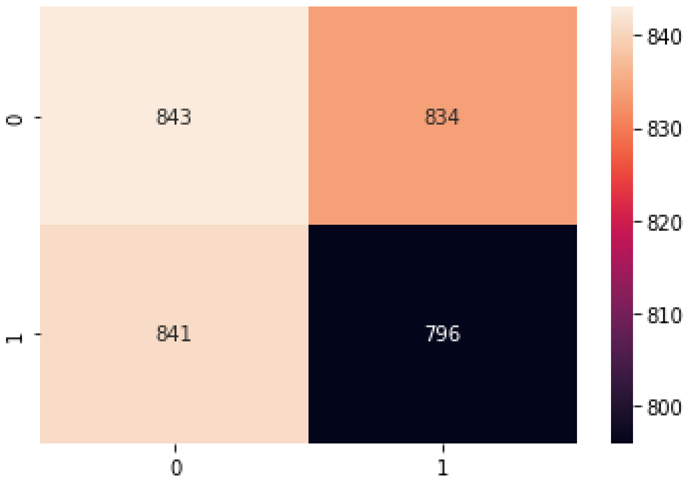

```
print("On Random Under Sample")
model_apply(random_forest, X_train_under, y_train_under, X_val_under, y_val_under, 'Random
```

```
On Random Under Sample
Accuracy: 0.4945684972842486
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.50      0.50      1677
           1       0.49      0.49      0.49      1637

    accuracy                           0.49      3314
   macro avg       0.49      0.49      0.49      3314
weighted avg       0.49      0.49      0.49      3314

Confusion Matrix:
```



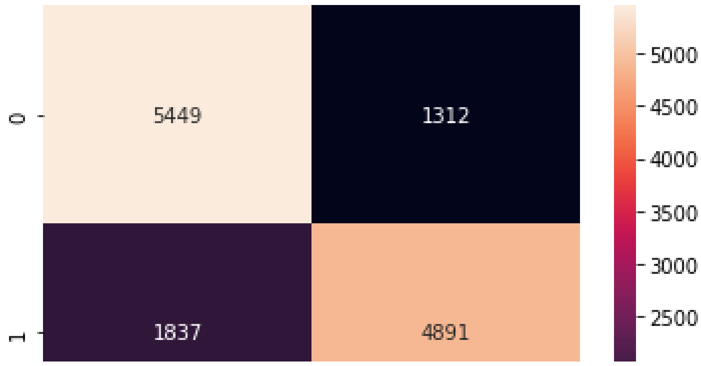

```
print("On SMOTE Sample")
model_apply(random_forest, X_train_over, y_train_over, X_val_over, y_val_over, 'RandomFore
```

```
On SMOTE Sample
Accuracy: 0.7665505226480837
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.81      0.78      6761
           1       0.79      0.73      0.76      6728

    accuracy                           0.77     13489
   macro avg       0.77      0.77      0.77     13489
weighted avg       0.77      0.77      0.77     13489

Confusion Matrix:
```



Nearest Neighbours

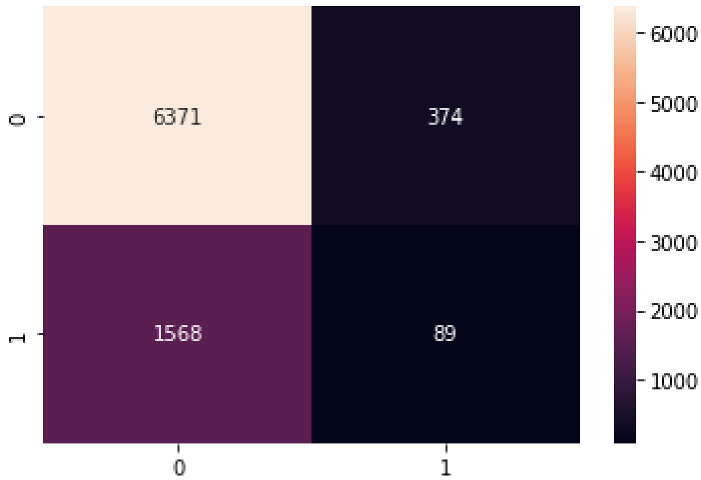
```
print("On entire Data")
model_apply(neighbours, X_train, y_train, X_val, y_val, 'KNeighbors')
```

```
On entire Data
Accuracy: 0.7688645560580815
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.94      0.87      6745
           1       0.19      0.05      0.08      1657

    accuracy                           0.77      8402
   macro avg       0.50      0.50      0.48      8402
weighted avg       0.68      0.77      0.71      8402

Confusion Matrix:
```



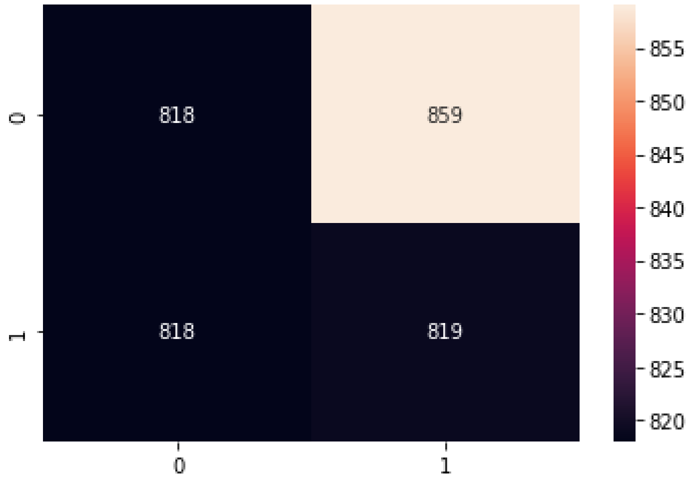
```
print("On Random Under Sample")
model_apply(neighbours, X_train_under, y_train_under, X_val_under, y_val_under, 'KNeighbor
```

```
On Random Under Sample
Accuracy: 0.4939649969824985
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.49      0.49      1677
           1       0.49      0.50      0.49      1637

    accuracy                           0.49      3314
   macro avg       0.49      0.49      0.49      3314
weighted avg       0.49      0.49      0.49      3314

Confusion Matrix:
```




```
print("On SMOTE Sample")
model_apply(neighbours, X_train_over, y_train_over, X_val_over, y_val_over, 'KNeighborsOve
```

```
On SMOTE Sample
Accuracy: 0.6706946400770999
Classification Report:
              precision    recall  f1-score   support
```

Neural Networks

```
def train_neural_networks(model, X_training, y_training, X_validation, y_validation, learr
  Adam(learning_rate=learning_rate)
  model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=[metrics.Recall(), metrics.Precision()]
  )
  return model.fit(X_training, y_training, validation_data=(X_validation, y_validation), e
```

```
def show_and_save_model(model, name, X_validation, y_validation):
  y_pred = model.predict(X_validation) > 0.5

  print("Accuracy: {}".format(accuracy_score(y_validation, y_pred)))

  print("Classification Report: \n{}".format(classification_report(y_validation, y_pred)))

  conf_matrix = confusion_matrix(y_validation, y_pred)
  print("Confusion Matrix: ")
  sns.heatmap(conf_matrix, annot=True, fmt='d')

  filename = "model/{}.pkl".format(name)
  pickle.dump(model, open(filename, 'wb'))
```

```
neg_under, pos_under = np.bincount(y_train_under)
```

```
neg_over, pos_over = np.bincount(y_train_over)
```

```
initial_bias_under = np.log([pos_under/neg_under])
initial_bias_under
```

```
array([0.00603502])
```

```
initial_bias_over = np.log([pos_over/neg_over])
initial_bias_over
```

```
array([0.00122324])
```

On Random Under Sample

```
model = Sequential()
```

```
model.add(Dense(128, input_shape=(11,), activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid', bias_initializer=Constant(initial_bias_under)))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1536

 dense_1 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 26,369
Trainable params: 26,369
Non-trainable params: 0
_________________________________________________________________
```

```
hist_us = train_neural_networks(model, X_train_under, y_train_under, X_val_under, y_val_ur
```

```
Epoch 1/25
884/884 [==============================] - 5s 4ms/step - loss: 0.6969 - recall: 0.52
Epoch 2/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6945 - recall: 0.51
Epoch 3/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6933 - recall: 0.58
Epoch 4/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6931 - recall: 0.61
Epoch 5/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6929 - recall: 0.56
Epoch 6/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6918 - recall: 0.63
Epoch 7/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6925 - recall: 0.63
Epoch 8/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6920 - recall: 0.60
Epoch 9/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6926 - recall: 0.58
Epoch 10/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6919 - recall: 0.53
Epoch 11/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6911 - recall: 0.57
Epoch 12/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6909 - recall: 0.58
Epoch 13/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6908 - recall: 0.60
Epoch 14/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6905 - recall: 0.54
Epoch 15/25
```

```
884/884 [==============================] - 3s 3ms/step - loss: 0.6898 - recall: 0.60
Epoch 16/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6897 - recall: 0.55
Epoch 17/25
884/884 [==============================] - 5s 6ms/step - loss: 0.6889 - recall: 0.54
Epoch 18/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6883 - recall: 0.57
Epoch 19/25
884/884 [==============================] - 3s 4ms/step - loss: 0.6871 - recall: 0.54
Epoch 20/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6875 - recall: 0.55
Epoch 21/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6859 - recall: 0.52
Epoch 22/25
884/884 [==============================] - 4s 4ms/step - loss: 0.6857 - recall: 0.52
Epoch 23/25
884/884 [==============================] - 3s 3ms/step - loss: 0.6853 - recall: 0.51
Epoch 24/25
884/884 [==============================] - 5s 6ms/step - loss: 0.6835 - recall: 0.54
Epoch 25/25
884/884 [==============================] - 6s 7ms/step - loss: 0.6823 - recall: 0.52
```

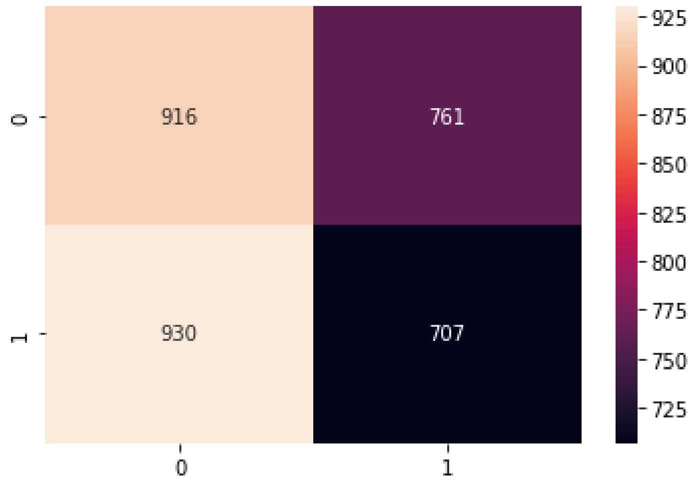
```
show_and_save_model(model, 'TwoLayerBiasUnder', X_val_under, y_val_under)
```

```
104/104 [==============================] - 0s 2ms/step
Accuracy: 0.489740494870247464
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.55      0.52      1677
           1       0.48      0.43      0.46      1637

    accuracy                           0.49      3314
   macro avg       0.49      0.49      0.49      3314
weighted avg       0.49      0.49      0.49      3314

Confusion Matrix:
```



On SMOTE Sample

```
model = Sequential()
```

```
model.add(Dense(256, input_shape=(11,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid', bias_initializer=Constant(initial_bias_over)))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 256)               3072

 dropout_8 (Dropout)         (None, 256)               0

 dense_23 (Dense)            (None, 256)               65792

 dropout_9 (Dropout)         (None, 256)               0

 dense_24 (Dense)            (None, 128)               32896

 dense_25 (Dense)            (None, 128)               16512

 dense_26 (Dense)            (None, 1)                 129

=================================================================
Total params: 118,401
Trainable params: 118,401
Non-trainable params: 0
_________________________________________________________________
```

```
eural_networks(model,·X_train_over,·y_train_over,·X_val_over,·y_val_over,·learning_rate=0.
```

```
===========] - 7s 7ms/step - loss: 0.5933 - recall_6: 0.6315 - precision_6: 0.7094

===========] - 7s 6ms/step - loss: 0.5935 - recall_6: 0.6389 - precision_6: 0.7035

===========] - 7s 7ms/step - loss: 0.5935 - recall_6: 0.6314 - precision_6: 0.7081

===========] - 7s 7ms/step - loss: 0.5930 - recall_6: 0.6304 - precision_6: 0.7115

===========] - 7s 7ms/step - loss: 0.5929 - recall_6: 0.6351 - precision_6: 0.7097

===========] - 7s 7ms/step - loss: 0.5926 - recall_6: 0.6361 - precision_6: 0.7080

===========] - 8s 7ms/step - loss: 0.5917 - recall_6: 0.6473 - precision_6: 0.7034

===========] - 9s 8ms/step - loss: 0.5911 - recall_6: 0.6463 - precision_6: 0.7048

===========] - 8s 8ms/step - loss: 0.5902 - recall_6: 0.6436 - precision_6: 0.7096

===========] - 7s 7ms/step - loss: 0.5908 - recall_6: 0.6450 - precision_6: 0.7062

===========] - 7s 7ms/step - loss: 0.5906 - recall_6: 0.6461 - precision_6: 0.7095
```

```
===========] - 7s 7ms/step - loss: 0.5901 - recall_6: 0.6461 - precision_6: 0.7047
===========] - 8s 7ms/step - loss: 0.5887 - recall_6: 0.6508 - precision_6: 0.7057
===========] - 7s 7ms/step - loss: 0.5885 - recall_6: 0.6478 - precision_6: 0.7084
===========] - 7s 7ms/step - loss: 0.5898 - recall_6: 0.6457 - precision_6: 0.7074
===========] - 7s 7ms/step - loss: 0.5885 - recall_6: 0.6398 - precision_6: 0.7127
===========] - 7s 7ms/step - loss: 0.5879 - recall_6: 0.6472 - precision_6: 0.7123
===========] - 7s 7ms/step - loss: 0.5881 - recall_6: 0.6561 - precision_6: 0.7066
===========] - 7s 7ms/step - loss: 0.5883 - recall_6: 0.6398 - precision_6: 0.7085
===========] - 7s 7ms/step - loss: 0.5870 - recall_6: 0.6399 - precision_6: 0.7117
===========] - 8s 7ms/step - loss: 0.5868 - recall_6: 0.6420 - precision_6: 0.7134
===========] - 7s 7ms/step - loss: 0.5863 - recall_6: 0.6365 - precision_6: 0.7157
===========] - 7s 7ms/step - loss: 0.5874 - recall_6: 0.6350 - precision_6: 0.7127
===========] - 7s 7ms/step - loss: 0.5862 - recall_6: 0.6456 - precision_6: 0.7118
===========] - 7s 7ms/step - loss: 0.5866 - recall_6: 0.6414 - precision_6: 0.7128
===========] - 7s 7ms/step - loss: 0.5873 - recall_6: 0.6453 - precision_6: 0.7115
===========] - 7s 7ms/step - loss: 0.5865 - recall_6: 0.6483 - precision_6: 0.7123
===========] - 7s 7ms/step - loss: 0.5860 - recall_6: 0.6488 - precision_6: 0.7115
===========] - 7s 7ms/step - loss: 0.5847 - recall_6: 0.6466 - precision_6: 0.7140
```

```
hist_os = train_neural_networks(model, X_train_over, y_train_over, X_val_over, y_val_over,
```

```
loss: 0.5805 - recall_7: 0.6507 - precision_7: 0.7181 - val_loss: 0.5858 - val_reca
loss: 0.5813 - recall_7: 0.6527 - precision_7: 0.7160 - val_loss: 0.5833 - val_reca
loss: 0.5808 - recall_7: 0.6438 - precision_7: 0.7176 - val_loss: 0.5845 - val_reca
loss: 0.5820 - recall_7: 0.6412 - precision_7: 0.7196 - val_loss: 0.5856 - val_reca
loss: 0.5817 - recall_7: 0.6479 - precision_7: 0.7172 - val_loss: 0.5827 - val_reca
loss: 0.5804 - recall_7: 0.6523 - precision_7: 0.7159 - val_loss: 0.5831 - val_reca
loss: 0.5801 - recall_7: 0.6536 - precision_7: 0.7156 - val_loss: 0.5836 - val_reca
loss: 0.5804 - recall_7: 0.6540 - precision_7: 0.7149 - val_loss: 0.5822 - val_reca
loss: 0.5800 - recall_7: 0.6500 - precision_7: 0.7187 - val_loss: 0.5836 - val_reca
loss: 0.5792 - recall_7: 0.6614 - precision_7: 0.7148 - val_loss: 0.5829 - val_reca
loss: 0.5797 - recall_7: 0.6542 - precision_7: 0.7174 - val_loss: 0.5843 - val_reca
```

```
loss: 0.5797 - recall_7: 0.6542 - precision_7: 0.7174 - val_loss: 0.5843 - val_reca
loss: 0.5800 - recall_7: 0.6542 - precision_7: 0.7168 - val_loss: 0.5817 - val_reca
loss: 0.5809 - recall_7: 0.6507 - precision_7: 0.7160 - val_loss: 0.5857 - val_reca
loss: 0.5800 - recall_7: 0.6597 - precision_7: 0.7143 - val_loss: 0.5824 - val_reca
loss: 0.5796 - recall_7: 0.6490 - precision_7: 0.7202 - val_loss: 0.5837 - val_reca
loss: 0.5802 - recall_7: 0.6487 - precision_7: 0.7207 - val_loss: 0.5858 - val_reca
loss: 0.5786 - recall_7: 0.6545 - precision_7: 0.7166 - val_loss: 0.5827 - val_reca
loss: 0.5795 - recall_7: 0.6471 - precision_7: 0.7203 - val_loss: 0.5855 - val_reca
loss: 0.5798 - recall_7: 0.6460 - precision_7: 0.7219 - val_loss: 0.5826 - val_reca
loss: 0.5791 - recall_7: 0.6481 - precision_7: 0.7233 - val_loss: 0.5824 - val_reca
loss: 0.5793 - recall_7: 0.6536 - precision_7: 0.7176 - val_loss: 0.5826 - val_reca
loss: 0.5779 - recall_7: 0.6517 - precision_7: 0.7200 - val_loss: 0.5826 - val_reca
loss: 0.5778 - recall_7: 0.6591 - precision_7: 0.7187 - val_loss: 0.5830 - val_reca
loss: 0.5779 - recall_7: 0.6544 - precision_7: 0.7175 - val_loss: 0.5832 - val_reca
loss: 0.5800 - recall_7: 0.6506 - precision_7: 0.7172 - val_loss: 0.5818 - val_reca
loss: 0.5779 - recall_7: 0.6526 - precision_7: 0.7209 - val_loss: 0.5816 - val_reca
loss: 0.5787 - recall_7: 0.6581 - precision_7: 0.7180 - val_loss: 0.5854 - val_reca
loss: 0.5777 - recall_7: 0.6601 - precision_7: 0.7139 - val_loss: 0.5836 - val_reca
loss: 0.5784 - recall_7: 0.6584 - precision_7: 0.7138 - val_loss: 0.5821 - val_reca
```

```
show_and_save_model(model, 'TwoLayerBiasOver', X_val_over, y_val_over)
```

```
422/422 [==============================] - 1s 2ms/step
Accuracy: 0.6959003632589518
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.67      0.69      6761
           1       0.69      0.72      0.70      6728

    accuracy                           0.70     13489
```

Prediction on Test Model

```
test_data.drop('ID', axis=1, inplace=True)
```

```
categorical_cols = [col for col in test_data.columns if test_data[col].dtype=="O"]

label_encoder = LabelEncoder()

for col in categorical_cols:
    test_data[col] = label_encoder.fit_transform(test_data[col])
```

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Income             28000 non-null  int64
 1   Age                28000 non-null  int64
 2   Experience         28000 non-null  int64
 3   Married/Single     28000 non-null  int64
 4   House_Ownership    28000 non-null  int64
 5   Car_Ownership      28000 non-null  int64
 6   Profession         28000 non-null  int64
 7   CITY               28000 non-null  int64
 8   STATE              28000 non-null  int64
 9   CURRENT_JOB_YRS    28000 non-null  int64
 10  CURRENT_HOUSE_YRS  28000 non-null  int64
dtypes: int64(11)
memory usage: 2.3 MB
```

```
cols = test_data.columns
```

```
test_data = scaler.transform(test_data.to_numpy())
test_data = pd.DataFrame(test_data, columns=list(cols))
test_data
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	0.837459	0.531939	1.506277	0.332830	0.283607	-0.6585
1	-1.312342	-1.460938	-0.811160	0.332830	0.283607	-0.6585
2	1.362288	0.004413	0.347558	0.332830	0.283607	-0.6585
3	-1.058526	-0.054201	-0.149036	-3.004535	0.283607	1.5184
4	-1.730457	-1.460938	1.340745	0.332830	0.283607	1.5184
...	...	...	...	...	...	
27995	1.729099	0.414711	0.513089	0.332830	0.283607	-0.6585
27996	-0.719829	-0.171429	-0.149036	0.332830	0.283607	-0.6585
27997	1.077325	-1.519552	-0.811160	0.332830	0.283607	-0.6585
27998	1.561620	0.063027	0.513089	0.332830	0.283607	1.5184
27999	1.483733	-0.464499	-0.149036	0.332830	0.283607	-0.6585

28000 rows × 11 columns

```
def load_and_test(filename):
  model = pickle.load(open(filename, 'rb'))
  y_pred = model.predict(test_data.iloc[:, : 11])
  return y_pred
```

Logistic Regression

```
test_data['lr_predict'] = list(load_and_test('model/LogisticRegressionOver.pkl'))
test_data
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	0.837459	0.531939	1.506277	0.332830	0.283607	-0.6585
1	-1.312342	-1.460938	-0.811160	0.332830	0.283607	-0.6585

```
test_data['lr_predict'].value_counts()
```

```
1    14850
0    13150
Name: lr_predict, dtype: int64
```

Decision Tree

27996	-0.719829	-0.171429	-0.149036	0.332830	0.283607	-0.6585

```
test_data['dt_predict'] = list(load_and_test('model/DecisionTreeOver.pkl'))
test_data
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	0.837459	0.531939	1.506277	0.332830	0.283607	-0.6585
1	-1.312342	-1.460938	-0.811160	0.332830	0.283607	-0.6585
2	1.362288	0.004413	0.347558	0.332830	0.283607	-0.6585
3	-1.058526	-0.054201	-0.149036	-3.004535	0.283607	1.5184
4	-1.730457	-1.460938	1.340745	0.332830	0.283607	1.5184
...	...	...	...	...	...	
27995	1.729099	0.414711	0.513089	0.332830	0.283607	-0.6585
27996	-0.719829	-0.171429	-0.149036	0.332830	0.283607	-0.6585
27997	1.077325	-1.519552	-0.811160	0.332830	0.283607	-0.6585
27998	1.561620	0.063027	0.513089	0.332830	0.283607	1.5184
27999	1.483733	-0.464499	-0.149036	0.332830	0.283607	-0.6585

28000 rows × 13 columns

```
test_data['dt_predict'].value_counts()
```

```
1    16859
0    11141
Name: dt_predict, dtype: int64
```

Random Forest

```
test_data['rf_predict'] = list(load_and_test('model/RandomForestOver.pkl'))
test_data
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	0.837459	0.531939	1.506277	0.332830	0.283607	-0.6585
1	-1.312342	-1.460938	-0.811160	0.332830	0.283607	-0.6585
2	1.362288	0.004413	0.347558	0.332830	0.283607	-0.6585
3	-1.058526	-0.054201	-0.149036	-3.004535	0.283607	1.5184
4	-1.730457	-1.460938	1.340745	0.332830	0.283607	1.5184
...	...	...	...	...	...	
27995	1.729099	0.414711	0.513089	0.332830	0.283607	-0.6585
27996	-0.719829	-0.171429	-0.149036	0.332830	0.283607	-0.6585
27997	1.077325	-1.519552	-0.811160	0.332830	0.283607	-0.6585
27998	1.561620	0.063027	0.513089	0.332830	0.283607	1.5184
27999	1.483733	-0.464499	-0.149036	0.332830	0.283607	-0.6585

28000 rows × 14 columns

```
test_data['rf_predict'].value_counts()
```

```
0    15453
1    12547
Name: rf_predict, dtype: int64
```

K Nearest Neighbours

```
test_data['knn_predict'] = list(load_and_test('model/KNeighborsOver.pkl'))
test_data
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	0.837459	0.531939	1.506277	0.332830	0.283607	-0.6585
1	-1.312342	-1.460938	-0.811160	0.332830	0.283607	-0.6585
2	1.362288	0.004413	0.347558	0.332830	0.283607	-0.6585
3	-1.058526	-0.054201	-0.149036	-3.004535	0.283607	1.5184
4	-1.730457	-1.460938	1.340745	0.332830	0.283607	1.5184
...	...	...	...	...	...	
27995	1.729099	0.414711	0.513089	0.332830	0.283607	-0.6585

```
test_data['knn_predict'].value_counts()
```

```
0    15955
1    12045
Name: knn_predict, dtype: int64
```

27999	1.483733	0.464499	0.149036	0.332830	0.283607	0.6585

Two Layer with Bias Network

```
test_data['b_predict'] = [1 if item else 0 for sublist in list(load_and_test('model/TwoLay
test_data
```

```
875/875 [==============================] - 2s 2ms/step
```

	Income	Age	Experience	Married/Single	House_Ownership	Car_Ownersh
0	0.837459	0.531939	1.506277	0.332830	0.283607	-0.6585
1	-1.312342	-1.460938	-0.811160	0.332830	0.283607	-0.6585
2	1.362288	0.004413	0.347558	0.332830	0.283607	-0.6585
3	-1.058526	-0.054201	-0.149036	-3.004535	0.283607	1.5184
4	-1.730457	-1.460938	1.340745	0.332830	0.283607	1.5184
...	...	...	...	...	...	
27995	1.729099	0.414711	0.513089	0.332830	0.283607	-0.6585
27996	-0.719829	-0.171429	-0.149036	0.332830	0.283607	-0.6585
27997	1.077325	-1.519552	-0.811160	0.332830	0.283607	-0.6585
27998	1.561620	0.063027	0.513089	0.332830	0.283607	1.5184
27999	1.483733	-0.464499	-0.149036	0.332830	0.283607	-0.6585

28000 rows × 16 columns

```
test_data['b_predict'].value_counts()
```

```
1    14700
0    13300
Name: b_predict, dtype: int64
```

Saving Output Dataset

```
predicted_data = pd.DataFrame({
    'Id': list(test_data.index + 1)
})
```

```
predicted_data = pd.concat([predicted_data, test_data.iloc[:, 11:]], axis=1)
predicted_data
```

	Id	lr_predict	dt_predict	rf_predict	knn_predict	b_predict
0	1	0	1	1	0	1
1	2	1	1	1	1	1
2	3	0	1	0	0	1
3	4	0	0	0	1	1
4	5	0	1	0	0	0
...	...	...	...	...	...	...
27995	27996	1	1	1	1	1
27996	27997	0	1	0	0	0
27997	27998	1	0	0	1	1
27998	27999	0	1	0	0	0
27999	28000	1	1	1	1	1

28000 rows × 6 columns

```
predicted_data.to_csv('data/Prediction on Test.csv')
```