**APPENDICES**

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*from sklearn.preprocessing import OneHotEncoder*

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

*from imblearn.over\_sampling import SMOTE*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.model\_selection import GridSearchCV*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.neighbors import KNeighborsClassifier*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.ensemble import RandomForestClassifier*

*from sklearn.metrics import accuracy\_score, confusion\_matrix*

*import warnings*

*warnings.filterwarnings('ignore')*

**Data Loading**

*d = pd.read\_csv("Hotel Reservations.csv")*

*d.head()*

**Data Preprocessing**

***Checking for the missing values***

*d.info()*

*d.columns.isnull()*

**Remove UnWanted Columns**

*d["type\_of\_meal\_plan"].unique()*

*d["room\_type\_reserved"].unique()*

*d["arrival\_year"].unique()*

*d["arrival\_year"].unique()*

*d["market\_segment\_type"].unique()*

*d = d.drop(columns = ["Booking\_ID","type\_of\_meal\_plan"],axis=1)*

*d*

**One-Hot Encoding**

*import pandas as pd*

*from sklearn.preprocessing import OneHotEncoder*

*one\_hot\_encoder = OneHotEncoder(sparse=False, drop=None*

*one\_hot\_encoded\_features = one\_hot\_encoder.fit\_transform(d[['room\_type\_reserved', 'market\_segment\_type']])*

*encoded\_column\_names = one\_hot\_encoder.get\_feature\_names\_out(['room\_type\_reserved', 'market\_segment\_type'])*

*data\_encoded = pd.DataFrame(one\_hot\_encoded\_features, columns=encoded\_column\_names)*

*data\_encoded = pd.concat([d, data\_encoded], axis=1)*

*data\_encoded.drop(['room\_type\_reserved', 'market\_segment\_type'], axis=1, inplace=True)*

*data\_encoded.info()*

*data\_encoded.head()*

**Poping target column inbetween the columns and adding it to the last**

*booking\_status\_col = data\_encoded.pop("booking\_status")*

*data\_encoded["booking\_status"] = booking\_status\_col*

*data\_encoded*

*data\_encoded.info()*

*data\_encoded["booking\_status"].value\_counts()*

**Data Visualization**

**Plotting the object columns counts using bar graph**

*object\_cols = [col for col in d.columns if d[col].dtype == "object"]*

*fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(19, 6))*

*for col, ax in zip(object\_cols, (ax1, ax2, ax3)):*

*d[col].value\_counts().plot(kind="bar", ax=ax, color="SkyBlue")*

*ax.set\_title(col)*

*ax.set\_ylabel("Count")*

*print("Object Columns: ",object\_cols)*

*plt.show()*

**Create a dictionary to store the counts of booking\_status for each room\_type\_reserved**

*counts = {}*

*for room\_type, booking\_status in zip(d["room\_type\_reserved"], d["booking\_status"]):*

*if room\_type not in counts:*

*counts[room\_type] = {}*

*if booking\_status not in counts[room\_type]:*

*counts[room\_type][booking\_status] = 0*

*counts[room\_type][booking\_status] += 1*

*print(counts)*

*# Create a NumPy array to store the counts of booking\_status for each room\_type\_reserved*

*bar\_data = np.array([[counts[room\_type]["Canceled"], counts[room\_type]["Not\_Canceled"]] for room\_type in counts])*

*bar\_keys = list(counts.keys())*

*print(bar\_data)*

*print(bar\_keys)*

**Grouped bar plot for cancellation status by room\_type\_reserved**

*room\_type\_reserved\_col = (tuple(bar\_keys))*

*counts = {*

*'Cancelled': tuple([bar\_data[i][0] for i in range(bar\_data.shape[0])]),*

*'Not Cancelled': tuple([bar\_data[i][1] for i in range(bar\_data.shape[0])]),*

*}*

*x = np.arange(len(room\_type\_reserved\_col)) # the label locations*

*width = 0.45 # the width of the bars*

*multiplier = 0*

*fig, ax = plt.subplots(layout='constrained')*

*for attribute, measurement in counts.items():*

*offset = width \* multiplier*

*rects = ax.bar(x + offset, measurement, width, label=attribute)*

*ax.bar\_label(rects, padding=5)*

*multiplier += 1*

*ax.set\_xticks(x + width / 2)*

*ax.set\_xticklabels(room\_type\_reserved\_col, rotation=45)*

*ax.set\_ylabel('Counts')*

*ax.set\_title('Cancellation Status by room\_type\_reserved')*

*ax.legend(loc='upper left', ncols=3)*

*ax.set\_ylim(0, 25000)*

*plt.show()*

*non\_float\_cols = [col for col in d.columns if (d[col].dtype != "object") and (col not in ("lead\_time", "avg\_price\_per\_room","no\_of\_previous\_bookings\_not\_canceled","arrival\_date"))]*

*float\_cols = [col for col in d[["lead\_time", "avg\_price\_per\_room","no\_of\_previous\_bookings\_not\_canceled","arrival\_date"]]]*

**Histogram for Average Price per room**

*fig, axes = plt.subplots(1, 1, figsize=(10, 5))*

*d["avg\_price\_per\_room"].plot(kind="hist", ax=axes, color="blue")*

*axes.set\_title("Average Price Per Room")*

*axes.set\_ylabel("Count")*

*# Set the x-axis limits to avoid clumbiness*

*axes.set\_xticks(np.arange(0, axes.get\_xlim()[1], 40))*

*axes.bar\_label(axes.containers[0])*

*plt.show()*

**Histogram for Lead Time**

*fig, axes = plt.subplots(1, 1, figsize=(20, 10))*

*d["lead\_time"].plot(kind="hist", ax=axes, color="blue")*

*axes.set\_title("Lead Time")*

*axes.set\_ylabel("Count")*

*axes.set\_xticks(np.arange(0, axes.get\_xlim()[1], 30))*

*plt.xticks(rotation=40,size=12)*

*axes.bar\_label(axes.containers[0])*

*plt.show()*

**Plotting the int data type columns**

*fig, axes = plt.subplots(3,2 , figsize=(15, 15))*

*for col, ax in zip(non\_float\_cols[0:6], axes.flatten()):*

*d[col].value\_counts().plot(kind="bar", ax=ax, color="blue")*

*ax.set\_title(col)*

*ax.set\_ylabel("Count")*

*ax.bar\_label(ax.containers[0])*

*plt.show()*

*fig, axes = plt.subplots(2,2 , figsize=(15, 15))*

*for col, ax in zip(non\_float\_cols[6:10], axes.flatten()):*

*d[col].value\_counts().plot(kind="bar", ax=ax, color="blue")*

*ax.set\_title(col)*

*ax.set\_ylabel("Count")*

*ax.bar\_label(ax.containers[0])*

*plt.show()*

**Bar Plot for the target variable**

*fig, ax = plt.subplots(figsize=(15, 10))*

*d['booking\_status'].value\_counts().plot(kind="bar", ax=ax, color=["green","red"])*

*ax.set\_title("Booking Status Distribution", size = 25)*

*plt.xticks(rotation=0,size=20)*

*ax.set\_ylabel("Count", size=25)*

*plt.yticks(size=20)*

*ax.bar\_label(ax.containers[0], size=20)*

*plt.show()*

**Data Separation**

*train\_data, test\_data, train\_target, test\_target = train\_test\_split(*

*data\_encoded.iloc[:, :-1], data\_encoded.iloc[:, -1], test\_size=0.15, random\_state=42*

*)*

*val\_data, test\_data, val\_target, test\_target = train\_test\_split(*

*test\_data, test\_target, test\_size=0.20, random\_state=42*

*)*

**# These will be used for Undersampling later.**

*train\_data\_us = train\_data*

*test\_data\_us = test\_data*

*val\_data\_us = val\_data*

*val\_target\_us = val\_target*

*train\_target\_us = train\_target*

*test\_target\_us = test\_target*

**Split for over sampled data training**

*print("train\_data.shape: ",train\_data.shape)*

*print("train\_target.shape: ",train\_target.shape)*

*print("val\_data,shape: ", val\_data.shape)*

*print("val\_target,shape: ", val\_target.shape)*

*print("test\_data.shape: ",test\_data.shape)*

*print("test\_target.shape: ",test\_target.shape)*

*train\_data.head()*

**Over Sampling**

*# Apply SMOTE for oversampling the minority class*

*smote = SMOTE(random\_state=42)*

*train\_data\_osampled, train\_target\_osampled = smote.fit\_resample(train\_data, train\_target)*

*train\_data\_osampled.shape, train\_target\_osampled.shape*

*train\_data\_osampled*

*train\_target\_osampled.head()*

**Standardization on oversampled data**

*scaler = StandardScaler()*

*train\_data\_osampled = scaler.fit\_transform(train\_data\_osampled)*

*test\_data\_osampled = scaler.transform(test\_data)*

*val\_data\_osampled = scaler.transform(val\_data)*

*train\_data\_osampled*

*test\_data\_osampled*

*val\_data\_osampled*

**Machine Learning algorithms on Over-Sampled data**

**Logistic Regression**

*from sklearn.metrics import classification\_report*

*# Logistic Regression*

*logistic\_regression = LogisticRegression()*

*# Train the Logistic Regression classifier*

*logistic\_regression.fit(train\_data\_osampled, train\_target\_osampled)*

*# Make predictions on the train and test data*

*logistic\_regression\_train\_pred = logistic\_regression.predict(train\_data\_osampled)*

*logistic\_regression\_test\_pred = logistic\_regression.predict(val\_data\_osampled)*

*# Print performance metrics for Logistic Regression*

*print('Logistic Regression Classifier:')*

*print('Training Accuracy:', accuracy\_score(train\_target\_osampled, logistic\_regression\_train\_pred))*

*print('Validation Accuracy:', accuracy\_score(val\_target, logistic\_regression\_test\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(val\_target, logistic\_regression\_test\_pred))*

*Log\_report = classification\_report(val\_target, logistic\_regression\_test\_pred)*

*print(Log\_report)*

*# Use the trained model to make predictions on the unseen data*

*unseen\_pred = logistic\_regression.predict(test\_data\_osampled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(test\_target, unseen\_pred))*

**KNN**

**GridSearch for KNN**

*param\_grid = {*

*'n\_neighbors': [3, 5, 7, 9, 11 ],*

*'weights': ['uniform', 'distance'],*

*'algorithm': ['ball\_tree', 'kd\_tree', 'brute']*

*}*

*# Create a KNN classifier*

*knn = KNeighborsClassifier()*

*# Perform grid search to find the best parameters*

*grid\_search = GridSearchCV(estimator=knn, param\_grid=param\_grid, scoring='accuracy', cv=5, n\_jobs=-1, verbose=1)*

*grid\_search.fit(train\_data\_osampled, train\_target\_osampled)*

*best\_params = grid\_search.best\_params\_*

*print("Best Parameters:", best\_params)*

***KNN with best parameters***

*knn\_clf = KNeighborsClassifier(\*\*best\_params)*

*knn\_clf.fit(train\_data\_osampled, train\_target\_osampled)*

*train\_pred = knn\_clf.predict(train\_data\_osampled)*

*val\_pred = knn\_clf.predict(val\_data\_osampled)*

*print('K-Nearest Neighbors Classifier:')*

*print('Training Accuracy:', accuracy\_score(train\_target\_osampled, train\_pred))*

*print('Validation Accuracy:', accuracy\_score(val\_target, val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(val\_target, val\_pred))*

*knn\_report = classification\_report(val\_target, val\_pred)*

*print(knn\_report)*

*# Use the trained model to make predictions on the unseen data*

*unseen\_pred = knn\_clf.predict(test\_data\_osampled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(test\_target, unseen\_pred))*

**Decision Tree**

***Grid Search***

*dt\_params = {*

*'criterion': ['gini', 'entropy'],*

*'max\_depth': [None, 5, 10, 15],*

*'min\_samples\_split': [2, 5, 10],*

*'min\_samples\_leaf': [1, 2, 5]*

*}*

*dt\_clf = DecisionTreeClassifier(random\_state=42)*

*dt\_grid\_search = GridSearchCV(dt\_clf, dt\_params, cv=5, n\_jobs=-1, verbose=3)*

*dt\_grid\_search.fit(train\_data\_osampled, train\_target\_osampled)*

*dt\_best\_params = dt\_grid\_search.best\_params\_*

*dt\_best\_params*

***decision tree with best params***

*dt\_clf = DecisionTreeClassifier(\*\*dt\_best\_params, random\_state=42)*

*dt\_clf.fit(train\_data\_osampled, train\_target\_osampled)*

*train\_pred = dt\_clf.predict(train\_data\_osampled)*

*val\_pred = dt\_clf.predict(val\_data\_osampled)*

*print('Decision Tree Classifier:')*

*print('Training Accuracy:', accuracy\_score(train\_target\_osampled, train\_pred))*

*print('Validation Accuracy:', accuracy\_score(val\_target, val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(val\_target, val\_pred))*

*dt\_report = classification\_report(val\_target, val\_pred)*

*print(dt\_report)*

*unseen\_pred = dt\_clf.predict(test\_data\_osampled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(test\_target, unseen\_pred))*

### Random Forest

***Grid Search***

*rf\_params = {*

*'n\_estimators': [100, 300, 500],*

*'criterion': ['gini', 'entropy'],*

*'max\_depth': [None, 5, 10, 15],*

*'min\_samples\_split': [2, 5, 10],*

*'min\_samples\_leaf': [1, 2, 5]*

*}*

*rf\_clf = RandomForestClassifier(random\_state=42)*

*rf\_grid\_search = GridSearchCV(rf\_clf, rf\_params, cv=5, n\_jobs=-1, verbose=1)*

*rf\_grid\_search.fit(train\_data\_osampled, train\_target\_osampled)*

*rf\_best\_params = rf\_grid\_search.best\_params\_*

***Random Forest with Best Parameters***

*rf\_clf = RandomForestClassifier(\*\*rf\_best\_params, random\_state=42)*

*rf\_clf.fit(train\_data\_osampled, train\_target\_osampled)*

*train\_pred = rf\_clf.predict(train\_data\_osampled)*

*val\_pred = rf\_clf.predict(val\_data\_osampled)*

*print('Random Forest Classifier:')*

*print('Train Accuracy:', accuracy\_score(train\_target\_osampled, train\_pred))*

*print('Test Accuracy:', accuracy\_score(val\_target, val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(val\_target, val\_pred))*

*rf\_report = classification\_report(val\_target, val\_pred)*

*print(rf\_report)*

*# Use the trained model to make predictions on the unseen data*

*unseen\_pred = rf\_clf.predict(test\_data\_osampled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(test\_target, unseen\_pred))*

***Random forest feature importance’s***

*feat\_importances = pd.Series(rf\_clf.feature\_importances\_, index=data\_encoded.columns[:-1])*

*feat\_importances.nlargest(10).plot(kind='barh')*

*plt.title('Feature Importances')*

*plt.show()*

**Under-Sampling**

*# These will be used for Undersampling later.*

*train\_data\_us = train\_data*

*test\_data\_us = test\_data*

*val\_data\_us = val\_data*

*val\_target\_us = val\_target*

*train\_target\_us = train\_target*

*test\_target\_us = test\_target*

*#unseen\_data\_us = unseen\_data*

*# variable copies for undersampling*

*under\_train\_data = train\_data\_us*

*under\_train\_target = train\_target\_us*

*under\_val\_data = val\_data\_us*

*under\_val\_target = val\_target\_us*

*under\_test\_data = test\_data\_us*

*under\_test\_target = test\_target\_us*

***Random Under Sampling (RUS)***

*import random*

*from imblearn.under\_sampling import RandomUnderSampler*

*def undersample(train\_data, train\_target):*

*rus = RandomUnderSampler(random\_state=0)*

*train\_data\_resampled, train\_target\_resampled = rus.fit\_resample(train\_data, train\_target)*

*return train\_data\_resampled, train\_target\_resampled*

*train\_data\_usampled, train\_target\_usampled = undersample(under\_train\_data, under\_train\_target)*

*print(train\_data\_usampled.shape)*

*print(train\_target\_usampled.shape)*

*print("train\_data\_usampled.shape: ",train\_data\_usampled.shape)*

*print("train\_target\_usampled.shape: ",train\_target\_usampled.shape)*

*print("under\_val\_data: ",under\_val\_data.shape)*

*print("under\_val\_target: ",under\_val\_target.shape)*

*print("under\_test\_data: ",under\_test\_data.shape)*

*print("under\_test\_target: ",under\_test\_target.shape)*

***Standardization of Undersampled Data***

*scaler = StandardScaler()*

*train\_data\_usampled\_scaled = scaler.fit\_transform(train\_data\_usampled)*

*val\_data\_scaled = scaler.transform(under\_val\_data)*

*test\_data\_scaled = scaler.transform(under\_test\_data)*

***Logistic Regression on Undersampled Data***

*logistic\_regression = LogisticRegression()*

*logistic\_regression.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*logistic\_regression\_train\_pred = logistic\_regression.predict(train\_data\_usampled\_scaled)*

*logistic\_regression\_val\_pred = logistic\_regression.predict(val\_data\_scaled)*

*print('Logistic Regression Classifier:')*

*print('Train Accuracy:', accuracy\_score(train\_target\_usampled, logistic\_regression\_train\_pred))*

*print('Test Accuracy:', accuracy\_score(under\_val\_target, logistic\_regression\_val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(under\_val\_target, logistic\_regression\_val\_pred))*

*rf\_report\_us = classification\_report(under\_val\_target, logistic\_regression\_val\_pred)*

*print(rf\_report\_us)*

*# Use the trained model to make predictions on the unseen data*

*unseen\_pred = logistic\_regression.predict(test\_data\_scaled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(under\_test\_target, unseen\_pred))*

***KNN on Undersampled Data¶***

***Grid Search***

*from sklearn.neighbors import KNeighborsClassifier*

*from sklearn.model\_selection import GridSearchCV*

*# Define the parameter grid*

*param\_grid = {*

*'n\_neighbors': [3, 5, 7, 9, 11 ],*

*'weights': ['uniform', 'distance'],*

*'algorithm': ['ball\_tree', 'kd\_tree', 'brute']*

*}*

*# Create a KNN classifier*

*knn = KNeighborsClassifier()*

*# Perform grid search to find the best parameters*

*grid\_search = GridSearchCV(estimator=knn, param\_grid=param\_grid, scoring='accuracy', cv=5, n\_jobs=-1, verbose=1)*

*grid\_search.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*beu\_params = grid\_search.best\_params\_*

*print("Best Parameters:", best\_params)*

***KNN with best parameters***

*knn\_clf = KNeighborsClassifier(\*\*best\_params)*

*knn\_clf.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*train\_pred = knn\_clf.predict(train\_data\_usampled\_scaled)*

*val\_pred = knn\_clf.predict(val\_data\_scaled)*

*print('K-Nearest Neighbors Classifier:')*

*print('Train Accuracy:', accuracy\_score(train\_target\_usampled, train\_pred))*

*print('Validation Accuracy:', accuracy\_score(under\_val\_target, val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(under\_val\_target, val\_pred))*

*knn\_report\_us = classification\_report(under\_val\_target, val\_pred)*

*print(knn\_report\_us)*

*unseen\_pred = knn\_clf.predict(test\_data\_scaled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(under\_test\_target, unseen\_pred))*

***Decision Tree CLassifier on Undersampled Data***

***Grid Search***

*dt\_params = {*

*'criterion': ['gini', 'entropy'],*

*'max\_depth': [None, 5, 10, 15],*

*'min\_samples\_split': [2, 5, 10],*

*'min\_samples\_leaf': [1, 2, 5]*

*}*

*dt\_clf = DecisionTreeClassifier(random\_state=42)*

*dt\_grid\_search = GridSearchCV(dt\_clf, dt\_params, cv=5, n\_jobs=-1, verbose=1)*

*dt\_grid\_search.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*dt\_best\_params = dt\_grid\_search.best\_params\_*

*print(dt\_best\_params)*

***Decision Tree with Best parameters***

*dt\_clf = DecisionTreeClassifier(\*\*dt\_best\_params, random\_state=42)*

*dt\_clf.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*train\_pred = dt\_clf.predict(train\_data\_usampled\_scaled)*

*val\_pred = dt\_clf.predict(val\_data\_scaled)*

*print('Decision Tree Classifier:')*

*print('Train Accuracy:', accuracy\_score(train\_target\_usampled, train\_pred))*

*print('Validation Accuracy:', accuracy\_score(under\_val\_target, val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(under\_val\_target, val\_pred))*

*rf\_report\_us = classification\_report(under\_val\_target, val\_pred)*

*print(rf\_report\_us)*

*# Use the trained model to make predictions on the unseen data*

*unseen\_pred = dt\_clf.predict(test\_data\_scaled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(under\_test\_target, unseen\_pred))*

***Random Forest Classifier on Undersampled Data¶***

***Grid Search***

*rf\_params = {*

*'n\_estimators': [100, 300, 500],*

*#n\_estimators : This is the number of trees you want to build before taking the maximum voting or averages of predictions.*

*#Higher number of trees give you better performance but makes your code slower.*

*'criterion': ['gini', 'entropy'],*

*'max\_depth': [None, 5, 10, 15],*

*'min\_samples\_split': [2, 5, 10],*

*'min\_samples\_leaf': [1, 2, 5]*

*}*

*rf\_clf = RandomForestClassifier(random\_state=42)*

*rf\_grid\_search = GridSearchCV(rf\_clf, rf\_params, cv=5, n\_jobs=-1, verbose=1)*

*rf\_grid\_search.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*rf\_best\_params = rf\_grid\_search.best\_params\_*

***Random Forest with Best parameters***

*# Train random forest classifier with best parameters and measure performance*

*rf\_clf = RandomForestClassifier(\*\*rf\_best\_params, random\_state=42)*

*rf\_clf.fit(train\_data\_usampled\_scaled, train\_target\_usampled)*

*train\_pred = rf\_clf.predict(train\_data\_usampled\_scaled)*

*val\_pred = rf\_clf.predict(val\_data\_scaled)*

*print('Random Forest Classifier:')*

*print('Train Accuracy:', accuracy\_score(train\_target\_usampled, train\_pred))*

*print('Validation Accuracy:', accuracy\_score(under\_val\_target, val\_pred))*

*print('Confusion Matrix:\n', confusion\_matrix(under\_val\_target, val\_pred))*

*rf\_report\_us = classification\_report(under\_val\_target, val\_pred)*

*print(rf\_report\_us)*

*# Use the trained model to make predictions on the unseen data*

*unseen\_pred = rf\_clf.predict(test\_data\_scaled)*

*print('Unseen Test Data Accuracy:', accuracy\_score(under\_test\_target, unseen\_pred))*

*# Plot feature importances for the random forest classifier with undersampled data*

*feat\_importances = pd.Series(rf\_clf.feature\_importances\_, index=data\_encoded.columns[:-1])*

*feat\_importances.nlargest(10).plot(kind='barh')*

*plt.title('Feature Importances')*

*plt.show()*

**Comparison of Results**

**Under Sampled Results**

*undersampled\_results = {'Logistic Regression': {'Training Accuracy (UnderSampled)': 77.79, 'Validation Accuracy (UnderSampled)': 77.92, 'Testing Accuracy (UnderSampled)': 78.23},*

*'K-Nearest Neighbours': {'Training Accuracy (UnderSampled)': 99.26, 'Validation Accuracy (UnderSampled)':84.58 ,'Testing Accuracy (UnderSampled)': 84.38},*

*'Decision Tree': {'Training Accuracy (UnderSampled)': 91.46,'Validation Accuracy (UnderSampled)':85.38 , 'Testing Accuracy (UnderSampled)': 85.21},*

*'Random Forest': {'Training Accuracy (UnderSampled)': 97.92 ,'Validation Accuracy (UnderSampled)':89.34 , 'Testing Accuracy (UnderSampled)': 89.25}}*

*US\_Data\_Results = pd.DataFrame(undersampled\_results).transpose()*

*US\_Data\_Results*

***Over Sampled Results***

*oversampled\_results = {'Logistic Regression': {'Training Accuracy (OverSampled)': 79.30,'Validation Accuracy (OverSampled)': 78.15, 'Testing Accuracy (OverSampled)':77.68},*

*'K-Nearest Neighbours': {'Training Accuracy (OverSampled)': 99.38,'Validation Accuracy (OverSampled)': 86.72, 'Testing Accuracy (OverSampled)': 87.87},*

*'Decision Tree': {'Training Accuracy (OverSampled)': 99.46,'Validation Accuracy (OverSampled)': 87.08, 'Testing Accuracy (OverSampled)': 88.33},*

*'Random Forest': {'Training Accuracy (OverSampled)': 99.46 ,'Validation Accuracy (OverSampled)': 90.55, 'Testing Accuracy (OverSampled)': 89.99}}*

*OS\_Data\_Results = pd.DataFrame(oversampled\_results).transpose()*

*OS\_Data\_Results*

*===================== THE END ==========================*