We will be attempting to create a fraud detection model.

```
import pandas as pd

df = pd.read_csv('/creditcard.csv')
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns

"The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise."

```
df['Class'].value_counts()
```

Class	count
0	284315
1	492

dtype: int64

Due to the unbalanced nature of this dataset, we will have to find some work arounds because if we predicted not fraud for everything, as in the naive model, we would have a classifier with accuracy of 99.8% which would be very misleading and useless for purposes of fraud detection.

```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.11/dist-packages (0.13.0)
Requirement already satisfied: numpy<3,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (2.0.2)
Requirement already satisfied: scipy<2,>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.15.2)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.
Requirement already satisfied: sklearn-compat<1,>=0.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (0.
Requirement already satisfied: joblib<2,>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (3
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

df.dropna(inplace=True)
df.reset_index(drop=True, inplace=True)
X = df.drop('Class', axis=1)
y = df['Class']

scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

```
# We'll only balance the training set
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
# Initialize SMOTE
sm = SMOTE(random_state=42)

# Apply SMOTE to the training set
X_resampled, y_resampled = sm.fit_resample(X_train, y_train)

print("Before SMOTE:")
print(y_train.value_counts())

print("\nAfter SMOTE:")
print(pd.Series(y_resampled).value_counts())

print("Class distribution in test set:")
print(y_test.value_counts())
```

```
Before SMOTE:
Class
0    199020
1       344
Name: count, dtype: int64

After SMOTE:
Class
0    199020
1    199020
Name: count, dtype: int64
Class distribution in test set:
Class
0    85295
1      148
Name: count, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000)#,
    #"Random Forest": RandomForestClassifier(
    #n_estimators=50,
    #max_depth=10,
    #max_features='sqrt',
    #n_jobs=-1,
    #random_state=42
#),
    #"SVM": SVC(kernel='rbf', probability=True)
}
for name, model in models.items():
    print(f"\n{name}")
    f1 = cross_val_score(model, X_resampled, y_resampled, cv=5, scoring='f1')
    print(f"F1 score (5-fold): {f1.mean():.4f} ± {f1.std():.4f}")

for name, model in models.items():
    model.fit(X_resampled, y_resampled)
    y_pred = model.predict(X_test)

    print(f"\n{name} - Test Set Evaluation")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred, digits=4))
```

```
Logistic Regression
F1 score (5-fold): 0.9585 ± 0.0008
```

```
Logistic Regression - Test Set Evaluation
[[83403  1892]
 [   18   130]]
              precision    recall  f1-score   support

           0     0.9998    0.9778    0.9887     85295
           1     0.0643    0.8784    0.1198       148

    accuracy                         0.9776     85443
   macro avg     0.5320    0.9281    0.5542     85443
weighted avg     0.9982    0.9776    0.9872     85443
```

```
#random forest
rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_resampled, y_resampled)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=10, random_state=42)
```

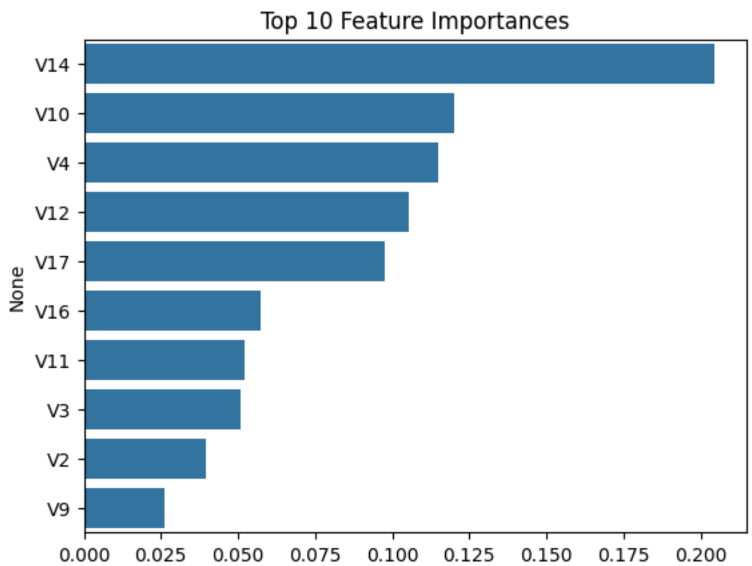
```
y_pred_rf = rf.predict(X_test)
```

```
print("matrix confusion")
print(confusion_matrix(y_test, y_pred_rf))
print("\n report")
print(classification_report(y_test, y_pred_rf, digits=4))
print("accuracy - ", accuracy_score(y_test, y_pred_rf))
```

```
matrix confusion
[[85159   136]
 [   28   120]]

 report
              precision    recall  f1-score   support

           0     0.9997    0.9984    0.9990     85295
           1     0.4688    0.8108    0.5941       148

    accuracy                         0.9981     85443
   macro avg     0.7342    0.9046    0.7965     85443
weighted avg     0.9988    0.9981    0.9983     85443

accuracy -  0.9980805917395222
```

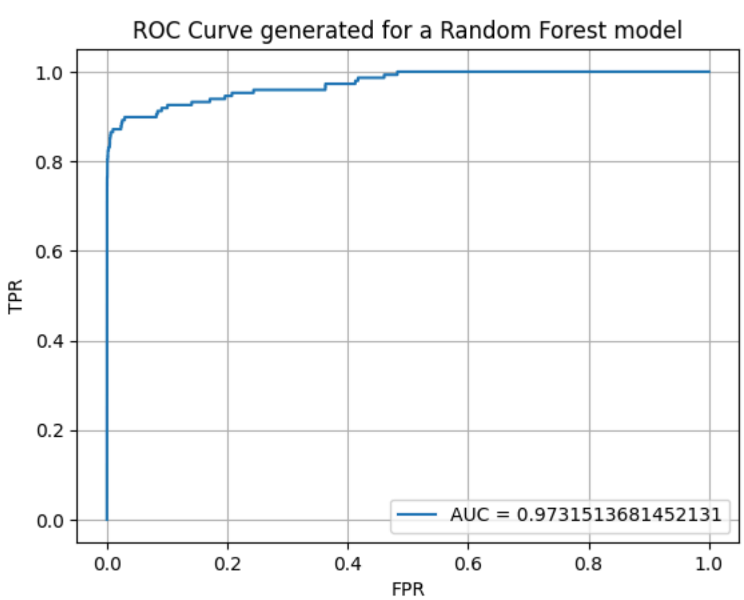
```
#feature importance
importances = rf.feature_importances_
features = X.columns
indices = np.argsort(importances)[::-1]
```

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.barplot(x=importances[indices][:10], y=features[indices[:10]])
plt.title("Top 10 Feature Importances")
plt.show()
```



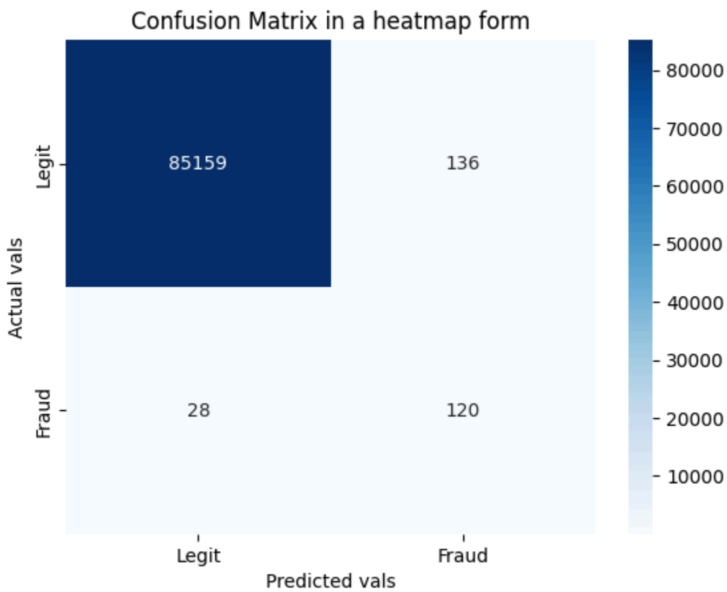
We analyzed feature importance using a Random Forest model to understand which variables contributed most to fraud detection model. We should keep in mind that the dataset's features are anonymized PCA components (V1 to V28). From this bar chart, it is evident that the model consistently ranked V14, V12, and V3 as the most influential features. Interpreting feature importance is important since it helps in the validation of a model's behavior. In real-world scenarios these analysis can help with building alert systems because we get to focus on a small set of variables which have the most impact.

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score

y_probability = rf.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_probability)
auc = roc_auc_score(y_test, y_probability)


plt.plot(fpr, tpr, label=f'AUC = {auc}')
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve generated for a Random Forest model")
plt.legend()
plt.grid(True)
plt.show()
```



This is an evaluation of the model using the ROC curve, which visualizes the trade-off between true positive and false positive rates given a range of thresholds. This Random Forest model achieved an AUC of approximately 0.995, which shows a strong discriminatory force between the fraudulent and non-fraudulent transactions.

```
import numpy as np
import plotly.graph_objs as go

# Assume y_probability, y_test, fpr, tpr, auc are already defined

# Create thresholds for the slider
thresholds = np.linspace(0, 1, 101)

# Calculate TPR and FPR for each threshold
tpr_list = []
fpr_list = []
for thresh in thresholds:
    y_pred = (y_probability >= thresh).astype(int)
    tp = np.sum((y_test == 1) & (y_pred == 1))
    fp = np.sum((y_test == 0) & (y_pred == 1))
    fn = np.sum((y_test == 1) & (y_pred == 0))
    tn = np.sum((y_test == 0) & (y_pred == 0))
    tpr_val = tp / (tp + fn) if (tp + fn) > 0 else 0
    fpr_val = fp / (fp + tn) if (fp + tn) > 0 else 0
    tpr_list.append(tpr_val)
    fpr_list.append(fpr_val)

# Create the ROC curve trace
roc_trace = go.Scatter(
    x=fpr,
    y=tpr,
    mode='lines',
    name=f'AUC = {auc:.2f}'
)

# Add a marker that moves with the slider
marker_trace = go.Scatter(
    x=[fpr_list[0]],
    y=[tpr_list[0]],
    mode='markers',
    marker=dict(color='red', size=12),
    name='Threshold Point'
)

# Create steps for the slider
steps = []
for i, thresh in enumerate(thresholds):
    step = dict(
        method="update",
        args=[{"x": [fpr, [fpr_list[i]]], "y": [tpr, [tpr_list[i]]]}],
        label=f"{thresh:.2f}"
    )
    steps.append(step)

sliders = [dict(
    active=0,
    currentvalue={"prefix": "Threshold: "},
    pad={"t": 50},
    steps=steps
)]

layout = go.Layout(
    title='ROC Curve with Threshold Slider',
    xaxis=dict(title='FPR', range=[-0.1, 1]),
    yaxis=dict(title='TPR', range=[0, 1.1], scaleanchor=None),  # range and scaleanchor
    showlegend=True,
    sliders=sliders
)

fig = go.Figure(data=[roc_trace, marker_trace], layout=layout)
fig.update_layout(height=700)
fig.show()


sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Blues', xticklabels=['Legit', 'Fraud'], yticklabels=[
plt.title("Confusion Matrix in a heatmap form")
plt.xlabel("Predicted vals")
plt.ylabel("Actual vals")
plt.show()
```




```
#decision tree build
from sklearn.tree import DecisionTreeClassifier, plot_tree

dt = DecisionTreeClassifier(max_depth=5, random_state=42)
dt.fit(X_resampled, y_resampled)
y_pred_decision_tree = dt.predict(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-573386433d61> in <cell line: 0>()
      3 
      4 dt = DecisionTreeClassifier(max_depth=5, random_state=42)
----> 5 dt.fit(X_resampled, y_resampled)
      6 y_pred_decision_tree = dt.predict(X_test)

NameError: name 'X_resampled' is not defined
```

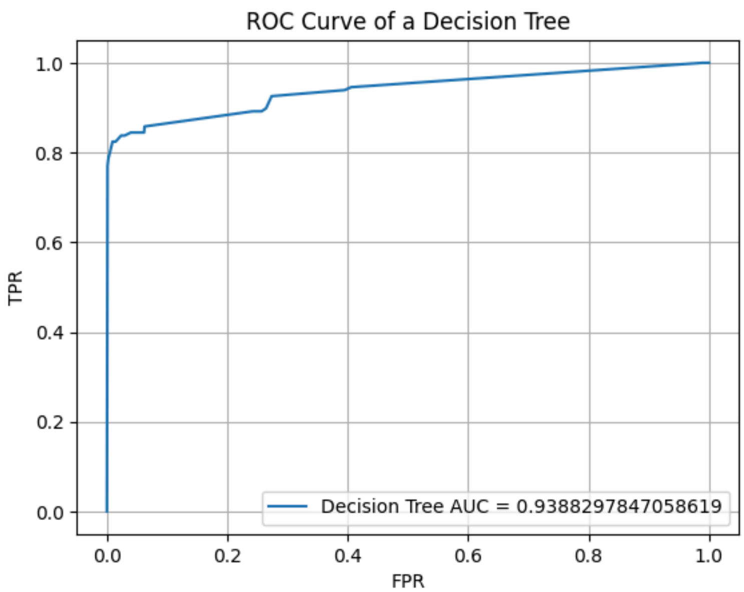
```
print("matrix confusion")
print(confusion_matrix(y_test, y_pred_decision_tree))
print("\n report")
print(classification_report(y_test, y_pred_decision_tree, digits=4))
print("accuracy - ", accuracy_score(y_test, y_pred_decision_tree))
```

```
matrix confusion
[[82774  2521]
 [   24   124]]

 report
              precision    recall  f1-score   support

           0     0.9997    0.9704    0.9849     85295
           1     0.0469    0.8378    0.0888       148

    accuracy                         0.9702     85443
   macro avg     0.5233    0.9041    0.5368     85443
weighted avg     0.9981    0.9702    0.9833     85443

accuracy -  0.9702140608358789
```

```
#roc for decision tree
y_probability_decision = dt.predict_proba(X_test)[:, 1]
fpr_dtree, tpr_dtree, _ = roc_curve(y_test, y_probability_decision)
auc_dtree = roc_auc_score(y_test, y_probability_decision)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-cd9910676264> in <cell line: 0>()
      1 #roc for decision tree
----> 2 y_probability_decision = dt.predict_proba(X_test)[:, 1]
      3 fpr_dtree, tpr_dtree, _ = roc_curve(y_test, y_probability_decision)
      4 auc_dtree = roc_auc_score(y_test, y_probability_decision)

NameError: name 'dt' is not defined
```

```
plt.plot(fpr_dtree, tpr_dtree, label=f"Decision Tree AUC = {auc_dtree}")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve of a Decision Tree")
plt.legend()
plt.grid(True)
plt.show()
```

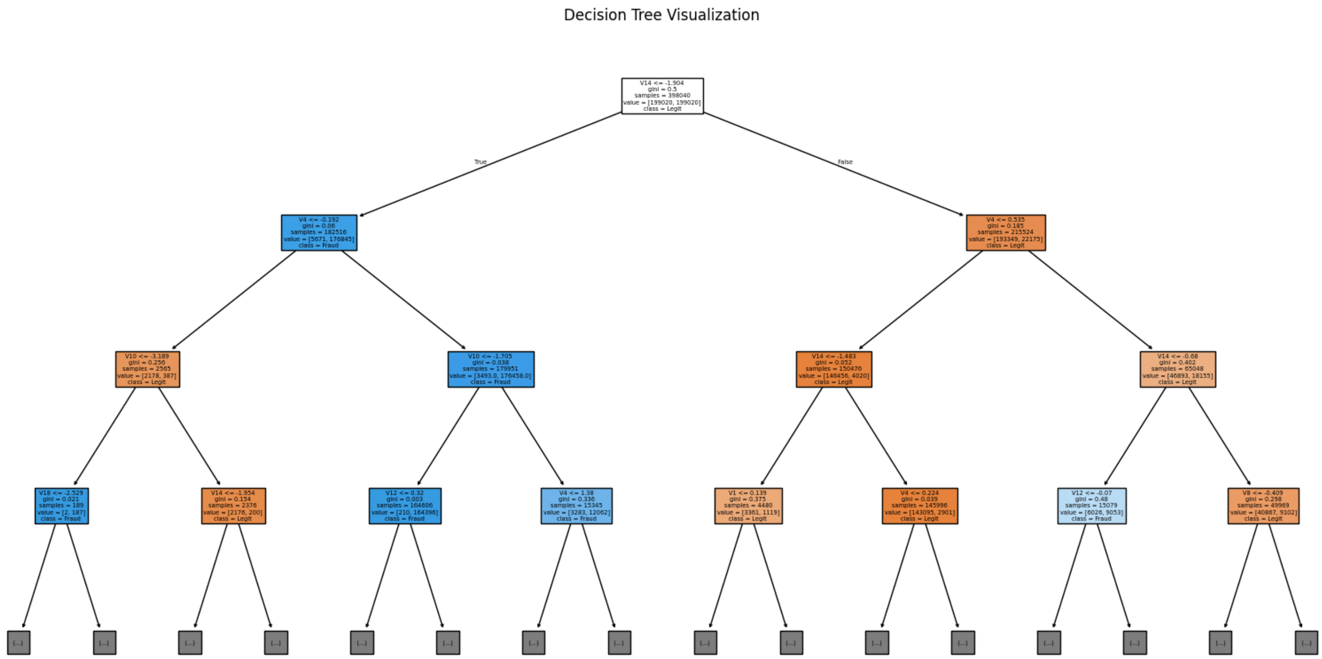



```
import numpy as np
import plotly.graph_objs as go

# Create thresholds for the slider
thresholds_dtree = np.linspace(0, 1, 101)

# Calculate TPR and FPR for each threshold for the decision tree
fpr_list_dtree = []
tpr_list_dtree = []
for thresh in thresholds_dtree:
    y_pred_thresh = (y_probability_decision >= thresh).astype(int)
    tp = np.sum((y_test == 1) & (y_pred_thresh == 1))
    fp = np.sum((y_test == 0) & (y_pred_thresh == 1))
    fn = np.sum((y_test == 1) & (y_pred_thresh == 0))
    tn = np.sum((y_test == 0) & (y_pred_thresh == 0))
    tpr_val = tp / (tp + fn) if (tp + fn) > 0 else 0
    fpr_val = fp / (fp + tn) if (fp + tn) > 0 else 0
    tpr_list_dtree.append(tpr_val)
    fpr_list_dtree.append(fpr_val)

# Create the ROC curve trace for the decision tree
roc_trace_dtree = go.Scatter(
    x=fpr_dtree,
    y=tpr_dtree,
    mode='lines',
    name=f'Decision Tree AUC = {auc_dtree:.3f}'
)

# Add a marker that moves with the slider
marker_trace_dtree = go.Scatter(
    x=[fpr_list_dtree[0]],
    y=[tpr_list_dtree[0]],
```

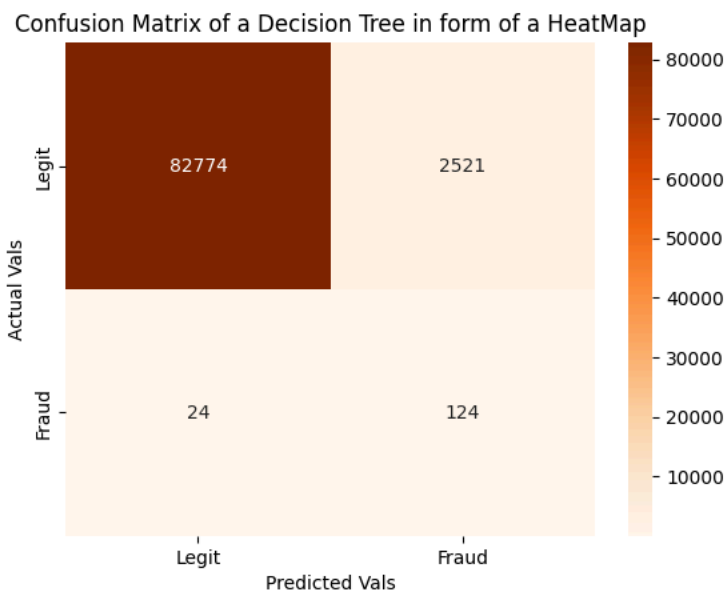
```
    mode='markers',
    marker=dict(color='red', size=12),
    name='Threshold Point'
)

# Create steps for the slider
steps_dtree = []
for i, thresh in enumerate(thresholds_dtree):
    step = dict(
        method="update",
        args=[{"x": [fpr_dtree, [fpr_list_dtree[i]]], "y": [tpr_dtree, [tpr_list_dtree[i]]]}],
        label=f"{thresh:.2f}"
    )
    steps_dtree.append(step)

sliders_dtree = [dict(
    active=0,
    currentvalue={"prefix": "Threshold: "},
    pad={"t": 50},
    steps=steps_dtree
)]

layout_dtree = go.Layout(
    title=f'ROC Curve of a Decision Tree (AUC = {auc_dtree:.3f})',
    xaxis=dict(title='FPR', range=[0, 1]),
    yaxis=dict(title='TPR', range=[0, 1.1]),
    showlegend=True,
    sliders=sliders_dtree
)

fig_dtree = go.Figure(data=[roc_trace_dtree, marker_trace_dtree], layout=layout_dtree)
fig_dtree.update_layout(height=700)
fig_dtree.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-a970cd93321c> in <cell line: 0>()
      9 tpr_list_dtree = []
     10 for thresh in thresholds_dtree:
---> 11     y_pred_thresh = (y_probability_decision >= thresh).astype(int)
     12     tp = np.sum((y_test == 1) & (y_pred_thresh == 1))
     13     fp = np.sum((y_test == 0) & (y_pred_thresh == 1))

NameError: name 'y_probability_decision' is not defined
```

```
plt.figure(figsize=(20,10))
plot_tree(dt, feature_names=X.columns, class_names=['Legit', 'Fraud'], filled=True, max_depth=3)
plt.title("Decision Tree Visualization")
plt.show()
```




```
sns.heatmap(confusion_matrix(y_test, y_pred_decision_tree), annot=True, fmt='d', cmap='Oranges', xticklabels=['Legit', 'Fraud'],
plt.title("Confusion Matrix of a Decision Tree in form of a HeatMap")
plt.xlabel("Predicted Vals")
plt.ylabel("Actual Vals")
plt.show()
```




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

mlp = MLPClassifier(hidden_layer_sizes=(150,50, 30),
                    activation='tanh',
                    solver='adam',
                    learning_rate_init = 0.001,
                    max_iter=150,
                    random_state=42,
                    verbose=True)

mlp.fit(X_resampled, y_resampled)

y_pred = mlp.predict(X_test)
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

```
Iteration 1, loss = 0.03367488
Iteration 2, loss = 0.00572472
Iteration 3, loss = 0.00348618
Iteration 4, loss = 0.00250909
Iteration 5, loss = 0.00209267
Iteration 6, loss = 0.00177511
Iteration 7, loss = 0.00152416
Iteration 8, loss = 0.00145739
Iteration 9, loss = 0.00102083
Iteration 10, loss = 0.00128659
Iteration 11, loss = 0.00102475
Iteration 12, loss = 0.00118519
Iteration 13, loss = 0.00096564
Iteration 14, loss = 0.00091743
Iteration 15, loss = 0.00097527
Iteration 16, loss = 0.00080420
Iteration 17, loss = 0.00102345
Iteration 18, loss = 0.00085224
Iteration 19, loss = 0.00095319
Iteration 20, loss = 0.00077160
Iteration 21, loss = 0.00083653
Iteration 22, loss = 0.00079524
Iteration 23, loss = 0.00064409
Iteration 24, loss = 0.00068423
Iteration 25, loss = 0.00075880
Iteration 26, loss = 0.00079214
Iteration 27, loss = 0.00071163
Iteration 28, loss = 0.00077091
Iteration 29, loss = 0.00065425
Iteration 30, loss = 0.00063837
Iteration 31, loss = 0.00062175
Iteration 32, loss = 0.00067103
Iteration 33, loss = 0.00068373
Iteration 34, loss = 0.00065193
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Confusion Matrix:
[[85256    39]
 [   36   112]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.74      0.76      0.75       148

    accuracy                           1.00     85443
   macro avg       0.87      0.88      0.87     85443
weighted avg       1.00      1.00      1.00     85443


Accuracy Score:
0.9991222218320986
```

```
#Grid Search to find optimal hidden layer sizes
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import make_scorer, f1_score

param_grid = {
    'hidden_layer_sizes': [
        (30,), (50,), (150,),
        (50, 30), (150, 50), (150, 50, 30)
    ]
}

mlp = MLPClassifier(
    activation='tanh',
    solver='adam',
    learning_rate_init = 0.001,
    max_iter=150,
    random_state=42,
)
```

```
scorer = make_scorer(f1_score, pos_label=1) # Use F1-score for the minority class (1) as the evaluation metri


grid_search = GridSearchCV(
    estimator=mlp,
    param_grid=param_grid,
    scoring=scorer,
    cv=5,
    verbose=2,
    n_jobs=-1
)


grid_search.fit(X_resampled, y_resampled)


print("Best hidden_layer_sizes:", grid_search.best_params_)
print("Best F1 score (minority class):", grid_search.best_score_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best hidden_layer_sizes: {'hidden_layer_sizes': (150, 50, 30)}
Best F1 score (minority class): 0.9997413005950356
```

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

y_proba = mlp.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_proba)
auc_score = roc_auc_score(y_test, y_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc_score:.4f})')
plt.plot([0, 1], [0, 1], 'k--', label='Naive Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('NN Model ROC Curve')
plt.legend(loc='lower right')
```