Karam Hussain 27857

File name: kaggle_task1_final.ipynb

Task 1: Fine-tuning Decision Tree

a) Loading dataset

Fetching and splitting

```
data = pd.read_csv('train1.csv')
data.info()
data.head()
```

✓ 4.1s

b) Train-test split 70-30, seed is ERP(27857)
And finding out if the class is imbalanced which it is which we can see because of the ratio.

```
X = data.drop('target', axis=1)
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=27857   #this test is 'validation' set and my erp is 27857 (karam)
)
```

✓ 1.1s

```
print(f"Target distribution:\n{y.value_counts()}")
```

✓ 0.0s

```
Target distribution:
target
0    281023
1     15186
Name: count, dtype: int64
```

c) Handling missing values, separating columns on their datatypes; binary, categorical, numeric in order to filter further. Filtered the columns with the help of the names of columns. Also noted that the categorical columns were actually not literally ‘categorical’ but instead in number form i.e. labelled encoded, which is an important detail.

Used mode to fill in the missing values of the categorical columns and median to fill in the numerical columns, which will help in overcoming outliers if they will be present in the dataset and will be more robust.

Imputing

+ Code + Markdown

```
from sklearn.impute import SimpleImputer

bin_cols = [c for c in X_train.columns if '_bin' in c]
cat_cols = [c for c in X_train.columns if '_cat' in c]
num_cols = [c for c in X_train.columns if c not in bin_cols + cat_cols + ['id']]

# Categorical columns → mode (most_frequent)
imputer_cat = SimpleImputer(strategy='most_frequent')
X_train[cat_cols] = imputer_cat.fit_transform(X_train[cat_cols])
X_test[cat_cols] = imputer_cat.transform(X_test[cat_cols])

# Fill numeric NaNs with median from training data
median_vals = X_train[num_cols].median()
X_train[num_cols] = X_train[num_cols].fillna(median_vals)
X_test[num_cols] = X_test[num_cols].fillna(median_vals)

# fill binary columns with mode
imputer_bin = SimpleImputer(strategy='most_frequent')
X_train[bin_cols] = imputer_bin.fit_transform(X_train[bin_cols])
X_test[bin_cols] = imputer_bin.transform(X_test[bin_cols])
```

And also output the columns for validation with their count.

```
print("Binary cols:", len(bin_cols))
print("Categorical cols:", len(cat_cols))
print("Numeric cols:", len(num_cols))

print("Binary Columns ({}):".format(len(bin_cols)), bin_cols, "\n")
print("Categorical Columns ({}):".format(len(cat_cols)), cat_cols, "\n")
print("Numeric Columns ({}):".format(len(num_cols)), num_cols)
```

✓ 0.0s

```
Binary cols: 17
Categorical cols: 14
Numeric cols: 34
Binary Columns (17): ['ps_ind_06_bin', 'ps_ind_07_bin', 'ps_ind_08_bin', 'ps_ind_09_bin', 'ps_inc

Categorical Columns (14): ['ps_ind_02_cat', 'ps_ind_04_cat', 'ps_ind_05_cat', 'ps_car_01_cat', 'p

Numeric Columns (34): ['ps_ind_01', 'ps_ind_03', 'ps_ind_14', 'ps_ind_15', 'ps_reg_01', 'ps_reg_0
```

d) Training model on baseline default Decision Tree Model and gettings its AUROC

Decision Tree = 0.5107

Baseline DT AUROC

```
#baseline TREE AUROC -->> 0.5107
model = DecisionTreeClassifier(random_state=27857)
model.fit(X_train, y_train)

y_pred = model.predict_proba(X_test)[:, 1]
baseline_auroc = roc_auc_score(y_test, y_pred)

print("Baseline AUROC:", baseline_auroc)
```

✓ 14.4s

```
Baseline AUROC: 0.5107187969548 35
```

e) Feature Selection (Filter Methods: ANOVA for numeric + Chi Square for categorical)

Out of the total 67 - target - id = 65 features,
- we used ANOVA on numeric only features with k=15 out of 34 numeric features
- and chi square for categorical + binary (combined) since both are discrete, with k =15 out of 31 combined cat_bin columns.

In the end we combined the best 15+15 columns we got into a selected features set of 30 columns.

AUROC with Filter methods: 0.5076

Filtering features (ANOVA + Chi2)

```python
from sklearn.feature_selection import SelectKBest, f_classif, chi2

# --- Confirm column groups --- -->>> 0.5076
print("Number of binary columns:", len(bin_cols))
print("Number of categorical columns:", len(cat_cols))
print("Number of numeric columns:", len(num_cols))

# ✅ Use ANOVA (f_classif) for numeric features
anova_selector = SelectKBest(score_func=f_classif, k=min(15, len(num_cols)))
anova_selector.fit(X_train[num_cols], y_train)
selected_num = np.array(num_cols)[anova_selector.get_support()]

# ✅ Use Chi-Square for categorical + binary (non-negative, discrete)
# Combine binary and categorical since both are discrete numeric
cat_bin_cols = bin_cols + cat_cols

chi_selector = SelectKBest(score_func=chi2, k=min(15, len(cat_bin_cols)))
chi_selector.fit(X_train[cat_bin_cols], y_train)
selected_catbin = np.array(cat_bin_cols)[chi_selector.get_support()]

# ✅ Combine both selected feature sets
selected_features = list(selected_num) + list(selected_catbin)
```

```
Selected numeric features (ANOVA):
['ps_ind_01' 'ps_ind_03' 'ps_ind_14' 'ps_ind_15' 'ps_reg_01' 'ps_reg_02'
 'ps_reg_03' 'ps_car_12' 'ps_car_13' 'ps_car_14' 'ps_car_15' 'feature2'
 'feature4' 'feature5' 'feature7']

Selected binary + categorical features (Chi-Square):
['ps_ind_06_bin' 'ps_ind_07_bin' 'ps_ind_08_bin' 'ps_ind_09_bin'
 'ps_ind_16_bin' 'ps_ind_17_bin' 'ps_ind_04_cat' 'ps_ind_05_cat'
 'ps_car_01_cat' 'ps_car_02_cat' 'ps_car_04_cat' 'ps_car_06_cat'
 'ps_car_08_cat' 'ps_car_09_cat' 'ps_car_11_cat']

✅ Combined selected features:
['ps_ind_01', 'ps_ind_03', 'ps_ind_14', 'ps_ind_15', 'ps_reg_01', 'ps_reg_02', 'ps_reg_03', 'ps_c

🌟 AUROC after ANOVA + Chi-Square feature selection: 0.5076825466716828
```

f) Wrapper based selection (Forward selection and backward selection)

For forward selection, we did that upon all 65 features with k=10 and CV = 5 for robustness, had to disable parallel proc due to internal issues hence there were some time constraints.

Forward selection AUROC: 0.5775

Forward Selection (all features)

+ Code + Markdown

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# --- Base model ---
base_model = DecisionTreeClassifier(random_state=27857)

# --- Forward Selection ---
sfs_forward = SFS(
    base_model,
    k_features=10,              # Select top 10 features
    forward=True,               # Forward stepwise
    floating=False,             # No floating (simple forward)
    scoring='roc_auc',          # Use AUROC for selection
    cv=5,                       # 5-fold CV
    n_jobs=1,                  # Use all cores
    verbose=2                   # Show progress
)
```

```
✅ Forward Selection Features:
['ps_car_04_cat', 'ps_car_10_cat', 'ps_car_11_cat', 'ps_ind_06_bin', '

🌟 AUROC after Forward Feature Selection: 0.5775532766858615
```

Now for backward selection, we will use the subset features which we got from anova +chisquare = 30.
With k=15 and CV=3 so it computes faster.

Backward Elimination AUROC: 0.5101

Backward Selection (with subset features)

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
import time

# ============================================
# Use the features selected by ANOVA + Chi2
# ============================================
selected_features_fs = [
'ps_ind_01', 'ps_ind_03', 'ps_ind_14', 'ps_ind_15','ps_reg_01', 'ps_reg_02', 'ps_reg_03', 'ps_car_12',
'ps_car_13', 'ps_car_14', 'ps_car_15', 'feature2','feature4', 'feature5', 'feature7', 'ps_ind_06_bin',
'ps_ind_07_bin', 'ps_ind_08_bin', 'ps_ind_09_bin','ps_ind_16_bin','ps_ind_17_bin', 'ps_ind_04_cat',
'ps_ind_05_cat','ps_car_01_cat','ps_car_02_cat', 'ps_car_04_cat', 'ps_car_06_cat', 'ps_car_08_cat',
'ps_car_09_cat', 'ps_car_11_cat'

]
```

```
sfs_backward = SFS(
    base_model,
    k_features=15,        # Finds optimal number of features automatically
    forward=False,            # Backward elimination
    floating=False,
    scoring='roc_auc',
    cv=3,                     # 3-fold CV to keep it faster
    n_jobs=1,                # Use all CPU cores
    verbose=2                 # Show progress
)
```

```
✅ Backward selection chosen features:
['ps_ind_03', 'ps_ind_15', 'ps_reg_02', 'ps_reg_03', 'ps_car_12', '

🔥 AUROC after backward selection: 0.5101
```

g) Hyperparameter Tuning

Defining a hyperparameter grid with many variables and their values and applying grid search to find the optimal set among them with CV=5, and applying on default Decision Tree classifier.

DT + HYP AUROC: 0.5993

```python
# --- Define the model ---
dt = DecisionTreeClassifier(random_state=27857)

# --- Define hyperparameter grid ---
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

# --- Grid Search with 5-fold CV ---
grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    n_jobs=1,
    verbose=2
)
```

```
🔍 Best Hyperparameters:
{'criterion': 'gini', 'max_depth': 7, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 10}

🏆 Best CV AUROC: 0.5992990920824123
```

h) Hyperparameters on Filtered features

Applying the tuned/optimal hyp values on the 30 filtered features, upon which we get

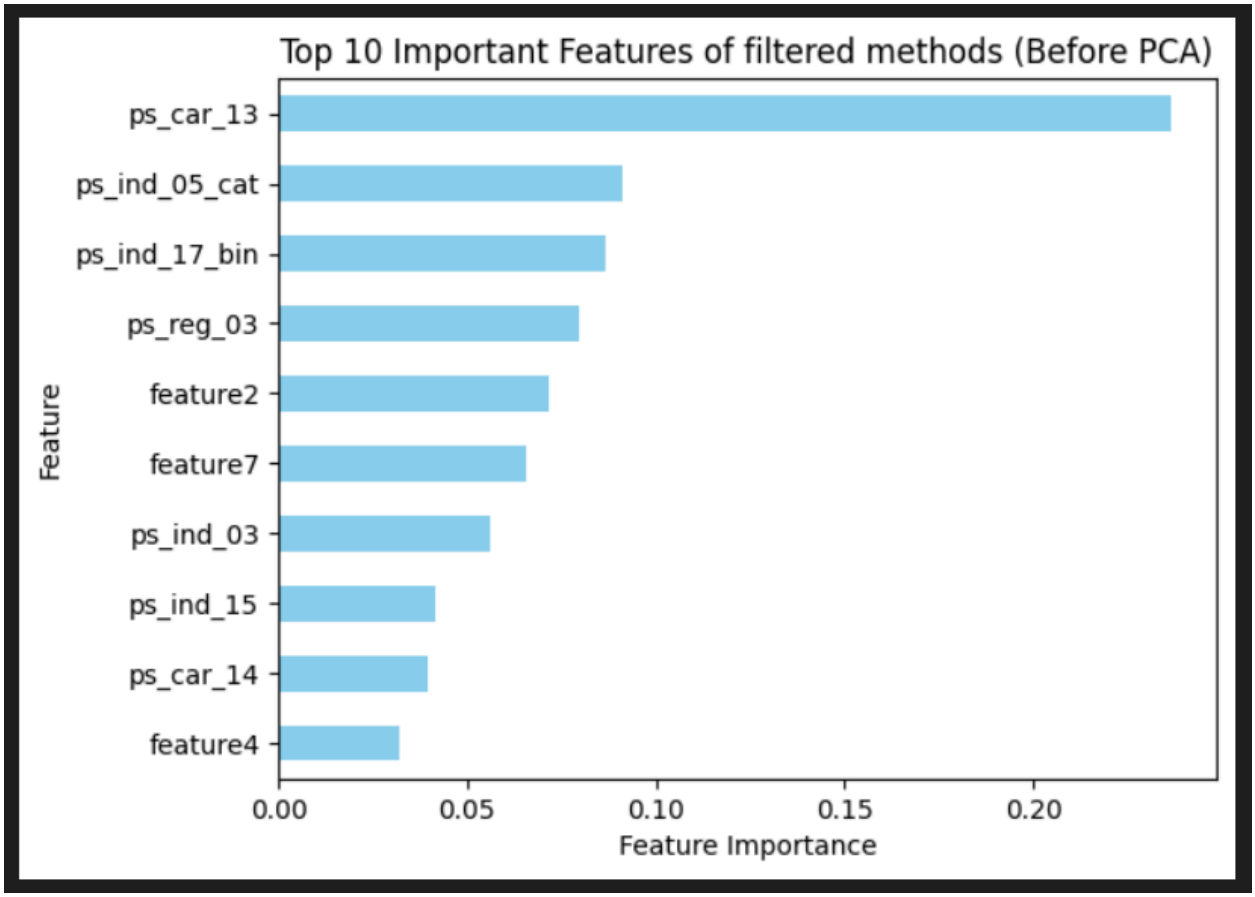
FILTER + HYP AUROC: 0.6045

Hyperparamters on Filtered Features (anova + chi2) (30)

```python
# === Final Feature Importance Analysis (Before PCA) ===
import matplotlib.pyplot as plt

# Using the sleected features thru anova + chi2

# Subset your data
X_train_sel = X_train[selected_features_fs]
X_test_sel = X_test[selected_features_fs]

# Use your best hyperparameters (replace values with your GridSearchCV results)
best_params = {
    'criterion': 'gini',
    'max_depth': 7,
    'min_samples_split': 10,
    'min_samples_leaf': 1,
    'max_features': None,
    'random_state': 27857
}
```

```
Final AUROC (best hyperparameters on Filtered methods, before PCA): 0.6045

Top 10 Important Features of filtered methods:
ps_car_13        0.236778
ps_ind_05_cat    0.091189
ps_ind_17_bin    0.086692
ps_reg_03        0.079926
feature2         0.071738
feature7         0.065871
ps_ind_03        0.055927
ps_ind_15        0.041727
ps_car_14        0.039596
feature4         0.032303
dtype: float64
```

We also find out the top 10 important features for the above in which we can clearly see **ps_car_13** which is a numeric variable dominating the Importance very easily, and we can only see one categorical feature **ps_ind_05_cat** and one binary feature **ps_ind_17_bin** in the list in second and third positions respectively but very far behind the first. The cat and bin features have similar importances, and **feature4** has the least among them even though it is a numeric feature.
We can see the filter methods have supported the numeric features very well, but not the cat and bin very much.



i) Hyperparameters on Wrapper methods

We also apply the tuned hyperparameters on the wrapper methods by combining the features we got from forward selection 10 + backward elimination 15 and basically taking all the unique non-repeating features which ended up being 22 in total.

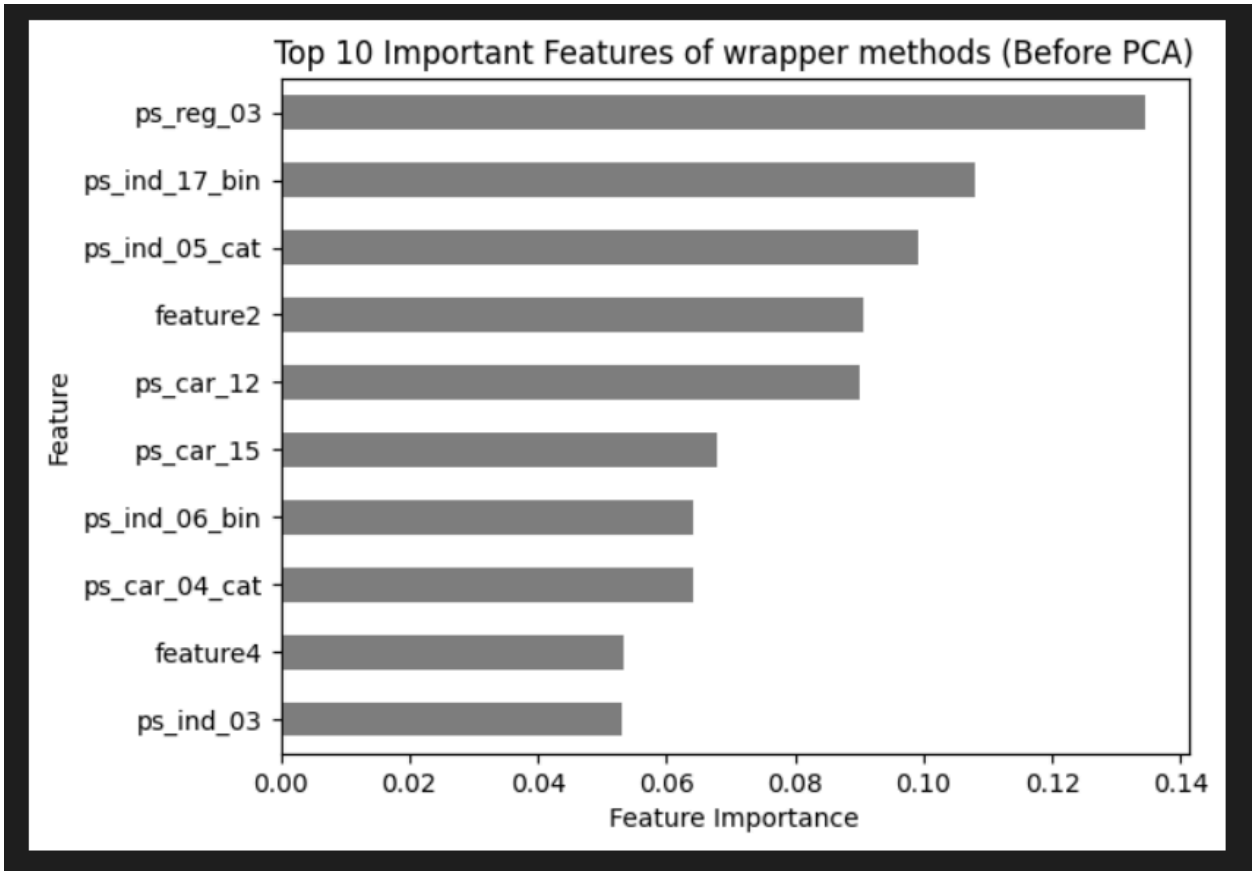
Wrapper + HYP: 0.5944

Hyperparamters on Wrapper Selected features (FW + BW) (22)

```python
# === Final Feature Importance Analysis (Before PCA) ===
import matplotlib.pyplot as plt

# Using the sleected features of forward selection
sel_features_forward = ['ps_ind_03', 'ps_ind_15', 'ps_reg_02', 'ps_reg_03', 'ps_car_12', 'ps_car_15',
 'feature2', 'feature4', 'feature7', 'ps_ind_06_bin', 'ps_ind_17_bin',
 'ps_ind_05_cat', 'ps_car_01_cat', 'ps_car_04_cat', 'ps_car_06_cat',
 'ps_car_10_cat', 'ps_car_11_cat', 'ps_ind_09_bin', 'ps_ind_10_bin',
 'ps_ind_11_bin', 'ps_ind_13_bin', 'feature1']

# Subset your data
X_train_sel = X_train[sel_features_forward]
X_test_sel = X_test[sel_features_forward]

# Use your best hyperparameters (replace values with your GridSearchCV results)
best_params = {
    'criterion': 'gini',
    'max_depth': 7,
    'min_samples_split': 10,
    'min_samples_leaf': 1,
    'max_features': None,
    'random_state': 27857
}
```

```
Final AUROC (best hyperparameters on Wrapper Selected featuers, before PCA): 0.5944

Top 10 Important Features of wrapper methods:
ps_reg_03        0.134454
ps_ind_17_bin    0.108138
ps_ind_05_cat    0.099237
feature2         0.090536
ps_car_12        0.090119
ps_car_15        0.067880
ps_ind_06_bin    0.064213
ps_car_04_cat    0.064164
feature4         0.053239
ps_ind_03        0.053187
dtype: float64
```

After doing that we also find the feature importance of the top 10 from the above as well. In which we can see another numeric feature on the top **ps_reg_03,** with a binary one on second position and a categorical one on third position, being **ps_ind_17_bin** and **ps_ind_05_cat** respectively. This time the distance between first and second/third is less. We can see another pair of bin and cat columns below making it a combined 4 of them this time vs the 6 numerical features. Wrapper methods have distributed the feature importances a bit more evenly and this time more inclusion for bin_cat columns compared to previous one.



j) PCA and dimensionality reduction

We will apply PCA on the numerical columns as PCA works best for numeric columns and then concatenate those converted components with the leftover bin_cat columns.

After applying PCA with capturing **90% variance**, we all of our 34 numeric columns got converted into **2 principal components** and 90% variance at least was portrayed by using both of them together.

After getting those 2 components, our feature size was now 2+31 instead of 34+31, reducing it from 65 to 33, which is a very significant difference considering we are also keeping 90%+ variance.

First feature **PCA1 explained 54.3%** variance while **PCA2 explained the remaining 45.7%** variance.

We also applied hyperparameters and top10 feature importance as well on this set.

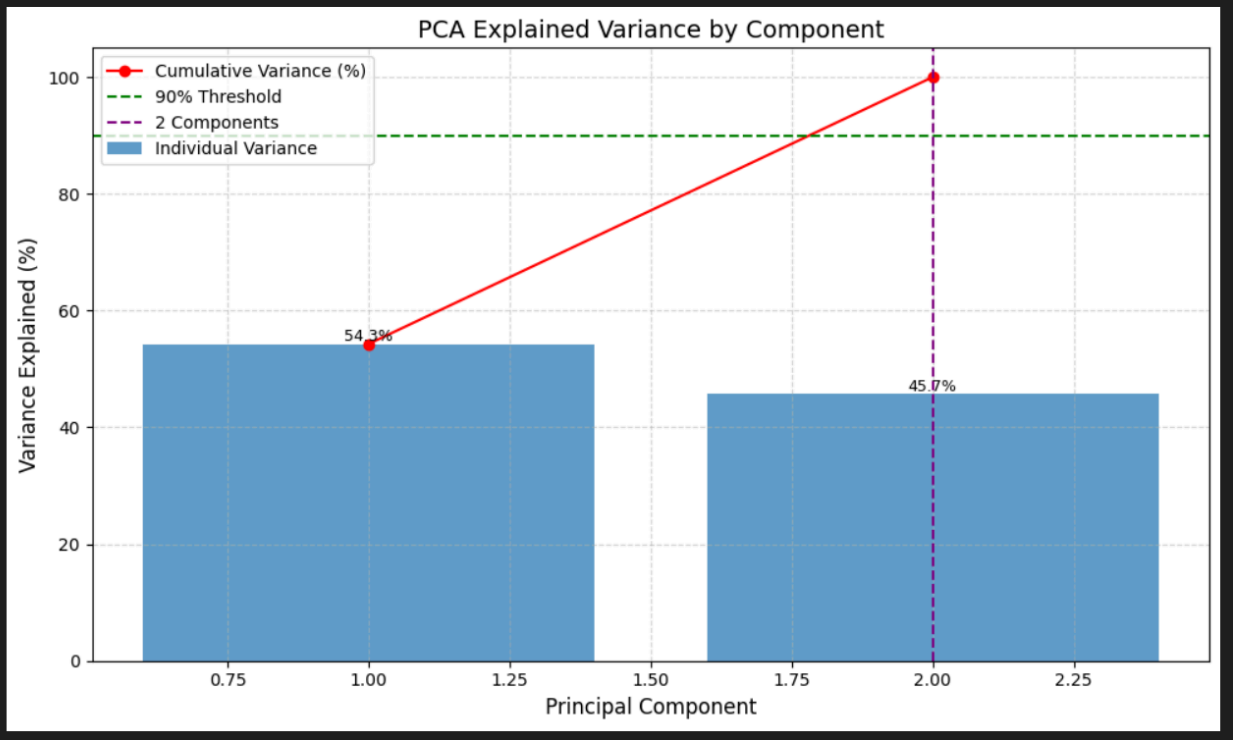
PCA + HYP AUROC: 0.5911

PCA [on numeric data] + concat(bin,cat) (2 + 31) with Hyperparamters

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


bin_cat_cols = bin_cols + cat_cols

# ====================================================
# 2 Fit PCA only on training numeric data
# ====================================================
pca = PCA(n_components=0.90, random_state=27857)  # keep 90% variance
X_train_pca = pca.fit_transform(X_train[num_cols])
X_test_pca = pca.transform(X_test[num_cols])
```

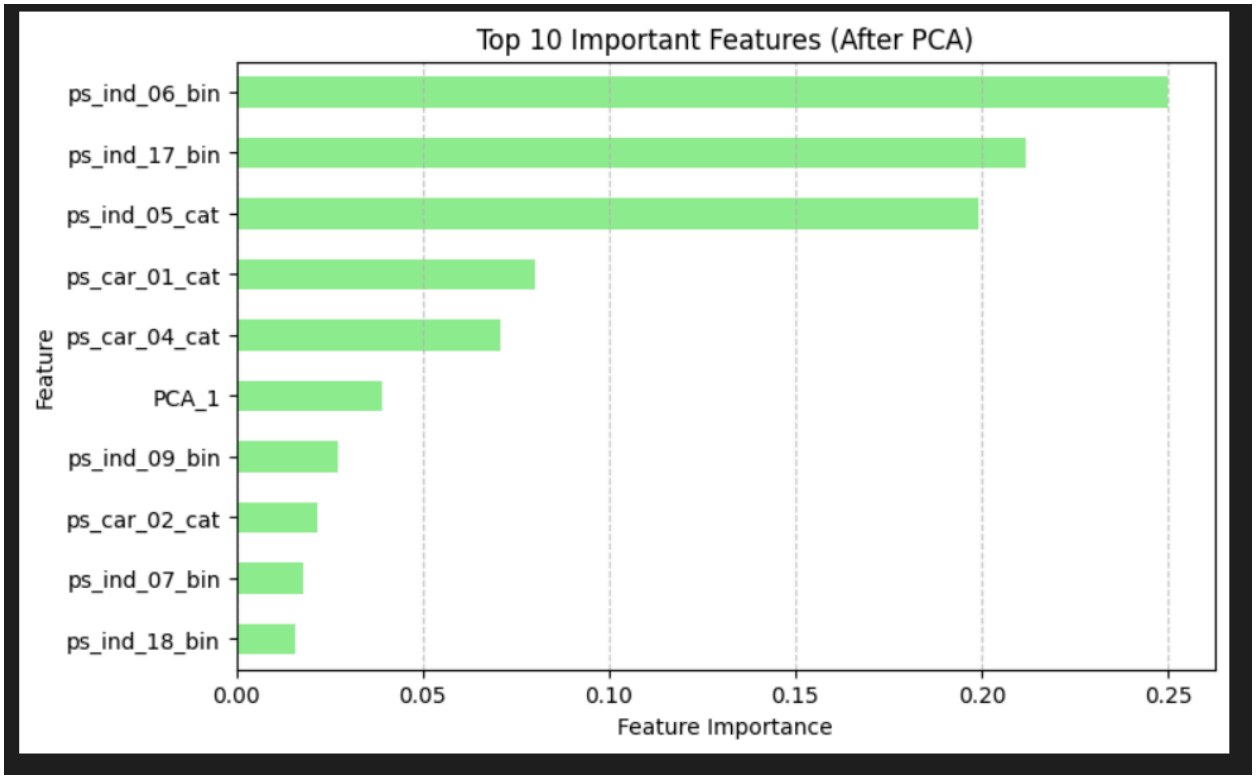
```
✅ Aligned Shapes:
X_train_final: (207346, 33)
y_train: (207346,)
X_test_final: (88863, 33)
y_test: (88863,)

Final AUROC (After PCA): 0.5911
```

```
Number of components capturing 90% variance: 2
```




```
Top 10 Important Features (After PCA):
ps_ind_06_bin    0.250019
ps_ind_17_bin    0.211883
ps_ind_05_cat    0.199273
ps_car_01_cat    0.080114
ps_car_04_cat    0.071020
PCA_1            0.039009
ps_ind_09_bin    0.027138
ps_car_02_cat    0.021853
ps_ind_07_bin    0.017846
ps_ind_18_bin    0.015888
dtype: float64
```

One very big benefit for using PCA was seen after extracting the top 10 features, this time as all the numeric columns got squeezed into two PCs, we can see one of them **PCA1** on the sixth position, while this time around binary and categorical columns have gotten to showcase themselves more.
In the top five we have two binary columns in first and second, being **ps_ind_06_bin** and **ps_ind_17_bin**, while in third, fourth and fifth positions we have categorical features, being **ps_ind_05_cat, ps_car_01_cat** and **ps_car_04_cat.** The top 3 have relatively higher importance than the rest.



****Final AUROC Comparison table****

	Method	AUROC
2	Filter methods (ANOVA + Chi2) + HYP	0.6045
3	Decision Tree + HYP	0.5993
4	Wrapper (Forward + Backward) + HYP	0.5944
5	PCA + HYP	0.5911
6	Wrapper (Forward Selection)	0.5775
7	Decision Tree (baseline)	0.5107
8	Wrapper (Backward Elimination)	0.5101
9	Filter methods (ANOVA + Chi2)	0.5076

AUROC improved from **0.5107 (baseline)** to **0.6045** after combining **filter-based feature selection and hyperparameter tuning**, marking a **~9% performance gain**.
Wrapper and PCA approaches showed moderate improvements (~0.59), confirming that **feature refinement plus tuning** provided the most effective enhancement in model generalization.

As we can see, filtered methods with HYP values performed the best crossing the 0.60 threshold alongside other methods with the same HYP values are very close as well above the 0.59+ range. While the highest non-hyp method was Wrapper with 0.57 which is a significant difference from the last HYP induced method.
This signifies the importance of using hyperparameters and their optimization, which can also be seen with the large difference of the similar methods with their HYP and non-HYP counterparts!

(hyperparameter insight on next page)

🔍 Best Hyperparameters:
{'criterion': 'gini', 'max_depth': 7, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 10}

Insight Summary:
The tuned Decision Tree (max_depth = 7, min_samples_split = 10, criterion = 'gini') achieved a strong balance between bias and variance. Limiting depth and increasing split size reduced overfitting and improved generalization. Using all/relevant features allowed the model to capture key patterns without losing important predictors, leading to a higher and more stable AUROC compared to the baseline.