Connected to Python 3.12.10

In [ ]:
```
# %pip install ucimlrepo
from ucimlrepo import fetch_ucirepo

# fetch dataset
us_census_data_1990 = fetch_ucirepo(id=116)

# This project uses the 1990 U.S. Census dataset from the UC
```

In [ ]:
```
# Load data (as pandas dataframes)
X = us_census_data_1990.data.features
y = us_census_data_1990.data.targets

# Combine features and targets (if targets exist)
df = us_census_data_1990.data.original

print(df.shape)        # rows, columns
print(df.head())       # peek at first 5 rows
```

```
(1534490, 69)
   caseid  dAge  dAncstry1  dAncstry2  iAvail  iCitizen  iCla
ss  dDepart  \
0   10000     5          0          1       0         0
5        3
1   10001     6          1          1       0         0
7        5
2   10002     3          1          2       0         0
7        4
3   10003     4          1          2       0         0
1        3
4   10004     7          1          1       0         0
0        0

   iDisabl1  iDisabl2  ...  iTmpabsnt  dTravtime  iVietnam  d
Week89  iWork89  \
0         2         2  ...        0.0        5.0       0.0
2.0      1.0
1         2         2  ...        0.0        1.0       0.0
2.0      1.0
2         2         2  ...        0.0        2.0       0.0
2.0      1.0
3         2         2  ...        0.0        1.0       0.0
1.0      1.0
4         2         2  ...        3.0        0.0       0.0
0.0      2.0

   iWorklwk  iWWII  iYearsch  iYearwrk  dYrsserv
0       1.0    0.0      11.0       1.0       0.0
1       1.0    0.0       5.0       1.0       0.0
2       1.0    0.0      10.0       1.0       0.0
3       1.0    0.0      10.0       1.0       0.0
4       2.0    0.0       5.0       6.0       0.0

[5 rows x 69 columns]
```

In [ ]:
```
# Exploratory Data Analysis (EDA)
# Check dimensions
print(X.shape)

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

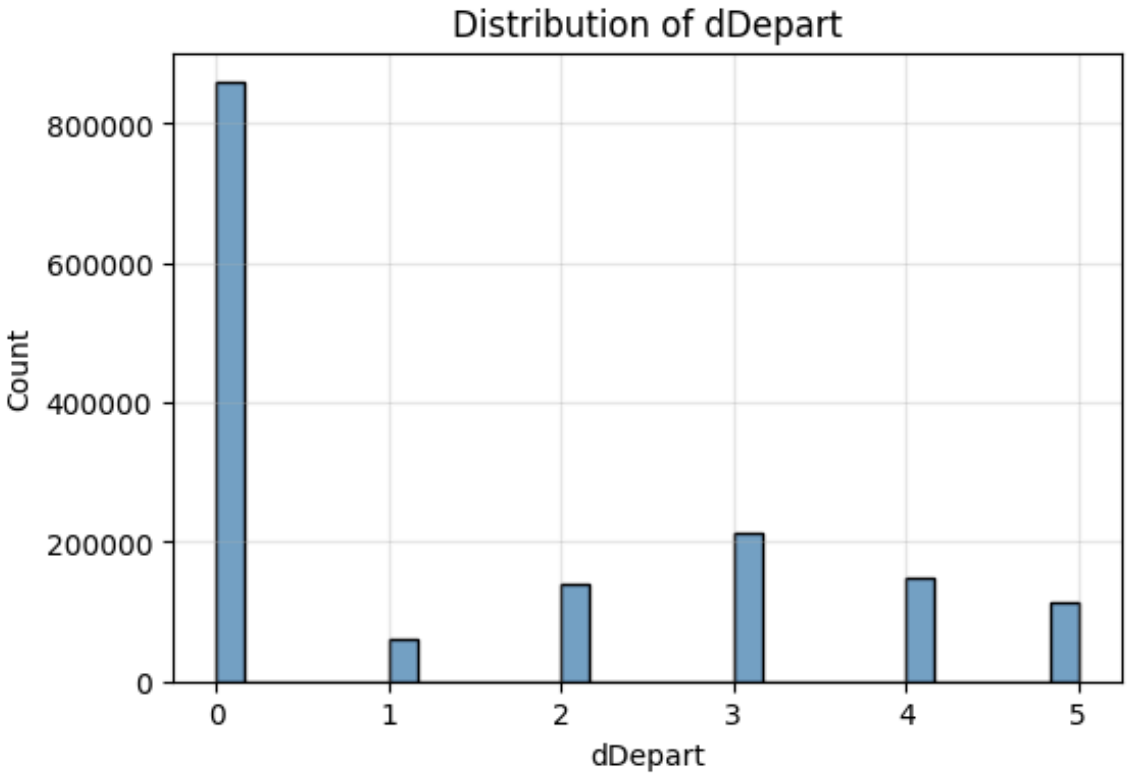
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardSca
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_r

#  Step 1: Separate features and target
X = df.drop(columns=['dTravtime'])
y = df['dTravtime'].astype('category')   # categorical outco

#  Step 2: Mark categorical predictors
categorical_vars = [
    "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstry2",
    "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1",
    "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
    "iDisabl2", "iMobillim", "iVietnam", "iWWII"
]
numeric_vars = [col for col in X.columns if col not in categ

# Define numeric and categorical variables from top features
numeric_features = [
    "dDepart", "iMeans", "dHours", "iClass", "dHour89", "iRi
    "dWeek89", "dRearning", "iYearsch", "iYearwrk", "dIncome
    "iLooking", "dAge", "dPwgt1", "iRPOB", "iTmpabsnt"
]

categorical_features = [
    "dOccup", "dIndustry", "iWorklwk"
]

# Function to plot numeric features
def plot_numeric_univariate(df, features, bins=30):
    for col in features:
        if col in df.columns:
            plt.figure(figsize=(6,4))
            sns.histplot(df[col], bins=bins, kde=False, colo
            plt.title(f"Distribution of {col}")
            plt.xlabel(col)
```

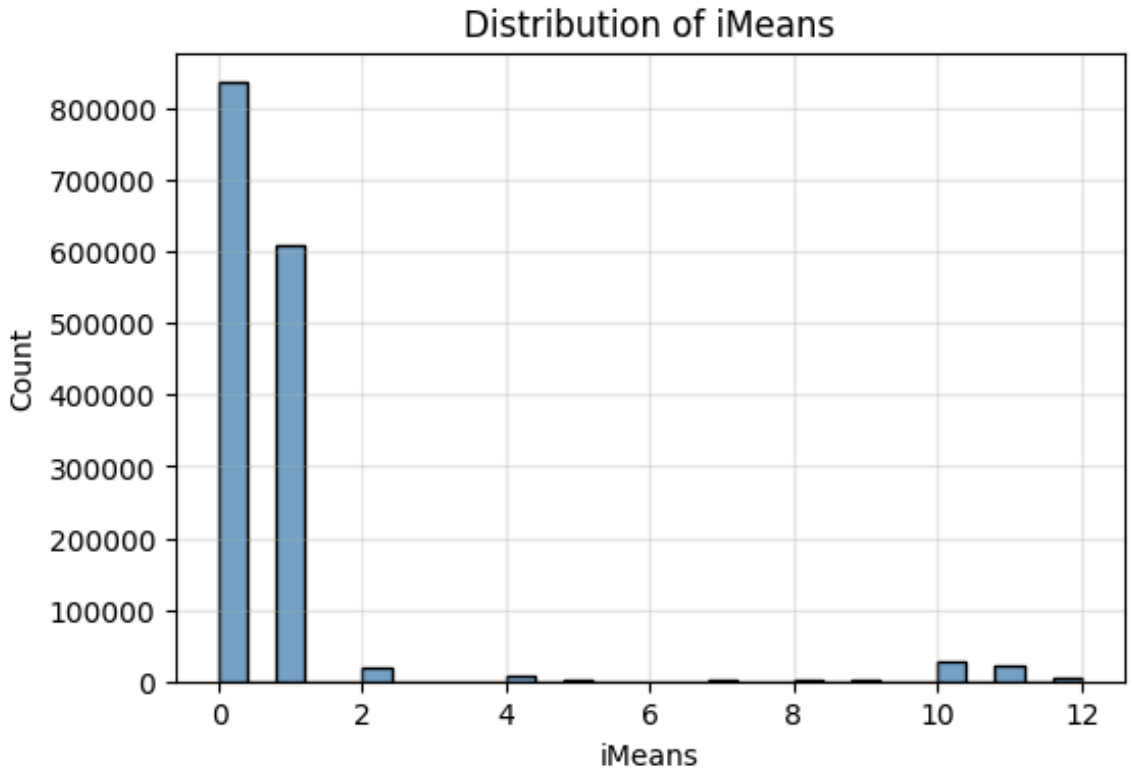
```
            plt.ylabel("Count")
            plt.grid(True, alpha=0.3)
            plt.show()

# Function to plot categorical features
def plot_categorical_univariate(df, features, top_n=10):
    for col in features:
        if col in df.columns:
            plt.figure(figsize=(8,4))
            counts = df[col].value_counts().nlargest(top_n)
            sns.barplot(x=counts.index.astype(str), y=counts
            plt.title(f"Top {top_n} Categories of {col}")
            plt.xlabel(col)
            plt.ylabel("Count")
            plt.xticks(rotation=45)
            plt.grid(True, alpha=0.3)
            plt.show()

# Run univariate plots on your sample dataframe
plot_numeric_univariate(df, numeric_features)
plot_categorical_univariate(df, categorical_features)


# Bivariate EDA
# Distribution of commute time categories
sns.countplot(x=y)
plt.xlabel("Commute time category (0-6)")
plt.ylabel("Count")
plt.title("Distribution of Commute Time Categories")
plt.show()

# Compare commute time categories vs age and earnings
sns.boxplot(x=y, y=df['dAge'])
plt.title("Commute Time Category vs Age")
plt.show()

sns.boxplot(x=y, y=df['dRearning'])
plt.title("Commute Time Category vs Earnings")
plt.show()

from scipy.stats import f_oneway, chi2_contingency

# Target variable
```

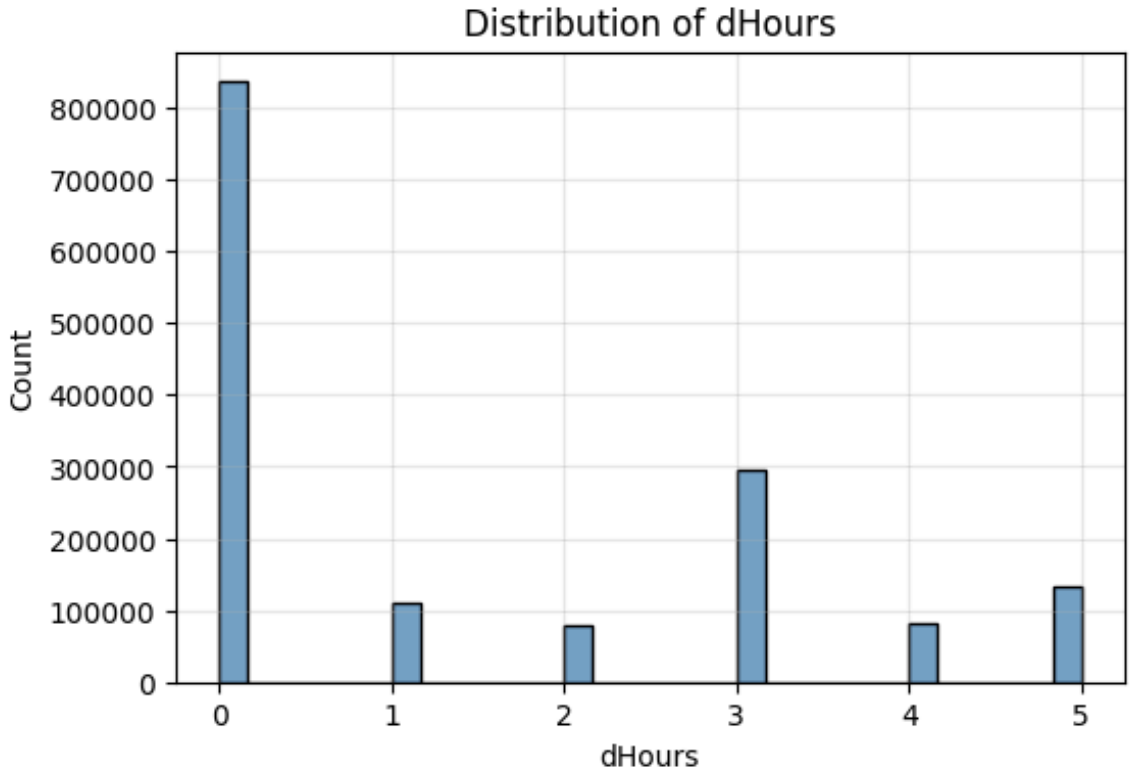
```
y = df['dTravtime'].astype('category')

# Separate numeric and categorical predictors
numeric_features = [
    "dDepart","iMeans","dHours","iClass","dHour89","iRiders"
    "dWeek89","dRearning","iYearsch","iYearwrk","dIncome1",
    "iLooking","dAge","dPwgt1","iRPOB","iTmpabsnt"
]

categorical_features = ["dOccup","dIndustry","iWorklwk"]

# ANOVA for numeric predictors vs. commute time
anova_results = {}
for col in numeric_features:
    if col in df.columns:
        groups = [df[col][y==cat] for cat in y.cat.categorie
        try:
            fstat, pval = f_oneway(*groups)
            anova_results[col] = pval
        except Exception:
            anova_results[col] = np.nan

print("ANOVA p-values (numeric predictors vs. commute time):
print(pd.Series(anova_results).sort_values())

# Chi-square for categorical predictors vs. commute time
chi2_results = {}
for col in categorical_features:
    if col in df.columns:
        table = pd.crosstab(df[col].astype(str), y.astype(st
        try:
            chi2, pval, _, _ = chi2_contingency(table)
            chi2_results[col] = pval
        except Exception:
            chi2_results[col] = np.nan

print("\nChi-square p-values (categorical predictors vs. com
print(pd.Series(chi2_results).sort_values())

import seaborn as sns
import matplotlib.pyplot as plt

# Compute correlation matrix for numeric features
```

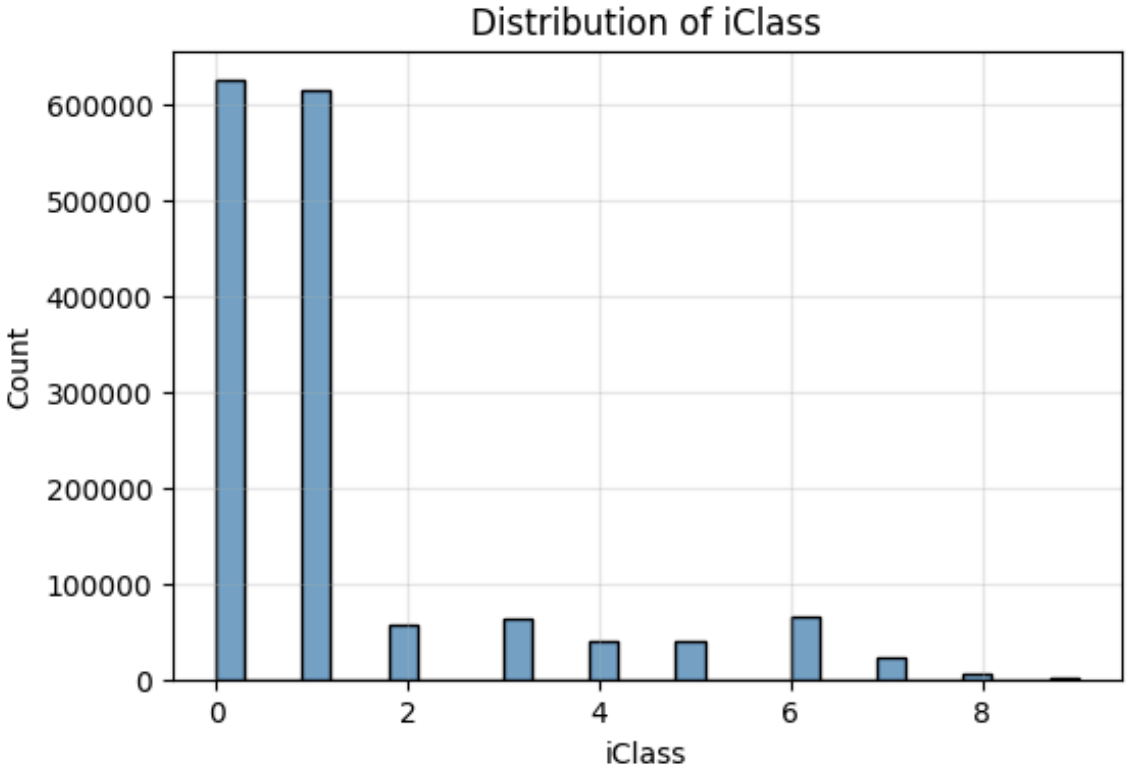
```
numeric_df = df[numeric_features].dropna()
corr_matrix = numeric_df.corr(method="pearson")

# Plot heatmap
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", center
plt.title("Pearson Correlation Heatmap (Numeric Predictors)"
plt.show()
```

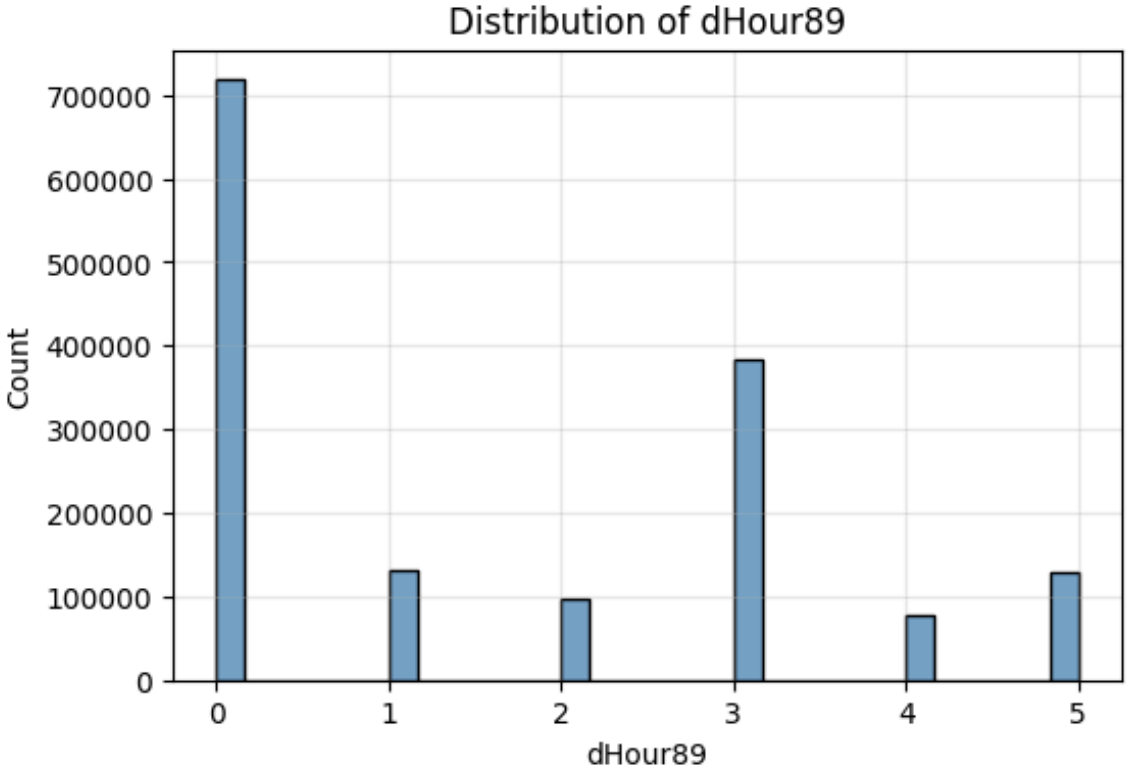
```
(1534490, 68)
```

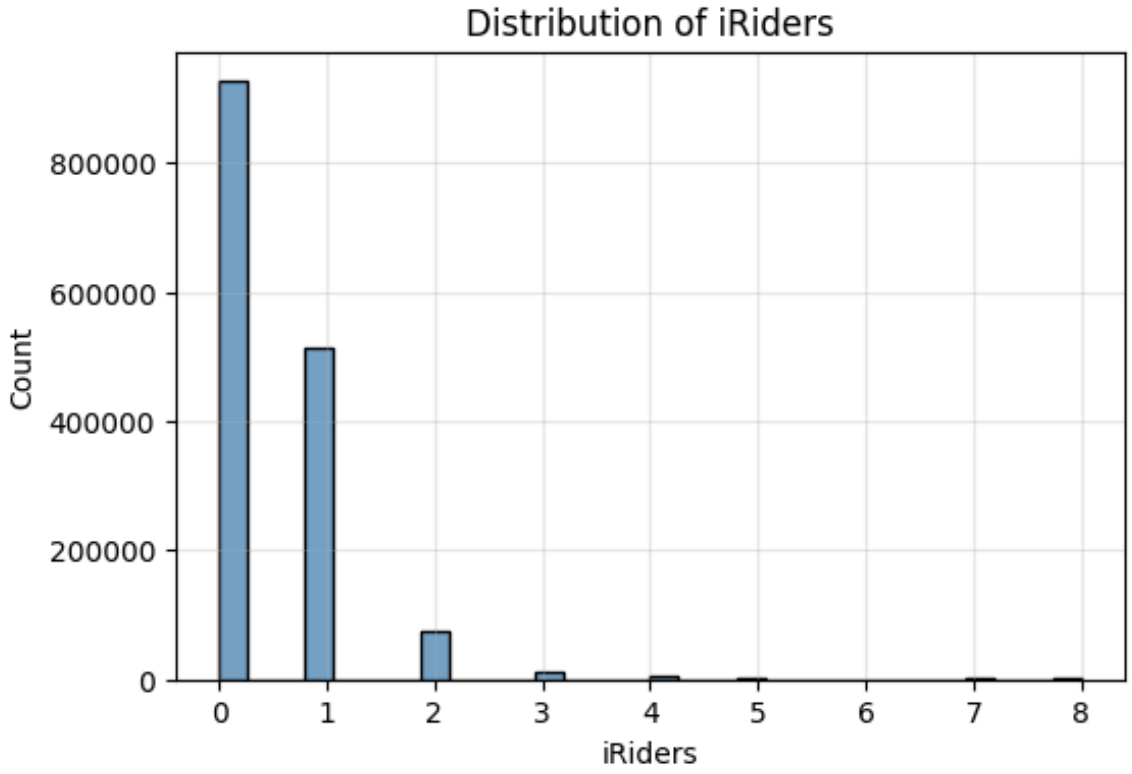


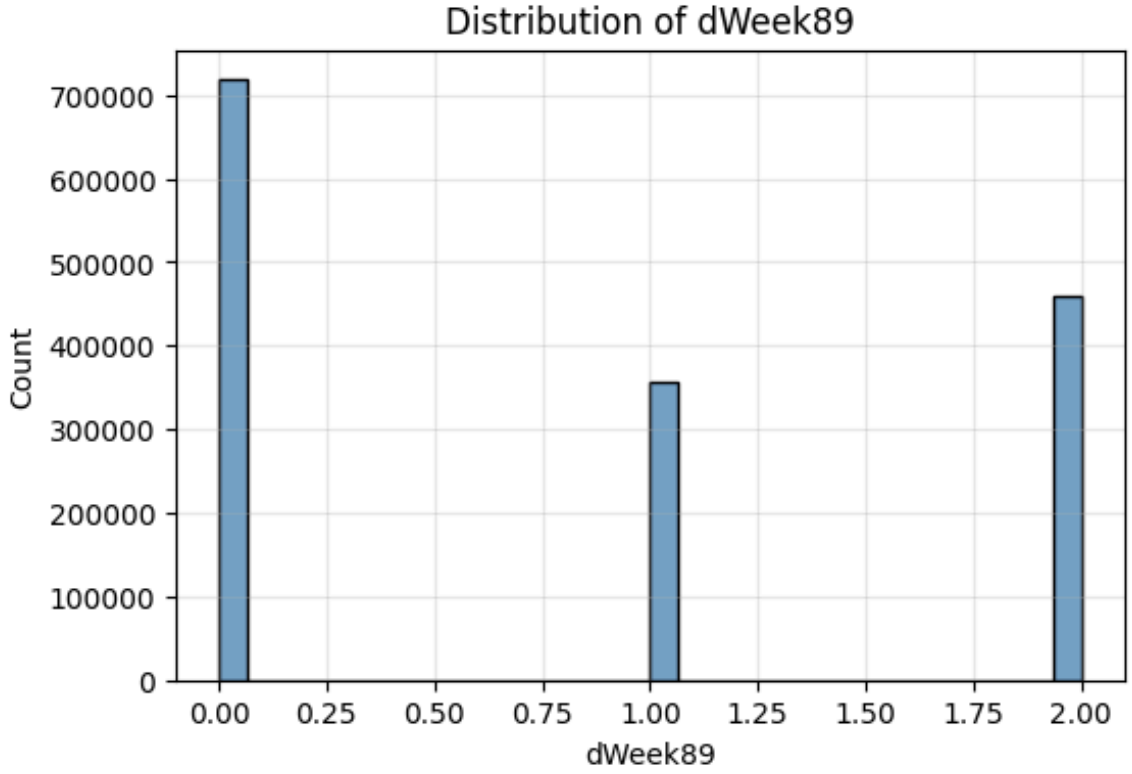

Distribution of iMeans
Count
800000
700000
600000
500000
400000
300000
200000
100000
0
0
2
4
6
8
10
12
iMeans

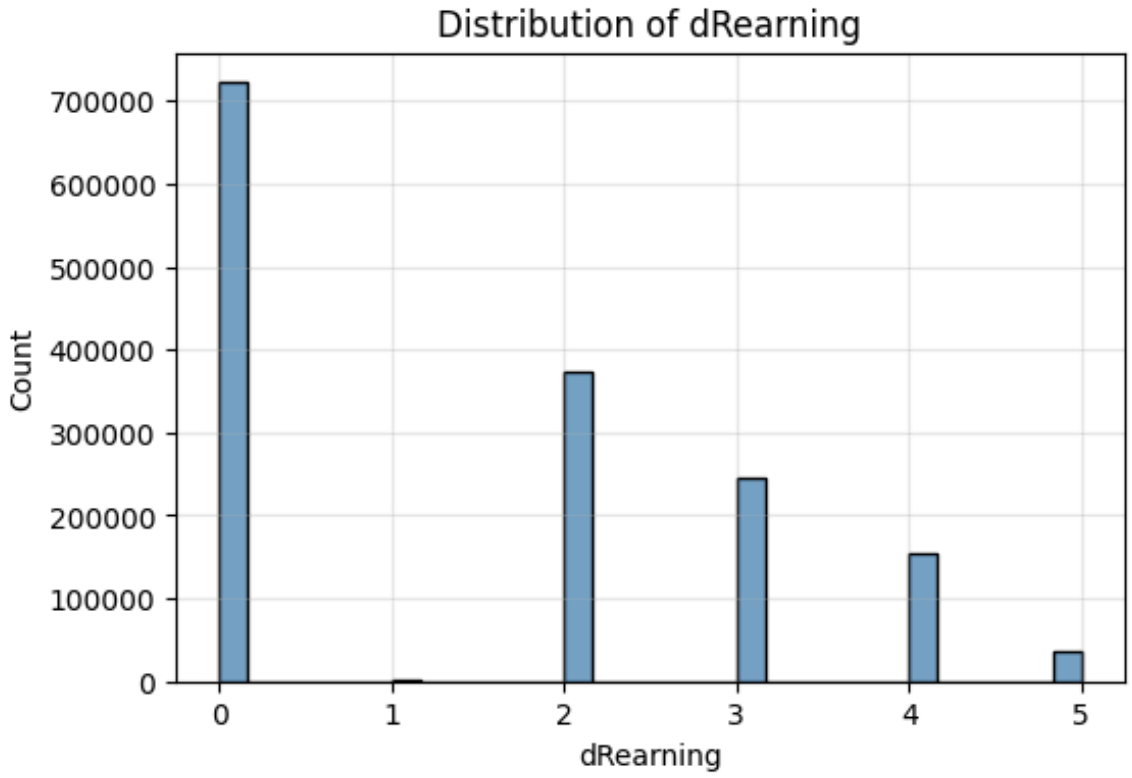

Distribution of dHours
Count
800000
700000
600000
500000
400000
300000
200000
100000
0
0
1
2
3
4
5
dHours

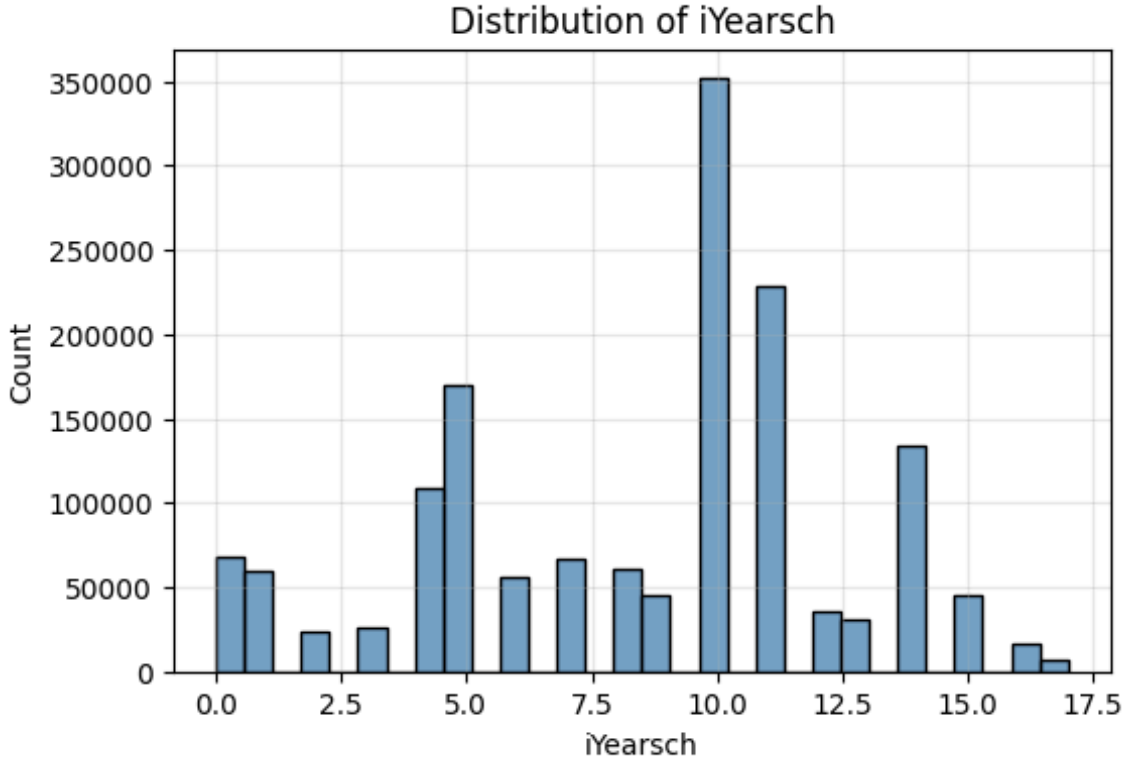

Distribution of iClass
Count
600000
500000
400000
300000
200000
100000
0
0
2
4
6
8
iClass

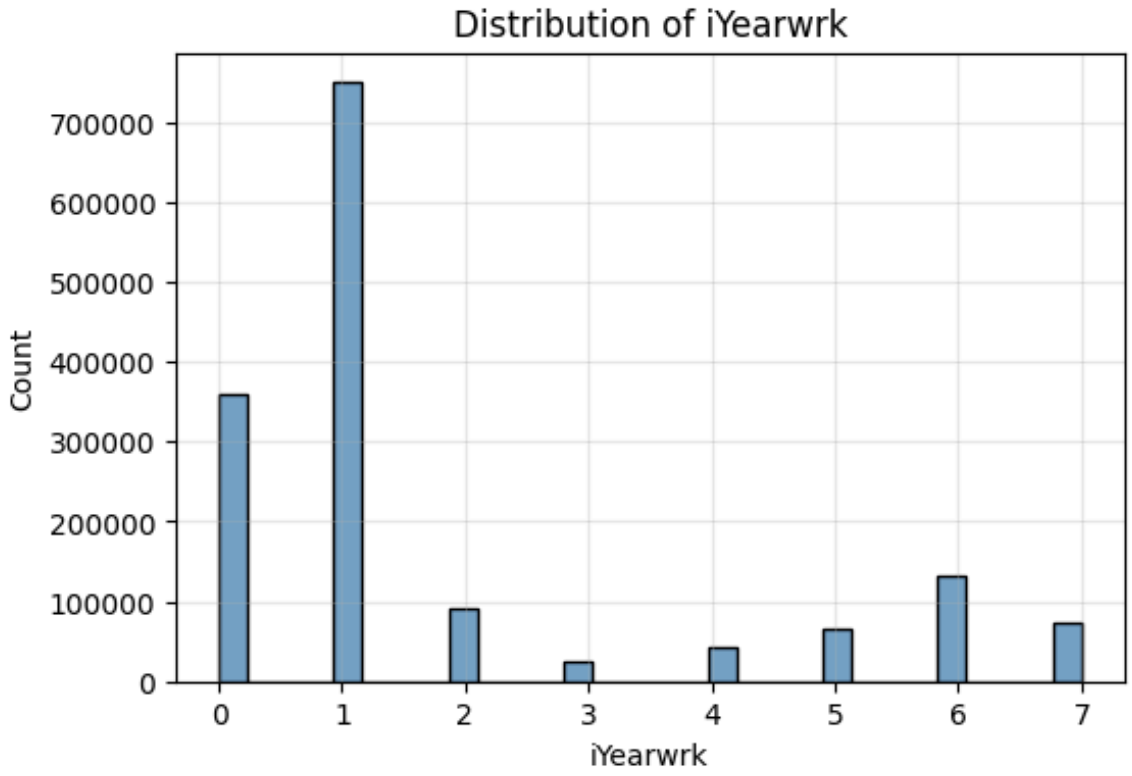

Distribution of dHour89
Count
700000
600000
500000
400000
300000
200000
100000
0
0
1
2
3
4
5
dHour89

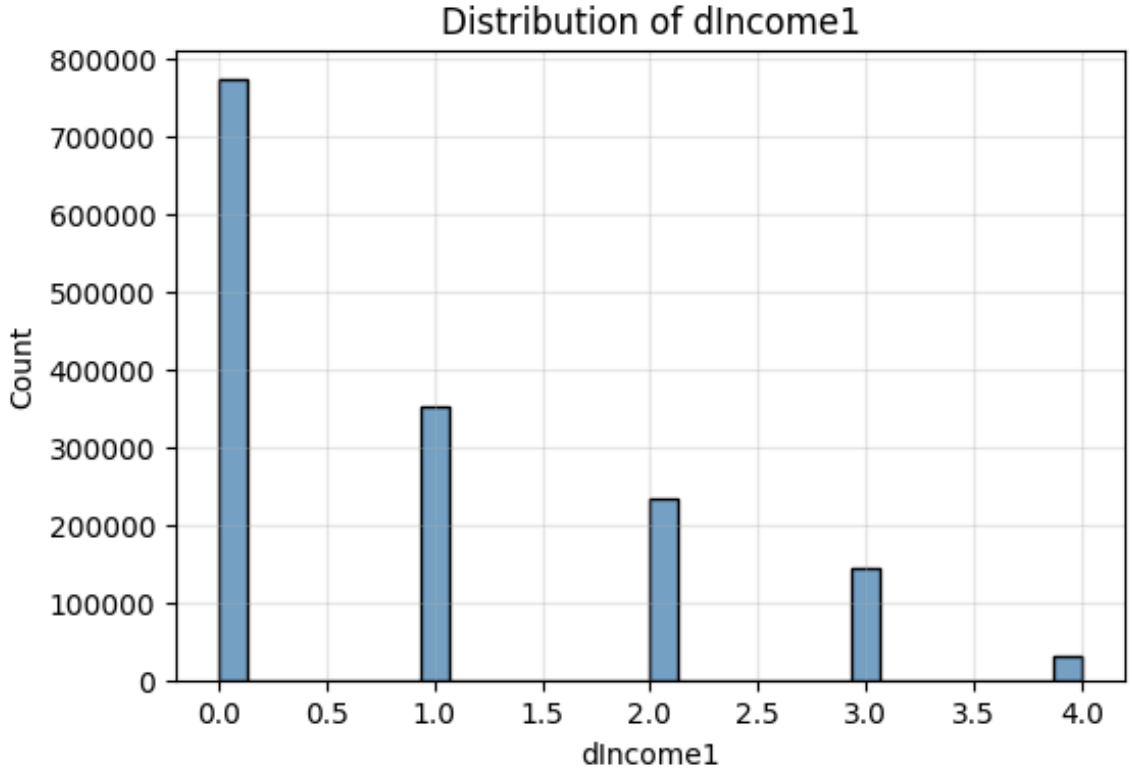

Distribution of iRiders
Count
800000
600000
400000
200000
0
0
1
2
3
4
5
6
7
8
iRiders

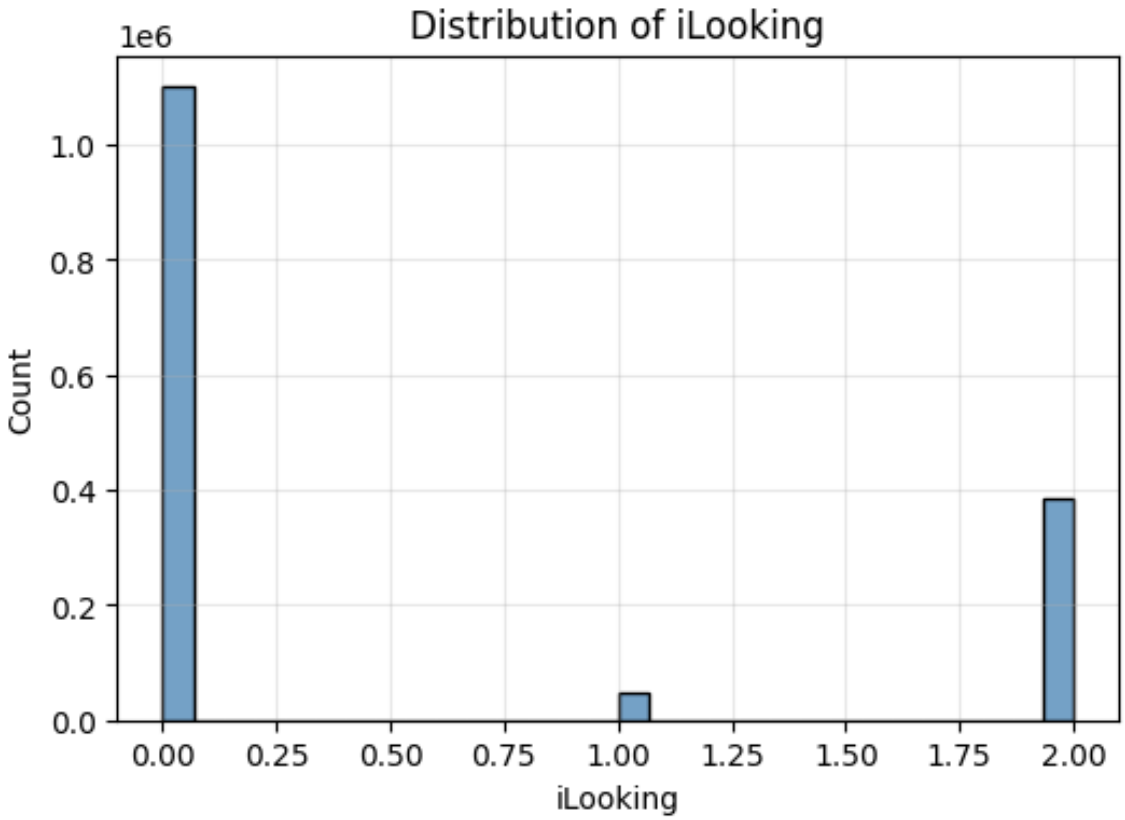

Distribution of dWeek89
Count
700000
600000
500000
400000
300000
200000
100000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
dWeek89

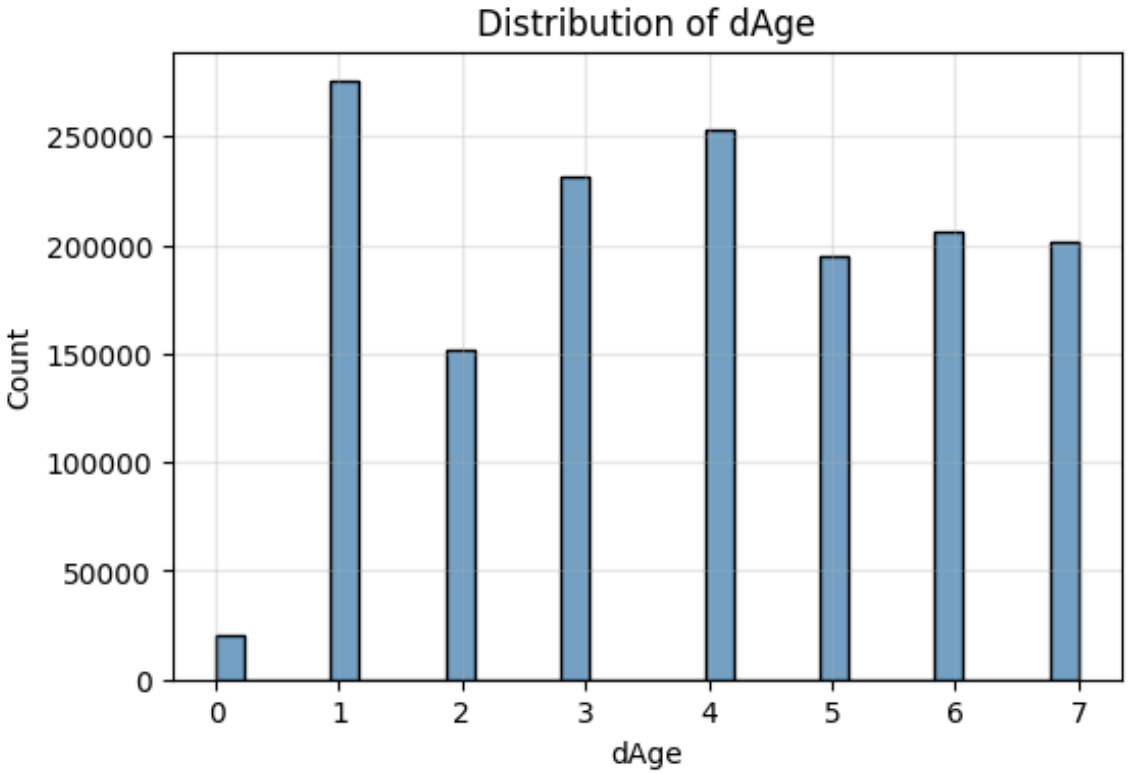

Distribution of dRearning
Count
dRearning
0
100000
200000
300000
400000
500000
600000
700000
0
1
2
3
4
5

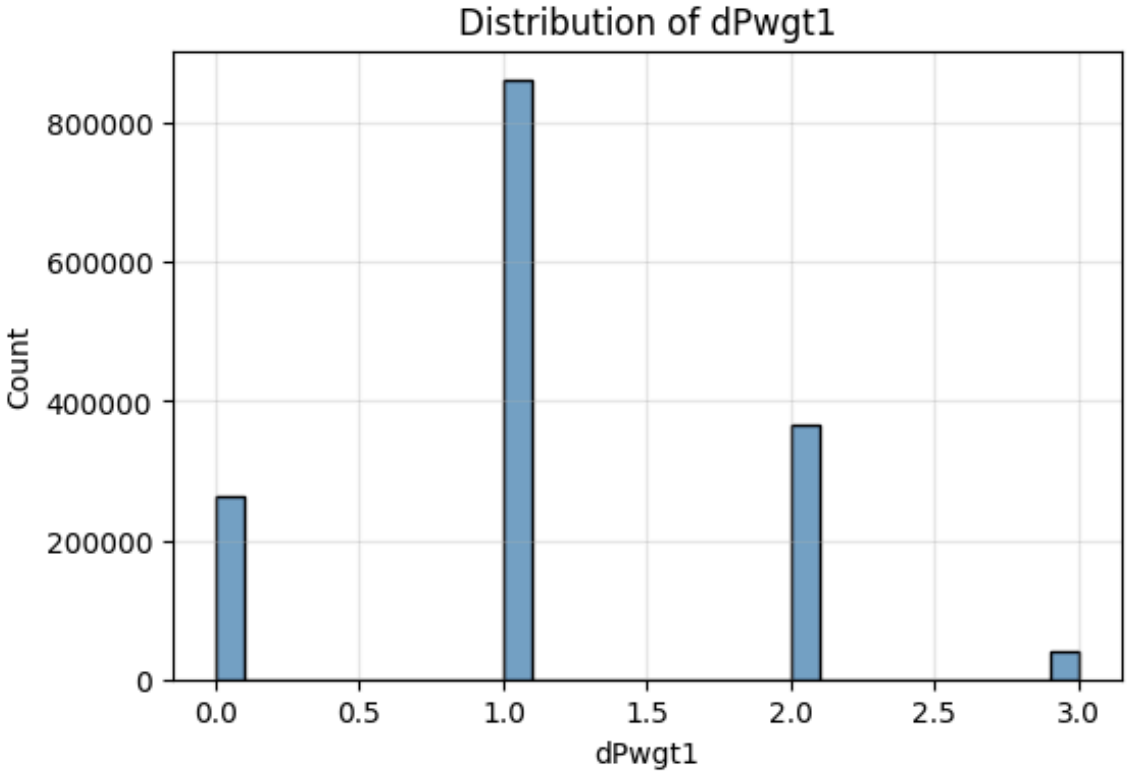

Distribution of iYearsch
Count
iYearsch
0
50000
100000
150000
200000
250000
300000
350000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

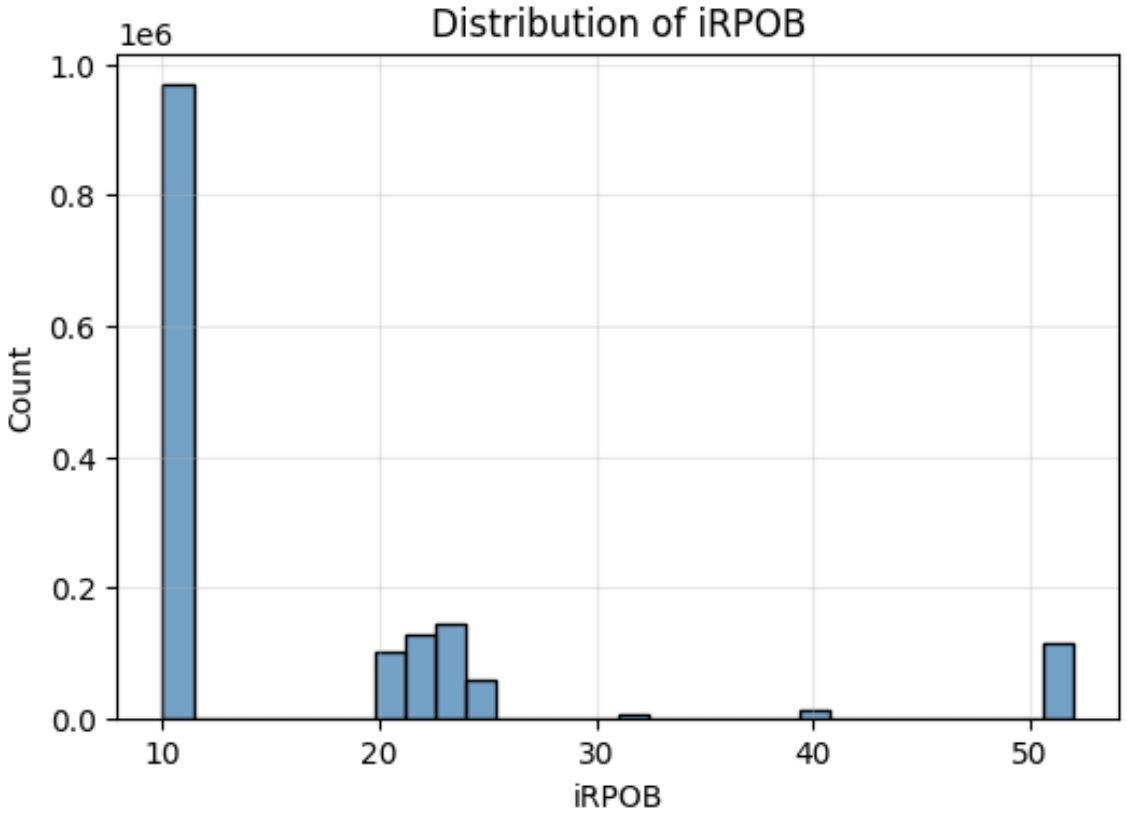

Distribution of iYearwrk
Count
iYearwrk
0
100000
200000
300000
400000
500000
600000
700000
0
1
2
3
4
5
6
7

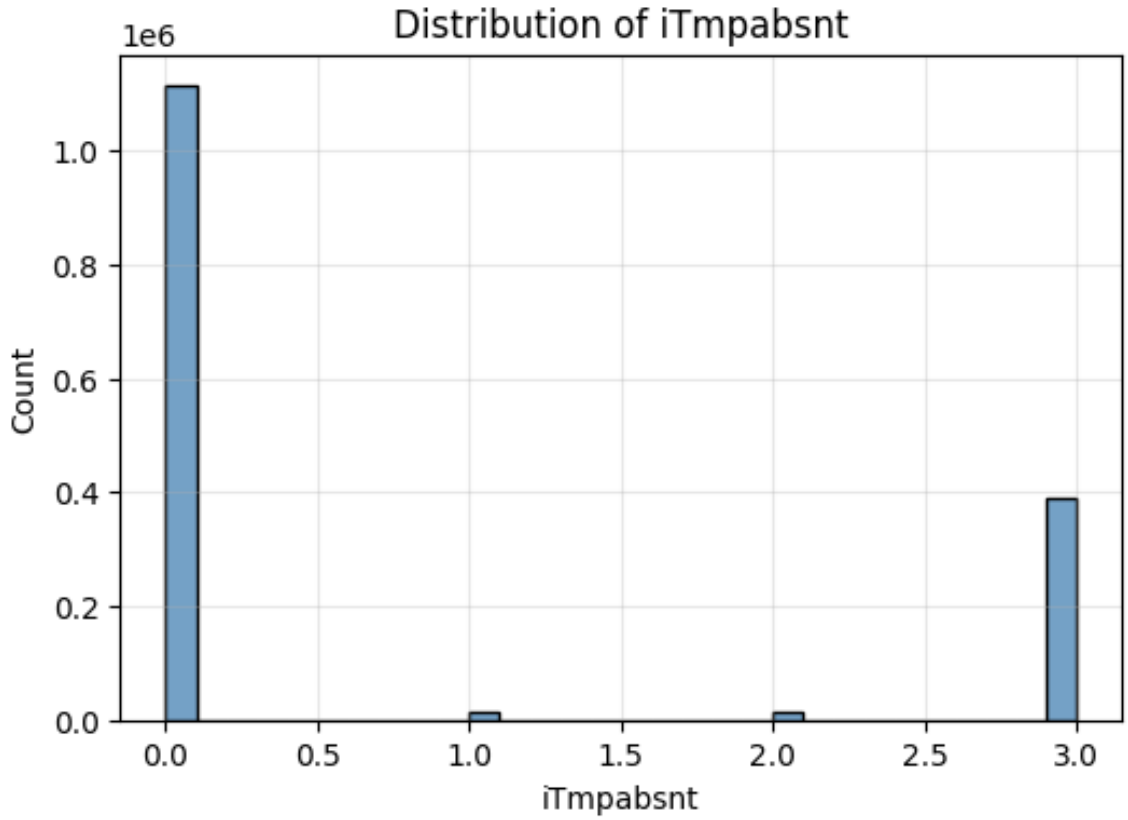

Distribution of dIncome1
Count
dIncome1
0
100000
200000
300000
400000
500000
600000
700000
800000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Distribution of iLooking
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
iLooking

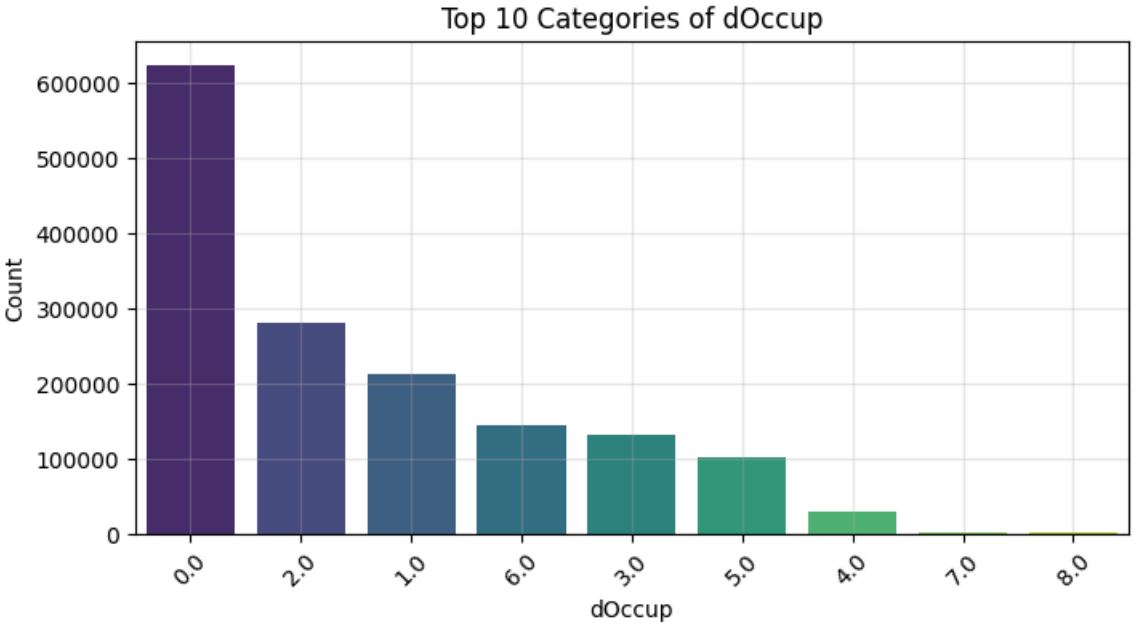

Distribution of dAge
250000
200000
150000
100000
50000
0
Count
0
1
2
3
4
5
6
7
dAge


Distribution of dPwgt1
Count
800000
600000
400000
200000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
dPwgt1

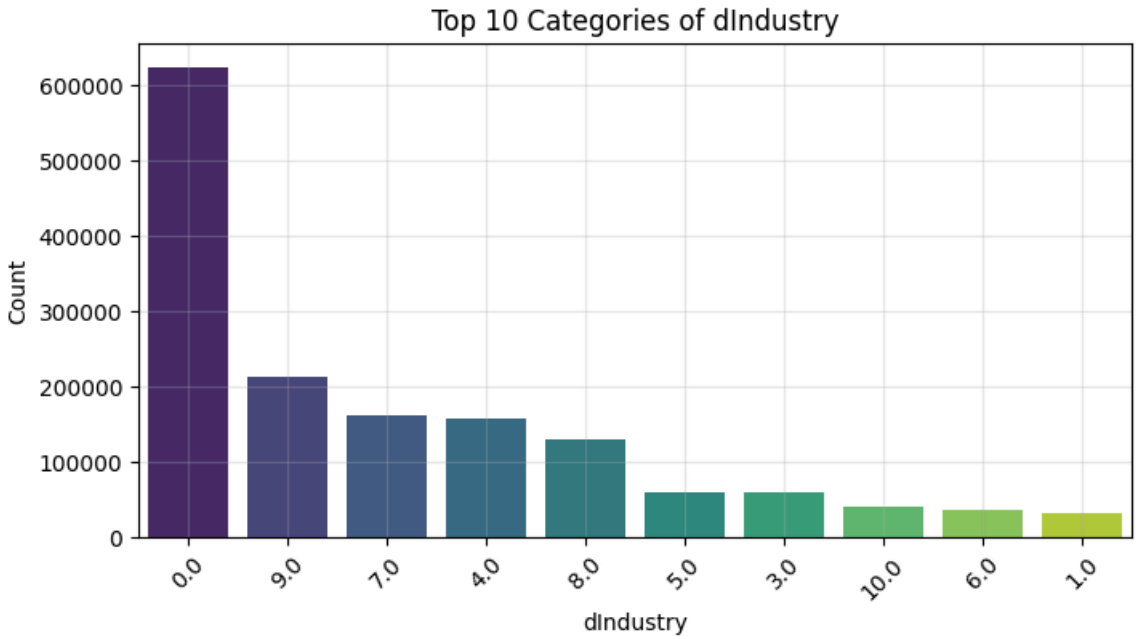

Distribution of iRPOB
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Count
10
20
30
40
50
iRPOB



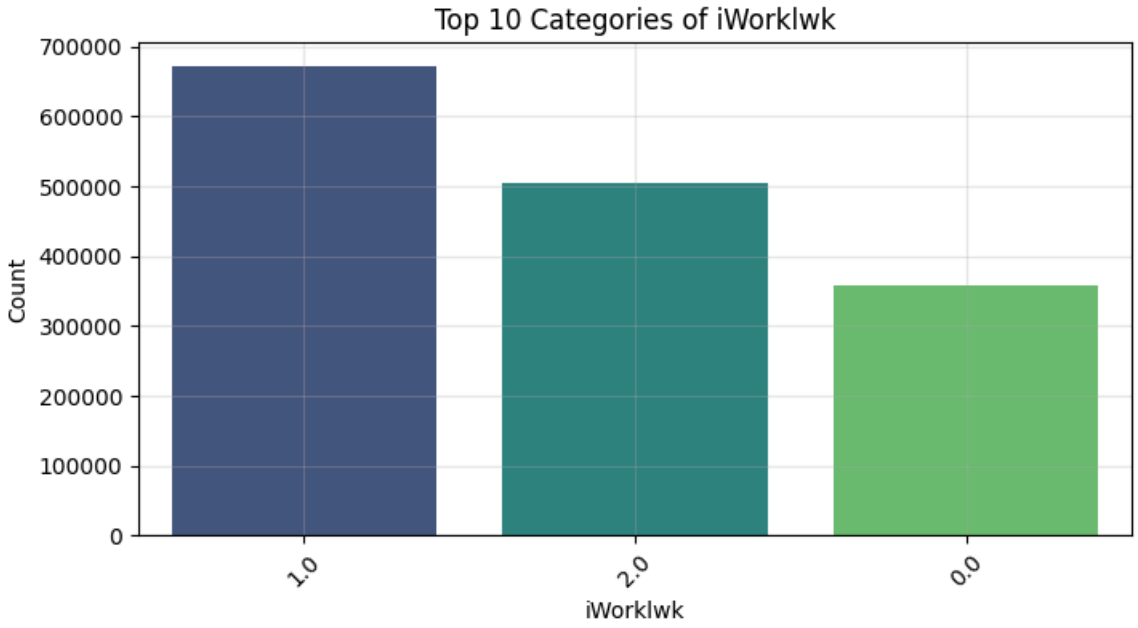

```
<ipython-input-3-3cd4f2b3c74b>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and w
ill be removed in v0.14.0. Assign the `x` variable to `hue` a
nd set `legend=False` for the same effect.

  sns.barplot(x=counts.index.astype(str), y=counts.values, pa
lette="viridis")
```

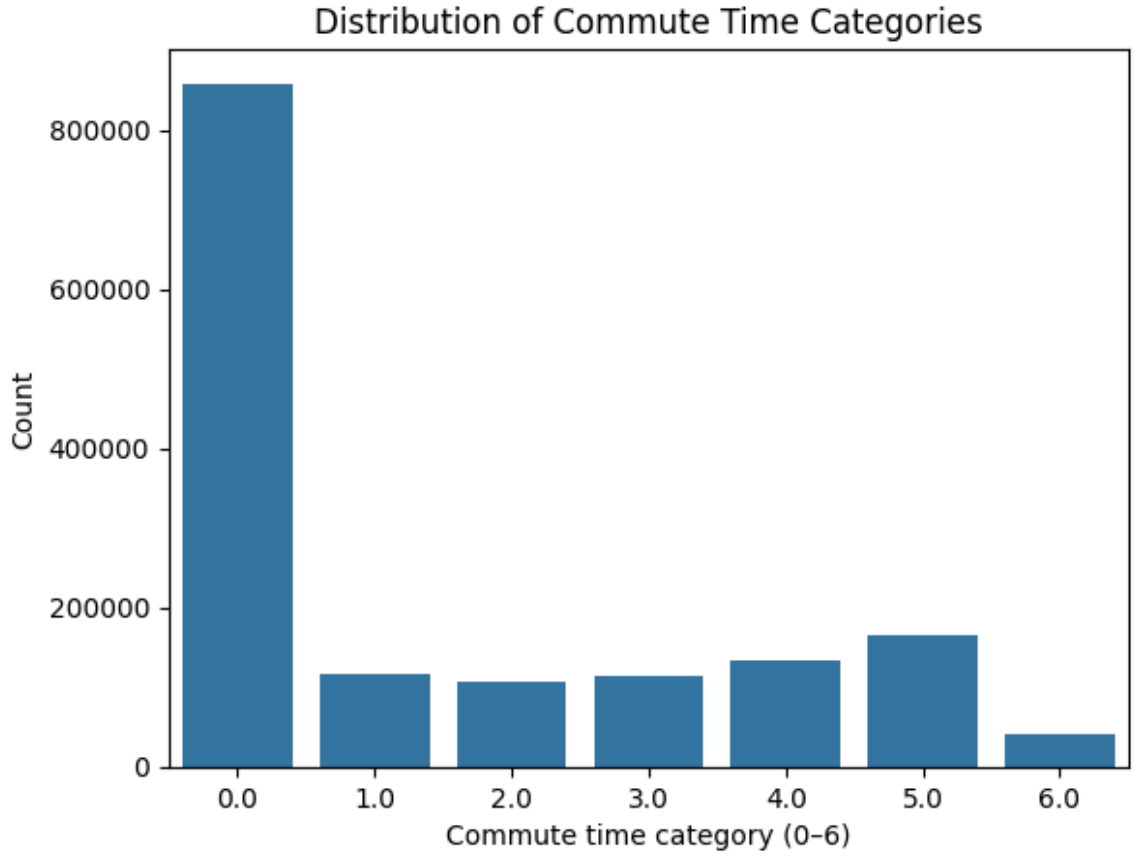


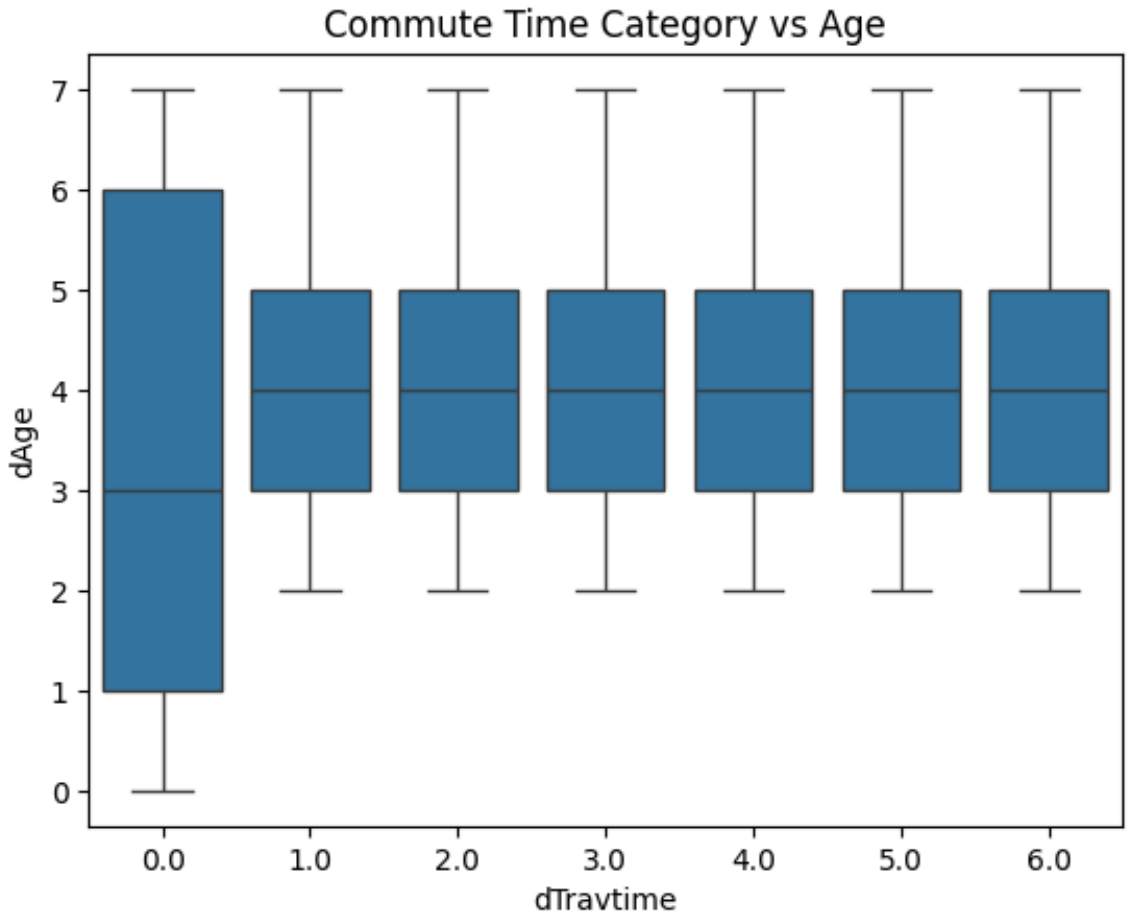
```
<ipython-input-3-3cd4f2b3c74b>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and w
ill be removed in v0.14.0. Assign the `x` variable to `hue` a
nd set `legend=False` for the same effect.

  sns.barplot(x=counts.index.astype(str), y=counts.values, pa
lette="viridis")
```

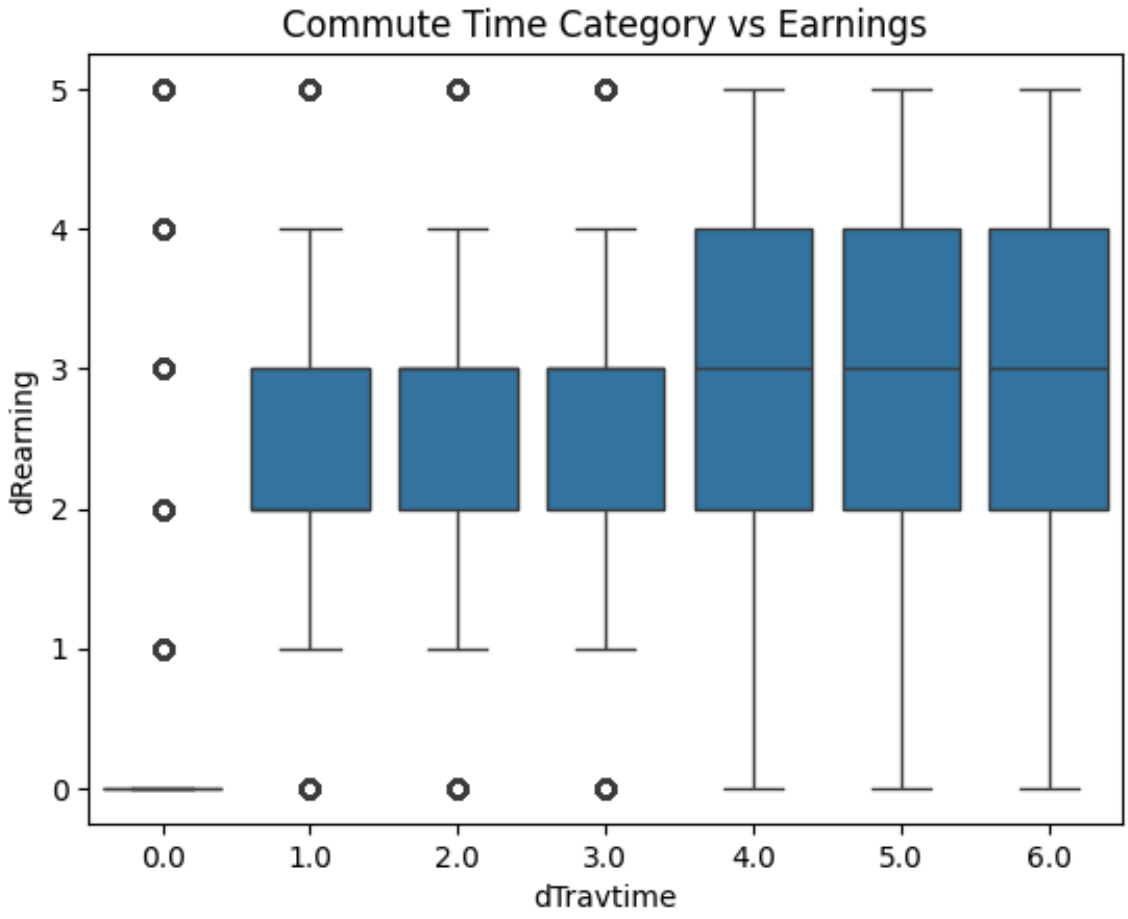



```
<ipython-input-3-3cd4f2b3c74b>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and w
ill be removed in v0.14.0. Assign the `x` variable to `hue` a
nd set `legend=False` for the same effect.

  sns.barplot(x=counts.index.astype(str), y=counts.values, pa
lette="viridis")
```

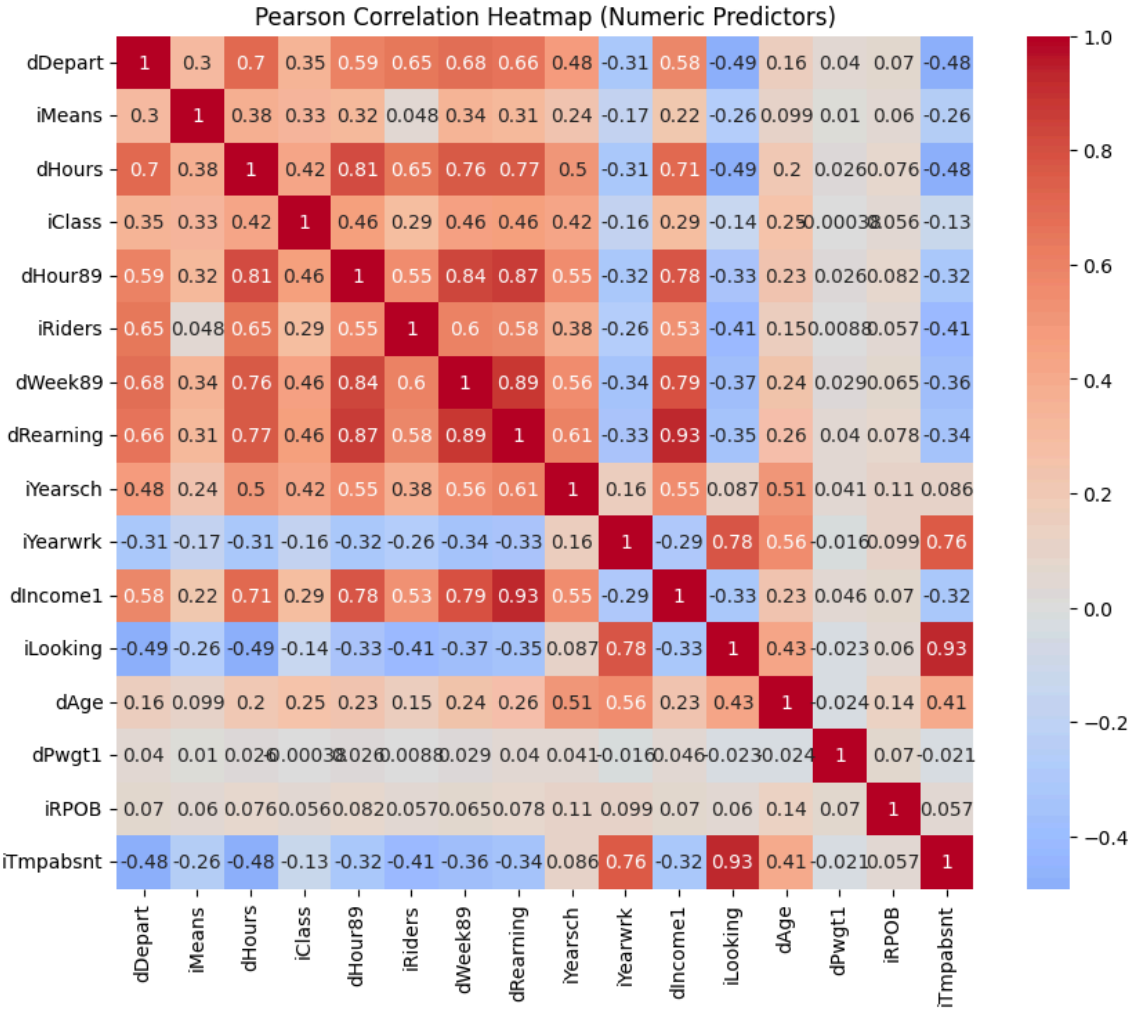

Top 10 Categories of iWorklwk
Count
iWorklwk
700000
600000
500000
400000
300000
200000
100000
0
1.0
2.0
0.0


Distribution of Commute Time Categories
Count
Commute time category (0–6)
800000
600000
400000
200000
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0


Commute Time Category vs Age
dAge
dTravtime
0
1
2
3
4
5
6
7
0.0
1.0
2.0
3.0
4.0
5.0
6.0


Commute Time Category vs Earnings
dRearning
dTravtime
0
1
2
3
4
5
0.0
1.0
2.0
3.0
4.0
5.0
6.0

```
ANOVA p-values (numeric predictors vs. commute time):
dDepart      0.0
iMeans       0.0
dHours       0.0
iClass       0.0
dHour89      0.0
iRiders      0.0
dWeek89      0.0
dRearning    0.0
iYearsch     0.0
iYearwrk     0.0
dIncome1     0.0
iLooking     0.0
dAge         0.0
dPwgt1       0.0
iRPOB        0.0
iTmpabsnt    0.0
dtype: float64

Chi-square p-values (categorical predictors vs. commute tim
e):
dOccup       0.0
dIndustry    0.0
iWorklwk     0.0
dtype: float64
```



In [ ]:
```
# Multinomial Logistic Regression with Backward Stepwise Sel
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_r
from scipy.stats import f_oneway, chi2_contingency
from collections import Counter

# Step 1: Sample 50,000 rows
df_sample = df.sample(n=50000, random_state=42)

# Step 2: Collapse high-cardinality categorical variables
def collapse_categories(series, top_n=4):
    top_cats = series.value_counts().nlargest(top_n).index
```

```
    return series.where(series.isin(top_cats), other='Other'

for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','iL
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

# Ensure categorical columns are all strings
for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','iL
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

# Also cast other categorical predictors to string
categorical_vars = [
    "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstry2",
    "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1",
    "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
    "iDisabl2", "iMobillim", "iVietnam", "iWWII"
]
for col in categorical_vars:
    if col in df_sample.columns:
        df_sample[col] = df_sample[col].astype(str)

# Step 3: Target and predictors
y = df_sample['dTravtime'].astype('category')
predictors = [col for col in df_sample.columns if col not ir

# Step 4: Pre-screen predictors
scores = {}
for col in predictors:
    if pd.api.types.is_numeric_dtype(df_sample[col]):
        groups = [df_sample[col][y==cat] for cat in y.cat.ca
        try:
            _, pval = f_oneway(*groups)
            scores[col] = pval
        except Exception:
            scores[col] = 1.0
    else:
        table = pd.crosstab(df_sample[col].astype(str), y.as
        try:
            _, pval, _, _ = chi2_contingency(table)
            scores[col] = pval
        except Exception:
            scores[col] = 1.0
```

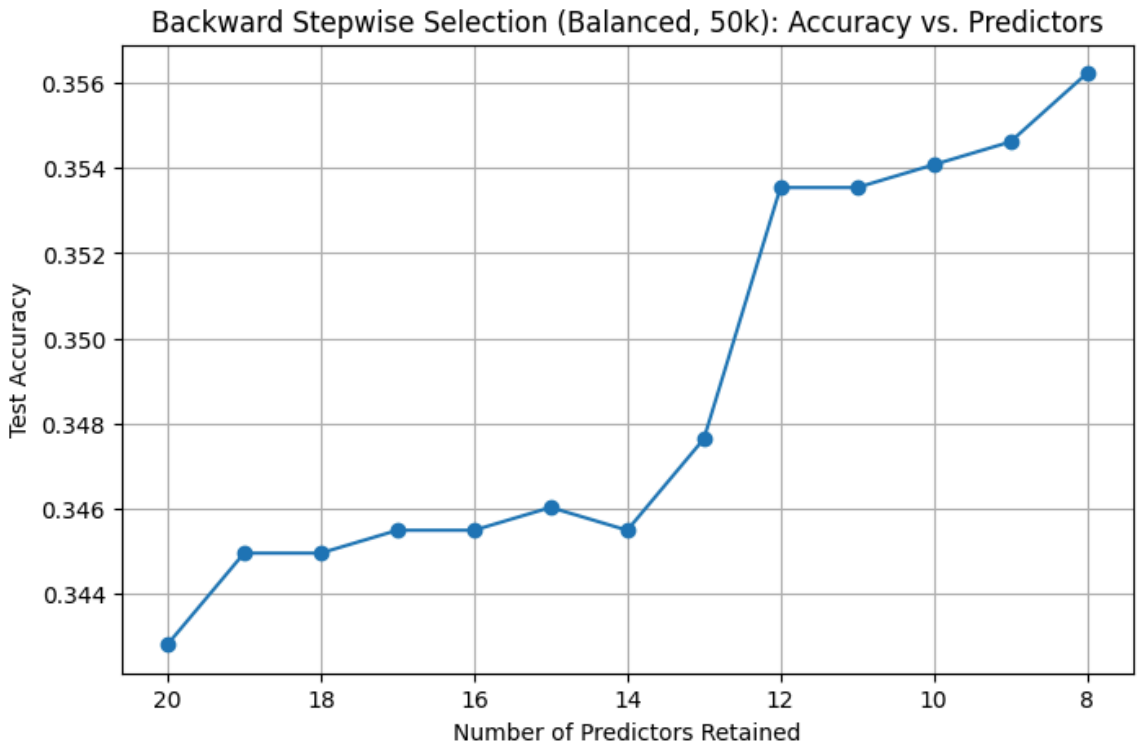
```
# Step 5: Select top 20 predictors
top_predictors = sorted(scores, key=scores.get)[:20]
print("Top 20 predictors:", top_predictors)

# Step 6: Build design matrix using top predictors
X = df_sample[top_predictors]
y_codes = y.cat.codes

# Step 7: Balance classes (undersample majority)
counts = Counter(y_codes)
min_count = min(counts.values())
balanced_idx = np.hstack([
    np.random.choice(np.where(y_codes == cls)[0], min_count,
    for cls in counts.keys()
])
X_balanced = X.iloc[balanced_idx]
y_balanced = y_codes.iloc[balanced_idx]

print("Balanced class counts:", Counter(y_balanced))

# Step 8: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42,
)

# Step 9: Build preprocessing pipeline (restricted to availa
def make_pipeline(predictor_list):
    categorical_vars = [col for col in predictor_list if col
        "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstr
        "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1
        "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
        "iDisabl2", "iMobillim", "iVietnam", "iWWII"
    ]]
    numeric_vars = [col for col in predictor_list if col not

    from sklearn.preprocessing import StandardScaler, OneHot
    from sklearn.compose import ColumnTransformer
    from sklearn.pipeline import Pipeline

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), numeric_vars),
```

```
            ("cat", OneHotEncoder(handle_unknown="ignore"),
        ]
    )

    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("model", LogisticRegression(
            multi_class="multinomial", solver="lbfgs", penal
    ])
    return pipeline

# Initial model
logit_pipeline = make_pipeline(list(X_train.columns))
logit_pipeline.fit(X_train, y_train)
init_acc = accuracy_score(y_test, logit_pipeline.predict(X_t
print("Initial accuracy with 20 predictors (balanced):", ini

# Step 10: Backward stepwise selection
def backward_stepwise(X_train, y_train, X_test, y_test, thre
    included = list(X_train.columns)
    acc_history = []
    pipeline = make_pipeline(included)
    pipeline.fit(X_train, y_train)
    best_acc = accuracy_score(y_test, pipeline.predict(X_tes
    acc_history.append((len(included), best_acc, included.co

    improved = True
    while improved and len(included) > min_vars:
        improved = False
        for col in included:
            trial = [c for c in included if c != col]
            trial_pipeline = make_pipeline(trial)
            trial_pipeline.fit(X_train[trial], y_train)
            acc = accuracy_score(y_test, trial_pipeline.pred
            if acc >= best_acc - threshold:
                included.remove(col)
                best_acc = acc
                acc_history.append((len(included), acc, incl
                print(f"Dropped {col}, accuracy {acc}")
                improved = True
                break
    return acc_history
```

```
acc_history = backward_stepwise(X_train, y_train, X_test, y_

# Step 11: Plot accuracy vs. predictors retained
steps, accs, _ = zip(*acc_history)
plt.figure(figsize=(8,5))
plt.plot(steps, accs, marker='o')
plt.gca().invert_xaxis()
plt.xlabel("Number of Predictors Retained")
plt.ylabel("Test Accuracy")
plt.title("Backward Stepwise Selection (Balanced, 50k): Accu
plt.grid(True)
plt.show()

# Step 12: Extract best model
best_step = max(acc_history, key=lambda x: x[1])
best_vars = best_step[2]
print("Best model predictors:", best_vars)
print("Best model accuracy:", best_step[1])

best_pipeline = make_pipeline(best_vars)
best_pipeline.fit(X_train[best_vars], y_train)

# Coefficient table
best_model = best_pipeline.named_steps["model"]
coef_df = pd.DataFrame(best_model.coef_, columns=best_pipeli
coef_df.index = [f"Class {c}" for c in best_model.classes_]
print("Coefficient table for best model (balanced, 50k):")
print(coef_df)

# ROC curves for best model
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

y_test_bin = label_binarize(y_test, classes=np.unique(y_test
y_score = best_pipeline.predict_proba(X_test[best_vars])

fpr, tpr, roc_auc = {}, {}, {}
plt.figure(figsize=(8,6))
for i in range(y_test_bin.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_a
```

```
plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC (OvR) for Best Multinomial Logisti
plt.legend()
plt.show()

# We get between 32 and 39% accuracy with 12 predictors reta

# Multinomial logistic regression has served us well as a tr
# That said, the approach has limitations. It assumes linear
# In short, multinomial logit is a strong pedagogical and di
```

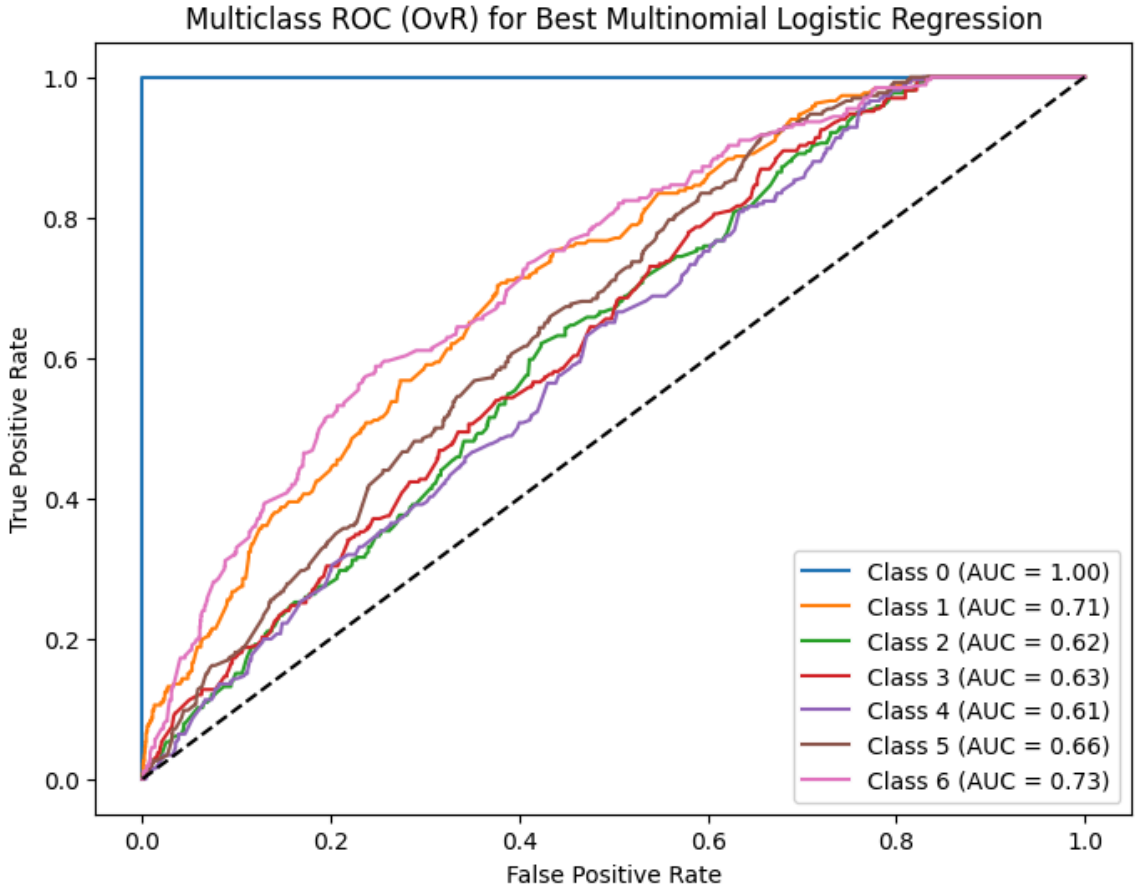
```
Top 20 predictors: ['dAge', 'iClass', 'dDepart', 'iDisabl1',
'iDisabl2', 'dHour89', 'dHours', 'dIncome1', 'dIncome5', 'dIn
dustry', 'iLang1', 'iLooking', 'iMarital', 'iMeans', 'iMilita
ry', 'iMobility', 'iMobillim', 'dOccup', 'iPerscare', 'dPover
ty']
Balanced class counts: Counter({5: 1331, 0: 1331, 2: 1331, 4:
1331, 3: 1331, 1: 1331, 6: 1331})
Initial accuracy with 20 predictors (balanced): 0.34281115879
82833
Dropped iClass, accuracy 0.3449570815450644
Dropped iDisabl1, accuracy 0.3449570815450644
Dropped dAge, accuracy 0.34549356223175964
Dropped iDisabl2, accuracy 0.34549356223175964
Dropped dHour89, accuracy 0.34603004291845496
Dropped dHours, accuracy 0.34549356223175964
Dropped dIncome5, accuracy 0.34763948497854075
Dropped iLang1, accuracy 0.35354077253218885
Dropped iLooking, accuracy 0.35354077253218885
Dropped iMobility, accuracy 0.3540772532188841
Dropped iMarital, accuracy 0.3546137339055794
Dropped iMobillim, accuracy 0.3562231759656652
```


Backward Stepwise Selection (Balanced, 50k): Accuracy vs. Predictors
Test Accuracy
0.356
0.354
0.352
0.350
0.348
0.346
0.344
20
18
16
14
12
10
8
Number of Predictors Retained

```
Best model predictors: ['dDepart', 'dIncome1', 'dIndustry',
'iMeans', 'iMilitary', 'dOccup', 'iPerscare', 'dPoverty']
Best model accuracy: 0.3562231759656652
Coefficient table for best model (balanced, 50k):
         num__dDepart  num__dIncome1  num__iMeans  num__iPers
care  \
Class 0     -8.571311      -0.730035    -0.088052       -0.15
3620
Class 1      1.666998      -0.107146     0.220191        0.17
9954
Class 2      1.539204       0.007208     0.081969        0.11
7289
Class 3      1.621859       0.065694    -0.119641       -0.14
0720
Class 4      1.557053       0.168993    -0.197709       -0.08
2571
Class 5      1.267848       0.284499    -0.077638        0.07
6030
Class 6      0.918350       0.310788     0.180880        0.00
3639

         num__dPoverty  cat__dIndustry_0.0  cat__dIndustry_7.
0  \
Class 0     -0.048645            0.584788          -0.43495
6
Class 1     -0.080234           -0.066610           0.13678
7
Class 2      0.012837           -0.087874           0.21537
4
Class 3      0.008620           -0.104121           0.10281
4
Class 4      0.094931           -0.090354           0.03589
4
Class 5      0.034221           -0.092932           0.00800
5
Class 6     -0.021730           -0.142897          -0.06391
8

         cat__dIndustry_9.0  cat__dIndustry_Other  cat__iMili
tary_0.0  \
Class 0          -0.101257            -0.415726
0.196323
Class 1           0.131661            -0.136821          -
```

```
0.014312
Class 2            0.073186             -0.131235           -
0.023320
Class 3            0.021155              0.040706           -
0.054329
Class 4            0.002019              0.114746           -
0.039497
Class 5           -0.104756              0.252170           -
0.023635
Class 6           -0.022007              0.276160           -
0.041230

         cat__iMilitary_1.0  cat__iMilitary_2.0  cat__iMilita
ry_3.0  \
Class 0          -0.243171           -0.223810            0.
091776
Class 1           0.142776            0.173319           -0.
173485
Class 2           0.467222           -0.110083           -0.
226605
Class 3           0.491270           -0.142478           -0.
135350
Class 4           0.080830           -0.062747            0.
072730
Class 5          -0.210814            0.080470            0.
052652
Class 6          -0.728114            0.285328            0.
318282

         cat__iMilitary_4.0  cat__dOccup_0.0  cat__dOccup_1.0
\
Class 0          -0.188269         0.584788        -0.184130
Class 1          -0.063282        -0.066610         0.041648
Class 2          -0.037763        -0.087874         0.064463
Class 3          -0.098560        -0.104121        -0.027677
Class 4           0.010988        -0.090354         0.041825
Class 5           0.163815        -0.092932         0.096909
Class 6           0.213072        -0.142897        -0.033038

         cat__dOccup_2.0  cat__dOccup_Other
Class 0        -0.087370          -0.680439
Class 1        -0.057498           0.147477
Class 2         0.007819           0.085044
```

```
Class 3          0.040121          0.152230
Class 4          0.035333          0.075501
Class 5         -0.038769          0.097278
Class 6          0.100364          0.122909
```



In [ ]:
```python
# K-Nearest Neighbors (KNN) with PCA
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, classi
from sklearn.preprocessing import StandardScaler, OneHotEncc
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from collections import Counter

# Step 1: Sample 50,000 rows
```

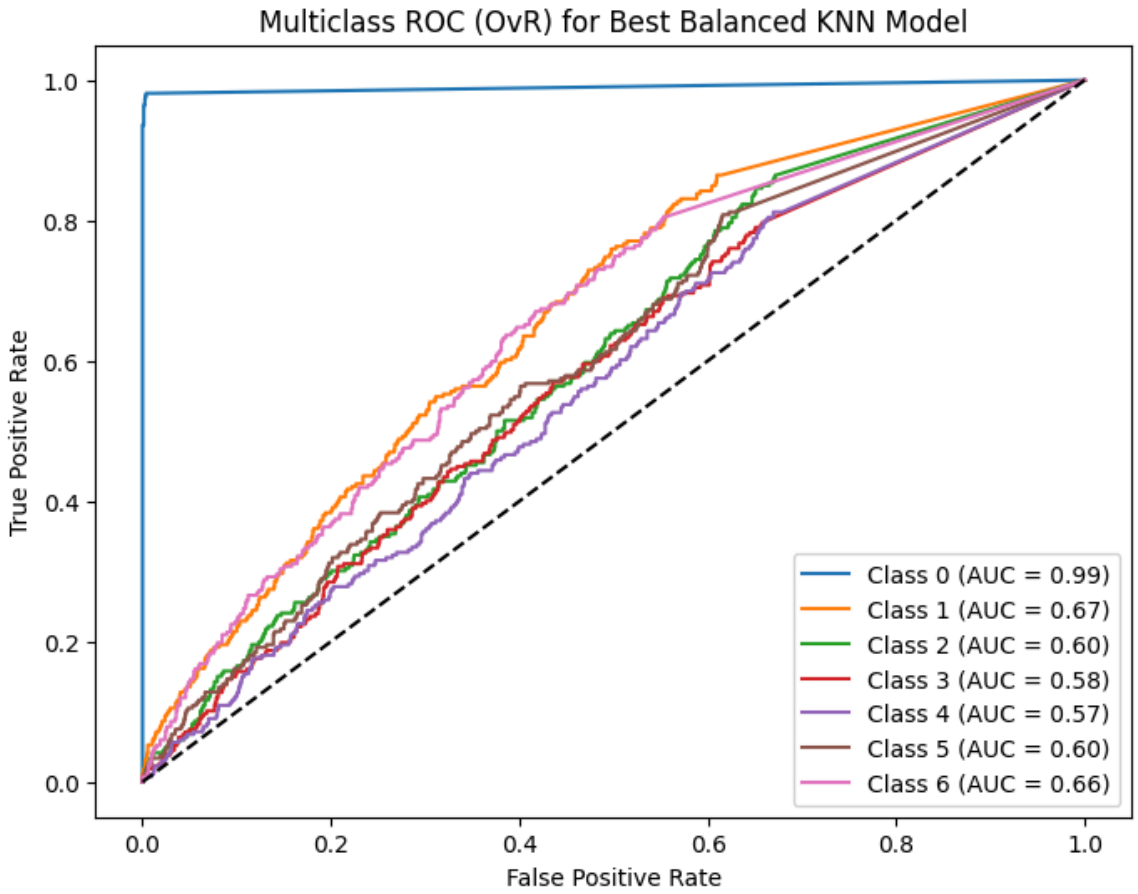
```
df_sample = df.sample(n=50000, random_state=42)

# Step 2: Collapse high-cardinality categorical variables
def collapse_categories(series, top_n=4):
    top_cats = series.value_counts().nlargest(top_n).index
    return series.where(series.isin(top_cats), other='Other'

for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','iL
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

# Cast other categorical predictors to string
categorical_vars_master = [
    "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstry2",
    "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1",
    "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
    "iDisabl2", "iMobillim", "iVietnam", "iWWII"
]
for col in categorical_vars_master:
    if col in df_sample.columns:
        df_sample[col] = df_sample[col].astype(str)

# Step 3: Target and predictors
y = df_sample['dTravtime'].astype('category')
predictors = [col for col in df_sample.columns if col not ir
X = df_sample[predictors]
y_codes = y.cat.codes

# Step 4: Balance classes (undersample majority)
counts = Counter(y_codes)
min_count = min(counts.values())
balanced_idx = np.hstack([
    np.random.choice(np.where(y_codes == cls)[0], min_count,
    for cls in counts.keys()
])
X_balanced = X.iloc[balanced_idx]
y_balanced = y_codes.iloc[balanced_idx]

print("Balanced class counts:", Counter(y_balanced))

# Step 5: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42,
```

```
)

# Step 6: Build preprocessing pipeline
def make_pipeline_knn(predictor_list, n_neighbors=5):
    categorical_vars = [col for col in predictor_list if col
    numeric_vars = [col for col in predictor_list if col not

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), numeric_vars),
            ("cat", OneHotEncoder(handle_unknown="ignore"),
        ]
    )

    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("pca", PCA(n_components=0.95, random_state=42)),  #
        ("model", KNeighborsClassifier(n_neighbors=n_neighbo
    ])
    return pipeline

# Step 7: Hyperparameter tuning for n_neighbors
results = []
best_model = None
best_f1 = -1

for k in [3, 5, 7, 9]:
    knn_pipeline = make_pipeline_knn(list(X_train.columns),
    knn_pipeline.fit(X_train, y_train)
    y_pred = knn_pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    macro_f1 = f1_score(y_test, y_pred, average="macro")
    results.append((k, acc, macro_f1))
    print(f"\nKNN (k={k})")
    print("Test accuracy:", acc)
    print("Macro F1:", macro_f1)
    print("Classification report:")
    print(classification_report(y_test, y_pred))
    if macro_f1 > best_f1:
        best_f1 = macro_f1
        best_model = knn_pipeline

# Step 8: ROC curves for best model
```

```
y_test_bin = label_binarize(y_test, classes=np.unique(y_test
y_score = best_model.predict_proba(X_test)

fpr, tpr, roc_auc = {}, {}, {}
plt.figure(figsize=(8,6))
for i in range(y_test_bin.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_a

plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC (OvR) for Best Balanced KNN Model"
plt.legend()
plt.show()

# Our PCA cumulative explained variance plot shows that near
# We see that k=5 gives the best test accuracy of around 30%
```

```
Balanced class counts: Counter({5: 1331, 0: 1331, 2: 1331, 4:
1331, 3: 1331, 1: 1331, 6: 1331})

KNN (k=3)
Test accuracy: 0.3084763948497854
Macro F1: 0.311146807929259
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.95      0.98       266
           1       0.22      0.23      0.23       266
           2       0.16      0.18      0.17       266
           3       0.22      0.21      0.21       267
           4       0.19      0.18      0.19       266
           5       0.19      0.19      0.19       266
           6       0.23      0.21      0.22       267

    accuracy                           0.31      1864
   macro avg       0.31      0.31      0.31      1864
weighted avg       0.31      0.31      0.31      1864


KNN (k=5)
Test accuracy: 0.3084763948497854
Macro F1: 0.3112538395355052
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       266
           1       0.23      0.24      0.23       266
           2       0.17      0.20      0.18       266
           3       0.18      0.18      0.18       267
           4       0.18      0.18      0.18       266
           5       0.21      0.22      0.22       266
           6       0.23      0.20      0.22       267

    accuracy                           0.31      1864
   macro avg       0.31      0.31      0.31      1864
weighted avg       0.31      0.31      0.31      1864


KNN (k=7)
Test accuracy: 0.3133047210300429
```

```
Macro F1: 0.3151779927794175
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       266
           1       0.25      0.26      0.26       266
           2       0.20      0.23      0.21       266
           3       0.19      0.18      0.18       267
           4       0.17      0.18      0.18       266
           5       0.21      0.21      0.21       266
           6       0.21      0.19      0.20       267

    accuracy                           0.31      1864
   macro avg       0.32      0.31      0.32      1864
weighted avg       0.32      0.31      0.32      1864


KNN (k=9)
Test accuracy: 0.3197424892703863
Macro F1: 0.3214678334062072
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       266
           1       0.27      0.29      0.28       266
           2       0.18      0.20      0.19       266
           3       0.19      0.19      0.19       267
           4       0.18      0.17      0.17       266
           5       0.20      0.20      0.20       266
           6       0.25      0.23      0.24       267

    accuracy                           0.32      1864
   macro avg       0.32      0.32      0.32      1864
weighted avg       0.32      0.32      0.32      1864
```

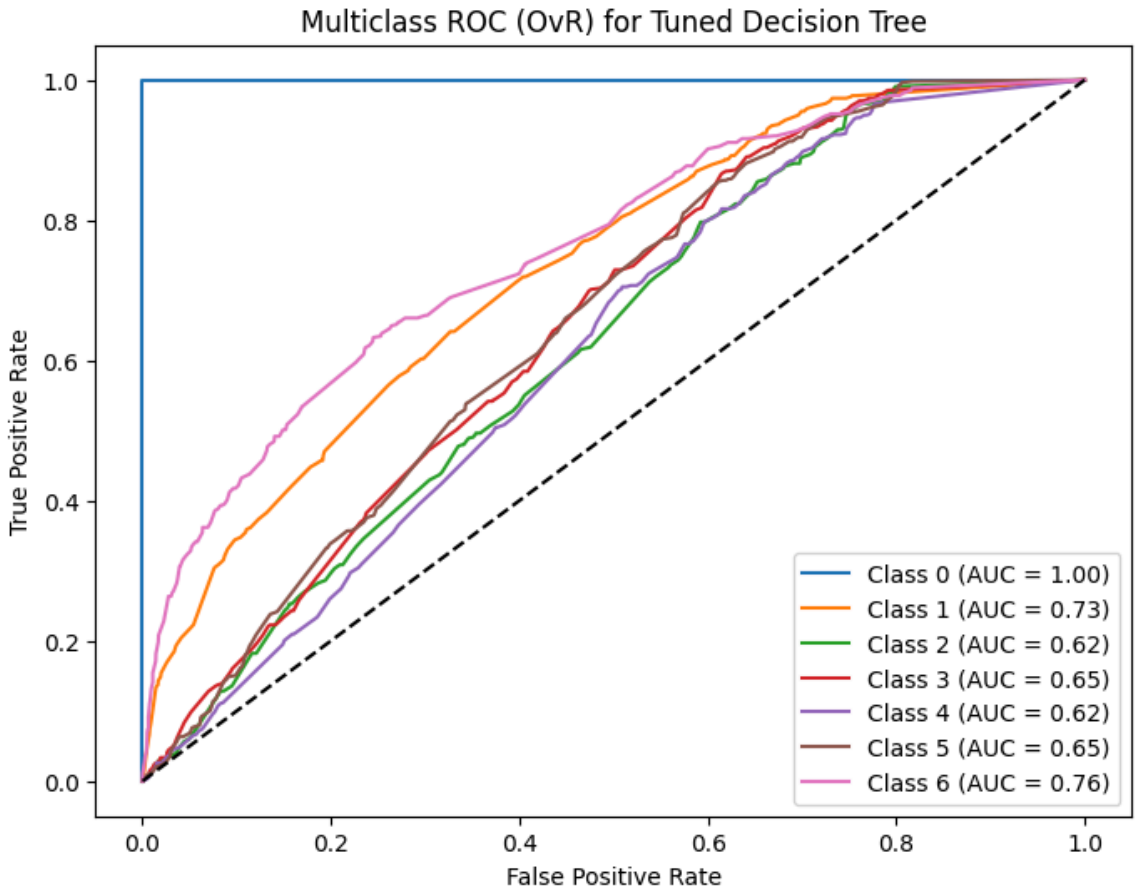


In [ ]:
```
# Single Tree, pruned
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSe
from sklearn.preprocessing import label_binarize
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_r
import matplotlib.pyplot as plt
from collections import Counter

#  Step 1: Sample 100,000 rows
df_sample = df.sample(n=100000, random_state=42)

#  Step 2: Collapse high-cardinality categorical variables
def collapse_categories(series, top_n=4):
    top_cats = series.value_counts().nlargest(top_n).index
    return series.where(series.isin(top_cats), other='Other'
```

```
for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','il
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

#  Step 3: Target and predictors
y = df_sample['dTravtime'].astype('category')
predictors = [col for col in df_sample.columns if col not ir

#  Step 4: One-hot encode
X = pd.get_dummies(df_sample[predictors], drop_first=True)
y_codes = y.cat.codes

#  Step 5: Balance classes (undersample majority)
counts = Counter(y_codes)
min_count = min(counts.values())
balanced_idx = np.hstack([
    np.random.choice(np.where(y_codes==cls)[0], min_count, r
    for cls in counts.keys()
])
X_balanced = X.iloc[balanced_idx]
y_balanced = y_codes.iloc[balanced_idx]

print("Balanced class counts:", Counter(y_balanced))

#  Step 6: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42,
)

#  Step 7: Grid search for pruning
param_grid = {
    "max_depth": [5, 10, 15],
    "min_samples_leaf": [5, 10, 20],
    "min_samples_split": [10, 20]
}

grid = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid,
    cv=3,
    scoring="f1_macro"
)
grid.fit(X_train, y_train)
```

```
best_tree = grid.best_estimator_
print("Best parameters:", grid.best_params_)
print("Best macro F1 (CV):", grid.best_score_)

#  Step 8: Evaluate best tree
y_pred = best_tree.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Decision Tree test accuracy:", acc)
print("Classification report:")
print(classification_report(y_test, y_pred))

#  Step 9: ROC curves (OvR)
y_test_bin = label_binarize(y_test, classes=np.unique(y_test
y_score = best_tree.predict_proba(X_test)

fpr, tpr, roc_auc = {}, {}, {}
plt.figure(figsize=(8,6))
for i in range(y_test_bin.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_a

plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC (OvR) for Tuned Decision Tree")
plt.legend()
plt.show()
```

```
Balanced class counts: Counter({5: 2692, 0: 2692, 2: 2692, 4:
2692, 3: 2692, 1: 2692, 6: 2692})
Best parameters: {'max_depth': 10, 'min_samples_leaf': 20, 'm
in_samples_split': 10}
Best macro F1 (CV): 0.3582501478876265
Decision Tree test accuracy: 0.36455293181215176
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       538
           1       0.33      0.37      0.35       538
           2       0.20      0.19      0.20       538
           3       0.21      0.27      0.24       539
           4       0.18      0.16      0.17       539
           5       0.22      0.17      0.19       538
           6       0.40      0.39      0.39       539

    accuracy                           0.36      3769
   macro avg       0.36      0.36      0.36      3769
weighted avg       0.36      0.36      0.36      3769
```

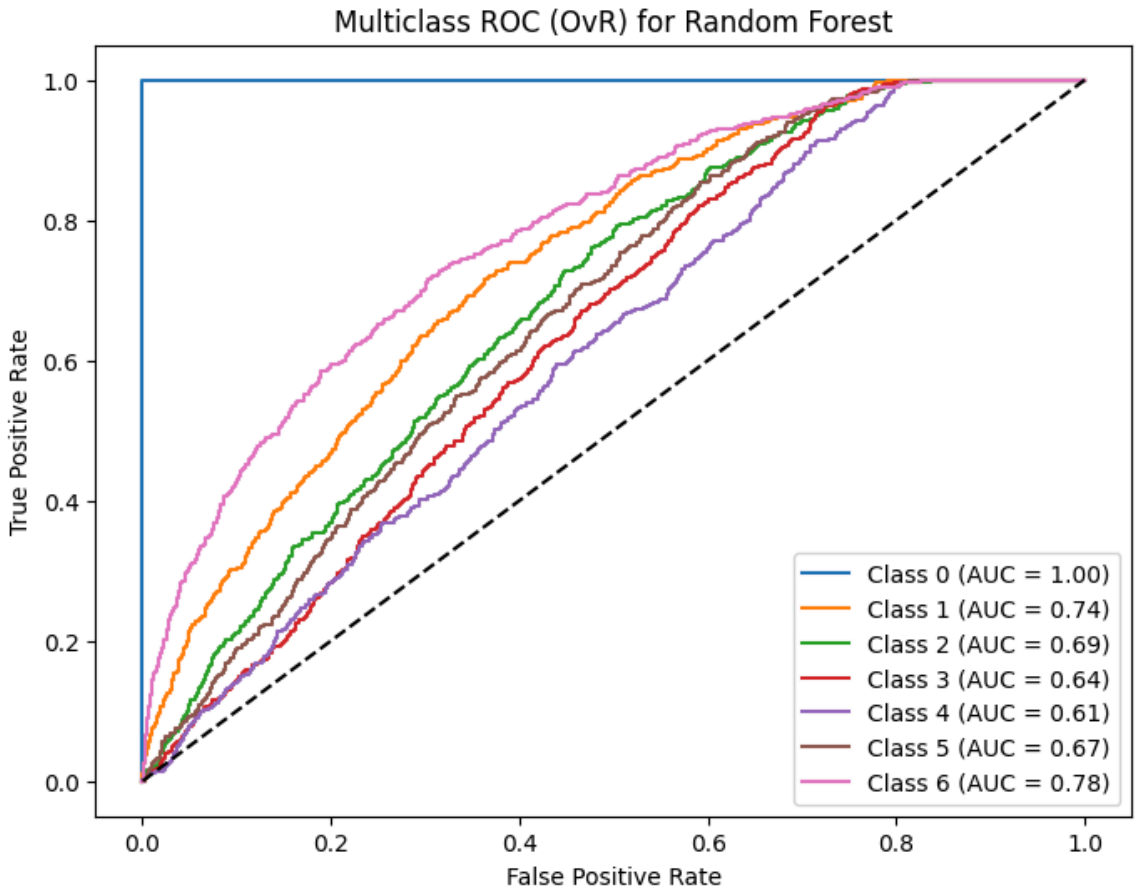


In [ ]:
```
# Random Forest
# Random Forest with standardized preprocessing
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, classi
from sklearn.preprocessing import StandardScaler, OneHotEncc
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from collections import Counter

# Step 1: Sample 100,000 rows (RF can handle larger sample s
df_sample = df.sample(n=100000, random_state=42)

# Step 2: Collapse high-cardinality categorical variables
def collapse_categories(series, top_n=4):
    top_cats = series.value_counts().nlargest(top_n).index
    return series.where(series.isin(top_cats), other='Other'

for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','iL
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

# Cast other categorical predictors to string
categorical_vars_master = [
    "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstry2",
    "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1",
    "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
    "iDisabl2", "iMobillim", "iVietnam", "iWWII"
]
for col in categorical_vars_master:
    if col in df_sample.columns:
        df_sample[col] = df_sample[col].astype(str)

# Step 3: Target and predictors
y = df_sample['dTravtime'].astype('category')
predictors = [col for col in df_sample.columns if col not ir
```

```
X = df_sample[predictors]
y_codes = y.cat.codes

# Step 4: Balance classes (undersample majority)
counts = Counter(y_codes)
min_count = min(counts.values())
balanced_idx = np.hstack([
    np.random.choice(np.where(y_codes == cls)[0], min_count,
    for cls in counts.keys()
])
X_balanced = X.iloc[balanced_idx]
y_balanced = y_codes.iloc[balanced_idx]

print("Balanced class counts:", Counter(y_balanced))

# Step 5: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42,
)

# Step 6: Build preprocessing + RF pipeline
def make_pipeline_rf(predictor_list, n_estimators=200, max_d
    categorical_vars = [col for col in predictor_list if col
    numeric_vars = [col for col in predictor_list if col not

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), numeric_vars),
            ("cat", OneHotEncoder(handle_unknown="ignore"),
        ]
    )

    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("model", RandomForestClassifier(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=42,
            n_jobs=-1
        ))
    ])
    return pipeline
```

```
# Step 7: Fit and evaluate
rf_pipeline = make_pipeline_rf(list(X_train.columns))
rf_pipeline.fit(X_train, y_train)
y_pred = rf_pipeline.predict(X_test)

acc = accuracy_score(y_test, y_pred)
macro_f1 = f1_score(y_test, y_pred, average="macro")
print("\nRandom Forest Results")
print("Test accuracy:", acc)
print("Macro F1:", macro_f1)
print("Classification report:")
print(classification_report(y_test, y_pred))

# Step 8: ROC curves
y_test_bin = label_binarize(y_test, classes=np.unique(y_test
y_score = rf_pipeline.predict_proba(X_test)

fpr, tpr, roc_auc = {}, {}, {}
plt.figure(figsize=(8,6))
for i in range(y_test_bin.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_a

plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC (OvR) for Random Forest")
plt.legend()
plt.show()

# Step 9: Feature importance
model = rf_pipeline.named_steps["model"]
importances = model.feature_importances_
feature_names = rf_pipeline.named_steps["preprocess"].get_fe

feat_imp = pd.DataFrame({"feature": feature_names, "importar
feat_imp = feat_imp.sort_values("importance", ascending=Fals
print("\nTop 20 Feature Importances:")
print(feat_imp)
```

```
Balanced class counts: Counter({5: 2692, 0: 2692, 2: 2692, 4:
2692, 3: 2692, 1: 2692, 6: 2692})

Random Forest Results
Test accuracy: 0.3730432475457681
Macro F1: 0.3657846653920608
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       538
           1       0.31      0.36      0.33       538
           2       0.24      0.19      0.21       538
           3       0.19      0.17      0.18       539
           4       0.20      0.23      0.21       539
           5       0.23      0.17      0.19       538
           6       0.38      0.50      0.43       539

    accuracy                           0.37      3769
   macro avg       0.36      0.37      0.37      3769
weighted avg       0.36      0.37      0.37      3769
```



```
Top 20 Feature Importances:
                 feature  importance
3          num__dDepart    0.099622
21          num__iMeans    0.077229
7           num__dHours    0.054176
85   cat__dIndustry_0.0    0.039829
2           num__iClass    0.036025
122   cat__iWorklwk_1.0    0.035366
32          num__iRiders    0.032131
28       num__dRearning    0.031898
6           num__dHour89    0.028821
44          num__dWeek89    0.028194
46        num__iYearsch    0.024217
0             num__dAge    0.019907
18        num__iLooking    0.017821
9         num__dIncome1    0.017693
36           num__iRPOB    0.015334
26          num__dPwgt1    0.015216
29         num__iRelat1    0.015089
35       num__dRpincome    0.014941
47         num__iYearwrk    0.014550
69      cat__iDisabl2_2    0.013041
```

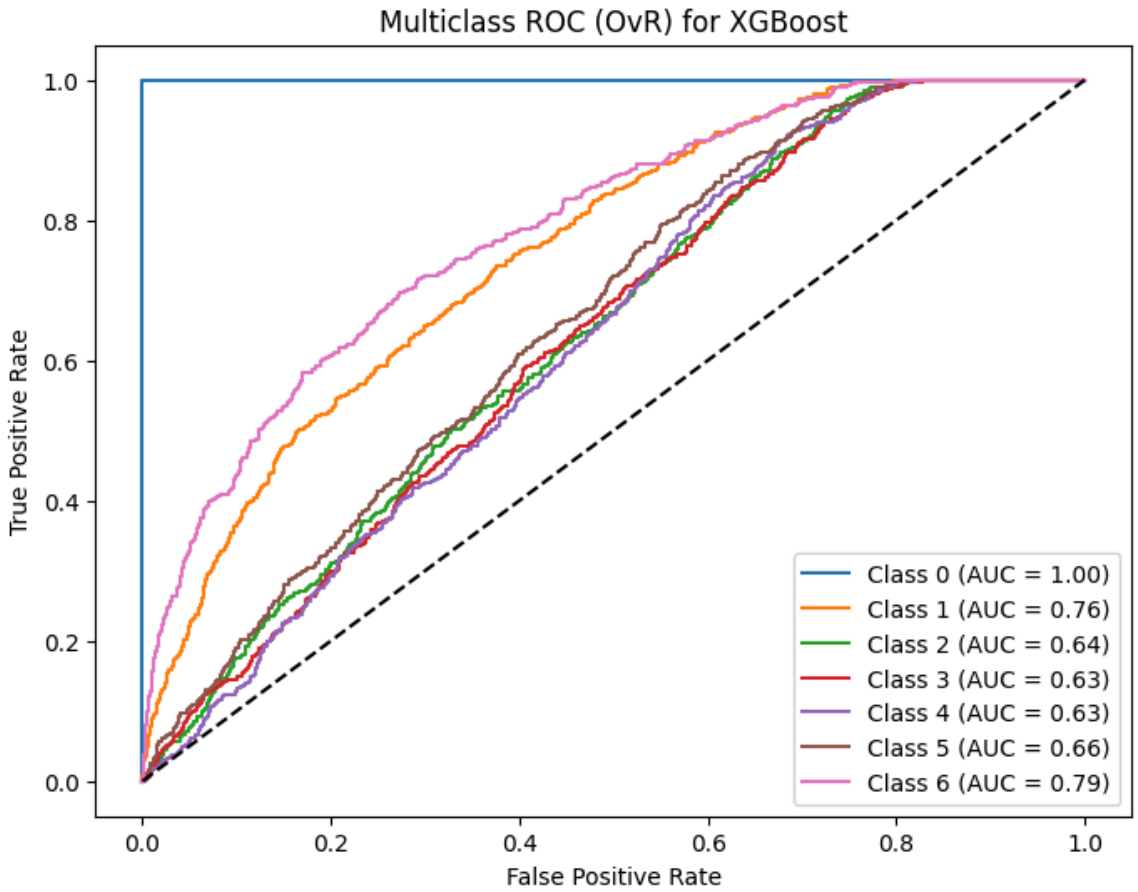
In [ ]:
```python
# Gradient Boosting
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, classi
from sklearn.preprocessing import StandardScaler, OneHotEncc
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
from collections import Counter

# Step 1: Sample 100,000 rows (XGB can handle larger sample
df_sample = df.sample(n=100000, random_state=42)

# Step 2: Collapse high-cardinality categorical variables
def collapse_categories(series, top_n=4):
```

```
    top_cats = series.value_counts().nlargest(top_n).index
    return series.where(series.isin(top_cats), other='Other'

for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','iL
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

# Cast other categorical predictors to string
categorical_vars_master = [
    "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstry2",
    "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1",
    "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
    "iDisabl2", "iMobillim", "iVietnam", "iWWII"
]
for col in categorical_vars_master:
    if col in df_sample.columns:
        df_sample[col] = df_sample[col].astype(str)

# Step 3: Target and predictors
y = df_sample['dTravtime'].astype('category')
predictors = [col for col in df_sample.columns if col not ir
X = df_sample[predictors]
y_codes = y.cat.codes

# Step 4: Balance classes (undersample majority)
counts = Counter(y_codes)
min_count = min(counts.values())
balanced_idx = np.hstack([
    np.random.choice(np.where(y_codes == cls)[0], min_count,
    for cls in counts.keys()
])
X_balanced = X.iloc[balanced_idx]
y_balanced = y_codes.iloc[balanced_idx]

print("Balanced class counts:", Counter(y_balanced))

# Step 5: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42,
)

# Step 6: Build preprocessing + XGB pipeline
def make_pipeline_xgb(predictor_list, n_estimators=300, lear
```

```
    categorical_vars = [col for col in predictor_list if col
    numeric_vars = [col for col in predictor_list if col not

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), numeric_vars),
            ("cat", OneHotEncoder(handle_unknown="ignore"),
        ]
    )

    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("model", XGBClassifier(
            n_estimators=n_estimators,
            learning_rate=learning_rate,
            max_depth=max_depth,
            subsample=0.8,
            colsample_bytree=0.8,
            random_state=42,
            use_label_encoder=False,
            eval_metric="mlogloss"
        ))
    ])
    return pipeline

# Step 7: Fit and evaluate
xgb_pipeline = make_pipeline_xgb(list(X_train.columns))
xgb_pipeline.fit(X_train, y_train)
y_pred = xgb_pipeline.predict(X_test)

acc = accuracy_score(y_test, y_pred)
macro_f1 = f1_score(y_test, y_pred, average="macro")
print("\nXGBoost Results")
print("Test accuracy:", acc)
print("Macro F1:", macro_f1)
print("Classification report:")
print(classification_report(y_test, y_pred))

# Step 8: ROC curves
y_test_bin = label_binarize(y_test, classes=np.unique(y_test
y_score = xgb_pipeline.predict_proba(X_test)

fpr, tpr, roc_auc = {}, {}, {}
```

```
plt.figure(figsize=(8,6))
for i in range(y_test_bin.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_a

plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC (OvR) for XGBoost")
plt.legend()
plt.show()

# Step 9: Feature importance
model = xgb_pipeline.named_steps["model"]
importances = model.feature_importances_
feature_names = xgb_pipeline.named_steps["preprocess"].get_f

feat_imp = pd.DataFrame({"feature": feature_names, "importar
feat_imp = feat_imp.sort_values("importance", ascending=Fals
print("\nTop 20 Feature Importances:")
print(feat_imp)
```

```
Balanced class counts: Counter({5: 2692, 0: 2692, 2: 2692, 4:
2692, 3: 2692, 1: 2692, 6: 2692})

XGBoost Results
Test accuracy: 0.3767577606792253
Macro F1: 0.37337428650665844
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       538
           1       0.36      0.41      0.38       538
           2       0.21      0.18      0.19       538
           3       0.20      0.20      0.20       539
           4       0.20      0.19      0.19       539
           5       0.23      0.22      0.23       538
           6       0.40      0.44      0.42       539

    accuracy                           0.38      3769
   macro avg       0.37      0.38      0.37      3769
weighted avg       0.37      0.38      0.37      3769
```



```
Top 20 Feature Importances:
                 feature  importance
85   cat__dIndustry_0.0    0.089014
3          num__dDepart    0.059748
7           num__dHours    0.031220
59      cat__iCitizen_0    0.024493
21          num__iMeans    0.024387
32         num__iRiders    0.016737
106       cat__dPOB_0.0    0.014239
34       num__iRownchld    0.012572
11        num__dIncome3    0.009854
28       num__dRearning    0.009527
33         num__iRlabor    0.009399
37       num__iRrelchld    0.009120
9         num__dIncome1    0.008600
35       num__dRpincome    0.008211
115    cat__iSchool_2.0    0.008027
27       num__iRagechld    0.008010
40         num__iSept80    0.007999
111       cat__dPOB_5.0    0.007953
16        num__dIncome8    0.007888
96   cat__iMilitary_3.0    0.007833
```

In [ ]:

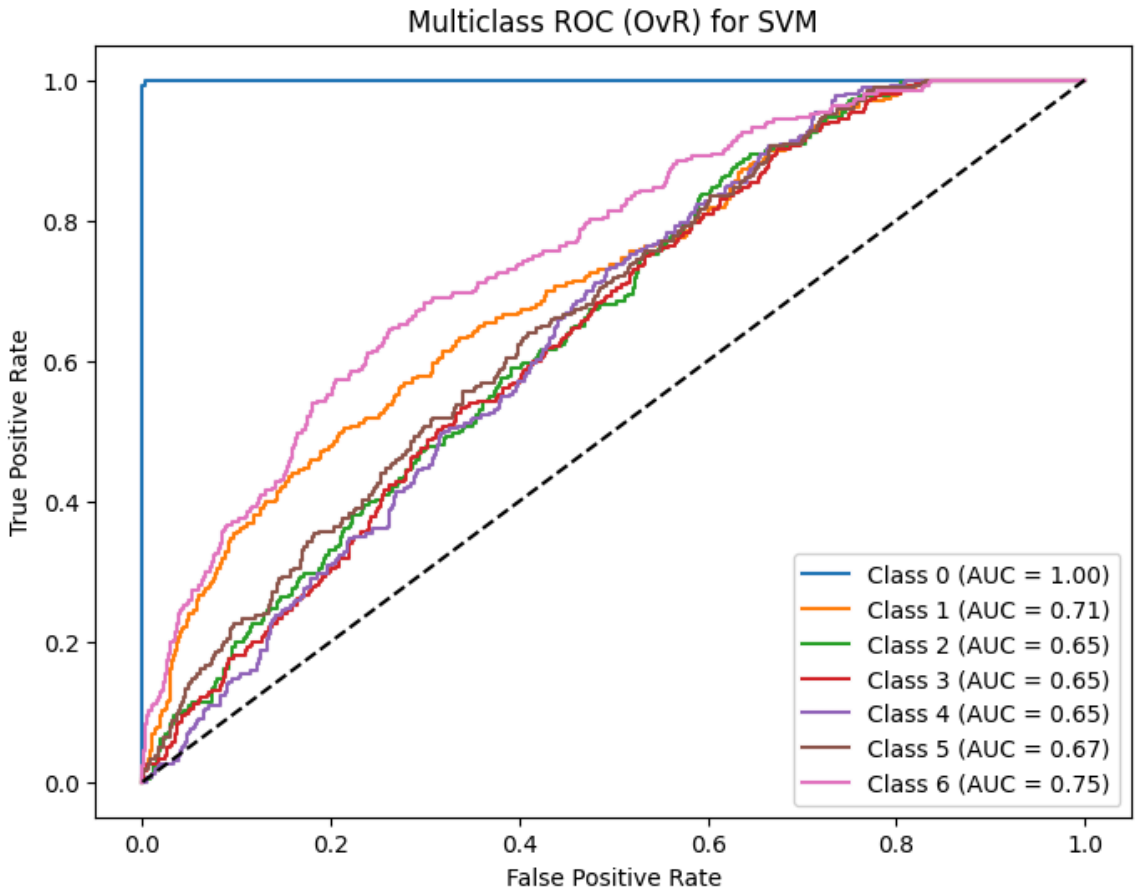
```
# SVM with PCA
# Support Vector Machine (SVM) with standardized preprocessi
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, classi
from sklearn.preprocessing import StandardScaler, OneHotEncc
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from collections import Counter

# Step 1: Sample 50,000 rows (SVM is computationally heavier
df_sample = df.sample(n=50000, random_state=42)
```

```
# Step 2: Collapse high-cardinality categorical variables
def collapse_categories(series, top_n=4):
    top_cats = series.value_counts().nlargest(top_n).index
    return series.where(series.isin(top_cats), other='Other'

for col in ['dIndustry','dOccup','dAncstry1','dAncstry2','iL
    if col in df_sample.columns:
        df_sample[col] = collapse_categories(df_sample[col],

# Cast other categorical predictors to string
categorical_vars_master = [
    "iSex", "dOccup", "dIndustry", "dAncstry1", "dAncstry2",
    "dHispanic", "dPOB", "iCitizen", "iEnglish", "iLang1",
    "iMilitary", "iSchool", "iWorklwk", "iDisabl1",
    "iDisabl2", "iMobillim", "iVietnam", "iWWII"
]
for col in categorical_vars_master:
    if col in df_sample.columns:
        df_sample[col] = df_sample[col].astype(str)

# Step 3: Target and predictors
y = df_sample['dTravtime'].astype('category')
predictors = [col for col in df_sample.columns if col not ir
X = df_sample[predictors]
y_codes = y.cat.codes

# Step 4: Balance classes (undersample majority)
counts = Counter(y_codes)
min_count = min(counts.values())
balanced_idx = np.hstack([
    np.random.choice(np.where(y_codes == cls)[0], min_count,
    for cls in counts.keys()
])
X_balanced = X.iloc[balanced_idx]
y_balanced = y_codes.iloc[balanced_idx]

print("Balanced class counts:", Counter(y_balanced))

# Step 5: Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42,
)
```

```
# Step 6: Build preprocessing + PCA + SVM pipeline
def make_pipeline_svm(predictor_list, C=1.0, gamma="scale"):
    categorical_vars = [col for col in predictor_list if col
    numeric_vars = [col for col in predictor_list if col not

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), numeric_vars),
            ("cat", OneHotEncoder(handle_unknown="ignore"),
        ]
    )

    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("pca", PCA(n_components=0.95, random_state=42)),  #
        ("model", SVC(kernel="rbf", C=C, gamma=gamma, probab
    ])
    return pipeline

# Step 7: Fit and evaluate
svm_pipeline = make_pipeline_svm(list(X_train.columns))
svm_pipeline.fit(X_train, y_train)
y_pred = svm_pipeline.predict(X_test)

acc = accuracy_score(y_test, y_pred)
macro_f1 = f1_score(y_test, y_pred, average="macro")
print("\nSVM Results")
print("Test accuracy:", acc)
print("Macro F1:", macro_f1)
print("Classification report:")
print(classification_report(y_test, y_pred))

# Step 8: ROC curves
y_test_bin = label_binarize(y_test, classes=np.unique(y_test
y_score = svm_pipeline.predict_proba(X_test)

fpr, tpr, roc_auc = {}, {}, {}
plt.figure(figsize=(8,6))
for i in range(y_test_bin.shape[1]):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f"Class {i} (AUC = {roc_a
```

```
plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC (OvR) for SVM")
plt.legend()
plt.show()
```

```
Balanced class counts: Counter({5: 1331, 0: 1331, 2: 1331, 4:
1331, 3: 1331, 1: 1331, 6: 1331})

SVM Results
Test accuracy: 0.3717811158798283
Macro F1: 0.36535901736055854
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       266
           1       0.34      0.37      0.35       266
           2       0.20      0.21      0.20       266
           3       0.22      0.17      0.19       267
           4       0.22      0.23      0.23       266
           5       0.23      0.16      0.19       266
           6       0.35      0.47      0.40       267

    accuracy                           0.37      1864
   macro avg       0.36      0.37      0.37      1864
weighted avg       0.36      0.37      0.37      1864
```



In [ ]:
```
# Compare all models
from sklearn.metrics import accuracy_score, f1_score

def compare_models(X_train, X_test, y_train, y_test):
    results = []

    #  Logistic Regression with backward stepwise selection
    acc_history = backward_stepwise(X_train, y_train, X_test
    best_step = max(acc_history, key=lambda x: x[1])
    best_vars = best_step[2]
    best_pipeline = make_pipeline(best_vars)
    best_pipeline.fit(X_train[best_vars], y_train)
    y_pred = best_pipeline.predict(X_test[best_vars])
    results.append({
        "Model": "Multinomial Logistic Regression (Stepwise)
        "Accuracy": accuracy_score(y_test, y_pred),
        "Macro F1": f1_score(y_test, y_pred, average="macro"
    })

    #  KNN
    knn_pipeline = make_pipeline_knn(list(X_train.columns),
```

```
    knn_pipeline.fit(X_train, y_train)
    y_pred = knn_pipeline.predict(X_test)
    results.append({
        "Model": "KNN (k=5)",
        "Accuracy": accuracy_score(y_test, y_pred),
        "Macro F1": f1_score(y_test, y_pred, average="macro"
    })

    #  Random Forest
    rf_pipeline = make_pipeline_rf(list(X_train.columns))
    rf_pipeline.fit(X_train, y_train)
    y_pred = rf_pipeline.predict(X_test)
    results.append({
        "Model": "Random Forest",
        "Accuracy": accuracy_score(y_test, y_pred),
        "Macro F1": f1_score(y_test, y_pred, average="macro"
    })

    #  XGBoost
    xgb_pipeline = make_pipeline_xgb(list(X_train.columns))
    xgb_pipeline.fit(X_train, y_train)
    y_pred = xgb_pipeline.predict(X_test)
    results.append({
        "Model": "XGBoost",
        "Accuracy": accuracy_score(y_test, y_pred),
        "Macro F1": f1_score(y_test, y_pred, average="macro"
    })

    #  SVM
    svm_pipeline = make_pipeline_svm(list(X_train.columns))
    svm_pipeline.fit(X_train, y_train)
    y_pred = svm_pipeline.predict(X_test)
    results.append({
        "Model": "SVM (RBF)",
        "Accuracy": accuracy_score(y_test, y_pred),
        "Macro F1": f1_score(y_test, y_pred, average="macro"
    })

    # Convert to DataFrame for readability
    results_df = pd.DataFrame(results)
    return results_df
```

```
results_df = compare_models(X_train, X_test, y_train, y_test
print(results_df)
```

```
Dropped dAge, accuracy 0.3540772532188841
Dropped dAncstry1, accuracy 0.361587982832618
Dropped iAvail, accuracy 0.3621244635193133
Dropped iCitizen, accuracy 0.3621244635193133
Dropped iDisabl1, accuracy 0.3621244635193133
Dropped iFertil, accuracy 0.36319742489270385
Dropped iClass, accuracy 0.3626609442060086
Dropped iDisabl2, accuracy 0.36373390557939916
Dropped iFeb55, accuracy 0.36373390557939916
Dropped dHour89, accuracy 0.3642703862660944
Dropped dHispanic, accuracy 0.3642703862660944
Dropped dIncome4, accuracy 0.3642703862660944
Dropped dIncome2, accuracy 0.36373390557939916
Dropped dAncstry2, accuracy 0.36319742489270385
Dropped iImmigr, accuracy 0.3626609442060086
Dropped dIncome1, accuracy 0.3642703862660944
Dropped iLang1, accuracy 0.36373390557939916
Dropped iEnglish, accuracy 0.3648068669527897
Dropped dIncome5, accuracy 0.3648068669527897
Dropped iKorean, accuracy 0.365343347639485
Dropped iLooking, accuracy 0.3648068669527897
Dropped dIncome8, accuracy 0.3648068669527897
Dropped iMay75880, accuracy 0.3648068669527897
Dropped iMilitary, accuracy 0.365343347639485
Dropped dIncome7, accuracy 0.3648068669527897
Dropped iMarital, accuracy 0.3642703862660944
Dropped iMobillim, accuracy 0.365343347639485
Dropped iRagechld, accuracy 0.3648068669527897
Dropped iMobility, accuracy 0.36587982832618027
Dropped iRelat1, accuracy 0.36587982832618027
Dropped dIncome6, accuracy 0.36587982832618027
Dropped iPerscare, accuracy 0.3669527896995708
Dropped iOthrserv, accuracy 0.3669527896995708
Dropped iRemplpar, accuracy 0.36641630901287553
Dropped iRiders, accuracy 0.3696351931330472
Dropped iRelat2, accuracy 0.3696351931330472
Dropped iRlabor, accuracy 0.3696351931330472
Dropped dRearning, accuracy 0.3717811158798283
Dropped iSchool, accuracy 0.37124463519313305
Dropped iSubfam1, accuracy 0.3707081545064378
                                          Model  Accuracy  Macr
o F1
0  Multinomial Logistic Regression (Stepwise)  0.371781  0.36
```

```
1939
1                    KNN (k=5)  0.315451  0.31
6421
2              Random Forest  0.368562  0.36
3224
3                    XGBoost  0.354077  0.35
1534
4                    SVM (RBF)  0.371781  0.36
5359
```