In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn import preprocessing
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.utils import resample
from sklearn.model_selection import GridSearchCV
```

EDA

In [ ]:
```
df = pd.read_csv('cs-training.csv', index_col=0)
df.head()
```

Out[ ]:

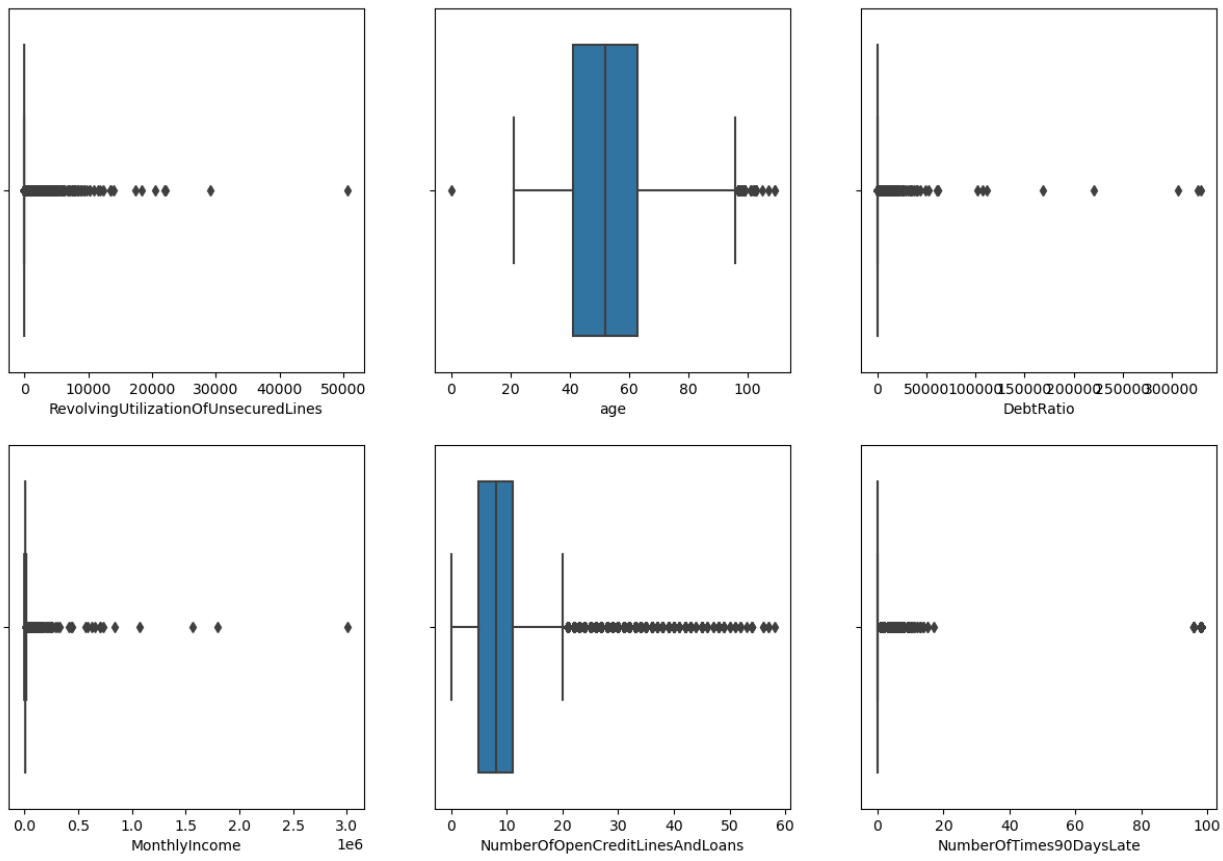
	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio
1	1	0.766127	45	2	0.802982
2	0	0.957151	40	0	0.121876
3	0	0.658180	38	1	0.085113
4	0	0.233810	30	0	0.036050
5	0	0.907239	49	1	0.024926

In [ ]:
```
df.describe()
```

Out[ ]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse
count	150000.000000	150000.000000	150000.000000	150000.000000
mean	0.066840	6.048438	52.295207	0.421033
std	0.249746	249.755371	14.771866	4.192781
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.029867	41.000000	0.000000
50%	0.000000	0.154181	52.000000	0.000000
75%	0.000000	0.559046	63.000000	0.000000
max	1.000000	50708.000000	109.000000	98.000000

In [ ]:
```
# make subplot for all variables to check for outliers
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
sns.boxplot(x=df.RevolvingUtilizationOfUnsecuredLines, ax=axes[0, 0])
sns.boxplot(x=df.age, ax=axes[0, 1])
sns.boxplot(x=df.DebtRatio, ax=axes[0, 2])
sns.boxplot(x=df.MonthlyIncome, ax=axes[1, 0])
sns.boxplot(x=df.NumberOfOpenCreditLinesAndLoans, ax=axes[1, 1])
sns.boxplot(x=df.NumberOfTimes90DaysLate, ax=axes[1, 2])
plt.show()
```

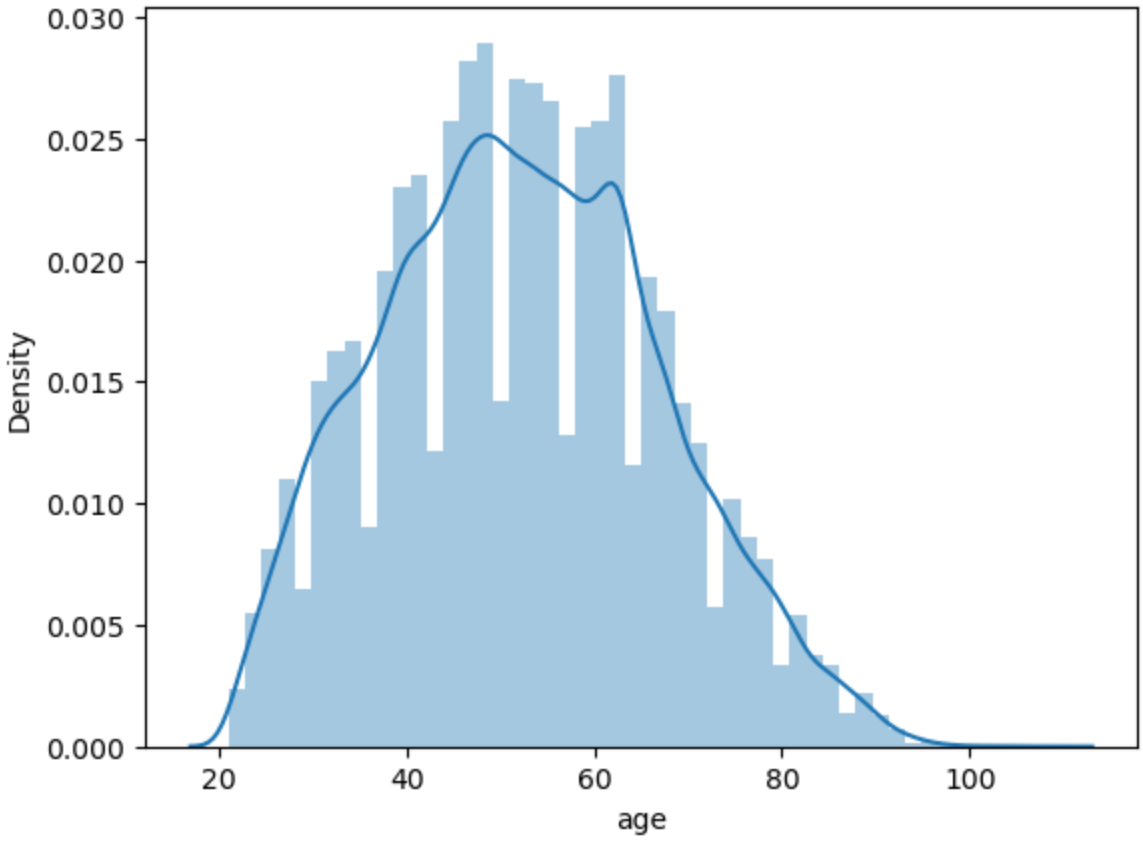


Age

In [ ]:
```
# replace when age is 0 with median

df.loc[df.age == 0, 'age'] = df.age.median()
```

In [ ]:
```
# plot age dist
sns.distplot(df['age'])
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\2153746949.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['age'])
```

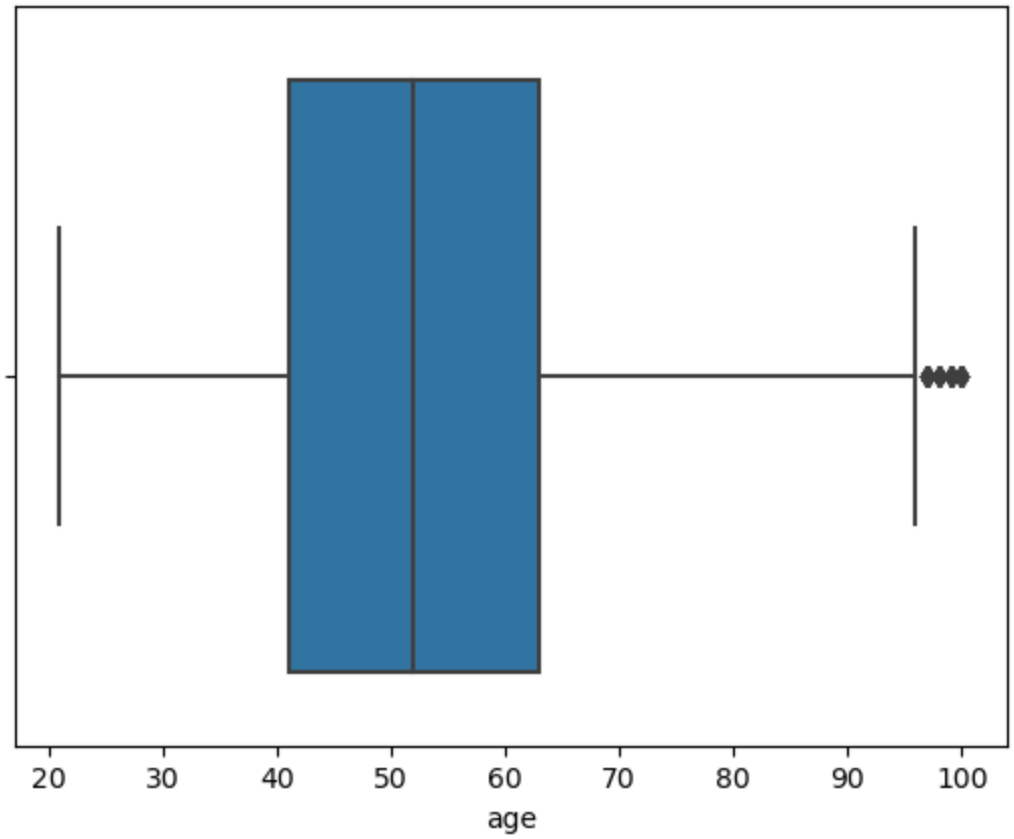


In [ ]:
```
# use T score to determine how likely it is for an age above 100 to be an outlier
from scipy.stats import ttest_ind
ttest_ind(df['age'], df['age'] > 100)
```

Out[ ]:
```
Ttest_indResult(statistic=1371.17338387618, pvalue=0.0)
```

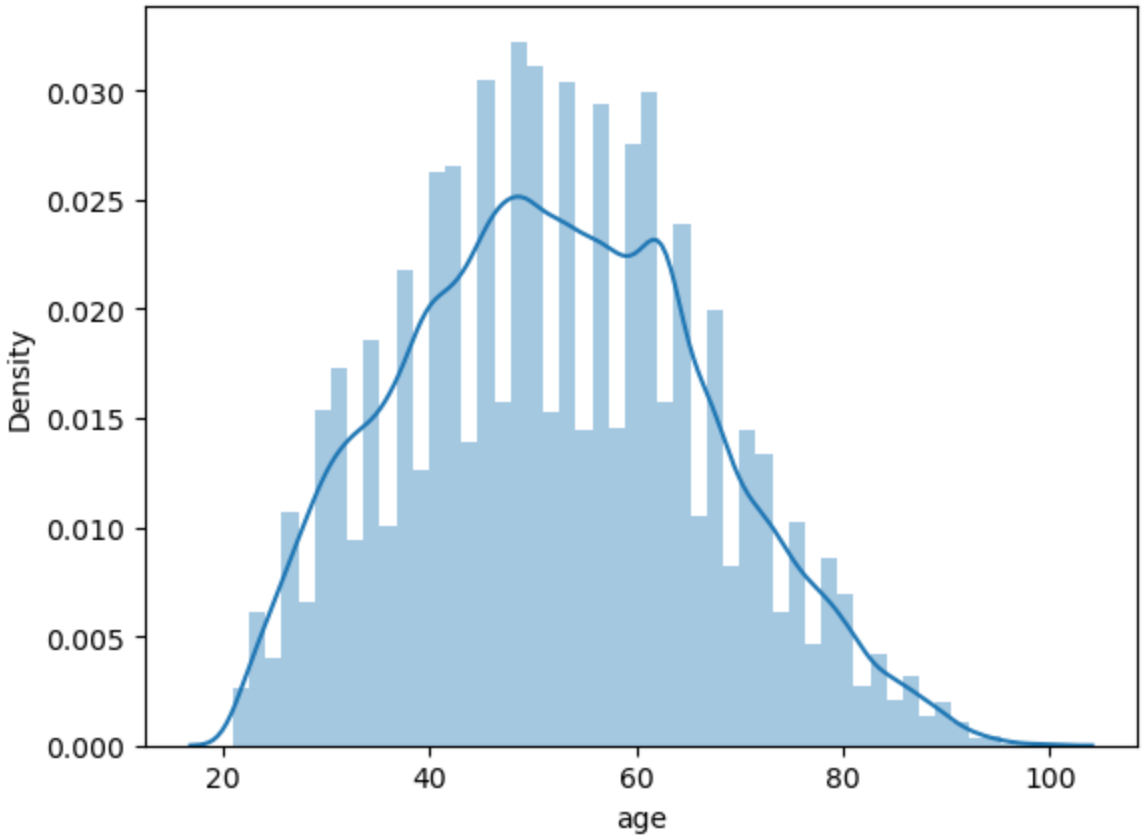
In [ ]:
```
df['age'] = df['age'].clip(upper=100)
```

In [ ]:
```
# age boxplot
sns.boxplot(x=df.age)
plt.show()
```



In [ ]:
```
sns.distplot(df['age'])
plt.show()
```

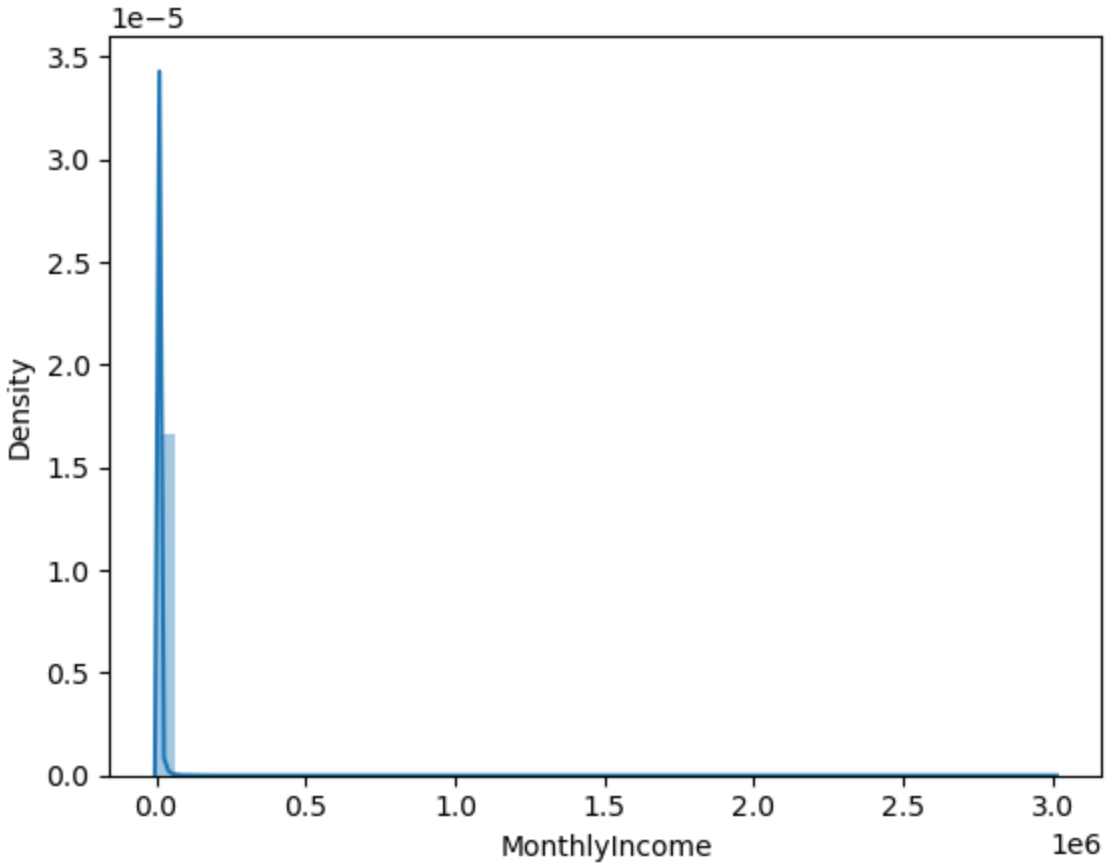
```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\2799180019.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['age'])
```



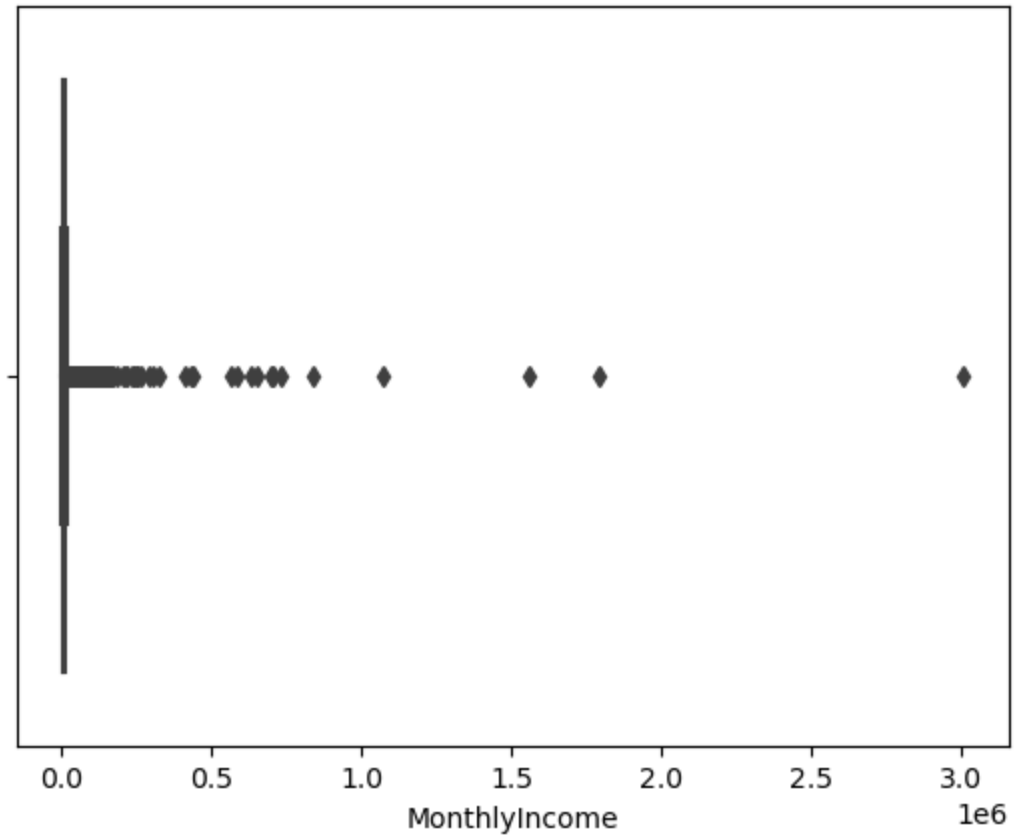
Monthly Income

In [ ]:
```
# plot dist of monthly income
sns.distplot(df['MonthlyIncome'])
plt.show()
```

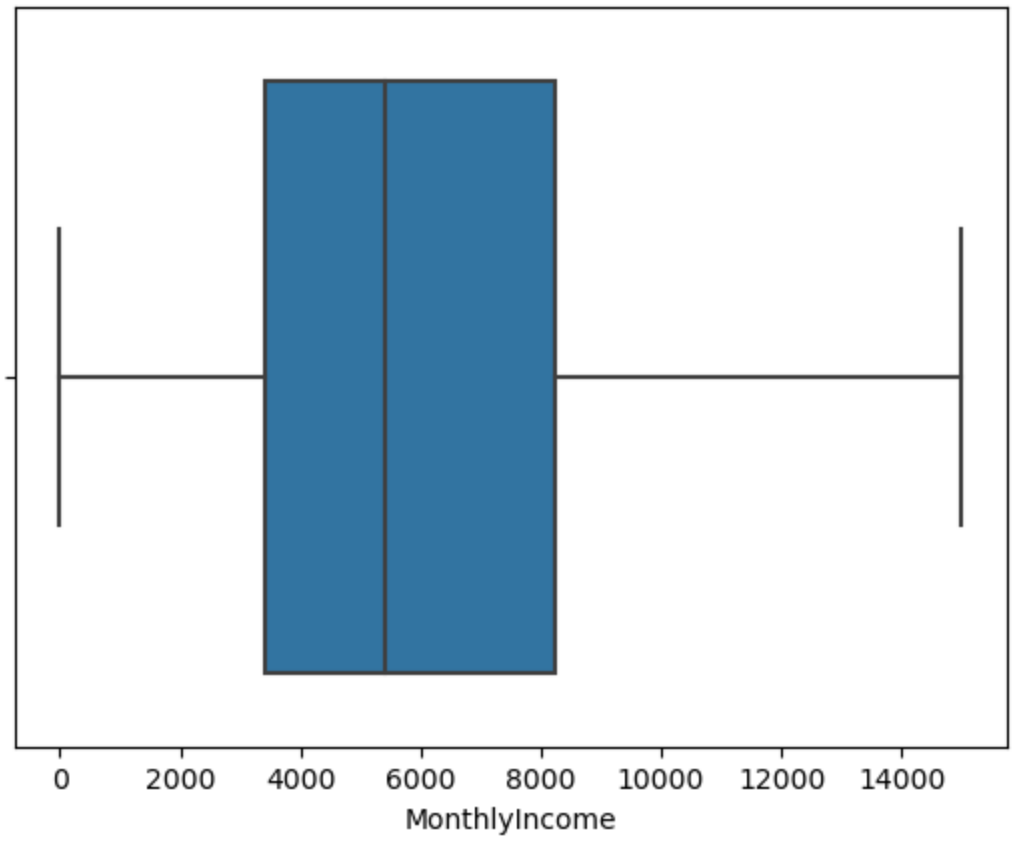
```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\3025500376.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['MonthlyIncome'])
```



In [ ]:
```
# plot the boxplot for monthly income
sns.boxplot(x=df.MonthlyIncome)
plt.show()
```

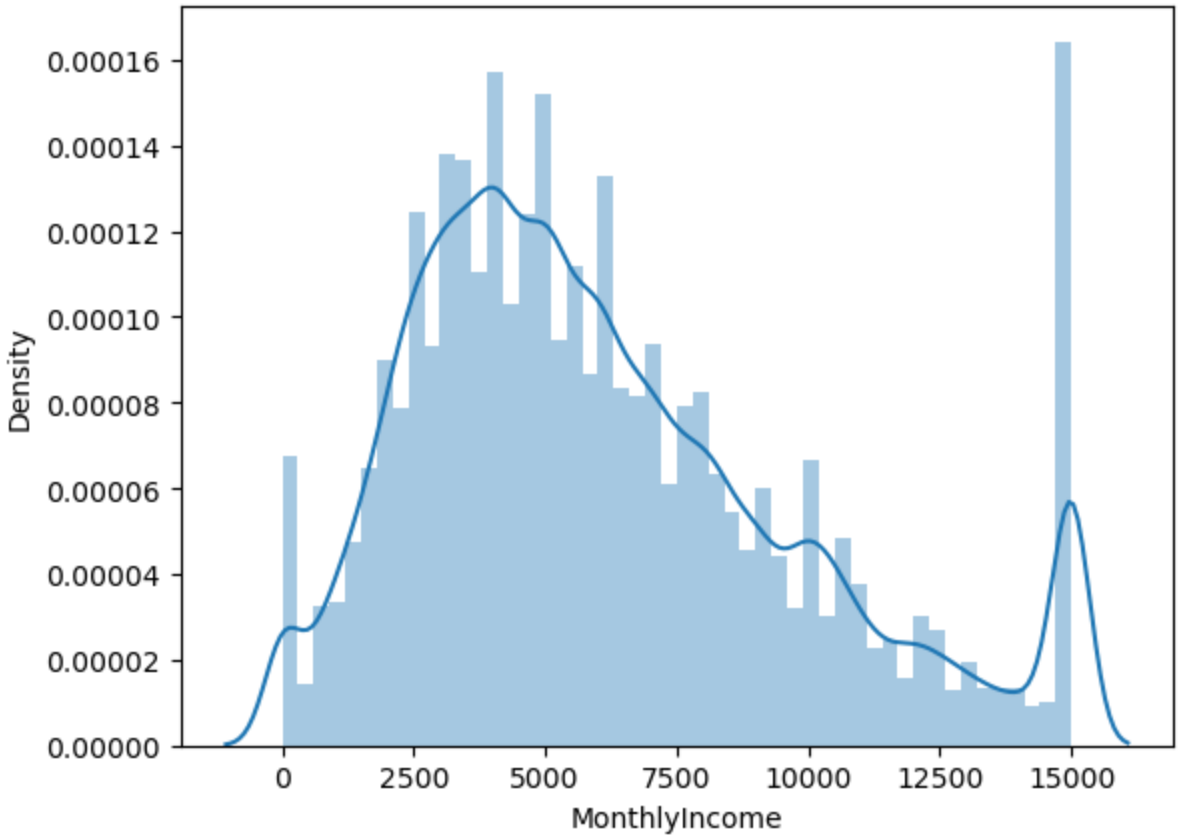


In [ ]:
```
# cap/ winsorize monthly income
df.loc[df.MonthlyIncome > 15000, 'MonthlyIncome'] = 15000
sns.boxplot(x=df.MonthlyIncome)
plt.show()
```



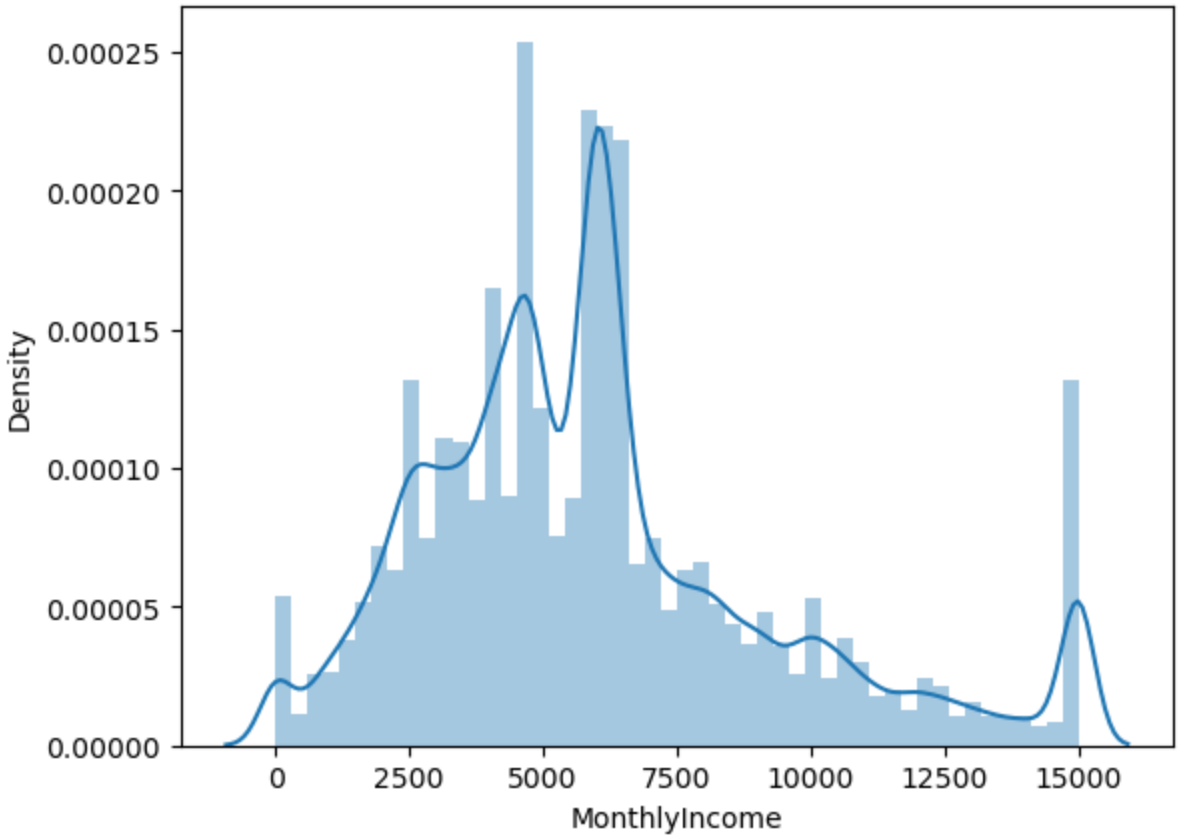
In [ ]:
```
sns.distplot(df['MonthlyIncome'])
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\3971867337.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['MonthlyIncome'])
```



In [ ]:
```
# Define age bins
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '80-89
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Calculate median income for each age group
median_incomes = df.groupby('age_group')['MonthlyIncome'].median()

# Impute missing values
for age_group in labels:
    median_income = median_incomes[age_group]
    df.loc[(df['age_group'] == age_group) & (df['MonthlyIncome'].isnull()), 'MonthlyIr

# Drop the age_group column if no longer needed
df.drop('age_group', axis=1, inplace=True)

sns.distplot(df['MonthlyIncome'])
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\1385070261.py:17: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['MonthlyIncome'])
```



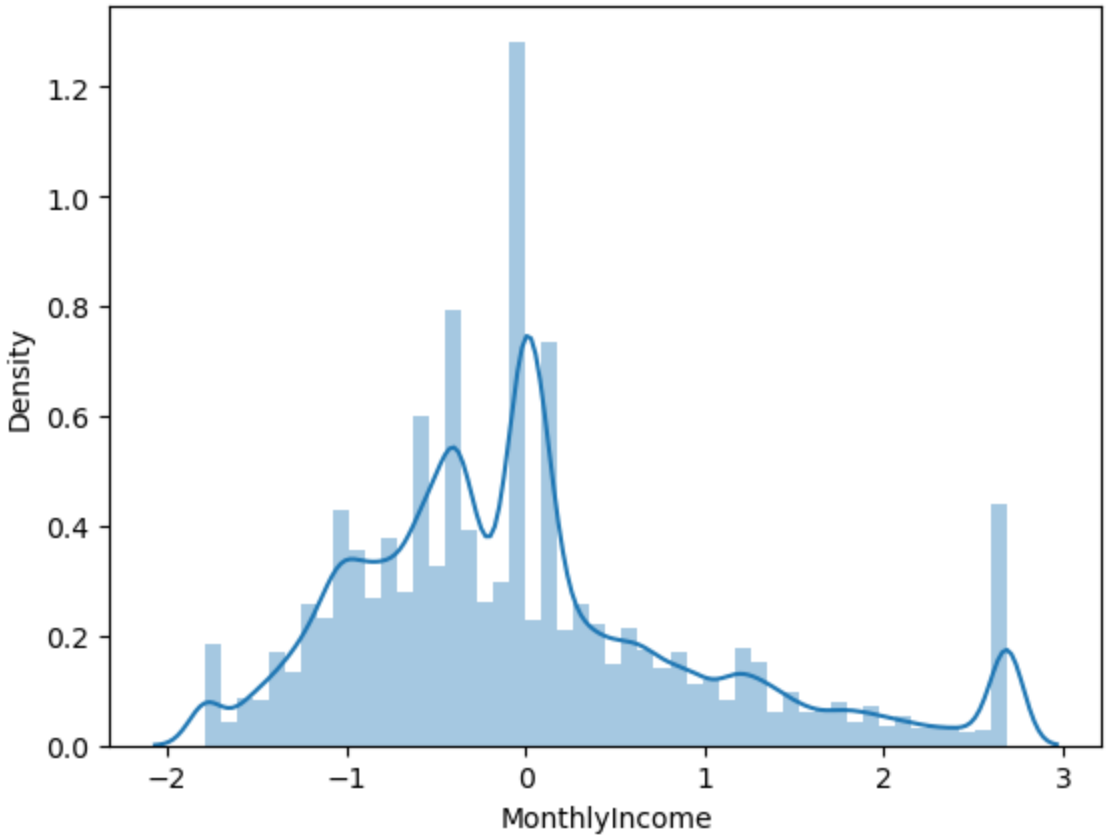
In [ ]:
```
#impute any remaining missing values in monthly income with median
df['MonthlyIncome'].fillna(df['MonthlyIncome'].median(), inplace=True)
```

In [ ]:
```

```

Out[ ]:
```
0
```

In [ ]:
```
# standardize monthly income using sklearn StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['MonthlyIncome'] = scaler.fit_transform(df[['MonthlyIncome']])
sns.distplot(df['MonthlyIncome'])
plt.show()
```

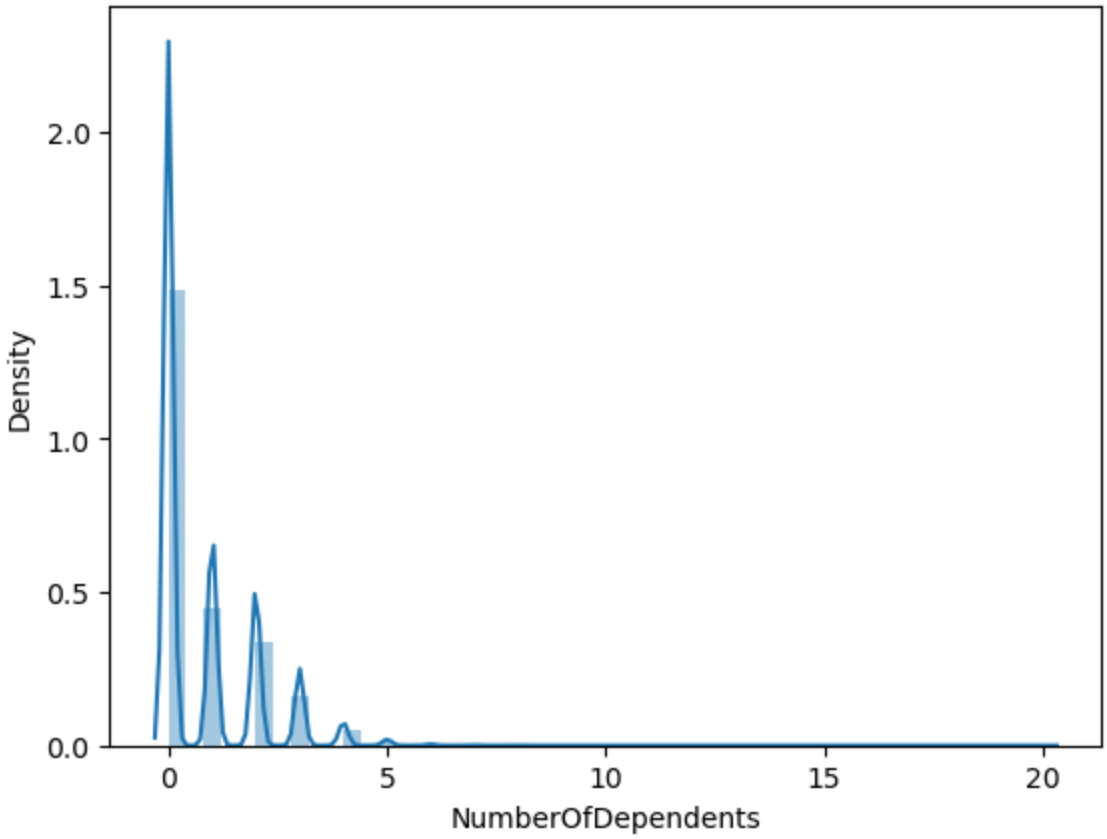
```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\3604040195.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['MonthlyIncome'])
```



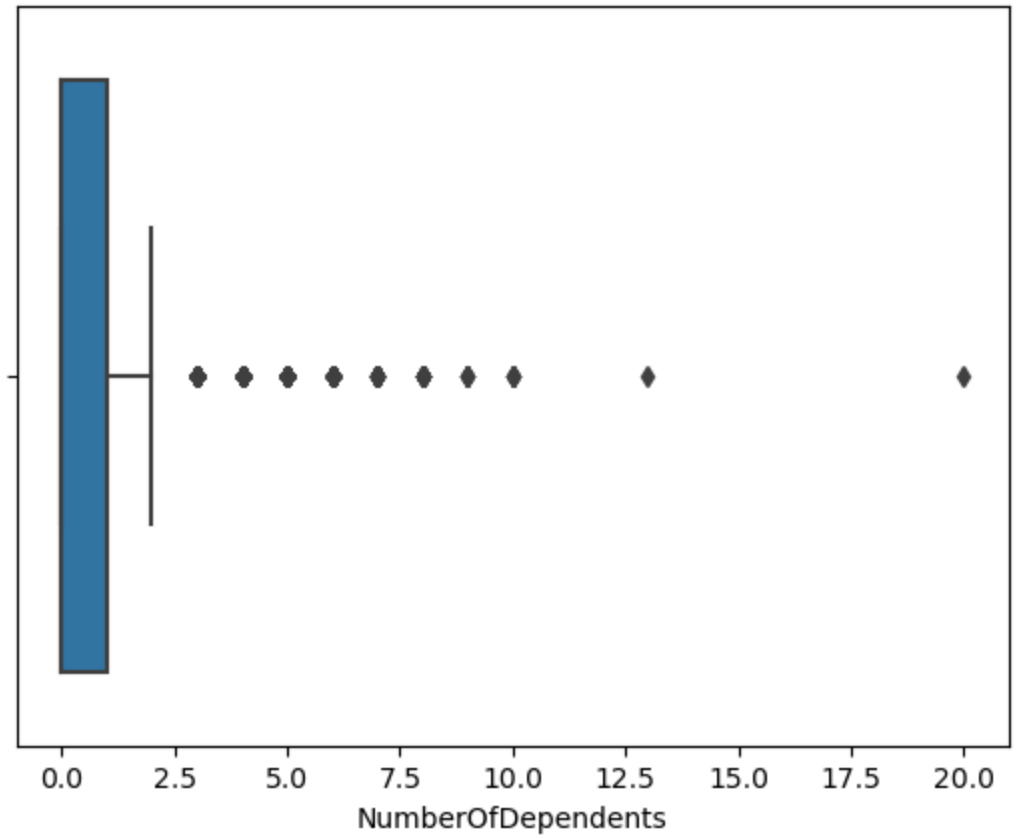
Dependents

In [ ]:
```
sns.distplot(df['NumberOfDependents'])
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\2263928766.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['NumberOfDependents'])
```



In [ ]:
```
# plot boxplot for number of dependents
sns.boxplot(x=df.NumberOfDependents)
plt.show()
```



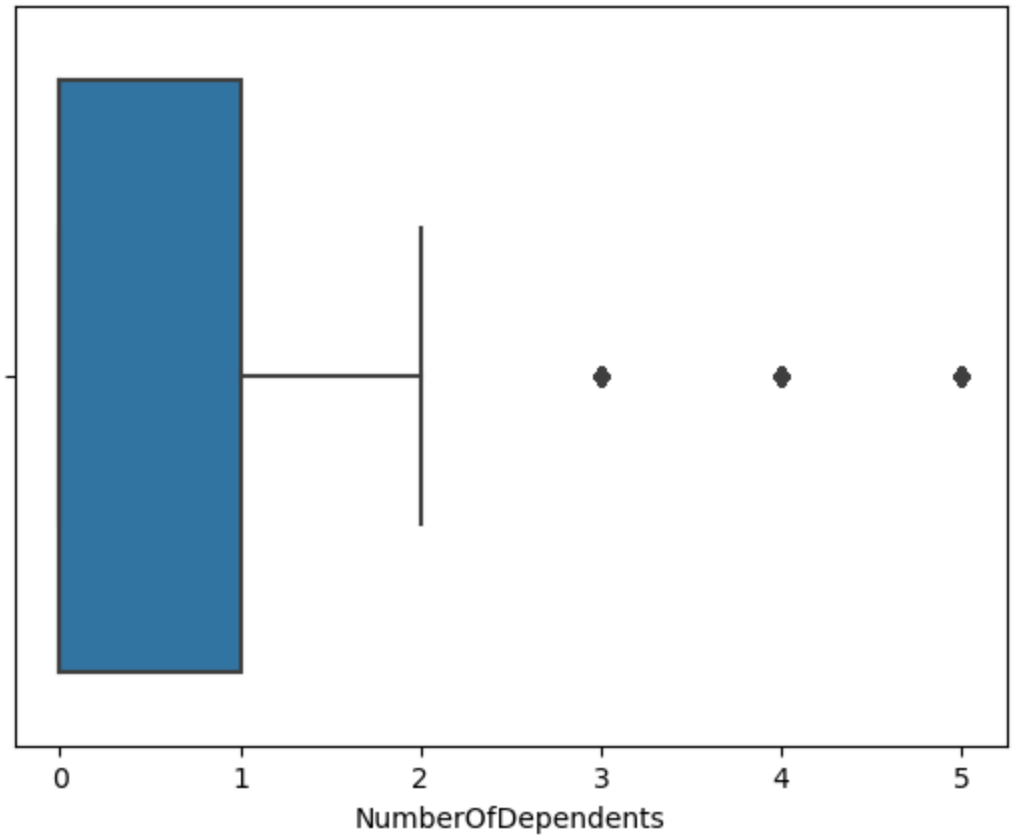
In [ ]:
```
# summary stats for number of dependents
df.NumberOfDependents.describe()
```

Out[ ]:
```
count    146076.000000
mean          0.757222
std           1.115086
min           0.000000
25%           0.000000
50%           0.000000
75%           1.000000
max          20.000000
Name: NumberOfDependents, dtype: float64
```

In [ ]:
```
# view counts for number of dependents
df.NumberOfDependents.value_counts()
```

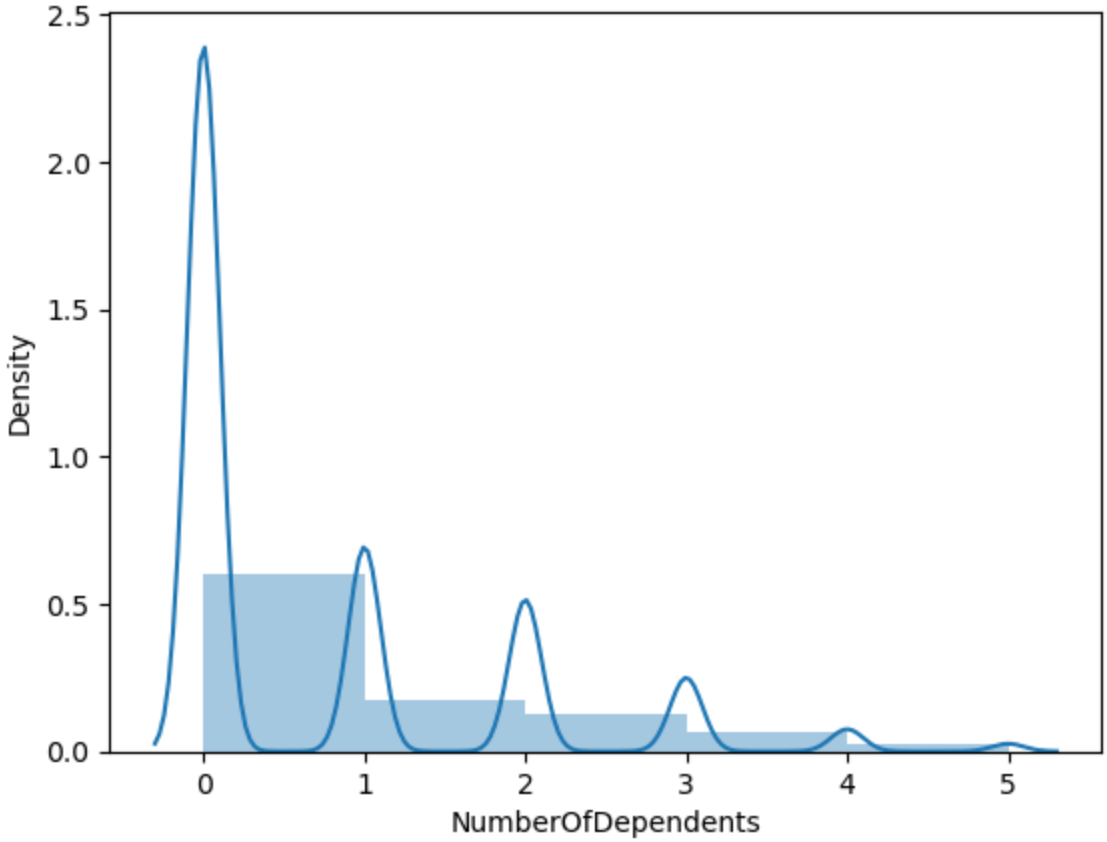
Out[ ]:
```
0.0     86902
1.0     26316
2.0     19522
3.0      9483
4.0      2862
5.0       746
6.0       158
7.0        51
8.0        24
10.0        5
9.0         5
20.0        1
13.0        1
Name: NumberOfDependents, dtype: int64
```

In [ ]:
```
# winsorize number of dependents
df.loc[df.NumberOfDependents > 5, 'NumberOfDependents'] = 5
sns.boxplot(x=df.NumberOfDependents)
plt.show()
```



In [ ]:
```
# impute using mediam number of dependents using sklearn SimpleImputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
df['NumberOfDependents'] = imputer.fit_transform(df[['NumberOfDependents']])
#Large bins for plot
bins = [0, 1, 2, 3, 4, 5]
sns.distplot(df['NumberOfDependents'], bins=bins)
plt.show()
```

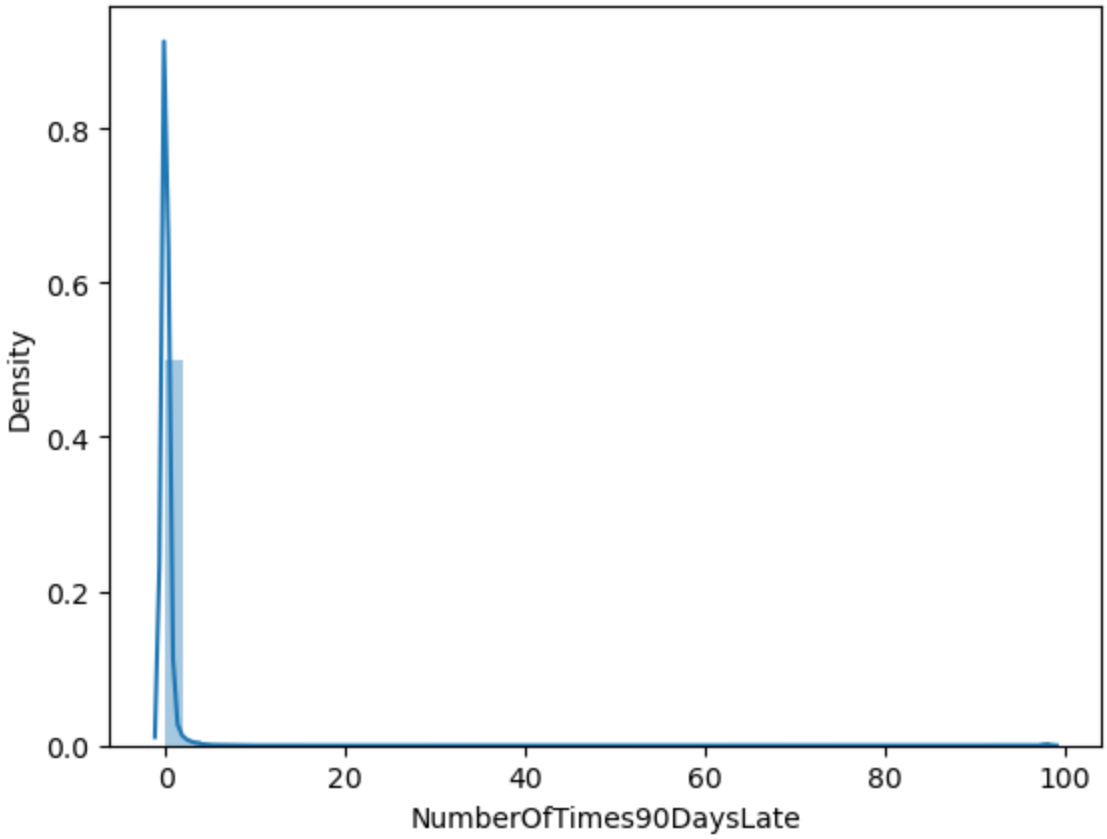
```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\1848479223.py:7: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['NumberOfDependents'], bins=bins)
```



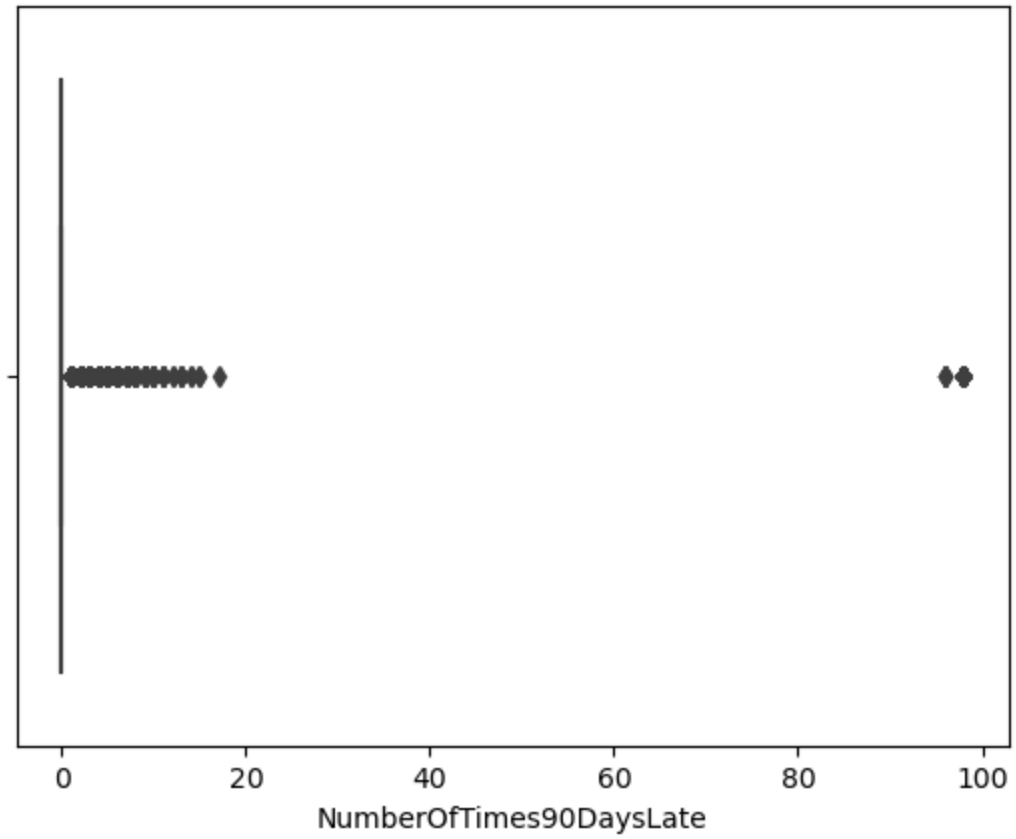
NumberOfTime90DaysLate

In [ ]:
```
# plot dist of debt ratio
sns.distplot(df['NumberOfTimes90DaysLate'])
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\2923087821.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['NumberOfTimes90DaysLate'])
```



In [ ]:
```
# plot boxplot of debt ratio
sns.boxplot(x=df.NumberOfTimes90DaysLate)
plt.show()
```



In [ ]:
```
# view outliers entires
df.loc[df.NumberOfTimes90DaysLate > 97]
```

Out[ ]:

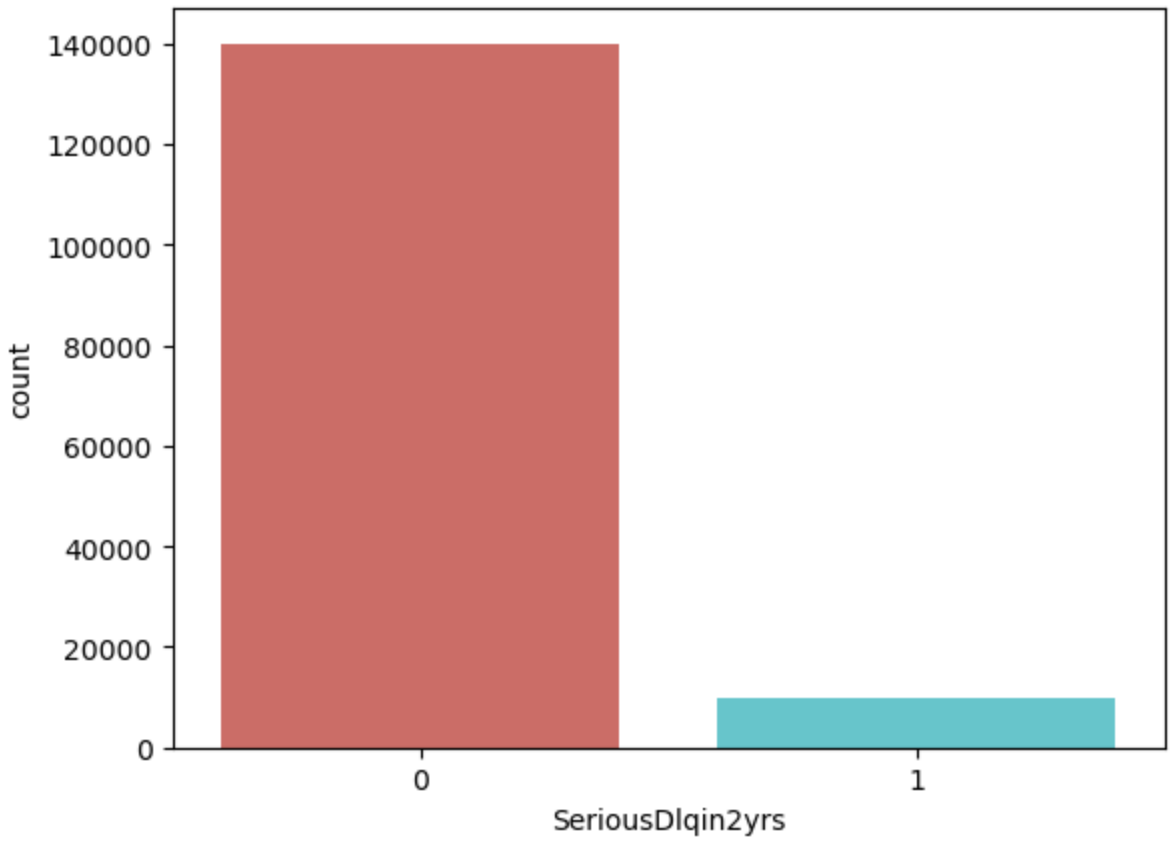
	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtR
1734	1	1.0	27	98	
2287	0	1.0	22	98	
3885	0	1.0	38	98	
4418	0	1.0	21	98	
4706	0	1.0	21	98	
...	...	...	...	...	
147775	1	1.0	68	98	2
149154	1	1.0	24	98	
149240	0	1.0	26	98	
149440	1	1.0	34	98	
149770	0	1.0	23	98	

264 rows × 11 columns

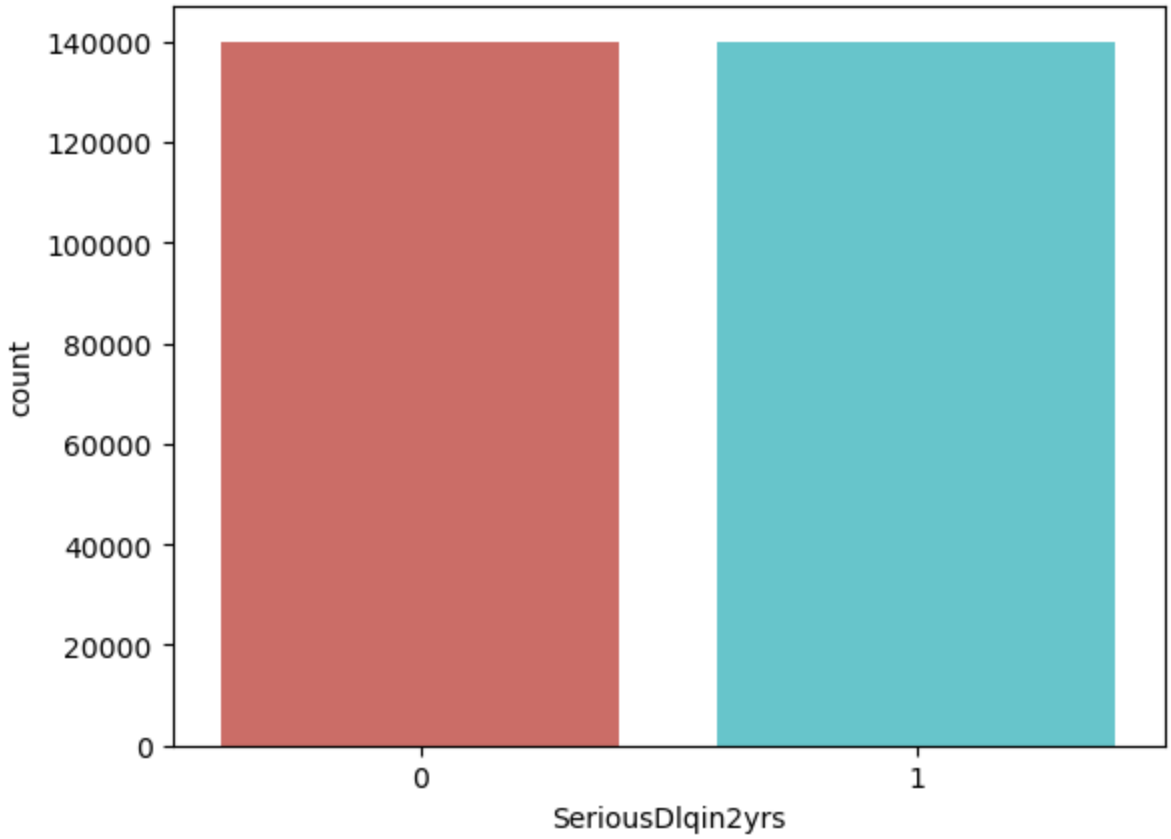
In [ ]:
```
df.SeriousDlqin2yrs.mean()
```

Out[ ]:
```
0.06684
```

In [ ]:
```
# plot the distribution of the target variable to see if there is a class imbalance
sns.countplot(x='SeriousDlqin2yrs', data=df, palette='hls')
plt.show()
```

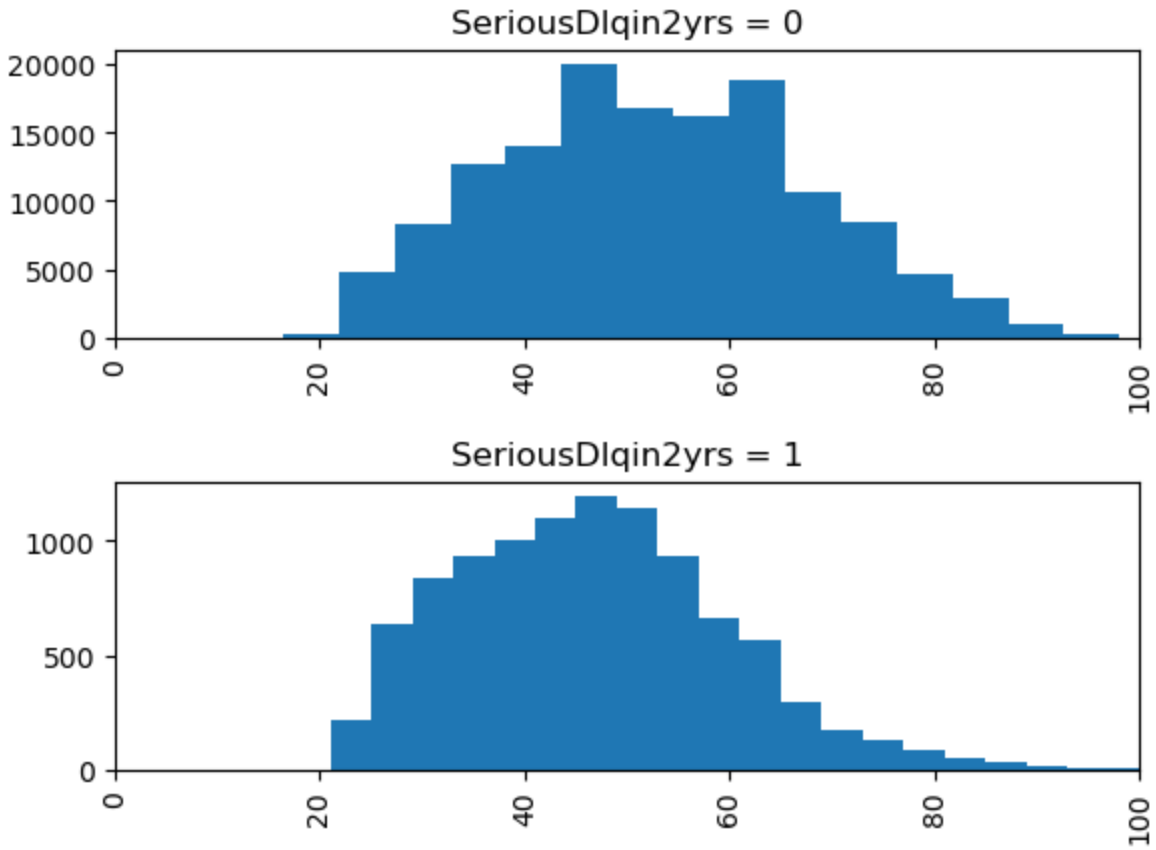


In [ ]:
```
majority = df[df.SeriousDlqin2yrs==0]
minority = df[df.SeriousDlqin2yrs==1]

# upsample minority class
df_upsampled = resample(minority, replace=True, n_samples=139974, random_state=47)

df_upsampled = pd.concat([majority, df_upsampled])
#plot new class counts
sns.countplot(x='SeriousDlqin2yrs', data=df_upsampled, palette='hls')
plt.show()
```



In [ ]:
```
age_hist = df['age'].hist(by=df['SeriousDlqin2yrs'], bins=20, layout=(2,1))
age_hist[0].set_xlim((0,100))
age_hist[0].set_title('SeriousDlqin2yrs = 0')
age_hist[1].set_xlim((0,100))
age_hist[1].set_title('SeriousDlqin2yrs = 1')
```

Out[ ]:
```
Text(0.5, 1.0, 'SeriousDlqin2yrs = 1')
```



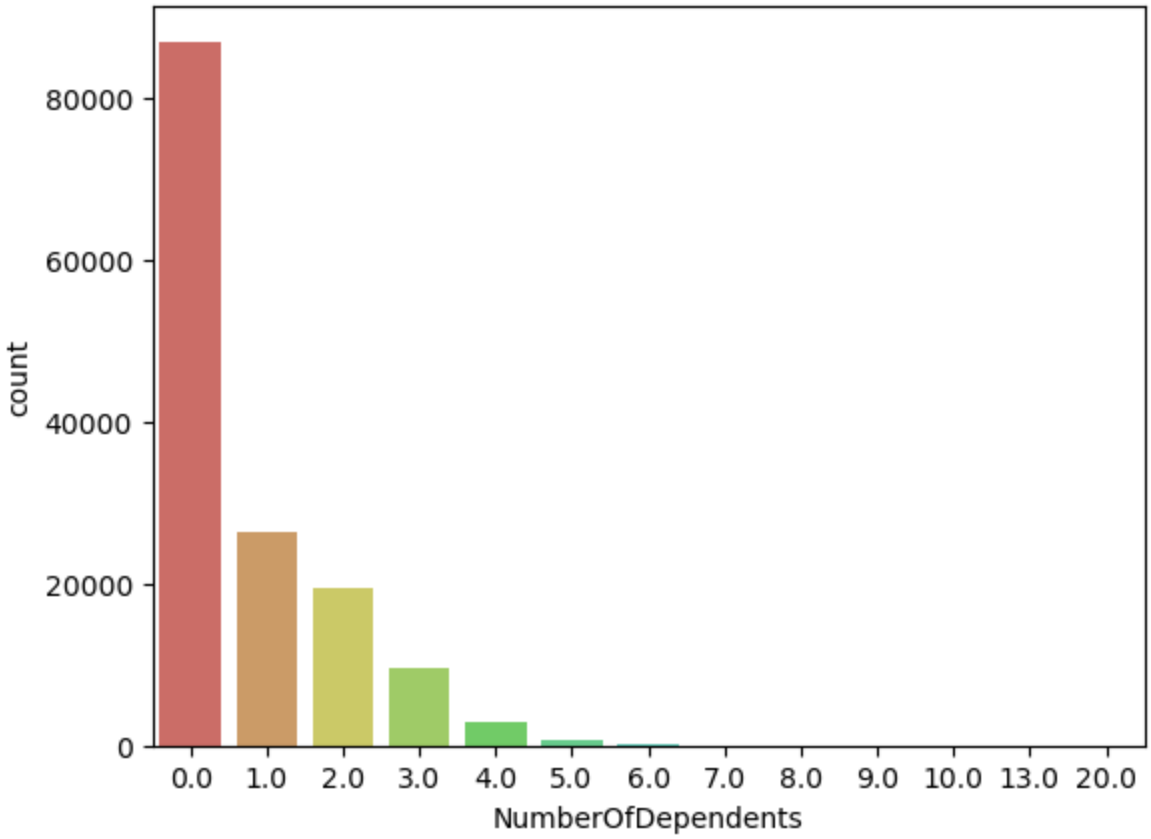
In [ ]:
```
# chance of outliers?
df.DebtRatio.quantile([.975])
```

Out[ ]:
```
0.975    3489.025
Name: DebtRatio, dtype: float64
```

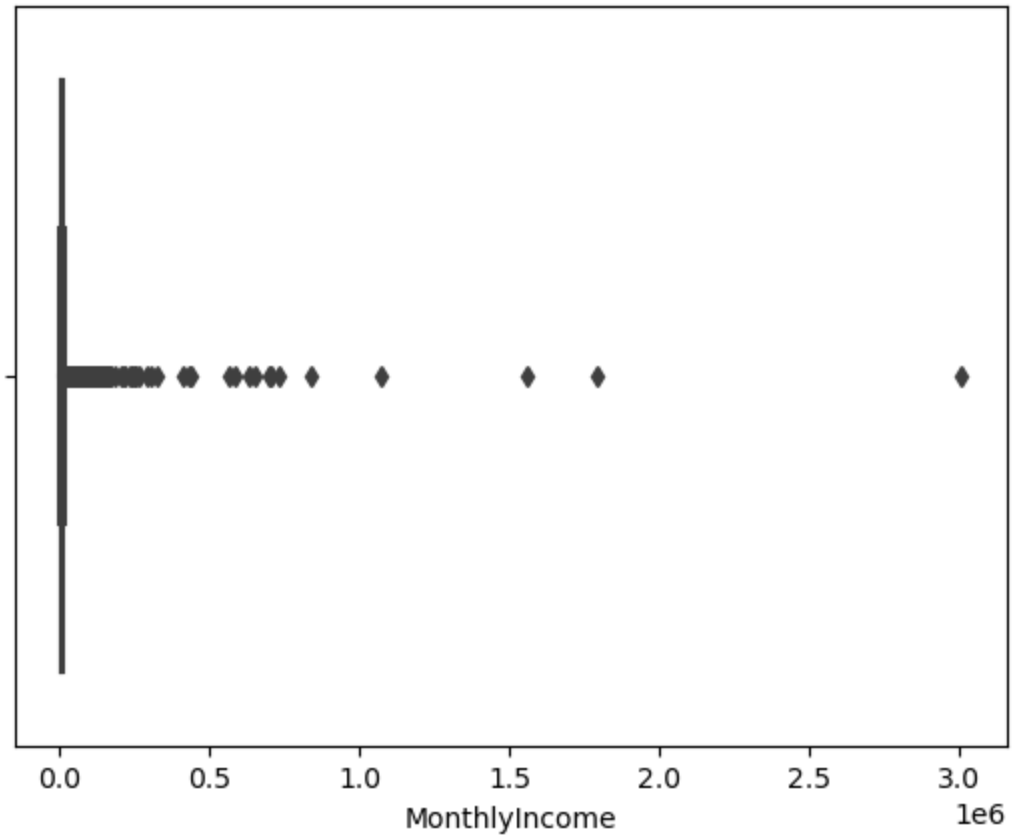
In [ ]:
```
# check monthly income and dependents have missing values
df.isnull().sum()
```

Out[ ]:
```
SeriousDlqin2yrs                        0
RevolvingUtilizationOfUnsecuredLines    0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
MonthlyIncome                       29731
NumberOfOpenCreditLinesAndLoans         0
NumberOfTimes90DaysLate                 0
NumberRealEstateLoansOrLines            0
NumberOfTime60-89DaysPastDueNotWorse    0
NumberOfDependents                   3924
dtype: int64
```

In [ ]:
```
# plot the distribution of dependents sns
sns.countplot(x='NumberOfDependents', data=df, palette='hls')
plt.show()
```



In [ ]:
```
# check for outliers in monthly income
sns.boxplot(x=df.MonthlyIncome)
plt.show()
```

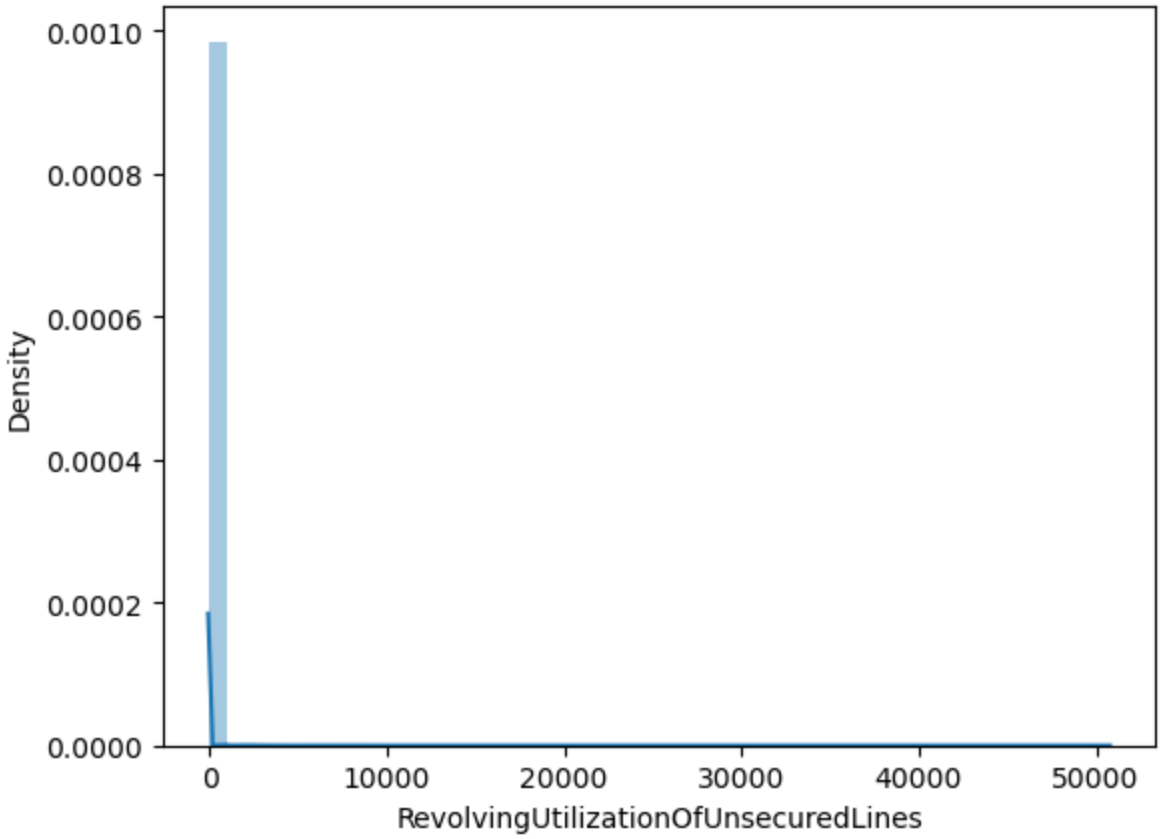


In [ ]:
```
# check Number of times 90 days late
df.groupby('NumberOfTimes90DaysLate').NumberOfTimes90DaysLate.count()
```

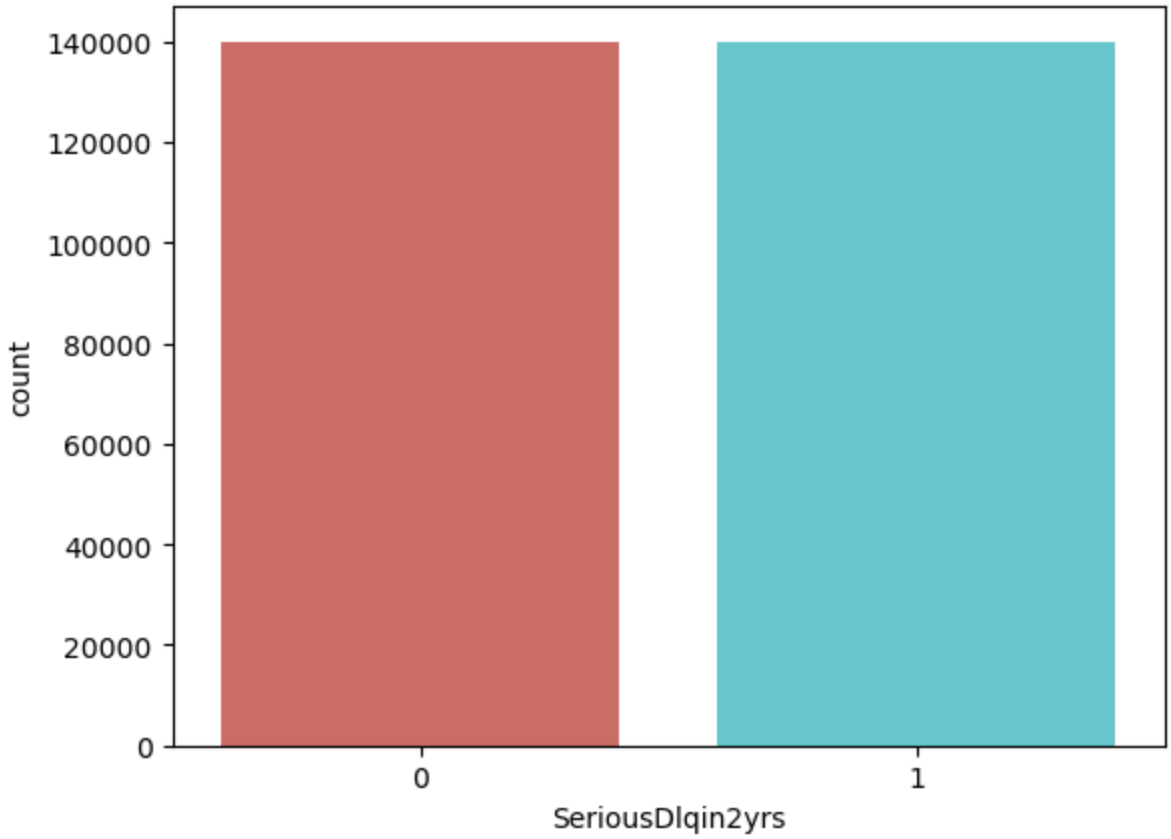
Out[ ]:
```
NumberOfTimes90DaysLate
0     141662
1       5243
2       1555
3        667
4        291
5        131
6         80
7         38
8         21
9         19
10         8
11         5
12         2
13         4
14         2
15         2
17         1
96         5
98       264
Name: NumberOfTimes90DaysLate, dtype: int64
```

In [ ]:
```
sns.distplot(df.RevolvingUtilizationOfUnsecuredLines)
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_34876\3114159400.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.RevolvingUtilizationOfUnsecuredLines)
```



In [ ]:
```
# upsample minority class

majority = df[df.SeriousDlqin2yrs==0]

minority = df[df.SeriousDlqin2yrs==1]

df_upsampled = resample(minority, replace=True, n_samples=139974, random_state=47)

df_upsampled = pd.concat([majority, df_upsampled])

sns.countplot(x='SeriousDlqin2yrs', data=df_upsampled, palette='hls')
plt.show()
```



Model Building

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

# train a random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=47)
rf.fit(X_train, y_train)


# cross validate for best parameters


param_grid = {
    'n_estimators': [100, 200, 300, 400 , 500],
    'max_depth': [5, 10, 15, 20 , 25],
    'min_samples_leaf': [1, 2, 3, 4, 5]
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbo
grid_search.fit(X_train, y_train)

# check best parameters
grid_search.best_params_
```

Out[ ]:
```
Fitting 5 folds for each of 125 candidates, totalling 625 fits
{'max_depth': 25, 'min_samples_leaf': 1, 'n_estimators': 200}
```

In [ ]:
```
# grid_search.best_params_
```

```
# train a random forest classifier with best parameters

rf = RandomForestClassifier(n_estimators=200, max_depth=20, min_samples_leaf=1, random
rf.fit(X_train, y_train)

# predict on test set
y_pred = rf.predict(X_test)

# check accuracy
metrics.accuracy_score(y_test, y_pred)

# check confusion matrix
metrics.confusion_matrix(y_test, y_pred)

# subplot for feature importance and ROC
fig, ax = plt.subplots(1, 2, figsize=(15, 5))

# plot feature importance

feature_importance = pd.Series(rf.feature_importances_, index=X.columns)

feature_importance.nlargest(10).plot(kind='barh', ax=ax[0])

# plot ROC curve using sns lineplot

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred)

sns.lineplot(x=fpr, y=tpr, ax=ax[1])
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Users\kevin\OneDrive\Desktop\154Kagge\154Kaggle.ipynb Cell 46 line 6
      <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=0'>1</a> # grid_search.best_params_
      <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=1'>2</a>
      <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=2'>3</a> # train a random forest classifier with best parameters
      <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=4'>5</a> rf = RandomForestClassifier(n_estimators=200, max_depth=20, min_samples_leaf=1, random_state=47)
----> <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=5'>6</a> rf.fit(X_train, y_train)
      <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=7'>8</a> # predict on test set
      <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X32sZmlsZQ%3D%3D?line=8'>9</a> y_pred = rf.predict(X_test)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\ensemble\_forest.py:345, in BaseForest.fit(self, X, y, sample_weight)
    343 if issparse(y):
    344     raise ValueError("sparse multilabel-indicator for y is not supported.")
--> 345 X, y = self._validate_data(
    346     X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    347 )
    348 if sample_weight is not None:
    349     sample_weight = _check_sample_weight(sample_weight, X)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\base.py:565, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, **check_params)
    563         y = check_array(y, input_name="y", **check_y_params)
    564     else:
--> 565         X, y = check_X_y(X, y, **check_params)
    566     out = X, y
    568 if not no_val_X and check_params.get("ensure_2d", True):

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\utils\validation.py:1106, in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
   1101         estimator_name = _check_estimator_name(estimator)
   1102     raise ValueError(
   1103         f"{estimator_name} requires y to be passed, but the target y is None"
   1104     )
-> 1106 X = check_array(
   1107     X,
   1108     accept_sparse=accept_sparse,
   1109     accept_large_sparse=accept_large_sparse,
   1110     dtype=dtype,
   1111     order=order,
   1112     copy=copy,
   1113     force_all_finite=force_all_finite,
   1114     ensure_2d=ensure_2d,
   1115     allow_nd=allow_nd,
   1116     ensure_min_samples=ensure_min_samples,
   1117     ensure_min_features=ensure_min_features,
   1118     estimator=estimator,
   1119     input_name="X",
```

```
   1120 )
   1122 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=est
imator)
   1124 check_consistent_length(X, y)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\utils\validation.py:9
21, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, for
ce_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimato
r, input_name)
    915         raise ValueError(
    916             "Found array with dim %d. %s expected <= 2."
    917             % (array.ndim, estimator_name)
    918         )
    920     if force_all_finite:
--> 921         _assert_all_finite(
    922             array,
    923             input_name=input_name,
    924             estimator_name=estimator_name,
    925             allow_nan=force_all_finite == "allow-nan",
    926         )
    928 if ensure_min_samples > 0:
    929     n_samples = _num_samples(array)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\utils\validation.py:1
61, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    144 if estimator_name and input_name == "X" and has_nan_error:
    145     # Improve the error message on how to handle missing values in
    146     # scikit-learn.
    147     msg_err += (
    148         f"\n{estimator_name} does not accept missing values"
    149         " encoded as NaN natively. For supervised learning, you might want"
   (...)
    159         "#estimators-that-handle-nan-values"
    160     )
--> 161 raise ValueError(msg_err)

ValueError: Input X contains NaN.
RandomForestClassifier does not accept missing values encoded as NaN natively. For su
pervised learning, you might want to consider sklearn.ensemble.HistGradientBoostingCl
assifier and Regressor which accept missing values encoded as NaNs natively. Alternat
ively, it is possible to preprocess the data, for instance by using an imputer transf
ormer in a pipeline or drop samples with missing values. See https://scikit-learn.or
g/stable/modules/impute.html You can find a list of all estimators that handle NaN va
lues at the following page: https://scikit-learn.org/stable/modules/impute.html#estim
ators-that-handle-nan-values
```

In [ ]:
```
# print auc
metrics.roc_auc_score(y_test, y_pred)
#0.8823701206845639
```

Out[ ]:
```
0.9757339185494316
```

Exporting the model and predictions For decision Trees

In [ ]:
```
df_test = pd.read_csv('cs-test.csv')
df_test.head()
# drop target variable
df_test.drop('SeriousDlqin2yrs', axis=1, inplace=True)
df_test.head()

# check for missing values
df_test.isnull().sum()

df_test['NumberOfDependents'].fillna(df_test['NumberOfDependents'].mode()[0], inplace=

# impute missing values for monthly income with median

df_test['MonthlyIncome'].fillna(df_test['MonthlyIncome'].median(), inplace=True)

# check if there are any missing values left
df_test.isnull().sum()


# get prediction probabilities for test set without the index column
y_pred_prob = rf.predict_proba(df_test.drop('Unnamed: 0', axis=1))



# add t datafram with application id
df_test['Probability'] = y_pred_prob[:,1]

df_test.head()

df_submission = df_test[['Unnamed: 0', 'Probability']]
df_submission.head()
#change first column name to Id
df_submission.rename(columns={'Unnamed: 0': 'Id'}, inplace=True)



df_submission.head()









df_submission.to_csv('submission.csv', index=False)
```

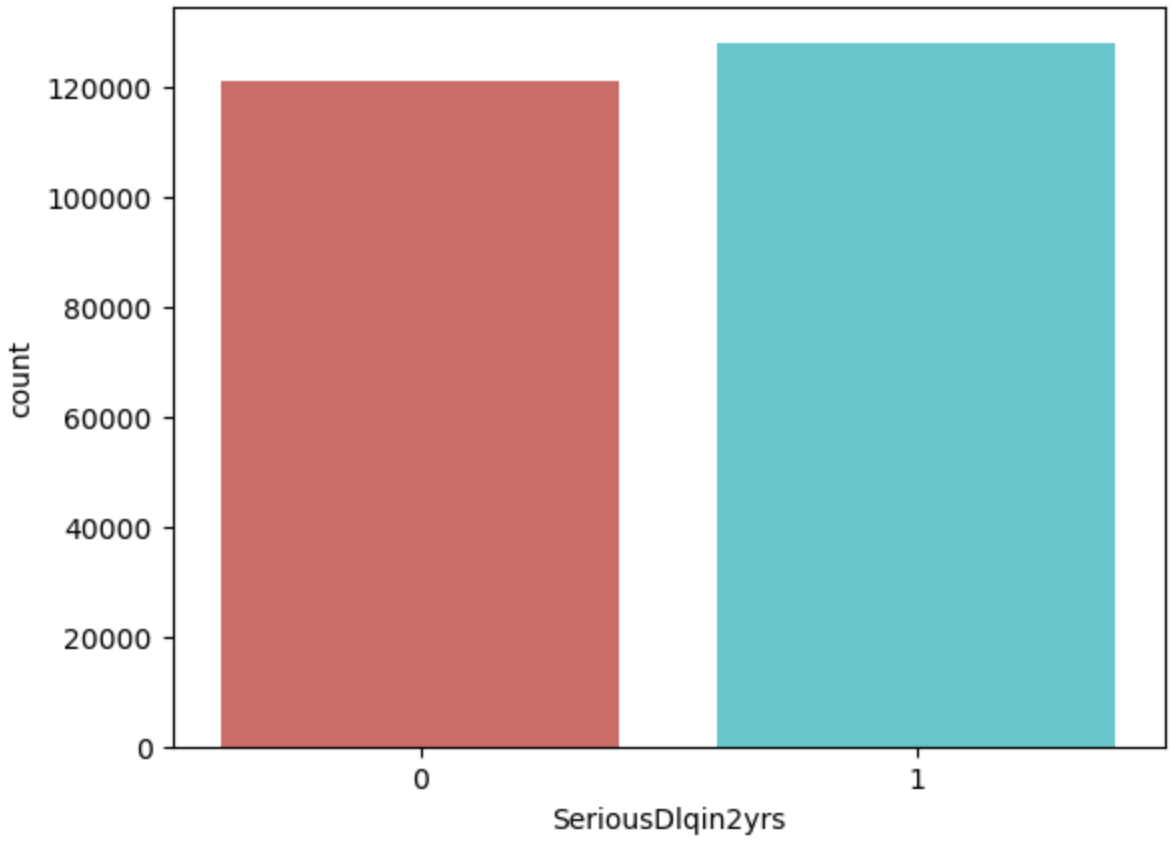
```
C:\Users\kevin\AppData\Local\Temp\ipykernel_27020\454482948.py:33: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_submission.rename(columns={'Unnamed: 0': 'Id'}, inplace=True)
```

Gradient Boosting

In [ ]:
```
gb = pd.read_csv('cs-training.csv')

# impute missing values for dependents with most freq
```

```
gb['NumberOfDependents'].fillna(gb['NumberOfDependents'].mode()[0], inplace=True)

# impute missing values for monthly income with median

gb['MonthlyIncome'].fillna(gb['MonthlyIncome'].median(), inplace=True)


majority = gb[gb.SeriousDlqin2yrs==0]
minority = gb[gb.SeriousDlqin2yrs==1]

# upsample minority class
gb_upsampled = resample(minority, replace=True, n_samples=139974, random_state=47)

gb_upsampled = pd.concat([majority, gb_upsampled])
# check if there are any missing values left
gb_upsampled.isnull().sum()
#drop outlier observations in monthly income
gb_upsampled = gb_upsampled[gb.MonthlyIncome < 10000]


# check class counts
sns.countplot(x='SeriousDlqin2yrs', data=gb_upsampled, palette='hls')
plt.show()
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_22176\2040137649.py:21: UserWarning: Bool
ean Series key will be reindexed to match DataFrame index.
  gb_upsampled = gb_upsampled[gb.MonthlyIncome < 10000]
```



In [ ]:

```
# find best parameters for gradient boosting
param_grid = {
    'n_estimators': [100, 300],
    'learning_rate': [0.1, 0.5],
    'max_depth': [5, 10, 15]
```

```
}

grid_search = GridSearchCV(estimator=GradientBoostingClassifier(), param_grid=param_gr

# train test split
Xgb = gb_upsampled.drop('SeriousDlqin2yrs', axis=1)
ygb = gb_upsampled['SeriousDlqin2yrs']

Xgb_train, Xgb_test, ygb_train, ygb_test = train_test_split(Xgb, ygb, test_size=0.3, r

# grid_search.fit(Xgb_train, ygb_train)

# # check best parameters

# grid_search.best_params_
```

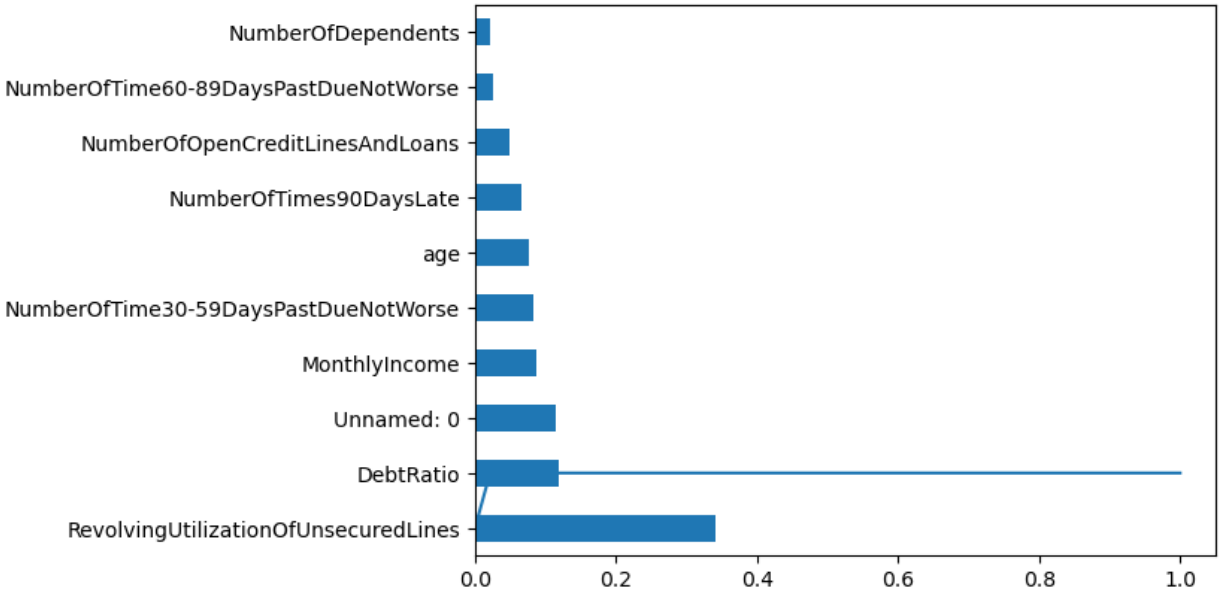
In [ ]:
```
# use randomserach to find best parameters
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
    'n_estimators': [100, 300],
    'learning_rate': [0.1, 0.5],
    'max_depth': [5, 10, 15]
}
Xgb = gb_upsampled.drop('SeriousDlqin2yrs', axis=1)
ygb = gb_upsampled['SeriousDlqin2yrs']
Xgb_train, Xgb_test, ygb_train, ygb_test = train_test_split(Xgb, ygb, test_size=0.3, r
random_search = RandomizedSearchCV(estimator=GradientBoostingClassifier(), param_distr

random_search.fit(Xgb_train, ygb_train)

# check best parameters

random_search.best_params_
```

Out[ ]:
```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'n_estimators': 300, 'max_depth': 15, 'learning_rate': 0.5}
```

In [ ]:
```
Xgb = gb_upsampled.drop('SeriousDlqin2yrs', axis=1)
ygb = gb_upsampled['SeriousDlqin2yrs']
Xgb_train, Xgb_test, ygb_train, ygb_test = train_test_split(Xgb, ygb, test_size=0.3, r
#  train a gradient boosting classifier with best parameters
gb = GradientBoostingClassifier(n_estimators=300, learning_rate=0.5, max_depth=15)
gb.fit(Xgb_train, ygb_train)

# predict on test set
ygb_pred = gb.predict(Xgb_test)

# check accuracy

metrics.accuracy_score(ygb_test, ygb_pred)

# check confusion matrix

metrics.confusion_matrix(ygb_test, ygb_pred)

# plot feature importance

feature_importance = pd.Series(gb.feature_importances_, index=Xgb.columns)
```

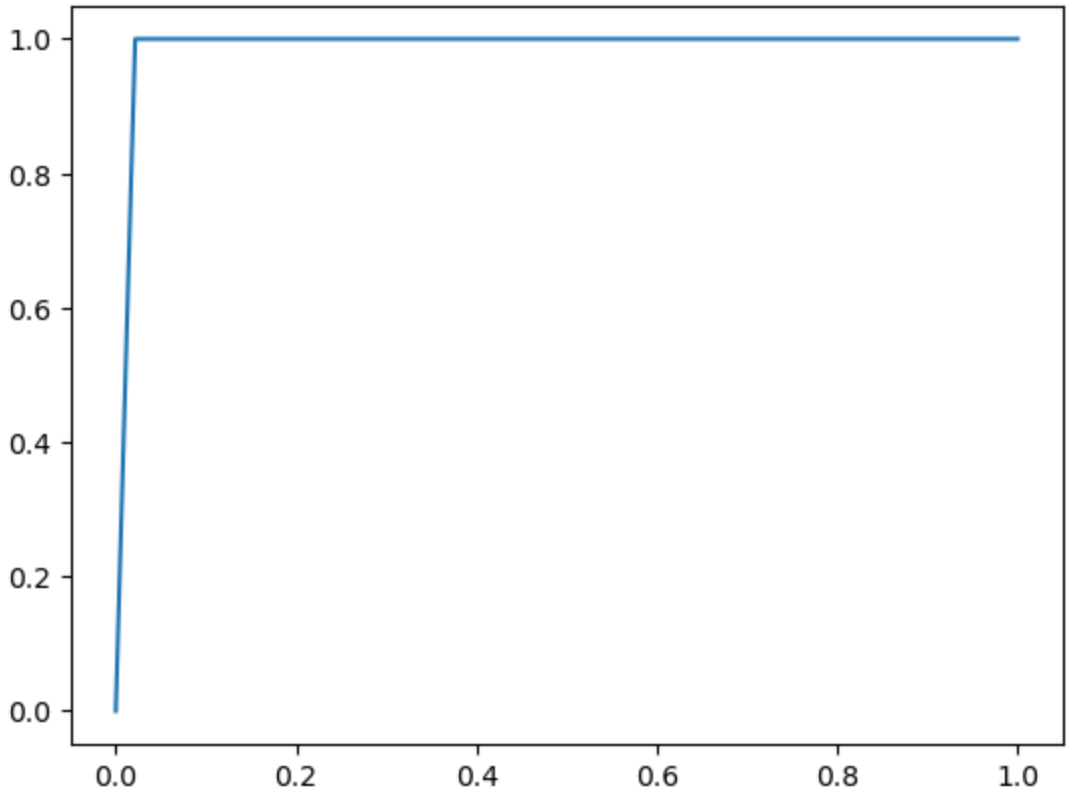
```
feature_importance.nlargest(10).plot(kind='barh')

# plot ROC curve using sns lineplot

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(ygb_test, ygb_pred)

sns.lineplot(x=fpr, y=tpr)


# print auc

metrics.roc_auc_score(ygb_test, ygb_pred)
```

Out[ ]: 0.9892378166565483



In [ ]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(ygb_test, ygb_pred)

sns.lineplot(x=fpr, y=tpr)
```

Out[ ]: <AxesSubplot: >



Export Predictions for Gradient Boosting

In [ ]:
```
df_test = pd.read_csv('cs-test.csv')
df_test.head()
# drop target variable
df_test.drop('SeriousDlqin2yrs', axis=1, inplace=True)
df_test.head()

# check for missing values
df_test.isnull().sum()

df_test['NumberOfDependents'].fillna(df_test['NumberOfDependents'].mode()[0], inplace=

# impute missing values for monthly income with median

df_test['MonthlyIncome'].fillna(df_test['MonthlyIncome'].median(), inplace=True)

# check if there are any missing values left
df_test.isnull().sum()

df_test.head()
# # get prediction probabilities for test set without the index column
y_pred_prob = gb.predict_proba(df_test)
y_pred_prob
df_submission = df_test[['Unnamed: 0']]
df_submission.head()
#change first column name to Id
df_submission.rename(columns={'Unnamed: 0': 'Id'}, inplace=True)
df_submission.head()

df_submission['Probability'] = y_pred_prob[:,1]
```

```
df_submission.head()
```

```
df_submission.to_csv('submissiongb.csv', index=False)
```

```
C:\Users\kevin\AppData\Local\Temp\ipykernel_22176\4180032406.py:26: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_submission.rename(columns={'Unnamed: 0': 'Id'}, inplace=True)
C:\Users\kevin\AppData\Local\Temp\ipykernel_22176\4180032406.py:29: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  df_submission['Probability'] = y_pred_prob[:,1]
```

Guassian Process Classifier

In [ ]:

```
# import Gaussian Process Classifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF

# import data
df = pd.read_csv('cs-training.csv', index_col=0)

# impute missing values for dependents with most freq

df['NumberOfDependents'].fillna(df['NumberOfDependents'].mode()[0], inplace=True)

# impute missing values for monthly income with median

df['MonthlyIncome'].fillna(df['MonthlyIncome'].median(), inplace=True)

# drop outlier observations in monthly income

df = df[df.MonthlyIncome < 10000]


# upsample minority class

majority = df[df.SeriousDlqin2yrs==0]


minority = df[df.SeriousDlqin2yrs==1]
```

```
df_upsampled = resample(minority, replace=True, n_samples=139974, random_state=47)


df_upsampled = pd.concat([majority, df_upsampled])


# train test split

X = df_upsampled.drop('SeriousDlqin2yrs', axis=1)

y = df_upsampled['SeriousDlqin2yrs']

Xgp_train, Xgp_test, ygp_train, ygp_test = train_test_split(X, y, test_size=0.3, randc

# train a gaussian process classifier

kernel = 1.0 * RBF(1.0)

gpc = GaussianProcessClassifier(kernel=kernel, random_state=47)

gpc.fit(Xgp_train, ygp_train)

# see ROC curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(ygp_test, gpc.predict(Xgp_test))

sns.lineplot(x=fpr, y=tpr)

# print auc

metrics.roc_auc_score(ygp_test, gpc.predict(Xgp_test))
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
c:\Users\kevin\OneDrive\Desktop\154Kagge\154Kaggle.ipynb Cell 38 line 4
     <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X54sZmlsZQ%3D%3D?line=44'>45</a> kernel = 1.0 * RBF(1.0)
     <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X54sZmlsZQ%3D%3D?line=46'>47</a> gpc = GaussianProcessClassifier(kernel=kernel, random_state=47)
---> <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X54sZmlsZQ%3D%3D?line=48'>49</a> gpc.fit(Xgp_train, ygp_train)
     <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X54sZmlsZQ%3D%3D?line=50'>51</a> # see ROC curve
     <a href='vscode-notebook-cell:/c%3A/Users/kevin/OneDrive/Desktop/154Kagge/154Kaggle.ipynb#X54sZmlsZQ%3D%3D?line=52'>53</a> from sklearn.metrics import roc_curve

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\_gpc.py:742, in GaussianProcessClassifier.fit(self, X, y)
    739     else:
    740         raise ValueError("Unknown multi-class mode %s" % self.multi_class)
--> 742 self.base_estimator_.fit(X, y)
    744 if self.n_classes_ > 2:
    745     self.log_marginal_likelihood_value_ = np.mean(
    746         [
    747             estimator.log_marginal_likelihood()
    748             for estimator in self.base_estimator_.estimators_
    749         ]
    750     )

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\_gpc.py:229, in _BinaryGaussianProcessClassifierLaplace.fit(self, X, y)
    225         return -self.log_marginal_likelihood(theta, clone_kernel=False)
    227 # First optimize starting from theta specified in kernel
    228 optima = [
--> 229     self._constrained_optimization(
    230         obj_func, self.kernel_.theta, self.kernel_.bounds
    231     )
    232 ]
    234 # Additional runs are performed from log-uniform chosen initial
    235 # theta
    236 if self.n_restarts_optimizer > 0:

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\_gpc.py:474, in _BinaryGaussianProcessClassifierLaplace._constrained_optimization(self, obj_func, initial_theta, bounds)
    472 def _constrained_optimization(self, obj_func, initial_theta, bounds):
    473     if self.optimizer == "fmin_l_bfgs_b":
--> 474         opt_res = scipy.optimize.minimize(
    475             obj_func, initial_theta, method="L-BFGS-B", jac=True, bounds=bounds
    476         )
    477         _check_optimize_result("lbfgs", opt_res)
    478         theta_opt, func_min = opt_res.x, opt_res.fun

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_minimize.py:696, in minimize(fun, x0, args, method, jac, hess, hessp, bounds, constraints, tol, callback, options)
    693     res = _minimize_newtoncg(fun, x0, args, jac, hess, hessp, callback,
    694                              **options)
    695 elif meth == 'l-bfgs-b':
--> 696     res = _minimize_lbfgsb(fun, x0, args, jac, bounds,
```

```
    697                             callback=callback, **options)
    698 elif meth == 'tnc':
    699     res = _minimize_tnc(fun, x0, args, jac, bounds, callback=callback,
    700                         **options)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_lbfgsb_py.py:
305, in _minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps, m
axfun, maxiter, iprint, callback, maxls, finite_diff_rel_step, **unknown_options)
    302     else:
    303         iprint = disp
--> 305 sf = _prepare_scalar_function(fun, x0, jac=jac, args=args, epsilon=eps,
    306                                bounds=new_bounds,
    307                                finite_diff_rel_step=finite_diff_rel_step)
    309 func_and_grad = sf.fun_and_grad
    311 fortran_int = _lbfgsb.types.intvar.dtype

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_optimize.py:3
32, in _prepare_scalar_function(fun, x0, jac, args, bounds, epsilon, finite_diff_rel_
step, hess)
    328     bounds = (-np.inf, np.inf)
    330 # ScalarFunction caches. Reuse of fun(x) during grad
    331 # calculation reduces overall function evaluations.
--> 332 sf = ScalarFunction(fun, x0, args, grad, hess,
    333                     finite_diff_rel_step, bounds, epsilon=epsilon)
    335 return sf

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_differentiabl
e_functions.py:158, in ScalarFunction.__init__(self, fun, x0, args, grad, hess, finit
e_diff_rel_step, finite_diff_bounds, epsilon)
    155     self.f = fun_wrapped(self.x)
    157 self._update_fun_impl = update_fun
--> 158 self._update_fun()
    160 # Gradient evaluation
    161 if callable(grad):

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_differentiabl
e_functions.py:251, in ScalarFunction._update_fun(self)
    249 def _update_fun(self):
    250     if not self.f_updated:
--> 251         self._update_fun_impl()
    252         self.f_updated = True

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_differentiabl
e_functions.py:155, in ScalarFunction.__init__.<locals>.update_fun()
    154 def update_fun():
--> 155     self.f = fun_wrapped(self.x)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_differentiabl
e_functions.py:137, in ScalarFunction.__init__.<locals>.fun_wrapped(x)
    133 self.nfev += 1
    134 # Send a copy because the user may overwrite it.
    135 # Overwriting results in undefined behaviour because
    136 # fun(self.x) will change self.x, with the two no longer linked.
--> 137 fx = fun(np.copy(x), *args)
    138 # Make sure the function returns a true scalar
    139 if not np.isscalar(fx):

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_optimize.py:7
6, in MemoizeJac.__call__(self, x, *args)
     74 def __call__(self, x, *args):
```

```
     75     """ returns the function value """
---> 76     self._compute_if_needed(x, *args)
     77     return self._value

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\scipy\optimize\_optimize.py:70, in MemoizeJac._compute_if_needed(self, x, *args)
     68 if not np.all(x == self.x) or self._value is None or self.jac is None:
     69     self.x = np.asarray(x).copy()
---> 70     fg = self.fun(x, *args)
     71     self.jac = fg[1]
     72     self._value = fg[0]

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\_gpc.py:220, in _BinaryGaussianProcessClassifierLaplace.fit.<locals>.obj_func(theta, eval_gradient)
    218 def obj_func(theta, eval_gradient=True):
    219     if eval_gradient:
--> 220         lml, grad = self.log_marginal_likelihood(
    221             theta, eval_gradient=True, clone_kernel=False
    222         )
    223         return -lml, -grad
    224     else:

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\_gpc.py:379, in _BinaryGaussianProcessClassifierLaplace.log_marginal_likelihood(self, theta, eval_gradient, clone_kernel)
    376     kernel.theta = theta
    378 if eval_gradient:
--> 379     K, K_gradient = kernel(self.X_train_, eval_gradient=True)
    380 else:
    381     K = kernel(self.X_train_)

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\kernels.py:940, in Product.__call__(self, X, Y, eval_gradient)
    912 """Return the kernel k(X, Y) and optionally its gradient.
    913
    914 Parameters
   (...)
    937     is True.
    938 """
    939 if eval_gradient:
--> 940     K1, K1_gradient = self.k1(X, Y, eval_gradient=True)
    941     K2, K2_gradient = self.k2(X, Y, eval_gradient=True)
    942     return K1 * K2, np.dstack(
    943         (K1_gradient * K2[:, :, np.newaxis], K2_gradient * K1[:, :, np.newaxis])
    944     )

File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\sklearn\gaussian_process\kernels.py:1253, in ConstantKernel.__call__(self, X, Y, eval_gradient)
   1250 elif eval_gradient:
   1251     raise ValueError("Gradient can only be evaluated when Y is None.")
-> 1253 K = np.full(
   1254     (_num_samples(X), _num_samples(Y)),
   1255     self.constant_value,
   1256     dtype=np.array(self.constant_value).dtype,
   1257 )
   1258 if eval_gradient:
   1259     if not self.hyperparameter_constant_value.fixed:
```

```
File c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\numpy\core\numeric.py:343, in full(shape, fill_value, dtype, order, like)
    341     fill_value = asarray(fill_value)
    342     dtype = fill_value.dtype
--> 343 a = empty(shape, dtype, order)
    344 multiarray.copyto(a, fill_value, casting='unsafe')
    345 return a

MemoryError: Unable to allocate 249. GiB for an array with shape (182713, 182713) and data type float64
```

Deep Learning

In [ ]:
```
# import tensor flow
import tensorflow as tf
# import batch normalization
from keras.layers import BatchNormalization
import numpy as np
from sklearn.model_selection import KFold
from keras.models import Sequential
from keras.layers import Dense, Dropout, Attention
from keras.regularizers import l1, l2
from skopt import gp_minimize
from skopt.space import Real, Categorical, Integer
from skopt.utils import use_named_args
from keras.optimizers import Adam
#import data


# train test split

# train test validation split

X = df_upsampled.drop('SeriousDlqin2yrs', axis=1)

y = df_upsampled['SeriousDlqin2yrs']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=




X_train.isnull().sum()
```

Out[ ]:
```
RevolvingUtilizationOfUnsecuredLines    0
age                                     0
NumberOfTime30-59DaysPastDueNotWorse    0
DebtRatio                               0
MonthlyIncome                           0
NumberOfOpenCreditLinesAndLoans         0
NumberOfTimes90DaysLate                 0
NumberRealEstateLoansOrLines            0
NumberOfTime60-89DaysPastDueNotWorse    0
NumberOfDependents                      0
dtype: int64
```

In [ ]:
```
# build a neural network
```

```
model = Sequential(
    [
        Dense(256, input_dim=10, activation='relu'),
        Dense(128, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(16, activation='relu'),
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ]
)

# compile model with low learning rate

model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.00001), metrics=['accura

# fit model

model.fit(X_train, y_train, epochs=50, batch_size=32)

# evaluate model

model.evaluate(X_test, y_test)

# predict on test set

y_pred = model.predict(X_test)

# check auc

metrics.roc_auc_score(y_test, y_pred)
```

```
c:\Users\kevin\anaconda3\envs\TF\lib\site-packages\keras\optimizers\optimizer_v2\ada
m.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

```
Epoch 1/40
6124/6124 [==============================] - 38s 6ms/step - loss: 0.6869 - accuracy:
0.5522
Epoch 2/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.6871 - accuracy:
0.5455
Epoch 3/40
6124/6124 [==============================] - 15s 2ms/step - loss: 0.6823 - accuracy:
0.5559
Epoch 4/40
6124/6124 [==============================] - 15s 2ms/step - loss: 0.6436 - accuracy:
0.6320
Epoch 5/40
6124/6124 [==============================] - 15s 2ms/step - loss: 0.6110 - accuracy:
0.6719
Epoch 6/40
6124/6124 [==============================] - 15s 2ms/step - loss: 0.5790 - accuracy:
0.7091
Epoch 7/40
6124/6124 [==============================] - 15s 2ms/step - loss: 0.5830 - accuracy:
0.7001
Epoch 8/40
6124/6124 [==============================] - 15s 3ms/step - loss: 0.5957 - accuracy:
0.6789
Epoch 9/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.5556 - accuracy:
0.7338
Epoch 10/40
6124/6124 [==============================] - 15s 3ms/step - loss: 0.6555 - accuracy:
0.5650
Epoch 11/40
6124/6124 [==============================] - 15s 3ms/step - loss: 0.5982 - accuracy:
0.6743
Epoch 12/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.5388 - accuracy:
0.7474
Epoch 13/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.5353 - accuracy:
0.7458
Epoch 14/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.5578 - accuracy:
0.7276
Epoch 15/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.5927 - accuracy:
0.6670
Epoch 16/40
6124/6124 [==============================] - 19s 3ms/step - loss: 0.6901 - accuracy:
0.5153
Epoch 17/40
6124/6124 [==============================] - 22s 4ms/step - loss: 0.6724 - accuracy:
0.5515
Epoch 18/40
6124/6124 [==============================] - 20s 3ms/step - loss: 0.6342 - accuracy:
0.6183
Epoch 19/40
6124/6124 [==============================] - 19s 3ms/step - loss: 0.5664 - accuracy:
0.7272
Epoch 20/40
6124/6124 [==============================] - 19s 3ms/step - loss: 0.5582 - accuracy:
0.7356
```

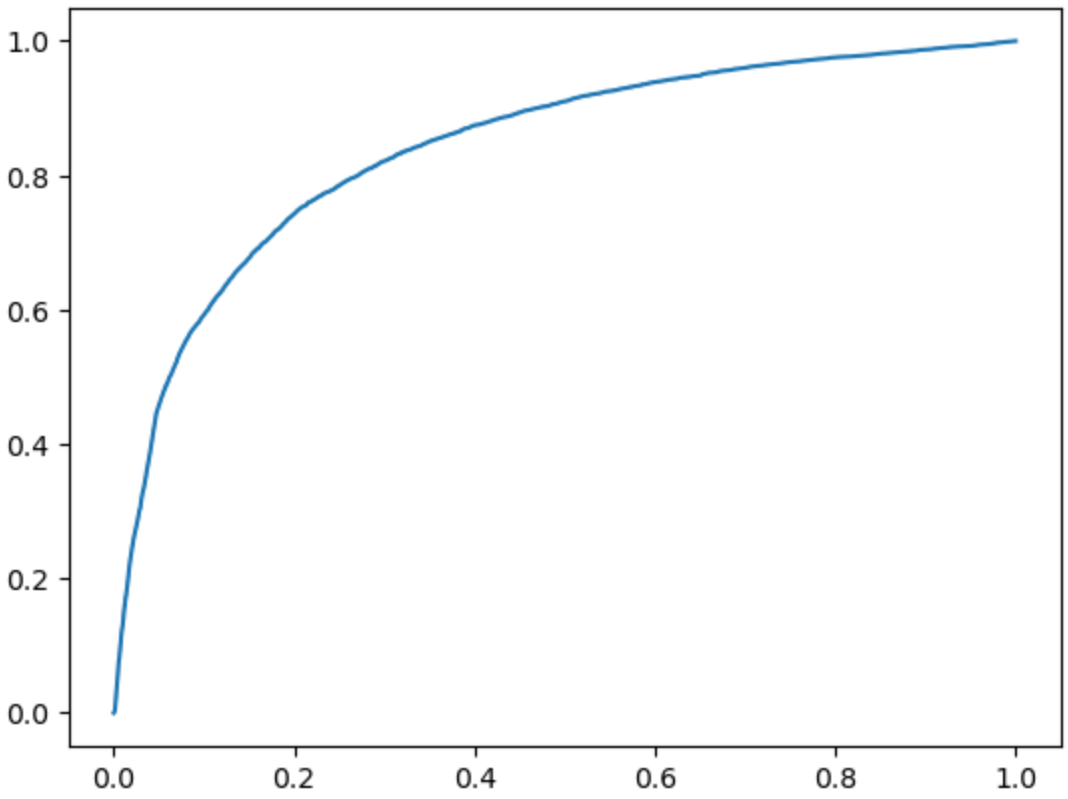
```
Epoch 21/40
6124/6124 [==============================] - 19s 3ms/step - loss: 0.5406 - accuracy:
0.7497
Epoch 22/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.5837 - accuracy:
0.6908
Epoch 23/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.6259 - accuracy:
0.6230
Epoch 24/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.6828 - accuracy:
0.5363
Epoch 25/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.6921 - accuracy:
0.5004
Epoch 26/40
6124/6124 [==============================] - 19s 3ms/step - loss: 0.6914 - accuracy:
0.5020
Epoch 27/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.6919 - accuracy:
0.5020
Epoch 28/40
6124/6124 [==============================] - 18s 3ms/step - loss: 0.6915 - accuracy:
0.5003
Epoch 29/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.6920 - accuracy:
0.5041
Epoch 30/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.6919 - accuracy:
0.5064
Epoch 31/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.6924 - accuracy:
0.5009
Epoch 32/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.6925 - accuracy:
0.5021
Epoch 33/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.6756 - accuracy:
0.5417
Epoch 34/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.6921 - accuracy:
0.5034
Epoch 35/40
6124/6124 [==============================] - 16s 3ms/step - loss: 0.6924 - accuracy:
0.5029
Epoch 36/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.6927 - accuracy:
0.5030
Epoch 37/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.6556 - accuracy:
0.5783
Epoch 38/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.5684 - accuracy:
0.7196
Epoch 39/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.5615 - accuracy:
0.7291
Epoch 40/40
6124/6124 [==============================] - 17s 3ms/step - loss: 0.5518 - accuracy:
0.7394
```

```
2625/2625 [==============================] - 4s 2ms/step - loss: 0.5903 - accuracy:
0.6958
2625/2625 [==============================] - 4s 1ms/step
0.8414450943727438
```

Out[ ]:

In [ ]:
```python
# plot ROC curve using sns lineplot

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred)

sns.lineplot(x=fpr, y=tpr)
```

Out[ ]:
```
<AxesSubplot: >
```



In [ ]:
```python
df_test = pd.read_csv('cs-test.csv')
df_test.head()


# check for missing values
df_test.isnull().sum()
# replace when age is 0 with median

df_test.loc[df_test.age == 0, 'age'] = df.age.median()
df_test['age'] = df_test['age'].clip(upper=100)
df_test.loc[df_test.MonthlyIncome > 15000, 'MonthlyIncome'] = 15000

# Define age bins
bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79', '80-89
df_test['age_group'] = pd.cut(df_test['age'], bins=bins, labels=labels, right=False)

# Calculate median income for each age group
```

```
median_incomes = df_test.groupby('age_group')['MonthlyIncome'].median()

# Impute missing values
for age_group in labels:
    median_income = median_incomes[age_group]
    df_test.loc[(df_test['age_group'] == age_group) & (df_test['MonthlyIncome'].isnull

# Drop the age_group column if no longer needed
df_test.drop('age_group', axis=1, inplace=True)


df_test.loc[df_test.NumberOfDependents > 5, 'NumberOfDependents'] = 5

# impute using mediam number of dependents using sklearn SimpleImputer

imputer = SimpleImputer(strategy='median')
df_test['NumberOfDependents'] = imputer.fit_transform(df_test[['NumberOfDependents']])


# impute missing values for monthly income with median

df_test['MonthlyIncome'].fillna(df_test['MonthlyIncome'].median(), inplace=True)

# check if there are any missing values left

# drop unnamed column and save for later

df_test_id = df_test['Unnamed: 0']

df_test.drop('Unnamed: 0', axis=1, inplace=True)

df_test.head()

#drop predictor variable
df_test.drop('SeriousDlqin2yrs', axis=1, inplace=True)


# get prediction probabilities for test set without the index column

y_pred_prob = model.predict(df_test)

df_submission = pd.DataFrame(df_test_id)
df_submission.head()
#change first column name to Id
df_submission.rename(columns={'Unnamed: 0': 'Id'}, inplace=True)
df_submission.head()

df_submission['Probability'] = y_pred_prob[:,0]




df_submission.head()







# df_submission.to_csv('submissiongb.csv', index=False)
```

```
3172/3172 [==============================] - 5s 2ms/step
```

Out[ ]:

	Id	Probability
0	1	0.701582
1	2	0.683642
2	3	0.488926
3	4	0.821051
4	5	0.815125

In [ ]:
```
df_submission.to_csv('submissionnn3.csv', index=False)
```