NBS Titanic Challenge

Purpose

This notebooks has the purpose to work with Titanic Challenge using machine learning algorithms.

Methodology

The goal of this experiment is to predict if a passenger surviver or not with base in some features (e.g. age, sex, passenger class).

First of all, I have to make some data exploration to verify if has any missing value on train set. If has more than 30% of the samples in the particular feature, I have to drop this feature, otherwise I have to implement some technique to handle with this technique.

After that, I have to build some machine learning models and evaluate this models. If the results are solid I can made the prediction in test set and send the predictions on Kaggle to check the performance of my models.

Results

Using Feature Engineering for generating new features, feature selection/discretization the follow model was used to fit the train set (80% of original train set), and after that we evaluated in test set (10% of original train set). The results of the original test set (send for submission) follow below:

	Validation Set	**Validation Set**	**Test Set**
Classifiers	Accuracy	F1-score	Public-Score
Logistic Regression	85%	79%	78%
SVM	85%	79%	77%
Decision Tree	83%	76%	74%
Random Forest	83%	77%	77%
AdaBoost	84%	79%	77%
Gradient Boosting	80%	73%	75%
XGBoost	85%	77%	76%

Voting Classifier (x̄)	85%	77%	78%

The results show that, Logistic Regression and ensemble voting (combine all predictions of classifiers) got the highest possible score, using the feature engineering process apply after a better understandable of the dataset.

Suggested next steps

Using another robust techniques in modeling or using another type of feature extraction (e.g. Principal Component Analysis).

Setup

Library import

We import all the required Python libraries

In [ ]:
```
import warnings
import pandas as pd
import numpy as np
warnings.filterwarnings("ignore")
import missingno as msno

from sklearn.feature_selection import VarianceThreshold, SelectKBest, chi
from sklearn.preprocessing import KBinsDiscretizer, StandardScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, OrdinalEnc
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, GridSearchCV, learn
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, roc_curve, roc_auc_


# Visualizations
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
%matplotlib inline
```

Parameter definition

We set all relevant parameters for our notebook. By convention, parameters are uppercase, while all the other variables follow Python's guidelines.

In [ ]:
```
TRAIN_DATA_PATH = "../data/train.csv"
```

Data import

We retrieve all the required data for the analysis.

In [ ]:
```
df = pd.read_csv(TRAIN_DATA_PATH)
df.tail()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Data Exploration

Look at the outcome/ground truth distribution

In [ ]:
```
def outcome_distribution(df, outcome):
    fig = px.pie(df, names=outcome, title="Grount Truth Distribution", ho
    fig.show()
```

In [ ]:
```
outcome_distribution(df, outcome="Survived")
```

In [ ]:
```
class_proportion = df["Survived"].value_counts(normalize=True)
print(f"Proportion: ")
print(f"Not Survived: {class_proportion[0]:.2f}")
print(f"Survived: {class_proportion[1]:.2f}")
```

```
Proportion: 
Not Survived: 0.62
Survived: 0.38
```

We have a umbalanced problem. The majority is that not surived passenger

Univariate Analysis

Pclass: A proxy for sicio-economic status

- 1st: Upper;
- 2nd: Middle;
- 3rd: Lower.

In [ ]:
```
def plot_piegraph(df, feature_name, title):
    fig = go.Figure(data=[go.Pie(labels=df[feature_name], title=title, ho
    fig.show()
```

In [ ]:
```
plot_piegraph(df, feature_name="Pclass", title="Socio Economic Class")
```

The majority of the passengers embarked in the titanic was of the **lower** class. And Pclass is categorical feature.

Sex analysis

In [ ]:
```
def feature_distribution(df, feature_name, title):
    if (len(list(df[feature_name].value_counts(normalize=True).values)) <
        fig = px.pie(df, names=feature_name, title=title, hole=0.3)
    else:
        fig = px.histogram(df, x=feature_name, marginal='box', histnorm="
    fig.show()
```

In [ ]:
```
feature_distribution(df, feature_name="Sex", title="Sex distribution")
```

We have 64% for male sex and 35% for female. Its a categorical feature

Age analysis

- Its fractional if less than 1;
- If the age is estimated, the value will be xx.5.

In [ ]:
```
feature_distribution(df, feature_name="Age", title="Age distribution")
```

Over this analysis we got that the Median is 28 years, the majority stays in 20 to 38. And we have a fews outliers (6 samples) over greater than 65.

SibSp: Family relations

- Sibling: borther, sister, stepbrother, stepsister;
- Spouse: husband, wife (mistress, and fiancés were ignored)

In [ ]:
```
feature_distribution(df, feature_name="SibSp", title="SibSp: Family Relat
```

Parch: Family Relation

- Parent: mother, father;
- Child: daughter, son, step daughter, stepson;

In [ ]:
```
feature_distribution(df, feature_name="Parch", title="Parch: Family Relat
```

Possible the **SibSp** and **Parch** feature are not linearly independent, and we also possible to sum together to form a new feature in the feature engineering process.

Ticket Number Analysis

In [ ]:
```
feature_distribution(df, feature_name="Ticket", title="Ticket Number") #
```

In [ ]:
```
df["Ticket"].value_counts().shape, df.shape # After this diversity we can
```

Out[ ]:
```
((681,), (891, 12))
```

Fare Analysis

In [ ]:
```
feature_distribution(df, feature_name="Fare", title="Fare")
```

For this continous feature (fare) we have a median 14.45, upper 65, and lower 0 respectively. The majority stay in 7.9-31.

Cabin Analysis

In [ ]:
```
fig = px.bar(df, x="Pclass", y="Survived", color="Cabin")
fig.show()
```

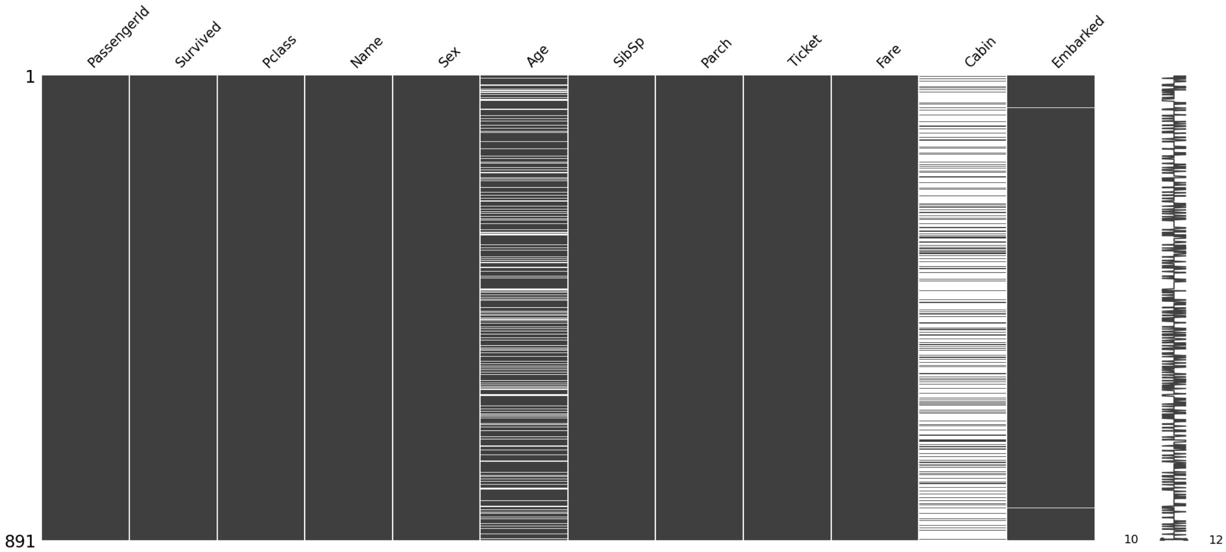
Embarked Analysis

In [ ]:
```
feature_distribution(df, feature_name="Embarked", title="Port of Embarkat
```

The embarked feature will be categorical and we have (S, C, and Q). Also have missing values

Handle with Missing Values

In [ ]:
```
msno.matrix(df)
plt.show()
```



In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Input missing value using Mean with Age and Class, based on data exploration

In [ ]:
```
df["Age"] = df["Age"].replace(np.NaN, df["Age"].mean())
df.isna().sum()
```

Out[ ]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

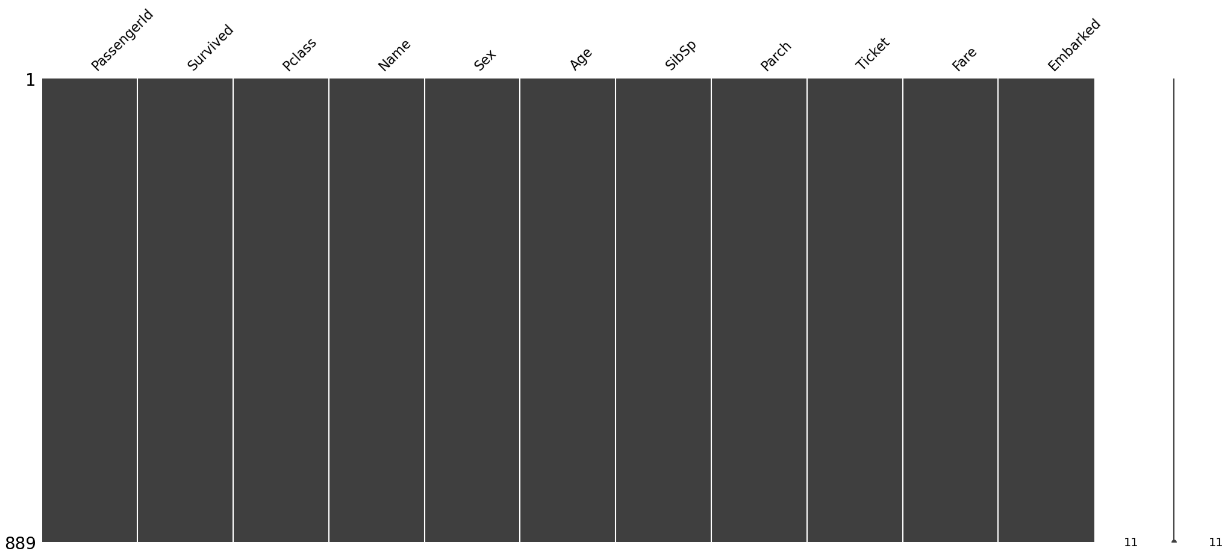
In [ ]:
```
df = df.drop(["Cabin"], axis=1)
```

Remove the two samples that have 2 missing value from feature Embarked

In [ ]:
```
df = df.dropna()
df.isna().sum()
```

Out[ ]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [ ]:
```
msno.matrix(df)
plt.show()
```



Feature Engineering

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

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0

Add Title feature

This feature will be create with base the name field

In [ ]:
```
def substring_in_string(big_string, substrings):
    for substring in substrings:
        if (substring in big_string):
            return substring
    print(big_string)
    return np.nan


def replace_title(x):
    title=x["Title"]
    if (title in ["Don", "Major", "Capt", "Jonkheer", "Rev", "Col"]):
        return "Mr"
    elif (title in ["Countess", "Mme"]):
        return "Mrs"
    elif (title in ["Mlle", "Ms"]):
        return "Miss"
    elif title == "Dr":
        if (x["Sex"] == "Male"):
            return "Mr"
        else:
            return "Mrs"
    else:
        return title
```

In [ ]:
```
title_list=['Mrs', 'Mr', 'Master', 'Miss', 'Major', 'Rev',
            'Dr', 'Ms', 'Mlle','Col', 'Capt', 'Mme', 'Countess',
            'Don', 'Jonkheer']
print(df["Name"].values[0])
substring_in_string(df["Name"].values[0], title_list)
```

```
Braund, Mr. Owen Harris
```

Out[ ]:
```
'Mr'
```

In [ ]:
```
title_list=['Mrs', 'Mr', 'Master', 'Miss', 'Major', 'Rev',
            'Dr', 'Ms', 'Mlle','Col', 'Capt', 'Mme', 'Countess',
            'Don', 'Jonkheer']
df["Title"] = df["Name"].map(lambda x: substring_in_string(x, title_list))
df.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2!
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2!
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9!
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1(
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0!

In [ ]:

```
df["Title"] = df.apply(replace_title, axis=1)
df.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2!
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2!
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9!
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1(
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0!

Get Ticket Number and Item from Ticket Feature

In [ ]:
```
def get_ticket_number(x):
    return x.split(' ')[-1]

def get_ticket_item(x):
    items = x.split(' ')
    if (len(items) == 1):
        return "NONE"
    return '_'.join(items[0:-1])

df["Ticket_Number"] = df["Ticket"].apply(get_ticket_number)
df["Ticket_Item"] = df["Ticket"].apply(get_ticket_item)
df.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2!
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2!
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0!

SibSp and Parch are the correlated features

In [ ]:
```
fig = px.density_contour(df, x="SibSp", y="Parch", color='Survived',
                         height=400, width=800)
fig.update_traces(contours_coloring="fill", contours_showlabels = True)
fig.show()
```

Using this contours we can see the feature Parch and SibSp are correlated and we can define this features as the **family size**

In [ ]:
```
df["Family Size"] = df["SibSp"] + df["Parch"]
```

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

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2!
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2!
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9:
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1(
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0!

Age times PClass Because are Numbers

In [ ]:
```
df["Age*Class"] = df["Age"]*df["Pclass"]
df.head(1)
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emba
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	

Fare Per Person - Proportion

In [ ]:
```
df["Fare_Per_Person"] = df["Fare"]/(df["Family Size"]+1)
df.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0

Remove Some Unsual Features

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   PassengerId      889 non-null    int64  
 1   Survived         889 non-null    int64  
 2   Pclass           889 non-null    int64  
 3   Name             889 non-null    object 
 4   Sex              889 non-null    object 
 5   Age              889 non-null    float64
 6   SibSp            889 non-null    int64  
 7   Parch            889 non-null    int64  
 8   Ticket           889 non-null    object 
 9   Fare             889 non-null    float64
 10  Embarked         889 non-null    object 
 11  Title            889 non-null    object 
 12  Ticket_Number    889 non-null    object 
 13  Ticket_Item      889 non-null    object 
 14  Family Size      889 non-null    int64  
 15  Age*Class        889 non-null    float64
 16  Fare_Per_Person  889 non-null    float64
dtypes: float64(4), int64(6), object(7)
memory usage: 125.0+ KB
```

In [ ]:
```
df = df.drop(["PassengerId", "Name",
    "SibSp", "Parch"], axis=1)
df.head()
```

Out[ ]:

	Survived	Pclass	Sex	Age	Ticket	Fare	Embarked	Title	Ticket_Number
0	0	3	male	22.0	A/5 21171	7.2500	S	Mr	21171
1	1	1	female	38.0	PC 17599	71.2833	C	Mrs	17599
2	1	3	female	26.0	STON/O2. 3101282	7.9250	S	Miss	3101282
3	1	1	female	35.0	113803	53.1000	S	Mrs	113803
4	0	3	male	35.0	373450	8.0500	S	Mr	373450

Preprocessing

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Survived         889 non-null    int64  
 1   Pclass           889 non-null    int64  
 2   Sex              889 non-null    object 
 3   Age              889 non-null    float64
 4   Ticket           889 non-null    object 
 5   Fare             889 non-null    float64
 6   Embarked         889 non-null    object 
 7   Title            889 non-null    object 
 8   Ticket_Number    889 non-null    object 
 9   Ticket_Item      889 non-null    object 
 10  Family Size      889 non-null    int64  
 11  Age*Class        889 non-null    float64
 12  Fare_Per_Person  889 non-null    float64
dtypes: float64(4), int64(3), object(6)
memory usage: 97.2+ KB
```

In [ ]:
```
categorical_df = df[["Pclass", "Sex", "Embarked", "Ticket", "Title", "Tic
numerical_df = df[["Age", "Fare", "Age*Class", "Family Size", "Fare_Per_P
y = df[["Survived"]]
```

Handle with Numerical Features

In [ ]:
```
numerical_df.describe(percentiles=[0.01, 0.05, 0.10, 0.25, 0.50,
                                   0.75, 0.90, 0.99])
```

Out[ ]:

	Age	Fare	Age*Class	Family Size	Fare_Per_Person
count	889.000000	889.000000	889.000000	889.000000	889.000000
mean	29.653446	32.096681	65.000488	0.906637	19.781204
std	12.968366	49.697504	32.972039	1.614703	35.767862
min	0.420000	0.000000	0.920000	0.000000	0.000000
1%	1.000000	0.000000	2.250000	0.000000	0.000000
5%	6.000000	7.225000	14.400000	0.000000	4.215000
10%	16.000000	7.550000	24.000000	0.000000	5.081933
25%	22.000000	7.895800	40.000000	0.000000	7.250000
50%	29.699118	14.454200	63.000000	0.000000	8.158300
75%	35.000000	31.000000	89.097353	1.000000	22.525000
90%	47.000000	77.287500	100.400000	3.000000	39.460000
99%	65.000000	249.303304	153.000000	7.000000	151.779500
max	80.000000	512.329200	222.000000	10.000000	512.329200

Looking at this percentiles we have to check if has any outlier to capping

In [ ]:

```
def outlier_cap(x):
    x = x.clip(lower=x.quantile(0.01))
    x = x.clip(upper=x.quantile(0.99))
    return x
```

In [ ]:

```
numerical_df = numerical_df.apply(lambda x: outlier_cap(x))
```

In [ ]:

```
numerical_df.describe(percentiles=[0.01, 0.05, 0.10, 0.25, 0.50,
                                   0.75, 0.90, 0.99])
```

Out[ ]:

	Age	Fare	Age*Class	Family Size	Fare_Per_Person
count	889.000000	889.000000	889.000000	889.000000	889.000000
mean	29.596450	31.118044	64.699206	0.883015	18.449413
std	12.781228	42.524308	31.901312	1.499187	24.323382
min	1.000000	0.000000	2.250000	0.000000	0.000000
1%	1.000000	0.000000	2.250000	0.000000	0.000000
5%	6.000000	7.225000	14.400000	0.000000	4.215000
10%	16.000000	7.550000	24.000000	0.000000	5.081933
25%	22.000000	7.895800	40.000000	0.000000	7.250000
50%	29.699118	14.454200	63.000000	0.000000	8.158300
75%	35.000000	31.000000	89.097353	1.000000	22.525000
90%	47.000000	77.287500	100.400000	3.000000	39.460000
99%	65.000000	247.734700	153.000000	7.000000	151.577540
max	65.000000	249.303304	153.000000	7.000000	151.779500

Feature Selection/Discretization - Numerical Features

VarianceThreshold will be remove all low-variance features. Can thus be used for unsupervised learning.

In [ ]:
```
vt = VarianceThreshold(threshold=0)
vt.fit_transform(numerical_df)

# get columns to keep and create new dataframe with those only
cols = vt.get_support(indices=True)
num_feat_non_var = numerical_df.iloc[:, cols]
```

KBinsDiscretizer will be discretizes features into `k` bins.

In [ ]:
```
kbdiscrete = KBinsDiscretizer(n_bins=5, encode="ordinal", strategy="quant
```

In [ ]:
```
num_binned = pd.DataFrame(kbdiscrete.fit_transform(num_feat_non_var), ind
                          columns=num_feat_non_var.columns).add_suffix('_
num_binned.head()
```

Out[ ]:

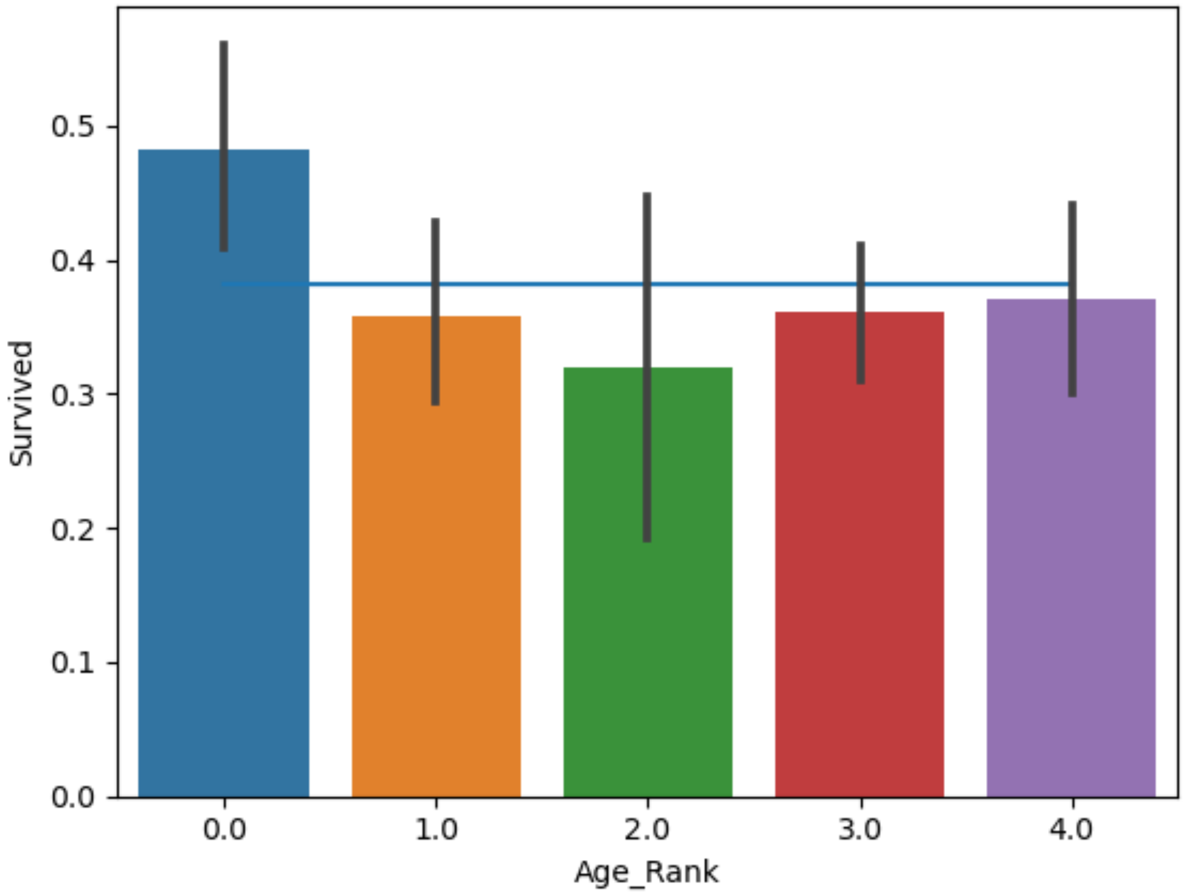
	Age_Rank	Fare_Rank	Age*Class_Rank	Family Size_Rank	Fare_Per_Person_Rank
0	1.0	0.0	2.0	0.0	0.0
1	4.0	4.0	1.0	0.0	4.0
2	1.0	1.0	3.0	0.0	2.0
3	3.0	4.0	1.0	0.0	4.0
4	3.0	1.0	4.0	0.0	2.0

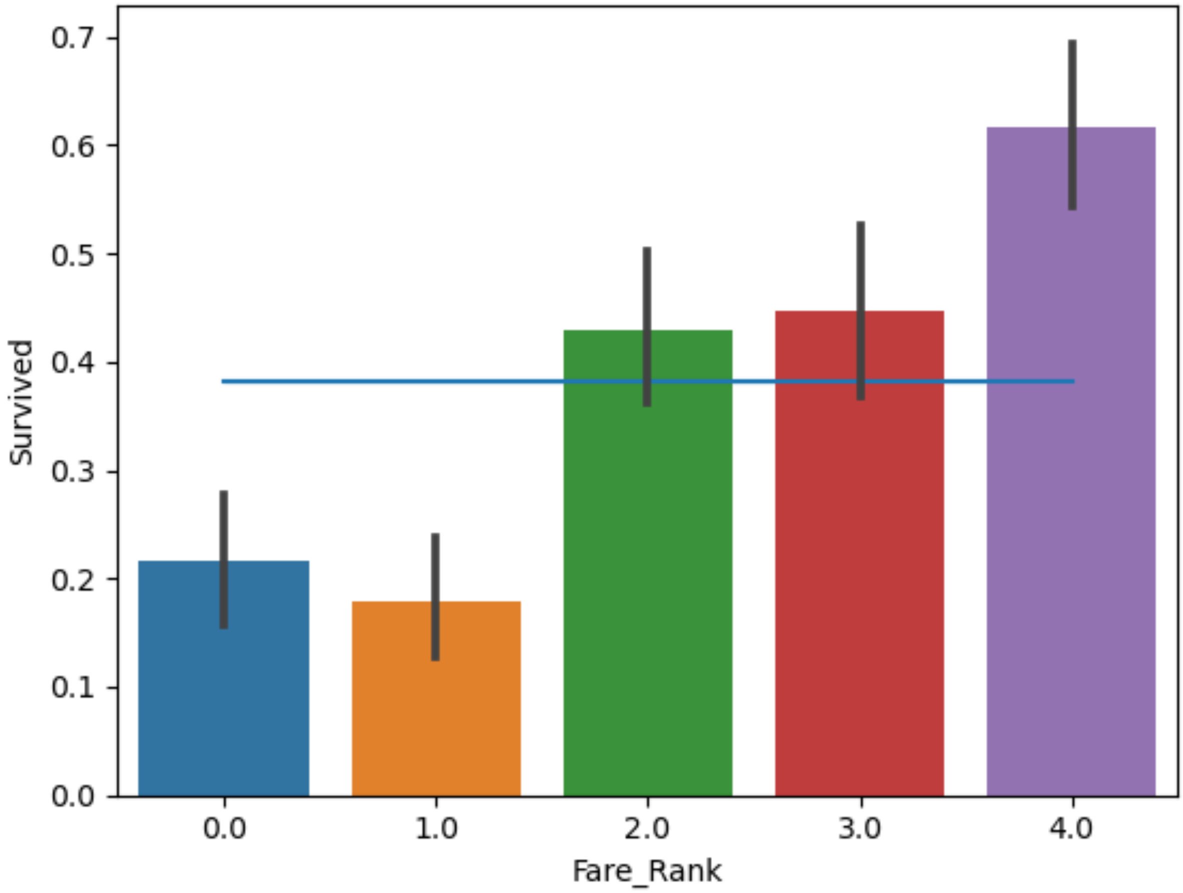
With this Rank features, we can check if the feature show a slop at all

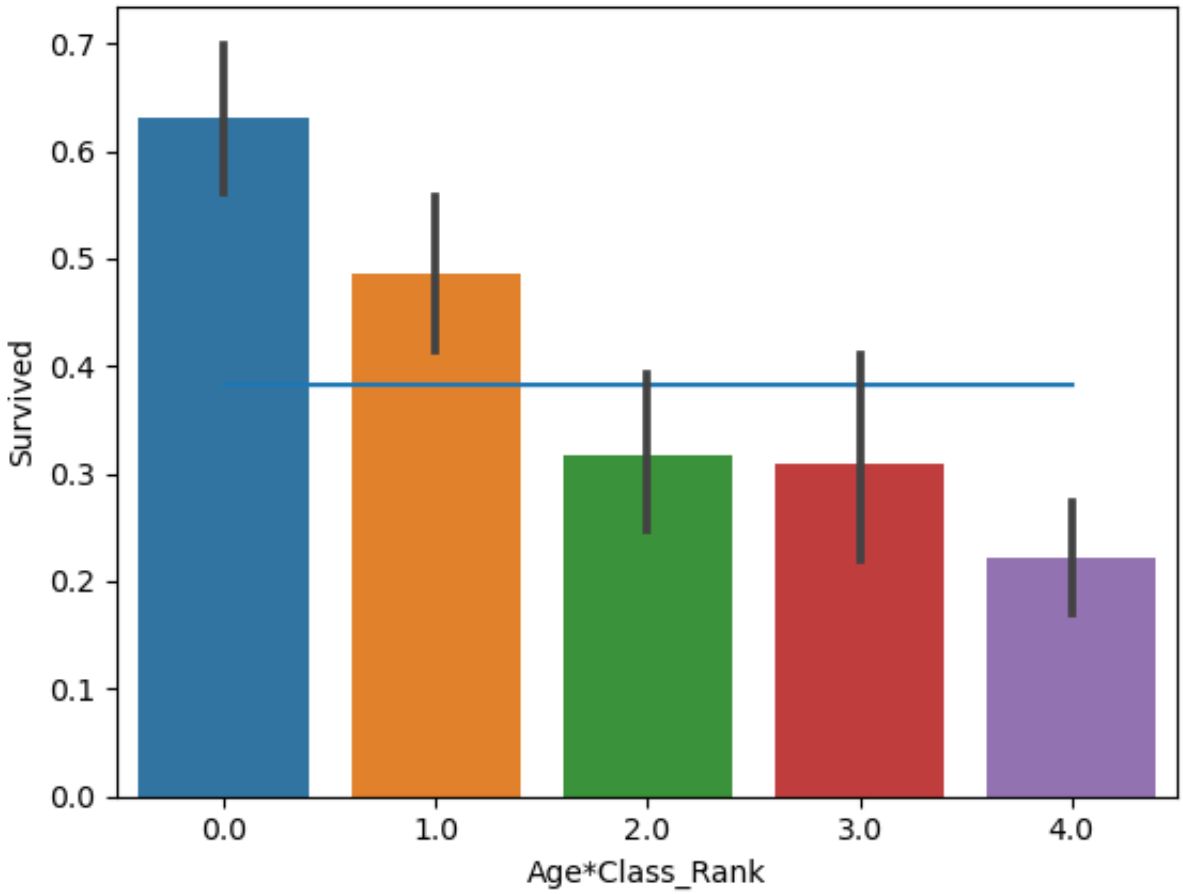
Note: a strong slope is indicate of the feature's ability to discriminate the event from non event making it a good predictor

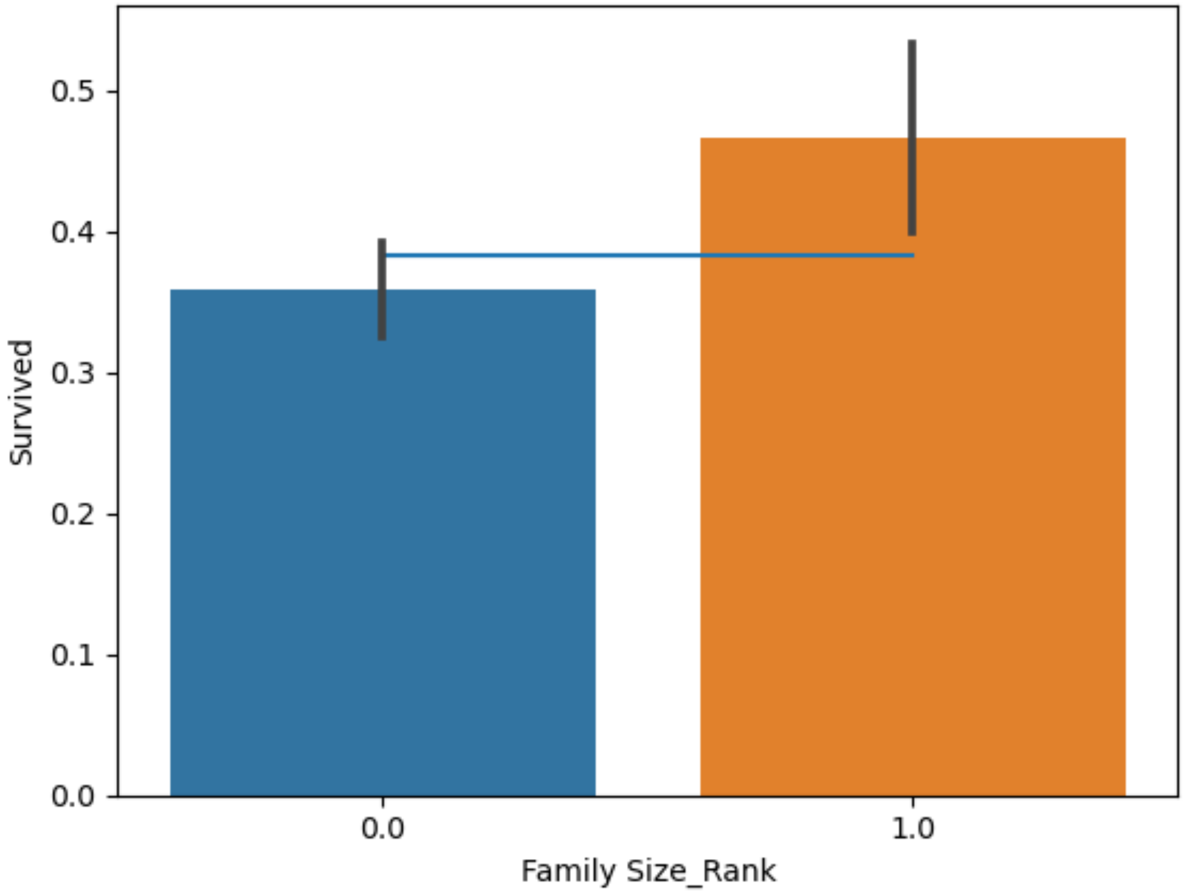
In [ ]:
```
def check_slope():
    X_bin_combined = pd.concat([y, num_binned], axis=1, join="inner")
    for col in num_binned.columns:
        sns.lineplot(x=col, y=X_bin_combined["Survived"].mean(), data=X_b
        sns.barplot(x=col, y="Survived", data=X_bin_combined, estimator=n
        plt.show()
```

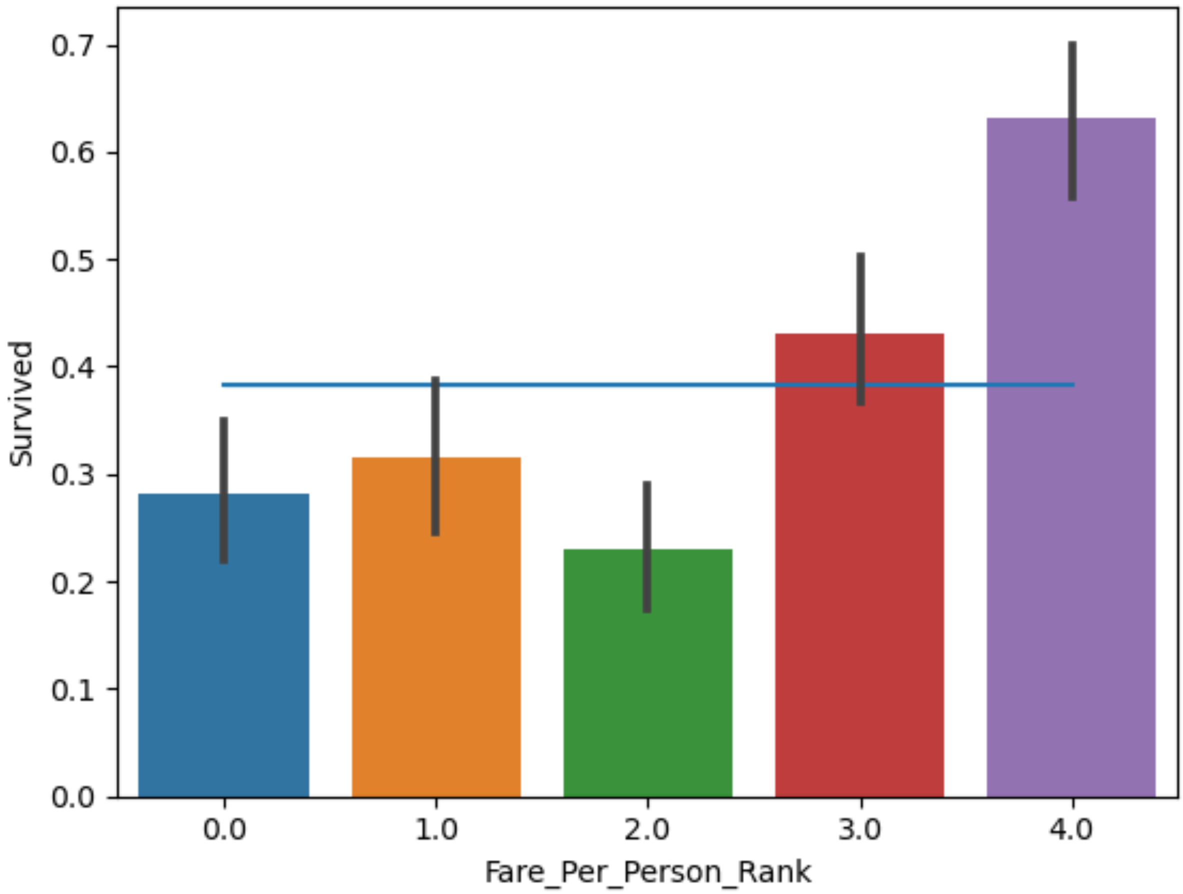
In [ ]:
```
check_slope()
```




0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Survived
0.0
1.0
2.0
3.0
4.0
Fare_Rank


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Survived
0.0
1.0
2.0
3.0
4.0
Age*Class_Rank





All feature has strong slope

In [ ]:
```
select_feature_num_df = num_feat_non_var.copy()
select_feature_num_df.head()
```

Out[ ]:

	Age	Fare	Age*Class	Family Size	Fare_Per_Person
0	22.0	7.2500	66.0	1.0	3.62500
1	38.0	71.2833	38.0	1.0	35.64165
2	26.0	7.9250	78.0	0.0	7.92500
3	35.0	53.1000	35.0	1.0	26.55000
4	35.0	8.0500	105.0	0.0	8.05000

In [ ]:
```
numerical = select_feature_num_df.columns.to_list()
numerical
```

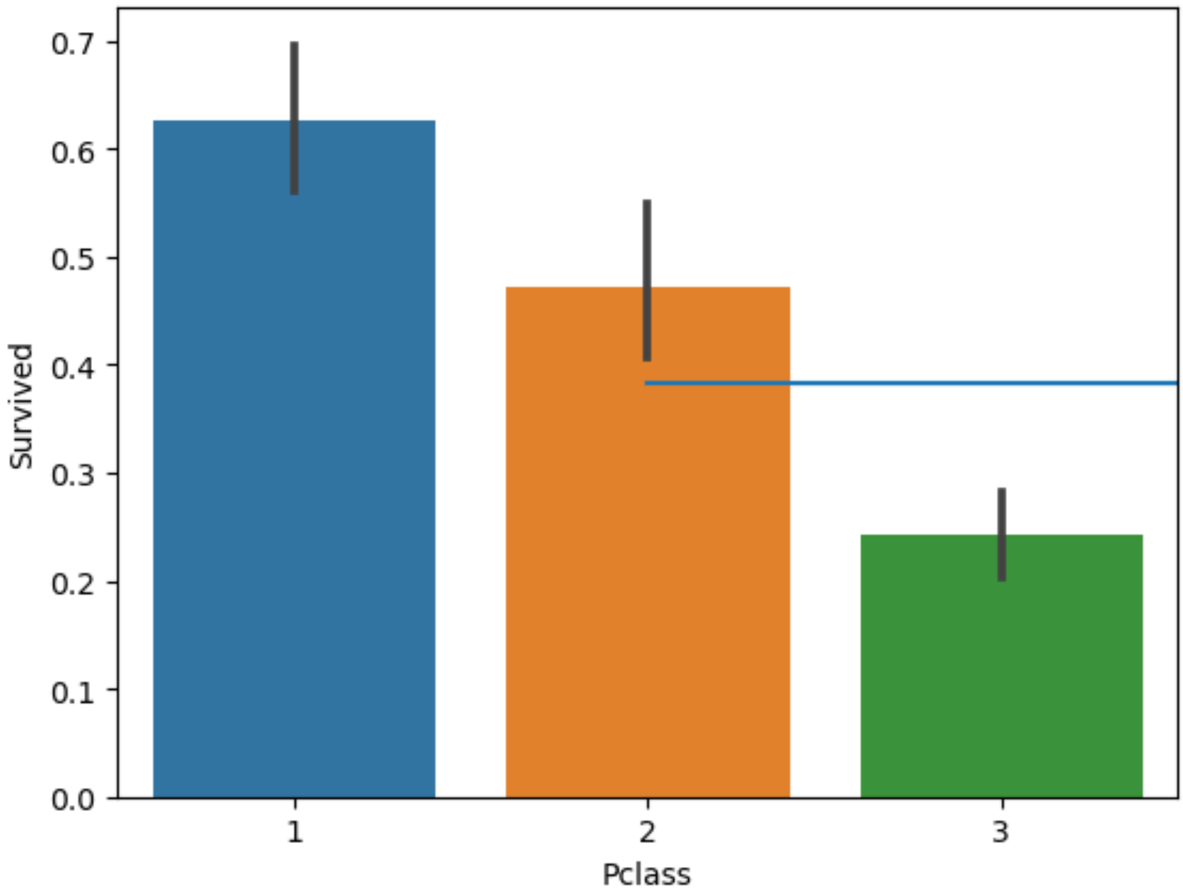
Out[ ]:
```
['Age', 'Fare', 'Age*Class', 'Family Size', 'Fare_Per_Person']
```

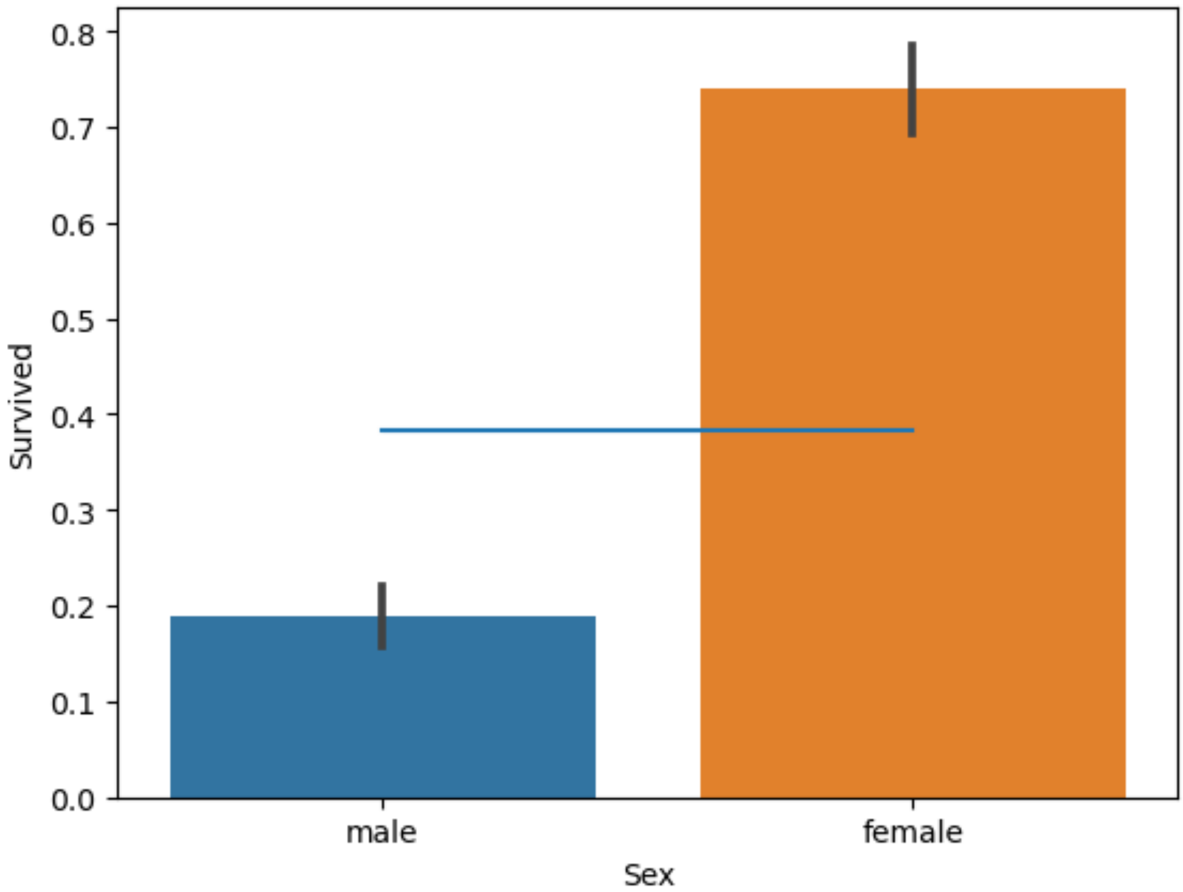
Handle with Categorical Features

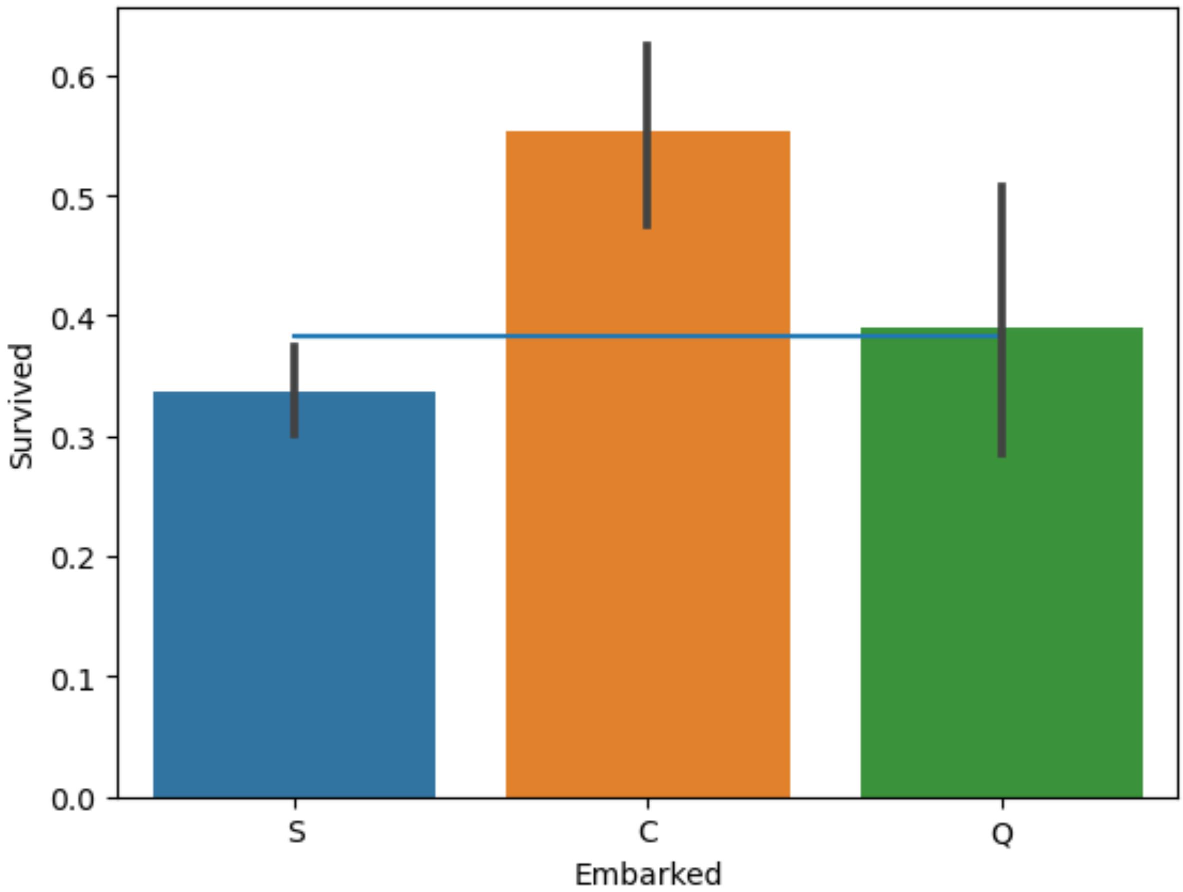
Feature Selection/Discretization

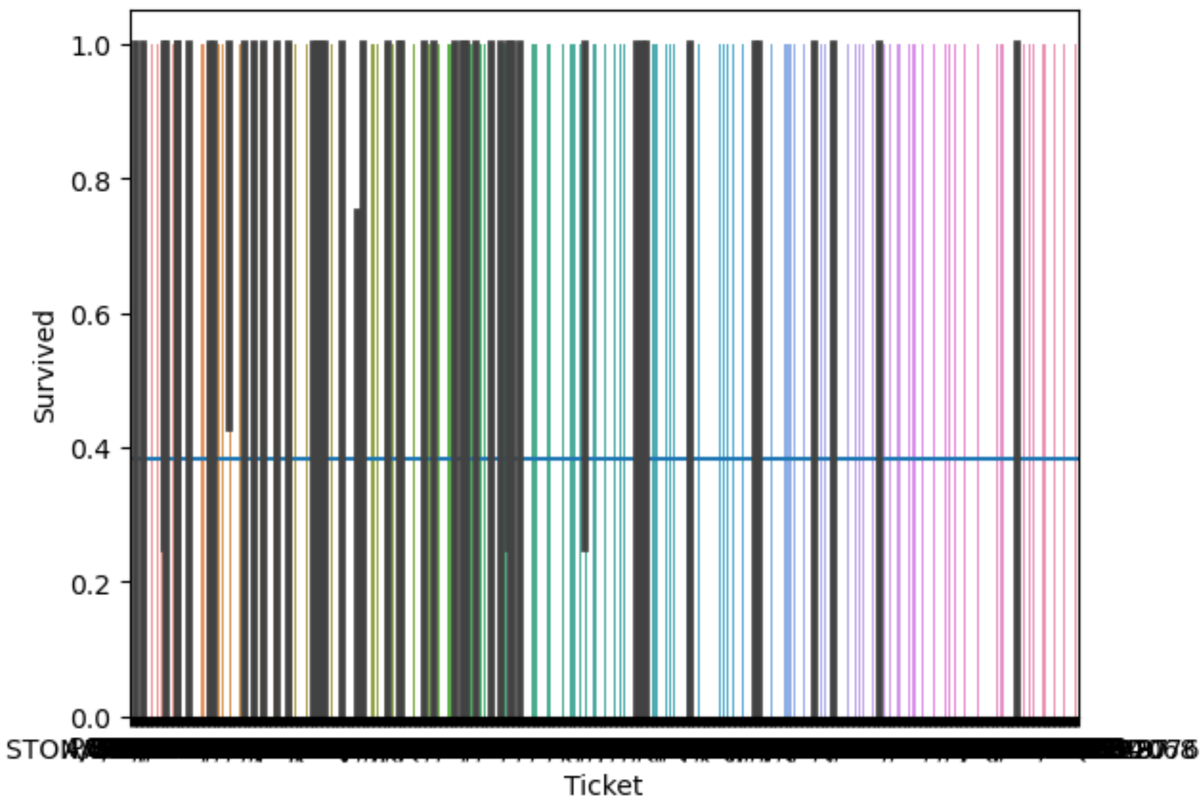
In [ ]:
```
def check_categorical_slope():
    X_categorical_merged = pd.concat([y, categorical_df], axis=1)

    for col in categorical_df.columns:
        sns.lineplot(x=col, y=X_categorical_merged["Survived"].mean(),
                     data=X_categorical_merged)
        sns.barplot(x=col, y="Survived", data=X_categorical_merged,
                    estimator=np.mean)
        plt.show()
```

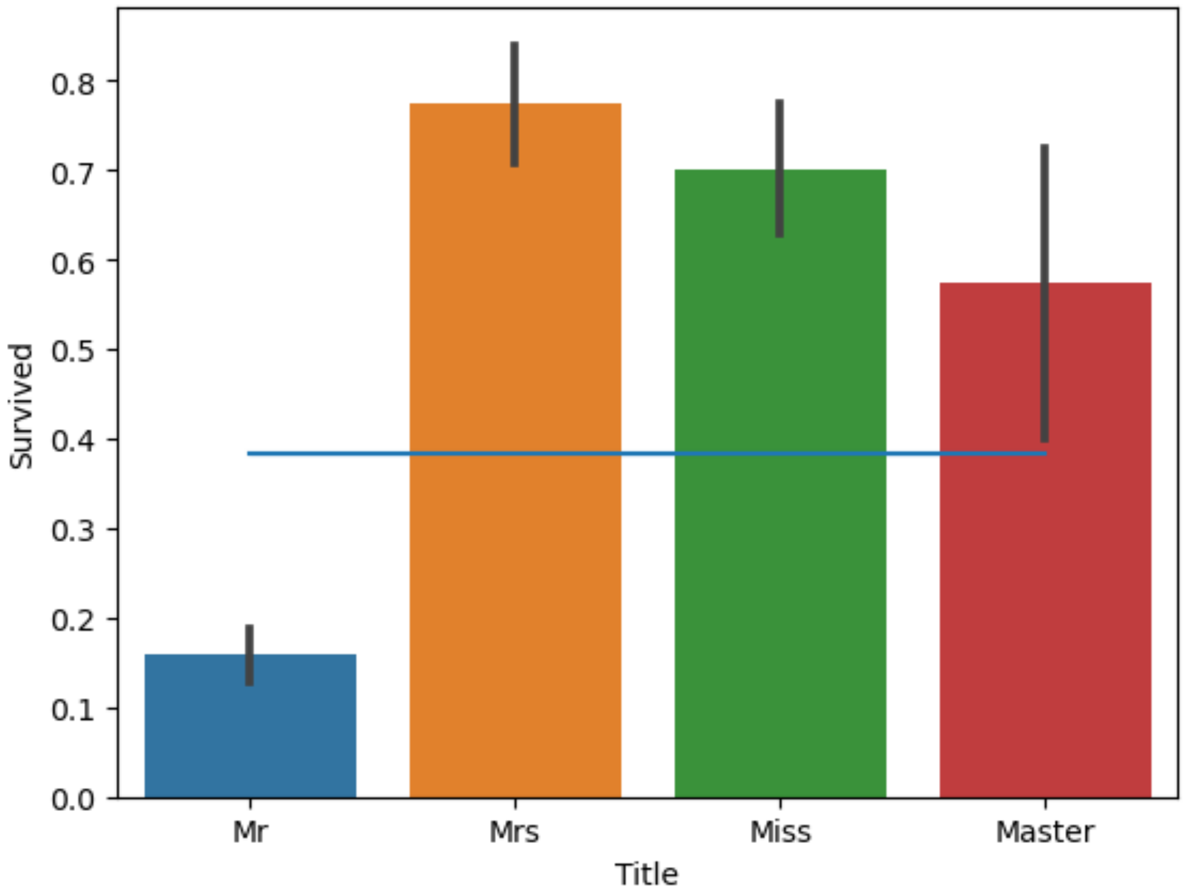
In [ ]:
```
check_categorical_slope()
```

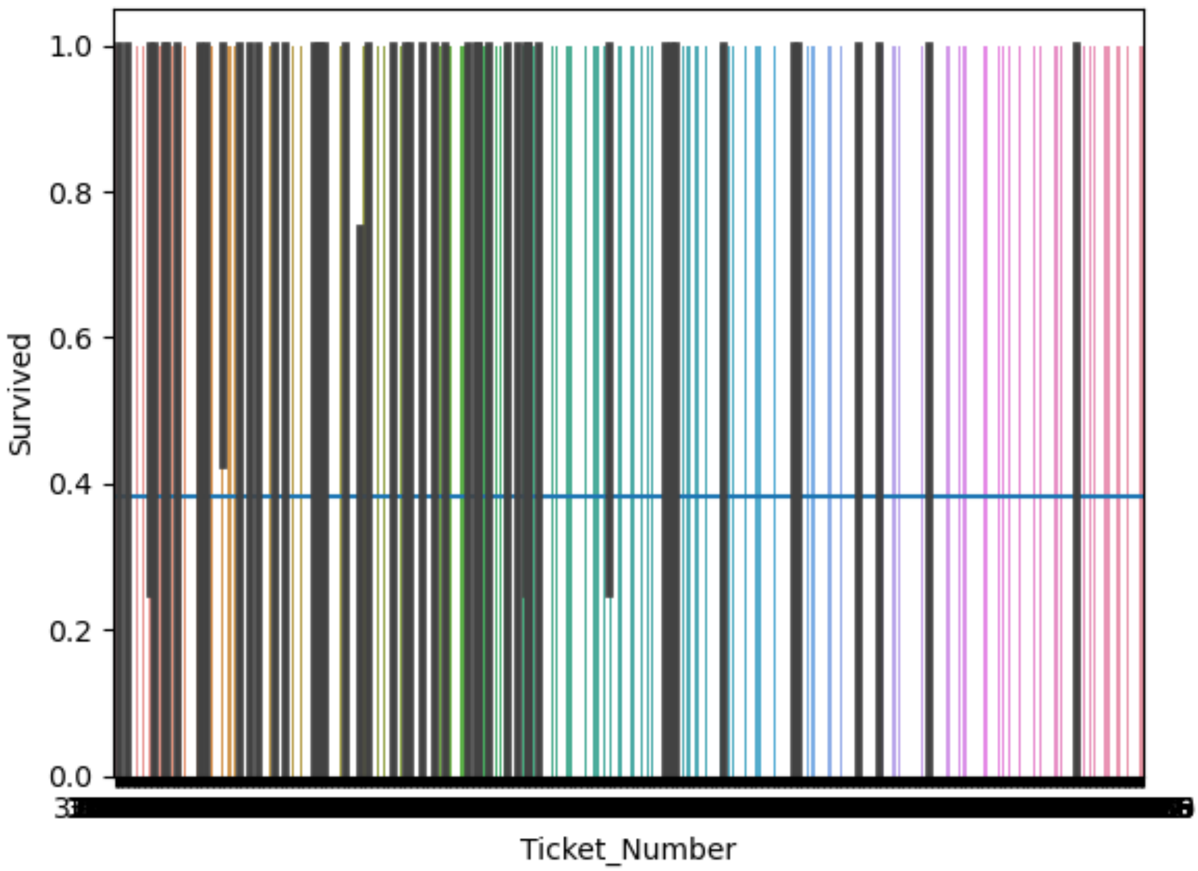

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Survived
1
2
3
Pclass

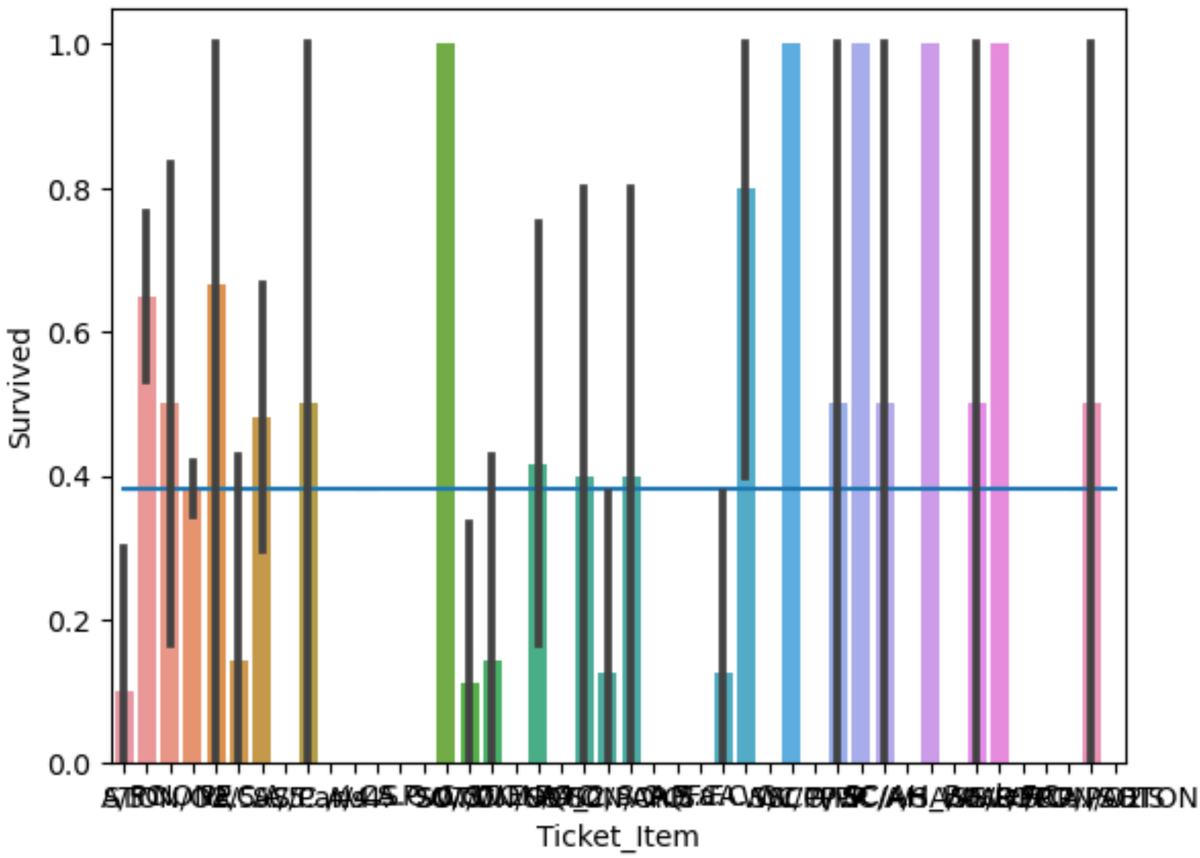

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Survived
male
female
Sex


0.6
0.5
0.4
0.3
0.2
0.1
0.0
Survived
S
C
Q
Embarked


1.0
0.8
0.6
0.4
0.2
0.0
Survived
Ticket


Survived
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Mr
Mrs
Miss
Master
Title


Survived
0.0
0.2
0.4
0.6
0.8
1.0
Ticket_Number



The Ticket feature have a weakly slope.

In [ ]:
```
select_feature_cat_df = categorical_df.copy()
# select_feature_cat_df = categorical_df.drop("Ticket", axis=1)

select_feature_cat_df.head()
```

Out[ ]:

	Pclass	Sex	Embarked	Ticket	Title	Ticket_Number	Ticket_Item
0	3	male	S	A/5 21171	Mr	21171	A/5
1	1	female	C	PC 17599	Mrs	17599	PC
2	3	female	S	STON/O2. 3101282	Miss	3101282	STON/O2.
3	1	female	S	113803	Mrs	113803	NONE
4	3	male	S	373450	Mr	373450	NONE

Dummies the Categorical Features

In [ ]:
```
select_feature_cat_df.head()
```

Out[ ]:

	Pclass	Sex	Embarked	Ticket	Title	Ticket_Number	Ticket_Item
0	3	male	S	A/5 21171	Mr	21171	A/5
1	1	female	C	PC 17599	Mrs	17599	PC
2	3	female	S	STON/O2. 3101282	Miss	3101282	STON/O2.
3	1	female	S	113803	Mrs	113803	NONE
4	3	male	S	373450	Mr	373450	NONE

In [ ]:
```
# lb = LabelEncoder() # for ordinal categorical features
ord = OrdinalEncoder(encoded_missing_value=-1, handle_unknown='use_encode
one_hot_enc = OneHotEncoder(sparse=False, handle_unknown="ignore") # for
```

In [ ]:
```
ordinal = ["Pclass", ]
nominal = ["Sex", "Embarked", "Title", "Ticket_Item", "Ticket_Number"]
```

In [ ]:
```
categorical_dum_df = select_feature_cat_df[ordinal].copy() # if I use Lab
# categorical_dum_df = pd.DataFrame()
categorical_dum_df.shape
```

Out[ ]: (889, 1)

In [ ]:
```
# for LabelEncoder()
# for ord_feat in ordinal:
    # categorical_dum_df[ord_feat] = lb.fit_transform(select_feature_cat_

# for ordinal encoder
categorical_dum_df[ordinal] =  ord.fit_transform(select_feature_cat_df[or

categorical_dum_df.head()
```

Out[ ]:

	Pclass
0	2.0
1	0.0
2	2.0
3	0.0
4	2.0

In [ ]:
```
categorical_dum_df.shape
```

Out[ ]: (889, 1)

In [ ]:
```
categorical_dum_df.tail()
```

Out[ ]:

	Pclass
886	1.0
887	0.0
888	2.0
889	0.0
890	2.0

In [ ]:
```
nominal_feat = one_hot_enc.fit_transform(select_feature_cat_df[nominal])
nominal_feat = pd.DataFrame(nominal_feat, columns=["nom_feat"+str(i+1).zf
nominal_feat.tail()
```

Out[ ]:

	nom_feat01	nom_feat02	nom_feat03	nom_feat04	nom_feat05	nom_feat06	non
884	0.0	1.0	0.0	0.0	1.0	0.0	
885	1.0	0.0	0.0	0.0	1.0	0.0	
886	1.0	0.0	0.0	0.0	1.0	0.0	
887	0.0	1.0	1.0	0.0	0.0	0.0	
888	0.0	1.0	0.0	1.0	0.0	0.0	

5 rows × 731 columns

In [ ]:
```
categorical_dum_df.shape, nominal_feat.shape
```

Out[ ]:
```
((889, 1), (889, 731))
```

In [ ]:
```
categorical_dum_df = pd.concat([categorical_dum_df.reset_index(drop=True)
```

In [ ]:
```
categorical_dum_df.head()
```

Out[ ]:

	Pclass	nom_feat01	nom_feat02	nom_feat03	nom_feat04	nom_feat05	nom_feat06
0	2.0	0.0	1.0	0.0	0.0	1.0	0.0
1	0.0	1.0	0.0	1.0	0.0	0.0	0.0
2	2.0	1.0	0.0	0.0	0.0	1.0	0.0
3	0.0	1.0	0.0	0.0	0.0	1.0	0.0
4	2.0	0.0	1.0	0.0	0.0	1.0	0.0

5 rows × 732 columns

Select K Best Categorical Features

This method removes all but the *k* highest scoring features

In [ ]:
```
skb = SelectKBest(score_func=chi2, k="all")
skb.fit_transform(categorical_dum_df, y)
columns= skb.get_support(indices=True)
print(f"Number of columns selected: {len(columns)}")
select_feature_cat_df = categorical_dum_df.iloc[:, columns]
```

```
Number of columns selected: 732
```

In other hand, all 732 categorical feature has highet score.

Put all Together

In [ ]:
```
X = pd.concat([select_feature_cat_df.reset_index(drop=True), select_featu
              axis=1, join="inner"
)
print(f"Number of samples: {X.shape[0]}, Number of features: {X.shape[1]}
```

```
Number of samples: 889, Number of features: 737
```

Train/Test Split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=42, shuffle=True
)

print(f"Amount of Train Samples: {len(X_train)}")
print(f"Amount of Test Samples: {len(X_test)}")
```

```
Amount of Train Samples: 800
Amount of Test Samples: 89
```

In [ ]:
```
print(f"Number of feature: {X_train.shape[1]}")
```

```
Number of feature: 737
```

Feature Scaling

In [ ]:
```
scaler = StandardScaler()
scaler.fit(X_train[numerical])
```

Out[ ]:
```
▾ StandardScaler
StandardScaler()
```

In [ ]:
```
X_train[numerical] = scaler.transform(X_train[numerical])
X_test[numerical] = scaler.transform(X_test[numerical])
```

Modeling

Logistic Regression

Build GridSearch Pipeline for Hyperparameter Tuning

In [ ]:
```
lr_pipeline = Pipeline([
    ("lr", LogisticRegression(penalty="l2", random_state=42))
])
lr_params = {
    "lr__C": [0.1, 1, 10, 100],
    "lr__solver": ["lbfgs", "newton-cg"],
    "lr__max_iter": [10, 50, 100]
}
lr_grid = GridSearchCV(lr_pipeline, lr_params, cv=10)
lr_grid.fit(X_train, y_train.values.reshape(-1,))
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: Pipeline
▸ LogisticRegression
```

In [ ]:
```
lr_grid.best_params_
```

Out[ ]:
```
{'lr__C': 10, 'lr__max_iter': 50, 'lr__solver': 'lbfgs'}
```

In [ ]:
```
lr_grid.best_score_
```

Out[ ]:
```
0.845
```

Using learning curve

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(estimator=Logisti

                                                        X=X_train, y=y_tr
                                                        train_sizes=np.li
                                                        cv=10, n_jobs=-1,
train_mean, train_std = np.mean(train_scores, axis=1), np.std(train_score
test_mean, test_std = np.mean(test_scores, axis=1), np.std(test_scores, a
```

In [ ]:
```
def plot_learning_curve(train_sizes, train_mean, train_std, test_mean, te
    """This function plot the learning curve to verify which scenario we

    Args:
        train_sizes (np.array): This array has all the size of train set
        train_mean (np.array): This array has the mean accuracy of train
        train_std (np.array): This array has the std accuracy of train se
```

```
        test_mean (np.array): this array has the mean accuracy of the tes
        test_std (np.array): This array has the std accuracy of the test

    """
    # Create the figure
    fig = go.Figure()

    # Add the training accuracy trace
    fig.add_trace(go.Scatter(
        x=train_sizes,
        y=train_mean,
        mode='markers+lines',
        marker=dict(color='blue', size=5),
        name='Training accuracy'
    ))

    # Add the fill between the training accuracy range
    fig.add_trace(go.Scatter(
        x=train_sizes + train_sizes[::-1],
        y=train_mean + train_std + train_mean[::-1] - train_mean - train_
        fill='tozerox',
        mode='none',
        fillcolor='rgba(0, 0, 255, 0.15)',
        line=dict(color='rgba(255, 255, 255, 0)'),
        name='Training accuracy range'
    ))

    # Add the validation accuracy trace
    fig.add_trace(go.Scatter(
        x=train_sizes,
        y=test_mean,
        mode='markers+lines',
        marker=dict(color='green', size=5, symbol='square-open-dot'),
        line=dict(dash='dash'),
        name='Validation accuracy'
    ))

    # Add the fill between the validation accuracy range
    fig.add_trace(go.Scatter(
        x=train_sizes + train_sizes[::-1],
        y=test_mean + test_std + test_mean[::-1] - test_mean - test_std[:
        fill='tozerox',
        mode='none',
        fillcolor='rgba(0, 128, 0, 0.15)',
        line=dict(color='rgba(255, 255, 255, 0)'),
        name='Validation accuracy range'
    ))

    # Set layout properties
    fig.update_layout(
        title='Accuracy vs. Number of training examples',
        xaxis_title='Number of training examples',
        yaxis_title='Accuracy',
        legend=dict(x=1.0, y=0.1),
        yaxis=dict(range=[0.4, 1.03]),
        template='plotly_white'
    )
```

```
    fig.show()
```

In [ ]:
```
plot_learning_curve(train_sizes, train_mean, train_std, test_mean, test_s
```

Analyze the learning curve for Logistic Regression algorithm we have a great scenario with a litte of overfitting but not great.

Feature Importance

In [ ]:
```
lr = LogisticRegression(C=1, random_state=42, solver="lbfgs", max_iter=10
lr.fit(X_train, y_train.values.reshape(-1, ))
```

Out[ ]:
```
▾          LogisticRegression
LogisticRegression(C=1, random_state=42)
```

In [ ]:
```
coeff_df = pd.DataFrame(X_train.columns)
coeff_df.columns=["features"]
coeff_df["Coefficient Estimate"] = pd.Series(lr.coef_[0])
coeff_df.sort_values(["Coefficient Estimate"],ascending=False)
```

Out[ ]:

	features	Coefficient Estimate
147	nom_feat147	2.324091
6	nom_feat06	1.622388
48	nom_feat48	1.566120
513	nom_feat513	1.122984
628	nom_feat628	1.119207
...	...	...
668	nom_feat668	-0.933015
327	nom_feat327	-0.945405
0	Pclass	-1.102129
2	nom_feat02	-1.103159
8	nom_feat08	-1.290875

737 rows × 2 columns

Evaluate the performance of Test Set

In [ ]:
```
lr_pred = lr_grid.predict(X_test)
acc = accuracy_score(y_test.values.reshape(-1, ), lr_pred)
f1 = f1_score(y_test.values.reshape(-1, ), lr_pred)
print(f"Accuracy: {round(acc,4)}, F1-score: {round(f1, 4)}")
```

```
Accuracy: 0.8652, F1-score: 0.8125
```

In [ ]:
```
def plot_roc_curve():
    # Create the ROC curve trace
    roc_trace = go.Scatter(
        x=fpr,
        y=tpr,
        mode='lines',
        line=dict(color='blue', width=2),
        name=f'ROC curve (AUC = {auc_score:.2f})'
    )

    # Create the diagonal trace (the gray dashed line)
    diagonal_trace = go.Scatter(
        x=[0, 1],
        y=[0, 1],
        mode='lines',
        line=dict(color='gray', dash='dash'),
        showlegend=False
    )

    # Create the layout
    layout = go.Layout(
        title='Receiver Operating Characteristic (ROC) Curve',
        xaxis=dict(title='False Positive Rate'),
        yaxis=dict(title='True Positive Rate'),
        width=800,
        height=600,
        legend=dict(x=0.02, y=0.98, bordercolor='Black', borderwidth=1)
    )

    # Create the figure and add the traces
    fig = go.Figure(data=[roc_trace, diagonal_trace],
                    layout=layout
    )

    # Show the plot
    fig.show()

y_pred_prob = lr_grid.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test.values.reshape(-1, ), y_pred_prob
auc_score = roc_auc_score(y_test.values.reshape(-1,), y_pred_prob)
plot_roc_curve()
```

Support Vector Machine - SVM

Build GridSearch Pipeline for Hyperparameter Tuning

In [ ]:
```
svm_pipeline = Pipeline([
    ("svm", SVC(random_state=42))
])
svm_params = {
    "svm__C": [0.1, 1, 10, 100, 500],
    "svm__gamma": [0.01, 0.01, 0.001],
    "svm__max_iter": [100, 500, 1000]
}
svm_grid = GridSearchCV(svm_pipeline, svm_params, cv=10)
svm_grid.fit(X_train, y_train.values.reshape(-1, ))
```

Out[ ]:

▸ **GridSearchCV**

▸ **estimator: Pipeline**

▸ SVC

Using learning curve

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(estimator=SVC(ran
                                                                    C=5
                                                        X=X_train, y=y_tr
                                                        train_sizes=np.li
                                                        cv=10, n_jobs=-1)
train_mean, train_std = np.mean(train_scores, axis=1), np.std(train_score
test_mean, test_std = np.mean(test_scores, axis=1), np.std(test_scores, a
```

In [ ]:
```
plot_learning_curve(train_sizes, train_mean, train_std, test_mean, test_s
```

Look to the learning curve in SVM algorithm we have scenarion with a little of overfitting but better than Logistic Regression.

In [ ]:
```
svm_grid.best_params_
```

Out[ ]:
```
{'svm__C': 500, 'svm__gamma': 0.001, 'svm__max_iter': 500}
```

In [ ]:
```
svm_grid.best_score_
```

Out[ ]:
```
0.8462500000000001
```

Evaluate the performance of Test Set

In [ ]:
```
svm_pred = svm_grid.predict(X_test)
acc = accuracy_score(y_test.values.reshape(-1, ), svm_pred)
f1 = f1_score(y_test.values.reshape(-1, ), svm_pred)
print(f"Accuracy: {round(acc,4)}, F1-score: {round(f1, 4)}")
```

```
Accuracy: 0.8539, F1-score: 0.7869
```

In [ ]:
```
y_pred_prob = svm_grid.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test.values.reshape(-1, ), y_pred_prob
auc_score = roc_auc_score(y_test.values.reshape(-1,), y_pred_prob)
plot_roc_curve()
```

Random Forest

Build GridSearchCV Pipeline for Hyperparameter Tuning

In [ ]:
```
rf_pipeline = Pipeline([
    ("rf", RandomForestClassifier(random_state=42))
])
rf_params = {
    "rf__n_estimators": [100, 200, 300],
    "rf__max_depth": [None, 5, 10]
}
rf_grid = GridSearchCV(rf_pipeline, rf_params, cv=10)
rf_grid.fit(X_train, y_train.values.reshape(-1,))
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: Pipeline
▸ RandomForestClassifier
```

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(estimator=RandomF
                                                                      max
                                                        X=X_train, y=y_tr
                                                        train_sizes=np.li
                                                        cv=10, n_jobs=-1)
train_mean, train_std = np.mean(train_scores, axis=1), np.std(train_score
test_mean, test_std = np.mean(test_scores, axis=1), np.std(test_scores, a
```

In [ ]:
```
plot_learning_curve(train_sizes, train_mean, train_std, test_mean, test_s
```

We have a scenarion where the train set performance don't change (still in 1.0) and the train accuracy is close to 0.85 in other word we have a overfitting scenario.

In [ ]:
```
rf_grid.best_params_
```

Out[ ]:
```
{'rf__max_depth': None, 'rf__n_estimators': 100}
```

In [ ]:
```
rf_grid.best_score_
```

Out[ ]:
```
0.8450000000000001
```

Feature Importance

In [ ]:
```
rf = RandomForestClassifier(max_depth=None, n_estimators=100)
rf.fit(X_train, y_train.values.reshape(-1, ))
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [ ]:
```
def plot_feature_importance(feature_importance):
    fig = go.Figure()

    fig.add_trace(go.Bar(x=feature_importance["feature"],
                         y=feature_importance["importance"]))

    # add annotations
    impo = feature_importance["importance"]

    for i, imp in enumerate(impo):
        fig.add_annotation(
            x=feature_importance["feature"][i],
            y=imp-0.8,
            text=f"{imp:.2f}%",
            showarrow=False,
            font=dict(color="white", size=12)
        )

    fig.update_layout(
        title="Feature Importance",
        xaxis_title="Features",
        yaxis_title="Importance (%)",
        width=900,
        height=500,
    )

    fig.show()
```

In [ ]:
```
# features_importances = pd.DataFrame({"importance": rf.feature_importanc
#                                     index=[f"PC_{str(i+1)}" for i in ra
#                                     columns=["importance"]).sort_values
# features_importances = features_importances.reset_index()
# features_importances = features_importances.rename(columns={"index": "f
# display(features_importances)

# plot_feature_importance(features_importances)
```

Evaluate the performance of Test Set

In [ ]:
```
rf_pred = rf_grid.predict(X_test)
acc = accuracy_score(y_test.values.reshape(-1, ), rf_pred)
f1 = f1_score(y_test.values.reshape(-1, ), rf_pred)
print(f"Accuracy: {round(acc,4)}, F1-score: {round(f1, 4)}")
```

```
Accuracy: 0.8315, F1-score: 0.7692
```

In [ ]:
```
y_pred_prob = rf_grid.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test.values.reshape(-1, ), y_pred_prob
auc_score = roc_auc_score(y_test.values.reshape(-1,), y_pred_prob)
plot_roc_curve()
```

AdaBoost

Build GridSearch Pipeline for Hyperparameter Tuning

In [ ]:
```
adaboost_pipeline = Pipeline([
    ("adaboost", AdaBoostClassifier())
])
adaboost_params = {
    "adaboost__n_estimators": [50, 100, 200],
    "adaboost__learning_rate": [0.1, 1, 10]
}
adaboost_grid = GridSearchCV(adaboost_pipeline, adaboost_params, cv=10)
adaboost_grid.fit(X_train, y_train.values.reshape(-1,))
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: Pipeline
▸ AdaBoostClassifier
```

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(estimator=AdaBoos
                                                                      lea
                                                        X=X_train, y=y_tr
                                                        train_sizes=np.li
                                                        cv=10, n_jobs=-1)
train_mean, train_std = np.mean(train_scores, axis=1), np.std(train_score
test_mean, test_std = np.mean(test_scores, axis=1), np.std(test_scores, a
```

In [ ]:
```
plot_learning_curve(train_sizes, train_mean, train_std, test_mean, test_s
```

We have a overfitting scenario.

In [ ]:
```
adaboost_grid.best_params_
```

Out[ ]:
```
{'adaboost__learning_rate': 1, 'adaboost__n_estimators': 200}
```

In [ ]:
```
adaboost_grid.best_score_
```

Out[ ]:
```
0.8287500000000001
```

Evaluate the performance of Test Set

In [ ]:
```
adaboost_pred = adaboost_grid.predict(X_test)
acc = accuracy_score(y_test.values.reshape(-1, ), adaboost_pred)
f1 = f1_score(y_test.values.reshape(-1, ), adaboost_pred)
print(f"Accuracy: {round(acc,4)}, F1-score: {round(f1, 4)}")
```

```
Accuracy: 0.8427, F1-score: 0.7879
```

In [ ]:
```
y_pred_prob = adaboost_grid.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test.values.reshape(-1, ), y_pred_prob
auc_score = roc_auc_score(y_test.values.reshape(-1,), y_pred_prob)
plot_roc_curve()
```