In [ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('HWs/HW3/Files/homework-3/data/titanic.csv')
```

In [ ]:
```python
data.head()
```

Out[ ]:

	passenger_id	survived	pclass	name	sex	age	sib_sp	parch	ticket	fare	cabin	emba
0	1	No	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	Yes	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	Yes	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	Yes	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	No	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [ ]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   passenger_id  891 non-null    int64  
 1   survived      891 non-null    object 
 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   sib_sp        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(4), object(6)
memory usage: 83.7+ KB
```

In [ ]:
```python
data['survived'].value_counts()
```

Out[ ]:
```
survived
No     549
Yes    342
Name: count, dtype: int64
```

Question 1

In [ ]:
```
from sklearn.model_selection import train_test_split
np.random.seed(10)

X = data.drop('survived', axis=1)
Y = data['survived']

x_train, x_test, y_train, y_test = train_test_split(X,Y, train_size=.7, stratify=Y)
```

In [ ]:
```
print(f"x_train size: {len(x_train)}, y_train size: {len(y_train)}")
print(f"x_test size: {len(x_test)}, y_test size: {len(y_test)}")
```

```
x_train size: 623, y_train size: 623
x_test size: 268, y_test size: 268
```

We can notice below that cabin contains many non-null values, age has over 100, and embarked has 1.

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 623 entries, 155 to 253
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   passenger_id  623 non-null    int64  
 1   pclass        623 non-null    int64  
 2   name          623 non-null    object 
 3   sex           623 non-null    object 
 4   age           509 non-null    float64
 5   sib_sp        623 non-null    int64  
 6   parch         623 non-null    int64  
 7   ticket        623 non-null    object 
 8   fare          623 non-null    float64
 9   cabin         135 non-null    object 
 10  embarked      622 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 58.4+ KB
```

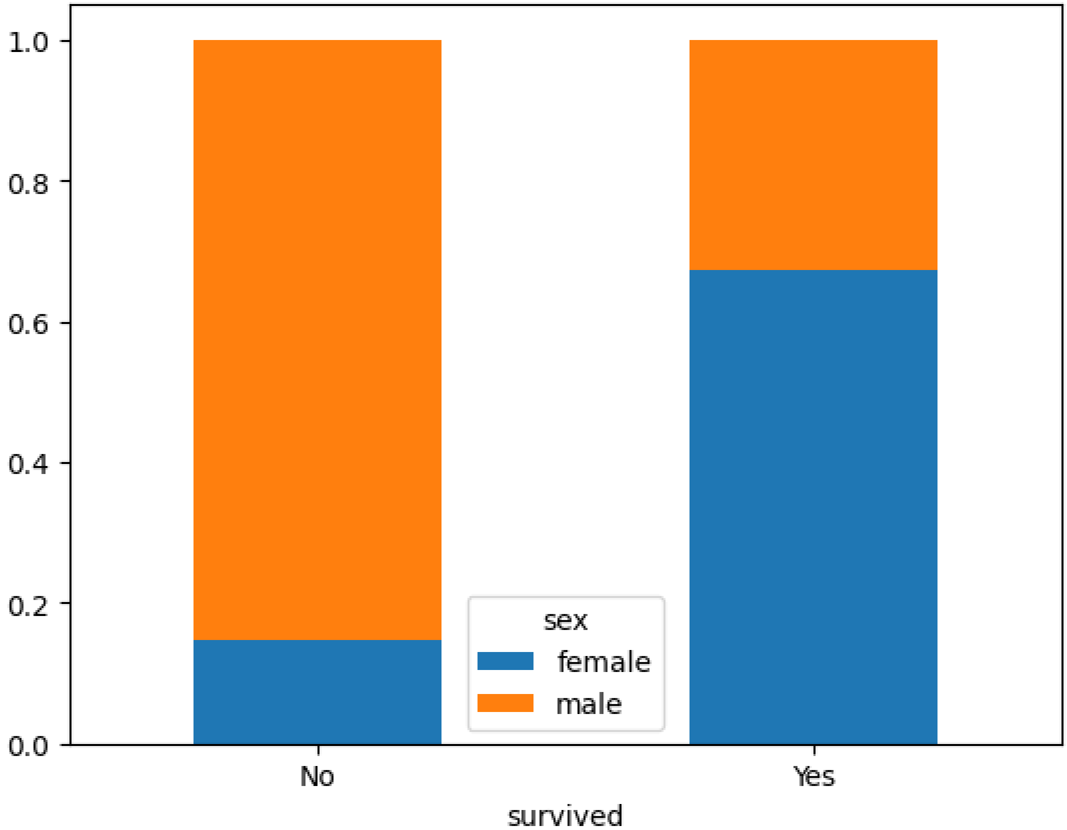
It is a good idea to use stratified sampling for this data since over 60% of passengers don't survive, potentially leading to a class imbalance when the data is split.

Question 2

Yes, it seems sex will be a good predictor of survival outcome, since a much larger proportion of women survived than men.

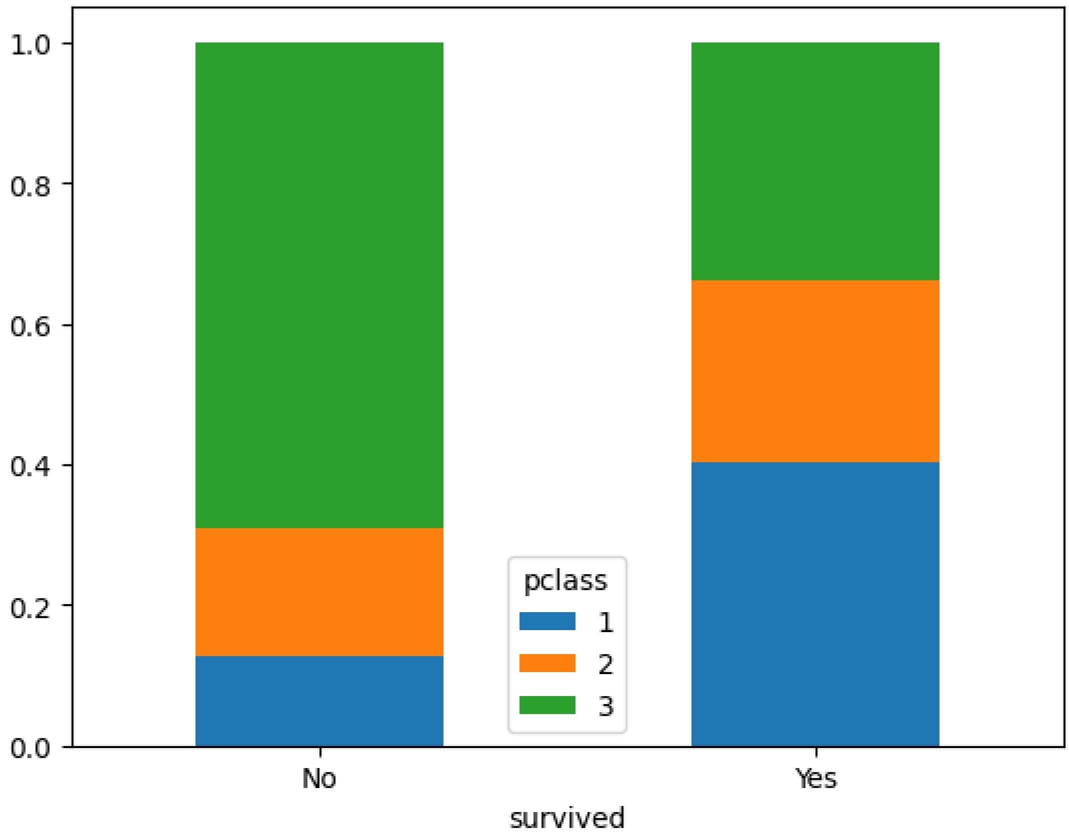
In [ ]:
```
pd.crosstab(index=y_train,
            columns = x_train['sex'],
            normalize='index').plot(kind='bar', stacked=True)
```

```
plt.xticks(rotation=0)
plt.show()
```



Yes, it seems that passenger class will be a good predictor of survival outcome, since a large portion of those in classes 1 and 2 survived, while very few in 3rd class survived.

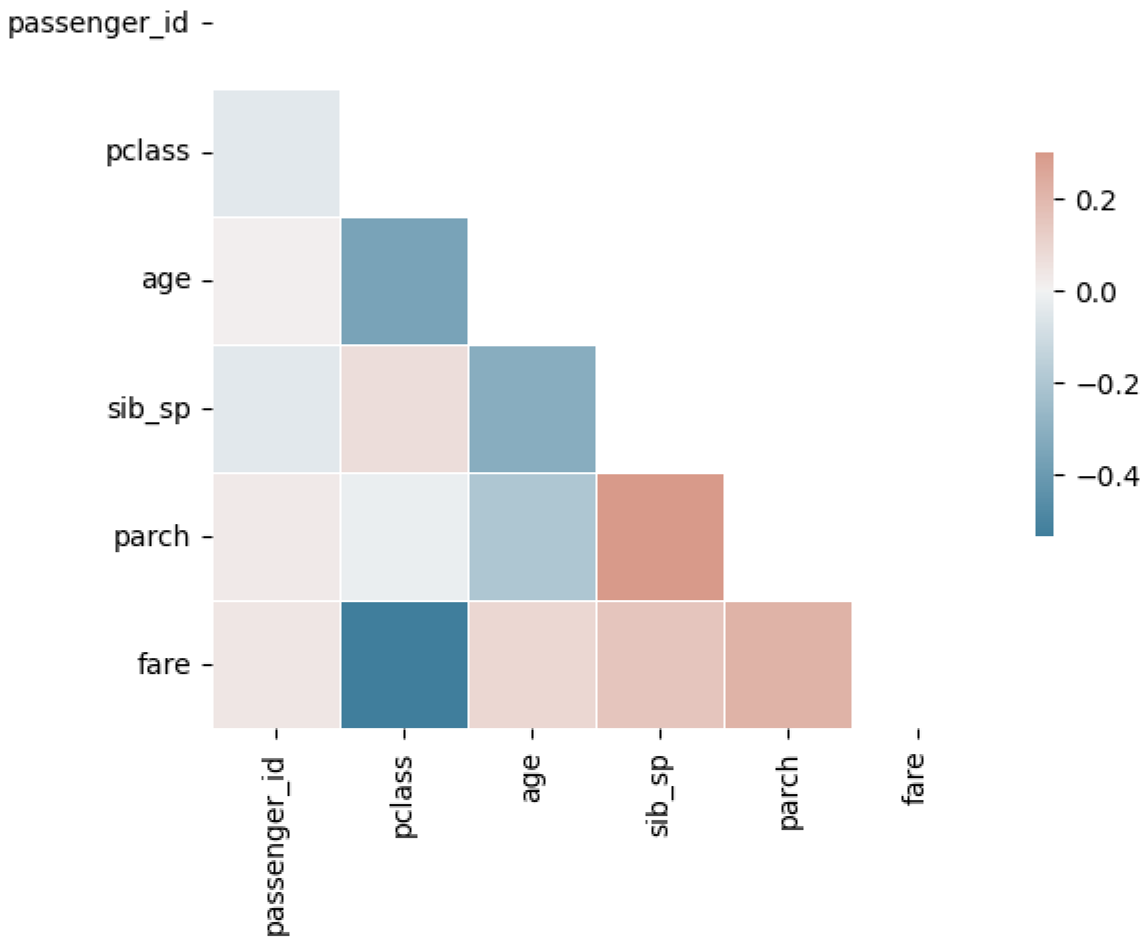
In [ ]:
```
pd.crosstab(index=y_train,
            columns = x_train['pclass'],
            normalize='index').plot(kind='bar', stacked=True)
plt.xticks(rotation=0)
plt.show()
```



It is useful to use a percent stacked bar chart because of the class imbalance survived and not survived. It also makes the message more clear to a viewer so that they don't have to do any mental calculations.

Question 3

In [ ]:
```
# Source: https://seaborn.pydata.org/examples/many_pairwise_correlations.html

corr = x_train._get_numeric_data().corr()
mask = np.triu(np.ones_like(corr, dtype=bool))

f, ax = plt.subplots(figsize=(7, 5))

cmap = sns.diverging_palette(230, 20, as_cmap=True)

sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

plt.show()
```



First, we can see a strong negative correlation between fare and pclass, indicating as expected that a 'lower' class (1 or 2) would be more expensive. Additionally, we also see a moderate, positive correlation between parch and sib_sp, which makes sense given that some parts of a family traveling together makes it likely that others would too. Lastly, there is a noticable, negative correlation between pclass and age, meaning that older pepole would more likely to be in a 'lower' class.

Question 4

In [ ]:
```
# Remaining features are selected: ticket class, sex, age, number of siblings or spouses
# number of parents or children abbord, and passenger fare

x_train.drop(['passenger_id', 'name', 'ticket', 'cabin', 'embarked'], axis=1, inplace=Tr
x_test.drop(['passenger_id', 'name', 'ticket', 'cabin', 'embarked'], axis=1, inplace=Tru
```

In [ ]:
```
# Fill in NA values in 'age'
x_train['age'] = x_train['age'].interpolate(method='linear')
x_test['age'] = x_test['age'].interpolate(method='linear')
```

In [ ]:
```
# Dummy code categorical variables
x_train = pd.get_dummies(x_train, columns=['pclass', 'sex'], dtype=int, drop_first=True)
x_test = pd.get_dummies(x_test, columns=['pclass', 'sex'], dtype=int, drop_first=True)

x_train.head()
```

Out[ ]:

	age	sib_sp	parch	fare	pclass_2	pclass_3	sex_male
155	51.0	0	1	61.3792	0	0	1
14	14.0	0	0	7.8542	0	1	0
421	21.0	0	0	7.7333	0	1	1
432	42.0	1	0	26.0000	1	0	0
484	25.0	1	0	91.0792	0	0	1

In [ ]:

```
# Add interaction terms between sex and fare
x_train['sex_male*fare'] = x_train['sex_male'] * x_train['fare']
x_test['sex_male*fare'] = x_test['sex_male'] * x_test['fare']

# Add interaction terms between age and fare
x_train['age*fare'] = x_train['age'] * x_train['fare']
x_test['age*fare'] = x_test['age'] * x_test['fare']
```

Question 5

In [ ]:

```
from sklearn.linear_model import LogisticRegression

# Fit the logistic regression model. Solver chosen to ensure convergence
LR_model = LogisticRegression(solver='newton-cg').fit(x_train, y_train)
```

Question 6

In [ ]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscrimin

# Fit an LDA model
LDA_model = LinearDiscriminantAnalysis().fit(x_train, y_train)
```

Question 7

In [ ]:

```
# Fit a QDA Model
QDA_model = QuadraticDiscriminantAnalysis().fit(x_train, y_train)
```

Question 8

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

# Fit KNN with arbitrary K
KNN_model = KNeighborsClassifier(n_neighbors=10).fit(x_train,y_train)
```

Question 9

Evaluate **Training** Performance

In [ ]:
```python
predictions = pd.DataFrame([LR_model.predict_proba(x_train)[:,1],
           LDA_model.predict_proba(x_train)[:,1],
           QDA_model.predict_proba(x_train)[:,1],
           KNN_model.predict_proba(x_train)[:,1]], index=['LR', 'LDA', 'QDA', 'KNN']).T
predictions.head()
```

Out[ ]:

	LR	LDA	QDA	KNN
0	0.410790	0.352063	0.221037	0.8
1	0.700724	0.789125	0.184691	0.1
2	0.161503	0.084738	0.004210	0.1
3	0.693886	0.777751	0.606448	0.5
4	0.375690	0.425123	0.677802	0.6

In [ ]:
```python
from sklearn.metrics import roc_auc_score, roc_curve

for col in predictions.columns:
    print(f"Area Under ROC curve ({col}): {roc_auc_score(y_train, predictions[col])}")
```

```
Area Under ROC curve (LR): 0.8389938546025104
Area Under ROC curve (LDA): 0.8402469055090656
Area Under ROC curve (QDA): 0.8447306485355648
Area Under ROC curve (KNN): 0.8172561453974896
```

We can see that model performance on training data was very similar for all four models, with KNN performing slightly worse than the other 3.

Question 10

Evaluate **Training** Performance

In [ ]:
```python
predictions = pd.DataFrame([LR_model.predict_proba(x_test)[:,1],
           LDA_model.predict_proba(x_test)[:,1],
           QDA_model.predict_proba(x_test)[:,1],
           KNN_model.predict_proba(x_test)[:,1]], index=['LR', 'LDA', 'QDA', 'KNN']).T
predictions.head()
```

Out[ ]:

	LR	LDA	QDA	KNN
0	0.141977	0.074279	0.003857	0.1
1	0.937197	0.967739	0.999448	0.5
2	0.182571	0.096587	0.004552	0.0
3	0.460594	0.482569	0.262232	0.5
4	0.832480	0.889982	0.778962	0.5

In [ ]:
```python
for col in predictions.columns:
    print(f"Area Under ROC curve ({col}): {roc_auc_score(y_test, predictions[col])}")
```

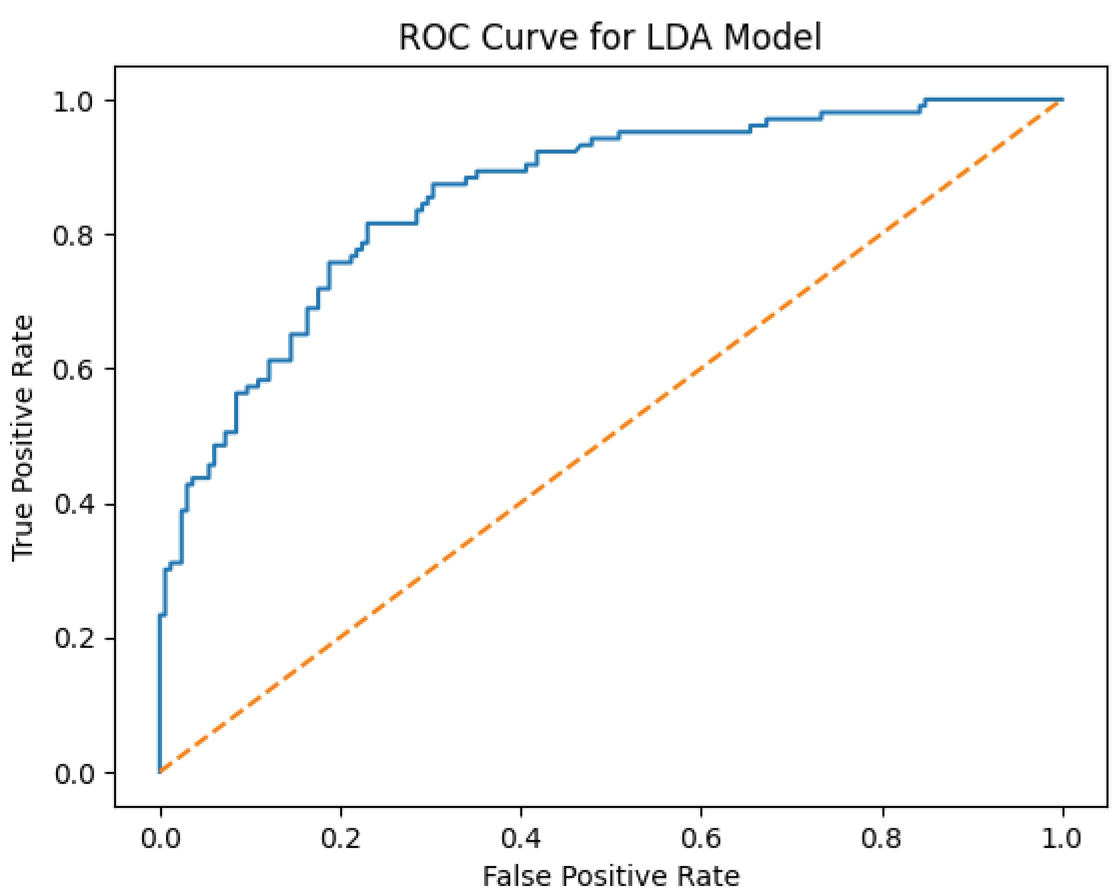
```
Area Under ROC curve (LR): 0.8476316563695205
Area Under ROC curve (LDA): 0.8574580759046779
Area Under ROC curve (QDA): 0.7999411591644602
Area Under ROC curve (KNN): 0.7019711679905855
```

Logistic Regression and LDA had almost identical (testing set) performance based on the Area Under ROC Curve metric. Interestingly, they both performed better on unseen testing data than their training data. This indicates that the assumption of a linear relationship between the log-odds and the features was reasonable, as well as the normal distribution of the features assumption (for LDA). However, we can see that QDA and KNN fit poorly to unseen test data compared to their training data. This indicataes that the more flexible models overfit to patterns in the training data that do not exist in the population.

In [ ]:
```python
from sklearn.metrics import confusion_matrix

# Make and display confusion matrix
confusion_matrix(y_test, LDA_model.predict(x_test), normalize='all')
```

Out[ ]:
```
array([[0.51492537, 0.10074627],
       [0.11940299, 0.26492537]])
```

In [ ]:
```python
y_test_numeric = np.where(y_test == 'Yes', 1, 0)

fpr, tpr, thresholds = roc_curve(y_true=y_test_numeric, y_score=predictions['LDA'])

plt.plot(fpr, tpr)
plt.plot([0,1], [0,1], ls='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for LDA Model')
plt.show()
```



Question 11

$$p = \frac{e^z}{1+e^z} \Rightarrow p + pe^z = e^z$$

$$e^z(1-p) = p \Rightarrow e^z = \frac{p}{1-p}$$

$$\boxed{z = \ln\left(\frac{p}{1-p}\right)}$$

Question 12

a) $\text{odds} = \dfrac{p}{1-p} = e^{z}$ $\Rightarrow$ $\dfrac{\text{odds}'}{\text{odds}} = \dfrac{e^{z'}}{e^{z}} = e^{\beta_0 + \beta_1(x_1+2) - (\beta_0 + \beta_1 x_1)}$

$\text{odds}' = \dfrac{p'}{1-p'} = e^{z'}$

$= e^{2\beta_1}$

$$\boxed{\text{odds}' = e^{2\beta_1} \cdot \text{odds}}$$

b) $p = \dfrac{1}{1+e^{-z}}$ $z = \beta_0 + \beta_1 X_1$

If $\beta_1 < 0$ and $X_1 \to \infty$ $\Rightarrow$ $z \to -\infty$

$\Rightarrow e^{-z} \to \infty \Rightarrow p \sim \frac{1}{\infty} \to 0$

If $\beta_1 < 0$ and $X_1 \to -\infty \Rightarrow z \to \infty$

$\Rightarrow e^{-z} \to 0 \Rightarrow p \to 1$