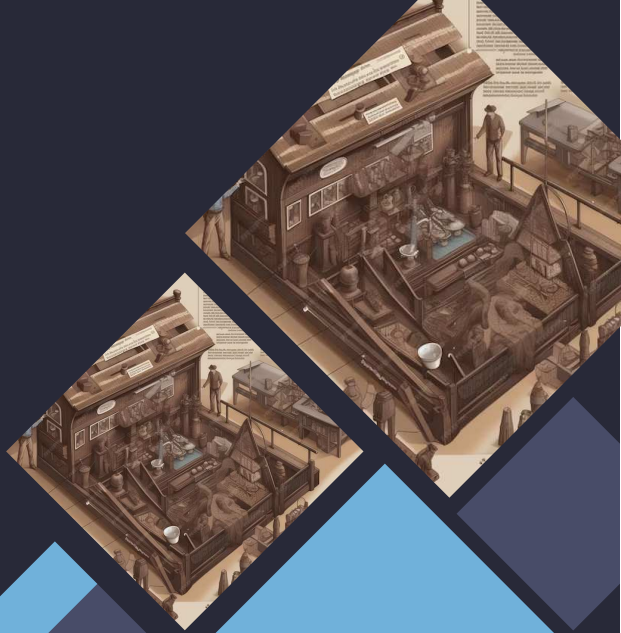
Analyzing Titanic Survival: Predictive Insights for Enhanced Preparedness



Introduction

The RMS Titanic was a British passenger liner that sank in the North Atlantic Ocean in the early morning hours of 15 April 1912, after it collided with an iceberg during its maiden voyage from Southampton to New York City. There were an estimated 2,224 passengers and crew aboard the ship, and more than 1,500 died, making it one of the deadliest commercial peacetime maritime disasters in modern history. The RMS Titanic was the largest ship afloat at the time it entered service and was the second of three Olympic-class ocean liners operated by the White Star Line. The Titanic was built by the Harland and Wolff shipyard in Belfast. Thomas Andrews, her architect, died in the disaster.

Passenger Demographics

Analyzing the **age**, **gender**, and **class** distribution of passengers who survived the Titanic disaster.

VISUALIZATION

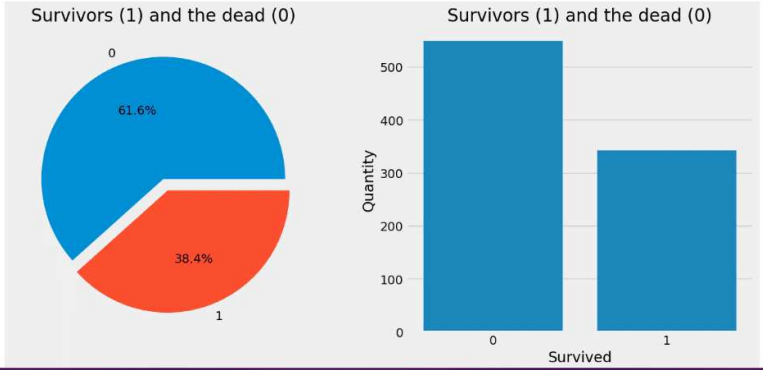
visualize the data using some pie charts and histograms to get a proper understanding of the data.

In [250]:
```
# Create a figure and a 1x2 grid of subplots
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Plot a pie chart on the first subplot (ax[0])
train['Survived'].value_counts().plot.pie(explode=[0, 0.1], autopct='%1.1f%%', ax=ax[0], shadow=False)
ax[0].set_title('Survivors (1) and the dead (0)')
ax[0].set_ylabel('')

# Plot a count plot on the second subplot (ax[1])
sns.countplot(x='Survived', data=train, ax=ax[1])  # Use x parameter instead of 'Survived'
ax[1].set_ylabel('Quantity')
ax[1].set_title('Survivors (1) and the dead (0)')

# Show the plots
plt.show()
```



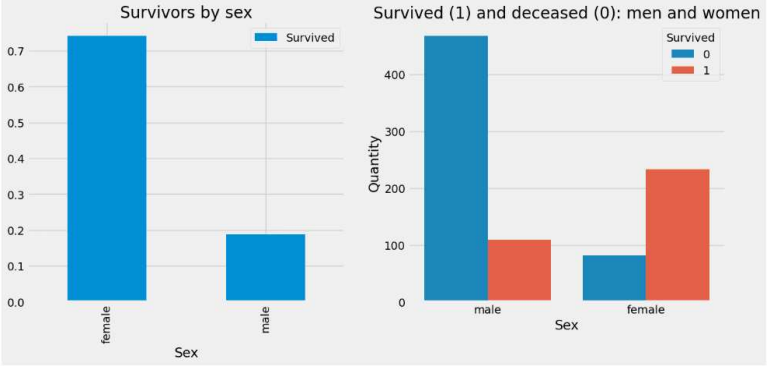
Gender feature

In [251]:
```
# Create a figure and a 1x2 grid of subplots
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Plot a bar chart on the first subplot (ax[0])
train[['Sex', 'Survived']].groupby(['Sex']).mean().plot.bar(ax=ax[0])
ax[0].set_title('Survivors by sex')

# Plot a count plot on the second subplot (ax[1])
sns.countplot(x='Sex', hue='Survived', data=train, ax=ax[1])
ax[1].set_ylabel('Quantity')
ax[1].set_title('Survived (1) and deceased (0): men and women')

# Show the plots
plt.show()
```



Survival Factors

Identifying the key factors that influenced **survival** rates, including *passenger class* and *age*.

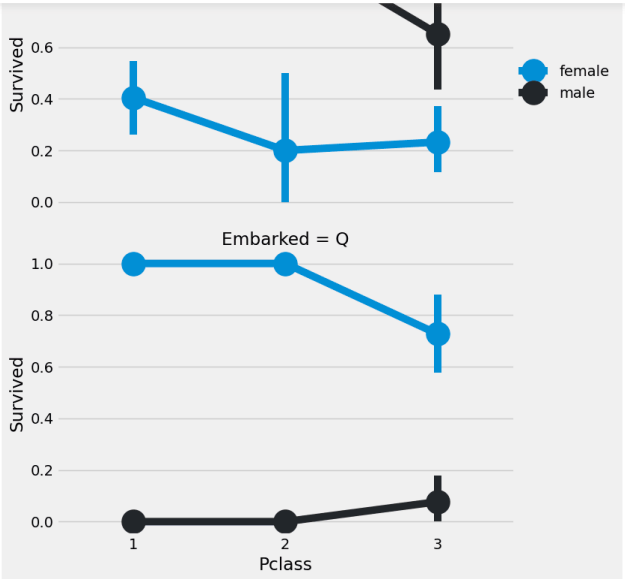
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

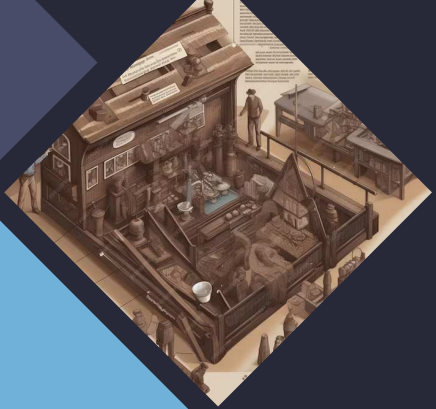
Run Markdown

Embarked, Pclass and Sex:

In [252]:
```
FacetGrid = sns.FacetGrid(train, row='Embarked', height=4.5, aspect=1.6)
FacetGrid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette=None, order=None, hue_order=None)
FacetGrid.add_legend()

plt.show()
```


Survived
0.6
0.4
0.2
0.0
female
male
Embarked = Q
Survived
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
Pclass



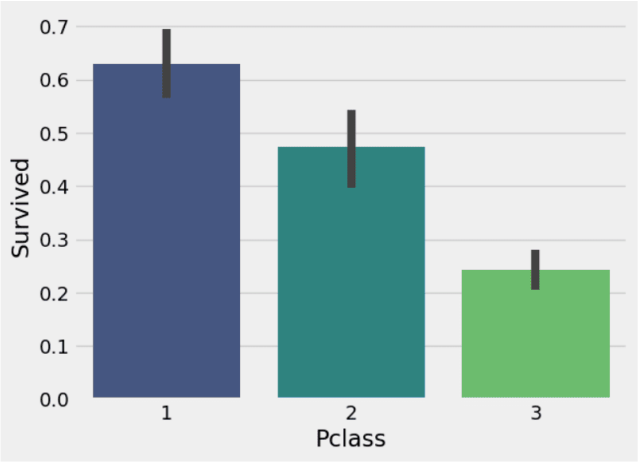
Cabin Locations

Examining the impact of **cabin locations** on passenger survival and drawing insights for modern ship design.

Pclass:

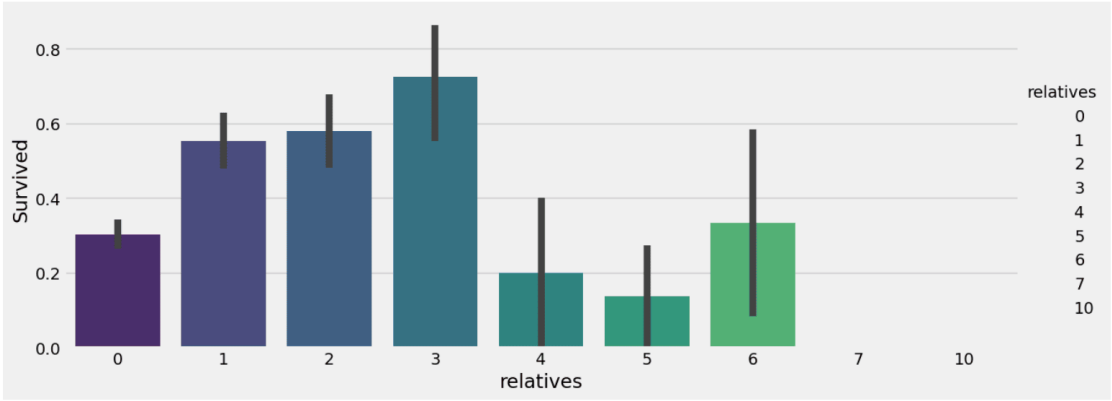
In [253]:
```
sns.barplot(x='Pclass', y='Survived', data=train, palette='viridis')
```

Out[253]:
```
<Axes: xlabel='Pclass', ylabel='Survived'>
```



In [256]:
```
sns.catplot(x='relatives', y='Survived', data=train, aspect=2.5, kind='bar', palette='viridis')
```

Out[256]: <seaborn.axisgrid.FacetGrid at 0x18a92aab3d0>



Building Machine Learning Models

Now we will train several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their testing-set, we need to use the predictions on the training set to compare the algorithms with each other. Later on, we will use cross validation.

2. Fare per Person

In [273]:
```
for dataset in data:
    dataset['Fare_Per_Person'] = dataset['Fare']/(dataset['relatives']+1)
    dataset['Fare_Per_Person'] = dataset['Fare_Per_Person'].astype(int)
# Let's take a last look at the training set, before we start training the models.
train.head(10)
```

Out[273]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Title	Age_Class	Fare_Per_Person
0	0	3	0	2	1	0	0	0	1	0	8	1	6	0
1	1	1	1	5	1	0	3	1	1	0	3	3	5	1
2	1	3	1	3	0	0	0	0	0	1	8	2	9	0
3	1	1	1	5	1	0	3	0	1	0	3	3	5	1
4	0	3	0	5	0	0	1	0	0	1	8	1	15	1
5	0	3	0	4	0	0	1	2	0	1	8	1	12	1
6	0	1	0	6	0	0	3	0	0	1	5	1	6	3
7	0	3	0	0	3	1	2	0	4	0	8	4	0	0
8	1	3	1	3	0	2	1	0	2	0	8	3	9	0
9	1	2	1	1	1	0	2	1	1	0	8	3	2	1

can form the categories.

In [270]:
```
train.head(10)
```

Out[270]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Title
0	0	3	0	2	1	0	7	0	1	0	8	1
1	1	1	1	5	1	0	71	1	1	0	3	3
2	1	3	1	3	0	0	7	0	0	1	8	2
3	1	1	1	5	1	0	53	0	1	0	3	3
4	0	3	0	5	0	0	8	0	0	1	8	1
5	0	3	0	4	0	0	8	2	0	1	8	1
6	0	1	0	6	0	0	51	0	0	1	5	1
7	0	3	0	0	3	1	21	0	4	0	8	4
8	1	3	1	3	0	2	11	0	2	0	8	3
9	1	2	1	1	1	0	30	1	1	0	8	3

In [271]:
```
data = [train, test]

for dataset in data:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
```

Another great quality of random forest is that they make it very easy to measure the relative importance of each feature. Sklearn measure a features importance by looking at how much the treee nodes, that use that feature, reduce impurity on average (across all trees in the forest). It computes this score automaticall for each feature after training and scales the results so that the sum of all importances is equal to 1. We will acces this below:

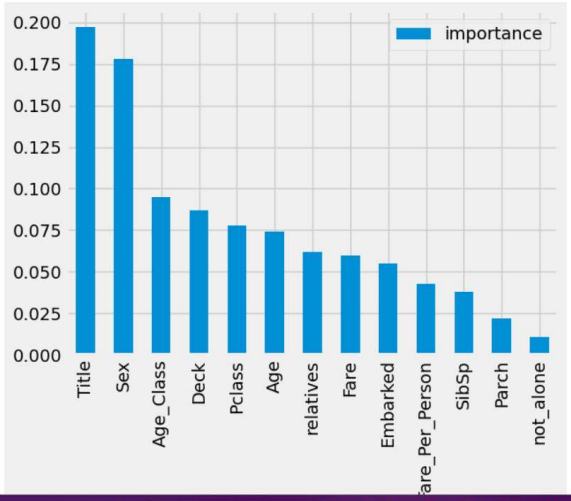
In [300]:
```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(random_forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
importances.head(15)
```

Out[300]:

feature	importance
Title	0.197
Sex	0.178
Age_Class	0.095
Deck	0.087
Pclass	0.078
Age	0.074
relatives	0.062
Fare	0.060
Embarked	0.055
Fare_Per_Person	0.043
SibSp	0.038
Parch	0.022
not_alone	0.011

In [301]:
```
importances.plot.bar()
```

Out[301]: <Axes: xlabel='feature'>



Precision Recall Curve

For each person the Random Forest algorithm has to classify, it computes a probability based on a function and it classifies the person as survived (when the score is bigger the than threshold) or as not survived (when the score is smaller than the threshold). That's why the threshold plays an important part.

Precision Recall Curve

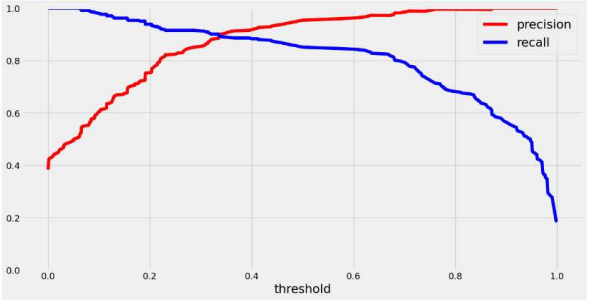
For each person the Random Forest algorithm has to classify, it computes a probability based on a function and it classifies the person as survived (when the score is bigger the than threshold) or as not survived (when the score is smaller than the threshold). That's why the threshold plays an important part.

In [305]:
```
from sklearn.metrics import precision_recall_curve

# getting the probabilities of our predictions
y_scores = random_forest.predict_proba(X_train)
y_scores = y_scores[:,1]

precision, recall, threshold = precision_recall_curve(Y_train, y_scores)
def plot_precision_and_recall(precision, recall, threshold):
    plt.plot(threshold, precision[:-1], "r-", label="precision", linewidth=5)
    plt.plot(threshold, recall[:-1], "b", label="recall", linewidth=5)
    plt.xlabel("threshold", fontsize=19)
    plt.legend(loc="upper right", fontsize=19)
    plt.ylim([0, 1])

plt.figure(figsize=(14, 7))
plot_precision_and_recall(precision, recall, threshold)
plt.show()
```



Another way is to plot the precision and recall against each other:

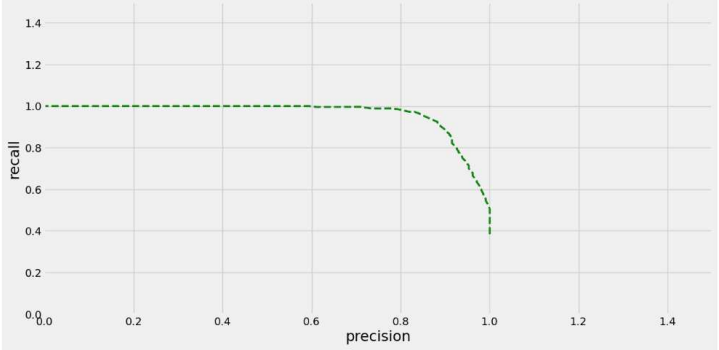
Recommendations

Providing actionable **recommendations** for improving safety measures and emergency protocols based on our findings.

Another way is to plot the precision and recall against each other:

In [306]:
```
def plot_precision_vs_recall(precision, recall):
    plt.plot(recall, precision, "g--", linewidth=2.5)
    plt.ylabel("recall", fontsize=19)
    plt.xlabel("precision", fontsize=19)
    plt.axis([0, 1.5, 0, 1.5])

plt.figure(figsize=(14, 7))
plot_precision_vs_recall(precision, recall)
plt.show()
```

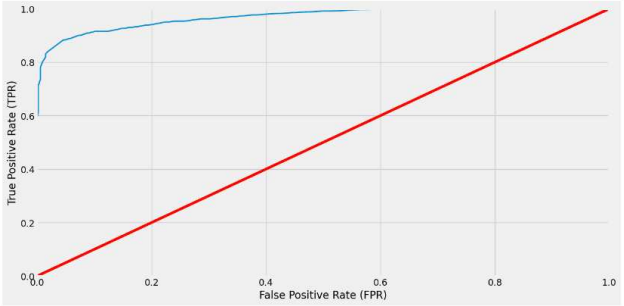


ROC AUC Curve

Another way to evaluate and compare your binary classifier is provided by the ROC AUC Curve. This curve plots the true positive rate (also called recall) against the false positive rate (ratio of incorrectly classified negative instances), instead of plotting the precision versus the recall.

In [307]:
```
from sklearn.metrics import roc_curve
# compute true positive rate and false positive rate
false_positive_rate, true_positive_rate, thresholds = roc_curve(Y_train, y_scores)
# plotting them against each other
def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'r', linewidth=4)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(14, 7))
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```



The red line in the middel represents a purely random classifier (e.g a coin flip) and therefore your classifier should be as far away from it as possible. Our Random Forest model seems to do a good job.

Conclusion

We started with the data exploration where we got a feeling for the dataset, checked about missing data and learned which features are important. During this process we used seaborn and matplotlib to do the visualizations. During the data preprocessing part, we computed missing values, converted features into numeric ones, grouped values into categories and created a few new features. Afterwards we started training 8 different machine learning models, picked one of them (random forest) and applied cross validation on it. Then we discussed how random forest works, took a look at the importance it assigns to the different features and tuned it's performace through optimizing it's hyperparameter values. Lastly, we looked at it's confusion matrix and computed the models precision, recall and f-score.Drawing insights from Titanic survival data can inform *strategies* for enhanced preparedness in modern maritime operations.

Thanks!

Do you have any questions?

- mk1423141@email.com
- +917587805856
- www.linkedin.com/in/manishkumarbelsare