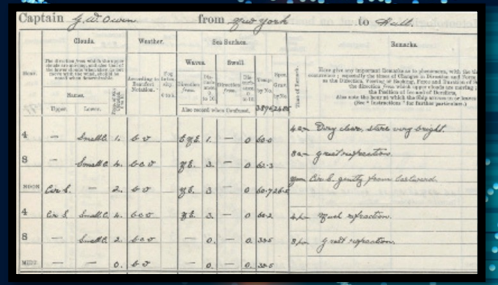
Titanic - Machine Learning from Disaster

Group 5: Ahmed Moustafa, Rohit Kala, and Justin Kilgo

Context


Captain E. J. Owen from New York to Hull.

- The RMS Titanic sank in the early morning hours of 15 April 1912 in the North Atlantic Ocean.
- Britain & the US launched an investigation which produced extensive tables of passengers and crew, broken down by age group, gender, class and survival (July 1912).

- According to investigations, ~1500 people died
- This data was organized and packaged for Data Analysis in December of 1999 → Kaggle competition.
    - What are some of the difficulties of using such old paper-sourced data?

Key Technologies

- Python Pandas
- Jupyter Notebook – Google Colaboratory

- **Python Libraries** for Data Analysis + Training:
  - numpy
  - seaborn
  - sklearn
  - xgboost

Exploratory Data Analysis

Training DataFrame (891 of 1309 total):

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
----------------------
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
Training Missing Values:
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
----------------------
Testing Missing Values:
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

Data Preprocessing

Combine the train & test DataFrames to unify preprocessing changes

```
# Extract the titles from Name and create a new Title Column with mapped titles
df['Title'] = df['Name'].map(lambda name:name.split(',')[1].split('.')[0].strip())
df['Title'] = df['Title'].map({
  "Capt": "Officer",
  "Col": "Officer",
  "Major": "Officer",
  "Jonkheer": "Royalty",
  "Don": "Royalty",
  "Sir" : "Royalty",
  "Dr": "Officer",
  "Rev": "Officer",
  "the Countess":"Royalty",
  "Mme": "Mrs",
  "Mlle": "Miss",
  "Ms": "Mrs",
  "Mr" : "Mr",
  "Mrs" : "Mrs",
  "Miss" : "Miss",
  "Master" : "Master",
  "Lady" : "Royalty"
})
```

```
# Group factors and their respective Age Medians

def get_age_median(row):
  train_median_df = df[:891].groupby(['Sex', 'Pclass', 'Title']).median().\
    reset_index()[['Sex', 'Pclass', 'Title', 'Age']]
  conditional = (
        (train_median_df['Sex'] == row['Sex']) &
        (train_median_df['Title'] == row['Title']) &
        (train_median_df['Pclass'] == row['Pclass'])
  )
  return train_median_df[conditional]['Age'].values[0]

df['Age'] = df.apply(lambda row: get_age_median(row) if np.isnan(row['Age']) else row['Age'], axis=1)
```

Data Preprocessing

- The name column doesn't provide much information for survival so we can drop it.
- After mapping the Titles, we can dummy encode them

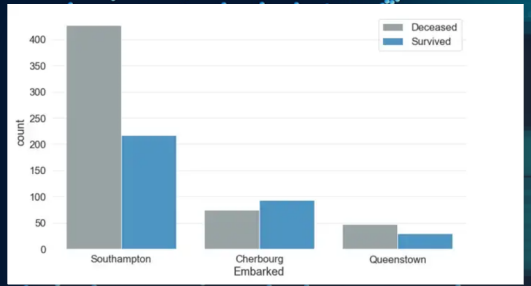
```
df.drop('Name', axis=1, inplace=True)

titles_dummies = pd.get_dummies(df['Title'], prefix='Title')
df = pd.concat([df, titles_dummies], axis=1)
df.drop('Title', axis=1, inplace=True)
```

- Embarked tells us the city of departure. S is the mode so we use that to fill in null rows and dummy encode

```
df.Embarked.fillna('S', inplace=True)

embarked_dummies = pd.get_dummies(df['Embarked'], prefix='Embarked')
df = pd.concat([df, embarked_dummies], axis=1)
df.drop('Embarked', axis=1, inplace=True)
```



Data Preprocessing

- We denoted null Cabin values with an ‘X’ and dummy encoded using the 1st letter (*train on sections of the ship rather than specific rooms*)

```
df.Cabin.fillna('X', inplace=True)
df['Cabin'] = df['Cabin'].map(lambda c: c[0])

cabin_dummies = pd.get_dummies(df['Cabin'], prefix='Cabin')
df = pd.concat([df, cabin_dummies], axis=1)
df.drop('Cabin', axis=1, inplace=True)
```

- Pclass gave us the status of the passengers: class-1 > class-2 > class-3. We also dummy encoded this.

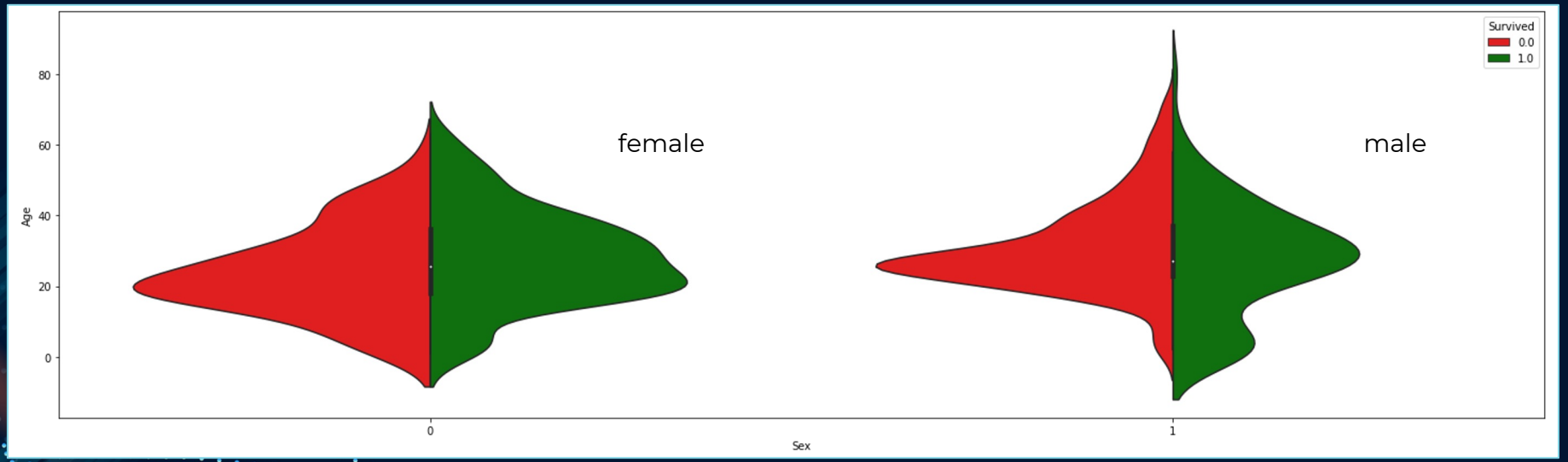
```
pclass_dummies = pd.get_dummies(df['Pclass'], prefix="Pclass")
df = pd.concat([df, pclass_dummies],axis=1)
df.drop('Pclass',axis=1,inplace=True)
```

```
# Replace missing values with the average fare
df['Fare'].fillna(df.iloc[:891]['Fare'].mean(), inplace=True)

# Replace "male" with 1 and "female" with 0
df['Sex'] = df['Sex'].map({"male":1, "female":0})
```

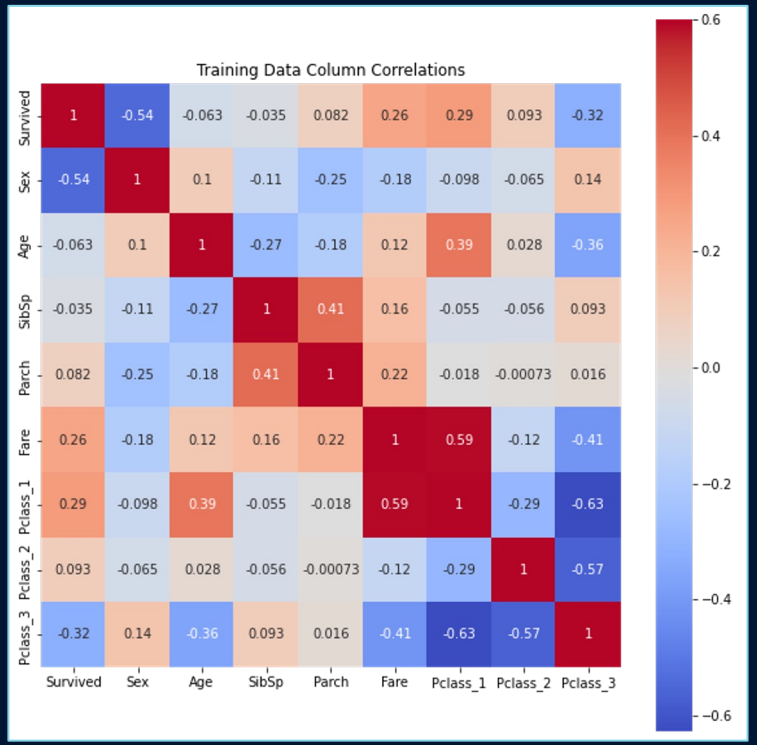
Remaining Preprocessing

Age & Sex Survival Analysis

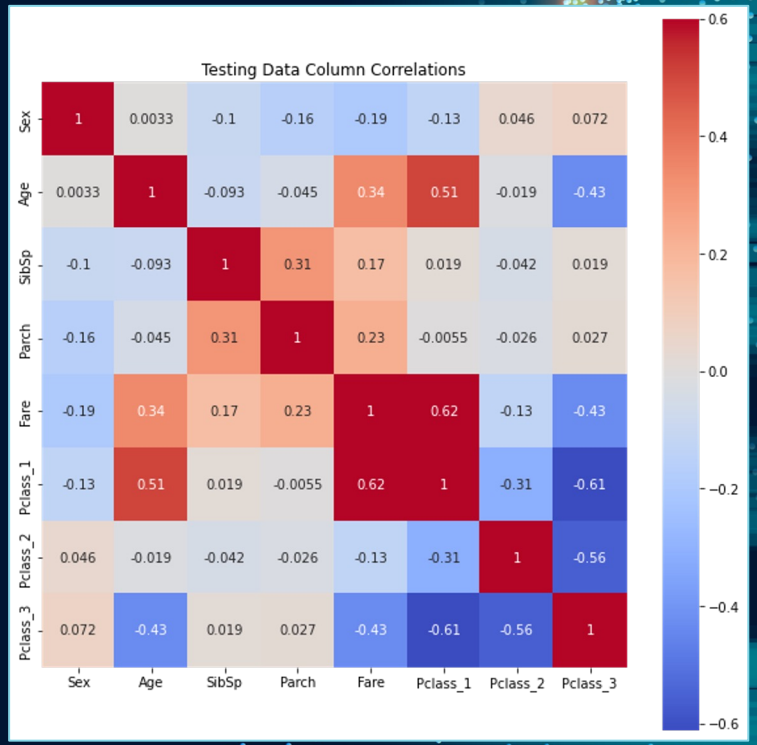



```
# Create a Violin Plot to show the difference in survival based on age & sex
fig = plt.figure(figsize=(25, 7))
sns.violinplot(x='Sex', y='Age',
               hue='Survived', data=train_df,
               split=True,
               palette={0: "r", 1: "g"}
              );
```

More Data Analysis



Seaborn heatmaps



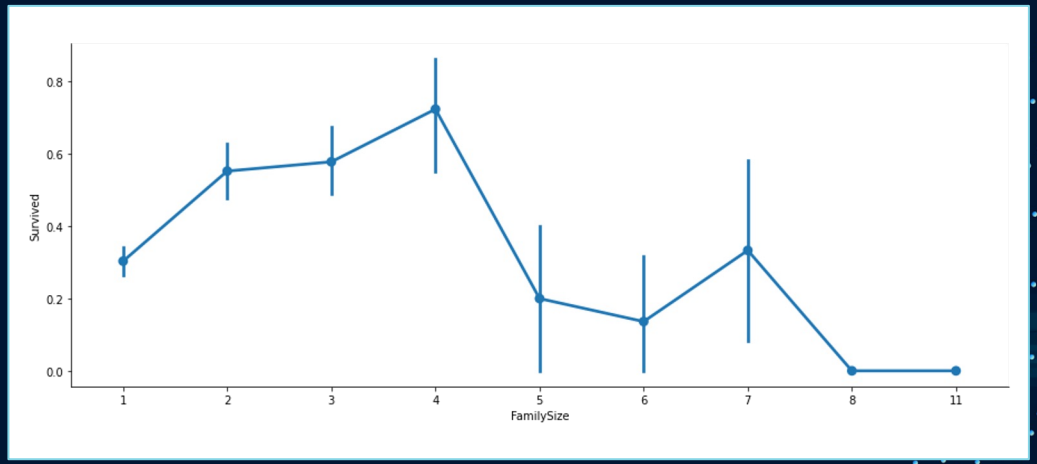
Feature Engineering

Parch and SibSp can be combined to represent the # of people a person was with (*spouse*, *parents*, *siblings* → *FamilySize*).

```
df['FamilySize'] = df['Parch'] + df['SibSp'] + 1

df['Singleton'] = df['FamilySize'].map(lambda s: 1 if s == 1 else 0)
df['SmallFamily'] = df['FamilySize'].map(lambda s: 1 if 2 <= s <= 4 else 0)
df['LargeFamily'] = df['FamilySize'].map(lambda s: 1 if 5 <= s else 0)
```

Seaborn factorplot



Models

```
# Create a Logistic Regression Model
lr = LogisticRegression(max_iter = 100000)

# Fit the Model using our training data
lr.fit(train_df[feature_columns], train_df[target_column])
```

```
# RANDOM FOREST CLASSIFIER

rf = RandomForestClassifier(n_estimators=200, min_samples_leaf=3, max_features=.5, n_jobs=-1)
rf.fit(train_df[feature_columns], train_df[target_column])
```

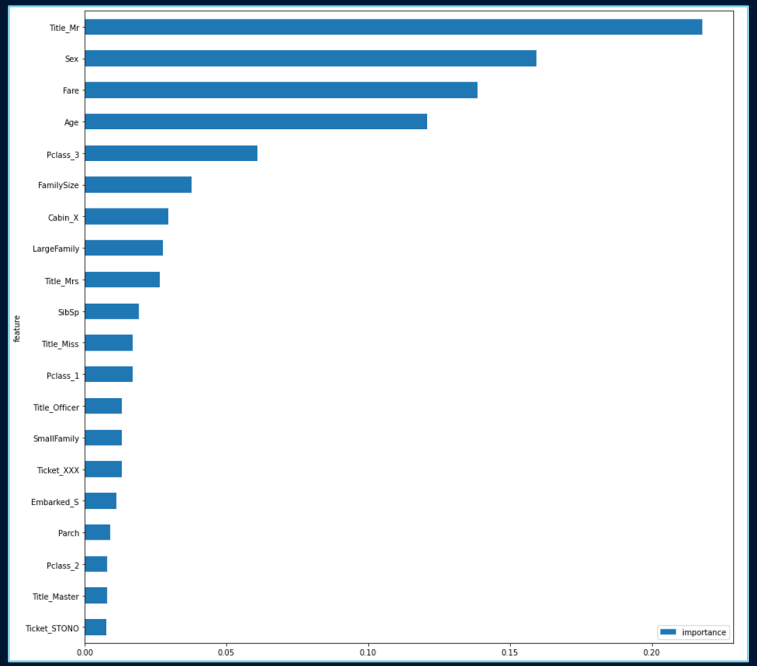
```
# EXTREME GRADIENT BOOSTING

xgb = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=1, gpu_id=0,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.03, max_delta_step=0, max_depth=8, min_child_weight=1,
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)
```

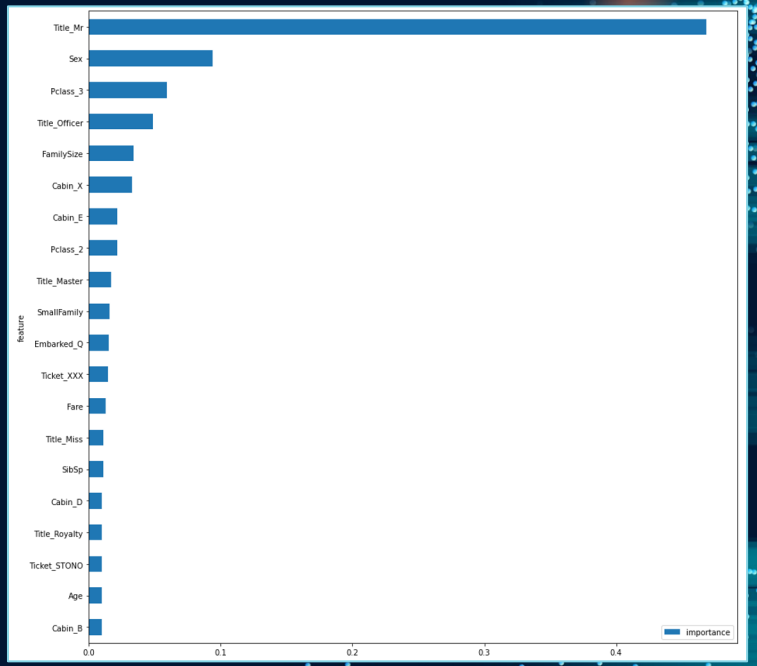
Feature Importance

```
# Create a dataframe and plot of the importance of each feature
features = pd.DataFrame()
features['feature'] = feature_columns
features['importance'] = xgb.feature_importances_
features.sort_values(by=['importance'], ascending=True, inplace=True)
features.set_index('feature', inplace=True)
features[-20:].plot(kind='barh', figsize=(15, 15))
```

Random Forest



XGB



Meta Model

```
# DECISION TREE CLASSIFIER

from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

params = [{'max_leaf_nodes': list(range(2,100)), 'min_samples_split': [2, 3, 4]}]
b_clf=BaggingClassifier(GridSearchCV(DecisionTreeClassifier(random_state=42),params,cv=3,verbose=1),n_estimators=1000,max_samples=100,bootstrap=True,n_jobs=-1)
b_clf.fit(train_df[feature_columns],train_df[target_column])
y_pred=b_clf.predict(test_df[feature_columns]).astype(int)
```

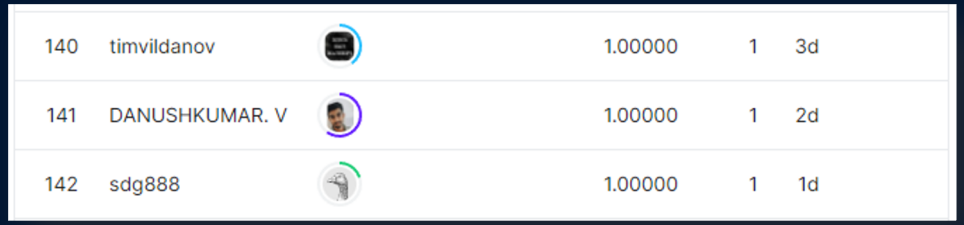
A Bagging classifier is “an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions.”

bootstrap = True → samples are drawn w/ replacement

n_jobs = -1 → all processors used to run jobs in parallel

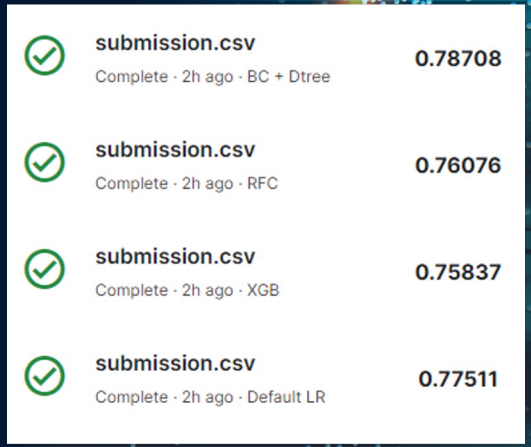
Evaluation

- Decision Tree with the Bagging Classifier was our best model → top 8%
- Only gained ~1% over Logistic Regression!



140	timvildanov	1.00000	1	3d
141	DANUSHKUMAR. V	1.00000	1	2d
142	sdg888	1.00000	1	1d

Over 140 cheaters



submission.csv	Complete · 2h ago · BC + Dtree	0.78708
submission.csv	Complete · 2h ago · RFC	0.76076
submission.csv	Complete · 2h ago · XGB	0.75837
submission.csv	Complete · 2h ago · Default LR	0.77511

Future Improvements

More feature engineering and fine-tuning models

References

- https://rss.onlinelibrary.wiley.com/doi/10.1111/j.1740-9713.2019.01229.x
- https://www.kaggle.com/competitions/titanic
- https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db#:~:text=Both%20expand%20the%20feature%20space,avoids%20the%20dummy%20variable%20trap.
- https://towardsdatascience.com/a-beginners-guide-to-kaggle-s-titanic-problem-3193cb56f6ca#:~:text=Embarked%20implies%20where%20the%20traveler,the%20rest%20boarded%20from%20Queenstown.
- https://xgboost.readthedocs.io/en/stable/parameter.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html