TITANIC

Case study

Agenda

1. Dataset analysis
2. Preprocessing techniques
3. Output features, correlations and graphs
4. KNN classifier
5. Random forest classifier
6. Decision tree classifier

On April 15, 1912, after colliding with an iceberg, “unsinkable” Titanic sank resulting in death of 1502 out of 2224 passengers and crew.”

Dataset analysis

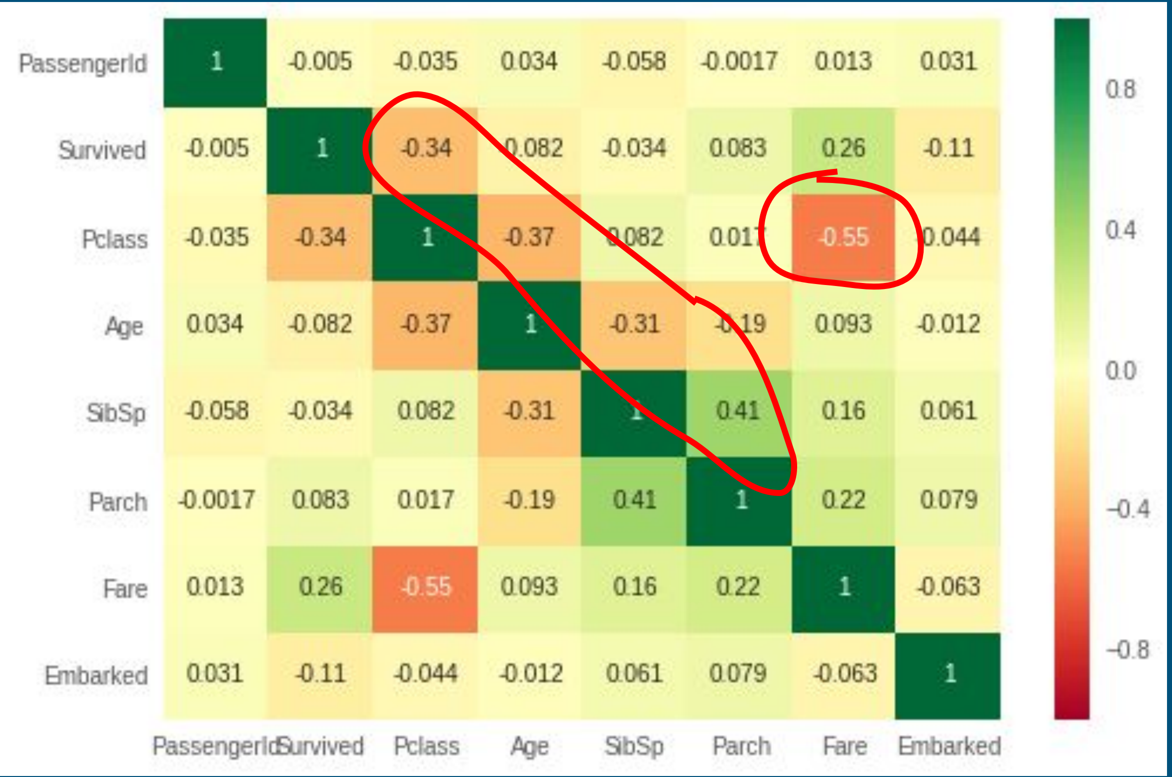
The dataset contains data of 891 passengers:

- PassengerId, Name, Sex, Age
- PClass - Class of the passenger (1, 2 or 3)
- Ticket - The ticket number (eg. “PC 17955”)
- Fare - How much did the passenger pay (eg. “14.45”)
- Cabin - Contains the cabin number (eg. “C85”)
- Embarked - Port of embarkation (S, C or Q)
- Parch - number of parents and children
- SibSp - number of siblings and spouses
- Survived - whether the passenger survived (1 or 0)

Total: 11 features and 1 binary decision label

Raw correlations

- Pclass and Fare
- SibSp and Parch
- Pclass and Age
- Survived and PClass
- Age and SibSp



- Better class = higher fare
- More parents = more siblings
- Better class = higher age
- Better class = survived
- Lower age = more siblings

Preprocessing Techniques

a) Standardization (Fare)
b) Handling missing values (Age, Embarked)
c) One-hot encoding (Embarked, Title, Sex)
d) Discretization (AgeBin)
e) Derived attributes (FamilySize, Title)
f) Dropped columns (Name, Age, SibSp, Parch)

	Pclass	Fare	AgeBin	FamilySize	Embarked_Q	Embarked_S	Title_Miss	Title_Mr	Title_Mrs	Title_Other	Sex_male
0	3	-0.531122	2	2	False	True	False	True	False	False	True
1	1	1.099279	2	2	False	False	False	False	True	False	False
2	3	-0.513935	2	1	False	True	True	False	False	False	False
3	1	0.636300	2	2	False	True	False	False	True	False	False
4	3	-0.510753	2	1	False	True	False	True	False	False	True

Output features - explanation

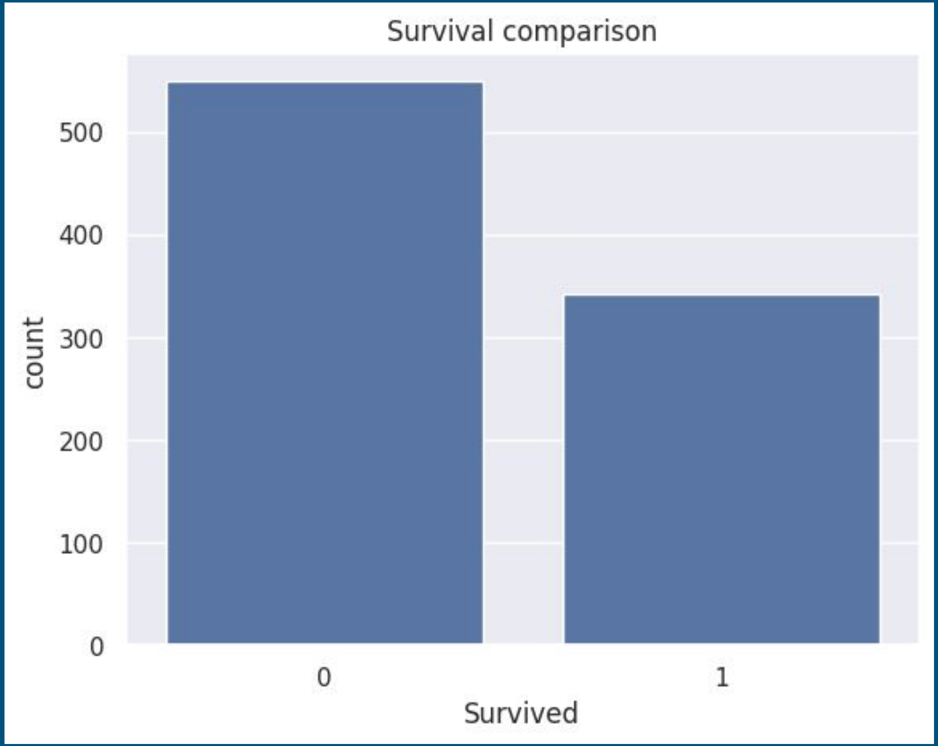
Output feature analysis:

- PClass - unchanged
- Fare - standardized *Fare* column
- AgeBin - discretized *Age* column
- FamilySize - Size of family (sum of SibSp and ParCh columns)
- Embarked_Q, Embarked_S - port of embarkment expressed as hot label coding
- Title_Miss, Title_Mrs, Title_Mr, Title_Other - one hot encoded title
- Sex_male - sex expressed as one hot encoding
- Survived - unchanged

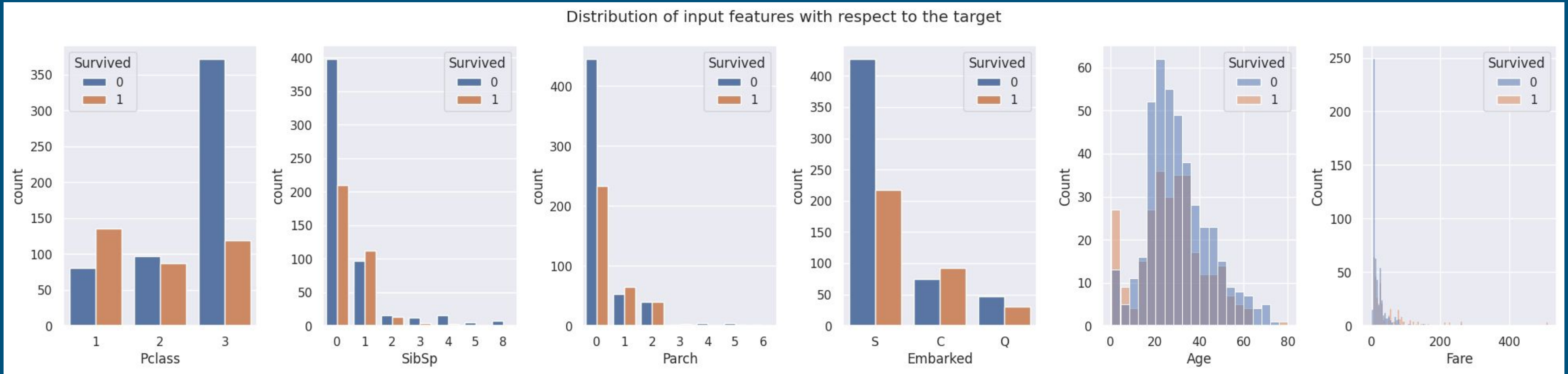
Preprocessing result - no missing values

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Pclass       891 non-null    int64
 1   Fare         891 non-null    float64
 2   AgeBin       891 non-null    category
 3   FamilySize   891 non-null    int64
 4   Embarked_Q   891 non-null    bool
 5   Embarked_S   891 non-null    bool
 6   Title_Miss   891 non-null    bool
 7   Title_Mr     891 non-null    bool
 8   Title_Mrs    891 non-null    bool
 9   Title_Other  891 non-null    bool
 10  Sex_male     891 non-null    bool
dtypes: bool(7), category(1), float64(1), int64(2)
memory usage: 28.1 KB
```

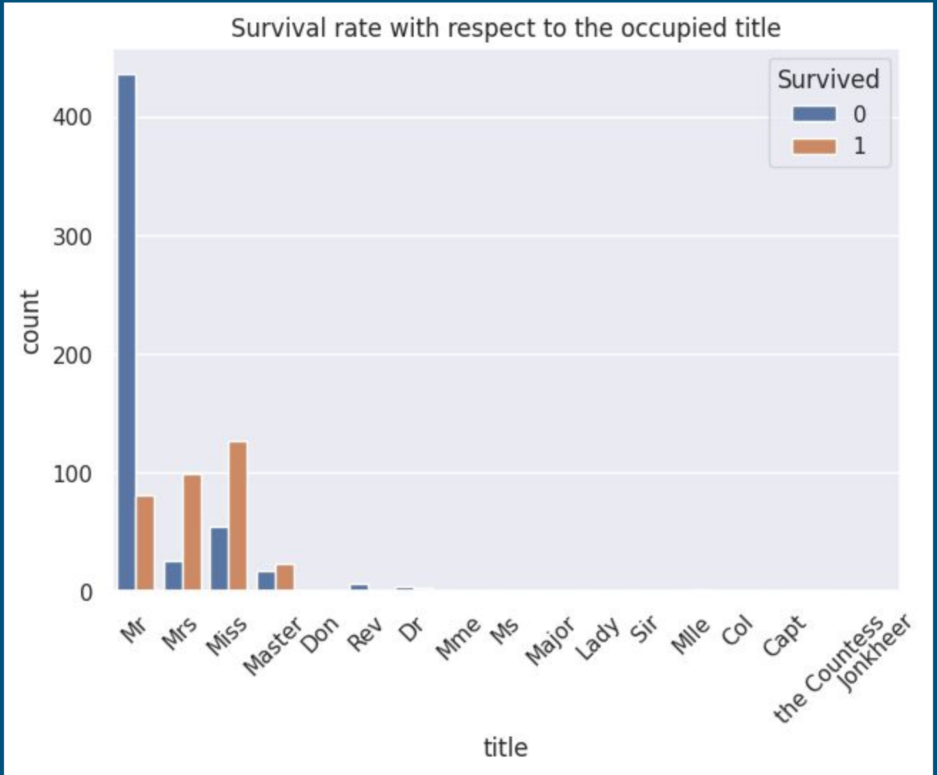
Distribution of target variable



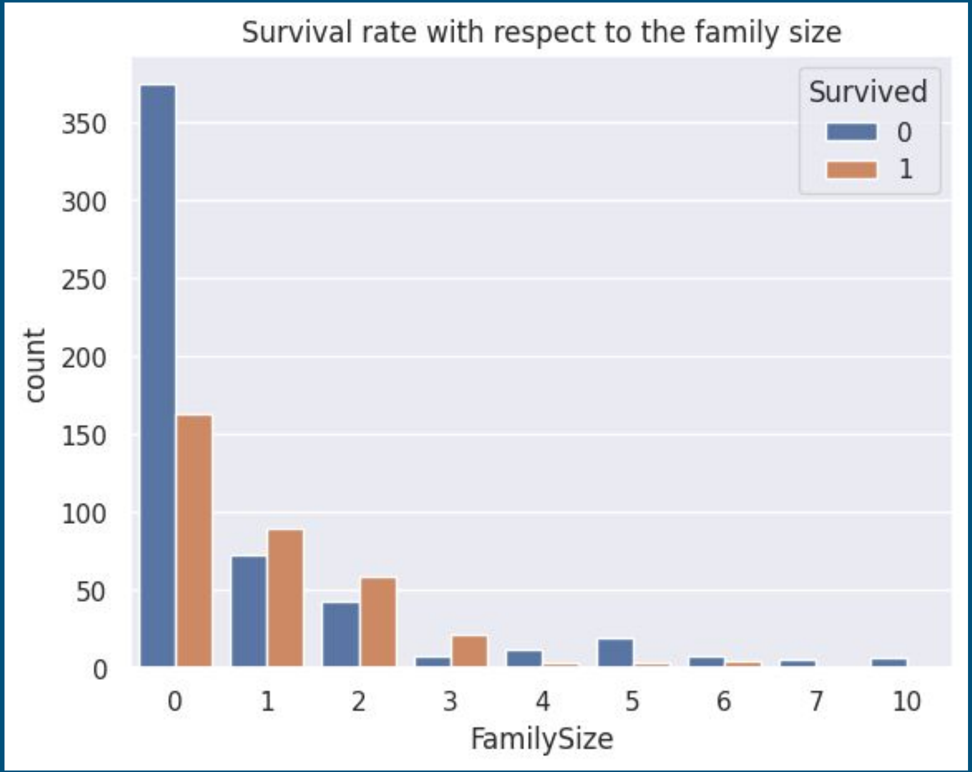
Distribution of input features with respect to target



Dependency between titles and survival status



Survival Rate with respect to FamilySize



K-Nearest Neighbours (KNN)

```
k_values = range(1, 21)
metric_values = ['euclidean', 'manhattan', 'chebyshev']
search_method_values = ['ball_tree', 'kd_tree', 'brute']
```

Overall:

- Manhattan gives best results
- K ~ 10
- No need for complicated search methods
- 84% best accuracy (average of 10 splits 20%)

k	metric	search_method	accuracy
10	manhattan	brute	0.840782
12	manhattan	brute	0.839665
11	manhattan	ball_tree	0.839106
7	manhattan	ball_tree	0.839106
7	manhattan	brute	0.837989
9	manhattan	brute	0.835754
20	euclidean	ball_tree	0.834078
20	manhattan	ball_tree	0.833520
14	manhattan	brute	0.831844
10	euclidean	brute	0.830726

K-Nearest Neighbours (KNN) - more parameters

- K is higher ~19
  For top 3
- Again manhattan
  Is the best
- Tree search methods
  more common
- Higher accuracy 85.2%
  (average of 10 splits 20%)
- 86.1% for 10% split

```
k_values = range(1, 21)
metric_values = ['chebyshev', 'minkowski']
p_values = [1, 2, 3, 4, 5]  # 1 is manhattan, 2 is euclidean
search_method_values = ['ball_tree', 'kd_tree', 'brute']
weights_values = ['uniform', 'distance']
leaf_size_values = [10, 20, 30, 40, 50]  # for trees
```

k	metric	search_method	weight	leaf_size	p	accuracy
18	minkowski	kd_tree	uniform	50	1	85.195531
20	minkowski	ball_tree	uniform	20	1	85.139665
18	minkowski	kd_tree	distance	10	1	84.972067
8	minkowski	kd_tree	uniform	20	1	84.916201
10	minkowski	ball_tree	uniform	20	1	84.860335
14	minkowski	kd_tree	uniform	40	1	84.804469
10	minkowski	ball_tree	uniform	40	2	84.692737
10	minkowski	kd_tree	uniform	20	1	84.692737

k	metric	search_method	weight	leaf_size	p	accuracy
12	minkowski	kd_tree	uniform	50	1	86.111111
9	minkowski	kd_tree	uniform	40	1	85.555556
13	minkowski	brute	uniform	10	1	85.444444

Random Forests!

RandomForestClassifier

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *,
criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None,
random_state=None, verbose=0, warm_start=False, class_weight=None,
ccp_alpha=0.0, max_samples=None, monotonic_cst=None) #
```

[source]

Random Forest - hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=3,
                           verbose=3,
                           n_jobs=-1)

grid_search.fit(X_train, y_train)
```

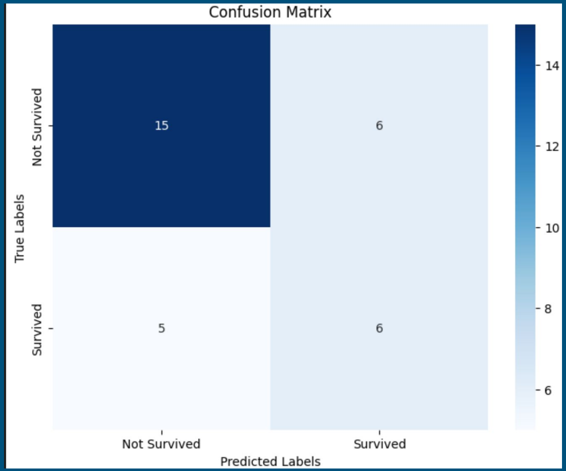
Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

params = {
    'bootstrap': True,
    'max_depth': 10,
    'min_samples_leaf': 1,
    'min_samples_split': 10,
    'n_estimators': 100
}

rf_classifier = RandomForestClassifier(**params)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Random Forest model: {accuracy:.2f}")
```

✓ 0.9s

```
Accuracy of the Random Forest model: 0.69
```

- Simple
- Sad accuracy :(



Decision Tree - sklearn

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
```
[source]

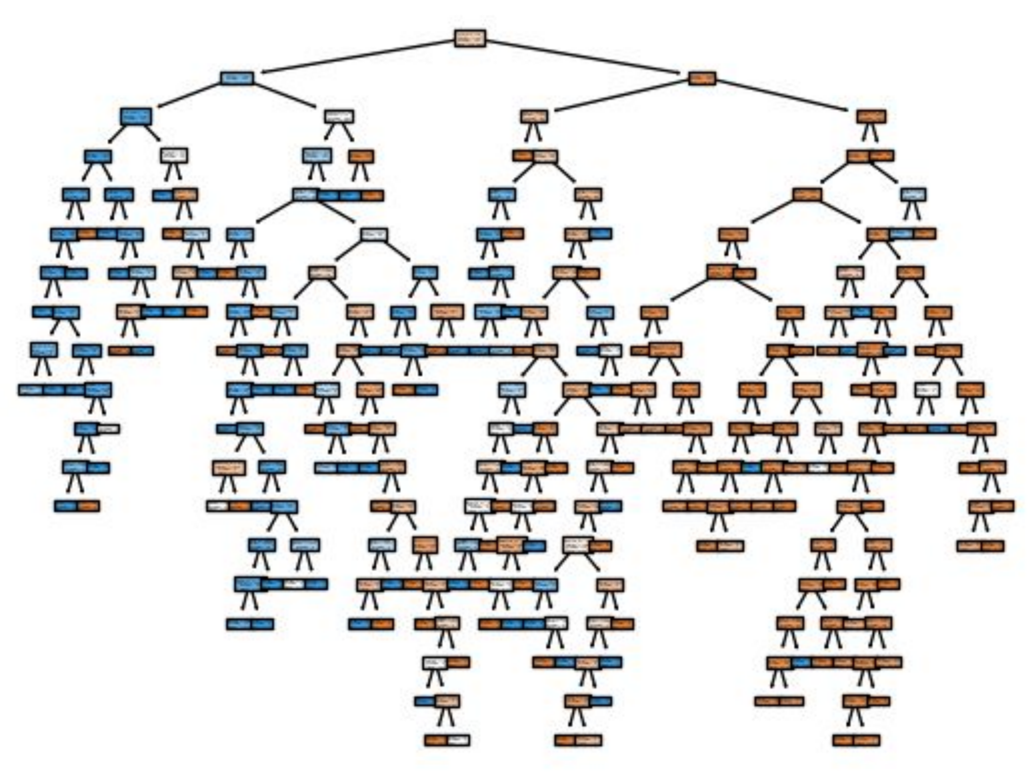
We will be interested in:

a) criterion
b) splitter
c) max_depth
d) min_samples_split
e) min_samples_leaf
f) ccp_alpha

Decision tree - how to use it?

```
clf = tree.DecisionTreeClassifier(criterion = "entropy")
clf = clf.fit(X_train, y_train)
tree.plot_tree(clf, filled = True, feature_names=X_train.columns)

y_tree = clf.predict(X_test)
accuracy = accuracy_score(y_tree, y_test)
print(f"Accuracy obtained for decision tree: {round(100*accuracy, 2)}%")
```

Decision Tree - First case (default)

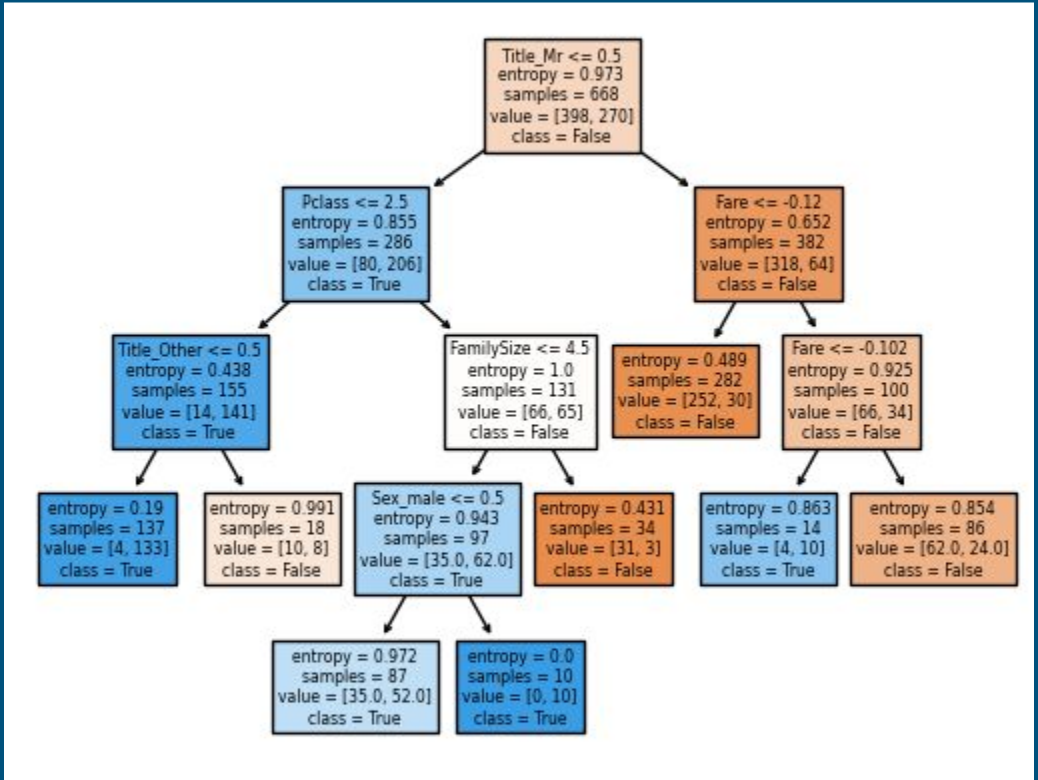


Parameters:

a) Criterion = “entropy”
b) Other parameters = default

Accuracy obtained (avg): 78.54%

Decision Tree - Second case (custom)

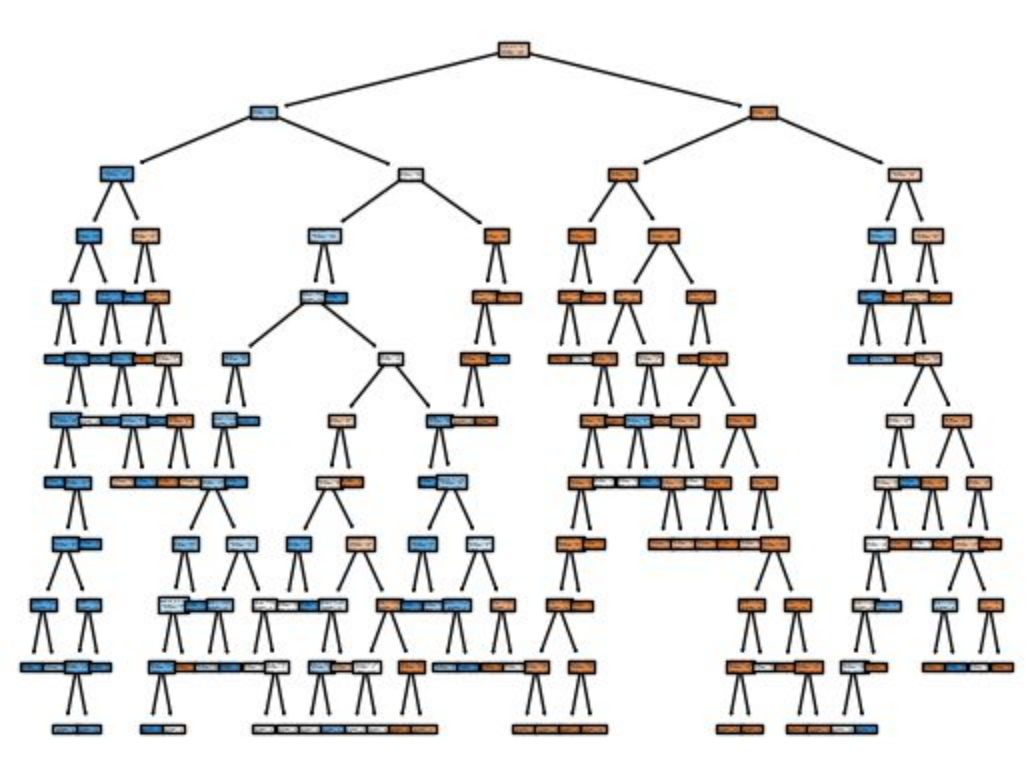


Parameters:

a) criterion: “entropy”
b) min_samples_split = 25
c) min_samples_leaf = 5
d) ccp_alpha = 0.01
e) Max_depth = 5

Accuracy obtained (avg): 81.42%

Decision Tree - Third case (trained, no ccp)

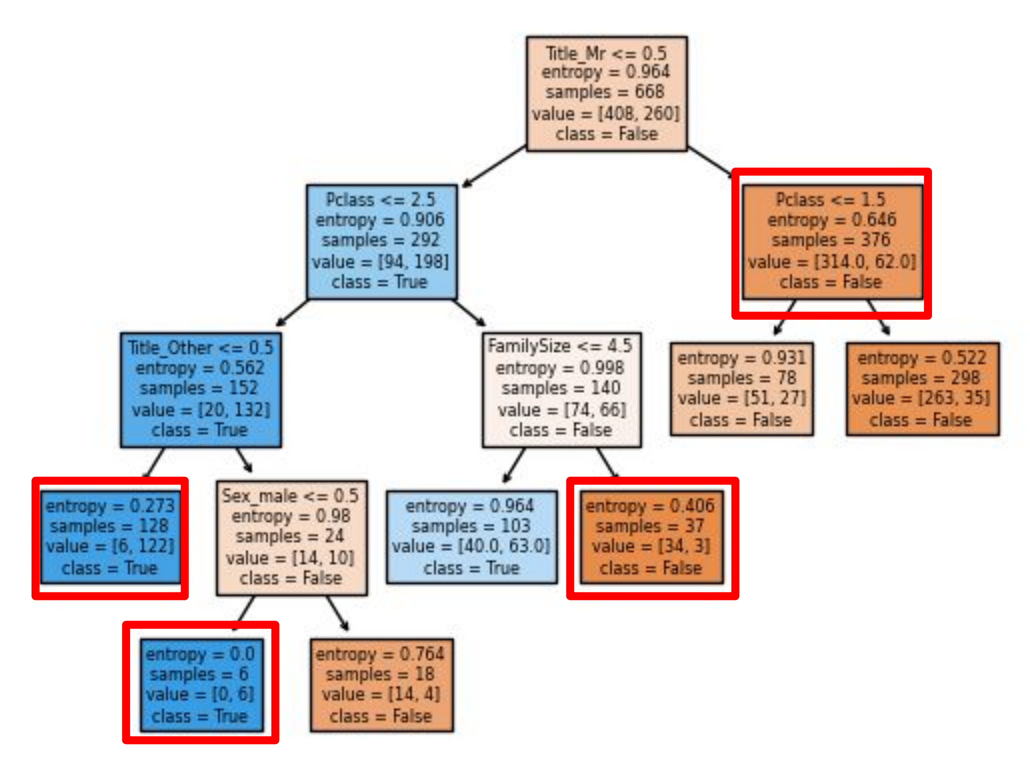


Parameters:

a) criterion: “entropy”
b) min_samples_split = 2
c) min_samples_leaf = 2
d) ccp_alpha = 0
e) Max_depth = 11

Accuracy obtained: 87.0%

Decision Tree - Fourth case (trained, ccp)

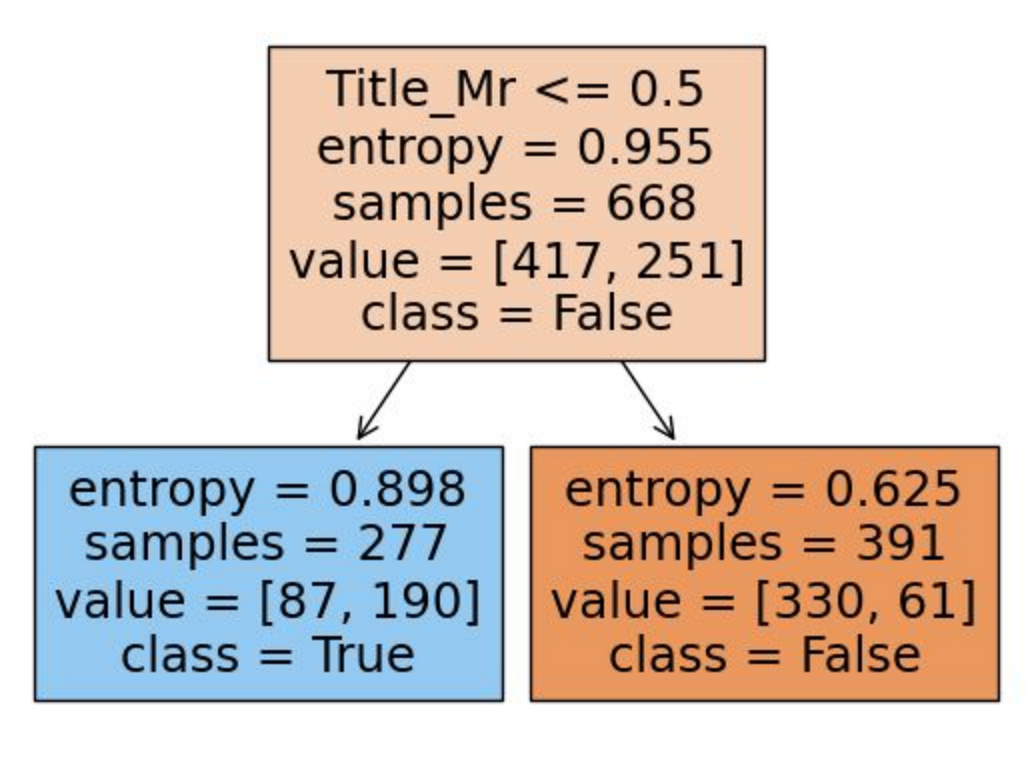


Parameters:

a) criterion: “entropy”
b) min_samples_split = 5
c) min_samples_leaf = 2
d) ccp_alpha = 0.01
e) Max_depth = 4

Accuracy obtained: 85.65%

Be careful with ccp_alpha!



Parameters:

a) criterion: "entropy"
b) min_samples_split = 5
c) min_samples_leaf = 2
d) ccp_alpha = 0.1
e) Max_depth = 4

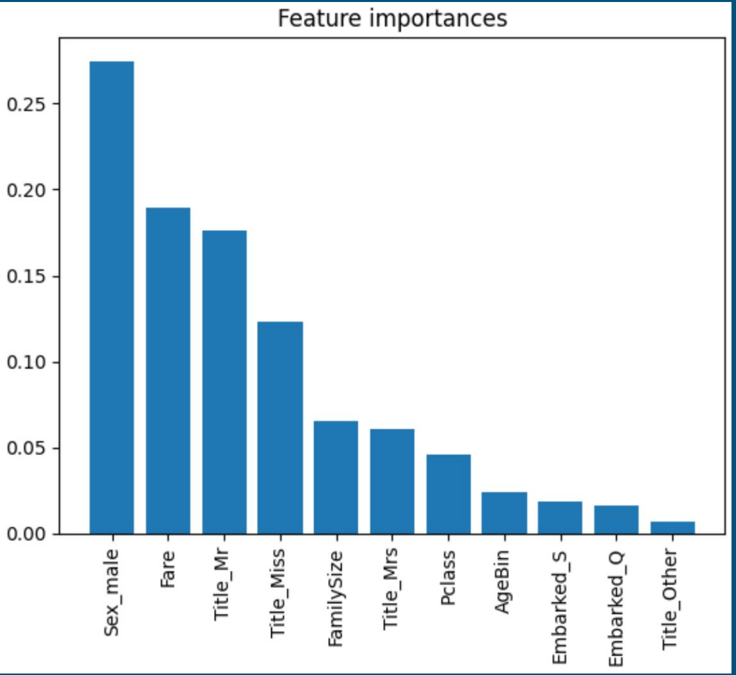
Accuracy obtained: 77.28%

More stats...

For more statistics (accuracies, confusion matrices etc) and code implementation go to jupyter notebook!

Conclusions

- Model Performance
- Feature Insights
- Preprocessing Impact



Sources

a) scikit-learn.org (documentation)
b) wikipedia.org
c) kaggle.com

Thank You

- Mateusz Idziejczak 155842
- Mateusz Stawicki 155900
- Kuba Czech 156035