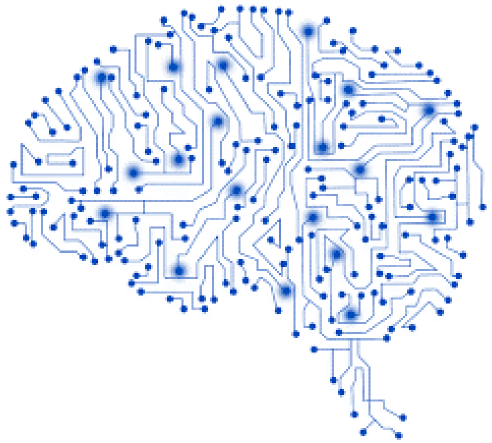
University of Minho
School of Engineering



Dados e Aprendizagem Automática

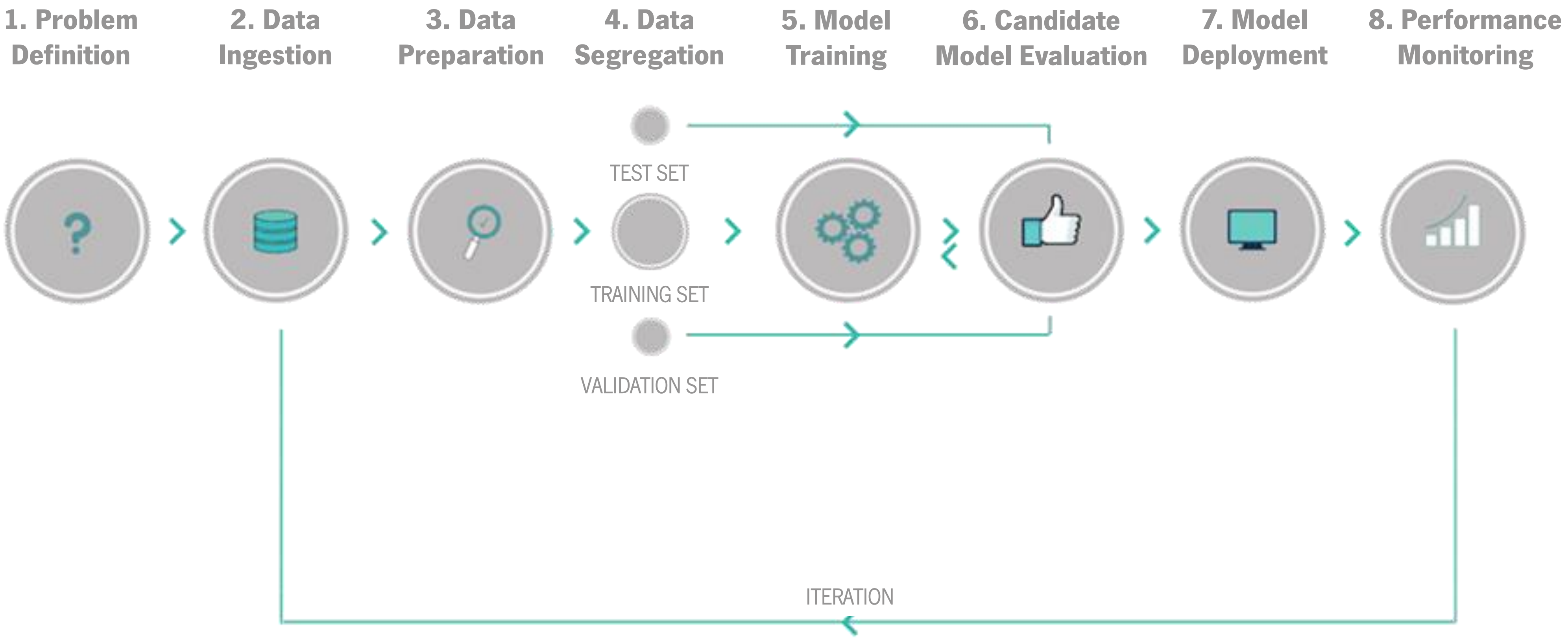
Interpretability and Explainability

DAA @ MEI-1º/MiEI-4º – 1º Semestre

Bruno Fernandes, Dalila Alves, Filipa Ferraz, Victor Alves

Part VII

Contents


1. Problem Definition
2. Data Ingestion
3. Data Preparation
4. Data Segregation
5. Model Training
6. Candidate Model Evaluation
7. Model Deployment
8. Performance Monitoring
TEST SET
TRAINING SET
VALIDATION SET
ITERATION

Interpretability and Explainability

Interpretability and Explainability

How we can understand the cause of the ML model's decision

WHY

Helps to understand the behavior of the model

How to explain the model behavior *post hoc*

HOW

Helps understand the model's decision and its underlying mechanics and features

Both provide transparency into automated decisions, fostering trust, accountability, and the ability to debug algorithms when necessary.

The Problem and the Data

Dataset: table with information regarding the **passengers** with 891 entries and 12 features, including:

- *PassengerId*
- *Survived*
- *Pclass*
- *Name*
- *Sex*
- *Age*
- *SibSp*
- *Parch*
- *Ticket*
- *Fare*
- *Cabin*
- *Embarked*

Model: Machine Learning model capable of **predicting the passenger survived** to the Titanic disaster

Questions: Why was a particular passenger predicted to survive or not? How can we explain the model’s decision?

The Problem and the Data

Dataset: table with information regarding the **passengers** with 891 entries and 12 features, including:

- *PassengerId*
- *Survived*
- *Pclass*
- *Name*
- *Sex*
- *Age*
- *SibSp*
- *Parch*
- *Ticket*
- *Fare*
- *Cabin*
- *Embarked*

You may need to install `shap`. Use one of the following commands:

```
pip install shap
```

```
conda install -c conda-forge shap
```

Model: Machine Learning model capable of **predicting the passenger survived** to the Titanic disaster

Questions: Why was a particular passenger predicted to survive or not? How can we explain the model’s decision?

Feature Engineering and EDA

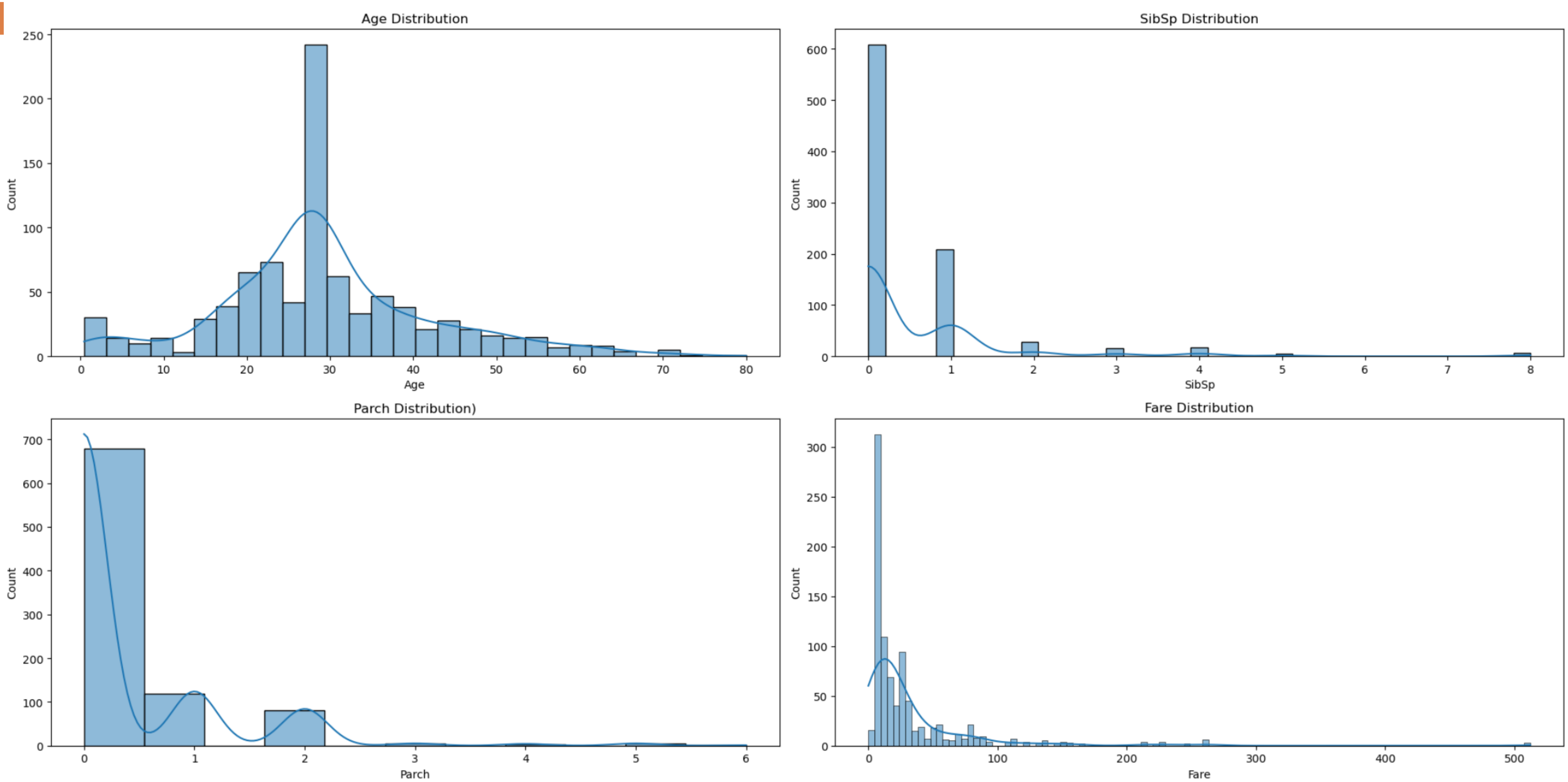

```
df.info()
```

```
<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
```

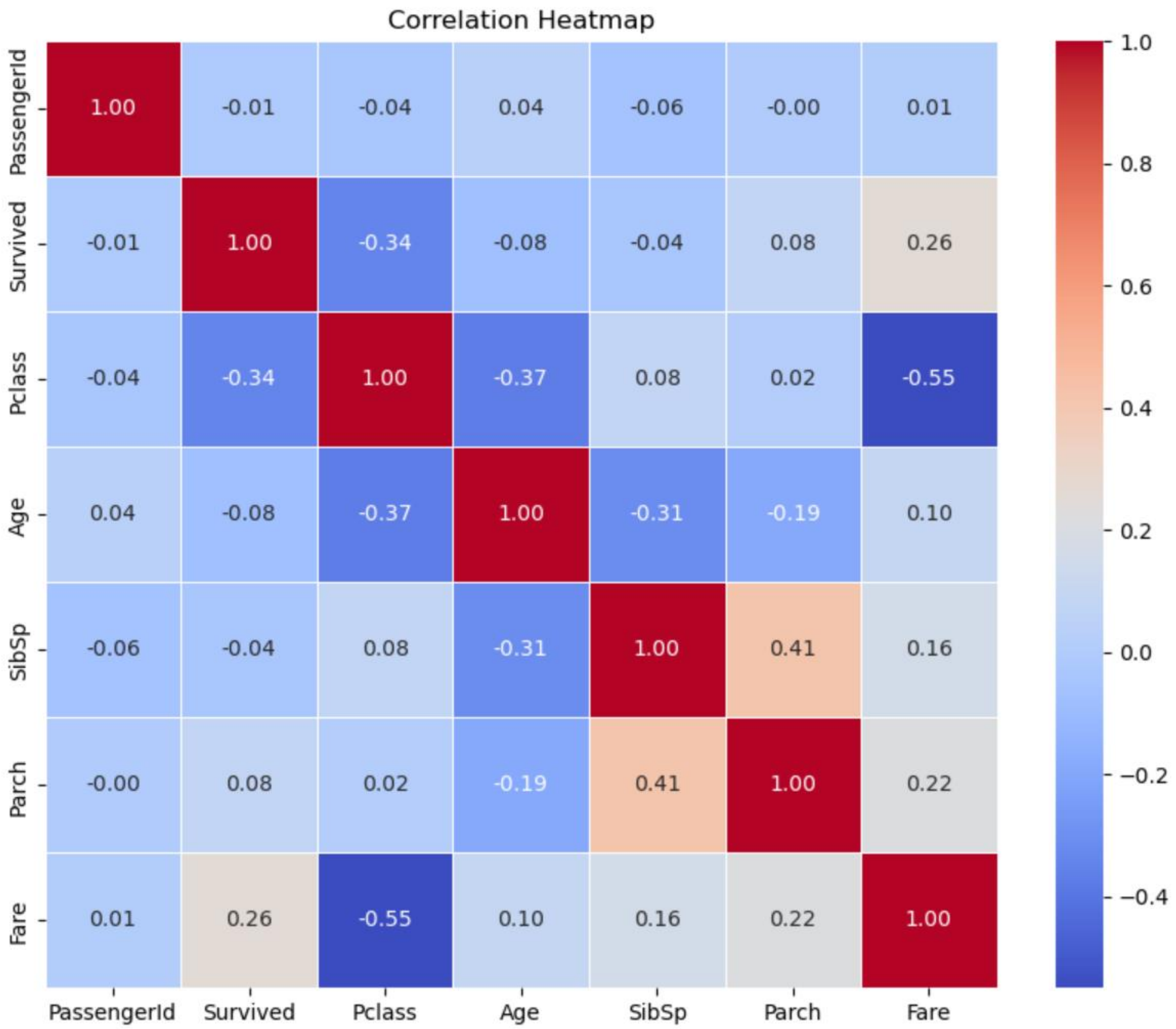
```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S

Feature Engineering and EDA

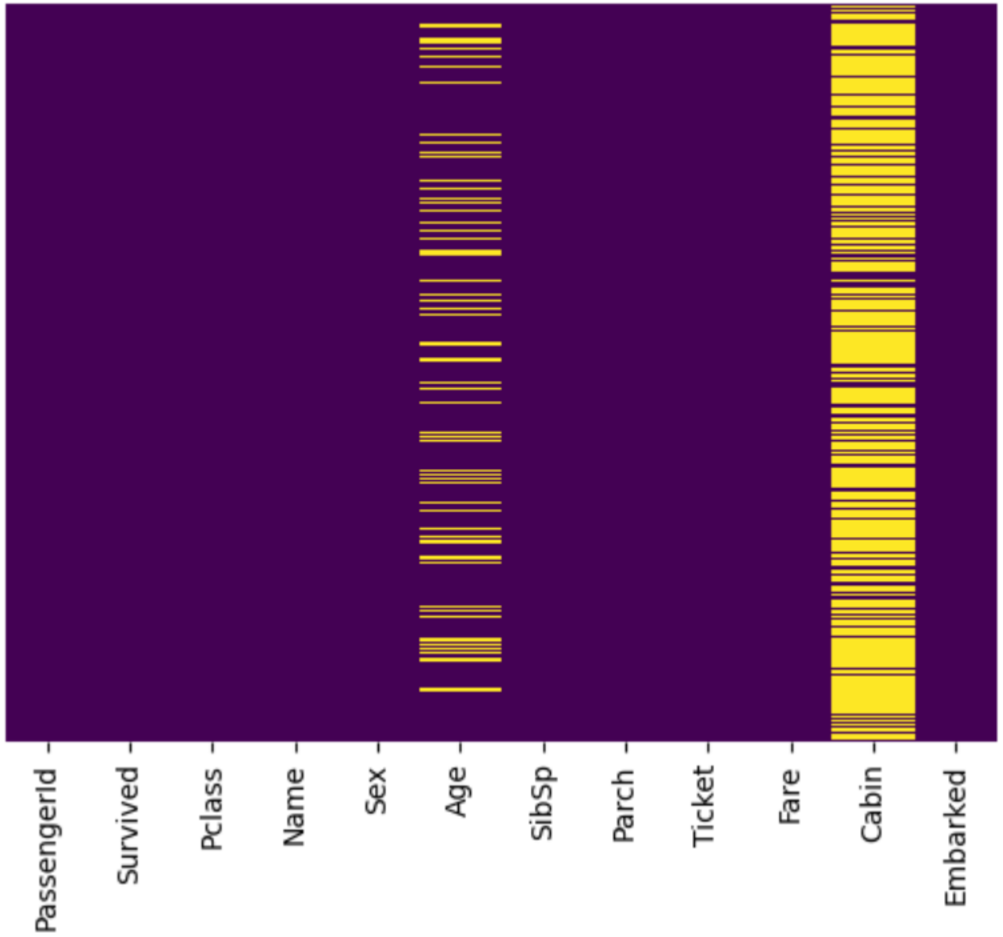


Feature Engineering and EDA



Feature Engineering and EDA

Analyzing the missing values:




```
df.isnull().sum()
```

```
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
```

Feature Engineering and EDA

Feature *Age* – let's impute values using *median*:

```
def med_impute_nan(df):
    med_impute = df.copy()
    med_impute["Age"] = med_impute["Age"].fillna(med_impute["Age"].median())
    return med_impute
```

```
med_impute = med_impute_nan(df)
```

```
med_impute.isnull().sum()
```

```
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
```

Feature Engineering and EDA

Feature *Age* – let's impute values using *median:*

```
med_impute.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
print(df['Age'].std())
print(med_impute['Age'].std())
```

```
14.526497332334044
13.019696550973194
```

Feature Engineering and EDA

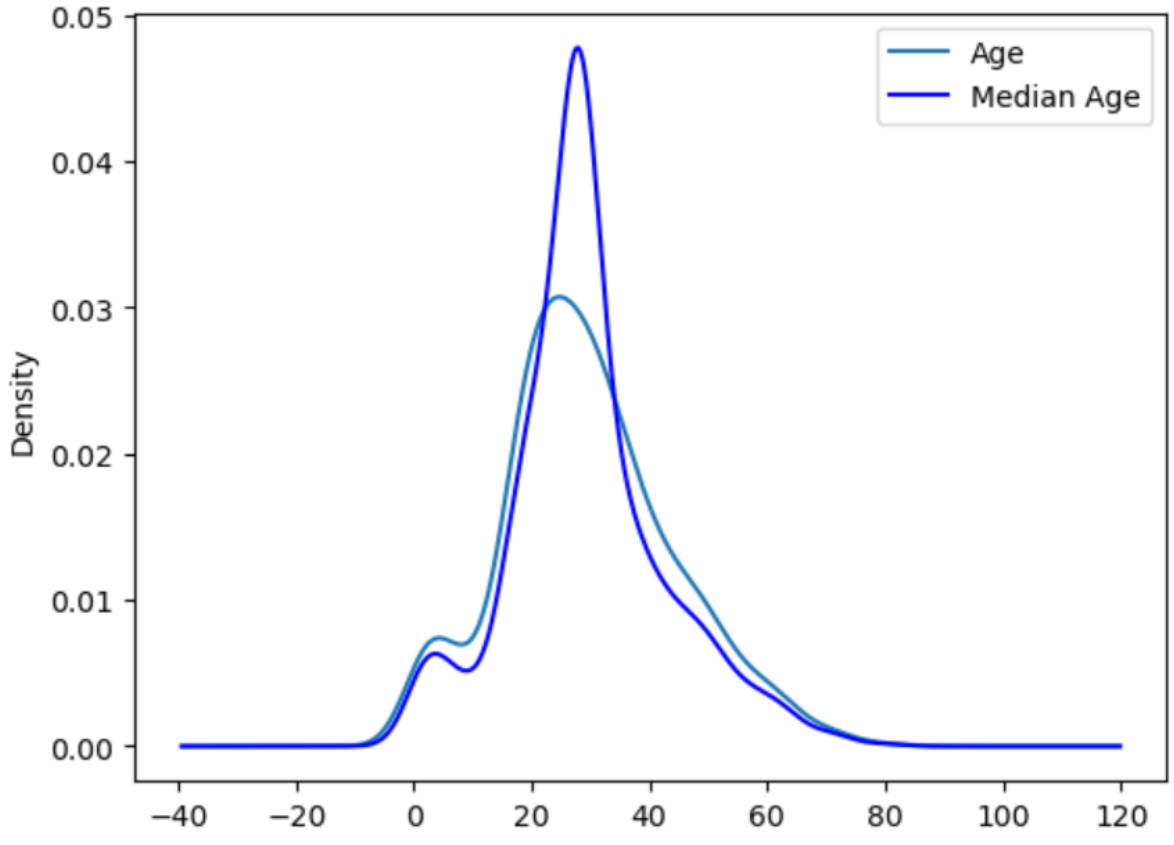
Feature *Age* – let's impute values using *median:*

```
df = med_impute
df.isnull().sum()
```

```
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
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
df['Age'].plot(kind='kde', ax=ax)
med_impute['Age'].plot(kind='kde', label='Median Age', ax=ax, color='blue')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

```
<matplotlib.legend.Legend at 0x1b6fd871220>
```



Feature Engineering and EDA

Feature *Embarked* – let's deal with `NaN` values:

```
df.Embarked.value_counts()
```

```
Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

```
emabark = df['Embarked'].dropna()
```

```
df[df['Embarked'].isnull()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN

```
df['Embarked'].mode()
```

```
0    S
Name: Embarked, dtype: object
```

Feature Engineering and EDA

Feature *Embarked* – let's deal with **NaN** values:

```
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
df.isnull().sum()
```

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

Feature Engineering and EDA

Feature *Cabin* – let's deal with `NaN` values:

```
df['Cabin'].value_counts()
```

```
Cabin
B96 B98        4
G6             4
C23 C25 C27    4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: count, Length: 147, dtype: int64
```

```
df['Cabin'].mode()
```

```
0        B96 B98
1    C23 C25 C27
2             G6
Name: Cabin, dtype: object
```

```
df['Cabin'].fillna(df['Cabin'].mode()[0], inplace=True)
```

Feature Engineering and EDA


```
df.isnull().sum()
```

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

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	B96 B98	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	B96 B98	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	B96 B98	S

Feature Engineering and EDA

Feature *Sex* – let's factorize: `male` = 1, `female` = 0:

```
df['Sex'] = df['Sex'].apply(lambda x: 1 if x == 'male' else 0)
df['Sex']
```

```
0      1
1      0
2      0
3      0
4      1
      ..
886    1
887    0
888    0
889    1
890    1
Name: Sex, Length: 891, dtype: int64
```

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	B96 B98	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	B96 B98	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	B96 B98	S

Feature Engineering and EDA

Features *Name*, *Ticket* and *Cabin* – let's drop:

```
df.drop(columns=['Name', 'Ticket', 'Cabin'], inplace=True)
```

```
df.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	S
1	2	1	1	0	38.0	1	0	71.2833	C
2	3	1	3	0	26.0	0	0	7.9250	S
3	4	1	1	0	35.0	1	0	53.1000	S
4	5	0	3	1	35.0	0	0	8.0500	S

Feature Engineering and EDA

Features *Embarked* – let's encode the values `S`, `C`, and `Q` using `LabelEncoder`:

```
print(df["Embarked"].value_counts())
```

```
Embarked
S    646
C    168
Q     77
Name: count, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
cols = ['Embarked']
le = LabelEncoder()

for col in cols:
    df[col] = le.fit_transform(df[col])
df.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	1	22.0	1	0	7.2500	2
1	2	1	1	0	38.0	1	0	71.2833	0
2	3	1	3	0	26.0	0	0	7.9250	2
3	4	1	1	0	35.0	1	0	53.1000	2
4	5	0	3	1	35.0	0	0	8.0500	2

Feature Engineering and EDA

Correlation analysis of the features:

```
correlation_matrix = df.corr()
correlation_with_target = correlation_matrix['Survived'].sort_values(ascending=False)
print("Correlation with target (Survived):\n", correlation_with_target)
```

```
Correlation with target (Survived):
 Survived       1.000000
Fare           0.257307
Parch          0.081629
PassengerId   -0.005007
SibSp         -0.035322
Age           -0.064910
Embarked      -0.167675
Pclass        -0.338481
Sex           -0.543351
Name: Survived, dtype: float64
```

In this case, *Sex*, *Pclass*, and *Fare* have the <u>highest absolute correlation</u> values with *Survived*, suggesting that they may be useful for prediction.

Feature Engineering and EDA

We are going to save the new data frame into a new file:

```
# Convert data to DataFrame
t = pd.DataFrame(df)

# Specify the CSV file name
filename = "titanic_ds.csv"

# Save to CSV
t.to_csv(filename, index=False, encoding='utf-8')
```

Random Forest Classifier

Let's split the data:

```
X = df.drop('Survived', axis=1)
y = df['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 2022)
```

Train the model:

```
rf_model = RandomForestClassifier()
```

```
rf_model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier

RandomForestClassifier()
```

Random Forest Classifier

Obtain the accuracy of the model:

```
rf_score = rf_model.score(X_test, y_test)
print("Accuracy: %.2f%%" % (rf_score * 100))
```

```
Accuracy: 82.09%
```

Save the predictions into a file:

```
op_rf = rf_model.predict(X_test)
```

```
op = pd.DataFrame(X_test['PassengerId'])
op['Survived'] = op_rf
op.to_csv("submission.csv", index=False)
```

Feature Importance

- Measures the contribution of each feature to the model's predictive performance, revealing which attributes have the greatest impact on its results

- **Global** Feature Importance vs. **Local** Feature Importance:
- **Model agnostic** vs. **model specific**

- Several techniques: permutation importance, tree-based importance scores, SHAP values

- Feature importance is fundamental for optimization and model refinement, guiding the selection of relevant features to improve predictive accuracy and model efficiency.

Random Forest Feature Importance

Feature importance analysis reveals the impact of factors like age, gender, and class on survival rates in the Titanic dataset.

We will use the `time` class to assess each FI type:

```
import time

start_time = time.time()

rf_importances = rf_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf_model.estimators_], axis=0)

elapsed_time = time.time() - start_time

print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds")
```

```
Elapsed time to compute the importances: 0.006 seconds
```

Obtaining the feature importances values:

```
print("Random Forest Feature Importances:\n", rf_importances)
```

```
Random Forest Feature Importances:
 [0.18566889 0.09170101 0.22728848 0.17159673 0.04398073 0.04376958
 0.20277537 0.0332192 ]
```

What do these values mean?

Feature Importance based on Mean Decrease in Impurity (MDI)


```
start_time = time.time()

mdi_importances = pd.Series(rf_model.feature_importances_, index=X_test.columns)

elapsed_time = time.time() - start_time

print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds")
```

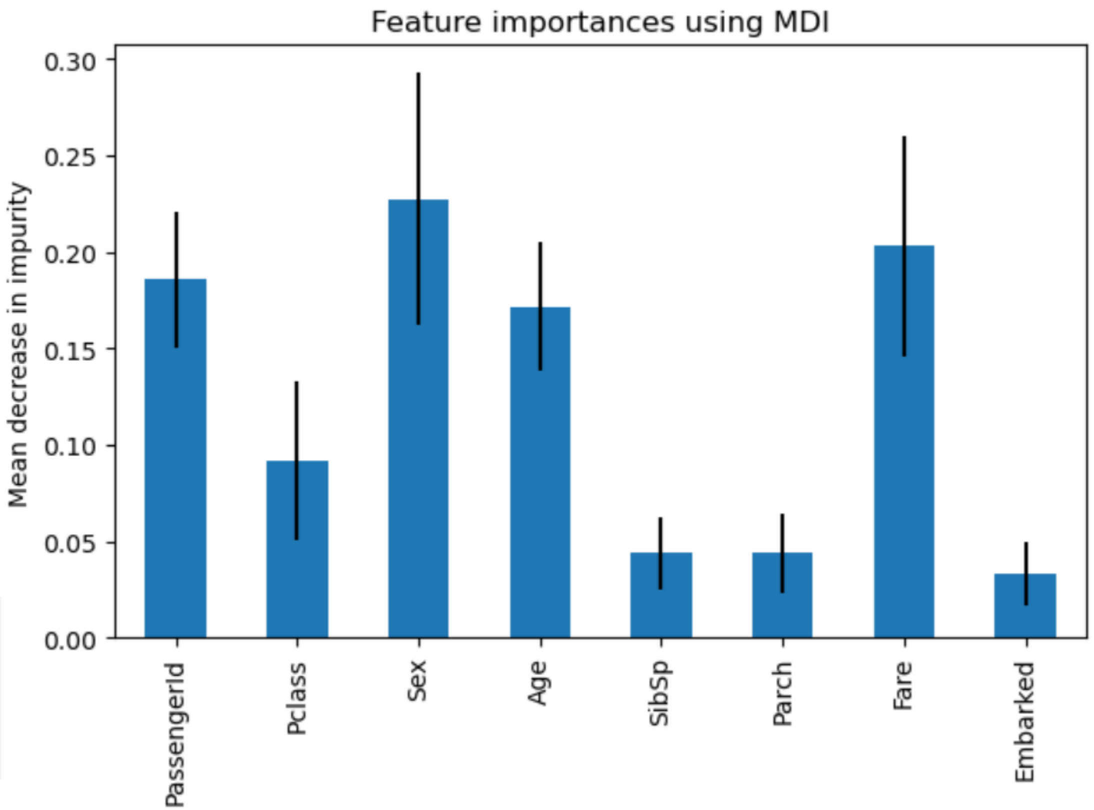
```
Elapsed time to compute the importances: 0.006 seconds
```

Obtaining the FI values:

```
print("Feature importances using MDI:\n", mdi_importances)
```

```
Feature importances using MDI:
 PassengerId    0.185669
Pclass         0.091701
Sex            0.227288
Age            0.171597
SibSp          0.043981
Parch          0.043770
Fare           0.202775
Embarked       0.033219
dtype: float64
```

```
fig, ax = plt.subplots()
mdi_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



Feature Importance based on Permutation Importance


```
from sklearn.inspection import permutation_importance

start_time = time.time()

result = permutation_importance(rf_model, X_test, y_test, n_repeats=10, random_state=42, n_jobs=2)

elapsed_time = time.time() - start_time
print(f"Elapsed time to compute the importances: {elapsed_time:.3f} seconds")
```

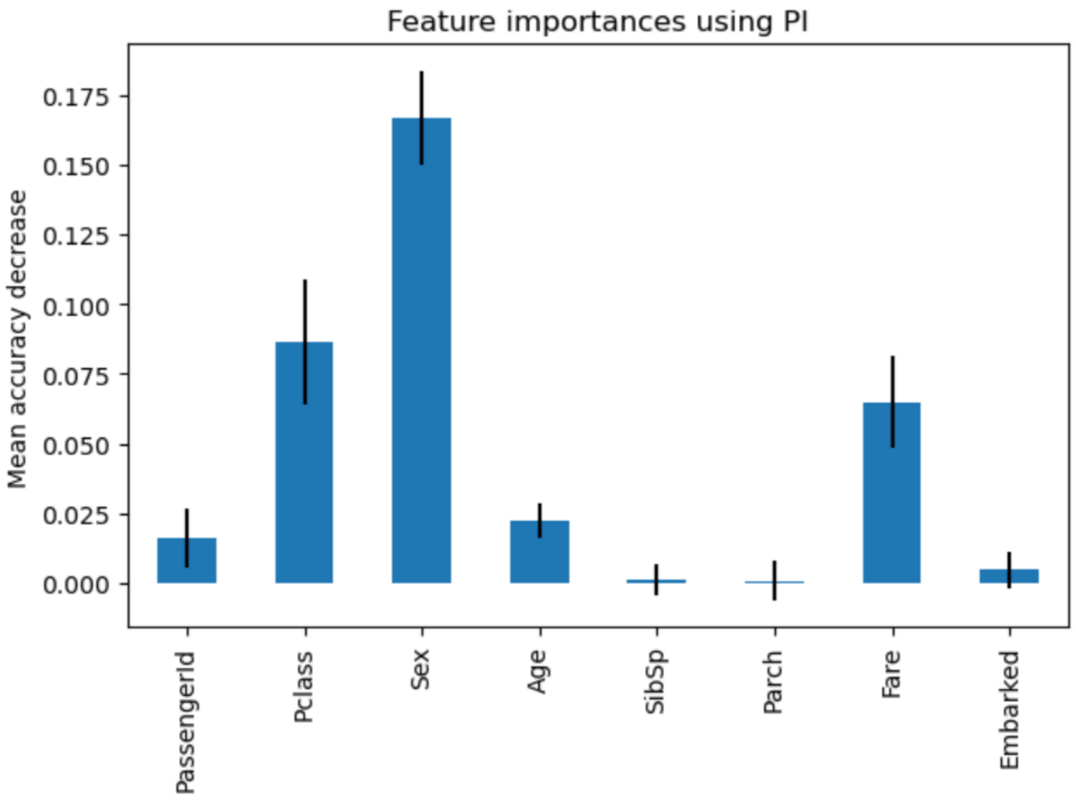
```
Elapsed time to compute the importances: 0.240 seconds
```

Obtaining the FI values:

```
p_importances = pd.Series(result.importances_mean, index=X_test.columns)
print("Feature importances using PI:\n", p_importances)
```

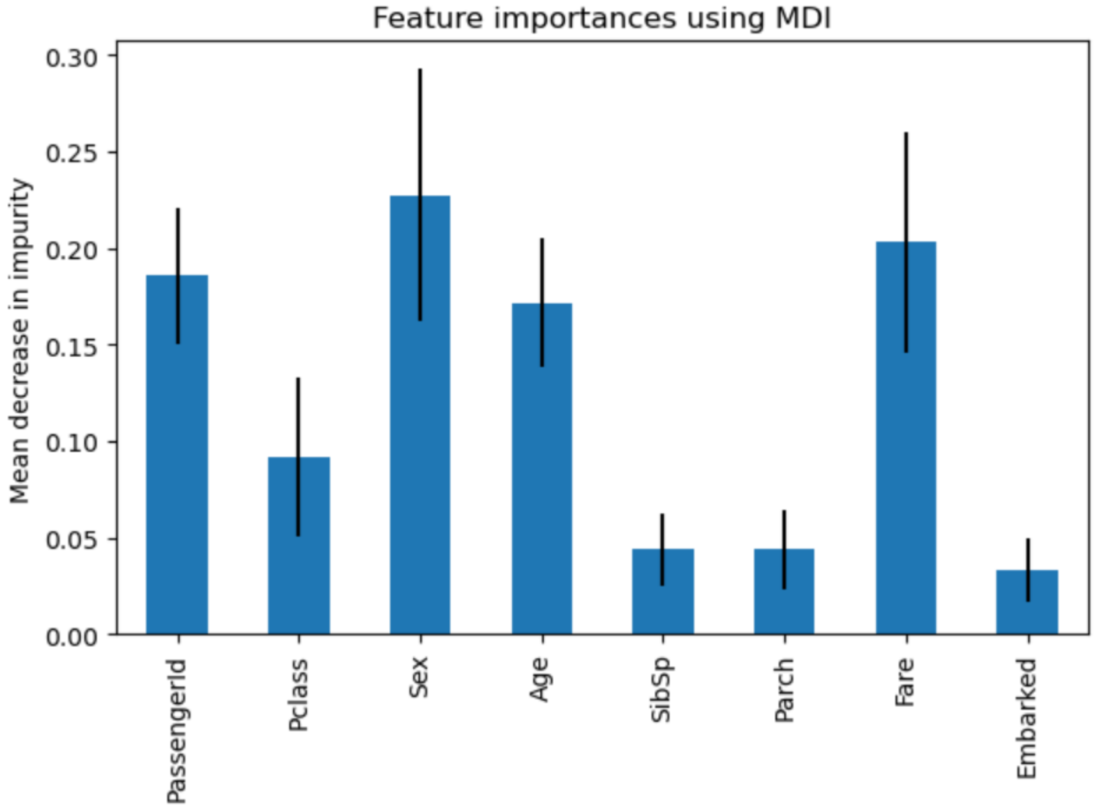
```
Feature importances using PI:
 PassengerId    0.016045
Pclass         0.086194
Sex            0.166791
Age            0.022388
SibSp          0.001119
Parch          0.000746
Fare           0.064925
Embarked       0.004851
dtype: float64
```

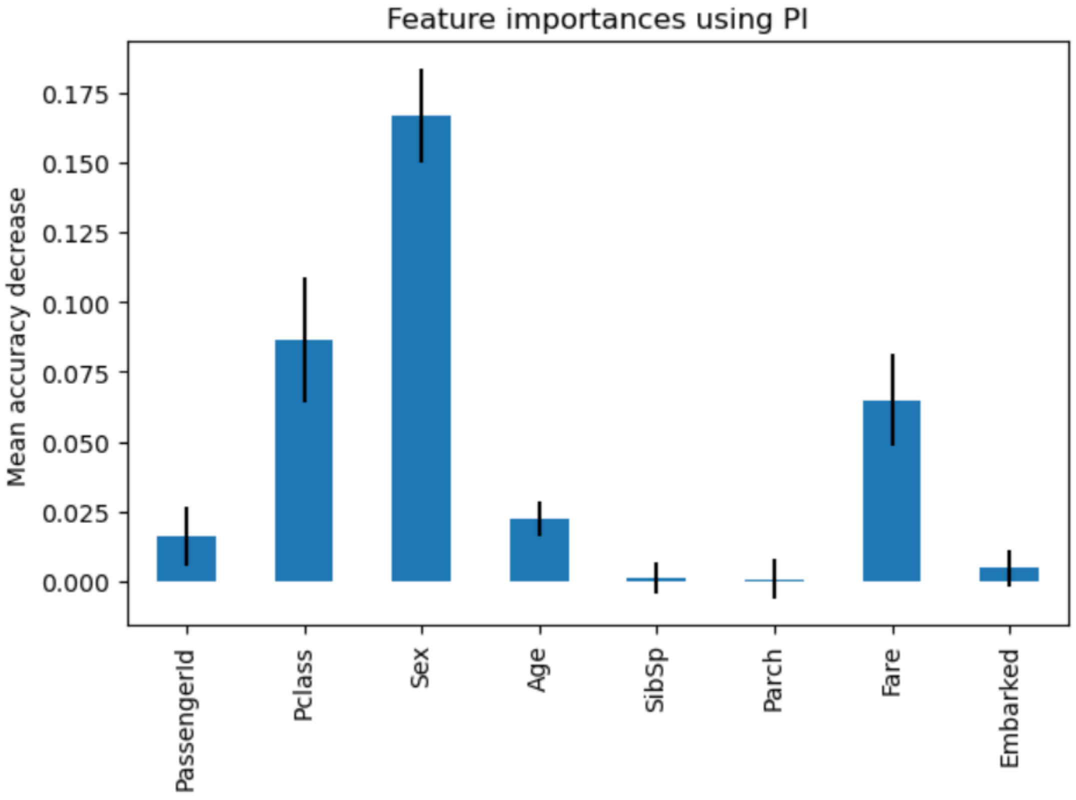
```
fig, ax = plt.subplots()
p_importances.plot.bar(yerr=result.importances_std, ax=ax)
ax.set_title("Feature importances using PI")
ax.set_ylabel("Mean accuracy decrease")
fig.tight_layout()
plt.show()
```



Feature Importance

Which features have more importance?





Model specific

Model agnostic

SHAP (SHapley Additive exPlanations) Analysis

- SHAP values, based on game theory to fairly distribute importance among features, provide a unified framework for interpreting the output of any ML model. By representing each feature's contribution to a model's output, SHAP values enhance our ability to interrogate and validate the decision-making process within any predictive model.
- Provide accurate and consistent explanations for both global and local predictions

- **Global interpretation**: overall importance of features across all predictions
- **Local interpretation**: why a specific passenger was predicted to survive or not

- Benefits:
  - Broader view of feature contributions by attributing influence based on the distribution of feature values rather than a simple average or count in traditional methods;
  - Fairness and comprehensiveness of SHAP scores provide better insight, help mitigate bias and ensure that the effects of interactions between features are effectively captured.

SHAP Analysis – Local Interpretation

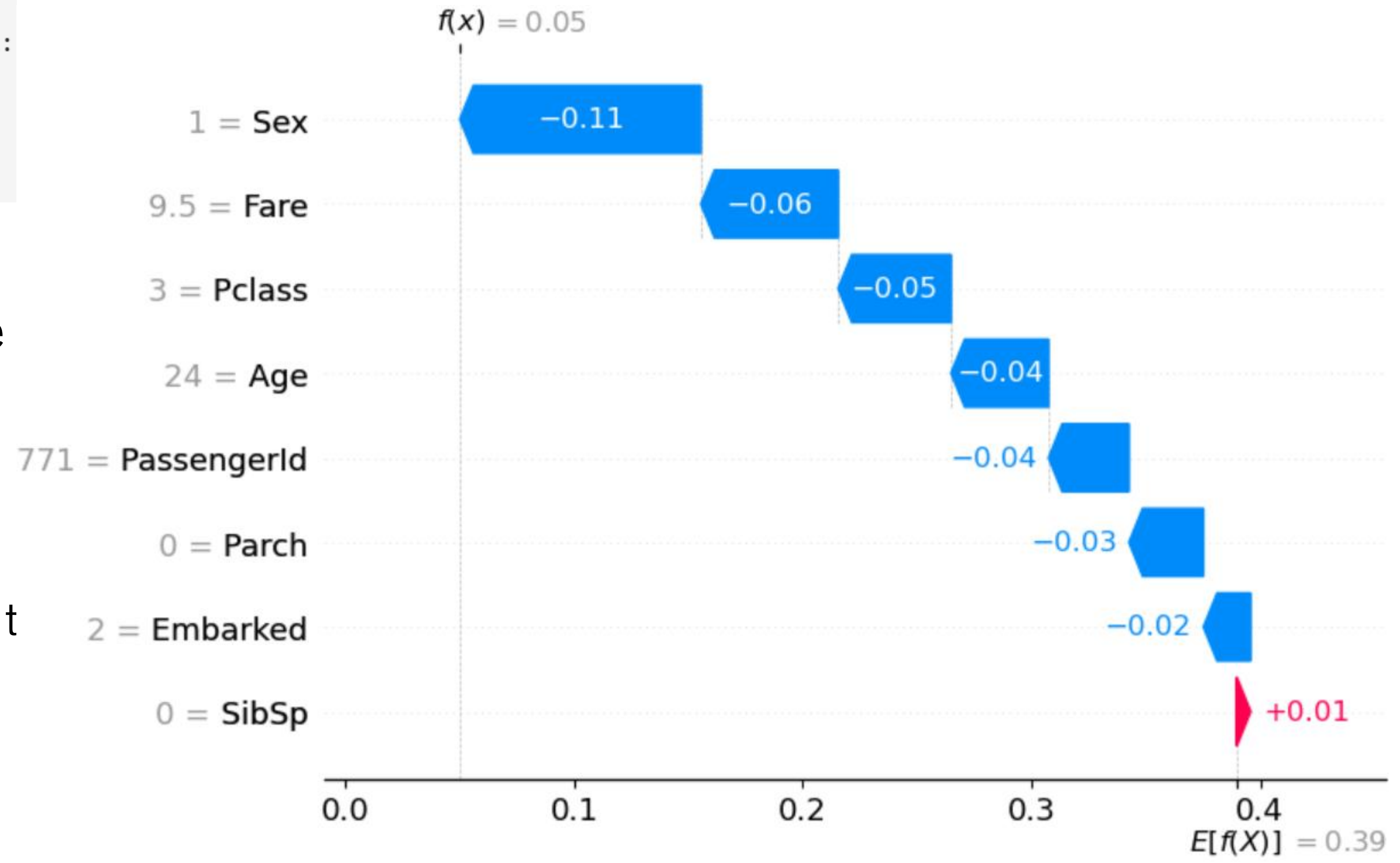
Local interpretability can be explained using the Titanic dataset. Let’s understand why a specific passenger, ID = 0, didn’t survived:

```
no = 0
if rf_model.predict(np.expand_dims(X_test.iloc[no],axis=0))[0] == 1:
    print("The passenger survived")
else:
    print("The passenger did not survive")
shap.plots.waterfall(shap_values[no,:,1])
```

```
The passenger did not survive
```

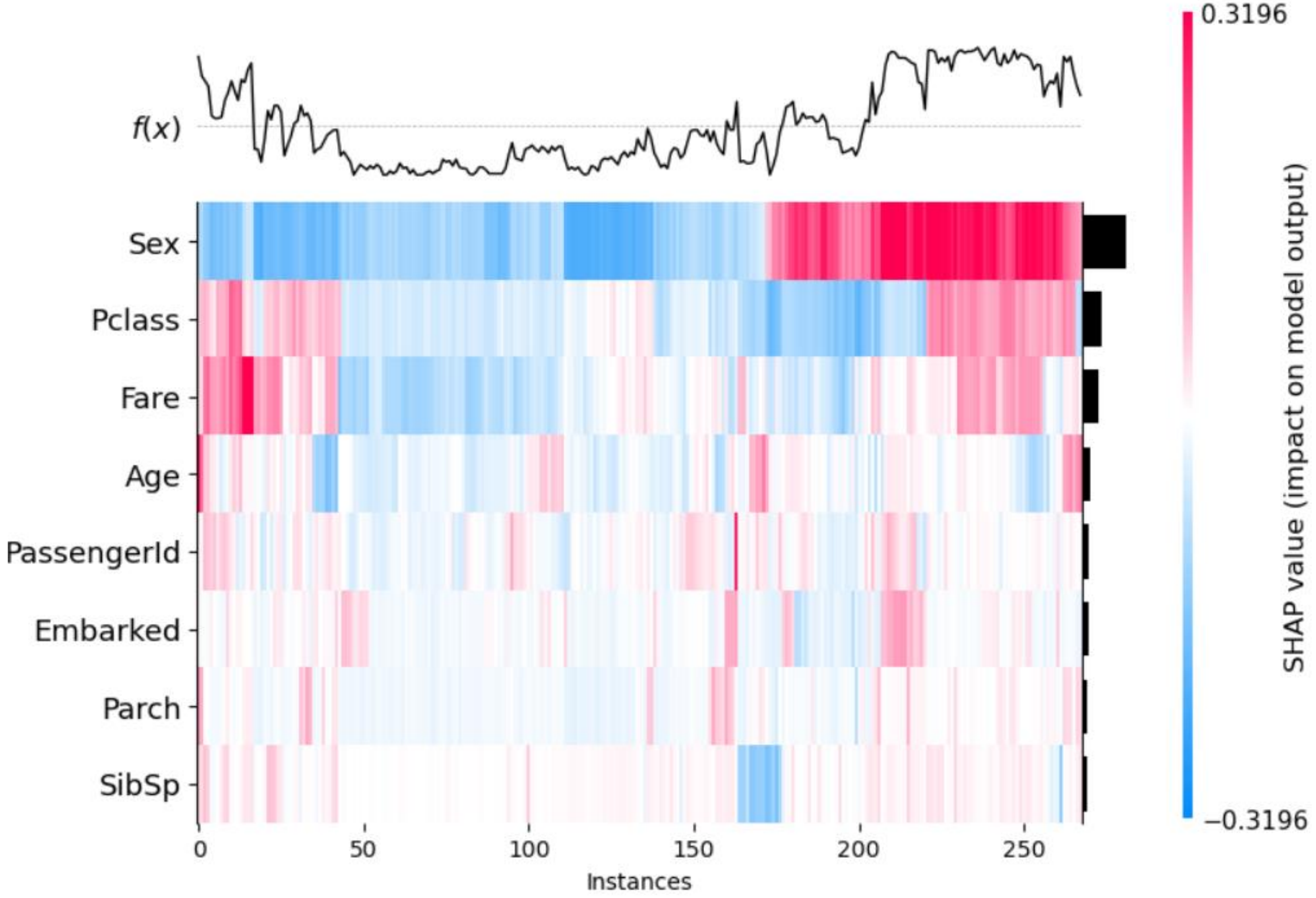
Contribution of each feature to the survival of the

- Sex = 1 – being male
- Fare = 9.5 – low fare
- Pclass = 3 – 3rd class
- Age = 24 – being young
- PassengerID = 771 – being among the last board t
- Parch = 0 – having no parent/children aboard
- Embarked = 2 – port of embarking 2
- SibSp = 0 – having no siblings/spouse aboard



SHAP Analysis – Global Interpretation


```
shap.plots.heatmap(shap_values[:,:,1])
```



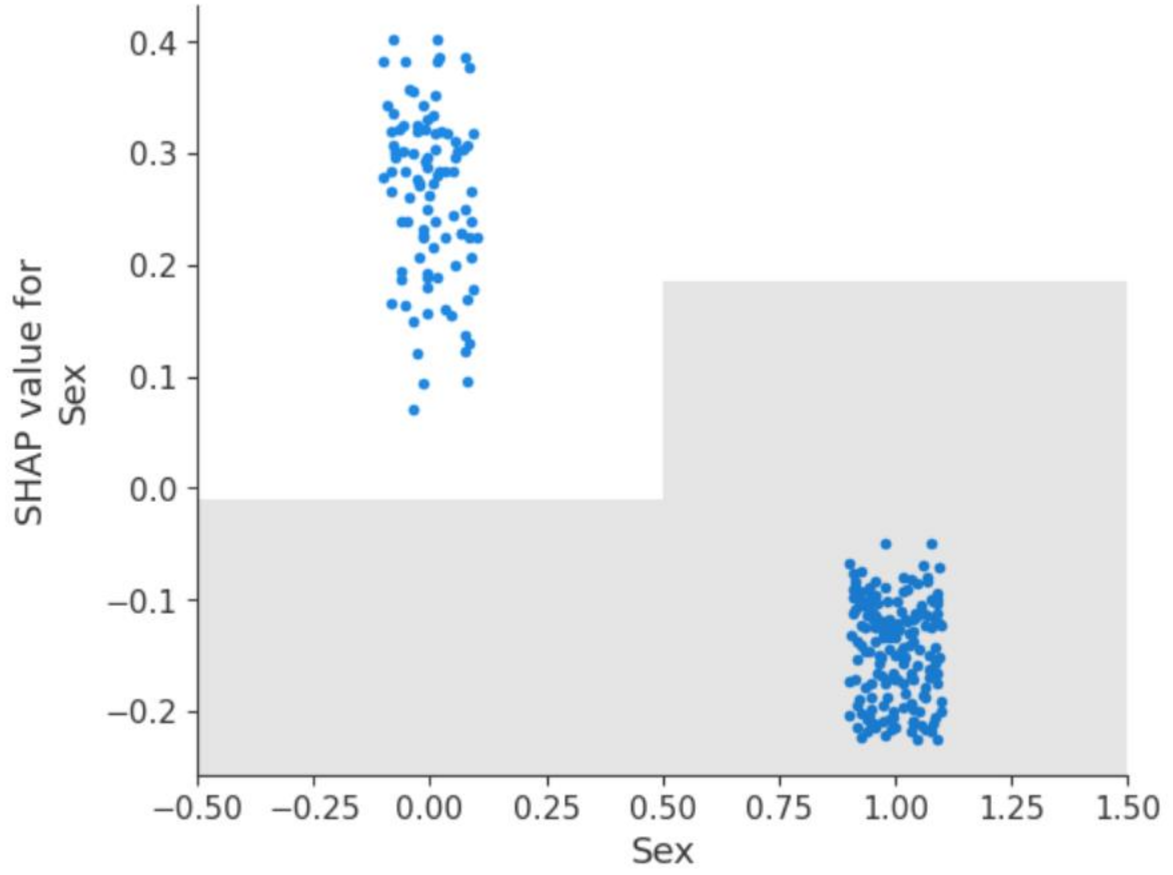
SHAP Analysis – Feature Wise

This tells us how each feature globally contributed to the model prediction.

Sex

- Being a male (1) reduced the chances of survival

```
shap.plots.scatter(shap_values[:,"Sex",1])
```

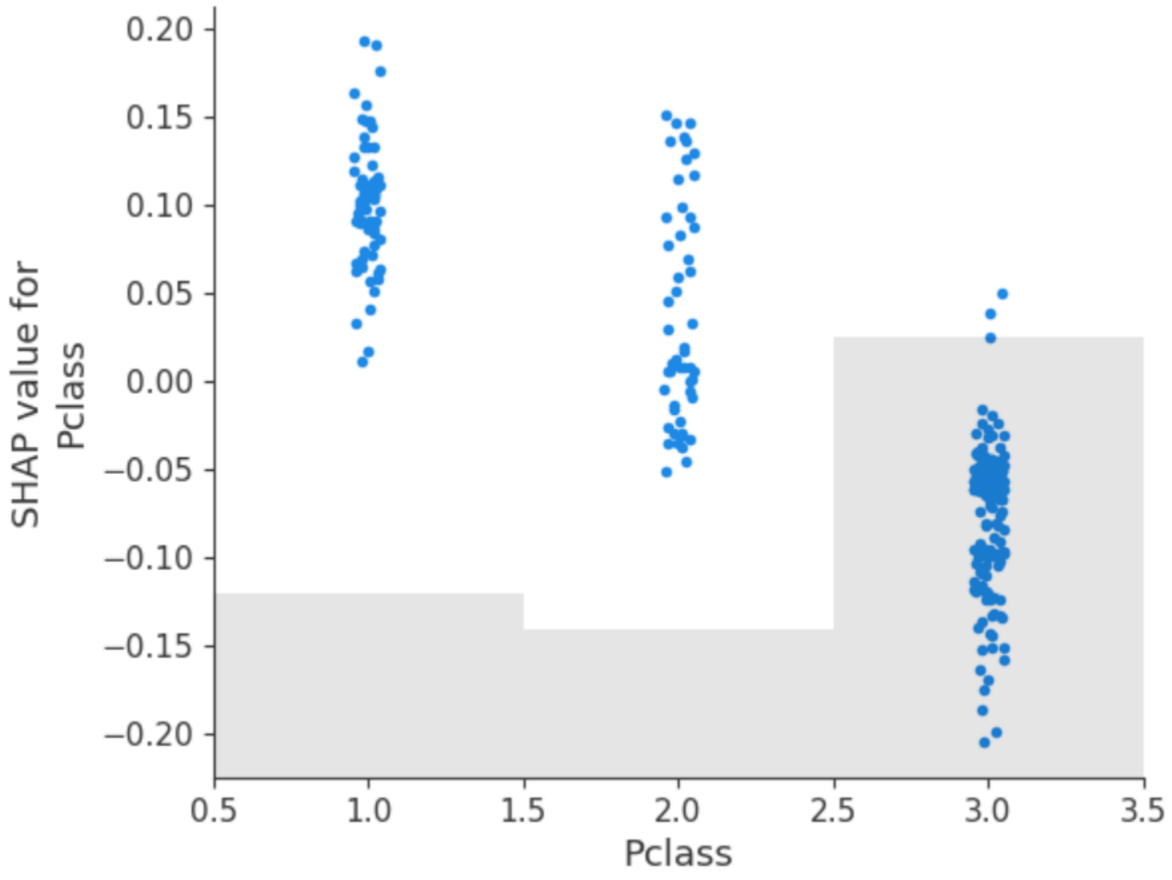


SHAP Analysis – Feature Wise

Pclass

- Being in the 3rd class (3) reduced the chances of survival
- Being in the 2nd class (2) contributed slightly positively to the survival

```
shap.plots.scatter(shap_values[:,"Pclass",1])
```

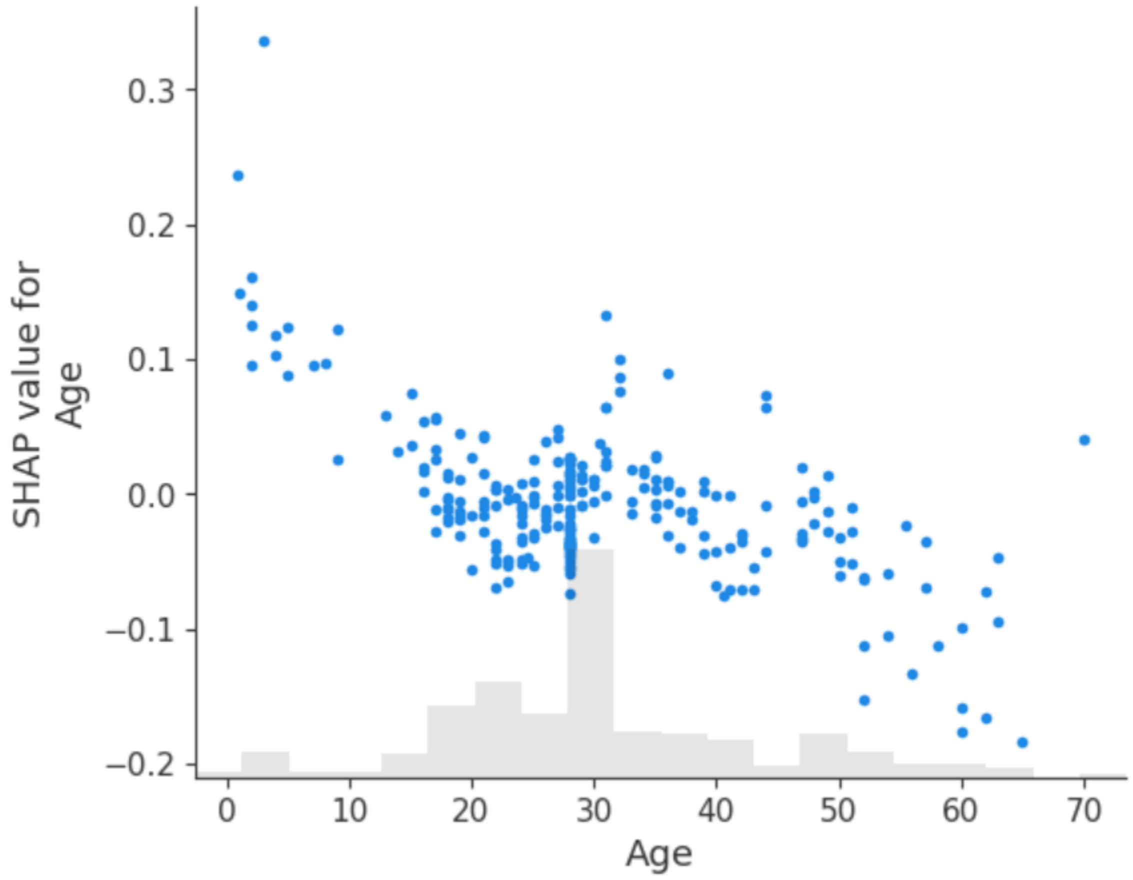


SHAP Analysis – Feature Wise

Age

- Being a child below 10 years-old contributed highly towards survival
- Being over 50 years-old contributed negatively to the survival

```
shap.plots.scatter(shap_values[:,"Age",1])
```

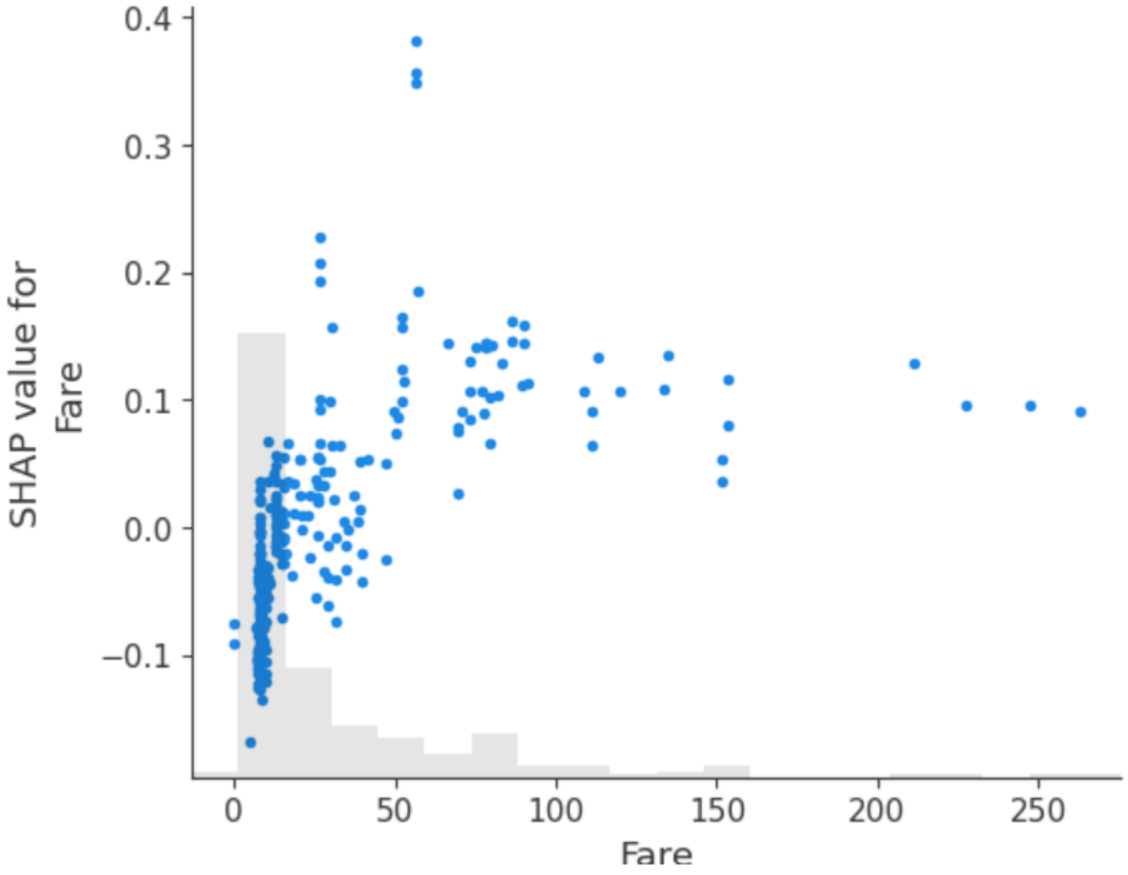


SHAP Analysis – Feature Wise

Fare

- A low fare contributed negatively towards survival
- A high fare (above 70) contributed positively towards survival

```
shap.plots.scatter(shap_values[:,"Fare",1])
```

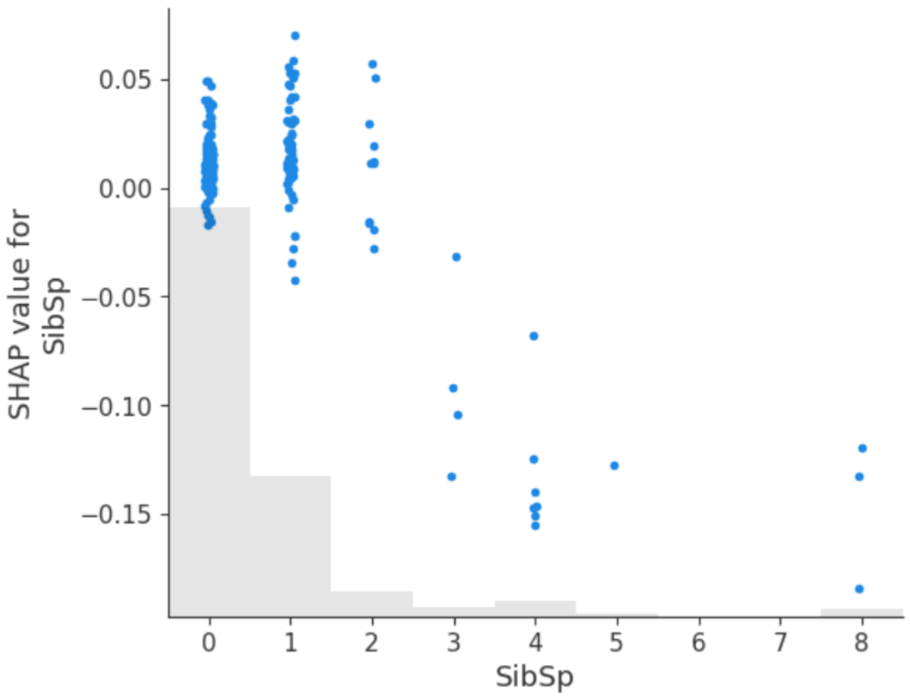


SHAP Analysis – Feature Wise

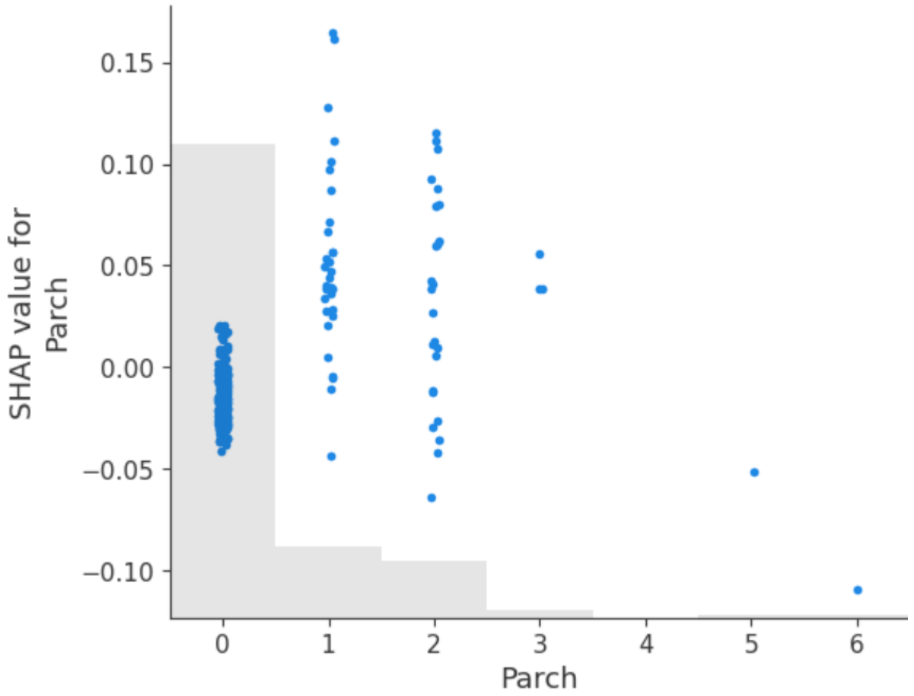
SibSp and **Parch**

- Higher sibling/spouse relation contributed negatively towards survival
- Having 0 or 1 sibling/spouse contribute positively towards survival
- Having 1 or 2 parent/children contribute slightly positively towards survival

```
shap.plots.scatter(shap_values[:,"SibSp",1])
```



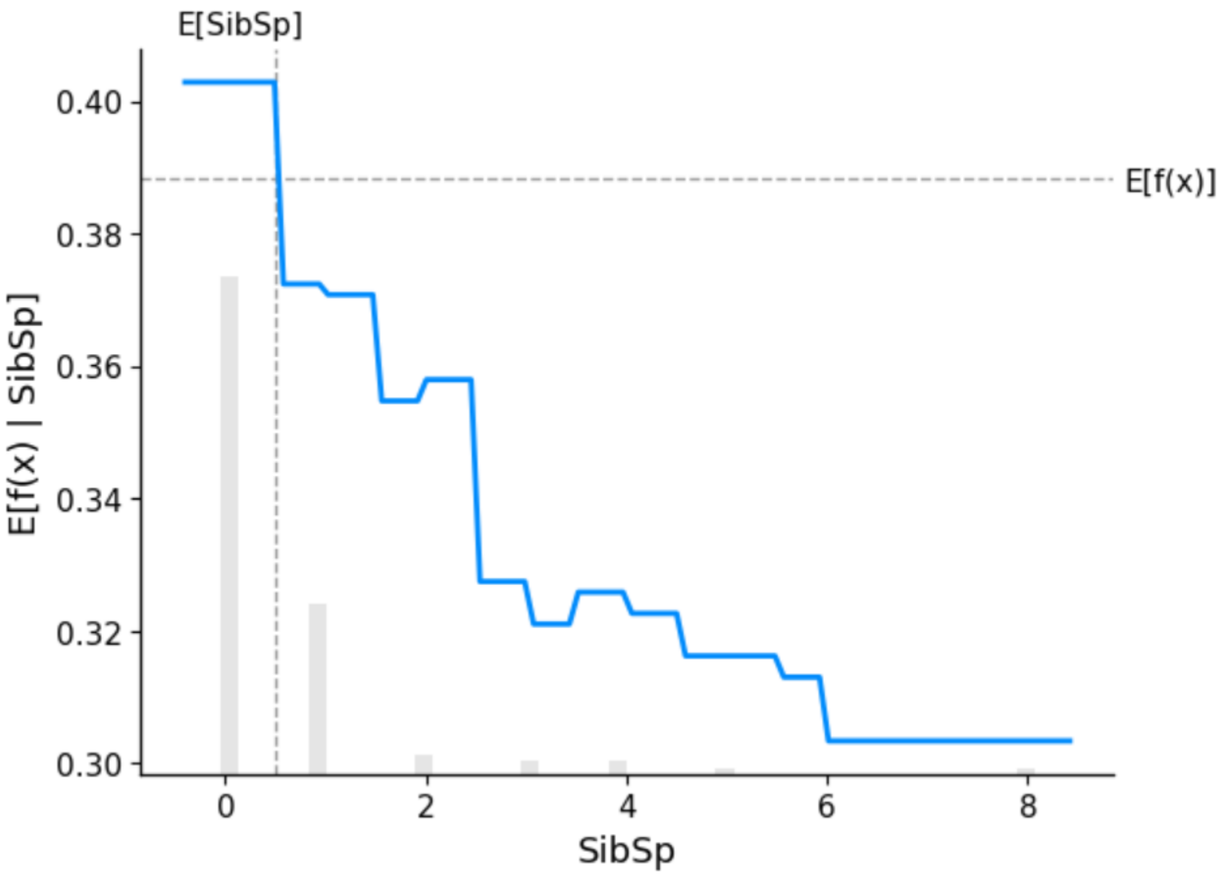

```
shap.plots.scatter(shap_values[:,"Parch",1])
```



SHAP Analysis – Feature Wise

Global interpretability is vital for comprehending overall model behavior. Using **Partial Dependence Plots (PDPs)** with the Titanic dataset, we can visualize how changes in *SibSp* influence the model's predictions:

```
shap.partial_dependence_plot("SibSp", rf_model.predict, X_train, ice=False, model_expected_value=True, feature_expected_value=True,)
```



SHAP Analysis – Linear Regression

Train the model:

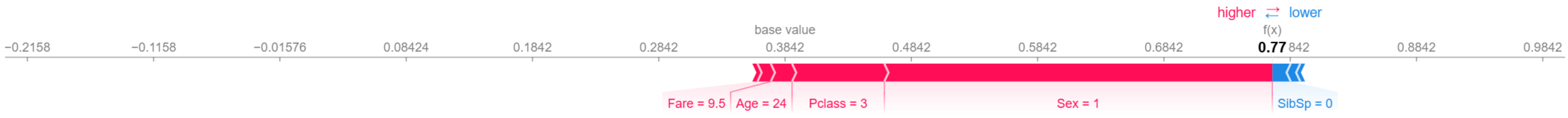
```
lm = linear_model.LinearRegression()
lm.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
explainer = shap.LinearExplainer(lm, X_train, feature_perturbation="interventional")
shap_values = explainer.shap_values(X_train)
```

For one passenger:

```
shap.force_plot(explainer.expected_value, shap_values[0,:], X_train.iloc[0,:])
```

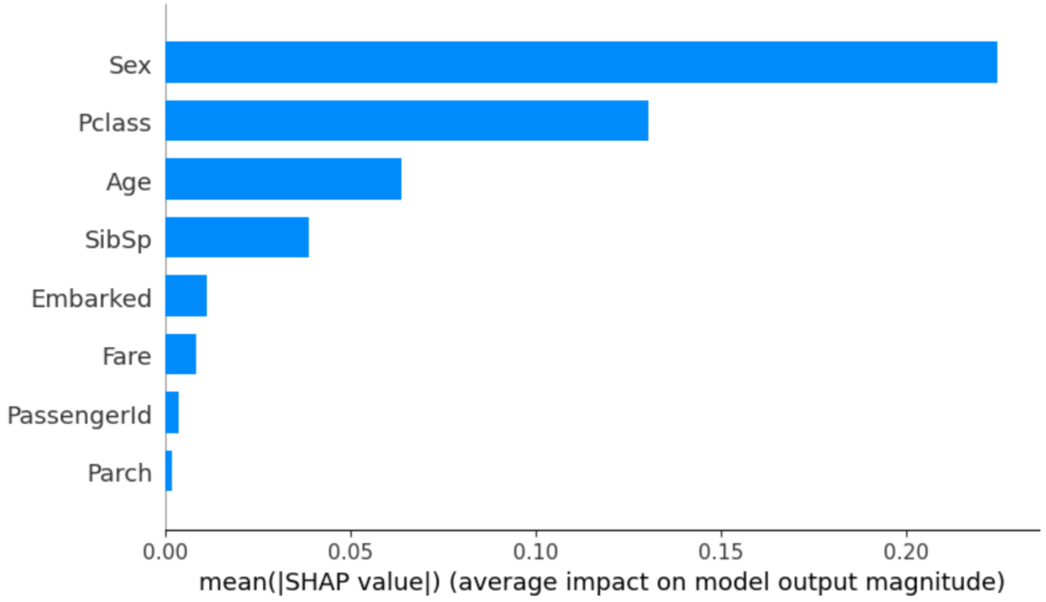


SHAP Analysis – Linear Regression

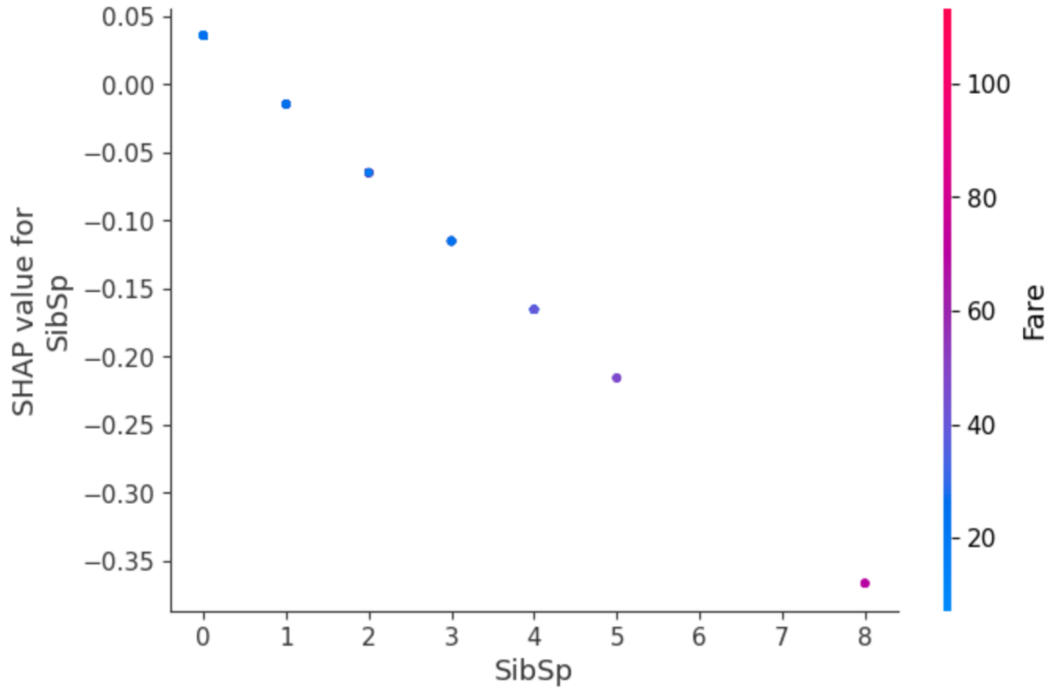
For all training data set:

```
explainer_shap = shap.LinearExplainer(model=lm, masker=X_train)
shap_values = explainer_shap.shap_values(X_train)
```

```
shap.summary_plot(shap_values, X_train, plot_type="bar")
```



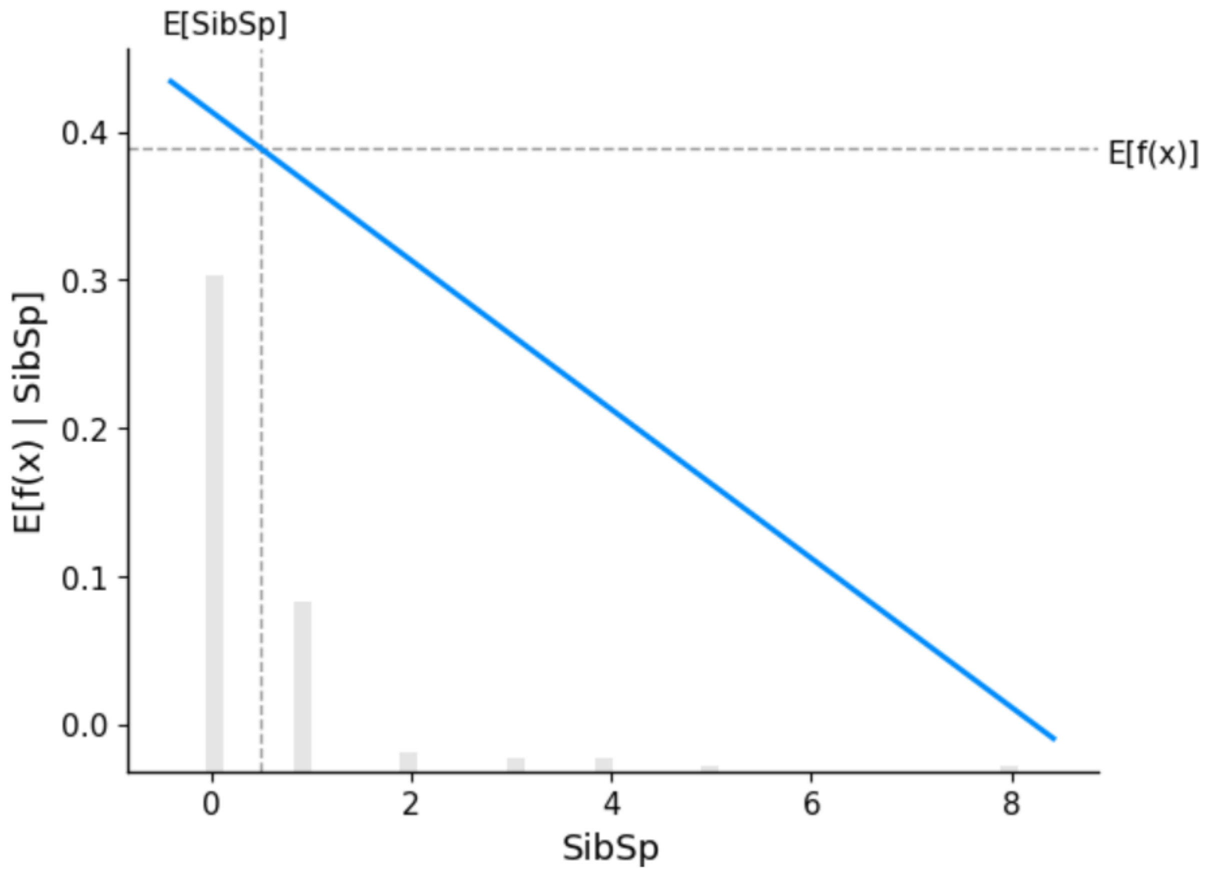

```
shap.dependence_plot("SibSp", shap_values, X_train)
```



SHAP Analysis – Feature Wise

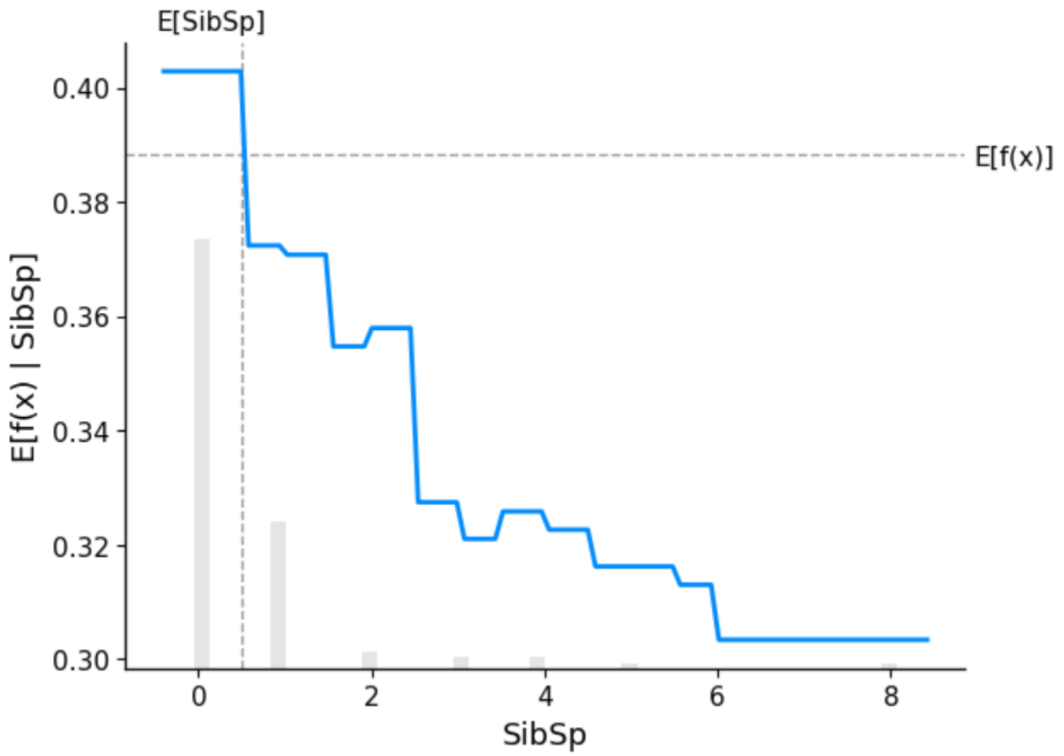
SibSp – Partial Dependence Plot

```
shap.partial_dependence_plot("SibSp", lm.predict, X_train, ice=False,
                             model_expected_value=True, feature_expected_value=True,)
```

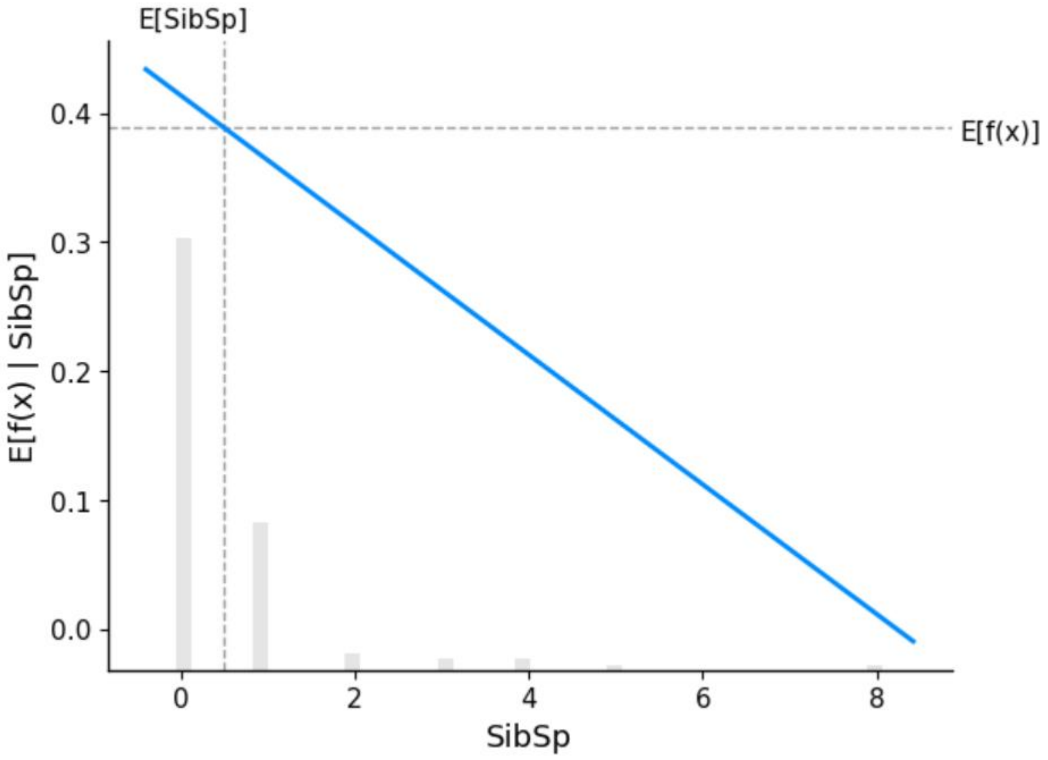


SHAP Analysis – Feature Wise

SibSp – Random Forest vs. Linear Regression



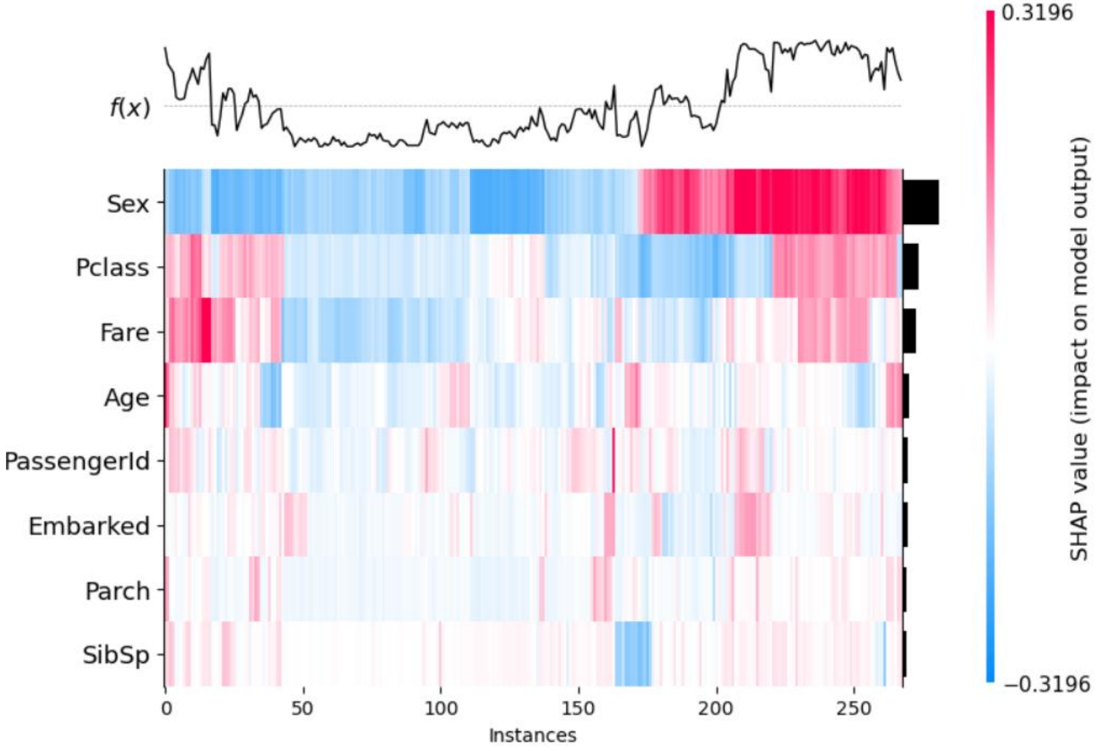
Random Forest



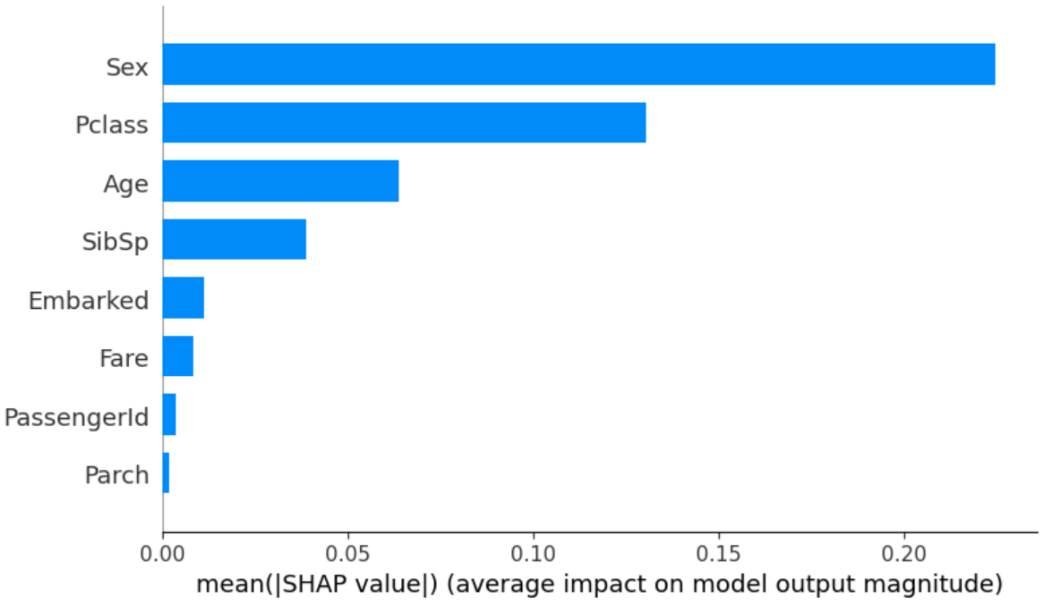
Linear Regression

SHAP Analysis – Random Forest vs. Linear Regression

Which features have more relevance to the model?



Random Forest



Linear Regression

Hands On