Machine Learning Workflow
Although the exact implementation details can vary, the general structure of a machine learning project stays relatively constant:
First need to define the project problem statement
1. Data cleaning and formatting
2. Exploratory data analysis
3. Feature engineering and selection
4. Establish a baseline and compare several machine learning models on a performance metric
5. Perform hyperparameter tuning on the best model to optimize it for the problem
6. Evaluate the best model on the testing set
7. Interpret the model results to the extent possible
8. Draw conclusions and write a well-documented report

1. Data cleaning and formatting

Reading Data
df_weather= pd.read_csv('New_York_Weather_2016.csv',parse_dates=['pickup_datetime'],usecols=["pickup_datetime" ,"icon"])

Data cleaning and formatting
#look at the data
df.head()

# See the column data types and non-missing values
data.info()

# count number of NA per column
df.isna().sum()

#create miss value percent tables
mis_val = df.isnull().sum()
# Percentage of missing values
mis_val_percent = 100 * df.isnull().sum() / len(df)
# Make a table with the results
mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

# Statistics for each column
data.describe()

#drop columns
data = data.drop(['Order', 'Property Id', 'Property Name', 'Parent Property Id', 'Parent Property Name', 'Address 1 (self-reported)', 'NYC Building Identification Number (BIN)', 'Address 2', 'NYC Borough, Block and Lot (BBL) self-reported', 'Street Name', 'BBL - 10 digits'], axis=1).reset_index(drop=True)

#remove rows
df_taxi = df_taxi[(df_taxi.RATECODE_ID!=99) & (df_taxi.RATECODE_ID!=6)]
#fill na with mean
sub2['income'] = sub2['income'].fillna((sub2['income'].mean()))
shotsLogDf = shotsLogDf.dropna(subset=columnsWithNa)

df_taxi.RATECODE_ID.value_counts()
RATECODE_ID values count:
1 15970395
5 349970
2 41959
3 12350

4 10094
6 289
99 3

Exploratory Data Analysis

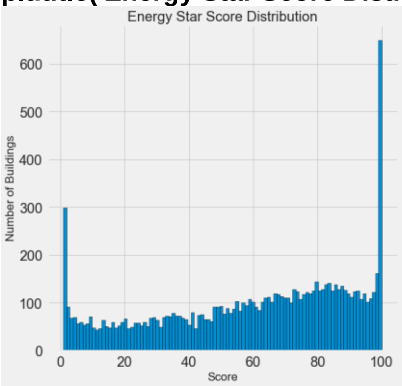
Exploratory Data Analysis (EDA) is an open-ended process where we make plots and calculate statistics in order to explore our data.

The purpose is to to find anomalies, patterns, trends, or relationships. These may be interesting by themselves (for example finding a correlation between two variables) or they can be used to inform modeling decisions such as which features to use.

In short, the goal of EDA is to determine what our data can tell us! EDA generally starts out with a high-level overview, and then narrows in to specific parts of the dataset once as we find interesting areas to examine.

To begin the EDA, we will focus on our target variable: score

```
plt.hist(data['score'].dropna(), bins = 100, edgecolor = 'k');
plt.xlabel('Score'); plt.ylabel('Number of Buildings');
plt.title('Energy Star Score Distribution');
```



Outliers

```
data['Site EUI (kBtu/ft²)'].describe()
```

count 11583.000000
mean 280.071484
std 8607.178877
min 0.000000
25% 61.800000
50% 78.500000
75% 97.600000
max 869265.000000

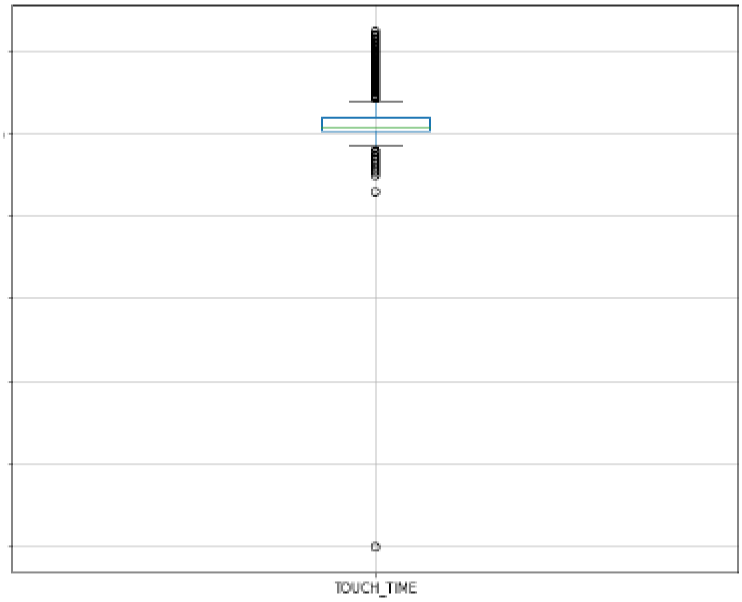
```
data['Site EUI (kBtu/ft²)'].dropna().sort_values().tail(10)
```

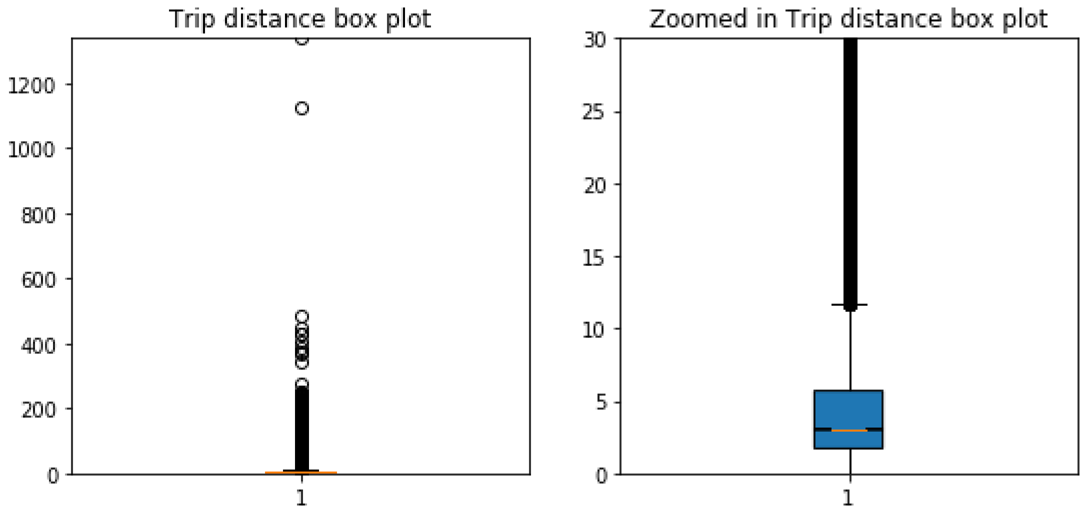
3173 51328.8
3170 51831.2
3383 78360.1
8269 84969.6
3263 95560.2
8268 103562.7
8174 112173.6
3898 126307.4
7 143974.4
8068 869265.0

```
shotsLogData.boxplot(column=['TOUCH_TIME'])
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(9, 4))
```

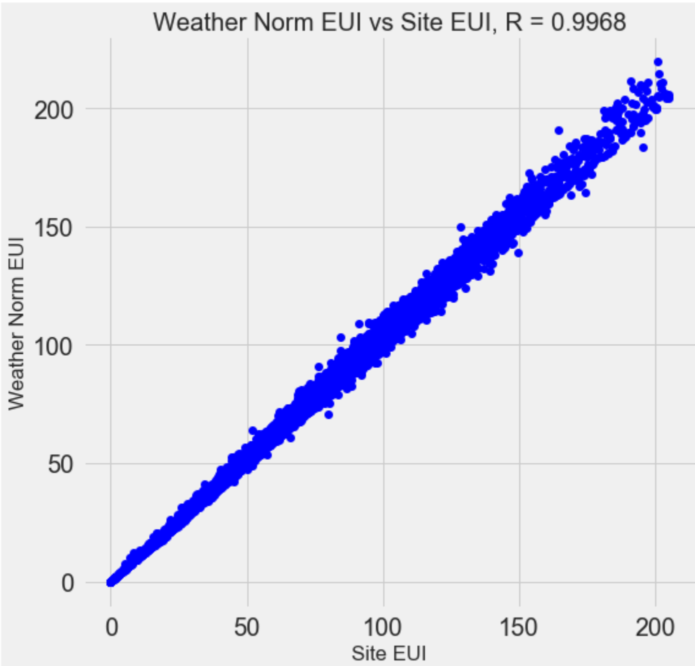
```
bplot1 = axes[0].boxplot(mergedDf.TRIP_DISTANCE,
                         vert=True,
                         patch_artist=True)
axes[0].set_ylim(0, mergedDf.TRIP_DISTANCE.max())
axes[0].set_title("Trip distance box plot")
bplot2 = axes[1].boxplot(mergedDf.TRIP_DISTANCE,
                        notch=True,
                        vert=True,
                         patch_artist=True)
axes[1].set_title("Zoomed in Trip distance box plot")

axes[1].set_ylim(0, 30)
```



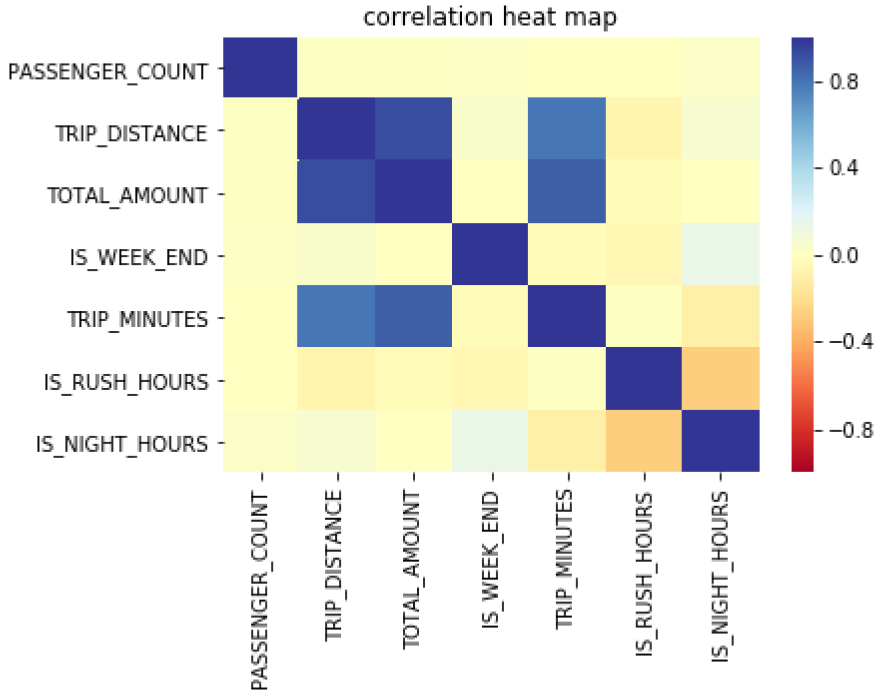


Remove highly correlated features¶

plt.plot(plot_data['Site EUI (kBtu/ft²)'], plot_data['Weather Normalized Site EUI (kBtu/ft²)'], 'bo')



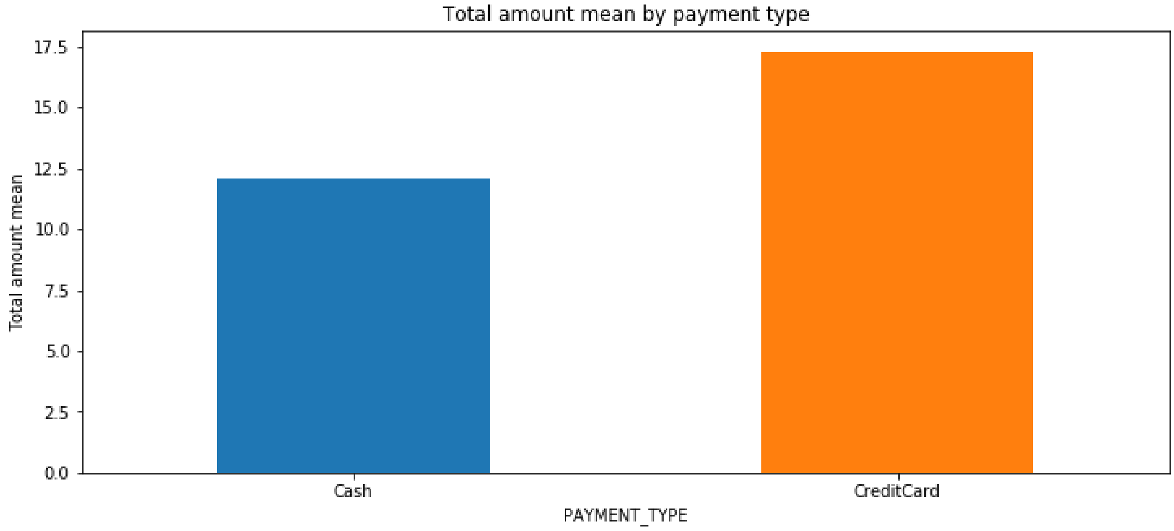

```
features=['PASSENGER_COUNT', 'TRIP_DISTANCE',
      'TOTAL_AMOUNT',
      'IS_WEEK_END', 'TRIP_MINUTES', 'IS_RUSH_HOURS', 'IS_NIGHT_HOURS',
     ]
sns.set_style()
corr = mergedDf[features].corr()
sns.heatmap(corr,cmap="RdYlBu",vmin=-1,vmax=1)
plt.title("correlation heat map")
plt.show()
```



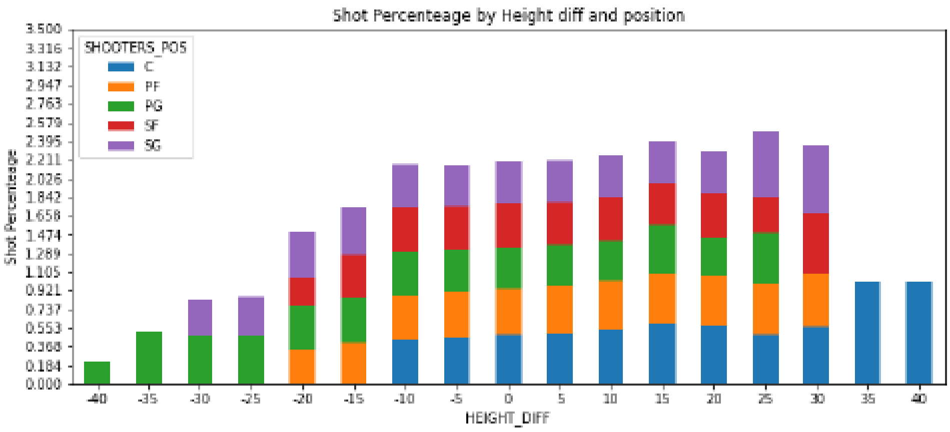
Categorical variables - Looking for Relationships

#mean price by pay type

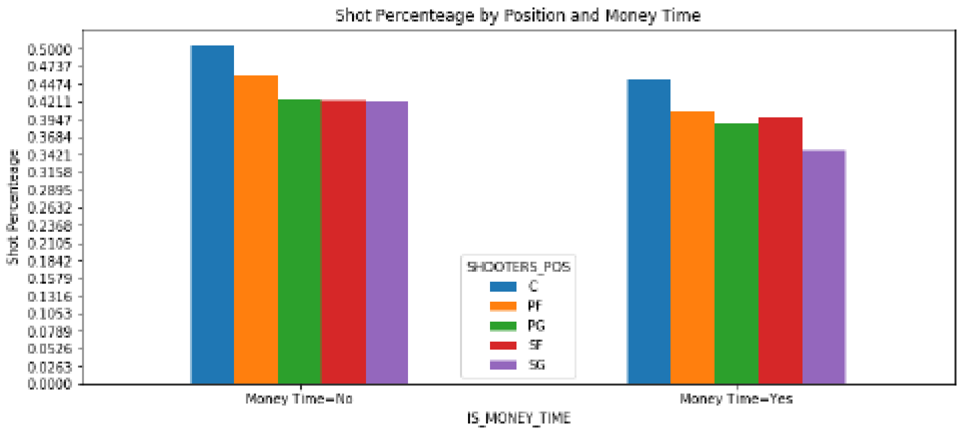
```
ax=mergedDf.groupby(['PAYMENT_TYPE']).mean()['TOTAL_AMOUNT'].plot.bar(figsize=(12,
5),rot=0)
ax.set_title('Total amount mean by payment type')
ax.set_ylabel('Total amount mean')
```



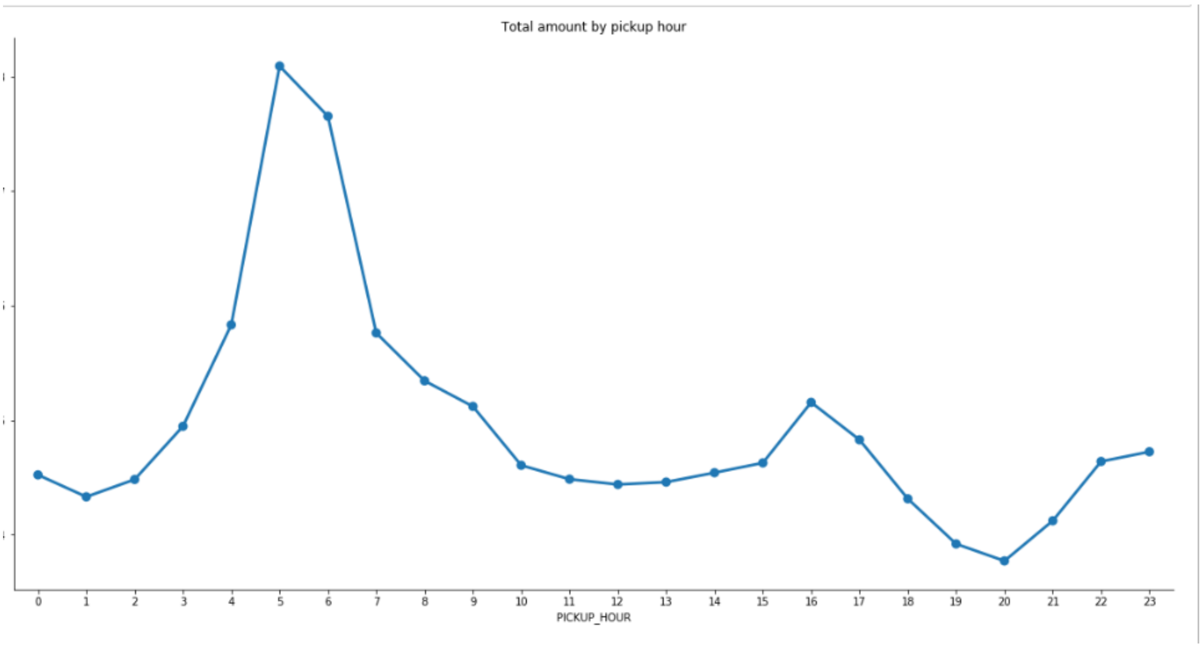

```
ax=cleanedDataShots.groupby(['HEIGHT_DIFF','SHOOTERS_POS']).mean()['SHOT_RESULT_INT'].
unstack().pl
ot.bar(stacked=True,xticks=cleanedDataShots['SHOT_DIST'].unique(),yticks=np.linspace(0,3.5,20),fi
gs
ize=(12, 5),rot=0)
ax.set_title('Shot Percenteage by Height diff and position')
ax.set_ylabel('Shot Percenteage')
plt.savefig('ShotPercenteageByHeightDiffPosition.png')
```



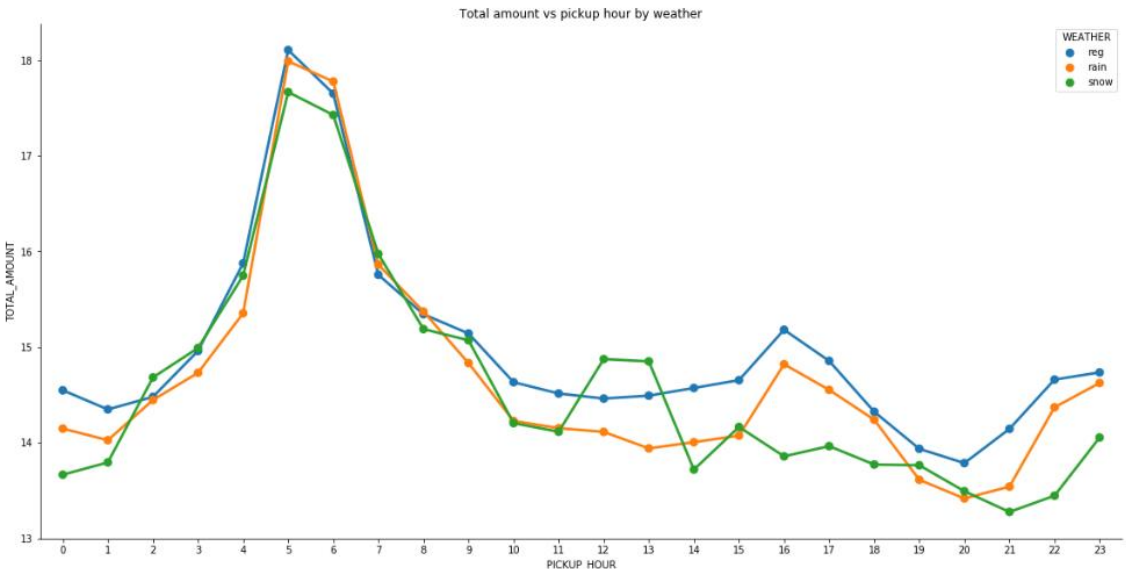

```
ax=cleanedDataShots.groupby(['IS_MONEY_TIME','SHOOTERS_POS']).mean()['SHOT_RESULT_IN
T'].unstack().
plot.bar(yticks=np.linspace(0,0.5,20),figsize=(12, 5),rot=0)
ax.set_title('Shot Percenteage by Position and Money Time')
ax.set_ylabel('Shot Percenteage')
ax.set_xticklabels(['Money Time=No','Money Time=Yes'])
```



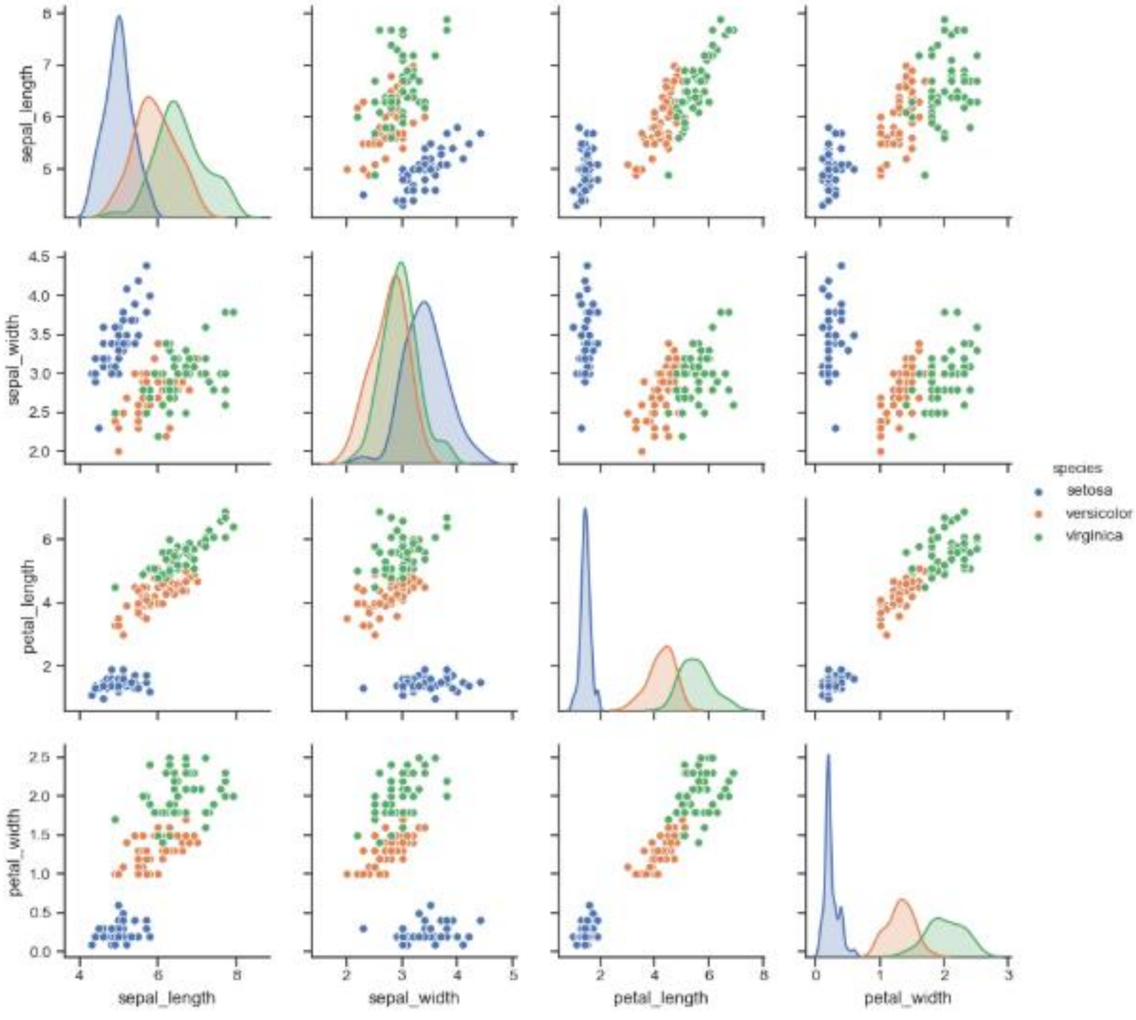

```
sns.factorplot('PICKUP_HOUR',
               'TOTAL_AMOUNT',
                estimator = np.mean,
                data = mergedDf,
               size = 8,
               aspect = 2,
                ci=None,
               legend_out=False)
plt.title("Total amount by pickup hour")
plt.show()
```




```
sns.factorplot('PICKUP_HOUR',
              'TOTAL_AMOUNT',
               hue = 'WEATHER',
              estimator = np.mean,
               data = mergedDf,
               size = 8,
               aspect = 2,
                ci=None,
               legend_out=False)
plt.title("Total amount vs pickup hour by weather")
```



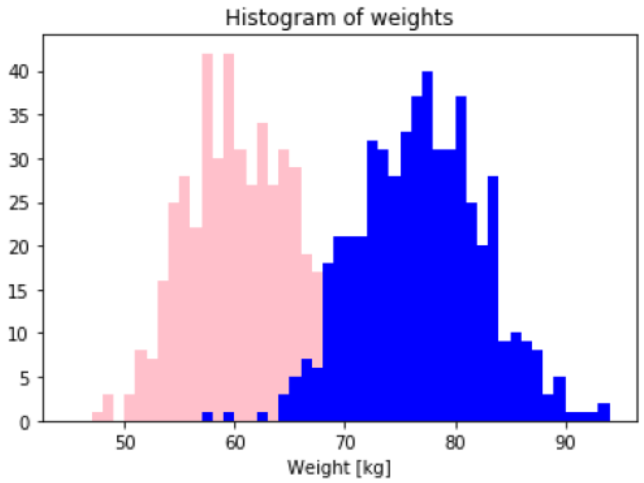

```
g = sns.pairplot(iris, hue="species")
```



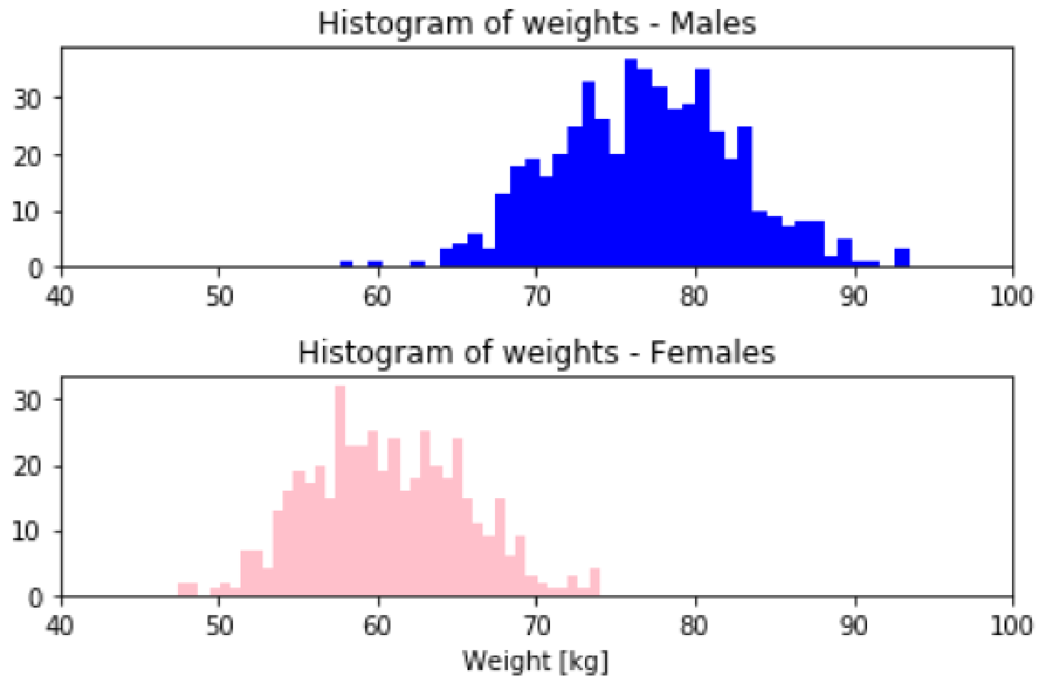
# Creating dummy variables
mergedDf_dummies=pd.get_dummies(mergedDf,columns=['WEATHER','PASSENGER_COUNT'])

In [31]:

```
fig = plt.figure()
ax = fig.gca()
ax.hist(weights[gender == 'f', 1], bins=bins, color='pink')
ax.hist(weights[gender == 'm', 1], bins=bins, color='blue')
ax.set_title('Histogram of weights')
ax.set_xlabel('Weight [kg]')
# ax.set_xlim([40, 100])
plt.show()
```



```
fig = plt.figure()

ax1 = fig.add_subplot(2, 1, 1)
ax1.hist(weights[gender == 'm', 1], bins=40, color='blue
ax1.set_title('Histogram of weights - Males')
ax1.set_xlim([40, 100])

ax2 = fig.add_subplot(2, 1, 2)
ax2.hist(weights[gender == 'f', 1], bins=40, color='pink
ax2.set_title('Histogram of weights - Females')
ax2.set_xlim([40, 100])
ax2.set_xlabel('Weight [kg]')

fig.tight_layout()
plt.show()
```



Feature Engineering and Selection

Feature Engineering and Selection

Now that we have explored the trends and relationships within the data, we can work on engineering a set of features for our models. We can use the results of the EDA to inform this feature engineering. In particular, we learned the following from EDA which can help us in engineering/selecting features:

- The score distribution varies by building type and to a lesser extent by borough. Although we will focus on numerical features, we should also include these two categorical features in the model.
- Taking the log transformation of features does not result in significant increases in the linear correlations between features and the score

Before we get any further, we should define what feature engineering and selection are! These definitions are informal and have considerable overlap, but I like to think of them as two separate processes:

- **Feature Engineering**: The process of taking raw data and extracting or creating new features that allow a machine learning model to learn a mapping beween these features and the target. This might mean taking transformations of variables, such as we did with the log and square root, or one-hot encoding categorical variables so they can be used in a model. Generally, I think of feature engineering as **adding** additional features derived from the raw data.
- **Feature Selection**: The process of choosing the most relevant features in your data. "Most relevant" can depend on many factors, but it might be something as simple as the highest correlation with the target, or the features with the most variance. In feature selection, we remove features that do not help our model learn the relationship between features and the target. This can help the model generalize better to new data and results in a more interpretable model. Generally, I think of feature selection as **subtracting** features so we are left with only those that are most important.

Feature engineering and selection are iterative processes that will usually require several attempts to get right. Often we will use the results of modeling, such as the feature importances from a random forest, to go back and redo feature selection, or we might later discover relationships that necessitate creating new variables. Moreover, these processes usually incorporate a mixture of domain knowledge and statistical qualitites of the data.
Feature engineering and selection often has the highest returns on time invested in a machine learning problem. It can take quite a while to get right, but is often more important than the exact algorithm and hyperparameters used for the model. If we don't feed the model the correct data, then we are setting it up to fail and we should not expect it to learn!

```
#transform features
df['Embarked'] = df['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)
In
shotsLogData['HEIGHT_DIFF'] = shotsLogData.HEIGHT_DIFF.apply(lambda x: roundNumber(x,5))
shotsLogData['IS_MONEY_TIME_GAME']=shotsLogData.apply(lambda x: isMoneyTimeGame(x),
axis=1)
# adding month,day,hour columns for merging with taxi df
df_weather["MONTH"]=df_weather["DATE"].dt.month
df_weather["DAY"]=df_weather["DATE"].dt.day
df_weather["HOUR"]=df_weather["DATE"].dt.hour
df_taxi['TRIP_MINUTES'] = (df_taxi['LPEP_DROPOFF_DATETIME'] -
df_taxi['LPEP_PICKUP_DATETIME'])
df_taxi['TRIP_MINUTES'] = df_taxi['TRIP_MINUTES']/np.timedelta64(1,'m')
data['duration'] = (data['deadline'] - data['launched']).dt.days
```

Split Into Training and Testing Sets

In machine learning, we always need to separate our features into two sets:

1. **Training set** which we provide to our model during training along with the answers so it can learn a mapping between the features and the target.
2. **Testing set** which we use to evaluate the mapping learned by the model. The model has never seen the answers on the testing set, but instead, must make predictions using only the features. As we know the true answers for the test set, we can then compare the test predictions to the true test targets to ghet an estimate of how well our model will perform when deployed in the real world.

For our problem, we will first extract all the buildings without an Energy Star Score (we don't know the true answer for these buildings so they will not be helpful for training or testing). Then, we will split the buildings with an Energy Star Score into a testing set of 30% of the buildings, and a training set of 70% of the buildings.
Splitting the data into a random training and testing set is simple using scikit-learn. We can set the random state of the split to ensure consistent results

```
X = mergedDf_dummies.drop("TOTAL_AMOUNT",axis=1)
y = mergedDf_dummies["TOTAL_AMOUNT"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
```

Establish a Baseline

- Choose random/not good model to improve

Models to Evaluate

We will compare five different machine learning models using the great Scikit-Learn library:

1. Linear Regression
2. Support Vector Machine Regression
3. Random Forest Regression
4. Gradient Boosting Regression
5. K-Nearest Neighbors Regression

```
chosenModel=['TRIP_DISTANCE', 'TRIP_MINUTES', 'IS_AIRPORT_TRIP', 'IS_CREDIT']
lm =LinearRegression()
model_X_train=X_train[chosenModel]
lm.fit(model_X_train,y_train)
model_X_valid=X_valid[chosenModel]
predictions_valid_set = lm.predict(model_X_valid)
MAE=metrics.mean_absolute_error(y_valid, predictions_valid_set)
MSE=metrics.mean_squared_error(y_valid, predictions_valid_set)
RMSE=np.sqrt(metrics.mean_squared_error(y_valid, predictions_valid_set))
RSQUARED=metrics.explained_variance_score(y_valid, predictions_valid_set)
print('MAE:', MAE)
print('MSE:',MSE )
print('RMSE:',RMSE )
print('RSQUARED:', RSQUARED)
```

Mean squared error	MSE	=	$\frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	RMSE	=	$\sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$
Mean absolute error	MAE	=	$\frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$
Mean absolute percentage error	MAPE	=	$\frac{100\%}{n}\sum_{t=1}^{n} \left\lvert \frac{e_t}{y_t} \right\rvert$

Scaling Features
StandardScaler() - Scales the data by assuming normal distribution
MinMaxScaler() - Scales the data between 0 and 1, where the minimal entry scales to 0 and the maximal to 1
MaxAbsScaler() - Scales the data between 0 and 1, where the maximal entry is scaled to 1, and the others correspond to

```
from sklearn.model_selection import GridSearchCV
tipPredictDfNormalize=pd.read_csv('tipPredictDf.csv')

scaler = StandardScaler()
tipPredictDfNormalize[tipPredictDfNormalize.columns[:-1]]=scaler.fit_transform(tipPredictDfNormalize.drop("DID_GAVE_TIP",axis=1))
y = tipPredictDfNormalize["DID_GAVE_TIP"]
X=tipPredictDfNormalize.drop("DID_GAVE_TIP",axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
X_valid, X_test, y_valid, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=100)
```

Grid Search

```
rfModel=RandomForestClassifier(random_state=100)
param_grid = {
    'n_estimators':[5,10,15,20,50],
    "min_samples_leaf" : [1,2, 4, 6,8,10],
    'criterion' :['gini', 'entropy']
}
rfModel = GridSearchCV(estimator=rfModel, param_grid=param_grid, cv= 5)
rfModel.fit(X_train, y_train)

predictions=rfModel.predict(X_valid)
print('\n accuracy:  ',accuracy_score(y_valid, predictions))
```

Imbalanced data

```
radar_clf = LogisticRegression(class_weight={'Plane': 10, 'Bird': 1}).fit(X, y)
print (report(radar_clf, X, y))


gaveTipDf=gaveTipDf.sample(200000,random_state=100)
didNotgaveTipDf=didNotgaveTipDf.sample(200000,random_state=100)
```

```
tipPredictDf=pd.concat([gaveTipDf,didNotgaveTipDf])
```

classification

• Supervised Machine learning models (Regression, Logistic Regression, KNN, Random Forests, Naive Bayes)

(Confusion Matrix) מטריצת הבילבול

- TP= hit
- TN = correct rejection
- FP = false alarm = type-I error
- FN = miss = type-II error

		Prediction	
		Negative	Positive
Actual	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

- Sensitivity = Recall = hit-rate = TPR = $\frac{TP}{P} = \frac{TP}{TP+FN}$
- Specificity = TNR = $\frac{TN}{N} = \frac{TN}{TN+FP}$
- Fall-out = FPR = 1 - TNR = $\frac{FP}{N} = \frac{FP}{TN+FP}$
- Precision = $\frac{TP}{TP+FP}$

<u>מדדים נפוצים</u>

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F1 = \frac{2 \times Precision \times Sensitivity}{Precision + Sensitivity}$$

Dimensionality reduction

VarianceThreshold

Removal of features which exhibit a negligible level of variance.
This is implemented by the *VarianceThreshold* transformer.

Unsupervised Feature extraction - PCA

Feature extraction refers to any method which creates new features that are (hopefully) more informative than the original ones.

Averaging

```
tipPredictDfNormalizeVotingClassifier=pd.read_csv('tipPredictDf.csv')
scaler = StandardScaler()
tipPredictDfNormalizeVotingClassifier[tipPredictDfNormalizeVotingClassifier.columns[:-1]]=scaler.fit_transform(tipPredictDfNormalizeVotingClassifier.drop("DID_GAVE_TIP",axis=1))
y = tipPredictDfNormalizeVotingClassifier["DID_GAVE_TIP"]
X=tipPredictDfNormalizeVotingClassifier.drop("DID_GAVE_TIP",axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
X_valid, X_test, y_valid, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=100)
print("train df size="+str(len(X_train))+", valid df size="+ str(len(X_valid))+", test df size="+str(len(X_test))+ ", full df size="+ str(len(tipPredictDfNormalize)))
clf1 = KNeighborsClassifier(n_neighbors=51)
clf2 = LogisticRegression()
clf3 = RandomForestClassifier()

classifiers = [('KNN', clf1), ('LR', clf2), ('RF', clf3)]
algorithmStr='VotingClassifier'
title='Running '+algorithmStr+ 'algorithm'
print(title)
print("------------------------------------------------------------------------")
clf_voting = VotingClassifier(estimators=classifiers,
                    voting='hard')
clf_voting.fit(X_train, y_train)
predictions=clf_voting.predict(X_valid)
print('\n accuracy:  ',accuracy_score(y_valid, predictions))
```

```
cm = confusion_matrix(y_true=y_valid,
                      y_pred=predictions)
cmDf=pd.DataFrame(cm,
                  index=clf_voting.classes_,
                  columns=clf_voting.classes_)
print(cmDf)
print ('\nclassification_report \n',classification_report(y_true=y_valid,
                      y_pred=predictions))
```