In [69]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import wget
```

About the data

This dataset contains features related to weather or data of bicycle rentals. It consists of the following columns:

- **instant** A unique row identifier.
- **dteday** date of observation
- **season** 1:winter, 2: spring, 3: summer, and 4:fall.
- **yr** year of the study 2011 is represented by 0 and 2012 by 1
- **mnth** 1 for Jan and 12 for December
- **holiday** binary value 0: no, 1: yes
- **weekday** 0: sunday, 6:Saturday
- **Workingday** binary value
- **weathersit** Categorical value indicating the weather situation, 1: clear, 2: mist/cloud, 3: lightran/snow
- **temp** Temperature in celsius (normalized)
- **atemp** Felt teamperature (normalized)
- **hum**: Humidity (normalized)
- **windspeed** The windspeed (normalized)
- **rentals**: The number of bicycle rentals recorded.

In [71]:
```
# Load the training dataset
url = 'https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-mach
filename = wget.download(url)
print(f"\nDownloaded file: {filename}")
```

```
100%
[.........................................................................] 488
00 / 48800
Downloaded file: daily-bike-share (3).csv
```

In [72]:
```
bike_data = pd.read_csv('daily-bike-share.csv')
bike_data.head()
```

Out[72]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	ten
0	1	1/1/2011	1	0	1	0	6	0	2	0.3441
1	2	1/2/2011	1	0	1	0	0	0	2	0.3634
2	3	1/3/2011	1	0	1	0	1	1	1	0.1963
3	4	1/4/2011	1	0	1	0	2	1	1	0.2000
4	5	1/5/2011	1	0	1	0	3	1	1	0.2269

Let's examine descriptive statistics of the numerical values and the label.

In [74]:
```
numeric_features = ['temp', 'atemp', 'hum', 'windspeed']
bike_data[numeric_features + ['rentals']].describe()
```

Out[74]:

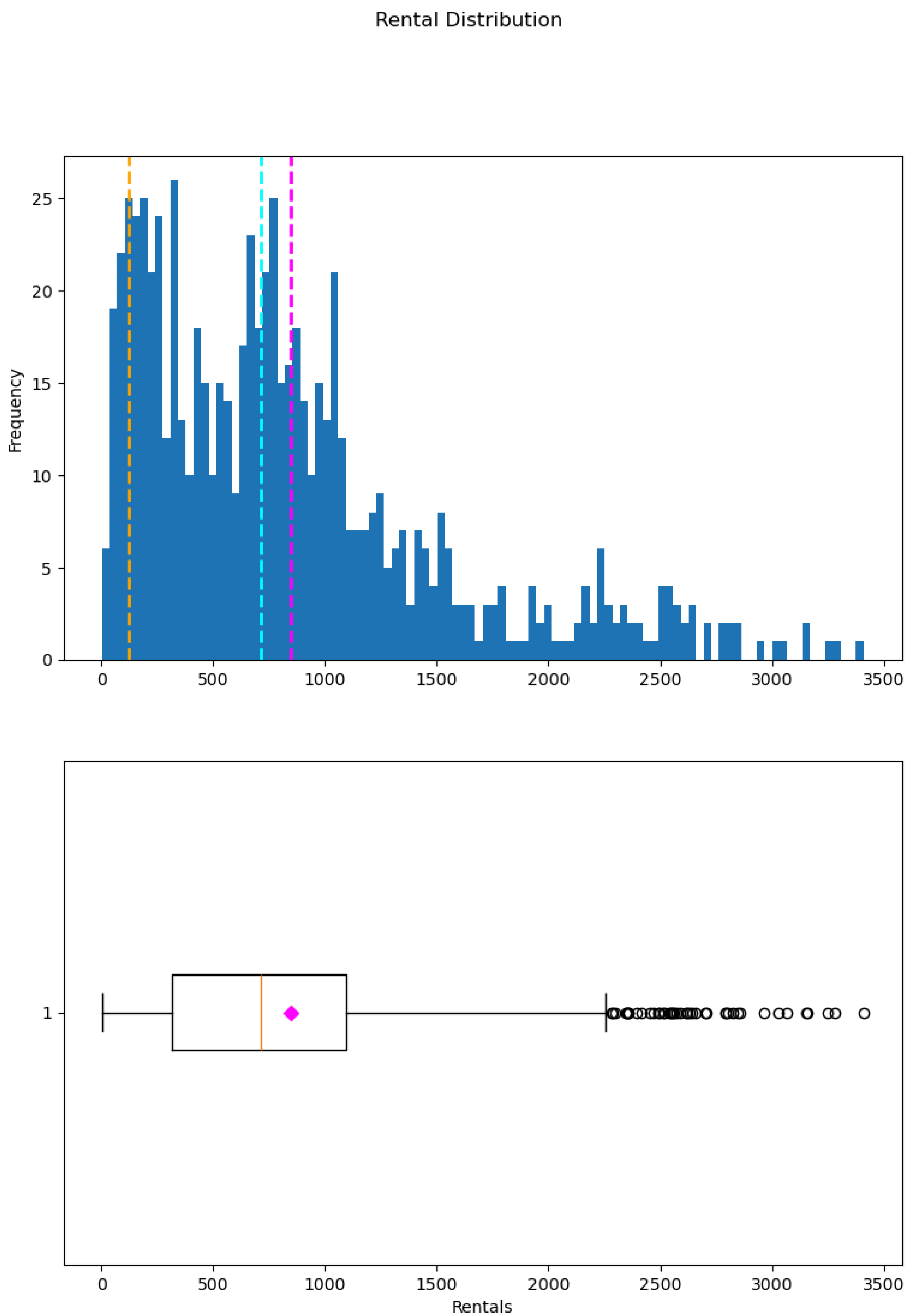
	temp	atemp	hum	windspeed	rentals
count	731.000000	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.474354	0.627894	0.190486	848.176471
std	0.183051	0.162961	0.142429	0.077498	686.622488
min	0.059130	0.079070	0.000000	0.022392	2.000000
25%	0.337083	0.337842	0.520000	0.134950	315.500000
50%	0.498333	0.486733	0.626667	0.180975	713.000000
75%	0.655417	0.608602	0.730209	0.233214	1096.000000
max	0.861667	0.840896	0.972500	0.507463	3410.000000

The mean of the rentals is 848 but standard variation is large, indicating lot of variance.

Let's visualize the distribtuion of renatlsL

In [77]:
```
label = bike_data['rentals']

# create a figure for two subplots
fig, ax = plt.subplots(2,1, figsize=(9,12))

# plot the histogram
ax[0].hist(label, bins=100)
ax[0].set_ylabel('Frequency')

# add lines for the mean, median, and mode
ax[0].axvline(label.mean(), color='magenta', linestyle='dashed', linewidth=2)
ax[0].axvline(label.median(), color = 'cyan', linestyle='dashed', linewidth=2)
ax[0].axvline(label.mode()[0], color = 'orange', linestyle='dashed', linewidth=2)

# box plot
```

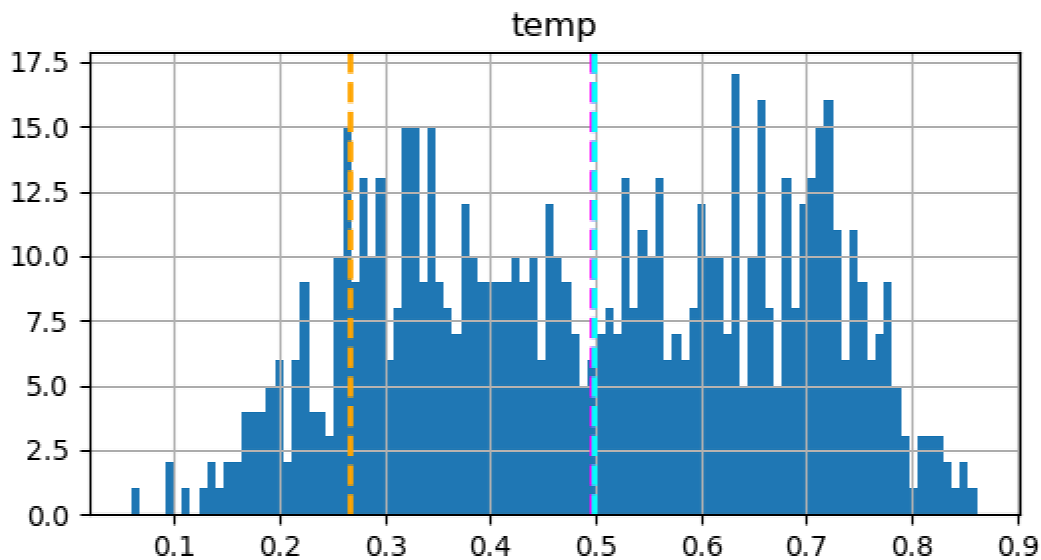
```
ax[1].boxplot(label, vert=False)
ax[1].set_xlabel('Rentals')
ax[1].scatter(label.mean(), 1, color='magenta', marker='D', label='Mean')

fig.suptitle('Rental Distribution');
```

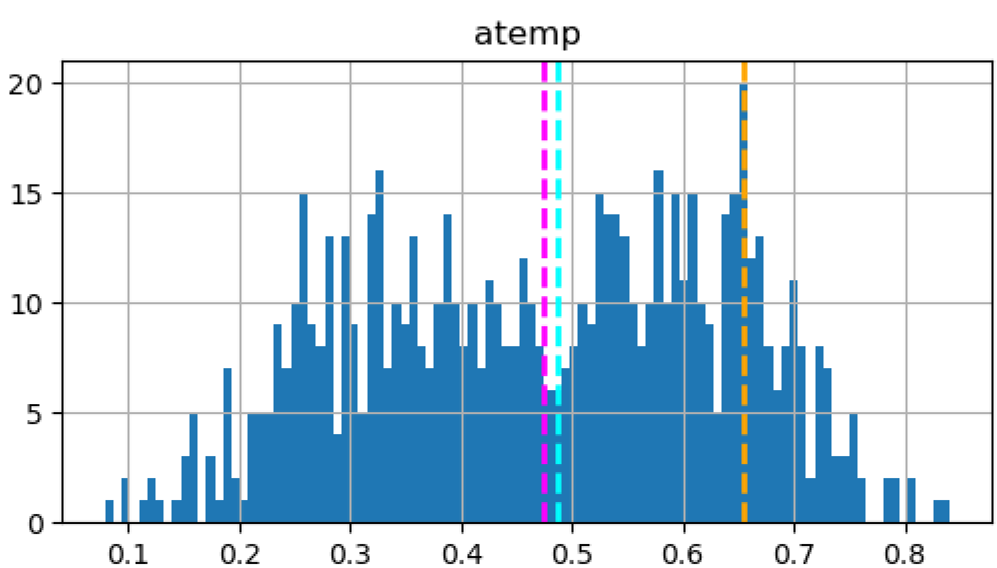


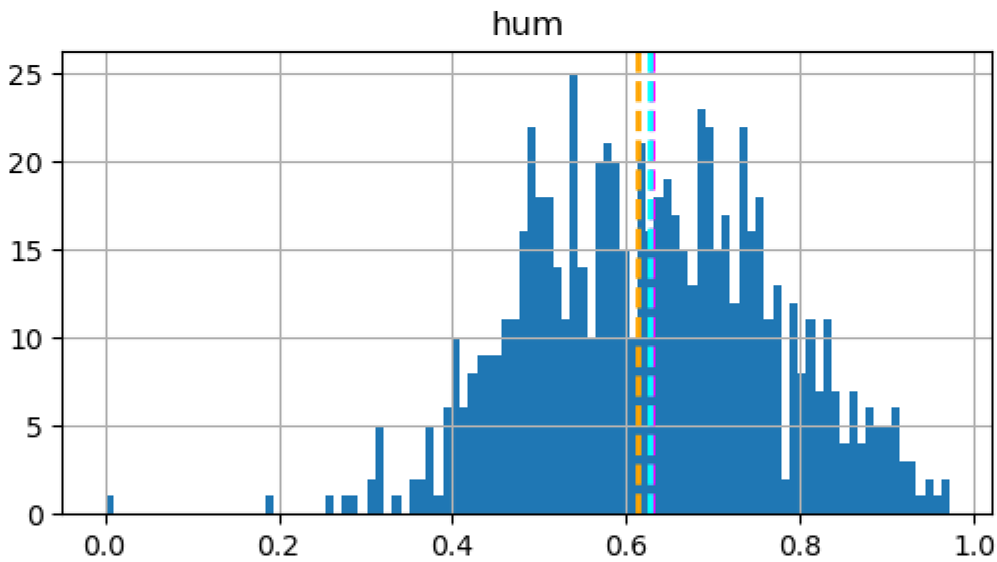
The plots show that the number of daily rentals ranges from 0 to 3,400. However mean and median are closer to the lower end.

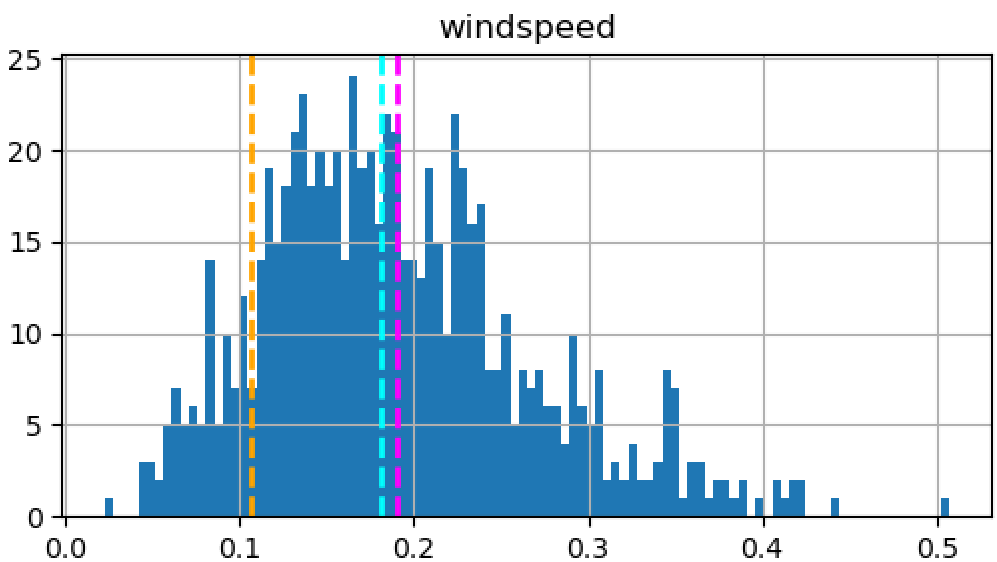
Visualizing the Distribution of Numveric Features

In [80]:
```
# ploting a histogram for each numeric feature

for col in numeric_features:
    fig = plt.figure(figsize=(6,3))
    ax = fig.gca()
    feature = bike_data[col]
    feature.hist(bins=100, ax=ax)
    ax.axvline(feature.mean(), color = 'magenta', linestyle='--', linewidth=2)
    ax.axvline(feature.median(), color = 'cyan', linestyle='--', linewidth=2)
    ax.axvline(feature.mode()[0], color = 'orange', linestyle='--', linewidth=2)

    ax.set_title(col);
```





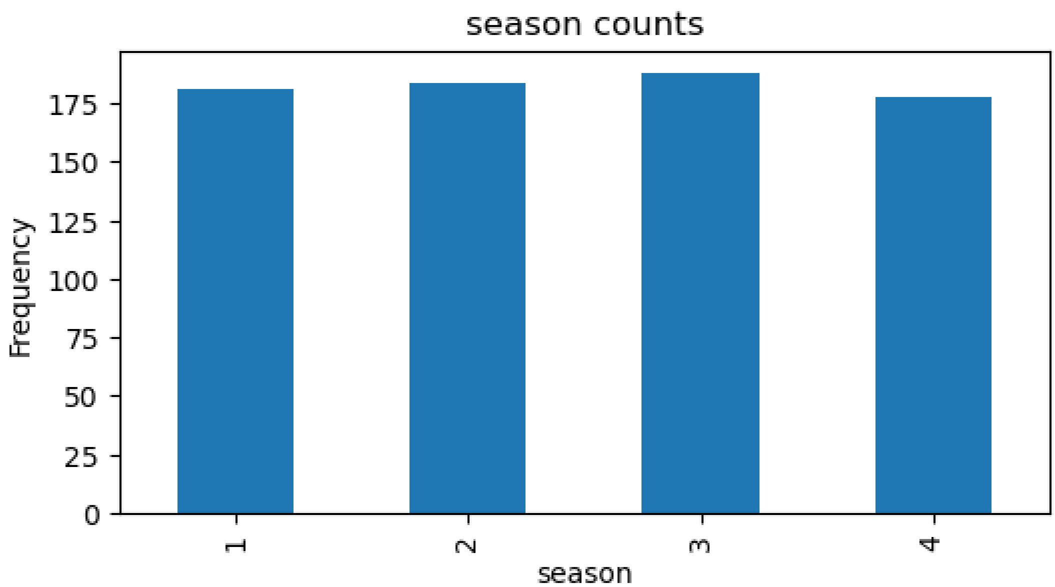


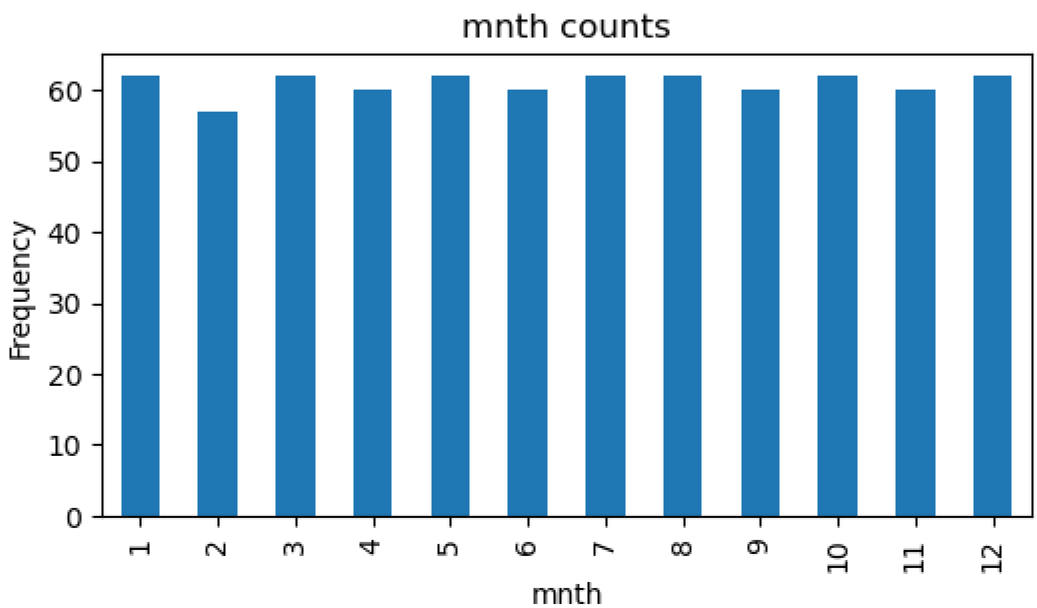


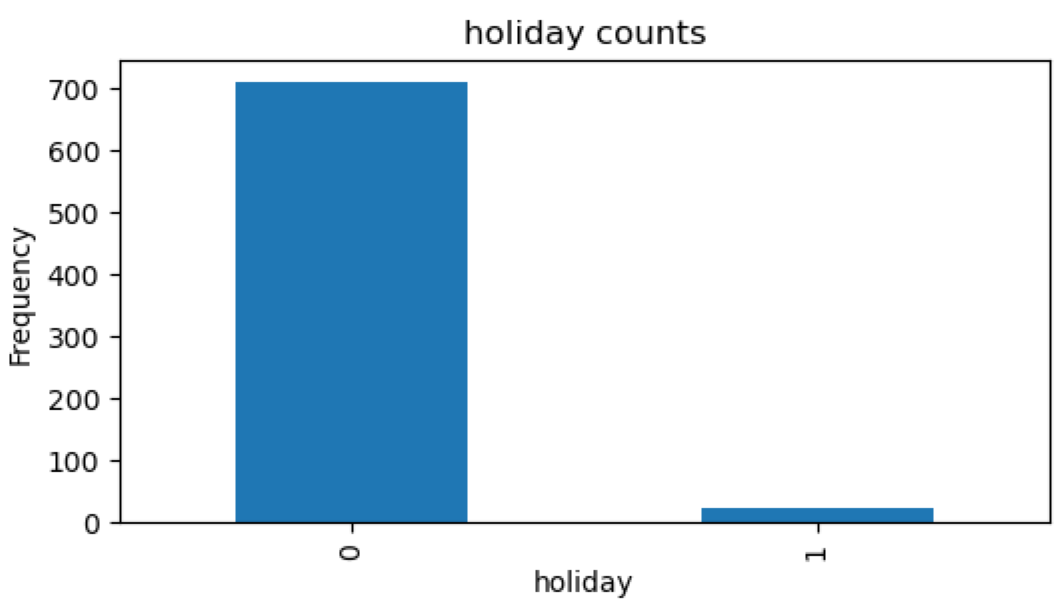
The mean and median of most numeric features is in the middle of the range, so they are closer to normal distribution.

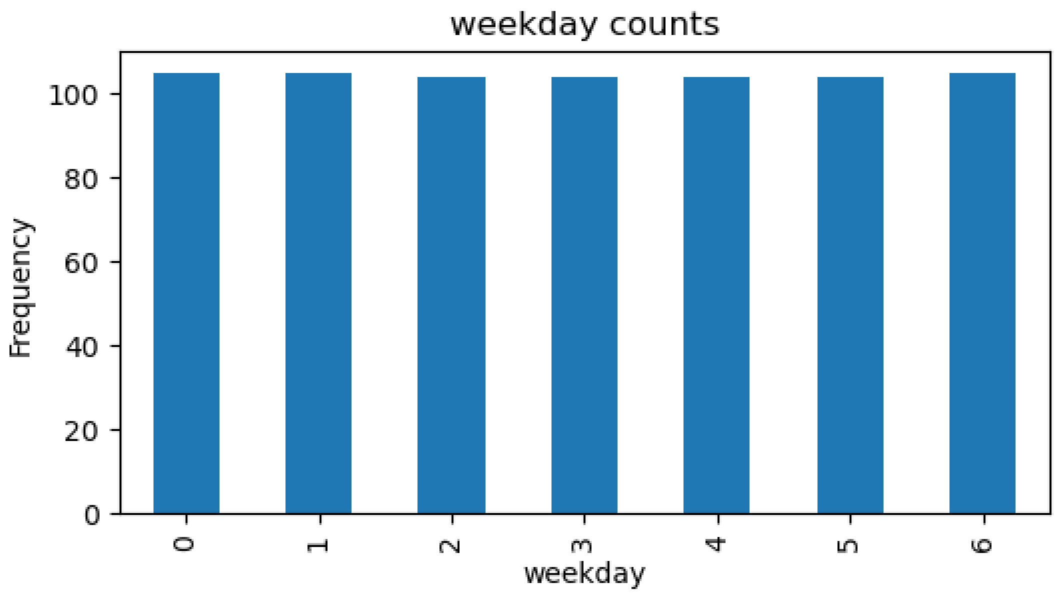
Distribution of Categocial Features

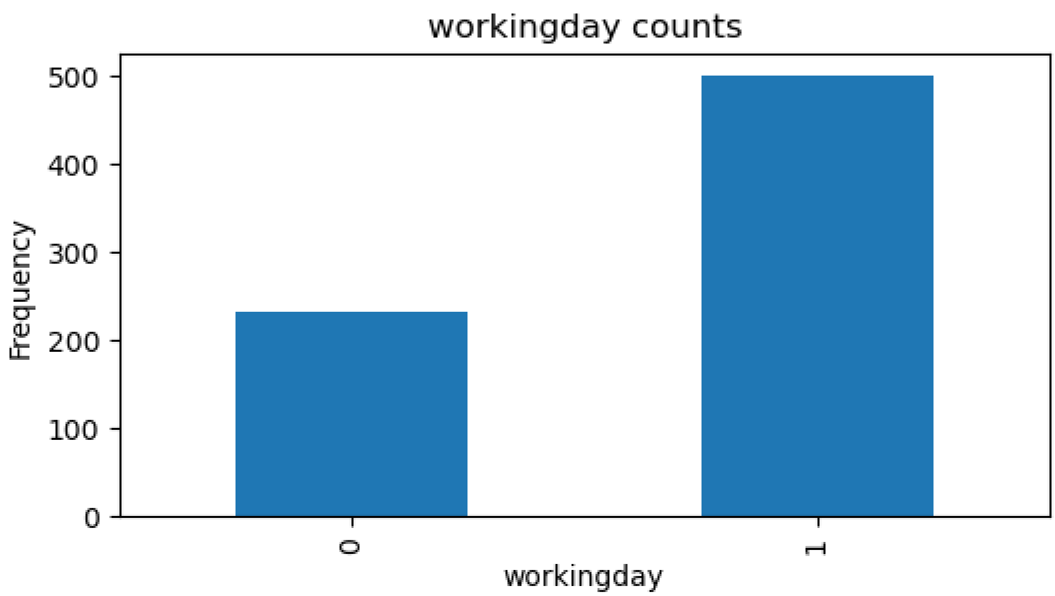
In [83]:
```
# plot a bar plot for each categorical feature count
categorical_features = ['season','mnth','holiday','weekday','workingday','weathersi

for col in categorical_features:
    counts = bike_data[col].value_counts().sort_index()
    fig = plt.figure(figsize=(6,3))
    ax=fig.gca()
    counts.plot(kind='bar', ax=ax)
    ax.set_title(col +' counts')
    ax.set_ylabel('Frequency')
    ax.set_xlabel(col);
```

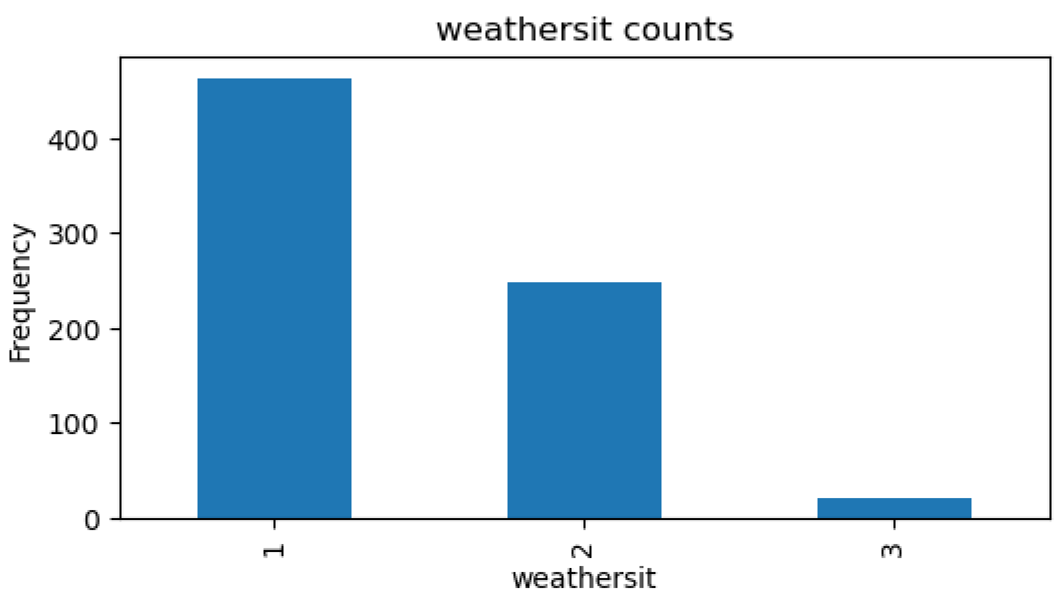

season counts
Frequency
season
0
25
50
75
100
125
150
175
1
2
3
4


mnth counts
Frequency
mnth
0
10
20
30
40
50
60
1
2
3
4
5
6
7
8
9
10
11
12


holiday counts
Frequency
holiday
0
100
200
300
400
500
600
700
0
1


weekday counts
Frequency
0
20
40
60
80
100
0
1
2
3
4
5
6
weekday


workingday counts
Frequency
0
100
200
300
400
500
0
1
workingday


weathersit counts
Frequency
0
100
200
300
400
1
2
3
weathersit

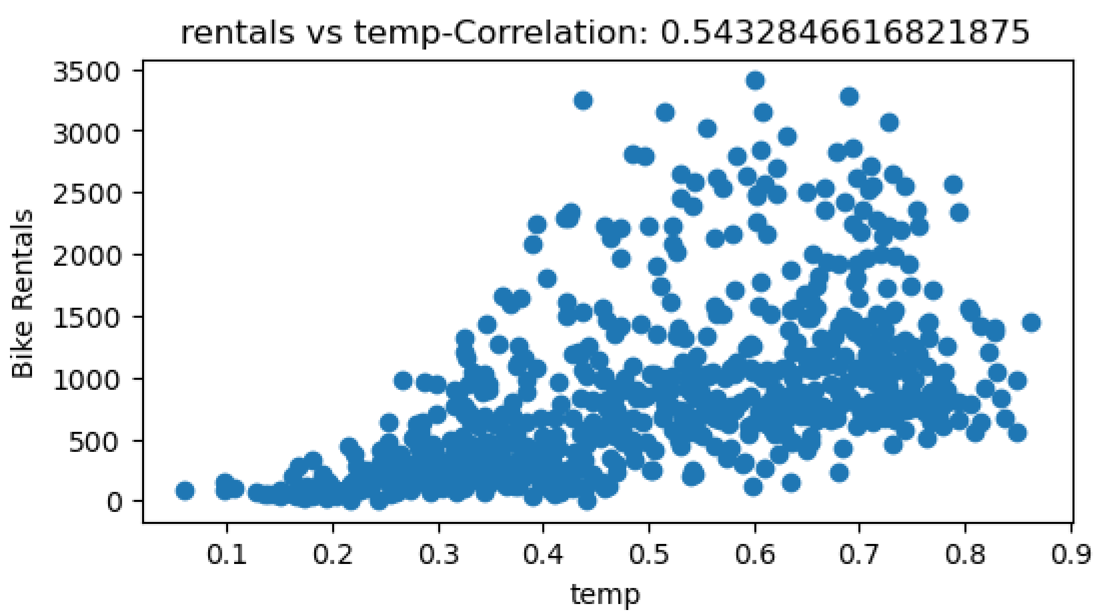
Many of the categorical features show a more or less **uniform distribution**, except for:

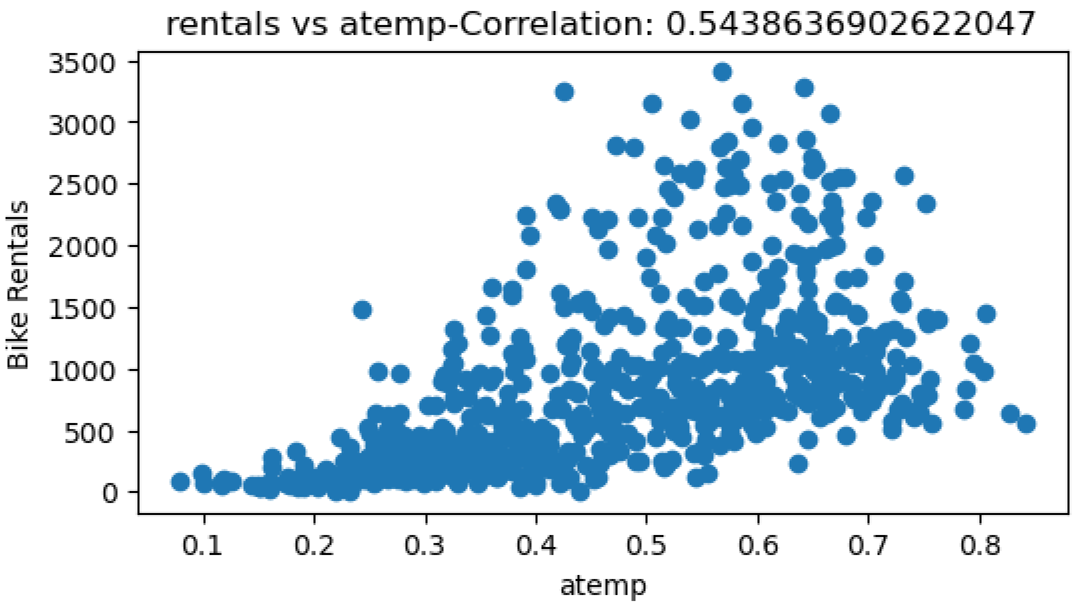
- **Holiday**: There are many fewer days that are holidays than days that aren't.
- **Workingday**: 5 working days and 2 weekends, so more working days.
- **Weatherit**: Most days are of category 1(clear), next most common is category 2 (mist or cloud), and lastly 3 (light rain or snow)

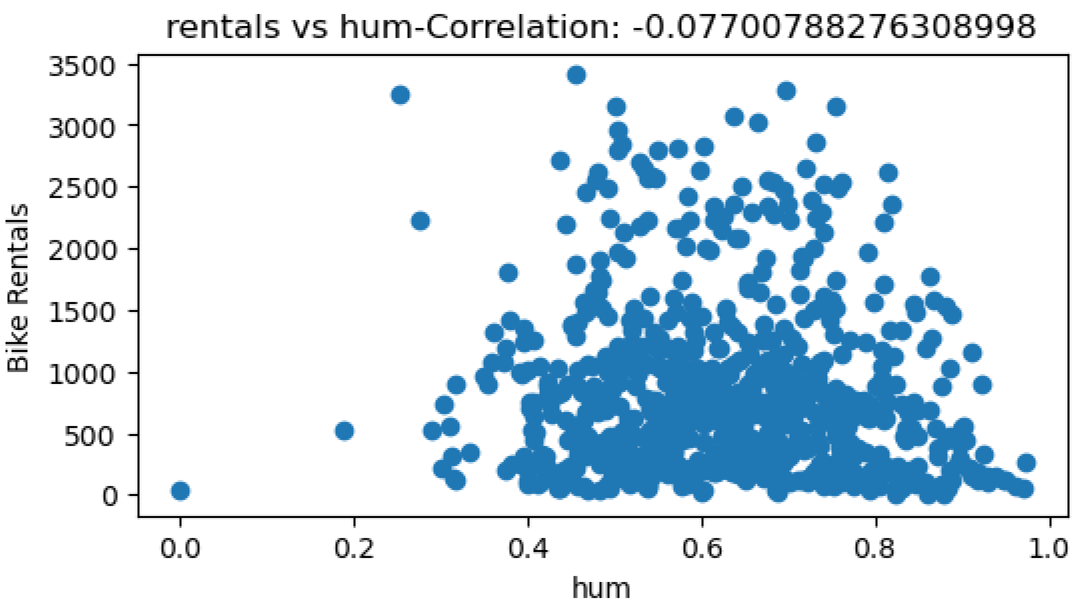
Relationship Between Numeric Features and Rentals

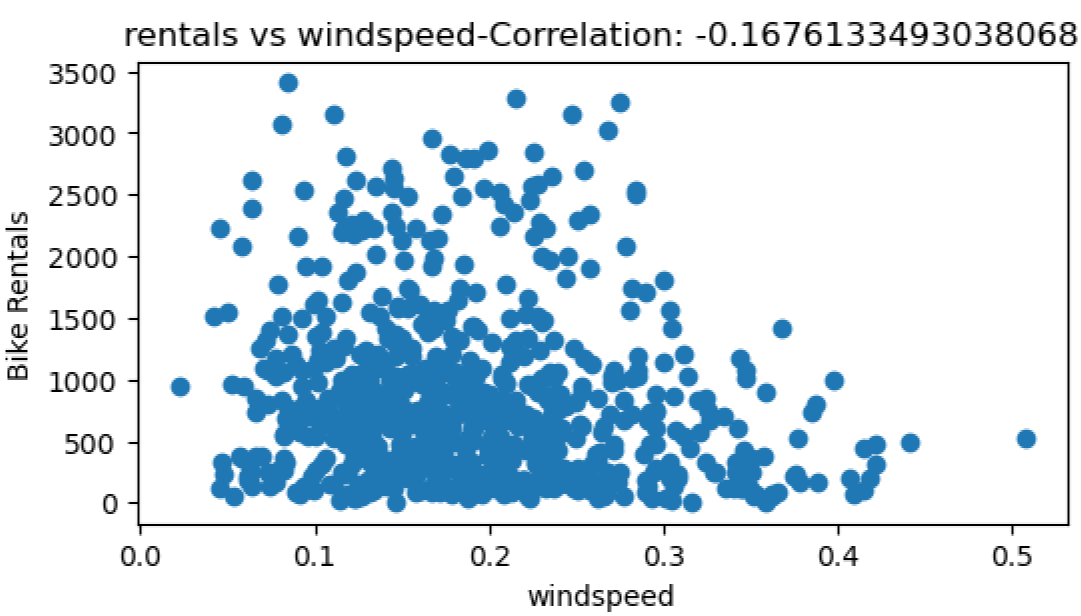
In [86]:
```
label = bike_data['rentals']
```

In [87]:
```
for col in numeric_features:
    fig = plt.figure(figsize=(6,3))
    ax = fig.gca()
    feature = bike_data[col]
    correlation = feature.corr(label)
    plt.scatter(x=feature, y=label)
    plt.xlabel(col)
    plt.ylabel('Bike Rentals')
    ax.set_title('rentals vs ' + col + '-Correlation: ' +str(correlation) );
```




rentals vs atemp-Correlation: 0.5438636902622047
Bike Rentals
atemp
0
500
1000
1500
2000
2500
3000
3500
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


rentals vs hum-Correlation: -0.07700788276308998
Bike Rentals
hum
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0

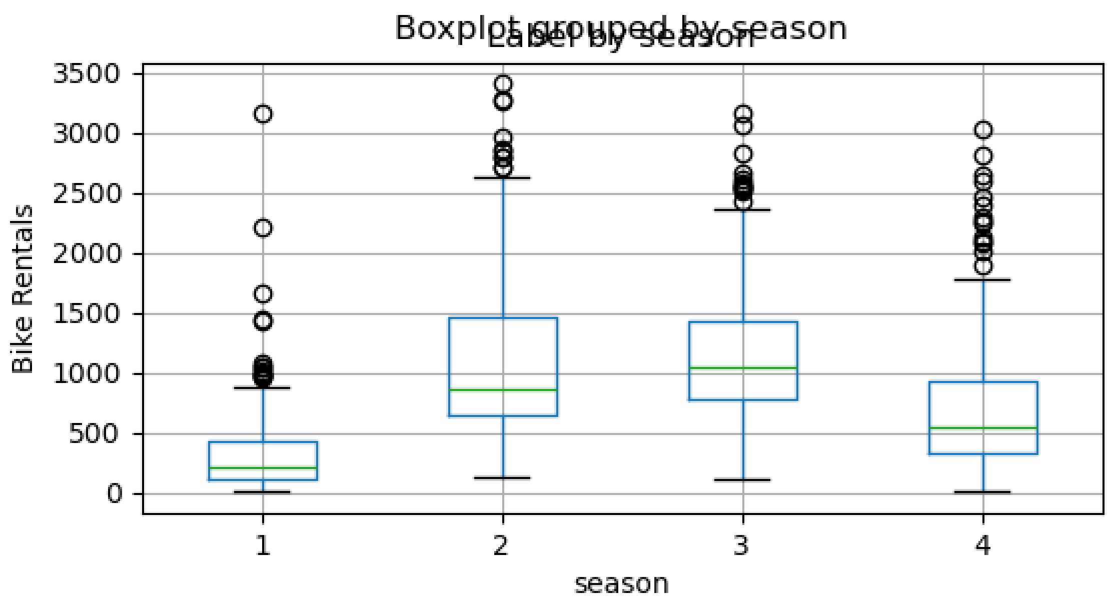

rentals vs windspeed-Correlation: -0.1676133493038068
Bike Rentals
windspeed
0
500
1000
1500
2000
2500
3000
3500
0.0
0.1
0.2
0.3
0.4
0.5

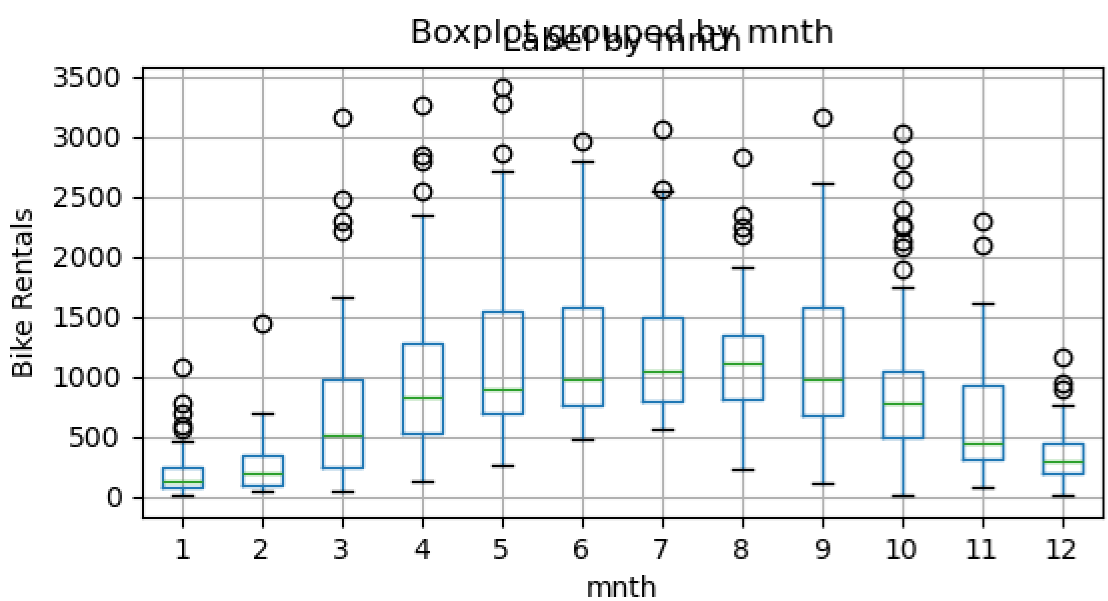
In [ ]:

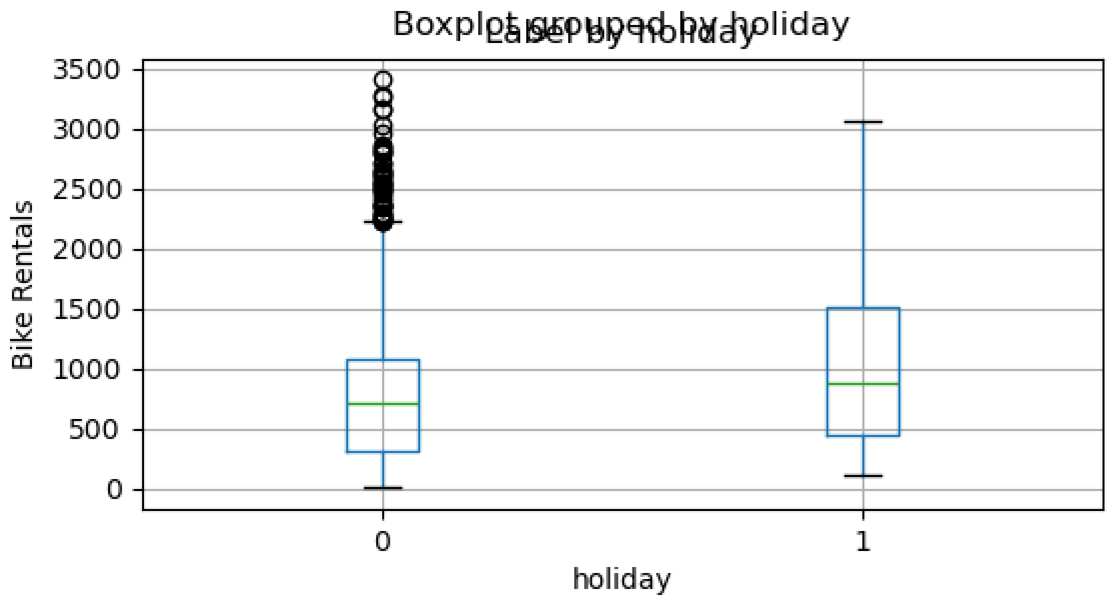
Relationship Between Categorical Features and Rentals

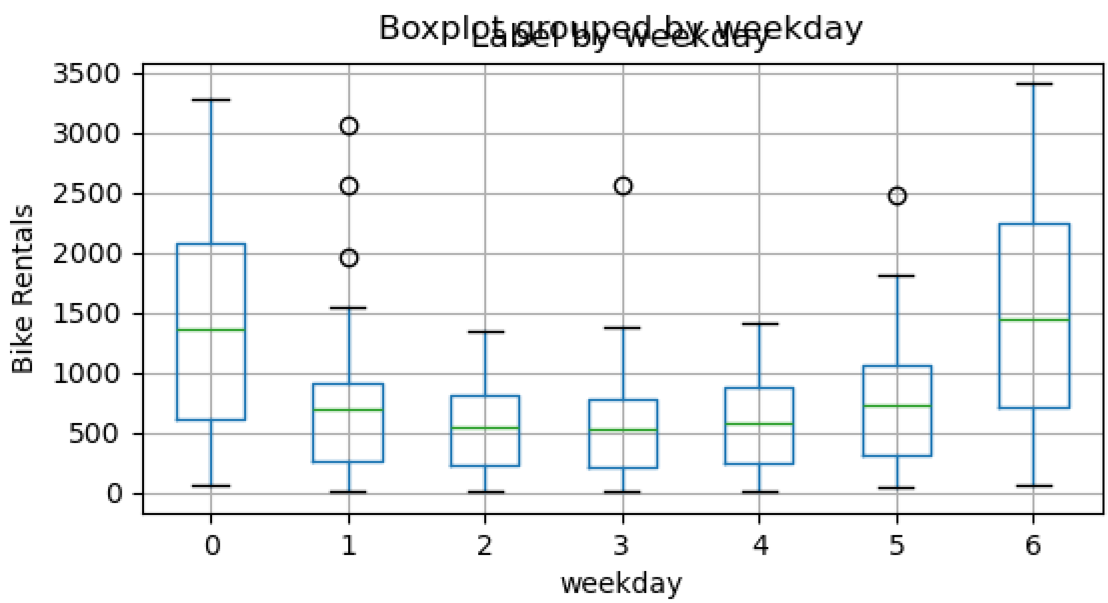
In [89]:

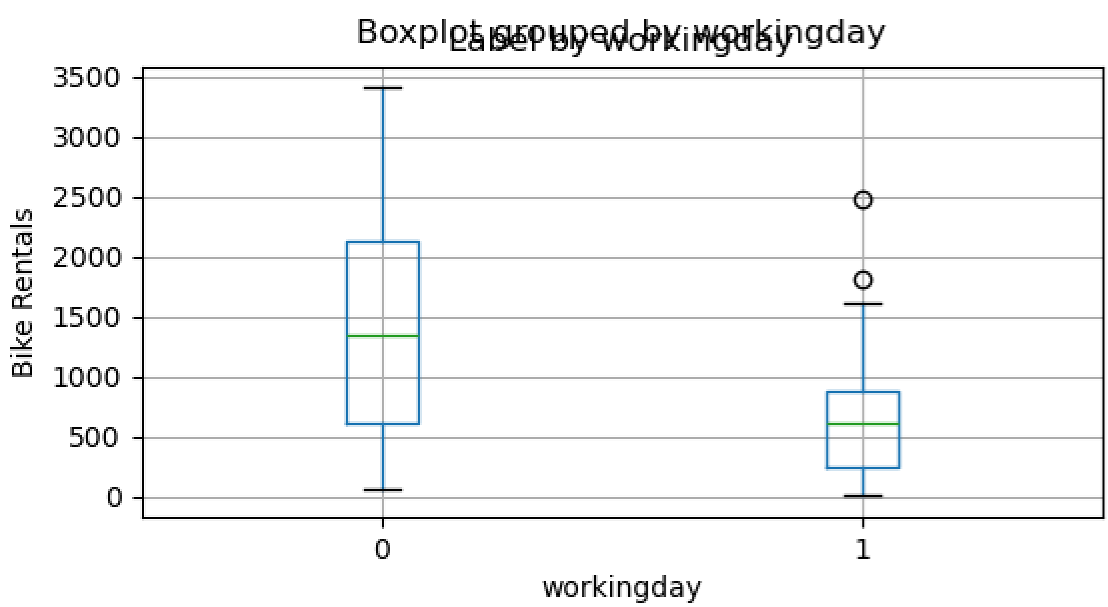
```
for col in categorical_features:
    fig = plt.figure(figsize=(6,3))
    ax = fig.gca()
    bike_data.boxplot(column = 'rentals', by = col, ax=ax)
    ax.set_title('Label by ' + col)
    ax.set_ylabel('Bike Rentals');
```

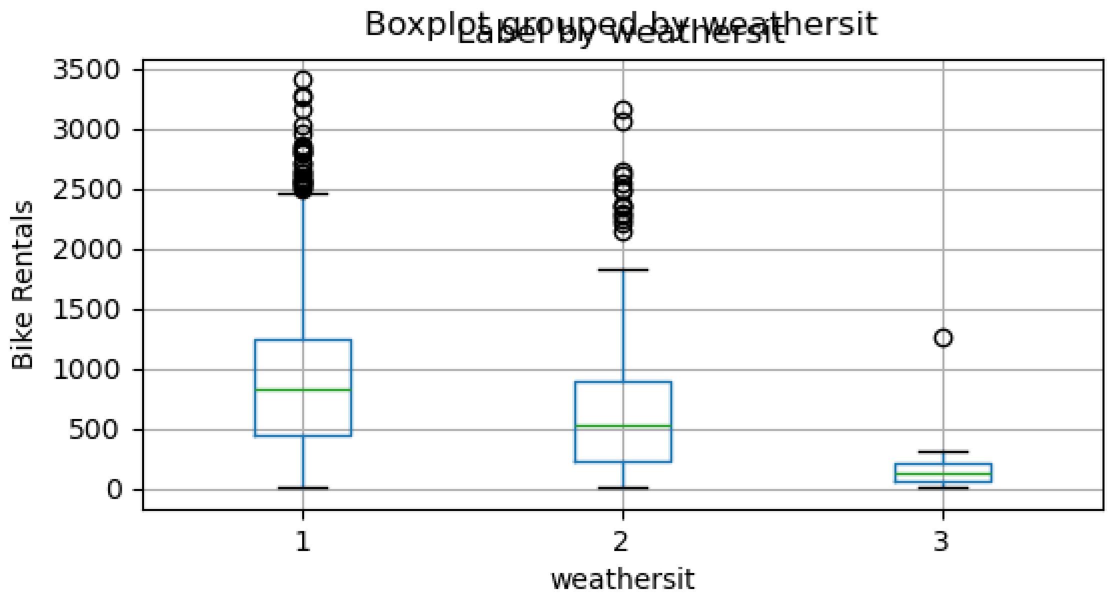





Boxplot grouped by holiday
Label by holiday
Bike Rentals
3500
3000
2500
2000
1500
1000
500
0
0
1
holiday


Boxplot grouped by weekday
Label by weekday
Bike Rentals
3500
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
weekday


Boxplot grouped by workingday
Label by workingday
Bike Rentals
3500
3000
2500
2000
1500
1000
500
0
0
1
workingday



Train a Regression Model

In [91]:
```
X, y = bike_data[['season','mnth', 'holiday','weekday','workingday','weathersit','t
```

In [92]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [93]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20, random_sta
print(X_train.shape[0], X_test.shape[0])
```

```
584 147
```

In [94]:
```
model = LinearRegression().fit(X_train, y_train)
predictions = model.predict(X_test)

mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)

rmse = np.sqrt(mse)
print("RMSE:", rmse)

r2 = r2_score(y_test, predictions)
print("R2:", r2)
```

```
MSE: 210673.09677936204
RMSE: 458.9913907464518
R2: 0.6013016737003891
```

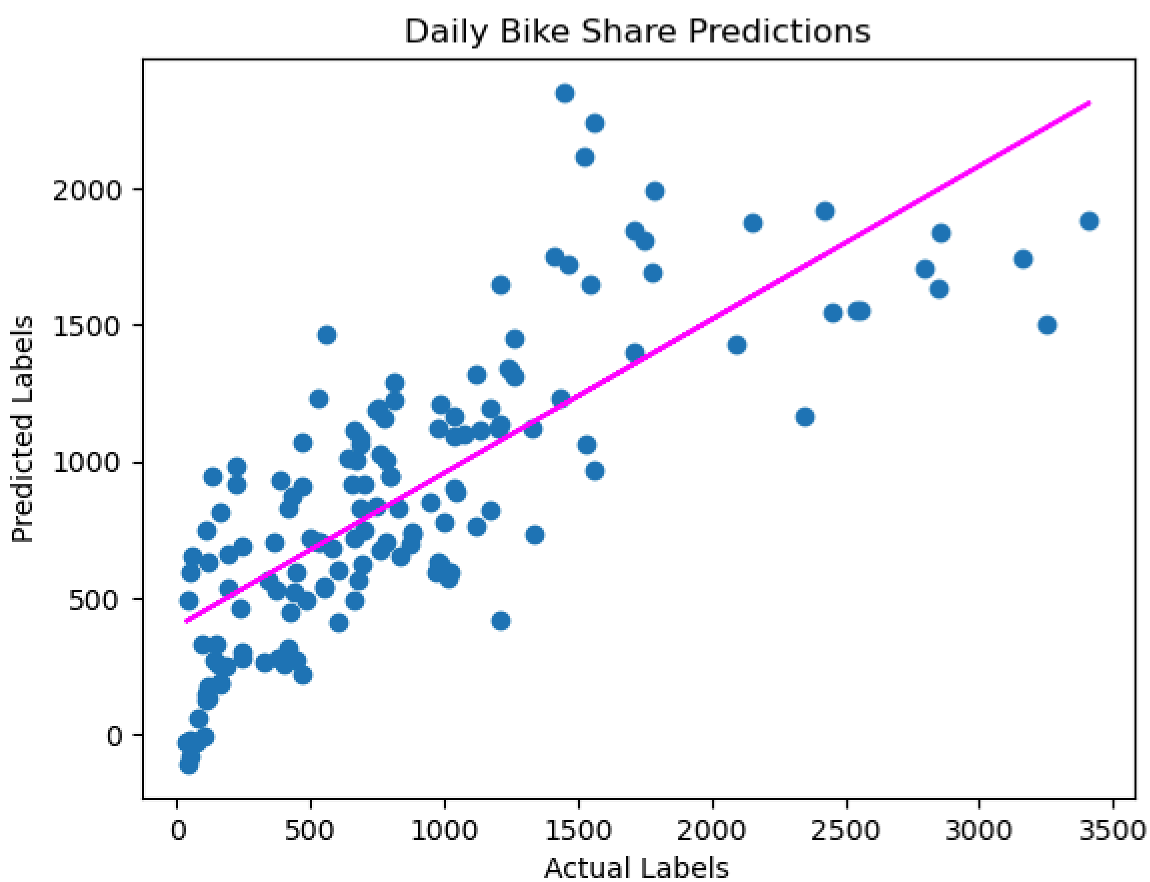
Laso Linear Regression

In [96]:
```
from sklearn.linear_model import Lasso
```

In [97]:
```python
# Fit a lasso model on the training set
model = Lasso().fit(X_train, y_train)
#print (model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
```

```
MSE: 210148.31184862508
RMSE: 458.4193624277067
R2: 0.6022948279130089
```

Out[97]: [<matplotlib.lines.Line2D at 0x17ae2f04530>]



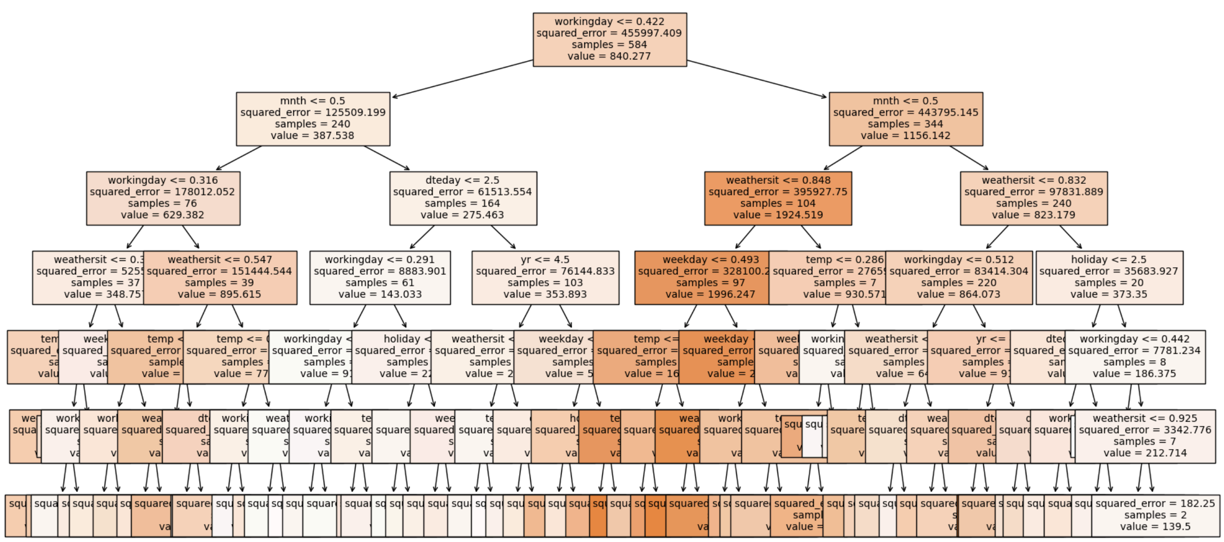
Decision Tree

In [99]:
```
from sklearn.tree import DecisionTreeRegressor
#from sklearn.tree import export_text

DT = DecisionTreeRegressor(random_state=42, max_depth=6).fit(X_train, y_train)

#tree = export_text(DT)
#print(tree)
DT_predictions = DT.predict(X_test)
```

In [100…
```
bike_data.columns
```

Out[100…
```
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'rentals'],
      dtype='object')
```

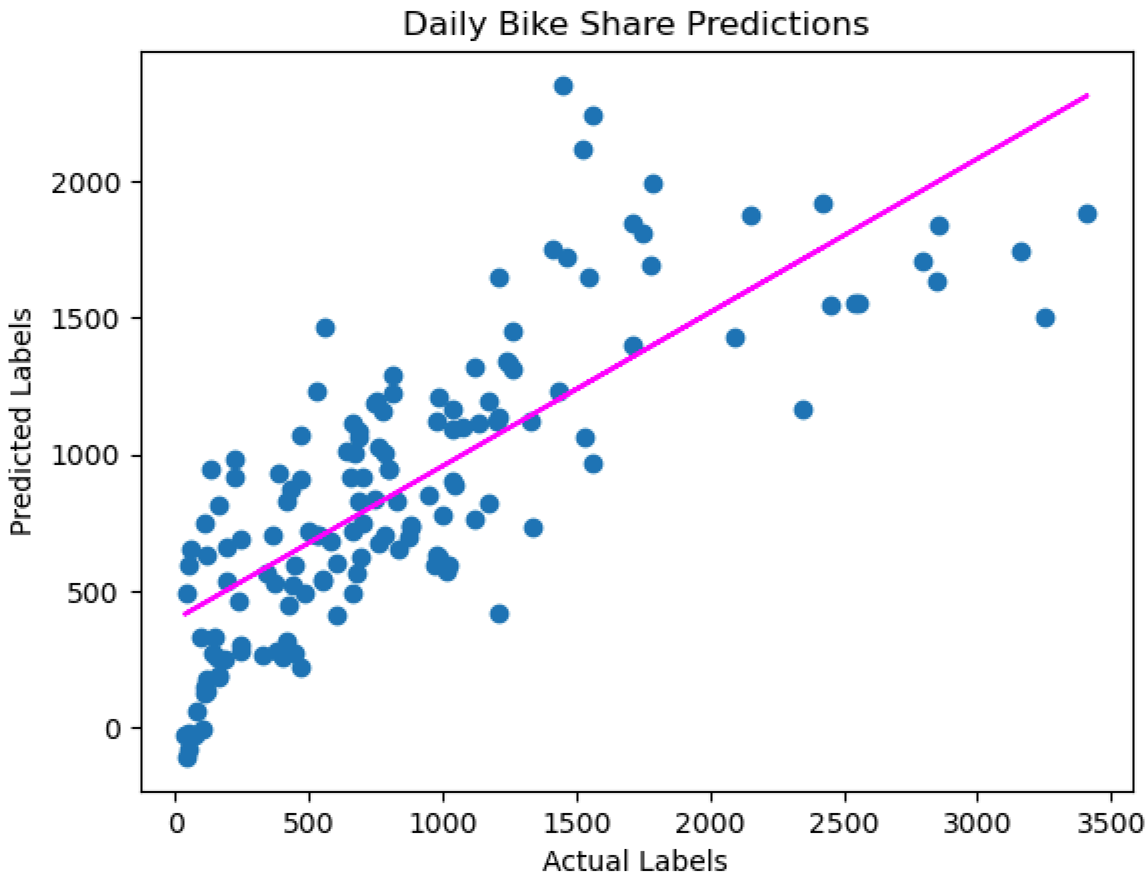
In [101…
```
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(DT, feature_names=bike_data.columns, filled=True, fontsize=10);
```



In [102…
```
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
```

```
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta');
```

```
MSE: 210148.31184862508
RMSE: 458.4193624277067
R2: 0.6022948279130089
```



In [ ]:

Ensemble Model: Random Forest Regressor

In [104…

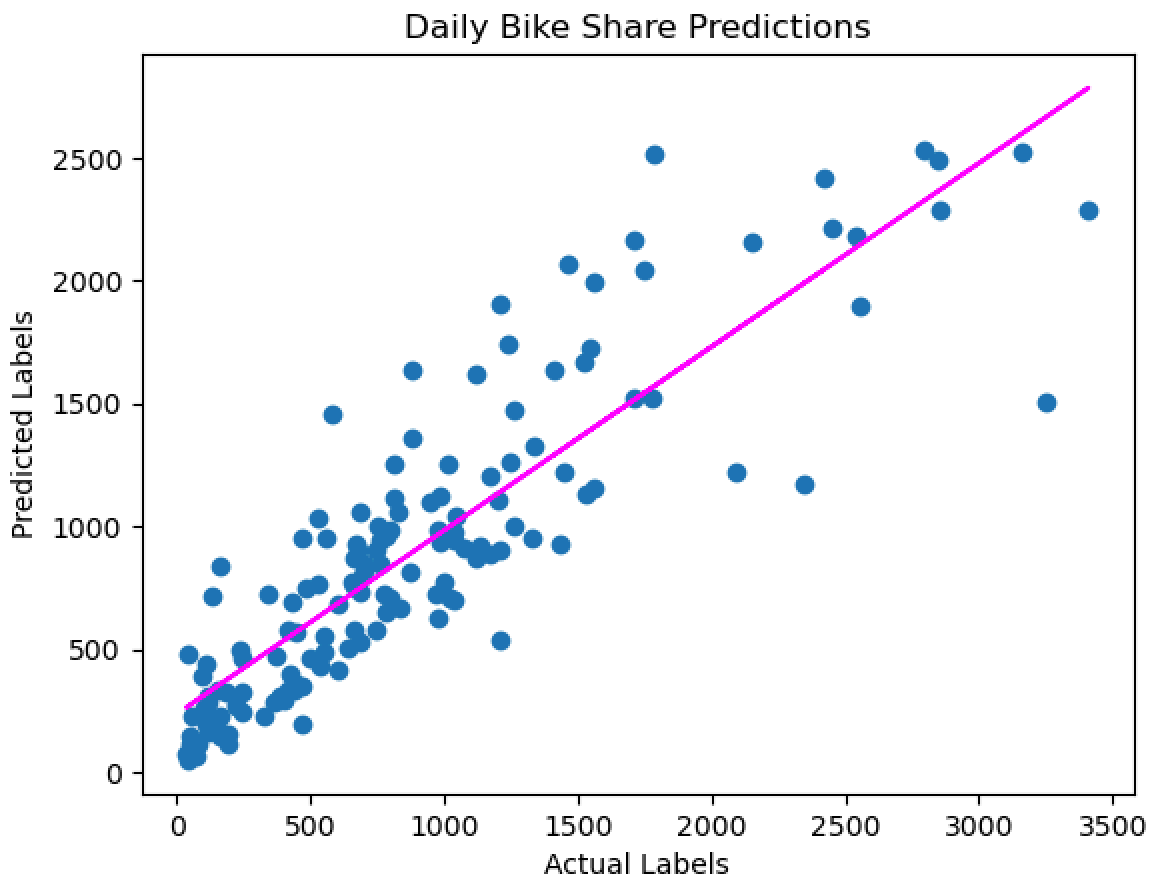
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor().fit(X_train, y_train)
print (model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
```

```
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
```

```
RandomForestRegressor()

MSE: 123507.22756462585
RMSE: 351.4359508710312
R2: 0.76626287044381
```

Out[104…

```
[<matplotlib.lines.Line2D at 0x17ae0eea810>]
```



Gradient Boosting Regressor

IT's like Random Forest, builds multiple trees. But instead of building them all independently and taking the average results, each tree is build on the output of the previous one in an attempt to incrementally reduce the loss in the model.

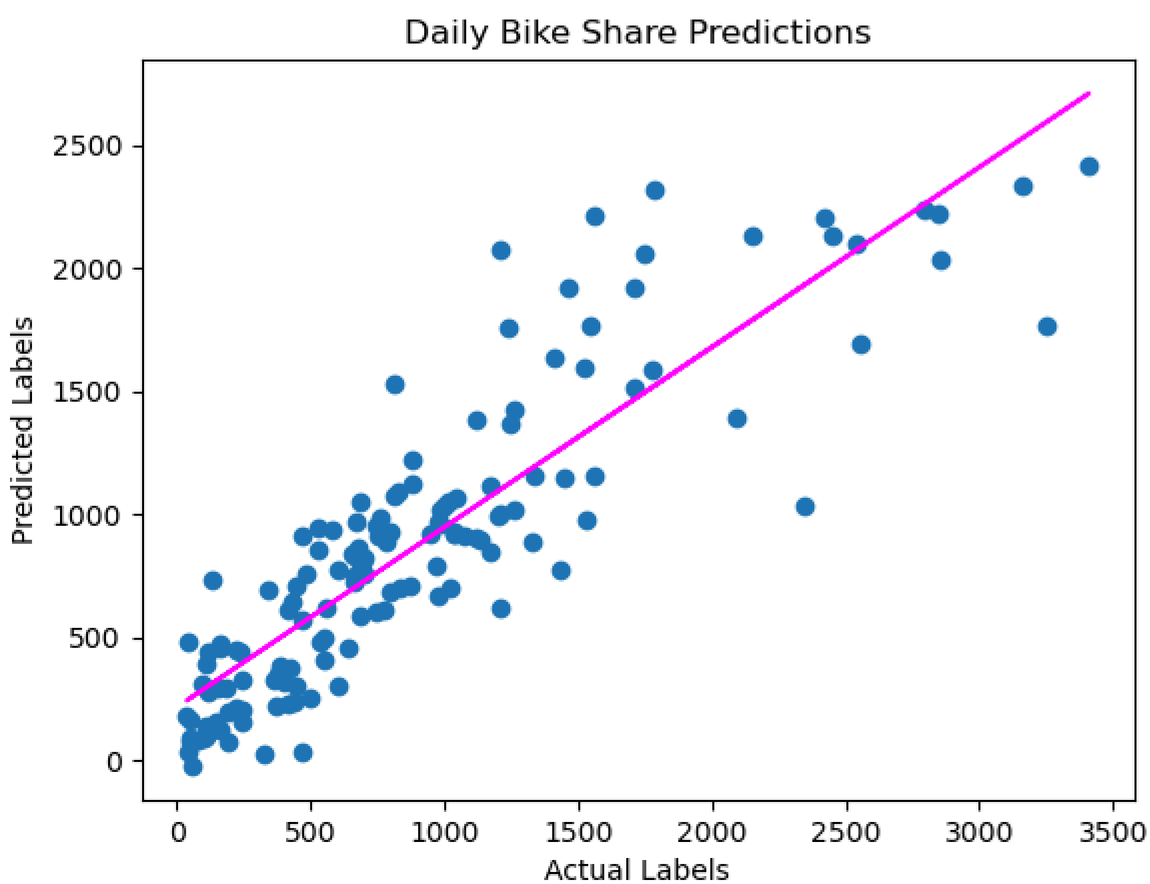
In [106…

```
# Train the model
from sklearn.ensemble import GradientBoostingRegressor

# Fit a lasso model on the training set
model = GradientBoostingRegressor().fit(X_train, y_train)
print (model, "\n")
```

```python
# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
```

```
GradientBoostingRegressor()

MSE: 118690.52589600885
RMSE: 344.5149138948978
R2: 0.7753784667060769
```

Out[106… `[<matplotlib.lines.Line2D at 0x17ae2d21a60>]`



HyperParameter Tunning

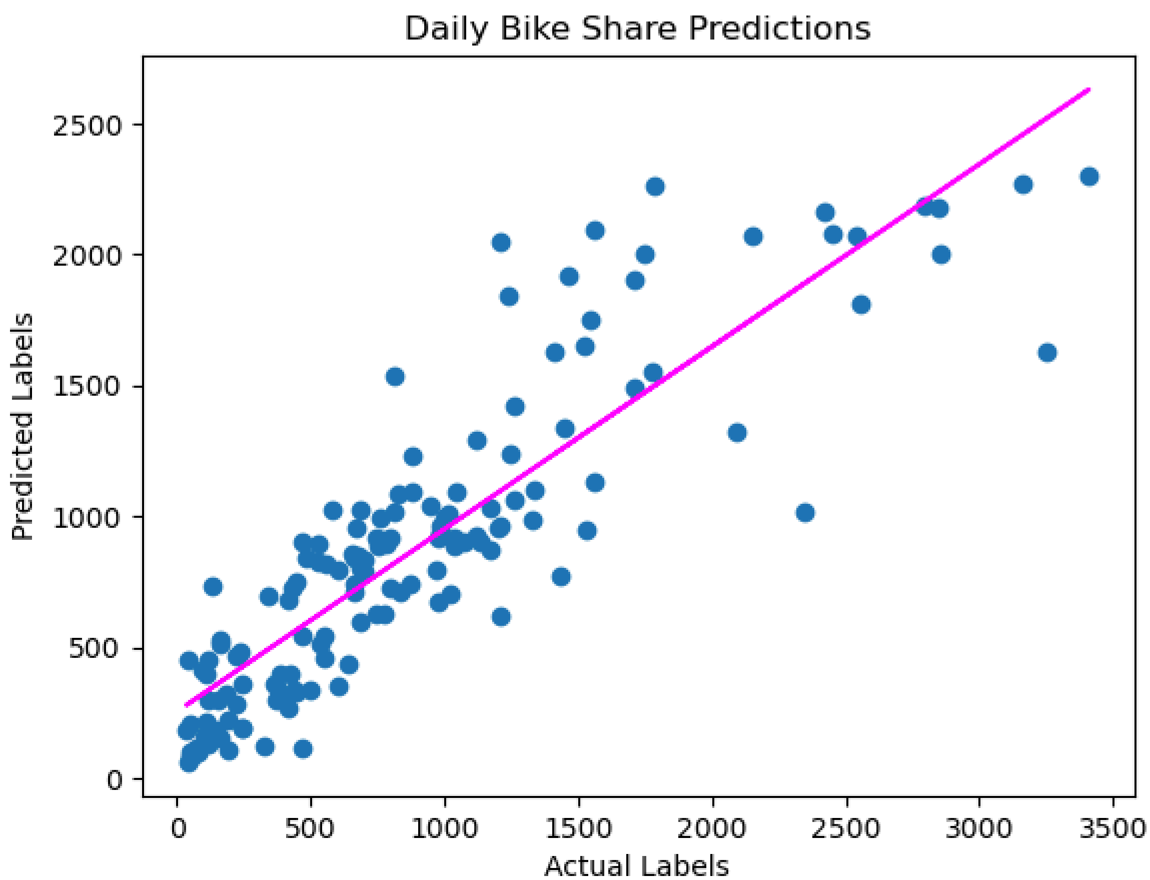
In [108…

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, r2_score

# Use a Gradient Boosting algorithm
alg = GradientBoostingRegressor()

# Try these hyperparameter values
params = {
 'learning_rate': [0.1, 0.5, 1.0],
 'n_estimators' : [50, 100, 150]
 }

# Find the best hyperparameter combination to optimize the R2 metric
score = make_scorer(r2_score)
gridsearch = GridSearchCV(alg, params, scoring=score, cv=3, return_train_score=True
gridsearch.fit(X_train, y_train)
print("Best parameter combination:", gridsearch.best_params_, "\n")

# Get the best model
model=gridsearch.best_estimator_
print(model, "\n")

# Evaluate the model using the test data
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
# overlay the regression line
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
```

```
Best parameter combination: {'learning_rate': 0.1, 'n_estimators': 50}

GradientBoostingRegressor(n_estimators=50)

MSE: 124171.14455564888
RMSE: 352.3792623802497
R2: 0.7650064091434857
```

Out[108…

```
[<matplotlib.lines.Line2D at 0x17ae0f840b0>]
```



Note: The use of random values in the Gradient Boosting algorithm results in slightly different metrics each time. In this case, the best model produced by hyperparameter tuning is unlikely to be significantly better than one trained with the default hyperparameter values; but it's still useful to know about the hyperparameter tuning technique!

Pipeline in Scikit-Learn

In [111…

```
# Train the model
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
import numpy as np

# Define preprocessing for numeric columns (scale them)
numeric_features = [6,7,8,9]
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())])

# Define preprocessing for categorical features (encode them)
categorical_features = [0,1,2,3,4,5]
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Combine preprocessing steps
```

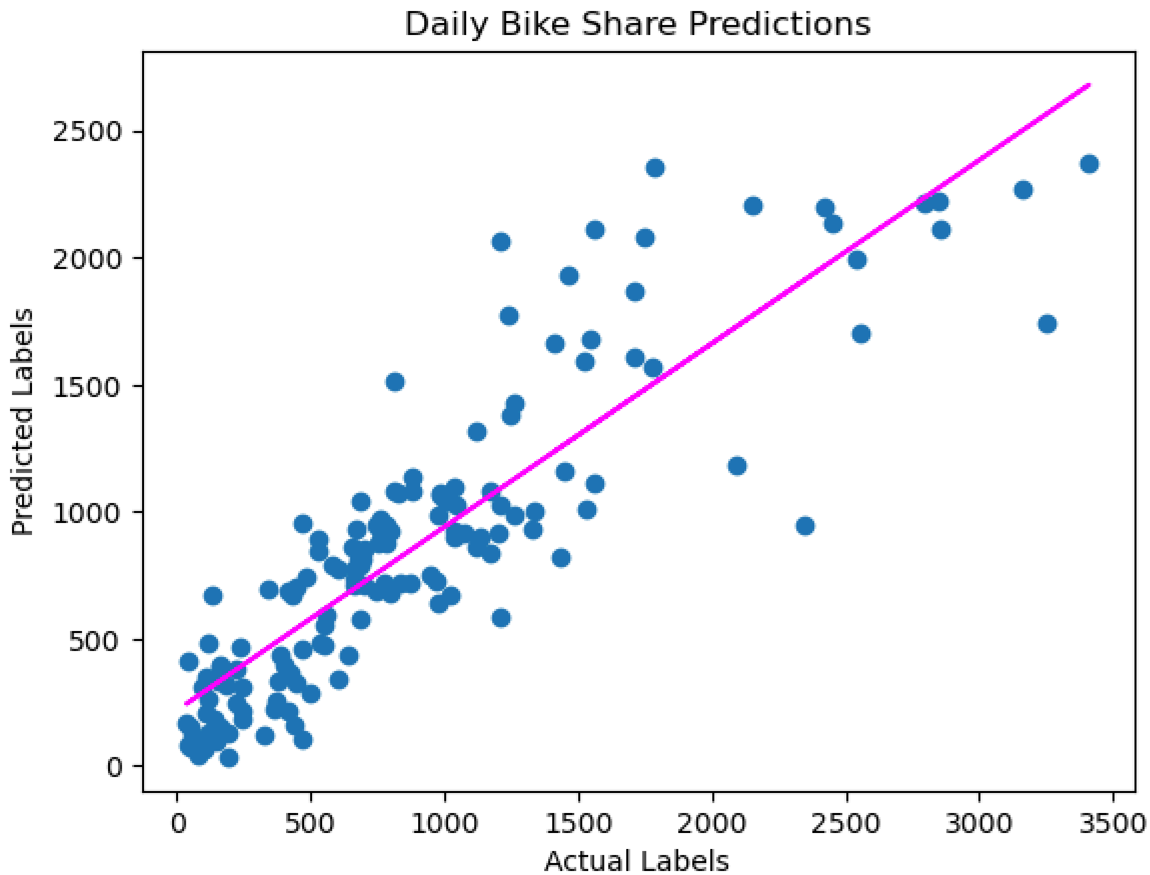
```python
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Create preprocessing and training pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', GradientBoostingRegressor())])


# fit the pipeline to train a linear regression model on the training set
model = pipeline.fit(X_train, (y_train))
print (model)
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler
())]),
                                                  [6, 7, 8, 9]),
                                                 ('cat',
                                                  Pipeline(steps=[('onehot',
                                                                   OneHotEncoder(han
dle_unknown='ignore'))]),
                                                  [0, 1, 2, 3, 4, 5])])),
                ('regressor', GradientBoostingRegressor())])
```

In [112…

```python
# Get predictions
predictions = model.predict(X_test)

# Display metrics
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions')
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta');
```

```
MSE: 121126.3693885471
RMSE: 348.03213844205123
R2: 0.7707686387857141
```



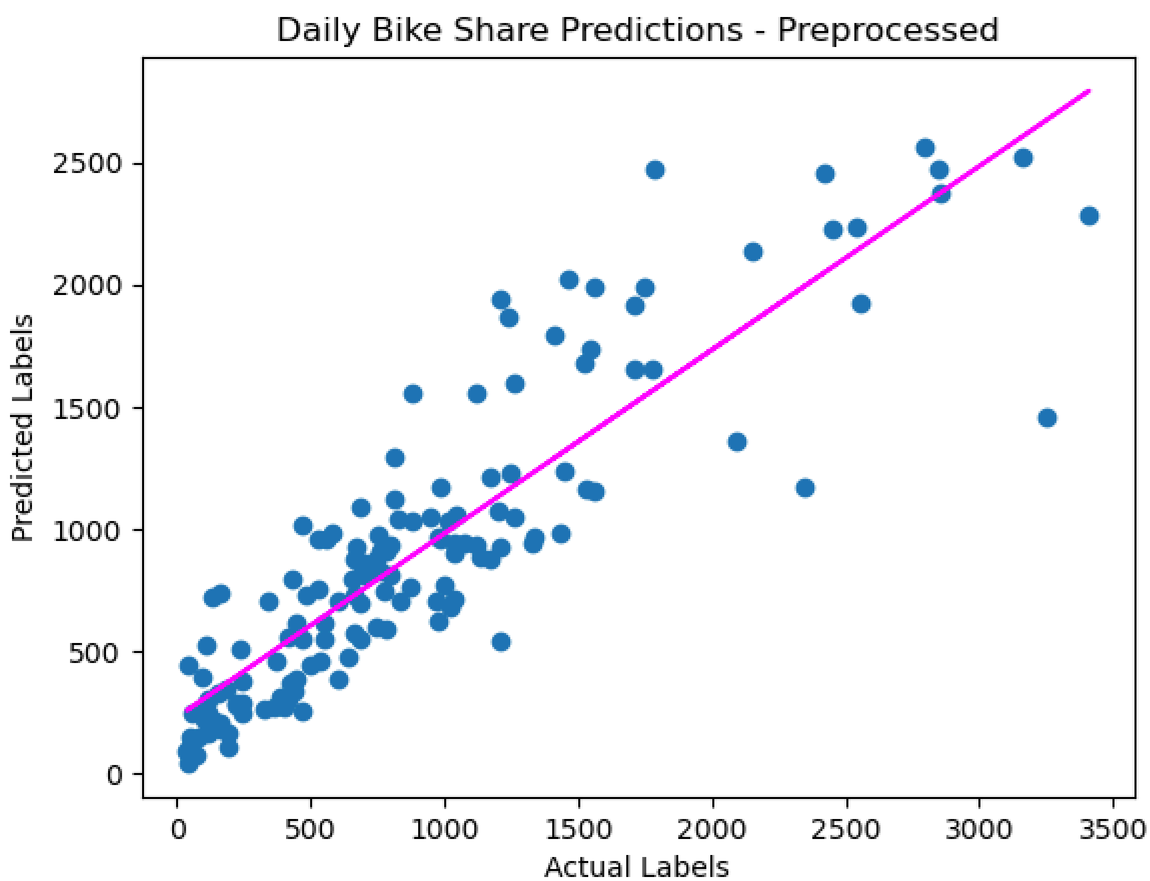
In [113…

```
# Use a different estimator in the pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('regressor', RandomForestRegressor())])


# fit the pipeline to train a linear regression model on the training set
model = pipeline.fit(X_train, (y_train))
print (model, "\n")

# Get predictions
predictions = model.predict(X_test)

# Display metrics
mse = mean_squared_error(y_test, predictions)
print("MSE:", mse)
rmse = np.sqrt(mse)
print("RMSE:", rmse)
r2 = r2_score(y_test, predictions)
print("R2:", r2)

# Plot predicted vs actual
plt.scatter(y_test, predictions)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Daily Bike Share Predictions - Preprocessed')
z = np.polyfit(y_test, predictions, 1)
p = np.poly1d(z)
```

```
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler
()),
                                                  [6, 7, 8, 9]),
                                                 ('cat',
                                                  Pipeline(steps=[('onehot',
                                                                   OneHotEncoder(han
dle_unknown='ignore'))]),
                                                  [0, 1, 2, 3, 4, 5])])),
                ('regressor', RandomForestRegressor())])
```

```
MSE: 114696.91759863944
RMSE: 338.66933371452376
R2: 0.7829363607532935
```



Save the Trained Model

In [115…

```
import joblib

# save the model as a pickle file
filename= './bike-share.pkl'
joblib.dump(model, filename)
```

Out[115… ['./bike-share.pkl']

In [116…

```
# Load the model from the file
loaded_model = joblib.load(filename)

# Create a numpy array containing a new observation (for example tomorrow's seasona
X_new = np.array([[1,1,0,3,1,1,0.226957,0.22927,0.436957,0.1869]]).astype('float64'
print ('New sample: {}'.format(list(X_new[0])))

# Use the model to predict tomorrow's rentals
result = loaded_model.predict(X_new)
print('Prediction: {:.0f} rentals'.format(np.round(result[0])))
```

```
New sample: [1.0, 1.0, 0.0, 3.0, 1.0, 1.0, 0.226957, 0.22927, 0.436957, 0.1869]
Prediction: 109 rentals
```

In [117…

```
# Load the model from the file
loaded_model = joblib.load(filename)

# Create a numpy array containing a new observation (for example tomorrow's seasona
X_new = np.array([[1,1,0,3,1,1,0.226957,0.22927,0.436957,0.1869]]).astype('float64'
print ('New sample: {}'.format(list(X_new[0])))

# Use the model to predict tomorrow's rentals
result = loaded_model.predict(X_new)
print('Prediction: {:.0f} rentals'.format(np.round(result[0])))
```

```
New sample: [1.0, 1.0, 0.0, 3.0, 1.0, 1.0, 0.226957, 0.22927, 0.436957, 0.1869]
Prediction: 109 rentals
```

In [118…

```
# An array of features based on five-day weather forecast
X_new = np.array([[0,1,1,0,0,1,0.344167,0.363625,0.805833,0.160446],
                  [0,1,0,1,0,1,0.363478,0.353739,0.696087,0.248539],
                  [0,1,0,2,0,1,0.196364,0.189405,0.437273,0.248309],
                  [0,1,0,3,0,1,0.2,0.212122,0.590435,0.160296],
                  [0,1,0,4,0,1,0.226957,0.22927,0.436957,0.1869]])

# Use the model to predict rentals
results = loaded_model.predict(X_new)
print('5-day rental predictions:')
for prediction in results:
    print(np.round(prediction))
```

```
5-day rental predictions:
553.0
767.0
232.0
207.0
267.0
```