Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Домашнее задание №1
по дисциплине
«Методы машинного обучения»
на тему

Выполнил:
студент группы ИУ5-21М
Хтет Мин Паинг Вин

Москва — 2020 г.

1. Задание

Требуется выполнить следующие действия [1]:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или ре- грессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кроссвалидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

2. Ход выполнения работы

2.1. Выбор набора данных

В качестве набора данных используются метрологические данные с метеостанции HI-SEAS (Hawaii Space Exploration Analog and Simulation) за четыре месяца (с сентября по декабрь 2016 года) и использовался в соревновании Space Apps Moscow 2017 в категории «You are my Sunshine» для построения приложения для предсказания мощностисолнечного излучения и планирования работы исследовательской станции [2, 3]. Данный набор данных доступен по следующему адресу: https://www.kaggle.com/dronio/SolarEnergy.

2.2. Проведение разведочного анализа данных

In [0]:
```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

In [0]:
```
# Enable inline plots
%matplotlib inline
# Set plot style
sns.set(style="ticks")
# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

In [0]:
```
pd.set_option("display.width", 70)
```

In [0]:
```
data = pd.read_csv("./SolarPrediction.csv")
```

In [0]:
```
data["Time"] = (pd
.to_datetime(data["UNIXTime"], unit="s", utc=True)
.dt.tz_convert("Pacific/Honolulu")).dt.time
data["TimeSunRise"] = (pd
.to_datetime(data["TimeSunRise"],
infer_datetime_format=True)
.dt.time)
data["TimeSunSet"] = (pd
.to_datetime(data["TimeSunSet"],
infer_datetime_format=True)
.dt.time)
data = data.rename({"WindDirection(Degrees)": "WindDirection"},
axis=1)
```

In [6]:
```
data.dtypes
```

Out[6]:
```
UNIXTime          int64
Data             object
Time             object
Radiation       float64
Temperature       int64
Pressure        float64
Humidity          int64
WindDirection   float64
Speed           float64
TimeSunRise      object
TimeSunSet       object
dtype: object
```

In [7]:
```
data.head()
```

Out[7]:

	UNIXTime	Data	Time	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	TimeSunRise	Time
0	1475229326	9/29/2016 12:00:00 AM	23:55:26	1.21	48	30.46	59	177.39	5.62	06:13:00	18:13
1	1475229023	9/29/2016 12:00:00 AM	23:50:23	1.21	48	30.46	58	176.78	3.37	06:13:00	18:13
2	1475228726	9/29/2016 12:00:00 AM	23:45:26	1.23	48	30.46	57	158.75	3.37	06:13:00	18:13
3	1475228421	9/29/2016 12:00:00 AM	23:40:21	1.21	48	30.46	60	137.71	3.37	06:13:00	18:13
4	1475228124	9/29/2016 12:00:00 AM	23:35:24	1.17	48	30.46	62	104.95	5.62	06:13:00	18:13

In [0]:
```
def time_to_second(t):
  return ((datetime.combine(datetime.min, t) - datetime.min)
.total_seconds())
```

In [10]:
```
df = data.copy()
timeInSeconds = df["Time"].map(time_to_second)
sunrise = df["TimeSunRise"].map(time_to_second)
sunset = df["TimeSunSet"].map(time_to_second)
df["DayPart"] = (timeInSeconds - sunrise) / (sunset - sunrise)
df = df.drop(["UNIXTime", "Data", "Time",
"TimeSunRise", "TimeSunSet"], axis=1)
df.head()
```

Out[10]:

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
0	1.21	48	30.46	59	177.39	5.62	1.475602
1	1.21	48	30.46	58	176.78	3.37	1.468588
2	1.23	48	30.46	57	158.75	3.37	1.461713
3	1.21	48	30.46	60	137.71	3.37	1.454653
4	1.17	48	30.46	62	104.95	5.62	1.447778

In [11]:
```
df.dtypes
```

Out[11]:
```
Radiation       float64
Temperature       int64
Pressure        float64
Humidity          int64
WindDirection   float64
Speed           float64
DayPart         float64
dtype: object
```

In [12]:
```
df.shape
```

Out[12]: (32686, 7)

In [13]:
```
df.describe()
```

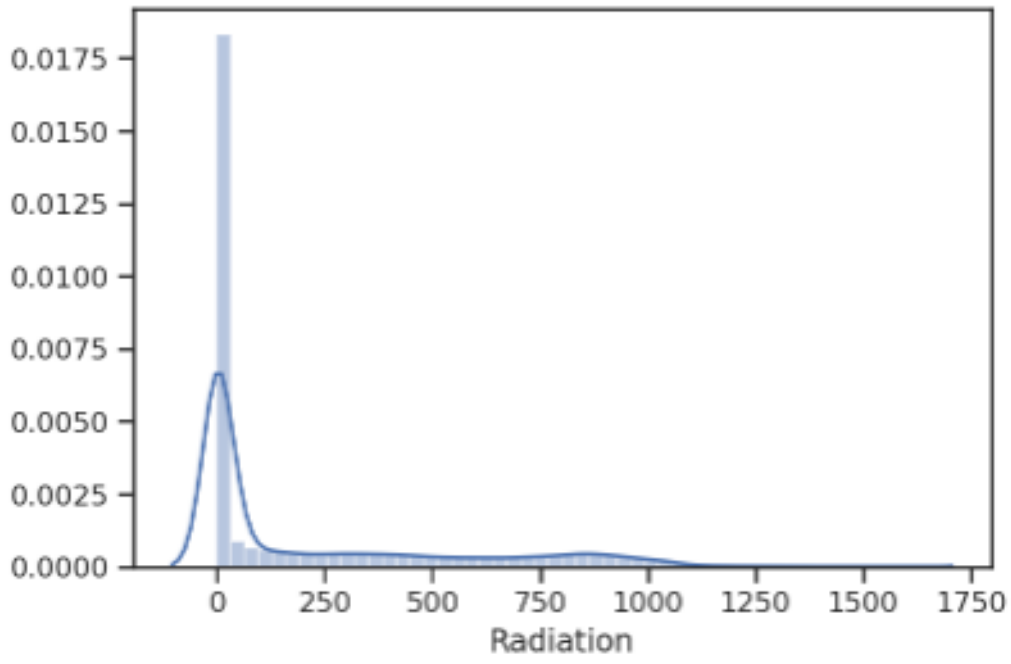
Out[13]:

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
count	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000	32686.000000
mean	207.124697	51.103255	30.422879	75.016307	143.489821	6.243869	0.482959
std	315.916387	6.201157	0.054673	25.990219	83.167500	3.490474	0.602432
min	1.110000	34.000000	30.190000	8.000000	0.090000	0.000000	-0.634602
25%	1.230000	46.000000	30.400000	56.000000	82.227500	3.370000	-0.040139
50%	2.660000	50.000000	30.430000	85.000000	147.700000	5.620000	0.484332
75%	354.235000	55.000000	30.460000	97.000000	179.310000	7.870000	1.006038
max	1601.260000	71.000000	30.560000	103.000000	359.950000	40.500000	1.566061

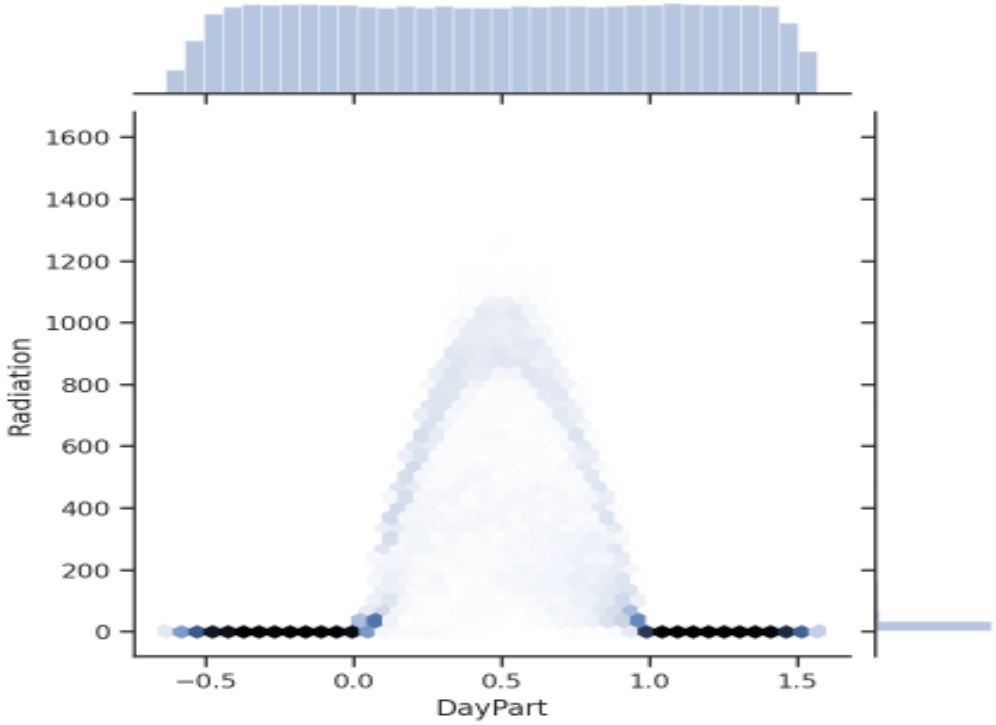
In [14]:
```
df.isnull().sum()
```

Out[14]:
```
Radiation      0
Temperature    0
Pressure       0
Humidity       0
WindDirection  0
Speed          0
DayPart        0
dtype: int64
```

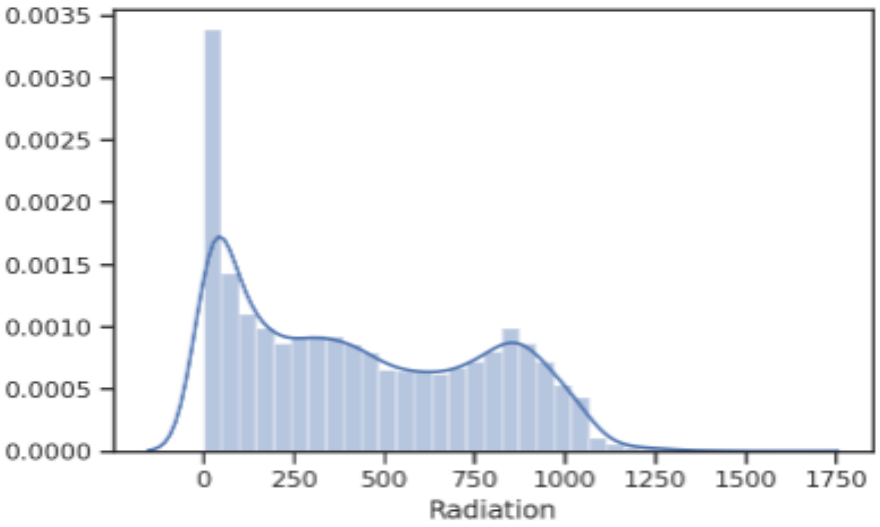
In [15]:
```
sns.distplot(df["Radiation"]);
```



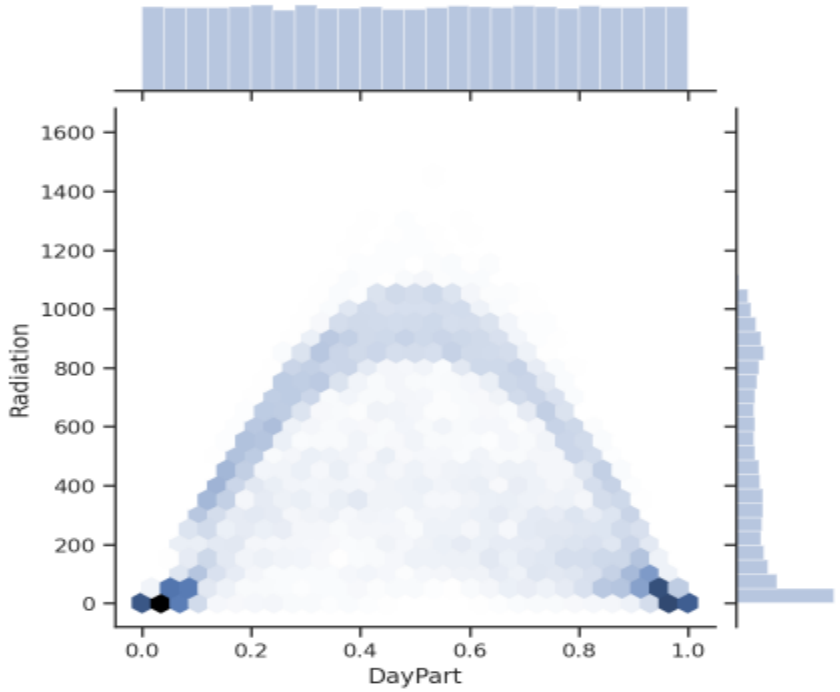
In [16]:
```
sns.jointplot(x="DayPart", y="Radiation", data=df, kind="hex");
```



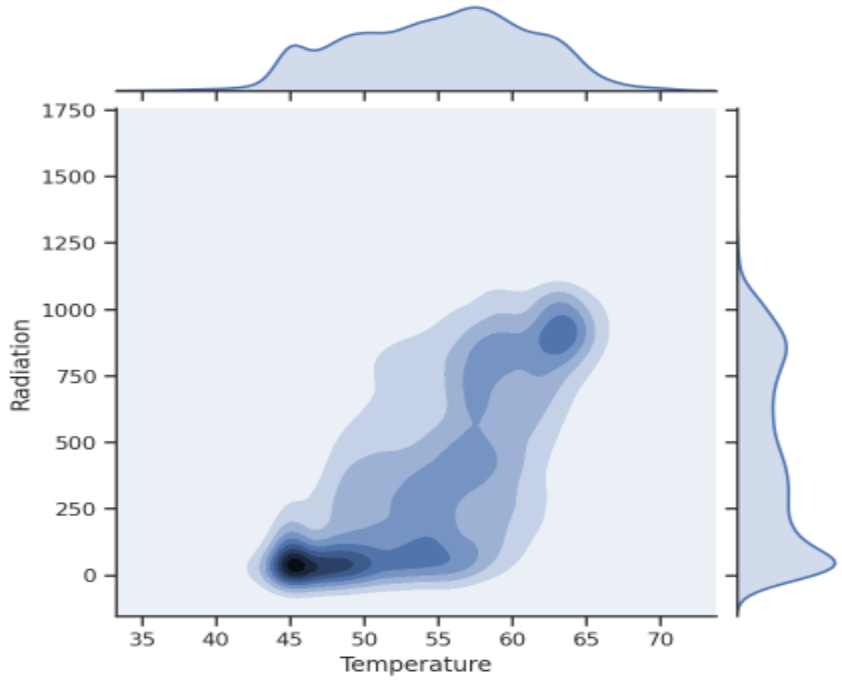
In [17]:
```
dfd = df[(df["DayPart"] >= 0) & (df["DayPart"] <= 1)]
sns.distplot(dfd["Radiation"]);
```



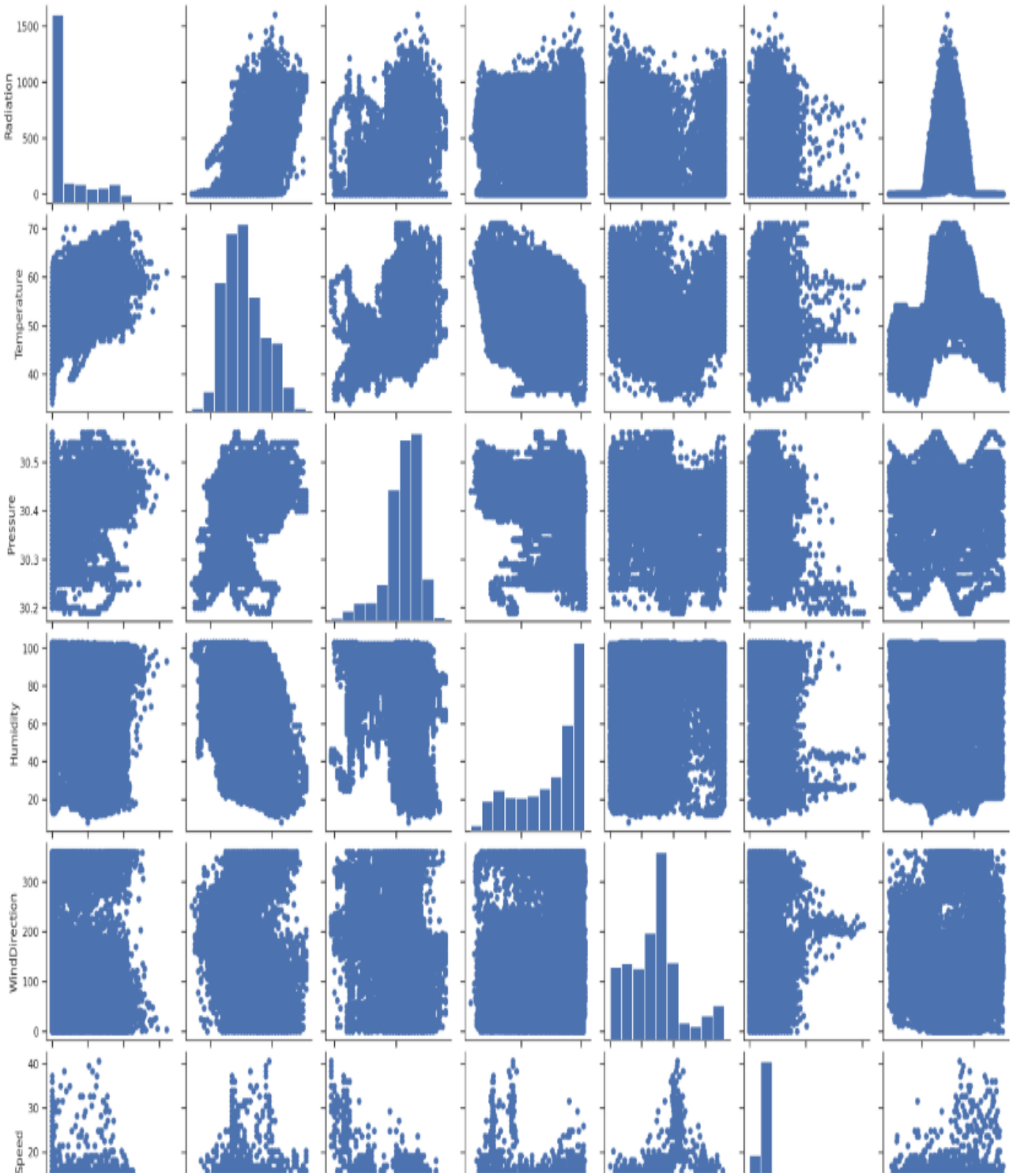
In [18]:
```
sns.jointplot(x="DayPart", y="Radiation", data=dfd, kind="hex");
```



In [19]:
```
sns.jointplot(x="Temperature", y="Radiation", data=dfd, kind="kde");
```



In [20]:
```
sns.pairplot(df, plot_kws=dict(linewidth=0));
```

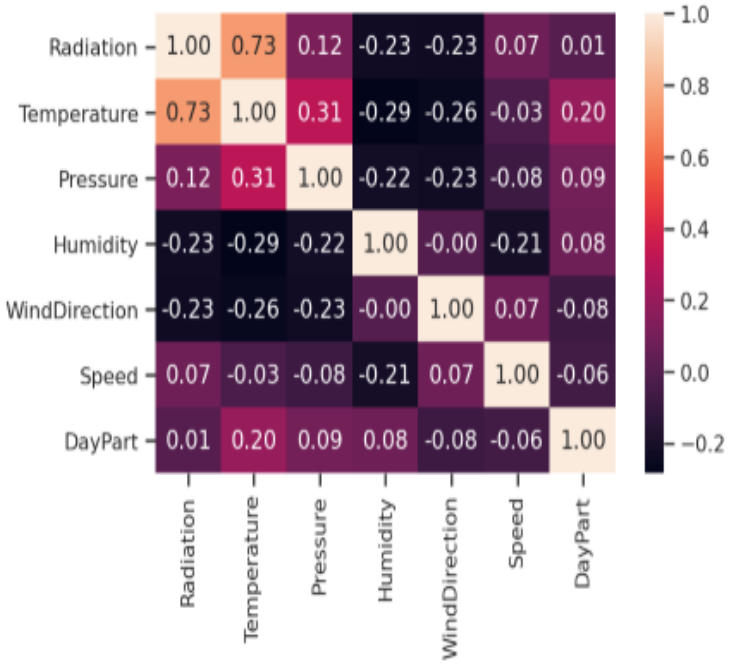


In [21]:
```
df.corr()
```

Out[21]:

	Radiation	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
Radiation	1.000000	0.734955	0.119016	-0.226171	-0.230324	0.073627	0.005980
Temperature	0.734955	1.000000	0.311173	-0.285055	-0.259421	-0.031458	0.198520
Pressure	0.119016	0.311173	1.000000	-0.223973	-0.229010	-0.083639	0.094403
Humidity	-0.226171	-0.285055	-0.223973	1.000000	-0.001833	-0.211624	0.075513
WindDirection	-0.230324	-0.259421	-0.229010	-0.001833	1.000000	0.073092	-0.078130
Speed	0.073627	-0.031458	-0.083639	-0.211624	0.073092	1.000000	-0.056095
DayPart	0.005980	0.198520	0.094403	0.075513	-0.078130	-0.056095	1.000000

In [22]:
```
sns.heatmap(df.corr(), annot=True, fmt=".2f");
```



2.3. Подготовка данных для обучения моделей

In [0]:
```
X = df.drop("Radiation", axis=1)
y = df["Radiation"]
```

In [24]:
```
print(X.head(), "\n")
print(y.head())
```

```
   Temperature  Pressure  Humidity  WindDirection  Speed  DayPart
0           48     30.46        59         177.39   5.62  1.475602
1           48     30.46        58         176.78   3.37  1.468588
2           48     30.46        57         158.75   3.37  1.461713
3           48     30.46        60         137.71   3.37  1.454653
4           48     30.46        62         104.95   5.62  1.447778

0    1.21
1    1.21
2    1.23
3    1.21
4    1.17
Name: Radiation, dtype: float64
```

In [25]:
```
print(X.shape)
print(y.shape)
```

```
(32686, 6)
(32686,)
```

In [26]:
```
from sklearn.preprocessing import StandardScaler
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

Out[26]:

	Temperature	Pressure	Humidity	WindDirection	Speed	DayPart
count	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04	3.268600e+04
mean	8.257741e-15	-8.589409e-14	9.563964e-16	-6.186353e-16	-2.072571e-14	-2.846377e-17
std	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00	1.000015e+00
min	-2.758117e+00	-4.259540e+00	-2.578560e+00	-1.724255e+00	-1.788859e+00	-1.855112e+00
25%	-8.229646e-01	-4.184734e-01	-7.316829e-01	-7.366250e-01	-8.233591e-01	-8.683240e-01
50%	-1.779139e-01	1.302504e-01	3.841386e-01	5.062367e-02	-1.787376e-01	2.279483e-03
75%	6.283995e-01	6.789742e-01	8.458578e-01	4.307058e-01	4.658840e-01	8.682924e-01
max	3.208603e+00	2.508053e+00	1.076717e+00	2.602741e+00	9.814329e+00	1.797910e+00

2.4. Выбор метрик

In [0]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import r2_score
def test_model(model):
  print("mean_absolute_error:",
mean_absolute_error(y_test, model.predict(X_test)))
  print("median_absolute_error:",
median_absolute_error(y_test, model.predict(X_test)))
  print("r2_score:",
r2_score(y_test, model.predict(X_test)))
```

In [0]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [0]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=346705925)
```

In [31]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(24514, 6)
(8172, 6)
(24514,)
(8172,)
```

In [32]:
```
knn_5 = KNeighborsRegressor(n_neighbors=5)
knn_5.fit(X_train, y_train)
```

Out[32]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

In [33]:
```
test_model(knn_5)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.017000000000004
r2_score: 0.8677873476991447
```

In [34]:
```
dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)
```

Out[34]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [35]:
```
test_model(dt_none)
```

```
mean_absolute_error: 50.341986049926575
median_absolute_error: 0.7149999999999999
r2_score: 0.8292768488175892
```

In [36]:
```
ran_100 = RandomForestRegressor(n_estimators=100)
ran_100.fit(X_train, y_train)
```

Out[36]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [37]:
```
test_model(ran_100)
```

```
mean_absolute_error: 37.981940332843855
median_absolute_error: 0.594450000000001
r2_score: 0.9159077026674405
```

In [0]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
```

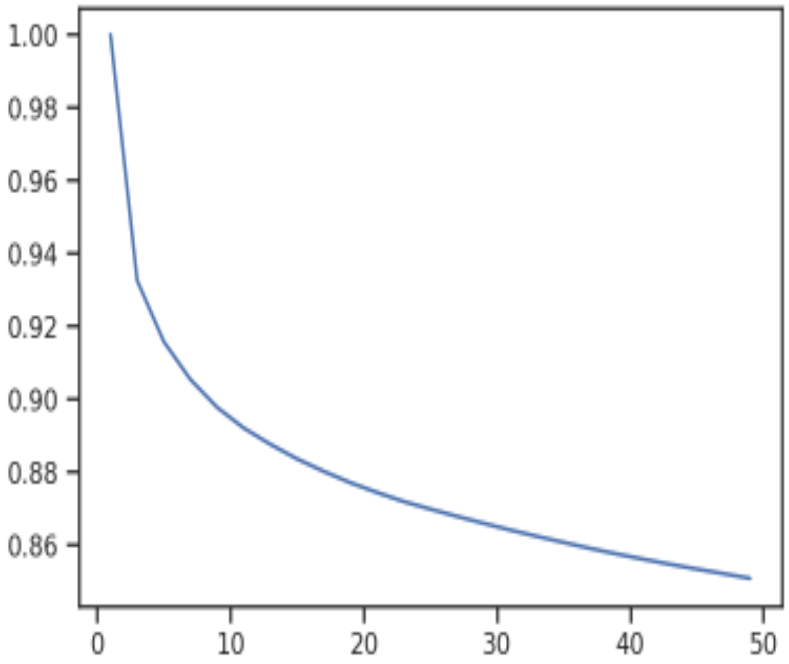
In [39]:
```
param_range = np.arange(1, 50, 2)
tuned_parameters = [{'n_neighbors': param_range}]
tuned_parameters
```

Out[39]:
```
[{'n_neighbors': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
        35, 37, 39, 41, 43, 45, 47, 49])}]
```

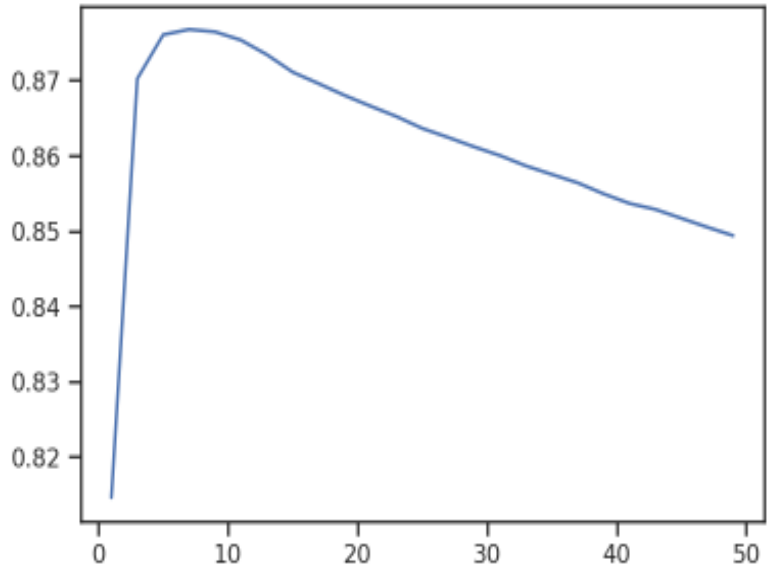
In [40]:
```
gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
cv=ShuffleSplit(n_splits=10), scoring="r2",
return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

Out[40]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                    weights='uniform')
```

In [41]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



In [42]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



In [43]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 56.07154831829942
median_absolute_error: 4.7735714285714295
r2_score: 0.8687906728428422
```

In [44]:
```
test_model(knn_5)
```

```
mean_absolute_error: 55.39857905041605
median_absolute_error: 4.017000000000004
r2_score: 0.8677873476991447
```

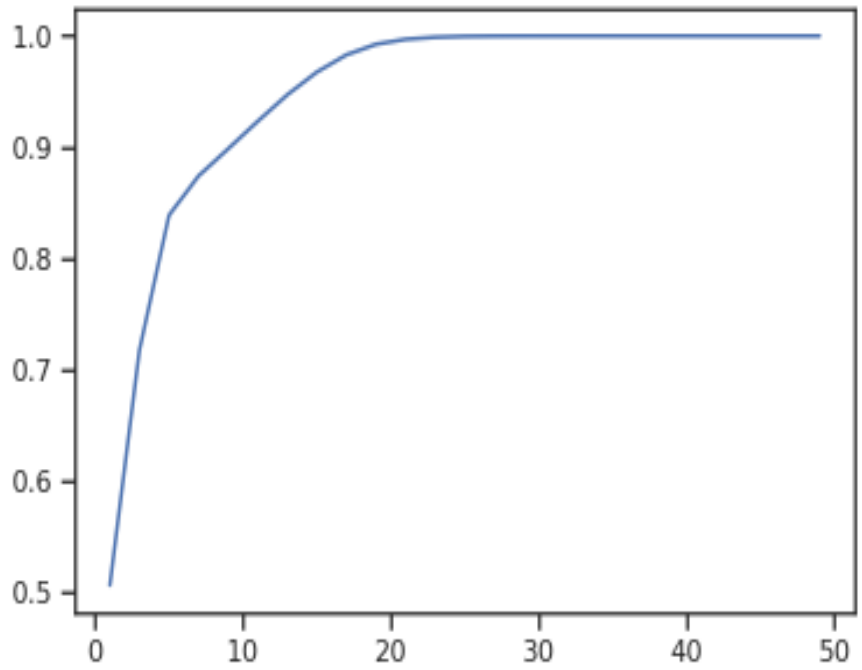
In [45]:
```
param_range = np.arange(1, 50, 2)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

Out[45]:
```
[{'max_depth': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
        35, 37, 39, 41, 43, 45, 47, 49])}]
```

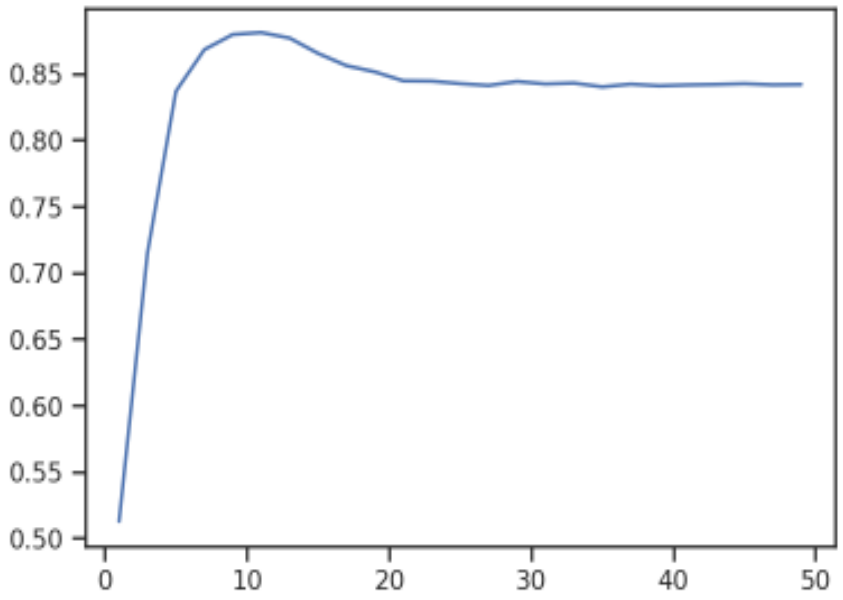
In [46]:
```
gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters,
cv=ShuffleSplit(n_splits=10), scoring="r2",
return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

Out[46]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=11,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [47]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



In [48]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



In [49]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 48.484004365068806
median_absolute_error: 0.8948072344320924
r2_score: 0.8702592310645209
```

In [50]:
```
test_model(dt_none)
```

```
mean_absolute_error: 50.341986049926575
median_absolute_error: 0.7149999999999999
r2_score: 0.8292768488175892
```

In [51]:
```
param_range = np.arange(20, 201, 20)
tuned_parameters = [{'n_estimators': param_range}]
tuned_parameters
```

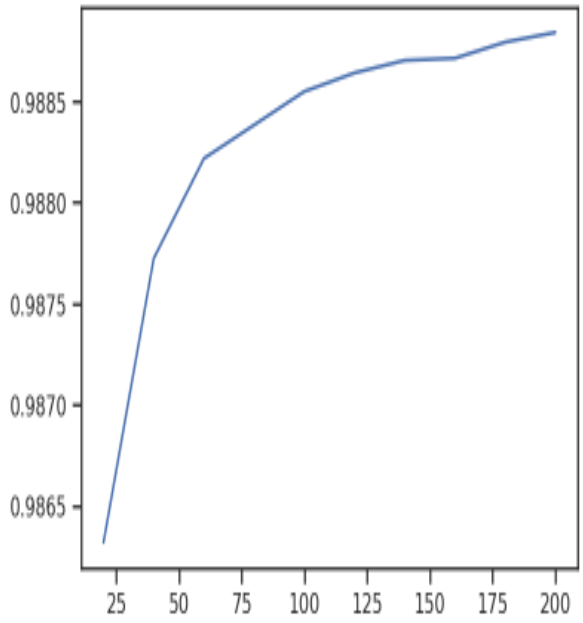
Out[51]:
```
[{'n_estimators': array([ 20,  40,  60,  80, 100, 120, 140, 160, 180, 200])}]
```

In [52]:
```
gs = GridSearchCV(RandomForestRegressor(), tuned_parameters,
cv=ShuffleSplit(n_splits=10), scoring="r2",
return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

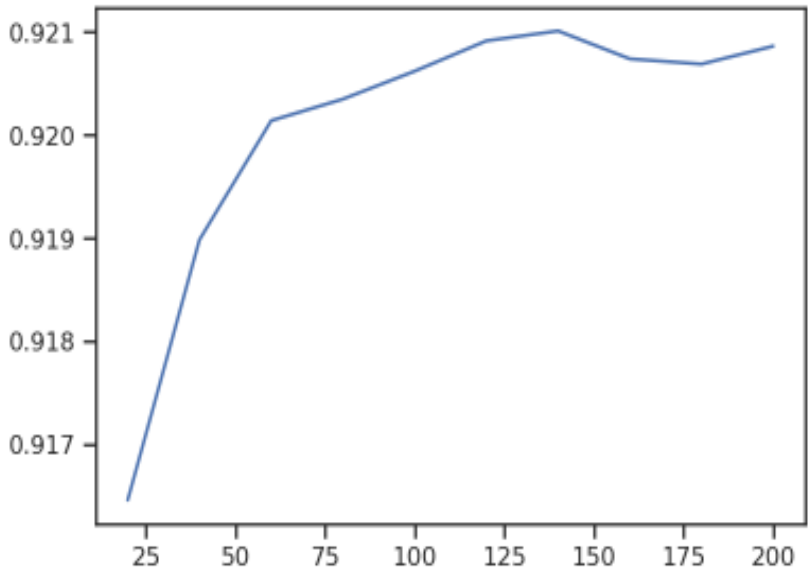
```
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

Out[52]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=140, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [54]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



In [55]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



In [56]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 37.84232099153906
median_absolute_error: 0.6201428571428569
r2_score: 0.9164532720317664
```

In [57]:
```
test_model(ran_100)
```

```
mean_absolute_error: 37.981940332843855
median_absolute_error: 0.594450000000001
r2_score: 0.9159077026674405
```

3. Выводы

Все построенные модели обладают очень хорошими показателями. Ансамблевая модель при этом обладает наилучшими характеристиками. Таким образом для дальнейшей работы стоит использовать именно ее.

Список литературы

[1] Гапанюк Ю. Е. Домашнее задание по дисциплине «Методы машинного обучения»[Электронный ресурс] // GitHub. — 2019. — Режим доступа: https://github.com/ugapanyuk/ml_course/wiki/MMO_DZ (дата обращения: 06.05.2019).

[2] You are my Sunshine [Electronic resource] // Space Apps Challenge. — 2017. Access mode: https://2017.spaceappschallenge.org/challenges/earth-and-us/ you-are-my-sunshine/details (online; accessed: 22.02.2019).

[3] dronio. Solar Radiation Prediction [Electronic resource] // Kaggle. — 2017. — Access mode: https://www.kaggle.com/dronio/SolarEnergy (online; accessed: 18.02.2019).

[4] Team The IPython Development. IPython 7.3.0 Documentation [Electronic resource] //Read the Docs. — 2019. — Access mode: https://ipython.readthedocs.io/en/stable/ (online; accessed: 20.02.2019).

[5] Waskom M. seaborn 0.9.0 documentation [Electronic resource] // PyData. — 2018. Access mode: https://seaborn.pydata.org/ (online; accessed: 20.02.2019).

[6] pandas 0.24.1 documentation [Electronic resource] // PyData. — 2019. — Access mode:http://pandas.pydata.org/pandas-docs/stable/ (online; accessed: 20.02.2019).

[7] Chrétien M. Convert datetime.time to seconds [Electronic resource] // Stack Overflow. 2017. — Access mode: https://stackoverflow.com/a/44823381 (online; accessed:20.02.2019).