How to Ace the Data Science Coding

1. Interview flow

1.1. Data Import and Exploration

```
# Import Libraries
import pandas as pd
import numpy as np

# Read Data from Various Formats
df_csv = pd.read_csv('data.csv')
df_excel = pd.read_excel('data.xlsx')
df_json = pd.read_json('data.json')

# Initial Exploration
df.head()  # First 5 rows
df.tail()  # Last 5 rows
df.info()  # Summary of data types and null values
df.describe()  # Statistical summary

# Column Names
columns = df.columns.tolist()

# Data Types
dtypes = df.dtypes

# Unique Values in a Column
unique_values = df['column'].unique()

# Count of Unique Values
unique_count = df['column'].nunique()

# Null Values
null_values = df.isnull().sum()

# Subsetting Data
df_subset = df[['col1', 'col2']]

# Filtering Data
filtered_df = df[df['col1'] > 10]
```

1.2. Data Cleaning

```
# Handle Null Values
df.dropna(inplace=True)  # Drop rows with NaN values
df.fillna(0, inplace=True)  # Replace NaN values with 0
df['col'].fillna(df['col'].mean(), inplace=True)

# Remove Duplicates
df.drop_duplicates(inplace=True)

# Change Data Types
df['col1'] = df['col1'].astype('int')  # Convert to integer
df['col2'] = df['col2'].astype('float')  # Convert to float
df['col3'] = df['col3'].astype('str')  # Convert to string

# Standardize Text
df['text_col'] = df['text_col'].str.lower()  # to lowercase

# String Manipulations (Split and take first element)
df['new_col'] = df['text_col'].str.split('␣').str[0]

# Replace Values
df['col'].replace({'old_value': 'new_value'}, inplace=True)

# Outlier Handling
Q1 = df['col'].quantile(0.25)
Q3 = df['col'].quantile(0.75)
IQR = Q3 - Q1
df_filtered = df[(df['col'] >= Q1 - 1.5 * IQR) & (df['col'] <= Q3 + 1.5

# One-Hot Encoding for Categoricals
df = pd.get_dummies(df, columns=['cat_col'], drop_first=True)
```

1.3. Feature Engineering

```
# Create New Columns
df['sum_col'] = df['col1'] + df['col2']
df['ratio_col'] = df['col1'] / df['col2']

# Apply Functions to Columns
df['log_col'] = df['col'].apply(np.log)
df['squared_col'] = df['col'].apply(lambda x: x**2)

# Categorical to Numerical Encoding
df['encoded_col'] = df['categorical_col'].map({'class1': 1, 'class2': 2})

# One-Hot Encoding
df_one_hot = pd.get_dummies(df['categorical_col'])
df = pd.concat([df, df_one_hot], axis=1)

# Binning
bins = [0, 10, 20, 30]
labels = ['low', 'medium', 'high']
df['binned_col'] = pd.cut(df['numerical_col'], bins=bins, labels=labels)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['scaled_col'] = scaler.fit_transform(df[['numerical_col']])

# Date and Time Features
df['year'] = df['date_col'].dt.year
df['month'] = df['date_col'].dt.month
df['day'] = df['date_col'].dt.day

# Lag Features for Time-Series
df['lag1'] = df['time_series_col'].shift(1)
df['lag2'] = df['time_series_col'].shift(2)

# Rolling Statistics for Time-Series
df['rolling_mean'] = df['time_series_col'].rolling(window=3).mean()
```

1.4. Modeling

```
# Import Models and Metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Initialize Model
lin_reg = LinearRegression()
rf_clf = RandomForestClassifier()
svc_clf = SVC()

# Train Model
lin_reg.fit(X_train, y_train)
rf_clf.fit(X_train, y_train)
svc_clf.fit(X_train, y_train)

# Predict
lin_pred = lin_reg.predict(X_test)
rf_pred = rf_clf.predict(X_test)
svc_pred = svc_clf.predict(X_test)

# Cross-Validation
from sklearn.model_selection import cross_val_score
cv_score = cross_val_score(rf_clf, X, y, cv=5)

# Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV
param_grid = {'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20
grid_search = GridSearchCV(rf_clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get Best Params
best_params = grid_search.best_params_

# Use Best Model
best_rf_clf = grid_search.best_estimator_
```

1.5. Model Evaluation

```
# Import Evaluation Metrics
from sklearn.metrics import mean_squared_error, accuracy_score, f1_scor
roc_auc_score, confusion_matrix

# Evaluate Regression Model
mse = mean_squared_error(y_test, lin_pred)
rmse = np.sqrt(mse)

# Evaluate Classification Model
accuracy = accuracy_score(y_test, rf_pred)
f1 = f1_score(y_test, rf_pred)
roc_auc = roc_auc_score(y_test, rf_pred)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, rf_pred)

# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, rf_pred))

# AUC-ROC Curve for Classification
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
fpr, tpr, thresholds = roc_curve(y_test, rf_pred)
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

# Feature Importance
feature_importance = rf_clf.feature_importances_
```

2. Data Visualization

2.1. Plotting with pandas

```
# Bar Plot with Pandas
df['col1'].value_counts().plot(kind='bar')

# Histogram with Pandas
df['col1'].plot(kind='hist')

# Line Plot with Pandas
df['col1'].plot(kind='line')

# Scatter Plot with Pandas
df.plot(x='col1', y='col2', kind='scatter')

# Box Plot with Pandas
df.boxplot(column='col1', by='col2')
```

2.2. Plotting with matplotlib

```
# Import Matplotlib
import matplotlib.pyplot as plt

# Import Seaborn for styling
import seaborn as sns

# Set Style
sns.set(style="whitegrid")

# Matplotlib Bar Plot
plt.bar(df['col1'], df['col2'])
plt.xlabel('col1')
plt.ylabel('col2')

# Matplotlib Histogram
plt.hist(df['col1'], bins=20)
plt.xlabel('col1')

# Matplotlib Line Plot
plt.plot(df['col1'], df['col2'])
plt.xlabel('col1')
plt.ylabel('col2')

# Matplotlib Scatter Plot
plt.scatter(df['col1'], df['col2'])
plt.xlabel('col1')
plt.ylabel('col2')

# Matplotlib Box Plot
plt.boxplot(df['col1'])
plt.ylabel('col1')

# Seaborn Pairplot
sns.pairplot(df)

# Seaborn Heatmap for Correlation
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

3. Other commands of important packages

3.1. NumPy Commands

```
# Import NumPy
import numpy as np

# Create Array
arr = np.array([1, 2, 3])

# Create Zeros, Ones, and Identity Matrix
zeros = np.zeros((3, 3))
ones = np.ones((3, 3))
identity = np.eye(3)

# Create Range and Space
range_arr = np.arange(0, 10, 2)
space_arr = np.linspace(0, 10, 5)

# Random Numbers
random_arr = np.random.rand(3, 3)
random_int = np.random.randint(0, 10, (3, 3))

# Array Shape and Reshape
shape = arr.shape
reshaped = arr.reshape((3, 1))

# Indexing and Slicing
first_element = arr[0]
slice_elements = arr[0:2]

# Basic Operations (+, -, *, /)
sum_arr = arr + arr
diff_arr = arr - arr
prod_arr = arr * arr
div_arr = arr / arr

# Aggregation Functions
sum_all = np.sum(arr)
mean_all = np.mean(arr)
max_all = np.max(arr)

# Boolean Masking
mask = (arr > 1)
filtered = arr[mask]

# Matrix Operations
dot_product = np.dot(arr, arr)
transpose = np.transpose(arr)
```

3.2. Re: Text Processing

```
# Import Libraries
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

# Basic String Operations
text = "This␣is␣a␣sample␣text."
lower_text = text.lower()
upper_text = text.upper()
split_text = text.split()

# Regular Expressions
email_extract = re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text)
words_extract = re.findall(r'\w+', text)

# Tokenization
tokens = word_tokenize(text)

# Remove Stopwords
filtered_words = [word for word in tokens if word.lower() not in stopwords.words('english')]

# Stemming
stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in filtered_words]

# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform([text])

# N-grams
bigrams = [(tokens[i], tokens[i + 1]) for i in range(len(tokens) - 1)]
```